Enabling Secure High-Performance Wireless Ad Hoc Networking

Yih-Chun Hu

CMU-CS-03-144

May 29, 2003

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee

David B. Johnson, Chair
Edward W. Knightly
Srinivasan Seshan
Peter Steenkiste

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

This work was supported in part by the NSF under grant CCR-0209204 at Rice University, by NASA under grant NAG3-2534 at Rice University, by a gift from Schlumberger to Rice University, by the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061, and by an NSF Graduate Fellowship. The views and conclusions contained here are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, NASA, Schlumberger, AFMC, DARPA, Rice University, Carnegie Mellon University, or the U.S. Government or any of its agencies.

Keywords: ad hoc networks, mobility, performance, security, wireless

Contents

Illustrations

List of Tables

Acknowledgements

First, I'd like to thank my parents for all their support over the years; it goes without saying that without them, I wouldn't be where I am today (or even extant, for those interested in technicalities). I'm also grateful for my sister, Yih-Mei, who has in many things preceded me and passed her experiences back to me. Cathy Lu's support and encouragement throughout much of my time in graduate school is also much appreciated. Thanks to the Father for the incomparable riches of His grace, in which I have life to the fullest.

Academically, I'm deeply indebted to the mentorship of my advisor, Dave Johnson. The long hours he dedicated to working on papers and talking through ideas helped seed and solidify many of the new concepts in this thesis. His effort in shielding me from undesirable red tape redeemed many hours for research (and recreation). I'm also thankful for the opportunity to work with Adrian Perrig, who provided much motivation, ideas, and insight into our work on security.

I'd also like to thank my other committee members: Ed Knightly, Srini Seshan, and Peter Steenkiste. Their feedback on my proposal and my thesis, their time in reading this monstrosity, and their understanding throughout the various changes of direction, are greatly appreciated.

The members of my research group, the Monarch Project, have also been a great help in this work: Dave Maltz and Josh Broch laid the framework for the simulations and implementations described in this thesis, and Dave also initially designed two of the mechanisms which I later finalized and reduced to practice for this thesis. Jorjeta Jetcheva and Qifa Ke provided indispensable help with the preparations for the Quality-of-Service demonstration. Amit Saha helped validate the implicit source routing simulations and provided some source code for it as well. Santashil PalChaduri has always been a fun target to bounce ideas off of, and his work on the Ad Hoc City simulations is greatly appreciated and highly motivating.

I'd also like to thank all the referees that reviewed this work as pieces of it were submitted to various conferences. Their suggestions have improved both the presentation and content. Dawn Song also provided valuable feedback to many of the security chapters in their paper forms. Additionally, I'd like to thank Victor Bahl and Dan Wallach for their unwavering insistence that I graduate as soon as possible.

I'd like to thank my friends, who have provided me much support, both in working on this thesis and otherwise. Dave Matsumoto and Esther Chen helped directly in helping with the QoS demo, Edwin Chan pushed me to work on my thesis proposal, and friends too numerous to mention supported me through good times and bad.

On the administrative side, Sharon Burkes and Catherine Copetas helped coordinate many events leading up to my graduation, from speaking requirement talks to thesis proposals to my defense. Debbie Cavlovich put up with my travel reimbursements while I was at CMU, and the support staff at the Information Networking Institute was instrumental in aquiring the needed hardware for the QoS demo. I finished this thesis as a visiting student at Rice University, where the support staff there helped me navigate the maze of regulations; in particular, Iva Jean Jorgensen, Rhonda Guajardo, Tammy Luc, Darnell Price, and Lena Sifuentes were of great help during my time at Rice.

Financially, I'd like to thank the NSF for supporting me through the NSF Graduate Fellowship program, as well as under grant CCR-0209204. I'd also like to thank NASA for their support under grant NAG3-2534 at Rice University, Schlumberger for their support in the form of a gift to Rice University, and the Air Force Materiel Command (AFMC) for their support under DARPA contract number F19628-96-C-0061.

Chapter 1

Introduction

1.1. Why is Service Important in Ad Hoc Networks?

A revolution has occurred in wireless communications over the last decade. Advances in processing power have enabled widespread deployment of radio networks as well as portable devices that can make use of such networks. Wireless networks have become more convenient and affordable and have spread to even fairly stationary applications, such as home networking and community networking. At the same time, the importance of networking as a field of computer science has greatly increased, as the Internet's exponential growth demonstrated Metcalfe's law: the usefulness of a network is proportional to the square of the number of users. Network connectivity is more frequently than ever the most important use of a computer, due to the development of many useful networked applications, such as email, the World Wide Web (WWW), peer-to-peer file sharing, and presense and instant messaging. Even mobile devices are increasingly expected to support some level of data network connectivity; Palm and Pocket PC devices, RIM Blackberries, and many cell phones are capable of connecting to the Internet.

Wired networks use routing because it is impractical to have all hosts connected to the same physical wire; similarly, in wireless networks, it is often undesirable to increase propagation range, due to increased propagation delay, reduced utilization, higher power requirements, and increased interference that results from such an approach. *Routing protocols* are used to automatically find routes between nodes that wish to communicate. A number of routing protocols have been proposed as the Internet transitioned from research network into the commercial network that it is today. Even today, many different routing protocols are used in the Internet. Unfortunately, previously proposed wired network routing protocols are generally unsuitable for *ad hoc networks*. An ad hoc network is a network in which nodes cooperate to dynamically establish routing among themselves; a packet sent by one node may be forwarded in turn by a sequence of other nodes, allowing the packet to reach a destination beyond the sender's wireless transmission range. An *ad hoc network routing protocol* is protocol designed to operate in this environment. Such protocols must continue to function when node mobility, node failure, and changing wireless propagation conditions cause rapid topology change. It is also desirable for an ad hoc network routing protocol to continue to function when malicious nodes are present.

This thesis discusses improvements to *service* in ad hoc network routing. Since different networks require different types of service, I propose several mechanisms to improve service in various network conditions. One example of service is Quality-of-Service (QoS), since certain networks may desire some flows to have priority over other flows. Many of the mechanisms presented here make no distinction between lower and higher priority traffic, and in my evaluation, I only examine performance metrics aggregated over all flows, rather than the performance of a few select flows.

Another area encompassed by service is security. In particular, when an attacker is present in the network, a protocol that provides security against such an attacker should provide better service than one that does not. For example, a secure protocol should deliver more packets, incur less overhead, or conserve overall network power usage better than in insecure protocol when the network is under attack. As a network experiences attack, a secure network routing protocol may continue to provide some level of service, whereas a traditional network routing protocol may fail completely.

Since service can be judged by many metrics, an increase in service measured by some metrics may result in a decrease in service as measured by other metrics. For example, our secure routing protocols have significantly higher overhead due to the authentication information that they carry. In this thesis, analyze the performance of each mechanism I present to clarify the circumstances under which the use of each mechanism may be desirable.

1.2. Description of Several Ad Hoc Network Routing Protocols

1.2.1. Distance-Vector Routing Protocols

The idea of treating each wireless node as a router was first developed by DARPA PRNET [91]. PRNET introduced many ideas for optimizing *distance-vector* routing protocols for use in an ad hoc network. In a distance-vector protocol, each node keeps a *routing table*. Each entry in this table contains all the information necessary to route packets to a single destination; in a distance vector protocol, a table entry typically contains the destination address, the number of hops to the destination (called the *metric*), and the next hop along the route to the destination. Each node periodically sends its routing table to its neighbors in a packet called an *advertisement*. When a node receives a routing table from its neighbor, it checks to see if the table contains any destinations that this node does not yet have a route to, or if the table provides a better route. In particular, if a node receives an advertisement from node **A** indicating some metric x to a destination **D**, it has learned a route through **A** with metric $x + 1$. If this route of metric $x + 1$ is better than this node's current route, it updates its routing table to use this newly found route, by setting its metric to $x+1$ and its next-hop destination to **A**. This algorithm is called the *distributed Bellman-Ford algorithm* [12, 53].

In 1994, two new ad hoc network routing protocols were introduced: the Destination Sequence Distance Vector (DSDV) protocol [140], and the Dynamic Source Routing (DSR) protocol [87–90]. DSDV is based on distance-vector routing, but introduces a sequence number in addition to the metric. This destination sequence number prevents loops, but requires *hard state*: that is, the protocol requires that nodes retain state for an unbounded period of time. In particular, DSDV cannot recover from a reboot without knowing its previous sequence number. A node increases its sequence number each time it sends an update. A routing update with a higher sequence number takes priority, and updates with equal sequence numbers are chosen based on metric. DSDV also imposes a delay between the receipt of an advertisement and the readvertisement of a new route based on that advertisement; however, this delay does not necessarily improve performance (Chapter 6).

1.2.2. Dynamic Source Routing (DSR)

DSR is a completely on-demand protocol; that is, control packets are sent in DSR only when needed. DSR divides the routing problem into two parts: *Route Discovery* and *Route Maintenance*.

In DSR, a node sending a packet includes in that packet a source route selected from the source's Route Cache. This Route Cache is populated using Route Discovery. In Route Discovery, a node that wishes to send a packet to a destination, but does not have a route to that destination in its Route Cache, initiates Route Discovery by broadcasting a ROUTE REQUEST packet to its neighbors. The ROUTE REQUEST packet includes the addresses of the initiator (this node) and target (the destination this node wishes to reach) of the Discovery, a unique identifier from the initiator, and a route record listing the nodes traversed by this packet. A node hearing a REQUEST checks if it is the target of the REQUEST. If so, it returns a ROUTE REPLY to the initiator, containing the route from the route record in the received REQUEST. This ROUTE REPLY can be routed by searching the target's Route Cache, by reversing the route recorded in the REQUEST, or by piggybacking the ROUTE REPLY on another Route Discovery with a target of the original initiator. Otherwise, the node checks the unique identifier to determine if it has previously rebroadcasted a REQUEST from the same Route Discovery, and if so, it silently drops the REQUEST. The node also checks if its address already appears in the route record carried in the REQUEST, and if so, it discards the REQUEST. Finally, since the node is not the target, and has not previously rebroadcasted a REQUEST from this Route Discovery, then it appends its address to the route record in the REQUEST and rebroadcasts it.

Route Maintenance is the mechanism by which DSR detects that a route has stopped working. Each node forwarding a packet along a source route performs Route Maintenance by attempting to confirm that the forwarded packet successfully reaches the next-hop destination. This confirmation can be received through a link-layer acknowledgement such as in IEEE 802.11 [84], through a passive acknowledgement [91], or through an explicit network-layer acknowledgement. If a node is unable to confirm receipt at the next-hop destination, the node returns a ROUTE ERROR to the source of the packet, indicating that its link to the next-hop destination is broken. Any node hearing this ERROR removes that link from its Route Cache.

A number of optimizations have been proposed to improve the performance of DSR; this thesis discusses interactions between the work described here and two optimizations: *salvaging* and *automatic route shortening*. In salvaging, a node that detects a broken link and returns a ROUTE ERROR then the node may attempt to *salvage* the packet if it has in its own Route Cache a different route to the packet's destination; to do so, the node replaces the original route with the route from its cache and transmits the packet to the new next-hop node. As another optimization, DSR supports *automatic route shortening* to allow source routes in use to be shortened when two nodes in the source route move close enough together so that one or more intermediate hops are no longer necessary. With this optimization, each node operates its network interface in *promiscuous mode*, such that each packet wirelessly received by the network interface is passed to the network protocol stack, regardless of the packet MAC-layer destination address. If a node is able to promiscuously receive a packet not intended for it as the next hop, but for which this node is listed in the unused portion of the packet's source route, then this node returns a "gratuitous" ROUTE REPLY to the original sender of the packet; this REPLY gives the shorter route that does not include the intermediate nodes between the node that transmitted the packet and this node. For example, if a packet was sent along a source route (**A**,**B**,**C**,**D**), and node **C** overhears node **A**'s transmission, then **C** returns a REPLY to **A** giving the route (**A**,**C**,**D**).

1.2.3. Ad-hoc On-Demand Distance Vector (AODV)

In 1999, the Ad-hoc On-Demand Distance Vector (AODV) protocol [141] was introduced. AODV turns DSDV into an on-demand protocol, borrowing many features from DSR. AODV performs on-demand Route Discovery, using ROUTE REQUEST and ROUTE REPLY packets (called RREQ and RREP packets in AODV). Rather than accumulating in the ROUTE REQUEST a route record listing the nodes traversed by this request and then returning this route record in a ROUTE REPLY, AODV uses these packets to propagate distance vector updates, providing routing information back to the node initiating the Discovery. For example, if node **A** sends an RREQ for target node **D**, the RREQs establish routes to **A**, and the RREP sent by **D** establish routes to **D**. These packets establish routes on a hop-by-hop basis so no source routes are needed. Because AODV does not send RREQs and RREPs periodically, it also needs a mechanism for link breakage notification; AODV borrows DSR's ROUTE ERROR (called RERR in AODV). A node detecting link breakage transmits an RERR to each upstream node that is using that link. Because of its similarity to DSR, many optimizations I present for DSR can also be used in AODV.

1.2.4. Other On-Demand and Periodic Protocols

A number of other ad hoc network routing protocols have been proposed [13, 19, 36, 47, 55, 62, 99, 129, 137, 154, 181]. Some of these protocols exchange messages periodically, and some only exchange messages only on-demand. On-demand routing protocols are often more suitable than periodic protocols for ad hoc network routing for two reasons. First, a periodic protocol that does not adapt the rate at which it transmits routing information is unable to successfully route packets to their destinations when the rate of network topology change between the source and the destination is too high, whereas on-demand protocols automatically scale their use of routing packets to adapt to the rate of topology change. Second, in scenarios with a lower rate of topology change, on-demand protocols transmit routing packets only when a topology change affects a route actually in use or when a packet is sent to a new destination, whereas periodic protocols require the transmission of routing packets even when no topology change occurs and when there are no packets to be delivered. These additional routing packets reduce the bandwidth available to applications and require battery power to transmit and receive. Numerous simulation results have confirmed that on-demand protocols provide higher performance than periodic protocols in most situations. As a result, in this thesis, I focus on service in on-demand protocols. I focus my protocol-specific discussions on DSR and AODV, since they are the leading proposals for on-demand ad hoc network routing protocols.

1.3. Thesis Contributions

This thesis presents a number of contributions to the area of improving service in on-demand protocols for wireless ad hoc network routing. For scenarios without attackers and without real-time requirements, I present the mechanism of *link-state caching*, which improves the ability of source-routed on-demand routing protocols to retain much of the information useful for routing while discarding potentially stale information. I show that this can be done in an adaptive manner, and that an adaptive cache can often outperform a statically parameterized cache. I also contribute *implicit source routes*, which improve performance by removing a major disadvantage of source routing (in particular, the per-hop overhead of carrying source routes) while retaining the advantages of source routing.

This thesis also presents contributions to the area of providing real-time services in networks that are not under attack. I contribute the use of implicit source routing for packet classification, allowing the selection of per-hop behavior based on which flow to which a packet belongs. I also contribute a technique that uses physical layer information (specifically, the Signal-to-Noise Ratio) to choose routes, allowing the protocol to choose longer-lived routes, and also enabling the routing protocol to find new routes before old ones break. Finally, I show how MAC layer utilization information can be used to choose less congested paths, thus enabling higher network-wide throughput and providing significantly improved TCP fairness.

Finally, I provide contributions to secure ad hoc network routing which allow improved service in hostile environments. These contributions include SEAD and Ariadne, two routing protocols resiliant to a wide variety of attacks. I also contribute general mechanisms for efficiently securing distance-vector routing and Quality-of-Service routing, and mechansisms for resisting the tunneling and rushing attacks. In addition, I contribute *skiplists*, an alternative to very long hash chains, which significantly reduces the computational cost of following a long hash chain at the cost of increased storage overhead.

1.4. Thesis Overview

In this thesis, I present two types of service improvements. Part I covers improvements originally designed for a trusted environment. I present *link-state caching* (Chapter 2), which improves packet delivery and overhead in source-routed protocols. This technique allows higher levels of service for applications less tolerant of packet loss (such as TCP), and also provides an approach to improving other types of on-demand protocols. *Implicit source routing* (Chapter 3) substantially reduces the overhead of source routing and marginally improves packet loss and latency. More importantly, the use of implicit source routes allows trivial packet classification, so forwarding nodes can identify packets belonging to high-priority flows and can forward such packets with higher priority. Finally, implicit source routing supports multipath routing without additional support from intermediate nodes, allowing a node to spread its traffic across multiple disjoint routes for improved throughput. In *preemptive Route Maintenance* (Chapter 4.5), forwarding nodes can preemptively warn a source of impending link breakage, which can reduce the latency caused by route breakage. This technique can improve service to real-time applications such as videoconferencing and Voice-over-IP. I present a mechanism for *limiting Route Discovery based on Signal-to-Noise Ratio (SNR)* (Chapter 4.5), which prevents the discovery of routes that are likely to break quickly or that may not work at all. Such routes might otherwise be discovered due rapid channel fluctuation often experienced in real-world radio propagation. Finally, I present *cross-layer interactions* (Chapter 4), which use measurements from lower layers to improve service to higher-layer protocols.

In Part II, I present mechanisms for securing ad hoc network routing protocols. First, I introduce some security primitives and define an attacker model in Chapter 5. I then present two secure ad hoc network routing protocols. In Chapter 6, I present *SEAD*, which is based on DSDV but can be extended to support on-demand distance vector protocols such as AODV. Chapter 7 presents *Ariadne*, a secure on-demand routing protocol based on DSR that resists many kinds of attacks. In Chapter 8, I present several efficient security mechanisms: hash chains for metric and sequence number authentication, hash tree chains to force a router to increase the distance when forwarding a route update, tree-authenticated one-way chains for more efficient routing update authentication, skiplists for cases in which the maximum metric is large, and cumulative authentication, which substantially reduces Ariadne's network overhead. I then present *packet leashes* (Chapter 9), which protect against a powerful attack against ad hoc network routing protocols, called the "wormhole attack." Finally, I present mechanisms for securing *QoS-guided Route Discovery* (Chapter 11), which allows a node to find a route meeting its QoS requirements, limiting an attacker's ability to inflating the service level provided by the path that it is on.

Part I

Improving Service in Trusted Environments

Chapter 2

Using Link-State Caching in Ad Hoc Networks

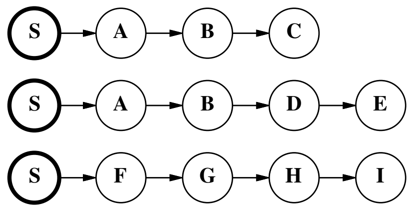
Section 1.2.2 gives a brief overview of Dynamic Source Routing (DSR). As originally evaluated [26, 88], DSR worked essentially as a path-vector routing protocol. A path vector routing protocol differs from a distance vector routing protocol in that instead of only retaining the metric, or distance, to each destination, each node maintains the entire path to the destination. In DSR, as it had been evaluated prior to this work, each Route Discovery provided several paths for the Route Cache. Each path could be used in whole or in part, but information from different paths was never joined together. This Route cache had a limited capacity, statically chosen to work well for the scenarios tested. Intuitively, this cache of paths (called a *path cache*) is less powerful than a *link cache* which contains links and uses a shortest path algorithm such as Dijkstra's algorithm to find routes as necessary. Furthermore, the use of a static capacity required tuning depending on how many destinations each source sent to. In this chapter, we explore several options for cache design, specifically in the areas of cache structure, cache capacity, and cache timeout. These choices substantially affect the service provided by a network, in the areas of packet loss and routing overhead. We explore these differences in service level through detailed simulation of the interaction between DSR and various cache designs.

2.1. Caching Strategy Design Choices

2.1.1. Cache Structure

In developing a caching strategy for an on-demand routing protocol for wireless ad hoc networks, one of the most fundamental design choices that must be made is the type of data structure used to represent the cache. In DSR, the route returned in each ROUTE REPLY that is received by the initiator of a Route Discovery represents a complete path (a sequence of links) leading to the destination node. By caching each of these paths separately, a *path cache* can be formed; Figure 2.1(a) illustrates an example path cache for some node **S** in the ad hoc network. Alternatively, a *link cache* could be created, in which each individual link in the routes returned in ROUTE REPLY packets is added to a unified graph data structure of this node's current view of the network topology; Figure 2.1(b) illustrates an example link cache for node **S**.

A path cache is very simple to implement and easily guarantees that all routes are loop-free, since each individual route from a ROUTE REPLY is loop-free. To find a route in a path cache, the sending node can simply search its cache for any path (or prefix of a path) that leads to the intended



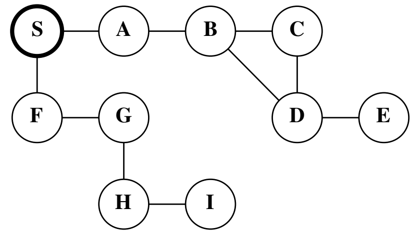


Figure 2.1: Alternative Cache Data Structures for a Node **S**

destination node. On the other hand, to find a route in link cache, a node must use a much more complex graph search algorithm, such as the well-known Dijkstra's shortest-path algorithm, to find the current best path through the graph to the destination node. Such an algorithm is more difficult to implement and may require significantly more CPU time to execute.

However, a path cache data structure cannot effectively utilize all of the potential information that a node might learn about the state of the network. In a link cache, links learned from different Route Discoveries or from the header of any overheard packets can be merged together to form new routes in the network, but this is not possible in a path cache due to the separation of each individual path in the cache. For example, if node **S** with the cache as shown in Figure 2.1(b) learns of a new link from node **A** to node **G**, it can use this link to also form new routes to nodes **H** and **I** (through **A** and **G**) that it could use if the link from **F** to **G** later breaks, but a node using a path cache would be unable to take advantage of these additional routes.

2.1.2. Cache Capacity

The capacity of a route cache is another important area of choice in designing a caching strategy for on-demand routing protocols. For a link cache, the logical choice is to allow the cache to store any links that are discovered, since there is a fixed maximum of N^2 links that may exist in an ad hoc network of N nodes. However, for a path cache, the maximum storage space that could be required is much larger, since each path is stored separately and there is no sharing in the data structure even when two paths share a number of common links. We thus consider the effects of different limits on the capacity of path caches in terms of the number of individual paths it can store. In general, our intuition was that the larger the capacity of a path cache, the better the routing protocol should perform, since it is able to keep a more complete set of routes. However, as we show in Section 3.3, a *smaller* cache size actually can have an indirect effect in improving performance.

An additional design choice with respect to cache capacity that we consider is the division of the cache into two halves: one half for paths that have been used by this node (the primary cache) and a second half for paths that have not yet been used since being learned (the secondary cache); when a path (or a prefix of a path) in the secondary cache is first used, that path (or prefix) is promoted to the primary cache. This division of the cache avoids forcing out of the cache paths that this node has found useful, when attempting to insert some new path into the cache that has just been learned and has not yet been used (and may never be used). Old paths in the secondary cache are removed due to the natural operation of the cache when adding new paths as they are learned, whereas old paths in the primary cache are more actively removed due to the operation of Route Maintenance as they are used. We refer to such a divided cache as a *generational* cache, in a manner similar to the way a generational garbage collector works in a language runtime system with dynamic storage allocation.

2.1.3. Cache Timeout

As with cache capacity, cache timeout policy introduces a number of design choices to consider in a caching strategy. Because a path cache generally has a mechanism for removing entries through a capacity limit, we did not implement a timeout for path caches. For link caches, the timeout on each link in the cache may be either fixed or adaptive.

For a fixed timeout, each link is removed from the cache after a specified amount of time has elapsed since the link was added to the cache. For an adaptive timeout, a node adding a link to its cache attempts to determine a suitable timeout after which the link will be deleted from the cache, and this timeout value should be based on properties of the link or the nodes that are the endpoints of the link. Finally, similar to the generational path caching alternative, it is possible to allow a link that is being used to not expire by increasing its timeout when it is used.

2.2. Caching Algorithms Studied

From the caching strategy design choices given in Section 2.1, we chose a collection of path caches and link caches to simulate and evaluate. We also simulated an “omniscient expiration” cache, which although unimplementable in a real system, gives us a benchmark against which our other cache algorithms can be compared.

2.2.1. Path Caches

Path caches store a set of complete paths (sequences of links), each starting at the caching node. We analyzed the following algorithms that use path caches:

- *Path-Inf* is a path cache with no capacity limit.
- *Path-FIFO-64* is a path cache with a 64-path capacity limit. The cache replacement policy used on paths in the cache is FIFO.
- *Path-FIFO-32* is the same as *Path-FIFO-64*, except that it uses a 32-path capacity limit.
- *Path-Gen-64* is a generational path cache that employs a 30-element FIFO primary cache to store paths that have been used or were returned directly to this node in a ROUTE REPLY, and a separate 64-element FIFO secondary cache to store other paths; the total capacity of this cache is 94 elements.
- *Path-Gen-34* is the same as *Path-Gen-64*, except that the size of the secondary cache is 34-elements; the total capacity of this cache is 64 elements, the same as *Path-FIFO-64*. This specific caching algorithm, of this size, is the same as that used in our original *ns-2* simulation of DSR [26].

2.2.2. Link Caches

Link caches store a set of individual links, organized as a graph data structure. We analyzed the following algorithms that use link caches:

- *Link-NoExp* is a link cache with no timeout.
- *Link-Static-5* is a link cache in which links normally are expired 5 seconds after they are put into the cache. This is a generational cache, with links that are used to source packets sent by this node marked to not timeout.
- *Link-Adapt-1.25* is a link cache in which a link's timeout is chosen according to a stability table. Each node keeps a table recording the stability of each other node. When a link is used, the stability metric for both endpoints is incremented by the amount of time since that link was last used, multiplied by some factor; when a link is observed to break, the stability metric for both endpoints is multiplicatively decreased by a different factor. A link entering the cache is given a lifetime equal to the stability of the less-"stable" endpoint of the link, except that a link is not allowed to be given a lifetime under 1 second. As with *Link-Static-5*, this is a generational cache, with links that are used to source packets sent by this node marked to not timeout. For this cache, the additive increase factor is 4, and the multiplicative decrease factor is 1.25. The stability table for each node is initialized to 25 seconds.
- *Link-Adapt-2* is the same as *Link-Adapt-1.25*, except that the multiplicative decrease factor is 2.
- *Link-MaxLife* is the same as *Link-Adapt-2*, except that when a node chooses a route, it chooses the shortest-length route that has the longest expected lifetime (highest minimum timeout of any link in the path), as opposed to an arbitrary route of shortest length.

2.2.3. Omniscient Expiration Cache

For comparison against the other caching algorithms that we studied, we also analyzed the following "omniscient expiration" caching algorithm:

- *OmniExp* is a link cache that performs omniscient expiration of cached links, such that a link is removed from the cache exactly when it ceases to physically exist. The simulator has omniscient knowledge of the location of all nodes, and *OmniExp* bases cache expiration on a nominal wireless transmission range for each link of 250 m.

2.3. Methodology

2.3.1. Simulator

We analyzed the effects the different caching strategy design choices through detailed simulation of the different caching algorithms described in Section 2.2. The experiments were conducting using the *ns-2* network simulator [50], extended to support the simulation of wireless and mobile networks [26]. The simulator properly models signal strength, RF propagation, propagation delay, wireless medium contention, capture effect, interference, and arbitrary continuous mobility. The radio model is based on the Lucent Technologies WaveLAN 802.11 product, providing a 2 Mbps transmission rate and a nominal transmission range of 250 m. The link layer modeled is the Distributed Coordination Function (DCF) of the IEEE 802.11 wireless LAN standard [84]. Each of our simulations were run using 50 nodes moving over a simulated time of 900 seconds, and all nodes were confined to a 1500 m $\times$ 300 m space.

2.3.2. Communication Model Used

The communication model simulated in all scenarios was a script consisting of 20 Constant Bit Rate (CBR) data connections, each transmitting 4 packets per second; the size of each packet is 64 bytes. Each node was the source of at most 2 CBR connections.

2.3.3. DSR Performance Metrics

We evaluated the performance of DSR on each of the caching algorithms according to four metrics:

- *Packet Delivery Ratio*: The fraction of packets sent by the "application layer" on a source node that are received by the "application layer" on the corresponding destination node.
- *Packet Overhead*: The total number of packets transmitted by the routing protocol. This includes routing packets forwarded, but not data packets forwarded.
- *Average Latency*: The average delay from when a packet is sent by the "application layer" on a source node until it is received by the "application layer" on the corresponding destination node. This can only computed for packets that are successfully delivered.
- *Path Optimality*: The difference between the number of hops over which a packet was routed and the number of hops in the shortest route that physically existed when the packet was sent. The simulator is able to determine this theoretical shortest route at all times, based on the nominal wireless transmission range for each link of 250 m.

2.4. Mobility Metrics

When we performed these evaluation studies, the random waypoint mobility model modulated by the pause time was criticised as not being significantly different than Brownian motion. To verify that out mobility models were reasonable, we examined our movement patterns using mobility metrics. Previously, Johansson et al [86] had proposed one mobility metric, which we found did not correlate well with the difficulty of routing in any given movement pattern. As a result, we developed some mobility metrics to more accurately assess the relative difficulty of routing given multiple movement scenarios.

Johansson et al [86] describe a *geometric mobility metric* that is computed for a given scenario by

$$\frac{2}{n(n-1)T}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\int_{t=0}^{T}\left|\frac{d\|P_j(t)-P_i(t)\|_2}{dt}\right| dt$$

where each $P_k(t)$ is the position of a node k at time t, n is the number of nodes, T is the length of the simulation, and the sum is calculated over all pairs of nodes over all time. For the results in their paper, they approximated this metric by computing it with a 0.1-second time granularity and rearranged [104] the equation to compute

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\int_{t=0}^{T}\left|\frac{d\|P_j(t)-P_i(t)\|_2}{dt}\right| dt \;\approx\; \frac{1}{2}\sum_{i=1}^{n}\sum_{t=0}^{10T}\left|\sum_{j=1}^{n}\left\|P_j\left(\frac{t}{10}\right)-P_i\left(\frac{t}{10}\right)\right\|_2-\sum_{j=1}^{n}\left\|P_j\left(\frac{t-1}{10}\right)-P_i\left(\frac{t-1}{10}\right)\right\|_2\right|$$

This approximation, however, can lead to a very inaccurate calculation in some cases. For example, on scenarios generated using Brownian motion, as described in Section 2.5.1, the approximate mobility metric (with 0.1-second granularity) was too small by more than a factor of 2.2.

Table 2.1: Parameters for Brownian Motion

Movement interval duration	0.1 s
v_{max}	20 m/s

Instead, we used to following technique to calculate their geometric mobility metric precisely: split the integral so that each integral is along an interval in which there is no change in the velocity of either i or j. Define $f(t) = \|P_j(t) - P_i(t)\|_2$. We want $\int_{t_1}^{t_2} \left|\frac{df(t)}{dt}\right| dt$. If there is no relative velocity, then the integral is 0. If f has no local minima on $[t_1, t_2]$, then the integral evaluates to $\left|f(t)|_{t=t_1}^{t_2}\right|$. Otherwise, if $t' \in [t_1, t_2]$ is a local minima of f, then the integral is $f(t_1) + f(t_2) - 2f(t')$.

A difficulty with the geometric mobility metric [86] is that it cannot distinguish between mobility that changes the network topology and mobility that instead has no effect on any links in the network. If we have information about the nominal wireless transmission range of the radios used in the network, we can more accurately determine how mobility affects the difficulty of routing. The *minimal shortest route-change metric* for a pair of nodes i and j is the minimum number of times that i and j would need to change routes in order to always have a shortest (least hops) path to each other, assuming all links are bi-directional. An alternate metric, that we call the *minimal route-change metric*, is the same as the minimal shortest route-change metric, except that a route counted by the metric only changes when it breaks, not when a shorter route begins to exist. The minimal shortest route-change metric and the minimal route-change metric both provide a number per pair of nodes; to arrive at a metric for a scenario, we can either sum over only those node pairs that communicate at least once during the scenario, or simply over all pairs of nodes regardless of communication behavior.

2.5. Mobility Models Studied

2.5.1. Mobility Model Specifications

We chose the parameters for our different mobility models to make the average speed of a node 10 m/s, and to keep the nodes as randomly distributed as the scenario would allow. Unless otherwise noted, the initial position of each node is chosen as (x_0, y_0), with x_0 uniformly distributed over $[0, 1500 \text{ m}]$ and y_0 uniformly distributed over $[0, 300 \text{ m}]$. Ten different scenarios were generated for each of the five models we studied: Brownian motion, Column motion, Random Gauss-Markov motion, Random Waypoint motion, and Pursue motion. A description of each of these mobility models and their parameterizations is given below.

Nodes in our Brownian motion mobility model change speed and direction at discrete time intervals, such that at the beginning of each interval, each node chooses $r \in [0, v_{max}]$ and $\theta \in (-\pi, \pi]$ and moves with velocity vector $(r \sin\theta, r\cos\theta)$ during that interval. If this movement would cause a node to end the interval beyond the boundaries of the rectangular area, the node instead picks the point within the rectangular boundary closest to the intended destination and moves to that point at the originally chosen velocity. The parameters used in our implementation of this model are given in Table 2.1.

The column mobility model was developed by Sanchez [167]. In our implementation of this model, each node is either moving in the positive x direction or the negative x direction. The initial position of each node i is $(10i, 10i)$, and all nodes start moving in the positive x direction. The

Table 2.2: Parameters for Random Gauss-Markov Motion

Movement interval duration	0.1 s
Initial Velocities	0 m/s
$\overline{v_x} = \overline{v_y}$	0 m/s
$\sigma_{v_x} = \sigma_{v_y}$	10.484 m/s
α	0.9

Table 2.3: Parameters for Random Waypoint Motion

v_{max}	20 m/s
Pause time	0 s

motion of the nodes is divided into discrete intervals, such that at the beginning of each interval, each node chooses $r \in [0, v_{max}]$ and moves with that speed in the same direction as it has been moving. If this movement would cause the node to cross the boundary of the rectangular area, the direction is instead flipped, and the node moves with speed r in the new direction rather than in the original direction. The parameters used in our implementation of this model the same as those used in Brownian motion.

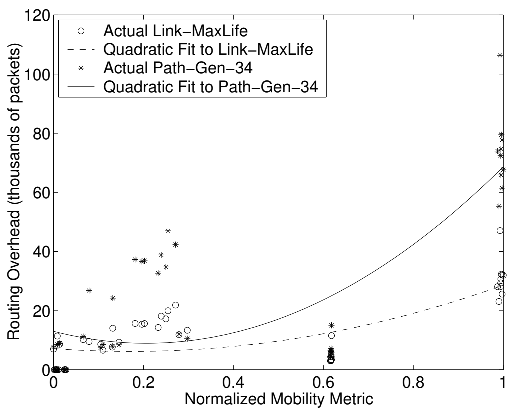
The random Gauss-Markov mobility model was developed by Liang and Haas [107] and was described by Sanchez [168]. The motion of the nodes is divided into discrete time intervals, such that at the beginning of each interval, a node updates its velocity vector as

$$v_{x_t} = \alpha v_{x_{t-1}} + (1-\alpha)\overline{v_x} + R\sqrt{1-\alpha^2}$$
$$v_{y_t} = \alpha v_{y_{t-1}} + (1-\alpha)\overline{v_y} + R\sqrt{1-\alpha^2}$$

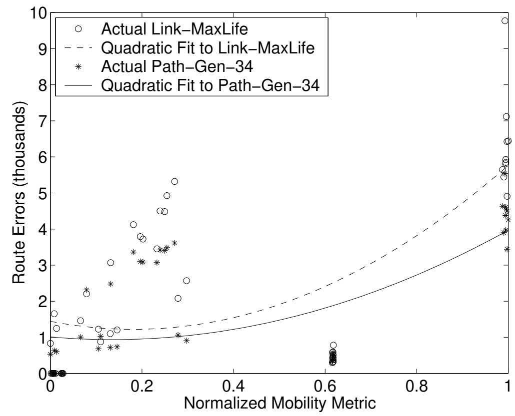
at interval t, where R is a normally distributed random variable with mean 0 and variance σ_{v_x}. When a movement would cause a node to exceed the boundaries of the rectangular area, the sign of the velocity vector in that dimension is flipped. The parameters used in our implementation of this model are given in Table 2.2. The choice of σ_{v_x} and σ_{v_y} was made to have the median of $\|(R_x, R_y)\|_2$ be 10 m/s. α was chosen to be equal to its value in Sanchez' implementation [169].

The random waypoint mobility model was developed by Johnson and Maltz [88]. In this model, a node chooses a destination with a uniform random distribution over the area, moves there with velocity v uniformly distributed over $[0, v_{max}]$, waits there for a *pause time*, and then repeats this behavior. We use a pause time of 0, meaning continuous motion of all nodes, and chose $v_{max} =$ 20 m/s. The parameters used in our implementation of this model are given in Table 2.3.

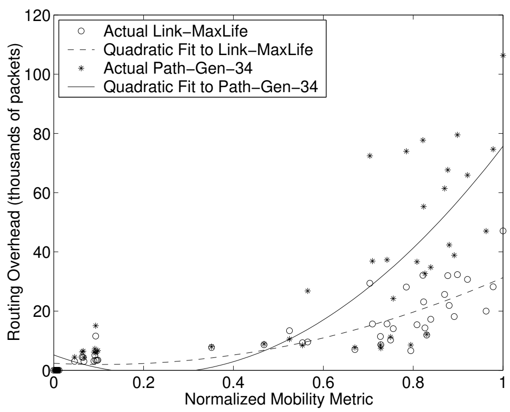
The pursue mobility model was developed by Sanchez [167]. In our implementation of this model, there are 10 groups of 5 nodes each. The motion of the nodes is divided into discrete time intervals, such that in each group, one node moves according to the random waypoint model, and the others attempt to "intercept" that node by choosing their velocity vector at each interval to be toward the point that the target node would be at at the end of the interval, given that the target node would continue to move with the same velocity. The velocity of the pursuing nodes is chosen uniform random for each interval to be in the range $[v_{pmin}, v_{pmax}]$. The parameters used in our implementation of this model are given in Table 2.4.



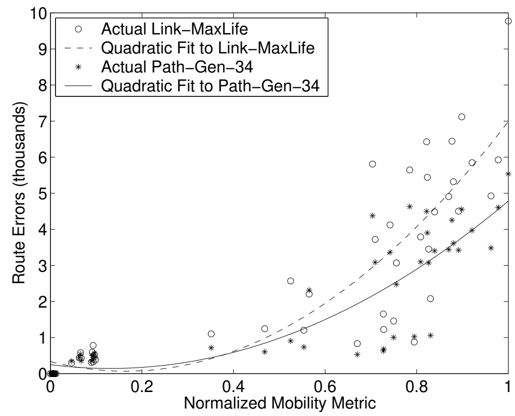
(a) Geometric Mobility Metric
(Packet overhead)



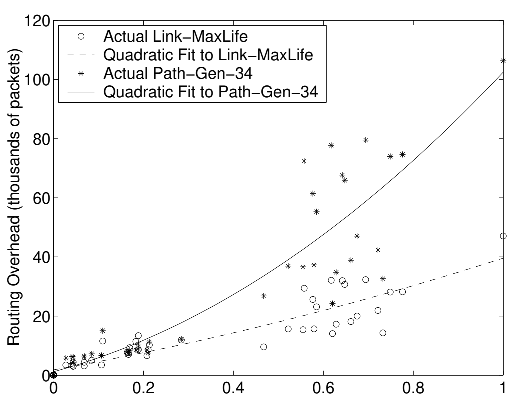
(b) Geometric Mobility Metric
(ROUTE ERRORS)



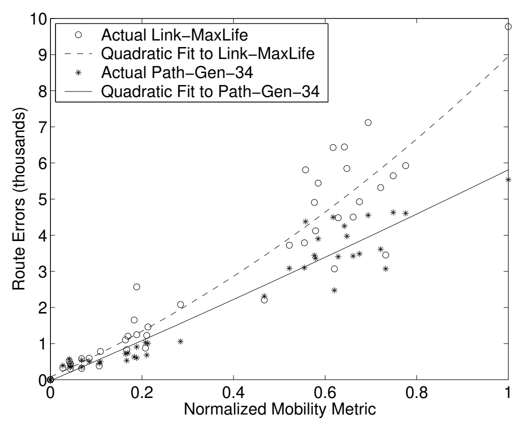
(c) Minimal Route-Change Metric,
All Pairs of Nodes (Packet overhead)



(d) Minimal Route-Change Metric,
All Pairs of Nodes (ROUTE ERRORS)



(e) Minimal Route-Change Metric,
Communicating Pairs (Packet overhead)



(f) Minimal Route-Change Metric,
Communicating Pairs (ROUTE ERRORS)

Figure 2.2: Correlation of Mobility Metrics to Packet Overhead and Number of ROUTE ERRORS

Table 2.4: Parameters Pursue Motion

Movement interval duration	0.1 s
v_{max}	20 m/s
v_{pmin}	5 m/s
v_{pmax}	15 m/s
Pause time	0 s

2.5.2. Evaluation of Mobility Metrics

We evaluated the mobility metrics described in Section 2.4 for each of the scenarios used throughout this chapter. The geometric mobility metric was evaluated with infinite precision using the technique described in Section 2.4. The mobility metrics were normalized so that over all 50 scenarios, the metrics would lie in $[0, 1]$. Figure 2.5.2 summarizes the degree to which the mobility metrics accurately characterize the difficulty of routing across the range of scenarios. Figure 2.2(a) shows the relationship between the normalized all-pairs geometric mobility metric and the routing packet overhead for DSR, for the *Link-MaxLife* and *Path-Gen-34* caching algorithms. Also shown in Figure 2.2(a) is the best quadratic fit to the individual data points, in a least-squares sense, for these two caching algorithms. We show the results for these two caching algorithms here, since *Link-MaxLife* generally performs the best of the adaptive link caching algorithms, and *Path-Gen-34* is representative of the path caching algorithms. Figure 2.2(c) shows the same relationship and type of quadratic fit for the normalized all-pairs minimal route-change metric, and Figure 2.2(e) shows this for the normalized minimal route-change metric summed only over communicating pairs. Similarly, Figures 2.2(b), 2.2(d), and 2.2(f) show the relationship between these three mobility metrics, respectively, and the number of ROUTE ERRORs originated during the simulations. Table 2.5 shows the norm of residuals for the respective quadratic fit for each mobility metric, including also the all-pairs minimal shortest route-change metric and the minimal shortest route-change metric summed only over communicating pairs.

The minimal shortest route-change metric does not reflect well the challenge presented to DSR, since DSR does not attempt to always discover the shortest route. Instead, DSR will continue to use its best route until it breaks or until it overhears a better route.

As shown in Figure 2.5.2 and Table 2.5, the four minimal route-change metrics correlate significantly better, for both routing packet overhead and number of ROUTE ERROR packets, than does the geometric mobility metric, since route changes are a more direct cause of overhead and ROUTE ERRORs than is just geometric mobility. Of the four minimal route-change metrics, the minimal route-change metric summed only over communicating pairs (Figure 2.2(e) and 2.2(f)) correlates best, since summing only among communicating pairs removes pairs which may undergo many route changes but that do not affect the routing algorithm. In addition, since the individual data points on the graphs relative to this metric are reasonably well spread and not tightly clustered, we conclude that the particular movement scenarios used in our study are generally representative of a fairly broad array of possible scenarios within the bounds used by these scenarios.

Although the four minimal route-change metrics correlate well to both the routing packet overhead and the number of ROUTE ERRORs, it correlates better for the number of ROUTE ERRORs. We believe this difference is due to the variable number of ROUTE REQUEST packets that may be sent as part of a Route Discovery, depending on the degree of containment of the ROUTE REQUEST flood that DSR is able to acheive for each individual Discovery attempt. We also examined the correlation of these metrics specifically to the number of Route Discoveries performed, and found fairly

Table 2.5: Norm of Residuals for Quadratic Fits of Packet Overhead and Number of ROUTE ERRORS

Path-Gen-34	Packet Overhead	ERRORS
Geometric	120,248	9,189
Min Route-Change over All Pairs	111,699	5,973
Min Route-Change over Comm Pairs	77,144	2,877
Min Shortest Route-Change over All Pairs	168,729	10,799
Min Shortest Route-Change over Comm Pairs	160,027	9,782
Link-MaxLife		
Geometric	53,392	12,896
Min Route-Change over All Pairs	40,988	8,282
Min Route-Change over Comm Pairs	32,478	5,219
Min Shortest Route-Change over All Pairs	64,697	15,291
Min Shortest Route-Change over Comm Pairs	65,668	14,814
OmniExp		
Geometric	18,963	116
Min Route-Change over All Pairs	17,616	105
Min Route-Change over Comm Pairs	17,885	106
Min Shortest Route-Change over All Pairs	19,988	116
Min Shortest Route-Change over Comm Pairs	23,093	122

good correlation for *Path-Gen-34* but not for *Link-MaxLife*, which we attribute to the very small, statistically insigificant number of Route Discoveries needed by *Link-MaxLife*. Even a change as small as 1 in number of Route Discoveries for any scenario with *Link-MaxLife* will result in a large relative change in the total, making correlation of any mobility metric difficult.

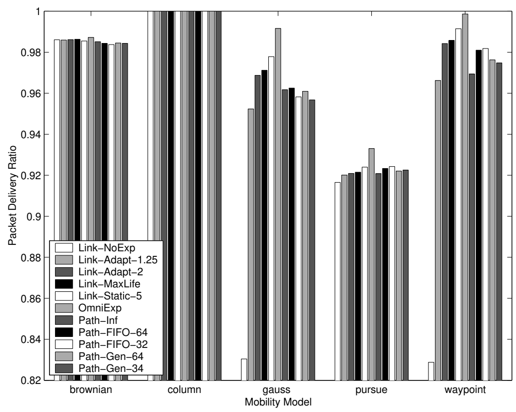
Another exception in the degree of correlation of the mobility metrics is those results obtained using the *OmniExp* caching algorithm, for all of the performance indicators that we studied for the routing protocol. For all indicators, *OmniExp* had relatively low correlation, since this caching algorithm creates very few Route Discoveries and even fewer ROUTE ERROR packets (and thus very small total routing packet overhead). As with the number of Route Discoveries needed by *Link-MaxLife*, as described above, all of the performance indicators that we studied for the routing protocol with *OmniExp* are not statistically sigificant.

2.6. Simulation Results

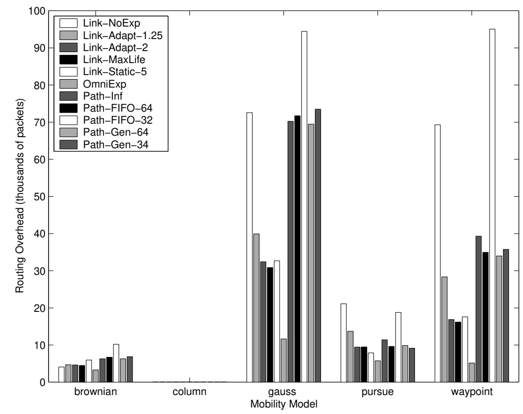
2.6.1. Overview of the Results

For each of the caching algorithms presented in Section 2.2, we ran 10 different scenarios of each mobility model described in Section 2.5.1. Figure 2.3(a) shows the packet delivery ratio achieved by each caching algorithm, averaged over the 10 scenarios for each mobility model. Figure 2.3(b) shows the average routing packet overhead, Figure 2.3(c) shows average latency, and Figure 2.3(d) shows the path optimality for each of the caching algorithms over the 10 scenarios from each mobility model, normalized and averaged over the 5 mobility models.

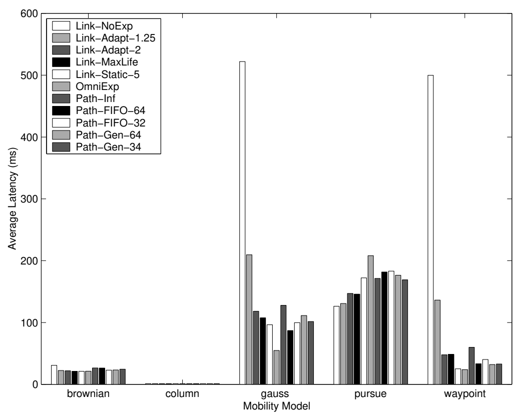
The *Link-Static-5* caching algorithm uses only a single fixed value for the cache timeout, although in general, no single value can perform best for all nodes in all ad hoc networks in all circumstances. In addition to the timeout value of 5 seconds shown in our results, we also evaluated a number of other timeouts ranging from 1 second to 40 seconds, and found that in our scenarios,



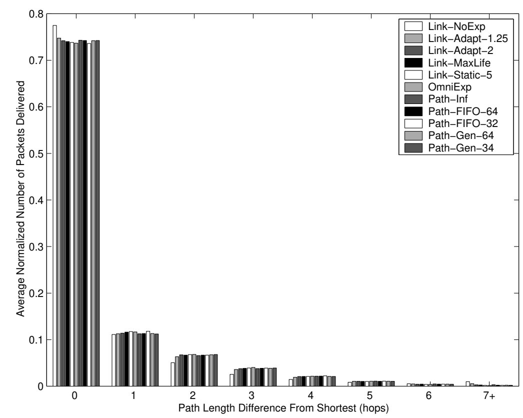
(a) Packet Delivery Ratio vs. Cache Type



(b) Routing Packet Overhead vs. Cache Type



(c) Average Latency vs. Cache Type



(d) Normalized Path Optimality vs. Cache Type

Figure 2.3: Performance of the Different Caching Algorithms on the Mobility Models

the 5-second timeout performed best in terms of packet delivery ratio. As shown in Figure 2.6.1, our scenarios represent a range of different challenges for the routing protocol, but in each of our individual scenarios, all nodes in a given scenario move according to the same pattern. Thus, the advantage of an algorithm that can adapt to different timeouts for different links (between different pairs of nodes) was not fully exercised. As a result, we simply present the results for *Link-Static-5* and omit further discussion of them in this chapter.

Although the column mobility model creates a large amount of motion among the nodes, there is very little *relative motion* among them and thus very little challenge to any of the caching algorithms. In fact, in each of our scenarios using the column mobility model, only 18 Route Discoveries were performed, regardless of the caching algorithm used. This property of the column model can also be seen using our new mobility metrics defined in Section 2.4; for example, the average geometric mobility metric over our column scenarios is 79.35% that of our random waypoint scenarios, appearing to indicate a comparable amount of mobility, yet when compared using our all-pairs minimal route-change mobility metric, this number drops to only 2.82%.

In the pursue mobility model, the network remains partitioned much of the time; the 5 nodes in each group stay very close to each other, while the 10 separate groups are free to move over the entire simulation area, often leaving large, unoccupied spaces between the groups. For example, across all of our scenarios using the pursue mobility model, the network is partitioned on average 76.07% of the time. Due to this high occurrence of partition, the behavior of any caching algorithm used with DSR will be very different than in more typically connected networks, making comparison of different caching algorithms in these scenarios difficult.

In the remainder of this chapter, we therefore focus in our analysis on only the scenarios using the Brownian, random Gauss-Markov, and random waypoint mobility models. As mentioned above, we ran 10 different scenarios for each of these mobility models for each of our caching algorithms.

2.6.2. Effects of Cache Structure

For the packet delivery ratio metric (Figure 2.3(a)), the *Link-Adapt-2* and *Link-MaxLife* link caching algorithms outperform all path caches, obtaining higher packet delivery ratio than the best path cache, evaluated individually for each mobility model. In most cases, *Link-MaxLife* performs somewhat better than *Link-Adapt-2*.

Similarly, for the packet overhead metric (Figure 2.3(b)), *Link-Adapt-2* and *Link-MaxLife* outperform all path caches, obtaining in most cases a reduction in packet overhead by a factor of about 2 or more over the best path cache for each mobility model. In addition, *Link-Adapt-1.25* performs better than the best path cache for each mobility model, although not by as much as do *Link-Adapt-2* and *Link-MaxLife*. This is consistent with the design intent of link caches over path caches, as link caches remove only a single link in response to a ROUTE ERROR (rather than removing a whole path) and are able to combine information from different Route Discoveries to form new routes from the cached information.

In the scenarios that we studied, two primary factors contribute to the total latency experienced by a packet: the time spent by the packet waiting for a Route Discovery to complete before the packet can be sent, and the time spent in Route Maintenance detecting (through retransmissions) broken links and performing salvaging. For our Brownian motion scenarios, the dominant factor of these two is Route Discovery, which favors the link caches for low latency, since link caches generally perform fewer Discoveries than path caches due to the increase in information that can be represented in the cache. For example, *Link-Adapt-1.25* (the *highest*-latency adpative link cache) performs on average 431.5 fewer Route Discoveries than *Path-FIFO-32* (the *lowest*-latency path cache) in these scenarios, but it sends on average only 130.7 more ROUTE ERRORS. For the Gauss and random waypoint scenarios, however, the number of ROUTE ERRORS becomes significant in the link caches, particularly for the *Link-NoExp* and *Link-Adapt-1.25*. For example, *Link-NoExp* and *Link-Adapt-1.25*, respectively, cause 31,117 and 10,652 ROUTE ERRORS, yet *Path-FIFO-32* (the highest-latency non-infinite path cache) causes only 1,973 ROUTE ERRORS.

All of the caching algorithms achive good path optimality, and the differences between the results with different caching algorithms is small. In particular, the 5 path caching algorithms perform almost identically on most scenarios. However, for the link caching algorithms, path optimality differs for the *Link-NoExp* and *Link-Adapt-1.25* algorithms; these algorithms deliver a greater fraction of packets along optimal routes (path optimality 0) than do the other caching algorithms, yet also deliver a greater fraction of packets along routes 6 and 7 or more hops longer than optimal than do the other algorithms.

Both of these algorithms are able to keep a large number of unused links in the cache, as *Link-NoExp* never times out such links and *Link-Adapt-1.25* increases the node stability metrics (and thus the link cache lifetimes) much more aggressively than it decreases them. As such, these algorithms are able to opportunistically combine results from different Route Discoveries and from other routing information learned from packets forwarded or overhead, in order to more often find the shortest route that exists. However, the many unused links that these algorithms can keep in the cache also at times are a liability; many of these links may be broken, increasing the number of packets that must be salvaged multiple times, and thus increasing the total hop count for salvaged packets that are ultimately successfully delivered. In our simulations, each packet was prevented from being salvaged more than 15 times, in order to prevent the packet from possibly looping yet also allow alternate routing and backtracking of the packet in the presence of some stale cached links.

Overall, *Link-MaxLife* outperforms the other caching algorithms on the set of performance metrics and scenarios studied. As an adaptive algorithm, it is able to adjust the cache timeout values for each node individually, depending on the node's behavior, and is not limited to a static timeout or a fixed capacity cache replacement policy. By also taking advantage of the lifetime values in the route selection algorithm to differentiate between multiple routes of equal length, *Link-MaxLife* attempts to avoid using routes that may soon result in a ROUTE ERROR and a possible new Route Discovery. For example, in 26 of the 30 scenarios, *Link-MaxLife* experiences fewer Route Errors than *Link-Adapt-2*, where *Link-Adapt-2* is the same algorithm as *Link-MaxLife* without the use of lifetimes in route selection. In addition, this route selection tends to spread the use of routes over all cached routes of equal length; the route selection in *Link-Adapt-2*, on the other hand, is based only on Dijkstra's algorithm and thus always finds the first route from a total order on all routes of shortest length.

2.6.3. Effects of Cache Capacity

For 3 of the 5 types of mobility models, the *Path-Inf* caching algorithm, with its unlimited cache size, performs much worse than the other path caches (with limited cache sizes) with respect to packet delivery ratio, as shown in Figure 2.3(a). These 3 types of mobility models (Gauss, pursue, and waypoint), are generally quite dynamic, creating many broken links, all of which are retained in each node's cache once discovered or overheard. Many of the routes that these nodes select to use from their caches, are thus likely to be broken even before the first packet is sent on them, severely reducing the effectiveness of DSR's packet salvaging mechanism and resulting in many dropped packets. Movement scenarios from the other two mobility models (Brownian and column) involve substantially less relative mobility, and all path caches perform well on these scenarios.

Average latency and routing packet overhead metrics for the *Path-Inf* algorithm also suffer in dynamic situations, as shown in Figures 2.3(b) and 2.3(c). Sending a ROUTE ERROR typically counts as several packets of overhead since it must in general traverse several hops. In addition, when a packet is salvaged, the combined route traveled by the packet will typically be longer than the original route with which the packet was sent; when a packet must be salvaged multiple times, the resulting routes can be quite long, causing significant increases in latency, and for each time a packet is salvaged, another ROUTE ERROR is returned to the original sender of the packet.

For the FIFO cache replacement policies studied here for path caches, no one cache size provides the best packet delivery ratio for all mobility models. For mobility models with large amounts of relative mobility, many Route Discoveries take place, causing a rapid turnover in each node's cache as it replaces existing cache entries with new entries learned from its own Route Discoveries or from others that it has overheard. This cache replacement is in effect a form of adaptation in the caching algorithm, since as the amount of mobility in the network increases, the average number of broken routes created in the network increases and the average time that entries remain in a node's cache decreases with the cache turnover. However, with cache capacity as the limiting factor causing increased cache turnover, the FIFO caching algorithm has little control over *which* cache entry is replaced at which time. In particular, in a movement scenario with highly non-uniform behavior between different nodes, FIFO cache replacement forces the replacement of all paths (containing nodes with different behaviors) to be treated equally.

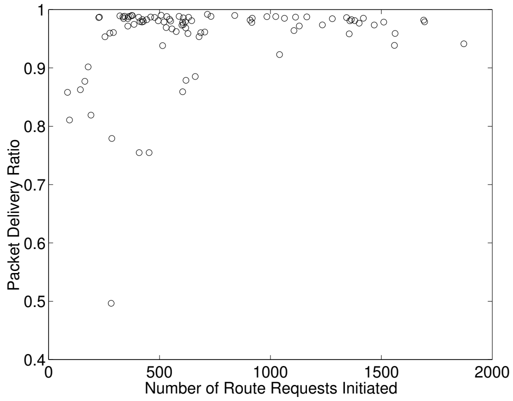
2.6.4. Effects of Cache Timeout

The use of a timeout on each cache entry in a link cache has a similar effect in cache replacement as the use of limited capacity has in a path cache, as described above. For example, the *Link-NoExp* algorithm, which has no timeout on cache entries, performs poorly with respect to packet delivery ratio for scenarios from the Gauss, pursue, and random waypoint mobility models, as shown in Figure 2.3(a). The movement in these 3 models are generally quite dynamic, causing many of the routes that a node selects to use from its cache to be broken even before the first packet is sent on it, causing the same type of ineffective salvaging and dropped packets as occurred when using the *Path-Inf* algorithm, with its unlimited path cache size. In addition, for algorithms using an adaptive cache timeout, since the timeout used for a cache entry is a function of the expected future lifetime of that entry, based on the ROUTE ERRORS observed by this node, as the amount of mobility in the network increases, the average number of broken routes created in the network increases and the average time that entries remain in a node's cache decreases.

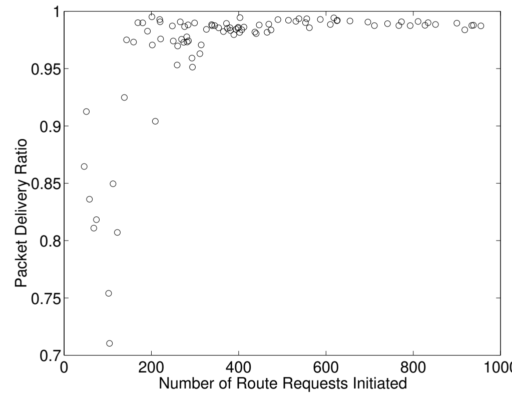
In order to assess the effect of different link cache entry timeout values on the number of ROUTE REQUESTS initiated and the number of ROUTE ERRORS sent, we ran additional simulations to collect results for each of 7 different static cache timeout values: $1, 2, 5, 10, 20, 40$ seconds, and no timeout. For each of these timeout values, we simulated each of the 10 Gauss and 10 waypoint scenarios used in the previous sections. Figure 2.6.4(a) shows the packet delivery ratio vs. the number of ROUTE REQUESTS initiated for the these Gauss simulations together with the Gauss simulations using *Link-Adapt-1.25*, *Link-Adapt-2*, and *Link-MaxLife* described above. Figure 2.6.4(b) shows the routing packet overhead, and Figure 2.6.4(c) shows the number of ROUTE ERRORS generated, for these same simulations.

As the average cache timeout value decreases, nodes generally initiate more Route Discoveries and cause more ROUTE REQUEST packets to be sent, since entries in the cache needed for future packets may be deleted. Since the results shown in Figure 2.4(a) are all simulated from the same set of 10 Gauss scenarios (with the same communication scenario and with movement scenarios all generated from the same Gauss mobility model), the results are all roughly comparable except for the differences in the cache timeout values used. That is, for these scenarios, below some average rate of initiating ROUTE REQUESTS, the routing protocol is unable to discover new routes at the rate at which the links needed for routing break. For example, if a very long cache timeout is used, a broken link may stay in a node's cache and may be used in an attempt to route a future packet, hurting the overall packet delivery ratio; if a very short cache timeout is used, links will be deleted from the cache before they are broken, creating extra overhead due to increased numbers of Route Discoveries, but the packet delivery ratio will not be affected since packets are buffered until a pending Route Discovery completes and a ROUTE REPLY is received (unless the ROUTE REQUEST load is extremely high, in which case network congestion may result). In Figure 2.4(a), if less than about 350 ROUTE REQUESTS are initiated in a scenario, the packet delivery ratio decreases sharply.

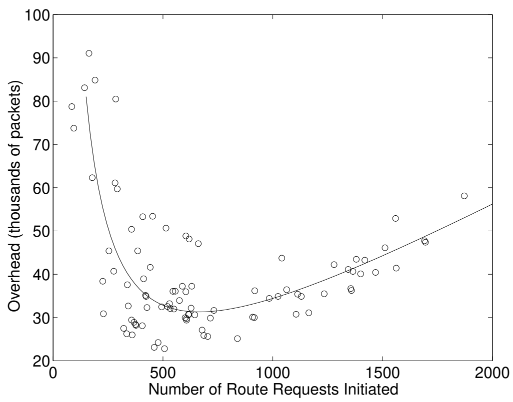
A similar effect is shown in Figure 2.4(c), where below about 350 ROUTE REQUESTS initiated, the routing packet overhead rises sharply. This effect is due to the increased number of ROUTE ERRORS that arise from using an increased number of cached links that are broken but have not yet been removed from the cache due to the long cache timeout value. In addition, as the number of ROUTE REQUESTS initiated increases, the routing packet overhead increases, since each transmission of each ROUTE REQUEST packet contributes to this overhead; above about 350 ROUTE REQUESTS initiated, the routing overhead in Figure 2.4(c) gradually rises.



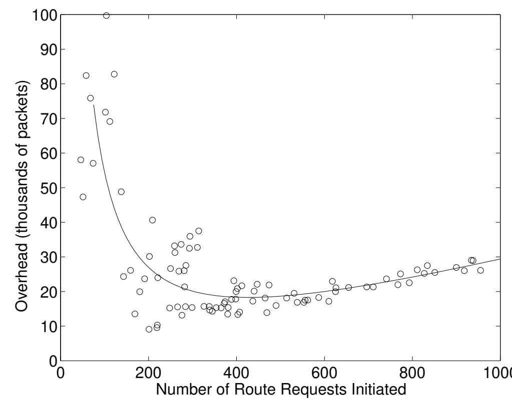
(a) Packet Delivery Ratio vs. Number of ROUTE REQUESTs Initiated (Gauss mobility model)



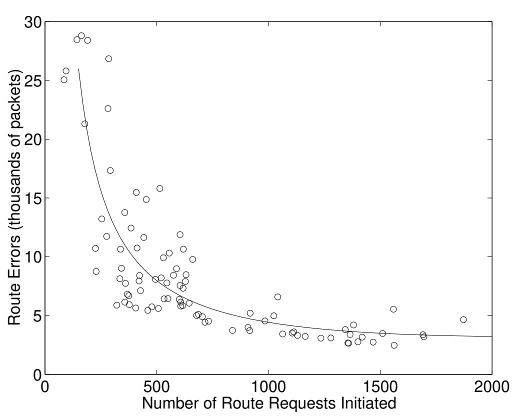
(b) Packet Delivery Ratio vs. Number of ROUTE REQUESTs Initiated (Random waypoint mobility model)



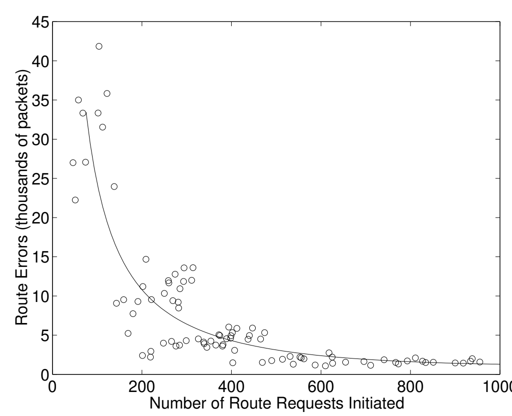
(c) Packet Overhead vs. Number of ROUTE REQUESTs Initiated (Gauss mobility model)



(d) Packet Overhead vs. Number of ROUTE REQUESTs Initiated (Random waypoint mobility model)



(e) Number of ROUTE ERRORs vs. Number of ROUTE REQUESTs Initiated (Gauss mobility model)



(f) Number of ROUTE ERRORs vs. Number of ROUTE REQUESTs Initiated (Random waypoint mobility model)

Figure 2.4: Correlation between Number of Route Requests Initiated and Three Different Performance Factors of the Routing Protocol, for Gauss and Random Waypoint Mobility Models

The effect shown in Figure 2.4(e) is also similar to that shown in Figure 2.4(a). Below about 350 ROUTE REQUESTS initiated, many cached links that have broken have not been removed from the cache, resulting in a rising number of ROUTE ERRORS generated. As the number of ROUTE REQUESTS initiated increases above about 350, the number of ROUTE ERRORS generated gradually decreases, up to about 1100 ROUTE REQUESTS initiated, beyond which there is a diminishing return from additional ROUTE REQUESTS initiated.

Finally, Figure 2.4(b) shows the results from a set of simulation runs the same as Figure 2.4(c), but using the random waypoint mobility model rather than the Gauss model. The basic results are very similar to those shown for Gauss, but the inflection point, where the routing packet overhead begins to rise, is lower. This difference suggests that a single average rate of initiating ROUTE REQUESTS and a single average cache timeout value is not optimal for all mobility models.

2.7. Related Work

This work has been expanded in many ways since it was published [74]; Camp et al [28] further study mobility models and metrics, Boleng et al [18] study cache adaptation strategies, Turget et al [185] approximate link lifetime given knowledge of the mobility pattern and Marina et al [115] study other cache-related optimizations to DSR. In recent work, we have examined the possibility of using epoch numbers to impose a strict ordering link discovery and link breakage at each link. This information can be used to somewhat reduce the spread of stale cache information, though it substantially increases the overhead of passing routes around, especially in the source route [76].

2.8. Chapter Summary

Key to the performance of many on-demand ad hoc network routing protocols is the design of an appropriate caching strategy for the protocol that can make effective use of the state information about the network collected by the protocol as part of the process of discovering routes to other nodes. Caching is important in order to avoid the overhead of discovering a new route before sending each data packet, but caching also brings with it the risk and associated expenses of retaining routing information in a cache after the information is no longer valid due to changes in different nodes' positions or changes in the wireless propagation environment.

This chapter has presented an analysis of the effects of different design choices in caching strategies for on-demand routing protocols in wireless ad hoc networks, dividing the problem into choices of cache *structure*, cache *capacity*, and cache *timeout*. Our analysis is based on the Dynamic Source Routing protocol (DSR) which operates entirely on-demand. Using detailed simulations of wireless ad hoc networks of 50 mobile nodes, we studied a large number of different caching algorithms that utilize a range of caching strategy design choices, and simulated each cache primarily over a set of 50 different movement scenarios drawn from 5 different types of mobility models. Our evaluations include the packet delivery ratio, routing packet overhead, average latency, and path optimality relative to the shortest path, achieved by each caching algorithm.

We found that adaptive caches are often able to significantly outperform static caches, and that by utilizing a cache data structure based on a graph representation of individual links, rather than based on complete paths through the network, the routing protocol was much better able to make use of the potential information available to it. In addition, we identified some subtle relationships between cache timeout policies and cache capacity limits, and between these choices and some performance metrics for the routing protocol, most notably the packet delivery ratio and the routing packet overhead caused by the routing protocol. Somewhat unexpectedly, we also found a strong indication that caches of unlimited capacity or with no cache timeout perform substantially *worse* than caches with reasonable capacity or timeout limits.

Chapter 3

Implicit Source Routes

In an ad hoc network, the use of *source routing* can provide many advantages, including simplicity, correctness, and flexibility [87–90]. For example, since all routing decisions for a packet are made by the sender of the packet, intermediate nodes that forward it need not maintain up-to-date, consistent routing tables for the destination. Forwarding at each hop consists simply of locally transmitting the packet to the next address indicated in the source route in the packet's header; the sequence of hops over which any packet is forwarded can easily be guaranteed to be loop-free by not allowing duplicates in the list of hops. By including the source route in the packet's header, additional routing information is also partially spread around the network without requiring additional packets to be transmitted. In addition, for reasons such as load balancing or differentiated treatment of different types or classes of packets for Quality of Service (QoS), it is possible for the sender to use different routes for different packets, without requiring coordination or explicit support by the intermediate nodes.

However, source routing has the disadvantage of increased per-packet overhead. With source routing, the size of each packet is increased in order to carry the source route of hops through which the packet is to be forwarded. Since the source route is always present in the packet, the extra network overhead caused by the presence of the source route is incurred not only when the packet is originated, but also each time it is forwarded to the next hop. This extra network overhead decreases the bandwidth available for transmission of data, increases the transmission latency of each packet, and consumes extra battery power in the network transmitter and receiver hardware.

In this chapter, I describe and analyze the use in ad hoc networks of *implicit* source routing. This technique preserves the advantages of source routing while improving service in source routing protocols in two ways: first, it greatly simplifies packet classification by allowing a flow identifier to be correlated with each packet, and second, it reduces network overhead, providing better throughput and latency. In a manner in part similar to techniques used for MPLS [166] or ATM virtual circuits [179], each packet is tagged with a *flow identifier* when the packet is sent by its original sender. The flow identifier indicates the route to be followed by this and all packets belonging to a logical flow from this sender to the destination of the packets. Intermediate nodes along the route retain *soft state* indicating the next hop to which packets belonging to that flow should be forwarded, avoiding the need to carry the full source route in each packet. This soft state should be retained until the specified timeout but may be discarded earlier, for example due to node failures, without impacting correctness.

We base our design and analysis of implicit source routing on extending the Dynamic Source Routing protocol (DSR) (Section 1.2.2), since it is based on source routing and has been shown by a number of groups to perform well when compared to other protocols [26, 86]. DSR allows nodes to dynamically discover, on demand, source routes to nodes to which they send packets, and allows these source routes to be maintained when links between nodes break due to node mobility, wireless propagation changes, or other factors. Our implicit source routing mechanism fits naturally into the existing structure of the DSR protocol [77] and preserves the important fundamental properties of DSR's operation. To evaluate our implicit source routing design, we conducted a set of detailed simulations of DSR, both with and without use of implicit source routing, and we analyze the differences in the behavior of these two protocols in terms of packet delivery ratio, latency, path optimality, and packet and byte routing overheads.

The addition of *implicit* source routing to DSR described in this chapter preserves the basic operation of DSR's Route Discovery and Route Maintenance mechanisms, including all of these important resulting properties of DSR. Although DSR with implicit source routing may be seen as similar to other on-demand routing protocols that do not use source routing, in practice all routes used in either version of DSR are still discovered and established as source routes, with the complete sequence of hops determined from the source to the destination. For example, AODV [141] borrows features of DSR's on-demand Route Discovery mechanism, but it uses only hop-by-hop routes and is not based on source routing; none of the properties of DSR described in this section hold for AODV.

3.1. Implicit Source Routing Operation

Conceptually, in implicit source routing, a tuple $\langle$source address, destination address, flow identifier$\rangle$ takes the place of the full source route in each packet. The source address and destination address can be placed in the IP header, and the flow identifier is placed in a special header. Each node participating in implicit source routing has a *Flow Table*, with one entry for each flow forwarded by that node. A Flow Table entry minimally must record the next hop address to which a packet for this flow should be forwarded, in addition to the source address, destination address, and flow identifier for this flow.

A source can establish a new flow by sending a *flow establishment packet*. A flow establishment packet is a packet with two headers (i.e., extension headers or options): one containing the flow identifier, and the other containing a source route and a timeout for the flow. When an intermediate node forwards such a packet, in addition to forwarding it according to the source route information, it creates a Flow Table entry for this flow and inserts the necessary information from the packet. A flow establishment packet is normally sent by including these two headers in an existing packet to be sent along that source route; it is also possible (but not necessary) to send the flow establishment packet as a special control packet along the source route.

A node is required to remove a flow's entry from the Flow Table when that node has not forwarded packets for that flow for a period of time specified by the timeout for the flow. As a result, a Flow Table entry is also required to keep the timeout, as well as the time at which this flow is to expire.

A source that has already sent one or more flow establishment packets for a given flow may decide that each node along that flow has established a Flow Table entry for that flow. This source may then send any subsequent packets routed solely by implicit source routing by adding a flow identifier header in lieu of a source route in each packet. A node forwarding a packet sent using implicit source routing checks its Flow Table for an entry corresponding to the flow identifier in the packet. If the node finds one, it forwards the packet by setting the MAC-layer destination address to the MAC address of the next hop indicated in the matching Flow Table entry. Otherwise, it sends a FLOW UNKNOWN error back to the source of the packet.

A source node receiving a FLOW UNKNOWN error addressed to itself marks its Flow Table entry for this flow to indicate that the flow must be reestablished. For the purposes of our simulation, we send three establishment packets when the flow is first created, and three establishment packets each time a FLOW UNKNOWN error is received; each flow establishment packet is sent only when there is data to be transmitted along the flow. By repeating the flow establishment packet for the first three data packets sent when the flow is established or reestablished, the protocol is able to tolerate loss of some flow establishment packets without triggering the overhead of a FLOW UNKNOWN error and the resulting latency for re-establishment.

When a packet is sent using implicit source routing forwarding, it still requires some small amount of overhead in the packet. These additional header bytes in the packet can be entirely eliminated by the use of *default flows*. Conceptually, a node is most likely to use a flow more recently established. Therefore, our protocol allows the use of the most recently established flow with no per-packet overhead for most packets.

To enable this, each node keeps a *Default Flow Table*. For each ⟨source address, destination address⟩ pair, a node keeps the greatest flow identifier for which it has sent or forwarded packets, as well as the expected TTL value of packets sent along the default flow (alternatively, expected TTL value can be stored in the Flow Table).

In order to allow the same flow to be used for both default flow forwarding as well as basic flow forwarding, the expected TTL in the Default Flow Table must be set only upon hearing a flow establishment packet. Additionally, we constrain a source wishing to use a given flow as a default flow to set the TTL of all flow establishment packets for that flow to the same value, and we disallow the use of default flow routing along paths that do not reduce the TTL value in forwarded packets by exactly one at each hop.

When a source node originates a packet along a route that is the default flow for that ⟨source, destination⟩ pair, and that packet has the same TTL as the flow establishment packets for that flow, it transmits the packet to the next hop specified in the Flow Table.

When a node receives a packet with neither a source route nor a flow identifier header, and it is not the node named in the IP Destination Address field, it checks its Default Flow Table; if it finds a flow for the ⟨source, destination⟩ pair specified in the IP header, and if the expected TTL matches the actual IP header TTL, the packet is processed as if it had a flow header specifying the flow identifier found in the Default Flow Table as the packet's flow identifier. Otherwise, a DEFAULT FLOW UNKNOWN error is returned to the source of the packet. A source node receiving a DEFAULT FLOW UNKNOWN error addressed to itself marks its Flow Table entry for the default flow to indicate that the flow must be reestablished.

A Default Flow Table entry at a node times out when all flows corresponding to the ⟨source, destination⟩ pair time out at that node, although this is necessary for correctness only when nodes may crash and lose their state, or when flow identifiers may wrap around.

As described in Section 1.2.2, in the base version of DSR [90], a mechanism exists by which routes actively being used can be shortened in certain ways. Specifically, when the transmission of a packet is promiscuously overheard by a node in the source route, that node determines if the packet is downstream of it (that is, has already been forwarded by it) or upstream of it (that is, has yet to be forwarded by it). If the packet is upstream of it, the route can be shortened by removing the intervening hops not yet traversed leading to this node's receipt of the packet. In this case, the node generates a DSR "gratuitous" ROUTE REPLY to the source of the packet, indicating the shortened route. For example, suppose node **S** is using the route **S** → **A** → **B** → **C** → **D** to send packets to destination node **D**; if node **C** overhears a packet from **S** being forwarded by node **A**, node **C** can return a gratuitous ROUTE REPLY to **S** providing the shorter route **S** → **A** → **C** → **D** for use with subsequent packets to **D**.

This mechanism, called *automatic route shortening*, takes advantage of the source route in each packet to determine whether or not a packet is upstream of a node that promiscuously overheard the packet. In order to enable automatic route shortening when no source route is present, a Hop Count field is added to the flow identifier header. A source node originating a packet initializes the Hop Count field to 0, and each node forwarding this packet using implicit source routing increments the Hop Count by 1. At each node, the expected Hop Count value, as well as the complete source route, is stored in the node's Flow Table during flow establishment. Packets overheard upstream are determined to be those that have a Hop Count *less than* the Hop Count in the corresponding Flow Table entry.

Similarly, automatic route shortening is possible for packets sent along default flows by examining the TTL; if the TTL value is *greater than* expected, it is considered to be upstream. Using the IP TTL for automatic route shortening for packets sent along a default flow avoids the need to carry *any* header information in the packet not normally already present in an IP header; we use an explicit Hop Count rather than the IP TTL in non-default flow routed packets to avoid placing any restriction on how the IP TTL field is used in forwarding the packet and to handle the case in which the IP TTL is used in some non-standard way by some hop along the flow (e.g., for nodes that decrement the TTL by more than 1 when forwarding the packet).

When a node promiscuously overhears a packet, it searches the packet for a flow identifier by looking in the flow header, if present, or by searching its Default Flow Table. If the packet is determined to be upstream of this node, the node stores a record in its limited-size *Automatic Route Shortening Table*, with the source and destination addresses, the flow identifier, the packet, and the number of hops by which the route could have been shortened. In order to reduce the possibility of polluting the source's Route Cache, a gratuitous ROUTE REPLY is sent only when the packet is forwarded by the node that previously overheard the packet.

DSR takes advantage of aggressively caching overheard routes in order to maintain high packet delivery ratio and low overhead. However, with implicit source routing, source routes do not appear in the majority of packets, lessening the opportunity for these routes to be overheard and cached by other nodes. As a heuristic to give DSR with implicit source routing a better ability to utilize these optimizations, we chose to send any data packet sent along a flow as a flow establishment packet (containing both a flow identifier header and a source route header) if an establishment packet has not been sent along that flow within the last 5 seconds.

Implicit source routing also has an effect on DSR's salvaging optimization. Since packets sent with both a flow identifier header and a source route header are considered to be establishment packets, a salvaging node must remove any existing flow identifier header. Furthermore, since flow identifiers may only be assigned at the source, intermediate nodes may not salvage by using a flow header and must instead use a full source route header for salvaging.

3.1.1. Correctness

In this section, we give a proof of the correctness properties of the implicit source routing mechanism.

Claim 1. No packet will be received by a single node twice while being forwarded using a flow identifier header.

Proof 1. Proof is by contradiction. Consider the first node that received a packet twice; call it **A**. Node **A** must have been sent the packet the first time by some node **B** and some second time by some node **C**. Now **B** and **C** are distinct nodes, since otherwise that node would have received the packet twice, prior to **A** receiving the packet twice. Since the next hop for an explicitly specified flow is the next hop in the source route of the flow establishment packet, then in the source route of the flow establishment packet, the address of **A** occurred immediately after the address of node **B**, and occurred immediately after the address of node **C**. Since we require source routes to be loop-free, this is impossible, which completes the proof.

Claim 2. After a packet *p* has been received by the same node **A** twice since the most recent default route change for that packet's $\langle \text{source}, \text{ destination} \rangle$ pair at **A**, packet *p* will no longer be forwarded using default flow forwarding.

Proof 2. Proof is by contradiction. Assume that there is a packet *p* that has been received by **A** twice since the time of the most recent default route change at **A**. We define t_s to be the time of the most recent default route change for this $\langle \text{source}, \text{ destination} \rangle$ pair at **A**.

Both times packet *p* was received, **A**'s default flow had the same expected TTL, since no default flow changes occurred since t_s. Let this expected TTL be TTL_e.

The first time *p* was received after t_s, the TTL must have been equal to TTL_e, since otherwise **A** would not have forwarded the packet using the default flow mechanism. Also, since TTL is strictly monotone decreasing, the second time **A** receives *p*, its TTL must be some value less than TTL_e. Node **A** would see that *p* does not have the correct TTL for this default flow and would stop forwarding the packet using default flow forwarding.

Claim 3. A routing loop in the default flow forwarding mechanism cannot persist indefinitely after the last default route change at any point in the network.

Proof 3. Whenever a transient routing loop is stopped in the way described in Proof 2, a DEFAULT FLOW UNKNOWN error is sent to the source of the packet. After one of those errors reaches the source, the source will attempt to reestablish the default flow. When the default flow has been reestablished across the entire source route, packets sent along it will not loop until there are further routing changes.

Protocols that rely entirely on hop-by-hop, per-flow state to forward packets are unable to detect routing loops if routing state continues to change in certain ways *while* the packet is in transit. In the implicit source routing extensions described above, a source particularly concerned about looping in such a fashion can send all packets with a flow identifier header for 4 bytes of overhead per packet for the flow identifier header.

3.2. Evaluation Methodology

To evaluate our implicit source routing mechanism, we utilized the *ns-2* network simulator [50], together with our Monarch Project wireless and mobile *ns-2* extensions (Section 2.3.1), to compare the behavior and performance of DSR with implicit source routing against the original operation of DSR without it. Our simulation parameters are as described in Section 2.3 with the exception of those parameters described here. In this chapter, we present results from 40 randomly generated scenarios at each mobility level (pause time). Nodes in our simulations move according to the Random Waypoint model (Section 2.5.1), with maximum speed of 20 m/s. We varied pause time between 0 s (a continuously moving network) and 900 s (a stationary network). Specifically, the following seven pause time values were used in our simulations: 0, 30, 60, 120, 300, 600, and 900 s. Each of the 280 scenarios used in our simulations was generated in advance, allowing identical scenarios to be used in the simulations of each version of the protocol. We used the same communication pattern as in Section 2.3.2, except that our packets had a 512 byte data payload rather than a 64 byte payload.

For the base version of DSR without implicit source routing, we used the simulation code from the simulations described in Section 2. The Route Cache we used is the "Link-MaxLife" cache, which is a link state cache in which link timeouts are chosen dynamically based on observed usage and errors. The cache also attempts to choose the longest-lived shortest path when searching the cache for a path to the destination. As described in Section 2, the base version of DSR with this Route Cache to performs very well. For example, in ad hoc networks of 50 mobile nodes moving continuously at maximum speeds of 20 m/s (average 10 m/s), over 98% of originated data packets are delivered.

The simulated implicit source routing protocol is based on the same version of the DSR simulation code, using the same Link-MaxLife cache (Section 2.2.2). However, in this case, the route selection made by the cache does not override the default route unless the selection is shorter than the current default route. All packets are sent using implicit source routing, and when a flow has not had an establishment packet sent for 5 or more seconds, the following packet originated along that flow includes the extra header to make it an establishment packet.

In addition to the four metrics described in Section 2.3.3, we also evaluated the implicit source routing mechanism using the *byte overhead* metric, which is defined as the total number of bytes of routing overhead, including the size of all routing overhead packets and the size of any routing headers added to data packets. The bytes are counted on transmission (whether original or forwarding) for each packet.

3.3. Results

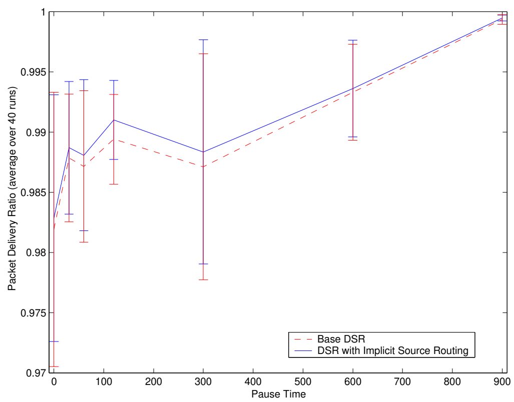
Packet delivery ratio is an important measure of the overall operation of any routing protocol. We measured the packet delivery ratio averaged over the 40 randomly generated scenarios for each pause time. These results are shown in Figure 3.1(a), with the error bars in the graph representing the 99% confidence interval of the average shown.

The use of implicit source routing marginally *improves* the packet delivery ratio. Two competing factors cause the difference in packet delivery ratio between the base version of DSR and DSR with implicit source routing. Since implicit source routing sends most packets without a source route, less-complete routing information is propagated through the network, reducing the success ratio of salvaging; across all 280 runs of each protocol, implicit source routing drops a packet due to the inability to find a route 70% more often (although both protocols drop very few packets). The other factor is congestion: since the packets transmitted by implicit source routing are smaller on average, each node is able to drain its network interface transmit queue more quickly, resulting in fewer drops from full interface queues; across a random sample of 140 runs of each protocol, implicit source routing drops 17% fewer packets as a result of full interface queues. Since drops resulting from inability to find a route are much less common, DSR with implicit source routing generally has slightly higher packet delivery ratio.

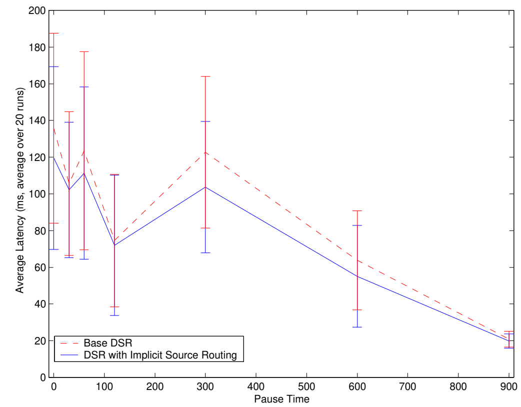
Another measure of the overall operation of the routing protocol is average latency. Our simulation results for average latency are shown in Figure 3.1(b). In general, DSR with implicit source routing has slightly *better* latency than does base DSR, due to the smaller average size of each packet without the source routing header present. The reduced packet size directly decreases the transmission time for the packet and also indirectly improves latency due to reduced contention for transmission bandwidth from other packets.

Finally, a third measure of the routing protocol's overall operation is the path optimality. As shown in Figure 3.1(c), however, the introduction of implicit source routing into DSR did not significantly alter DSR's path optimality.

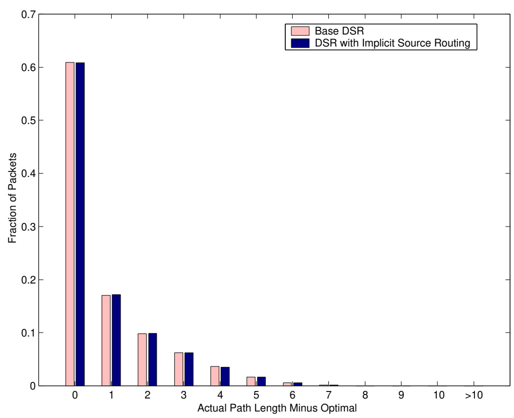
Beyond these three overall measures, the packet overhead and byte overhead provide an internal measure of the operation of the protocol. These metrics indirectly affect the three overall metrics discussed above and also contribute to other measures of the protocol such as CPU efficiency and battery power consumption. Figure 3.2(a) shows the packet overhead for base DSR and for DSR with implicit source routing, and Figure 3.2(b) shows this comparison for byte overhead.



(a) Effect of Implicit Source Routing on Packet Delivery Ratio



(b) Effect of Implicit Source Routing on Average Latency

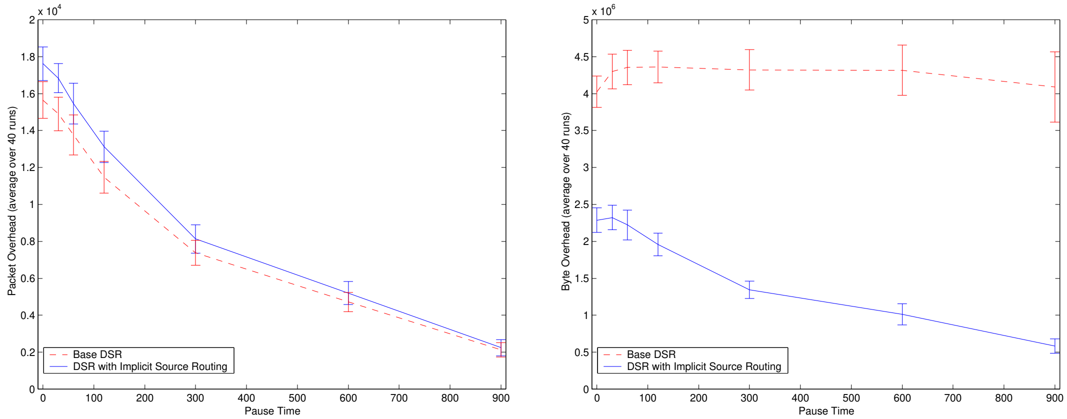


(c) Effect of Implicit Source Routing on Path Optimality

Figure 3.1: Effect of Implicit Source Routing on Performance

For packet overhead, we did not count flow establishment packets as overhead since they also contain data. In comparing DSR with and without implicit source routing, we found that implicit source routing incurs 12.3% more packet overhead. These overhead packets came from three sources: FLOW UNKNOWN errors, DEFAULT FLOW UNKNOWN errors, and additional Route Discoveries. The number of FLOW UNKNOWN errors and DEFAULT FLOW UNKNOWN errors, however, is quite small, since we do not model a limit on the Flow Table size, nor do we model nodes crashing and restarting. Most of the additional overhead packets thus are the result of additional Route Discoveries, which are required because fewer packets are sent with full source routes, lessening the ability of other nodes to cache overheard routes.

For counting byte overhead, we modeled packet sizes in our simulations according to the packet formats defined in our IETF Internet-Draft specifications for DSR [77, 90]. The byte overhead includes *all* bytes in overhead packets, *plus* the overhead such as any source routes and flow identifiers carried in data packets. In contrast to the packet overhead described above, the byte overhead with implicit source routing decreased substantially over the base version of DSR. For example, with



(a) Effect of Implicit Source Routing on Packet Overhead

(b) Effect of Implicit Source Routing on Byte Overhead

Figure 3.2: Effect of Implicit Source Routing on Overhead

continuous mobility of all nodes, the byte overhead for DSR decreased by 44% with implicit source routing, and in a stationary network, byte overhead decreased by 86%.

Furthermore, by avoiding the need to include a source route header in every data packet, the savings in byte overhead becomes proportional to the offered data packet load; in our simulations, each node originated only 4 packets per second in order to test the routing protocol's ability to find and maintain routes in a moving network, but many real applications would generate greater rates of data packets, further increasing the savings in byte overhead with implicit source routing.

3.4. Related Work

Before this work was published [75], the concept of routing flows using per-hop state had been included in Multi-Protocol Label Switching (MPLS) [166] and ATM [179]. However, unlike such protocols, our implicit source routing technique is designed for use in an ad hoc network, where nodes may move and the network topology may change often (or continuously) and nodes generally forward packets over the same wireless network interface on which they received them. In addition, in MPLS and ATM, the flow identifier changes at each hop. This change prevents the use of default flows as defined in implicit source routing.

On-demand routing based on per-hop state is also present in AODV [141]; however, nodes using AODV are unable to take advantage of multiple paths to the same destination and cannot choose which sequence of hops a packet will take, though it may be desirable to do so as different hops have different characteristics, such as security, cost, and available bandwidth. AODV also relies on the hard state of a sequence number at each node to ensure loop-freedom of its routing, while all of the state in DSR with implicit source routing is soft state. In addition, AODV cannot utilize unidirectional links in the network for communication between nodes, whereas DSR, with or without implicit source routing, can fully support unidirectional links.

Many additional optimizations have been proposed for various portions of DSR, such as core routing [173] and Location Aided Routing (LAR) [99]. All of these modifications can be used with

DSR with implicit source routing, although any optimizations that attempt to achieve better performance through improved route selection may conflict with optimal route selection for minimizing overhead.

3.5. Chapter Summary

The use of source routing in an ad hoc network has many advantages, yet these advantages come at the cost of increased packet header size and thus increased routing overhead bytes. In this chapter, we have presented the design and evaluation of *implicit source routing*, a technique that preserves the advantages of source routing while avoiding the associated per-packet overhead in most cases. In a manner in part similar to techniques used for MPLS [166] or ATM virtual circuits [179], the originator of a packet tags the packet a flow identifier implicitly indicating the sequence of hops through which the packet is to be forwarded on its way to its intended destination. All per-hop forwarding state is dynamically established when forwarding the first packet along this route and is maintained by each node along the route only as *soft state*; the soft state is automatically established as needed, and loss of any portion of this state has no effect on the correct operation of the protocol.

In addition, our implicit source routing mechanism includes support for DSR's *automatic route shortening* mechanism, allowing routes in use to be automatically shortened if one or more intermediate hops in the route become unnecessary, and DSR's *salvaging* mechanism, allowing packets to be forwarded along alternate routes if the original route for the packet encounters a broken link at some intermediate node. Implicit source routing also supports use of *default flows*, avoiding the need for a flow identifier header in the packet and thus avoiding *all* routing overhead in that packet.

We have evaluated this technique by extending the Dynamic Source Routing protocol (DSR) to include use of implicit source routing. Routing in DSR is based on source routing that is dynamically established on-demand when a sender needs a new route to some destination. The DSR protocol is simple and has been shown by a number of groups to perform well when compared to other protocols [26, 86]. Our implicit source routing mechanism fits naturally into the existing structure of the DSR protocol [77] and preserves the important fundamental properties of DSR's operation, including sender-selected routes, allowing the use of multiple routes to any destination, providing guarantees of loop-freedom even for packets sent with minimal overhead, routing based entirely on soft state, and the ability to use unidirectional links.

Although DSR makes extensive use of overheard routes for a number of important optimizations, applying implicit source routing to DSR improved both packet delivery ratio and average latency; although packet overhead increased slightly, byte overhead was reduced substantially. In particular, although packet overhead increased by about 12.3% with implicit source routing, byte overhead *decreased* by between 44 and 86%; on all other metrics evaluated, the performance of DSR either did not change significantly or actually improved somewhat, due to indirect effects of the reduced routing overhead.

Chapter 4

Exploiting MAC Layer Information in Higher Layer Protocols

Making use of information from one network protocol layer within other protocol layers has been used to improve performance in both wired and wireless networking (e.g., [24, 157]), and such *cross-layer optimization* techniques are regarded as one approach for best working within the constraints of the challenging wireless and mobile environment. However, in multihop wireless ad hoc networking, few specific cross-layer optimization techniques have been proposed other than those using *physical layer* information such as received signal strength.

In this chapter, we examine several cross-layer optimization techniques for exploiting information from the *Medium Access Control (MAC) layer* to improve service to higher layer protocols in the network. The MAC layer of a multiaccess network such as a wireless channel coordinates the transmissions of different nodes on the network, for example to avoid packet collisions when two or more nodes transmit at once [60]. As such, the MAC layer at a node must typically be able to determine the degree to which the wireless medium around that node is idle or busy. We describe in this chapter a number of general techniques for utilizing this MAC layer *utilization* information in a multihop wireless ad hoc network to improve performance at the *network* and *transport* layers, and we evaluate these techniques through detailed simulation of ad hoc networks.

Many different wireless MAC layers have been proposed and implemented, including ones based on methods such as random access, TDMA, and polling. In this chapter, we use the IEEE 802.11 Distributed Coordination Function (DCF) MAC protocol [84], since it has been adopted as a wireless LAN standard and is widely used in both traditional wireless systems and in multihop ad hoc networking research. Our techniques using MAC layer utilization information could also be applied easily with similar random access collision avoidance wireless MAC protocols such as MACA [94] and MACAW [16], and could be adopted with other types of MAC protocols as well.

The specific network and transport layer protocols we used in our study of exploiting MAC layer utilization information are the Dynamic Source Routing protocol (DSR) (Section 1.2.2) and the TCP transport protocol [150]. We make use of wireless medium utilization information from the MAC layer at a node to improve routing decisions in two areas: first, we modify Route Discovery to prevent the discovery of routes over which it is undesirable to carry additional traffic since the wireless medium over those hops is already quite busy, and second, we use this utilization information from the MAC layer to control the use of certain routing protocol optimizations such as packet salvaging. Finally, we also use MAC layer utilization information to influence the setting of the Explicit Congestion Notification (ECN) bits [155] in the IP header of packets carried through portions of the network where the wireless medium is particularly busy; this use of ECN allows higher layer protocols such as TCP to also make use of this MAC layer information.

4.1. MAC Layer Utilization Information

In this chapter, we focus specifically on cross-layer optimization techniques exploiting *utilization information* obtained from the MAC layer at a node, within higher layer protocols at that node. In this section, we describe in detail one method for obtaining this type of information. We also discuss a range of general possible uses of MAC layer utilization measurements in higher layer protocols within a node; Section 4.2 later describes three specific uses of such measurements within the context of the Dynamic Source Routing protocol (DSR) and TCP, and Section 4.3, evaluates these three techniques in detail through simulation.

4.1.1. Measuring MAC Layer Utilization

The average *MAC layer utilization* level at a node indicates the degree to which the wireless medium around that node is busy or idle. We define the average MAC layer utilization as measured by a node to be the fraction of time during which that node either

- has one or more packets to transmit in its transmission queue for that network interface, or
- if that node had attempted to transmit, it would not have been able to do so then, according to the rules of the MAC layer at that node.

Since the instantaneous MAC layer utilization at a node is either 0 or 1, we average this value over a period to obtain an average indication of the use of the wireless medium around that node. In our work, we average MAC layer utilization level at a node over a window size of 10 seconds.

The intuition behind this definition of MAC layer utilization is that the instantaneous value of this metric should be 0 only when the wireless medium around the node is available for the node to begin transmission of a *new* packet not already in that node's network interface transmission queue. Measuring this value requires the node to monitor the state of its own MAC layer. Although many current wireless network interface products such as commonly available IEEE 802.11 wireless LAN cards do not provide a MAC layer interface to support this monitoring by the operating system software in the node, it is supported by some interfaces such as the DARPA GloMo Radio API [15], and additional future wireless products may provide such an interface if it is proven useful.

As an example of measuring MAC layer utilization, we simulate it in this chapter based on a detailed model of the IEEE 802.11 DCF MAC protocol [84]. We consider instantaneous MAC layer utilization level at a node to be 1 at any time that the MAC layer at that node detects physical carrier to be present, and at any time that this node's MAC layer is deferring due to virtual carrier sensing, interframe spacing, or backoff. Instantaneous MAC layer utilization at the node is also 1 at any time that the node has at least one packet in transmission queue for its wireless network interface.

4.1.2. Uses within the Network Layer

Within the network layer, one use of measurements of the MAC layer utilization at a node is to allow the routing protocol to attempt to affect the routes chosen, such as to avoid choosing routes through portions of the network where the wireless medium is particularly busy. Routing protocols for ad hoc networks can be grouped into two types: *proactive* (or periodic) protocols, and *reactive* (or on-demand) protocols. Nodes using a proactive routing protocol exchange routing information with each other (e.g., periodically) such that each node attempts to always know a current route to all possible destinations (e.g., [91, 140]). In contract, nodes using a reactive routing protocol do not exchange routing information until necessary, and instead attempt to discover all routes on-demand by an active search within the network (e.g., [88, 141]). *Hybrid* routing protocols, that combine these two approaches, are also possible (e.g., [63]).

In a proactive routing protocol, nodes can affect the routes chosen by the protocol by using the local measurement of MAC layer utilization to alter the *metric* for certain routing table entries that it exchanges with other nodes. For example, in a distance vector routing protocol, the node could include an expression of its MAC layer utilization level in each of its own routing advertisements; neighbor nodes receiving such advertisements could use this value to treat the link from this node as having a metric that is a function of the advertised MAC layer utilization level, rather than as is common, treating each such link as having a metric of 1. For a link state routing protocol, a node could use its local measurement of MAC layer utilization similarly to increase the metric that it includes for its neighboring links in its own routing update packets to other nodes.

In a reactive routing protocol, nodes can affect the routes chose by the protocol through changes in the operation of the dynamic route discovery process. We describe this approach in the context of DSR in Section 4.2.1. In a hybrid routing protocol, a node may naturally use a combination of mechanisms using MAC layer utilization measurements, based on either the proactive or reactive portions of the hybrid protocol, to affect the routes chosen.

Another use of measurements of MAC layer utilization within the routing protocol at a node is to modify in general the behavior of the routing protocol itself, based on the level to which the wireless medium around the node is busy. For example, depending on the MAC layer utilization measured by a node, optional features or optimizations within the routing protocol can be enabled or disabled, if their effectiveness might depend on whether or not the wireless medium around the node is particularly busy. We describe a specific example of this type of optimization in Section 4.2.2 in the context of DSR.

Another example of such protocol modification would be an adaptive distance vector routing protocol, in which a node modifies the periodic transmission of its own routing advertisement packets. If the wireless medium around the node is particularly busy, the node could reduce the frequency of its own advertisements, and could reduce the number of routing table entries included in each advertisement to include only the most important or most recently changed entries. The ADV ad hoc network routing protocol [19] performs similar adaptive optimizations, but the adaptation in ADV is based on "trigger meter" values that use network layer information, not information directly from the MAC layer as we propose here.

As a final example in this section, the AODV routing protocol [141] could be modified such that a node does not attempt local route repair when the wireless medium around the node is particularly busy. If a node attempts and is successful at local repair, then the route to that destination will continue to pass through that node. If instead, in such cases, the node simply treated the link as broken as normal, the new route discovered for that destination could be made to route around that area of the network, when combined with modifications to Route Discovery similar to those we define in Section 4.2.1.

4.1.3. Uses within the Transport Layer

Within the transport layer, a number of different uses of MAC layer utilization measurements at a node are possible. Section 4.2.3 describes one approach in detail for TCP, based on setting the Explicit Congestion Notification (ECN) bits in a packet's IP header [155]; this same approach would also be applicable for any other transport layer that supported use of the ECN bits [46]. Below we suggest some other possible uses within the transport layer for MAC layer utilization measurements.

Beyond setting ECN bits to improve TCP performance, it may be possible to use information about the MAC layer utilization levels at nodes along a multihop ad hoc network route to allow

TCP to gain additional information about network conditions. Such information about the degree to which the wireless medium around nodes is busy might enable TCP to react better by helping to differentiate conditions of wireless packet loss, congestion packet loss, or simple wireless medium contention-based packet delay.

As a final example of use at the transport layer, the IETF Reliable Multicast Transport (RMT) Working Group, in attempting to standardize reliable multicast for the Internet, is considering solutions based on negative acknowledgements and on positive acknowledgements [85]. Each of these approaches represents a tradeoff of factors such as overhead and latency; with additional information such as MAC layer utilization measurements, it might be possible to adaptively balance between these two approaches.

4.1.4. Uses within Other Higher Layer Protocols

Above the transport layer, information on the MAC layer utilization at a node or along a path, as suggested in Section 4.1.3, can be used to adapt some traditional functions of of the presentation layer, such as data compression. If the MAC layer utilization level indicates that the wireless medium is particularly busy, a sending node could decide to compress the data before transmission. Such use of compression represents a tradeoff between the bandwidth used for transmission versus the CPU time consumed for compression and decompression and the latency in time taken for these functions. Based on the measured MAC layer utilization level, a sending node could more productively make such tradeoff decisions.

If an application programming interface (API) is available to pass the MAC layer utilization measurement values to user-level programs, these measurements could also, for example, be used to aid middleware application adaptation systems such as Odyssey [133] and Puppeteer [42].

4.2. Evaluation within DSR and TCP

This section describes the specific uses of MAC layer utilization measurements that we examined in our evaluation of these cross-layer optimization techniques within DSR and TCP. We modified the protocol behavior based on a combination of two metrics, the measured level of *MAC layer utilization* at a node and the *interface queue length* at that node; the interface queue length is the number of packets waiting buffered at that node for transmission over its wireless network interface. The first of these metrics provides the node with a view of the current condition of the shared wireless medium around the node; the second of these metrics indicates a prediction of the future load that this node will place on the wireless medium.

4.2.1. Modifications to DSR Route Discovery

In Route Discovery in DSR, a node searches for a route to the target destination node by performing a controlled flood of the network with ROUTE REQUEST packets. When one of the ROUTE REQUEST packets from this Route Discovery reaches the destination node or reaches another node with a route to the destination cached, this node returns a ROUTE REPLY packet to the originator of the Discovery.

Allowing this flood of ROUTE REQUEST packets from a Route Discovery to traverse an area of the network in which the wireless medium is already particularly busy creates several risks. First, the additional broadcast packets from the Route Discovery flood further increases the use of the wireless medium in those areas. Second, the route discovered by a Route Discovery is the sequence of hops through which the ROUTE REQUEST packet was forwarded that generated the ROUTE REPLY in response, and thus, any route discovered by forwarding a ROUTE REQUEST through an area of the network in which the wireless medium is already particularly busy can only result in a discovered route through this same area; such routes are less desirable than other routes. Finally, the additional traffic resulting from a new flow of data packets using a route through such an area can cause the wireless medium in this area to be used even more heavily, possibly leading to performance degradation for all users.

To alleviate these problems, we explored the effect of modifying DSR so that nodes do not process or forward a ROUTE REQUEST packet if the node determines that the wireless medium around itself is too busy; however, if the node is the target of the ROUTE REQUEST, it processes it and returns a ROUTE REPLY as usual.

This optimization is simple to implement, although in this form, it has two limitations. First, it may cause a node to be unable to discover a route to some destination, even when a route actually exists, if the only existing routes go through busy areas of the network. Second, by forcing the Route Discovery to route around busy areas, it may cause a node to discover a route that is longer than the minimum number of hops that could have been discovered; in saving overhead within busy areas of the network, this optimization may create additional overhead totaled across other areas of the network.

A modification to this optimization that could be made to address these limitations, is to add a flag to each ROUTE REQUEST, indicating whether or not to use this optimization. A node that has a packet to send to a destination would first check its Route Cache, and if it did not have a route, would initiate a Route Discovery with the flag off; that is, such that nodes in busy areas would not forward ROUTE REQUEST packets from that Route Discovery. If the source node does not receive a ROUTE REPLY from that Discovery, it would initiate another Discovery, this time turning the flag on, allowing all nodes to forward REQUESTS belonging to this Discovery. This modification is somewhat similar to an expanding ring search, although the search here expands into busy areas rather than simply into areas at a greater hop count from the source. In our simulation, we did not implement this modification to the Route Discovery optimization, since the performance of ordinary Discovery was sufficient.

4.2.2. Modifications to DSR Packet Salvaging

In DSR, packet salvaging is a mechanism used by an intermediate node to avoid dropping a packet when it detects that the next hop for the packet along its original route is broken. The intermediate node opportunistically checks its own Route Cache for a route to the packet's destination, contributing its own cache information to enhance the probability of successful delivery of the packet.

However, the route that this intermediate node may select from its own Route Cache for salvaging may not be a valid route to the destination, since the Cache is not actively maintained and some nodes may have moved since this route was cached. Packet salving usually is beneficial, though, because the nodes involved may not have moved extensively recently and because nodes update their Route Cache with routing information in forwarded and overheard packets, but in some cases, the extra overhead caused by forwarding the packet along the new route may not be worth the chance that the packet will be delivered correctly rather than just being dropped.

We explored the effect of modifying packet salvaging to not salvage a packet at an intermediate node (and to drop the packet instead when the next hop on the original route has broken) if the wireless medium around the node is particularly busy. This condition is an indication that attempting to salvage the packet may create more harm than good, since sending the packet along the new route will add more overhead to the wireless medium in the area.

Table 4.1: MAC Layer Utilization Levels for Triggering Optimizations

Optimization	MAC Layer Utilization	Interface Queue Length
Suppressing ROUTE REQUEST forwarding	15%	10
Suppressing salvaging	5%	20
Setting IP header ECN bits	1%	30

This modification to packet salvaging also addresses two related potential problems with salvaging in this situation. First, at a node where the wireless medium has been particularly busy, packets which could otherwise have been overheard may have a higher chance of loss, due to factors such as collision and increased noise floor. As a result, such a node will have been able to overhear less routing information from other packets and may thus have lower-quality routes in its Route Cache for any routes for which it is not directly involved in forwarding, making salvaging in this case even less desirable. Second, when a node attempts to transmit a packet to a next-hop node that is no longer a neighbor, an RTS packet is repeated several times (when using a MAC protocol like IEEE 802.11); each of RTS packets causes this node's neighbors to sense virtual carrier for the intended duration of the intended data packet. If several RTS attempts are made before determining that the link to the next hop has broken, possibly further increasing congestion around those nodes.

4.2.3. Use within TCP

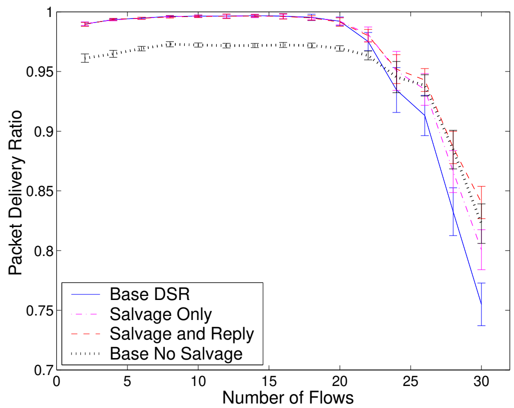
Ramakrishnan and Jain [156] proposed Explicit Congestion Notification (ECN) as a mechanism for signaling congestion, in packets traversing congested nodes or links. Floyd [51] presented a mechanism to use the ECN mechanism to improve the performance of TCP.

We made use of ECN as a mechanism for an intermediate node to signal to the TCP sender that the wireless medium around the node is particularly busy. Using ECN for TCP provides two benefits: first, it may prevent the loss of packets along that flow due to queue overflow, and second, it may allow better fairness for other flows also traversing this node.

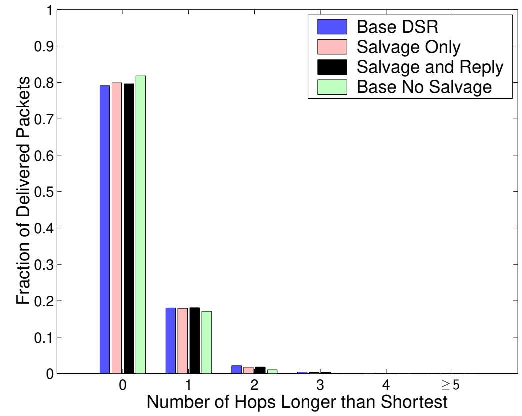
In typical use of ECN, routers use active queue management [21, 52] to set the Congestion Experienced (CE) codepoint [155] in a packet's IP header when the average queue length at that node exceeds some threshold. Instead, we set the CE codepoint in a packet based on our combined metrics of MAC layer utilization and queue length. When a TCP sender receives a packet with the CE codepoint set in its IP header, the TCP sender responds using its congestion control algorithm as it would to a packet drop [155]. Since our MAC layer utilization measurement represents an average of the recent level to which the wireless medium around the node is busy, the setting of the CE codepoint in a packet by an intermediate node indicates a sustained congestion condition needing action from the TCP sender.

4.3. Evaluation Methodology

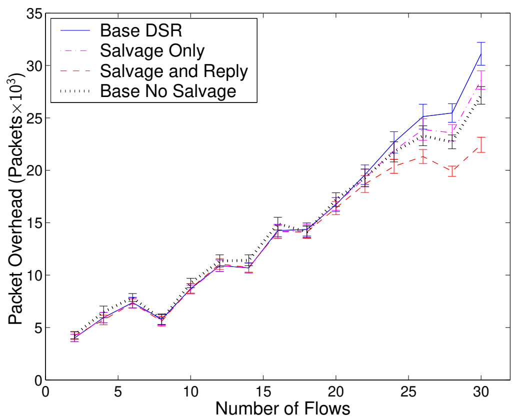
We based our evaluation of the use of MAC layer utilization measurements in DSR and TCP on the version of DSR that uses implicit source routing and flow state, as described in Section 1.2.2, since this version shows the best reported performance for the DSR protocol (Chapter 3). Using the *ns-2* network simulator (Section 2.3.1), we simulated this version of the DSR protocol, both with



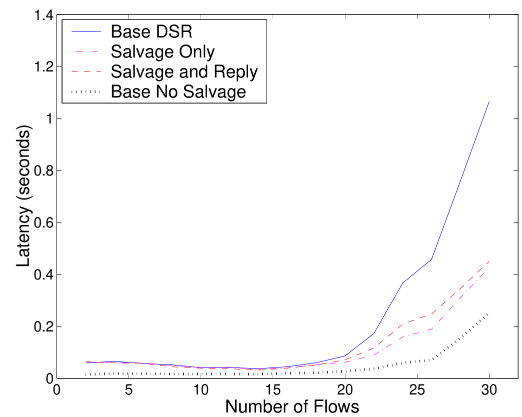
(a) Packet Delivery Ratio for 1500 m × 300 m



(b) Path Optimality for 1500 m × 300 m



(c) Packet Overhead for 1500 m × 300 m



(d) Average Latency for 1500 m × 300 m

Figure 4.1: Simulation results for CBR traffic in 1500 m × 300 m scenarios with 50 mobile nodes, with each source node sending 4 512-byte CBR packets per second; results are averaged over 40 simulation runs, with the error bars representing the 95% confidence interval of the mean.

and without the specific modifications for use of MAC layer utilization information within DSR and TCP described in Section 4.2. All behavior of TCP in our simulations was created by *ns* without modification, based on our setting of the ECN bits in the IP header of packets as appropriate.

Due to the varying affect that contention in the wireless medium and congestion has on each of our several optimizations, we chose different levels at which to enable each. Table 4.1 shows the measured MAC layer utilization and interface queue lengths at which we enabled each optimization. We chose these values by intuition and have not yet undertaken any attempt to tune them for performance.

We evaluated the performance of these modifications over a wide range of scenarios, with nodes moving according to the *random waypoint* mobility model (Section 2.5.1). We choose the same parameters for the random waypoint model as in Section 2.5.1.

The data traffic in our simulations was based both on Constant Bit Rate (CBR) sources and TCP sources. We performed three sets of experiments.

The first two sets of experiments used CBR traffic and evaluated the effect of our protocol modifications using MAC layer utilization measurements in DSR. One of these sets of experiments was performed using 50 mobile nodes in a simulation area of 1500 m $\times$ 300 m modeling 900 seconds of simulated time for each run, and the other set was performed using 100 mobile nodes in an area of 1000 m $\times$ 1000 m modeling 1000 seconds of simulated time for each run; in both of these sets of experiments, we simulated a number of CBR traffic sources, varying from 2 to 30 CBR sources per run, with each source sending 4 512-byte packets per second.

Our final set of simulation experiments evaluated the effect of our protocol modifications in DSR and TCP on a set of TCP flows; these experiments were performed using 100 mobile nodes in a simulation area of 1000 m $\times$ 1000 m modeling 1000 seconds of simulated time for each run. In these experiments, we simulated 20 TCP streams per run, with each TCP source sending data continuously during the execution of the simulation.

In the first two sets of experiments, we measured the four metrics described in Section 2.3.3. In the third set of experiments, we measured the goodput and fairness of the set of TCP connections. We define the goodput of each TCP stream here as the number of bytes of the TCP data stream correctly delivered to the receiver, such that that byte and all previous bytes of the stream were delivered with no missing TCP segments.

4.4. Results

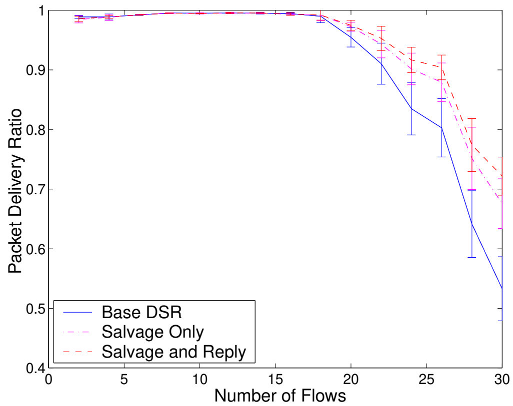
The results from the first two sets of simulation experiments described in Section 4.3 are shown in Figures 4.3 and 4.3. We defer the presentation of the results from our third set of experiments until Section 4.4.3, where we discuss those results.

Figure 4.3 shows the four metrics defined in Section 4.3 for simulation runs of 50 nodes in an area of 1500 m $\times$ 300 m, and Figure 4.3 shows the corresponding set of results for simulation runs of 100 nodes in an area of 1000 m $\times$ 1000 m. In these graphs, the error bars shown represent the 95% confidence interval of the mean. We discuss these results below in Sections 4.4.1 and 4.4.2.

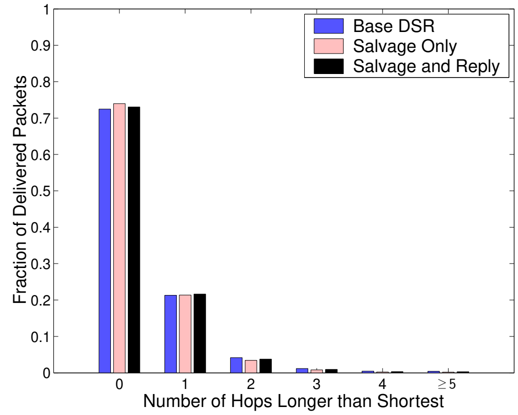
4.4.1. Suppressing Salvaging

When salvaging was disabled based on high MAC layer utilization levels, as in Section 4.2.2, performance was identical with lower load, but packet delivery ratio, overhead, and average latency all showed substantial improvements at higher load. For example, in the 1000 m $\times$ 1000 m scenarios, at the high load of 26 flows, representing a data rate of 426 kbps, the unoptimized version of DSR delivers just 80% of its packets, while the optimized version of DSR delivers almost 88% of its packets. At the same load, packet overhead decreased by over 25%, and average latency dropped by more than a factor of 4.

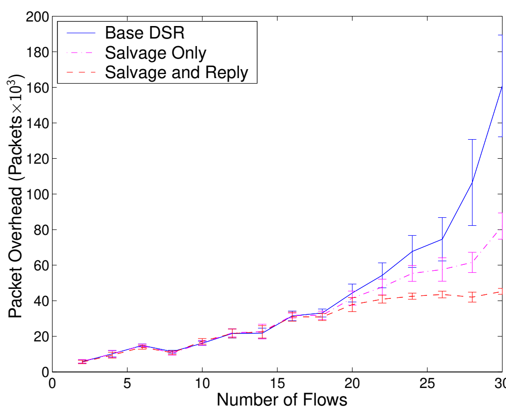
To evaluate the effectiveness in using MAC layer information in making decisions about whether or not to salvage, we also compared our scheme to a version of DSR that never salvages. We ran these simulations only for the 1500 m $\times$ 300 m scenarios. When compared to a version of DSR that never salvages, the salvaging optimization based on MAC layer information show significantly better performance at lower loads. For example, with 20 flows, representing a data rate of 327 kbps, the version of DSR using MAC layer information delivered 99.21% of offered packets, where on the same scenarios, the version of DSR that never salvaged delivered just 96.93% of packets. At the same load, packet overhead is also slightly lower without salvaging, due to the positive effects of spreading cache information through source routes. At higher loads, salvaging actually decreases packet delivery ratio relative to the base version of DSR; choosing whether or not to salvage based on MAC layer utilization level retains much of the benefit to packet delivery ratio of not salvaging when the utilization is low, without sacrificing the ability to salvage when congestion is not a problem.



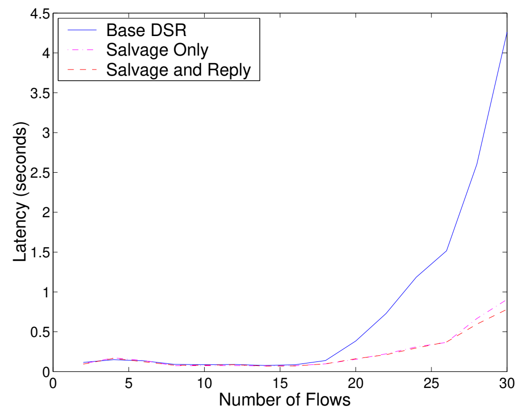
(a) Packet Delivery Ratio for 1000 m $\times$ 1000 m



(b) Path Optimality for 1000 m $\times$ 1000 m



(c) Packet Overhead for 1000 m $\times$ 1000 m



(d) Average Latency for 1000 m $\times$ 1000 m

Figure 4.2: Simulation results for CBR traffic in 1000 m $\times$ 1000 m scenarios with 100 mobile nodes, with each source node sending 4 512-byte CBR packets per second; results are averaged over 10 simulation runs, with the error bars representing the 95% confidence interval of the mean.

A possible improvement to this scheme would be to not forward salvaged packets at congested nodes; a node could examine the "salvage count" field in the DSR header of each packet that it forwards, and make forwarding decisions based on salvage count and local measured MAC layer utilization level. This technique would provide even more of the benefits of never salvaging, but only at nodes where not salvaging is beneficial. Preliminary results show that such an approach could split the difference between never salvaging and using MAC layer utilization information at higher levels of congestion, while maintaining the higher performance of using MAC layer utilization levels at lower congestion. This approach cannot fully achieve the benefits of not salvaging in congested areas because a congested node may have a neighbor that is not congested; if that neighbor salvages a packet and sends it to the node, the node would not forward it, so the initial transmission was wasted. It may also be possible to "push" MAC layer information one hop farther, allowing neighbors to see congestion levels of neighboring nodes, either by piggybacking the information on existing data and routing packets, or by including it as part of an RTS/CTS handshake, but such pushed information may be stale.

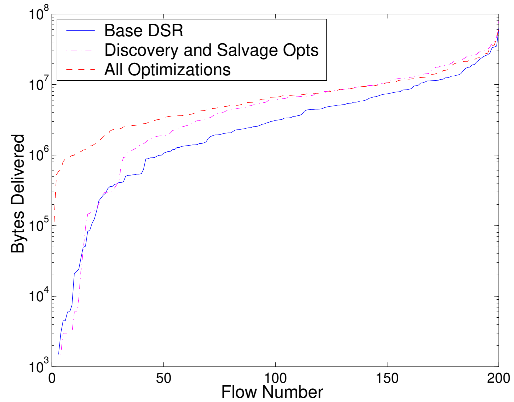


Figure 4.3: Number of bytes delivered per TCP flow in 1000 m × 1000 m scenarios with 100 mobile nodes and 20 TCP flows in each run; each TCP flow over 10 simulation runs is shown separately, sorted by the number of bytes of goodput delivered to the receiver for that flow.

The version of DSR that never salvages always has better latency and path optimality, since no packets are rerouted in-flight; however, at lower traffic loads, this is at the cost of some packets not being successfully delivered.

Using MAC layer utilization levels to influence salvaging decisions provides, to a large extent, the advantages of both choices.

4.4.2. Suppressing Route Discovery

When ROUTE REQUEST propagation was determined based on MAC layer utilization level, our simulations showed a slight but statistically significant increase in packet delivery ratio in the 1500 m × 300 m runs, as well as a more substantial improvement in packet overhead for both sets of scenarios. For example, in the 1500 m × 300 m scenarios, with an offered load of 26 flows, representing a data rate of 426 kbps, the packet delivery ratio with both salvaging and Route Discovery optimizations enabled was 94.29%, and was only 93.50% with just the salvaging optimizations; enabling Route Discovery optimizations also reduced overhead by 12%. At the same load in the 1000 m × 1000 m scenarios, enabling Route Discovery suppression based on MAC layer information increased packet delivery ratio from 87.91% to 90.41%, while decreasing overhead by 32%.

By using measured MAC layer utilization levels to avoid congested areas in discovered routes, DSR can more evenly spread the offered load across different forwarding paths in the network.

4.4.3. TCP Fairness

Figure 4.3 shows the results of our third set of experiments, described in Section 4.3, evaluating the effect on a set of TCP flows when using our protocol modifications using MAC layer utilization measurements. These experiments used all protocol modifications to DSR and TCP described in Section 4.2. This graph shows the number of bytes of goodput delivered for each TCP flow over 10 simulation runs with 20 TCP flows per run, or 200 total TCP flows; the y-axis scale on this graph is logarithmic, in order to show the detail in the curves plotted.

In these simulations, as described in Section 4.2.3, we caused each TCP sender to react using ECN when an area of the network through which that flow was routed experienced congestion in terms of high levels of usage of the wireless medium in that area or long queue length at an intermediate forwarding node on the route. In addition, since in an ad hoc network, routes can change frequently, to help ensure fairness, we also cause a TCP sender to begin slow-start as soon as that node receives a new ROUTE REPLY, indicating a change in the route for that TCP connection.

When ECN bits are set in congested areas of the network, flows traversing many hops, and other flows traversing few hops, are evenly penalized, improving TCP fairness. In our simulations, this ECN behavior substantially increased total throughput for more than half the total flows, relative to the results when this ECN modification is not used but all other protocol modifications are still present. Though setting ECN bits slightly decreases the overall throughput, more flows receive a reasonable level of service. This result is expected in any system designed to increase fairness: a multi-hop TCP flow will require more aggregate wireless bandwidth for the same amount of end-to-end delivered bandwidth, so increasing the throughput for connections traversing more hops will have an adverse effect on TCP connections traversing fewer hops.

4.5. A Quality-of-Service Demonstration

Ad hoc networks have the potential to provide effective communication services for groups of wireless mobile users, without requiring the aid of any centralized administration or fixed infrastructure such as base stations or access points. Certain applications of ad hoc networks require levels of service beyond best-effort packet delivery, such as the ability to support real-time multimedia streams such as live audio and video over the network. Examples of such applications include video conferencing, remote site monitoring, and unmanned vehicle operation. However, such support in an ad hoc network is particularly challenging due to the rapid changes in routing that may occur with node motion and due to occurrences of interference and congestion in the shared radio spectrum used by the nodes. Although a number of mechanisms for providing the necessary Quality of Service (QoS) support for such applications in ad hoc networks have been proposed, these mechanisms introduce complexity into the system and generally increase system overhead. In addition, published reports of testbed implementations or demonstrations of ad hoc networks [4, 114] do not provide mechanisms supporting these types of real-time multimedia streams.

In this section, I describe the design and demonstration of a set of three mechanisms we added to the *Dynamic Source Routing* protocol (DSR) for ad hoc networks (Section 1.2.2) to improve service by supporting live audio and video streams. DSR is a simple and efficient protocol for routing in ad hoc networks, which has been previously demonstrated through simulation and testbed implementation to perform well [26, 86, 113, 114]. Our goal in this work was to develop a set of lightweight extensions for DSR that perform well and that preserve the basic operation of the DSR protocol. These extensions can also be applied to other similar ad hoc network routing protocols such as AODV [141].

Our demonstration at the final DARPA Global Mobile Information Systems (GloMo) Principal Investigators Meeting in July 2000 showed smooth audio and clear video performance over a continuously moving multihop wireless ad hoc network, using off-the-shelf Microsoft Windows NetMeeting [125] audio and video software. Since our implementation and demonstration of these mechanisms at the GloMo PI meeting, some simulation results have been shown for these and simular mechanisms [57, 75], but this section presents the first and we believe currently only implementation of these mechanisms; this work thus represents the only experimental results showing the effectiveness of these mechanisms in a real mobile ad hoc network.

4.5.1. Preemptive Route Maintenance

The standard DSR Route Discovery and Route Maintenance procedures work very well for most types of data, but when a route in use breaks, some latency is introduced before the data can begin flowing again over a new route. Route Maintenance must attempt a number of retransmissions over the broken link before sending a ROUTE ERROR to the source of the data, and Route Discovery requires a network round-trip between the source node and the target node to discover a new route to the target. Although the total of this added latency when a route breaks is small (typically less than 100 ms total, with our hardware), its effect on real-time multimedia streams such as live audio and video is undesirable.

To avoid this latency, we used the measured signal-to-noise ratio (SNR) for received packets to detect when a route in use is likely to break soon. For the wireless LAN cards used by the nodes in our implementation (IEEE 802.11 Lucent WaveLAN), for each received packet, the card reports to the network interface device driver software the measured signal strength and noise values along with the contents of the packet; this is a common feature supported by most commercial wireless LAN cards and other radio devices. The term *preemptive Route Maintenance* refers to a number of possible approaches that make use of this SNR information to detect the likely imminent failure of a link, in order to use an alternate route if already known, or to be able to initiate a new Route Discovery in time to discover a new route before the old route actually breaks.

One approach to preemptive Route Maintenance is for each node to keep statistics of the SNR for recent packets it has received from each of its neighbors. When the node receives a new packet with an SNR below some threshold, it may choose to send a warning of this to the original sender of the packet. For example, if this packet and the last several packets received from this neighbor have all been below the threshold, it is likely that the link from that neighbor will soon break. To send the warning to the sender of the packet, this node could send a special type of ROUTE ERROR as a warning, indicating that the link may fail soon. A node receiving such a notification may continue to use the specified link, but if it is routing real-time multimedia packets over routes including that link, it should initiate a new Route Discovery to find a more suitable route soon.

A number of additional sources of SNR information are also available for use with preemptive Route Maintenance. For example, if a node makes use of passive acknowledgements for Route Maintenance, the SNR of a received passive acknowledgement from the next-hop node forwarding a packet transmitted by this node, can serve as an indication of the quality of the link from this node to that next-hop node. Furthermore, the measured SNR for any other packet overheard by this node (e.g., a broadcast packet or any packet received by this node while operating its network interface in "promiscuous" mode) may also provide an indication of the quality of the link between this node and the node transmitting that packet.

In some environments, it may be important to distinguish the meaning of these SNR measurements for the direction of transmission over the link. In particular, due to effects such as differing sources of wireless interference or possible differences in the radio and antenna hardware at both ends of a wireless link, wireless propagation may not work equally well in both directions between two nodes. For the SNR of a packet received by this node for forwarding along a route, the measured value directly indicates the quality of the link being used to forward this packet and future packets along the same route. For the SNR of a received passive acknowledgement, however, the measured value only directly indicates the quality of the link *from* this next-hop node *to* this node, for example as might be used in a reverse route used for end-to-end communication in the opposite direction between the same two end nodes (e.g., for the return of an end-to-end TCP acknowledgement packet). Similarly, the SNR value measured for any other packet overheard by this node only directly indicates the quality of the link *from* the node transmitting this packet *to* this node. In each of these cases, though, these SNR values also may be useful indirectly in helping to assess the quality of the link for future packets transmitted to that neighbor node by this node.

4.5.2. Using SNR to Limit Route Discovery

The ability of the network interface hardware to measure the signal-to-noise ratio (SNR) for each received packet, as described above in Section 4.5.1, can also be used to improve the behavior of the Route Discovery process for real-time multimedia streams such as live audio and video. Due to natural fluctuations in received signal strength from effects such as multipath propagation, some packets may be able to be received by nodes much further away from the transmitting node than is typical for packets transmitted by that node.

This phenomenon creates a potential problem for Route Discovery in that a ROUTE REQUEST packet may sometimes be received by a node that would not be able to receive most other packets transmitted by that same node. If this ROUTE REQUEST is processed normally by that node as part of this Route Discovery attempt, the discovered route may be very unreliable, since this node may not be able to receive most data packets sent along that route to it or through it as an intermediate node.

In addition, in the same way as preemptive Route Maintenance attempts to find alternate routes when a link in a route in use falls below the SNR threshold, Route Discovery should avoid finding such routes, if possible. That is, if a route is discovered by Route Discovery in which one or more links have a very low SNR, this may be an indication that such links in the route are likely to break soon; in this case, it may be much more efficient and cause much less disturbance to real-time multimedia streams over this route, for Route Discovery to be able to find a different route instead.

To address these problems, the processing of a Route Discovery when attempting to find a route for use by a multimedia stream could be modified to limit the spread of the ROUTE REQUESTs based on SNR. In this case, a node that receives a ROUTE REQUEST such that the SNR of this received packet falls below the threshold should not forward the REQUEST. Thus, all routes discovered will consist entirely of links with adequate SNR at the time that the route is discovered.

4.5.3. Per-Hop Flow State Maintenance

Audio and video applications tend to send very small packets to minimize the buffering latency at the transmitter. Unfortunately, this means that the source route normally present in the header of each DSR packet could represent considerable total overhead. In addition, we would like to be able to differentiate packets belonging to real-time multimedia flows and ordinary data packets.

We have designed a general mechanism called *flow state* to support these goals (Chapter 3) We provide a summary of the basic flow state mechanism here. With flow state, most DSR packets do not contain a source route header. Once a node sending a packet to some destination has discovered a source route to that destination, the node sends the packet as a normal DSR packet containing the full source route header. As the packet is forwarded to the destination based on this source route, the flow state mechanism allows each node forwarding the packet to remember the address of the next hop along this source route. Subsequent packets from this sender may then be forwarded along the same route to this destination with no source routing information present in packet header of any of these packets. The flow state established at each hop along the route is “soft state” and thus automatically expires when no longer needed.

Our current design of the flow state mechanism is more general than what we used in our demonstration, but for the use to which we put it here, the two versions are functionally the same. Each node maintains a Flow Table, which associates a (*source address*, *flow identifier*) pair with a full source route to the destination for that flow. To establish Flow Table entries, packets are sent with a source route as normal, except that a flow identifier is also included. A node receiving such a packet adds the new flow information to its Flow Table.

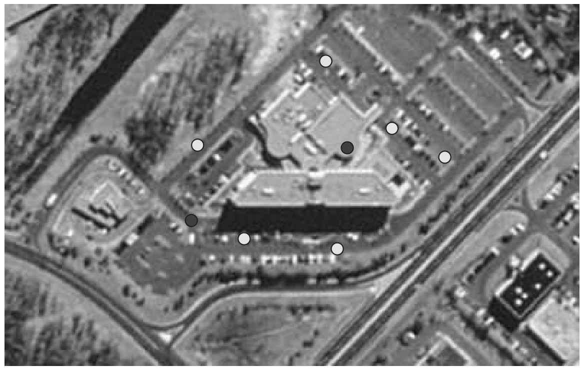


Figure 4.4: Location of nodes in the demonstration

To send a packet along that flow once all intermediate nodes have associated the flow identifier with the source route, the sending node replaces the destination address in the packet's standard IP header with a special encoding of the flow identifier for that flow. If a node receives a packet for forwarding without a source route in the packet's header, the node forwards the packet to the next hop indicated in this node's Flow Table entry for that flow identifier. If, however, the node has no corresponding Flow Table entry, it returns a special form of ROUTE ERROR packet to the source node; the source then reestablishes the flow state as when initially beginning to use that route. The flow state also allows a forwarding node to differentiate real-time multimedia packets from ordinary data packets, as this state can also be represented in the Flow Table entry.

4.5.4. Demonstration Design and Configuration

In our demonstration ad hoc network at the GloMo PI meeting, we used Microsoft Windows NetMeeting [125] to provide live audio and video between a stationary node and a moving car over multiple wireless hops. The demonstration was performed at the Sheraton hotel in Eatontown, New Jersey, a large 6-story hotel surrounded by parking lot on all sides; Figure 4.4 illustrates the configuration of the nodes in our ad hoc network. The stationary endpoint node was located in the hotel as shown on the right in Figure 4.4, with the moving car endpoint (shown on the left) driving in the hotel parking lot, continuously circling the hotel building (counterclockwise) without stopping. All routing in the ad hoc network was done using the Dynamic Source Routing protocol (DSR) integrated with the extensions for real-time multimedia support described above.

The ad hoc network allowed these two endpoint nodes to remain connected as the moving car endpoint circled the hotel. Each of the other six nodes in the ad hoc network shown in Figure 4.4 was implemented as a car parked in the hotel parking lot, with a FreeBSD Unix laptop in each car implementing DSR. Although these other nodes were stationary during the demonstration, the multihop wireless ad hoc network route between the stationary hotel endpoint node and the moving car endpoint changed rapidly throughout the demonstration. When the moving car endpoint node was near the hotel endpoint, the best route between these two nodes was a direct one-hop DSR route. As the moving car passed this point in circling the hotel, the route continued to change to incorporate more of the intermediate nodes, until the car reached the halfway point around the hotel. At this point, the best route between the moving car and the hotel endpoint switched to route around the hotel building in the opposite (shorter) direction, and this route then became progressively shorter as the moving car returned to near the hotel endpoint on its way again around the building. Keeping the intermediate nodes stationary also simplified the operation of the demonstration within the crowded hotel parking lot.

As mentioned, the two endpoint nodes in our demonstration (the moving car and the stationary node in the hotel) ran Microsoft Windows and NetMeeting. All application and operating system software on these two nodes were the standard, unmodified commercial version, and all communication was through standard IP packets. The DSR protocol and our implementation of it are thus entirely compatible with this commercial off-the-shelf software.

All wireless links in the ad hoc network were implemented using Lucent WaveLAN-II wireless LAN PCMCIA cards [93]. The radios support a maximum bitrate of 2 Mbps and are compatible with the IEEE 802.11 wireless LAN standard [84]; they operate in the 2.4 GHz ISM band with a transmitter power level of 15 dBm (30 mW). In setting up the network, we measured signal strength and packet transmission reliability between the stationary cars, and placed these nodes such that each was normally able to communicate only with its directly adjacent nodes around the hotel building

In order to provide DSR routing functionality at the two endpoint nodes, we used an additional laptop at each endpoint. At the moving car endpoint node, one laptop ran Microsoft Windows and NetMeeting, and the other ran FreeBSD Unix and DSR. The two laptops were connected by a short wired Ethernet segment, using static IP routing configuration on these two laptops: the Windows laptop's IP default router was the FreeBSD laptop, and the FreeBSD laptop was configured with a static route to the Windows laptop. The stationary endpoint in the hotel was configured in the same way, except that the connection between the two in this case used a 2 Mbps point-to-point wireless link based on Lucent WaveLAN-II hardware with Yagi directional antennas. This link was wireless rather than wired as in the moving car endpoint, in order to extend the signal outside the hotel into the parking lot where the ad hoc network operated; we used a different IEEE 802.11 frequency channel for this link than used in the ad hoc network nodes, so that this link was isolated to the two laptops implementing the stationary hotel endpoint.

4.5.5. Protocol Implementation

We implemented the DSR protocol extensions described above by modifying our existing DSR implementation in the FreeBSD Unix kernel. FreeBSD is a freely available open-source version of Unix based on the Berkeley 4.4BSD-Lite Release 2 Unix distribution. In total, we changed approximately 1600 lines of source code in our DSR kernel to support these changes.

For both preemptive Route Maintenance (Section 4.5.1) and using SNR in Route Discovery (Section 4.5.2), we needed a way to get the necessary SNR information out of the network interface device driver into the DSR routing code. When a packet is received, the device driver places the packet in a kernel memory buffer known as an *mbuf* [119]. Since the design of our extensions depend on the SNR of a received packet only in order to check if it is below threshold, we simply added a new flag to the mbuf header to indicate that the packet in this mbuf was received with low SNR. The use of this flag avoids major changes to the format of packets in mbufs and significantly reduces any compatibility concerns for this functionality with the rest of the protocol handling source code.

In implementing preemptive Route Maintenance, we took advantage of the node configuration used for our demonstration. Since we had only one moving node, we implemented preemptive Route Maintenance only for that node. When the SNR to the next hop for the moving node was fading, this node could then initiate a new Route Discovery itself, eliminating the need to actually send a special ROUTE ERROR packet when the SNR dropped below the threshold. In addition, in our implementation, the node reacted when just one packet fell below the SNR threshold. Though this approach may result in a number of unnecessary Route Discoveries, it reacts quickly to degrading link quality. In addition, it did not require an additional data structure to keep track of how many recent packets had signal strength below the threshold.

In implementing the flow state extensions to DSR (Section 4.5.3), we added a Flow Table to each node to track all of the flows originating at or being forwarded by that node. When a node forwards a packet using a source route, it checks to see if the packet also contains an option specifying a flow identifier for this flow. If so, the node adds the route to its Flow Table. At the sender, we added an application program that allowed real-time multimedia flows to be identified, using a flow specification similar to RSVP [22]. An entry was added to the Flow Table for each flow identified in this way. When a sending node begins to use a new route for some flow in its Flow Table (e.g., when first using the flow or after the route in use is changed due to Route Maintenance), for the next ten packets that the node sends along that flow, the node includes a normal full DSR source route header and the new flow identifier in each packet; all other packets were sent using only a flow identifier. Repeating the full source route on each of these initial packets helps ensure that all of the nodes along the route have received the new routing information and associated it with the new flow identifier. If a node receives a packet for forwarding without a source route in the packet, but this node has no corresponding Flow Table entry (e.g., because it did not receive the earlier packet establishing that flow), the node returns a special ROUTE ERROR to the source, which then reestablishes the flow state, as described in Section 4.5.3.

To send a packet using a flow identifier, we used an encoding of the flow identifier in the packet header that allowed such packets to carry no per-packet overhead for DSR. Each flow identifier is 16 bits long. In the packet header, we represented the flow identifier in the packet's IP Destination Address field as an address of the form 127.0.xxx.yyy, where xxx.yyy is the flow identifier. Such addresses are normally reserved for the loopback network interface [159] and thus do not otherwise appear in the Destination Address field of a packet sent over the network. Since each node in our ad hoc network supported DSR and flow state, this address could in our implementation be recognized as a special case. Alternatively, the flow identifier could be represented in a special small header in the packet, but in our implementation, we opted for the representation in the Destination Address to further reduce overhead for carrying the audio and video streams.

Finally, in this implementation, we modified the FreeBSD WaveLAN device driver and our DSR code to allow DSR Route Maintenance to utilize the link-layer acknowledgements built into the IEEE 802.11 MAC protocol. Specifically, after completing transmission (and any link-layer retransmission attempts) for a packet, the device driver calls a new procedure within Route Maintenance to indicate the success (802.11 acknowledgement received) or failure (no acknowledgement received) for that packet transmission; if a transmission failure is indicated, DSR's Route Maintenance reacts by sending a ROUTE ERROR as appropriate. This ability to use link-layer feedback is a part of DSR's design [87] but to our knowledge has not been implemented before on 802.11 for DSR or other routing protocols for ad hoc networks. Since this acknowledgement is already required for all IEEE 802.11 transmissions, DSR is then able to perform Route Maintenance with no extra overhead.

4.5.6. Demonstration Results

We set up the ad hoc network for the demonstration and tested it over one day and demonstrated it for different groups of people at the meeting frequently over the following two days. In each run of the demonstration, we operated the ad hoc network continuously, with most runs lasting several hours or more; during each run, the moving car continued to drive around the hotel building, with brief breaks only to change drivers of the car, without resetting the laptops or restarting the network. The routing between the stationary endpoint node in the hotel and the opposite endpoint node in the moving car thus continued to change frequently during the operation of the demonstration. In all runs of the demonstration, all aspects of the protocols and extensions for live audio and video support worked extremely well in almost all respects.

Observers of the demonstration in the hotel could converse with the driver of the car and could see the view from a camera in the car pointed out the car's front windshield. All audio and video to support this was provided through standard Microsoft Windows NetMeeting being transmitted in IP packets over the ad hoc network. A microphone, audio headset, and video camera were located in the moving car, and a microphone and speakers were located in the hotel; we did not provide video from the hotel to the car since the driver of the car could not watch the video while driving. The video camera used was a Logitech QuickCam Pro camera with a USB computer interface.

In addition to the NetMeeting audio and video data traffic on the ad hoc network, we generated additional background traffic by using one of the stationary cars to "ping" each of the other cars; the source node of the ping was changed every 5–10 minutes, and within this time, each other car was used as a destination of the ping, dividing the time approximately equally between the other cars for this source node.

In most operation of the demonstration, the routes over the ad hoc network ranged up to 3 hops in length. Until the moving car reached the halfway point around the hotel, DSR generally discovered the route of 1, 2, or 3 hops around the "top" side of the hotel, as shown in Figure 4.4; as the moving car continued around the hotel, DSR generally discovered the route of 3, 2, or 1 hops around the "bottom" side of the hotel. At one point during the demonstration, the laptop in one stationary car was shut down and not restarted, forcing DSR to discover routes up to 5 hops in length in order to route all the way around the hotel building past this disabled intermediate node.

Throughout all runs of the demonstration with our multimedia extensions enabled in DSR, the audio quality received through the speaker was excellent, and sounded roughly the same as for a NetMeeting connection over a standard 56 kbps modem not using DSR. The video quality displayed was also very good, although at times not quite of the same quality as the audio. These observations of the audio and the video quality in general seemed not to be affected by the frequent routing changes in the multihop wireless ad hoc network as the moving car endpoint node circled the hotel building.

The one factor that did seem to affect the perceived video quality while operating with our multimedia extensions enabled was the behavior of the video camera and NetMeeting as the moving car node turned a corner as it drove around the building. At such times, the rate of change in the video scene captured by the camera that must be encoded by NetMeeting was quite large relative to the behavior when simply driving roughly straight forward. The data rate required by the video during these times thus increased [125]; since NetMeeting gives priority for use of a fixed portion of the available bandwidth for audio data, using only what bandwidth is left available for video, the video image degraded during these times, whereas the audio seemed unaffected.

When we disabled our multimedia extensions in testing during the demonstration, however, the audio and video qualities both suffered significantly. The audio appeared to drop or substantially delay some packets, creating gaps of silence in the audio played through the speakers. During these tests, the video image on the screen also became "blocky" and inconsistent. This is due to the video encoding used by NetMeeting: NetMeeting transmits a full video frame every 15 seconds, but otherwise sends only deltas between these full frames [125]. If a packet is dropped leading to the loss of a full frame, the deltas until the next full frame could not be displayed correctly. After this time, the video image on the screen returned to normal, but after a short while, the degradation would reoccur. Once we re-enabled the multimedia extensions in DSR, both audio and video quality quickly returned to normal and maintained their performance.

4.5.7. Related Work

After our design and implementation of this demonstration, Goff et al independently developed a similar form of preemptive Route Maintenance and of using SNR in Route Discovery [57]. However, they have not implemented their techniques in any real ad hoc network, and instead evaluated them only through simulation. In addition, they based their techniques only on received signal strength, not on signal-to-noise ratio (SNR).

For preemptive Route Maintenance, they used a sequence of explicit request/response packets (which they refer to as “ping” and “pong” packets) to probe the quality of a link after receiving a packet below the signal strength threshold. Although this link probing can avoid some cases in which the link only falls below the threshold for a short time (e.g., due to transient channel fading), it also increases network overhead at the point of detecting the impending link failure, and adds latency before a new Route Discovery can then initiated if needed. We believe the best approach may be an adaptive probing mechanism that varies the use of probing based on channel measurements.

Also since our demonstration, we have studied the design and use of per-hop flow state maintenance in more detail (Chapter 3) but like Goff et al’s work, we evaluated it in that work only through simulation, not through implementation in a real ad hoc network. In addition, our simulation work there studied mainly only the use of this technique for reducing routing overhead in packet headers, not specificly for QoS support.

Many researchers have proposed schemes for QoS on IP networks, including Integrated Services [20], RSVP [22], RSVP over LSP [7], Multi-Protocol Label Switching [165], and Differentiated Services [132]. Some QoS routing protocols also have been designed for ad hoc networks [30, 32, 71, 83, 105, 108, 175, 188]. These protocols use relatively heavyweight mechanisms to achieve general QoS, and tend to somewhat increase the amount of routing traffic and overhead. In this work, we instead use lightweight mechanisms based on slight modifications to the existing behavior of DSR to achieve our more specific goals.

4.5.8. Demo Summary

This section has described the design and demonstration of a set of routing protocol mechanisms that substantially improve the transmission of real-time multimedia streams over a multihop wireless mobile ad hoc network. These mechanisms were implemented in the Dynamic Source Routing protocol (DSR) (Section 1.2.2) and could also be applied to other on-demand ad hoc network routing protocols such as AODV [141]. We described the implementation of these mechanisms and their public demonstration carrying live audio and video streams over a DSR ad hoc network at the final DARPA Global Mobile Information Systems (GloMo) Principal Investigators Meeting in July 2000. The demonstration consisted of an ad hoc network of 8 nodes, with one node in a moving car driving continuously for periods of several hours or more around the hotel building where the demonstration was held; the endpoint node in the moving car operated a Microsoft Windows NetMeeting session over the multihop ad hoc network to another endpoint node located in the hotel, showing the view from a video camera pointed out the front window of the car, and allowing observers in the hotel to converse interactively with the driver of the car.

Although a number of general mechanisms for providing Quality of Service (QoS) support in ad hoc networks have been proposed, these mechanisms introduce complexity into the system and generally increase system overhead. In addition, published reports of testbed implementations or demonstrations of ad hoc networks [4, 114] do not support these QoS mechanisms. Our goal in this work was to develop a set of lightweight extensions for DSR that perform well and that preserve the basic operation of the DSR protocol, and to gain actual experience with the effect of these mechanisms in a real ad hoc network implementation. The resulting audio and video performance were both very good to excellent, limited mainly by the performance of the video camera and NetMeeting rather than by the frequent routing changes taking place while the moving car continued to drive around the hotel building. In addition, all application and operating system software on the two endpoint nodes in our demonstration were the standard, unmodified commercial version, and all communication was through standard IP packets, thus demonstrating that the DSR protocol and our implementation of it are entirely compatible with this commercial off-the-shelf software.

4.6. Chapter Summary

In this chapter, we have explored a type of cross-layer optimization in multihop wireless ad hoc networks. Whereas most previous proposals for such optimizations have been based on information from the physical layer of the network, in this chapter, we examined the use of MAC layer information in our optimizations. By monitoring the operation of the MAC layer on its own wireless network interface, a node can establish an approximation of the degree to which the wireless medium in its area is busy. This measurement reflects not only the behavior of the node itself, but also the behavior of other nodes around it sharing the same wireless medium.

We suggested a number of uses of such measurements of MAC layer utilization in an ad hoc network, in the network, transport, and higher layers, and we simulated a set of such uses in the Dynamic Source Routing protocol (DSR) and TCP. Our simulations demonstrated substantial improvement to DSR and TCP in the areas of scalability, packet delivery, overhead, and fairness resulting from this use of MAC layer information. Although we applied our changes to some areas of DSR to quantitatively demonstrate the usefulness of these optimizations, similar techniques could be applied to other routing protocols and a number of other optimizations are possible as well. For example, a node using a distance vector routing protocol such as DSDV [140] or ADV [19] could increase the time between advertisements during periods in which the wireless medium around the node is particularly busy, and a node using AODV [141] could choose to not attempt local repair during periods such as this.

Many choices are made at the MAC and physical layers, such as specific contention and priority schemes, multiple data rates, multiuser detection, and directional antennas. Generally, network interfaces present an Ethernet-like interface to the higher layers; however, in this chapter, I show that higher layers can make effective use of information available at lower layers. Future work includes a more detailed analysis of TCP performance, examining how other lower-layer information can be used in the routing and transport layer, and whether or not a small set of metrics can provide sufficient information to upper layer protocols. If such a set of metrics could be isolated, upper layer protocols could gain much of the performance advantage of using lower layer information, while maintaining the abstraction benefits of layer separation.

Part II

Improving Service in Untrusted Environments

Chapter 5

Security in Ad Hoc Networks

5.1. Security

Security represents an important part of any routing protocol, and is especially important in wireless ad hoc network routing, due to the inherent increased vulnerability of a wireless channel. In a wired network, it is possible to secure a piece of wire by enclosing it in a pressurized pipe and detecting the resultant change in pressure when the pipe is cut [48]. In addition, mechanisms such as checking the Time-to-Live (TTL) field in the IP header may allow an endpoint to verify that a packet came directly from a neighboring router. In a wireless network, there are no such physical restrictions on which nodes can directly communicate with some destination.

Security is generally designed around three properties: *integrity*, *authentication*, and *non-repudiation*. Each of these properties is stronger than the previous: message integrity requires only that the message was the same when transmitted and received. With authentication, a node can determine who sent the message, and that the message has not been modified. Finally, non-repudiation allows a receiver to store data, and later prove to a court of law that the sender really did send the corresponding data.

Currently, mechanisms for assuring *non-repudiation* are quite computationally expensive, especially on resource-constrained nodes [10, 148]. For example, Brown et al analyze the computation time of digital signature algorithms on various platforms [27]: on a Palm Pilot or RIM pager, 163-bit Elliptic Curve Cryptography (ECC) [120] signature algorithms require 1.0–2.2 seconds of computation for one signature generation and 1.8–5.4 seconds for verification; a 512-bit RSA [163] signature requires 2.4–5.8 seconds of computation for generation and 0.1–0.6 seconds for verification, depending on the public exponent. Some other approaches provide efficient on-line signature generation but assume off-line preparation [49, 171, 172]. More recent signature algorithms provide short or efficient signatures, but their verification is slower than verification in RSA [40, 72, 151, 152]. Even on faster devices, link-speed signature generation and verification is not generally practical, especially when using high-speed links. In particular, the use of asymmetric primitives in routing protocols enables a number of denial-of-service attacks. For example, an attacker can flood a victim with bogus signatures at a rate faster than the victim could possibly verify them; this attack could prevent the verification of legitimate routing messages.

Because asymmetric primitives are so inefficient, we base the security of our secure routing protocols on efficient symmetric authentication primitives. In this chapter, we review those previously discovered symmetric primitives that we use in our protocols. In Chapter 8, we present the primitives that we developed for more efficient secure routing protocols.

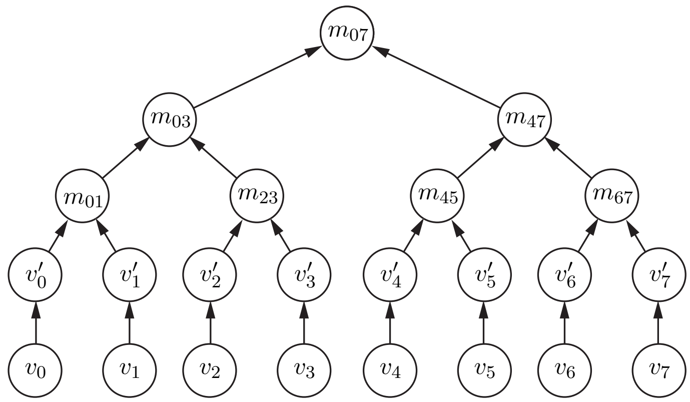


Figure 5.1: Tree authenticated values

5.1.1. Hash Functions

A hash function H is a "one-way" function, in that it is easy to compute but computationally infeisable to invert. Practically, if a hash function has ρ bits of output, for a given y, finding x such that $H(x) = y$ should require $2^{\rho-1}$ effort on average. Hash functions are also expected to be collision-resistant; that is, finding x and x' such that $H(x) = H(x')$ should require $2^{\frac{\rho}{2}}$ effort on average. Numerous such functions, such as MD5 [161] and SHA-1 [131], have been proposed, and are typically quite simple to compute.

5.1.2. Hash Trees

In this section, we review the efficient hash tree authentication mechanism. Merkle first presented this mechanism, also known as Merkle hash tree [122]. To authenticate values $v_0, v_1, \ldots, v_{w-1}$ we place these values at the leaf nodes of a binary tree. (For simplicity we assume a balanced binary tree, so w is a power of two.) We first blind all the values with a one-way hash function H to prevent disclosing neighboring values in the authentication information (as we describe below), so $v'_i = H[v_i]$. We then use the Merkle hash tree construction [122] to commit to the values $v'_0, \ldots, v'_{w-1}$. Each internal node of the binary tree is derived from its two child nodes. Consider the derivation of the parent node m_p from the left and right child nodes m_l and m_r: $m_p = H[m_l \mid\mid m_r]$. We compute the levels of the tree recursively from the leaf nodes to the root node. Figure 5.1 shows this construction over the eight values $v_0, v_1, \ldots, v_7$, e.g., $m_{01} = H(v'_0 \mid\mid v'_1)$, $m_{03} = H[m_{01} \mid\mid m_{23}]$, etc.

The root value of the tree is used to commit to the entire tree, and in conjunction with additional information can be used to authenticate any leaf value. To authenticate a value v_i the sender discloses i, v_i, and all the nodes necessary to verify the path up to the root. For example, if a sender wants to authenticate key v_2 in Figure 5.1, it includes the values v'_3, m_{01}, m_{47} in the packet. A receiver with an authentic root value m_{07} can then verify that

$$H\left[H\left[\ m_{01} \mid\mid H\left[\ H[\ v_2\ ] \mid\mid v'_3\ \right]\right]\ \mid\mid m_{47} \right]$$

equals the stored m_{07}. If the verification is successful, the receiver knows that v_2 is authentic.

The extra $v'_0, v'_1, \ldots, v'_7$ in Figure 5.1 are added to the tree to avoid disclosing (in this example) the value v_3 for the authentication of v_2.

5.1.3. One-Way Hash Chains

One-way hash chains are a widely used cryptographic primitive. One of the first uses of one-way chains was for one-time passwords by Lamport [103]. Haller later used the same approach for the S/KEY one-time password system [64]. These chains are also used in efficient one-time signature algorithms [49, 123, 124, 164], (micro-)payment mechanisms [3, 66, 138, 162], server-supported non-repudiation [6], conditional anonymity [67], and to authenticate link-state routing updates [33, 65, 191]. Coppersmith and Jakobsson present efficient mechanisms for storing and generating values of hash chains [38].

We create a one-way chain by selecting the final value v_n at random, and repeatedly apply a one-way hash function H, such that $v_i = H[v_{i+1}]$. (In our description, we discuss the one-way chain from the viewpoint of usage: so the first value of the chain is the last value generated, and the initially randomly chosen value is the last value of the chain.) The first value (last value generated) is called the *anchor*; generally, an authentic anchor is published to allow verification of hash chain elements. One-way chains have two main properties (assuming H is a cryptographically secure one-way hash function):

- Anybody can authenticate that a value v_j really belongs to the one-way chain, by using an earlier value v_i of the chain by checking that $H^{j-i}(v_j)$ equals v_i.
- Given the latest released value v_i of a one-way chain, an adversary cannot find a later value v_j such that $H^{j-i}(v_j)$ equals v_i. Even when value v_{i+1} is released, a *second pre-image collision resistant hash function* prevents an adversary from finding v'_{i+1} different from v_{i+1} such that $H[v'_{i+1}]$ equals v_i.

These two properties result in authentication of one-way chain values: if the current value v_i belongs to the one-way chain, and we see another value v_j with the property that $H^{j-i}(v_j)$ equals v_i, then v_j also originates from the same chain and was released by the creator of the chain.

5.1.4. The TESLA Broadcast Authentication Protocol

TESLA [145, 147] provides efficient broadcast authentication using hash chains and loose time synchronization. To provide broadcast authentication, some asymmetry must be created: that is, the sender must be able to authenticate packets and the receiver must be able to verify this authentication, but the receiver must not be able to authenticate packets as the sender. Unlike traditional digital signatures such as RSA [163], which rely on computationally expensive one-way trapdoor functions to create asymmetry, TESLA creates asymmetry using loose time synchronization and efficient *symmetric* primitives. In particular, TESLA only requires one Message Authentication Code in each packet.

Each sender using TESLA for authentication randomly generates a one-way hash chain, as described in Chapter 5.1.1. Each element in this chain is used as a key, so we refer to this chain as a TESLA one-way key chain, and refer to the i-th element e_i as K_i. The sender also determines a schedule for the release of keys in this chain; that is, the time t_i when each key K_i will be published. For example, a simple key disclosure schedule may be $t_i = t_0 + i \cdot t$. The sender then securely distributes the first element of its key chain K_1, together with the values of t_0 and t.

Central to TESLA's security is a receiver's ability to determine which keys a sender may have already published. A receiver using TESLA relies on loose time synchronization, together with a known maximum time synchronization error between any two nodes Δ, to determine the maximum time at the sender. If a sender discloses its first key K_0 at time T_0 (on its clock) and discloses an additional key each time interval, then the receiver can determine, based on its clock, Δ, and T_0, whether or not some key K_i has yet been disclosed.

To send a packet, the sender first estimates a pessimistic upper bound on the end-to-end network delay. The sender then picks a key K_i from its one-way key chain which the receiver will believe is still secret at the time the receiver is expected to receive the packet. For example, if the sender believes the end-to-end delay would at worst be τ, it could choose a key K_i that would not be disclosed until a time at least $\tau + 2\Delta$ in the future. The sender adds a message authentication code (MAC), computed using key K_i, to the packet, and sends it to the receiver.

The value 2Δ is added to τ (rather than Δ), since the receiver's clock may be ahead of the sender's clock by Δ, so at time t_s at the sender, it is $t_s + \Delta$ at the receiver. When the packet reaches the receiver, it could be as late as $t_s + \tau + \Delta$ at the receiver, and the receiver will discard the packet if the key *might* have been released; since the sender's clock may be Δ ahead of the receiver, the receiver will reject the packet if the key was to be released at time $t_s + \tau + 2\Delta$ or earlier. Even if the end-to-end delay is larger than τ, TESLA is still secure, but some receivers will need to discard the packet because the sender may have already disclosed the key, as we describe below.

When a receiver receives a packet authenticated with TESLA, it first verifies that the key K_i used to authenticate that packet is still secret. For example, given the packet arrival time t' and the maximum time synchronization error Δ, the receiver checks that the time elapsed between time T_0 and $t' - \Delta$ does not exceed i time intervals. If the check fails, the sender may have already published K_i and an attacker may have forged the packet contents; the receiver thus discards the packet. However, if the check is successful, the receiver buffers the packet and waits for the sender to publish key K_i. When the receiver receives K_i, it first authenticates K_i, and then authenticates stored packets authenticated with a key K_j, where $j \leq i$.

5.1.5. Hash to Obtain Random Subset (HORS)

Reyzin and Reyzin proposed the HORS broadcast authentication scheme [160]. To generate a key for authentication, the node randomly selects t values $b_1, b_2, \ldots, b_t$. The node then blinds these values by hashing them: $b_i' = H(b_i)$. These blinded values are published, and form a "public key" with which authenticators can be verified.

To authenticate a value v, the node performs a hash function on v which chooses a subset k of the t values it originally chose. Reyzin and Reyzin discuss in greater detail properties that are required of this hash function. The node receiving this authentication verifies that the correct subset of values was disclosed by computing the hash function and verifying that the

If each set of t values is used for r authentications, and each authentication includes k values, then the scheme has $k(\ln t - \ln r - \ln k)$ bits of security in the worst case. As a result, it is necessary to restrict any given set of t values to authenticate a fixed number of messages.

To provide a larger number of authentications when a single public key is provided, each value b_i in HORS can be replaced by a hash chain $b_{i,0}, b_{i,1}, \ldots, b_{i,n}$. The anchor of each hash chain $b_{i,n}$ is published in the same way that the blinded values were published in the unchained version. During the first time interval, the values $b_{1,n-1}, b_{2,n-1}, \ldots, b_{t,n-1}$ are used in place of $b_1, b_2, \ldots, b_t$. During the second time interval, the values used are $b_{1,n-2}, b_{2,n-2}, \ldots, b_{t,n-2}$. In general, the ith value during the jth time slot is $b_{i,n-j}$.

In order to use this chained version of HORS, nodes need to be time synchronized with a maxmimum error of Δ, and must estimate a network propagation time τ. (When the network propagation requires more than τ, security is not compromised, but some authentic packets may be rejected as forged.) Nodes also need to choose a network-wide schedule for the use of these values. For example, one schedule would be to use values from time interval j from time $T_0 + (j-1)I$ until time $T_0 + jI$. A receiver then rejects values from time interval j after time $T_0 + jI + \tau + \Delta$, under the assumption that even if the sender's clock was Δ behind the receiver's clock, and even if the packet took τ to traverse the network, a packet sent at time $T_0 + jI$ on the sender's clock will reach all receiveres by $T_0 + jI + \tau + \Delta$.

To obtain the same level of security as in unchained HORS, at most r authentications may be performed within a time of $I + \tau + \Delta$. This is because values disclosed during time interval $j + 1$ can be used to generate values valid during time interval j.

5.1.6. Amortized Authentication

It is possible to amortize the cost of authenticating messages by computing a Merkle hash tree (Section 5.1.2) over the messages to be authenticated, and authenticating only the root. In this construction, the messages are placed at the leaves of a binary tree, and interior nodes are formed by hashing the two children of that node together. If the root of this tree is authenticated, then each message can be authenticated by including the authentication of the root, combined with the path in the Merkle tree from the root to the leaf.

5.2. Ad Hoc Network Routing Security

In this section, we define a taxonomy of types of attackers and discuss specific attacks against ad hoc network routing. This approach allows us to categorize the security of an ad hoc network routing protocol based on the strongest attacker it withstands.

5.2.1. Attacker Model

We consider two main attacker classes, *passive* and *active*. The passive attacker does not send messages; it only eavesdrops on the network. Passive attackers are mainly threats against the privacy or anonymity of communication, rather than against the functioning of the network or its routing protocol, and thus we do not discuss them further here.

An active attacker injects packets into the network and generally also eavesdrops. We characterize the attacker based on the number of nodes it owns in the network, and based on the number of those that are good nodes it has compromised. We assume that the attacker owns all the cryptographic key information of compromised nodes and distributes it among all its nodes. We denote such an attacker Active-n-m, where n is the number of nodes it has compromised and m is the number of nodes it owns. We propose the following attacker hierarchy (with increasing strength) to measure routing protocol security: Active-0-1 (the attacker owns one node), Active-0-x (the attacker owns x nodes), Active-1-x (the attacker owns one compromised node and distributes the cryptographic keys to its $x - 1$ other nodes), and Active-y-x. In addition, we call an attacker that has compromised nodes an Active-VC attacker if it owns all nodes on a vertex cut through the network that partitions the good nodes into multiple sets, forcing good nodes in different partitions to communicate only through an attacker node. This attacker is particularly powerful, as it controls all traffic between nodes of the disjoint partitions.

Our protocols do not require a trusted Key Distribution Center (KDC) in the network, but some ad hoc networks may use one for key setup. We do not consider the case in which an attacker compromises the KDC, since the KDC is a central trust entity, and a compromised KDC compromises the entire network.

5.2.2. General Attacks on Ad Hoc Network Routing Protocols

Attacks on an ad hoc network routing protocols generally fall into one of two categories: *routing disruption* attacks and *resource consumption* attacks. In a routing disruption attack, the attacker attempts to cause legitimate data packets to be routed in dysfunctional ways. In a resource consumption attack, the attacker injects packets into the network in an attempt to consume valuable network resources such as bandwidth, or to consume node resources such as memory (storage) or computation power. From an application-layer perspective, both attacks are instances of a Denial-of-Service (DoS) attack.

An example of a routing disruption attack is for an attacker to send forged routing packets to create a *routing loop*, causing packets to traverse nodes in a cycle without reaching their destinations, consuming energy and available bandwidth. An attacker may similarly create a routing *black hole*, in which all packets are dropped: by sending forged routing packets, the attacker could route all packets for some destination to itself and then discard them, or the attacker could cause the route at all nodes in an area of the network to point "into" that area when in fact the destination is outside the area. As a special case of a black hole, an attacker could create a *gray hole*, in which it selectively drops some packets but not others, for example, forwarding routing packets but not data packets. An attacker may also attempt to cause a node to use *detours* (suboptimal routes) or may attempt to *partition* the network by injecting forged routing packets to prevent one set of nodes from reaching another. An attacker may attempt to make a route through itself appear longer by adding virtual nodes to the route; we call this attack *gratuitous detour*, as a shorter route exists and would otherwise have been used. In ad hoc network routing protocols that attempt to keep track of perceived malicious nodes in a "blacklist" at each node, such as is done in watchdog and pathrater [116], an attacker may *blackmail* a good node, causing other good nodes to add that node to their blacklists, thus avoiding that node in routes.

A more subtle type of routing disruption attack is the creation of a *wormhole* in the network (Chapter 9), using a pair of attacker nodes **A** and **B** linked via a private network connection. Every packet that **A** receives from the ad hoc network, **A** forwards through the wormhole to **B**, to then be rebroadcast by **B**; similarly, **B** may send all ad hoc network packets to **A**. Such an attack potentially disrupts routing by short circuiting the normal flow of routing packets, and the attackers may also create a virtual vertex cut that they control.

The *rushing* attack is a malicious attack that is targeted against on-demand routing protocols that use duplicate suppression at each node [80]. An attacker disseminates ROUTE REQUESTs quickly throughout the network, suppressing any later legitimate ROUTE REQUESTs when nodes drop them due to the duplicate suppression.

An example of a resource consumption attack is for an attacker to *inject extra data packets* into the network, which will consume bandwidth resources when forwarded, especially over detours or routing loops. Similarly, an attacker can *inject extra control packets* into the network, which may consume even more bandwidth or computational resources as other nodes process and forward such packets. With either of these attacks, an Active-VC attacker can try to extract maximum resources from the nodes on both sides of the vertex cut, for example by forwarding only routing packets and not data packets, such that the nodes waste energy forwarding packets to the vertex cut, only to have them dropped.

If a routing protocol can prevent an attacker from inserting routing loops, and if a maximum route length can be enforced, then an attacker that can inject extra data packets has limited attack power. In particular, if routes are limited to ν hops, then each data packet transmitted by the attacker only causes a fixed number of additional transmissions; more generally, if at most one control packet can be sent in response to each data packet (e.g., a ROUTE ERROR), and that control packet is limited to ν hops, then an individual data packet can cause only 2ν individual transmissions. We consider an attack a DoS attack only if the ratio between the total work performed by nodes in the network and the work performed by the attacker is on the order of the number of nodes in the network. An example of a DoS attack is where the attacker sends a single packet that results in a packet flood throughout the network.

5.2.3. Goals in Securing Ad Hoc Network Routing

In my work in securing ad hoc networks, I design protocols of varying strength, for use in different environments. For example, a military network would want to defend against all levels of attack, regardless of expense, whereas a sensor network may choose a cheaper form of security. Even when signficant resources are available, such as in the military environment, it does not suffice to secure only the physical layer, because node compromise is often possible, and secure hardware is often breakable with sufficient effort.

In this thesis, I present protocols with varying levels of security. SEAD (Chapter 6) provides basic security at little cost; Ariadne (Chapter 7) provides a higher level of security. Techniques such as Packet Leashes and Rushing Attack Prevention (Chapters 9 and 10) provide a much higher level of security, but at substantially increased cost.

Chapter 6

SEAD: Secure Efficient Distance Vector Routing

An insecure routing protocol can be subverted by an attacker in order to deny or reduce service available to participating nodes. In networks under attack, service can often be improved by using a secure routing protocol. The nature of many applications of ad hoc networking is such that some entity has a motive to attack the network. Such applications of ad hoc networking require security in order to guard against attacks such as malicious routing misdirection, but prior to the work described in this chapter, relatively little work had been done in securing ad hoc network routing protocols.

One reason for the slow development of secure ad hoc network routing protocols is that such protocols are difficult to design, due to the generally highly dynamic nature of an ad hoc network and due to the need to operate efficiently with limited resources, including network bandwidth and the CPU processing capacity, memory, and battery power (energy) of each individual node in the network. Existing *insecure* ad hoc network routing protocols are often highly optimized to spread new routing information quickly as conditions change, requiring more rapid and often more frequent routing protocol interaction between nodes than is typical in a traditional (e.g., wired and stationary) network. Expensive and cumbersome security mechanisms can delay or prevent such exchanges of routing information, leading to reduced routing effectiveness, and may consume excessive network or node resources, leading to many new opportunities for possible Denial-of-Service (DoS) attacks through the routing protocol.

In this chapter, we present a secure, periodic ad hoc network routing protocol based on a *distance vector* routing protocol. Our protocol, which we call the *Secure Efficient Ad hoc Distance vector* routing protocol (SEAD), is robust against multiple uncoordinated attackers creating incorrect routing state in any other node, even in spite of active attackers or compromised nodes in the network. We base the design of SEAD in part on the *Destination-Sequenced Distance-Vector* ad hoc network routing protocol (DSDV) (Section 1.2.2). In order to support use of SEAD with nodes of limited CPU processing capability, and to guard against Denial-of-Service attacks in which an attacker attempts to cause other nodes to consume excess network bandwidth or processing time, we use efficient *one-way hash functions* and do not use asymmetric cryptographic operations in the protocol.

Our mechanisms for securing SEAD can also be extended to on-demand versions of distance vector protocols, such as AODV (Section 1.2.2). In addition, in certain circumstances, periodic distance-vector protocols may be preferable, since distance vector routing protocols are easy to implement, require relatively little memory or CPU processing capacity compared to other types of routing protocols, and are widely used in networks of moderate size within the (wired) Internet [68, 110, 111].

6.1. Distance Vector Routing and DSDV

A distance vector routing protocol finds shortest paths between nodes in the network through a distributed implementation of the classical Bellman-Ford algorithm. Distance vector protocols are easy to implement and are efficient in terms of memory and CPU processing capacity required at each node. A popular example of a distance vector routing protocol is RIP [68, 111], which is widely used in IP networks of moderate size. Distance vector routing can be used for routing within an ad hoc network by having each node in the network act as a router and participate in the routing protocol.

In distance vector routing, each router maintains a routing table listing all possible destinations within the network. Each entry in a node's routing table contains the address (identity) of some destination, this node's shortest known distance (usually in number of hops) to that destination, and the address of this node's neighbor router that is the first hop on this shortest route to that destination; the distance to the destination is known as the *metric* in that table entry. When routing a packet to some destination, the node transmits the packet to the indicated neighbor router, and each router in turn uses its own routing table to forward the packet along its next hop toward the destination.

To maintain the routing tables, each node periodically transmits a routing update to to each of its neighbor routers, containing the information from its own routing table. Each node uses this information advertised by its neighbors to update its own table, so that its route for each destination uses as a next hop the neighbor that advertised the smallest metric in its update for that destination; the node sets the metric in its table entry for that destination to 1 (hop) more than the metric in that neighbor's update. A common optimization to this basic procedure to spread changed routing information through the network more quickly is the use of *triggered updates*, in which a node transmits a new update about some destination as soon as the metric in its table entry for that destination changes, rather than waiting for its next scheduled periodic update to be sent.

Distance vector routing protocols are simple, but they cannot guarantee not to produce routing loops between different nodes for some destination. Such loops are eventually resolved by the protocol through many rounds of routing table updates in what is known as "counting to infinity" in the metric for this destination; to reduce time needed for this resolution, the maximum metric value allowed by the protocol is typically defined to be relatively small, such as 15 as is used in RIP [68, 111]. To further reduce these problems, a number of extensions, such as *split horizon* and *split horizon with poisoned reverse* [68, 111], are widely used. These extensions, however, can still allow some loops, and the possible problems that can create routing loops are more common in wireless and mobile networks such as ad hoc networks, due to the motion of the nodes and the possible changes in wireless propagation conditions.

The primary improvement for ad hoc networks made in DSDV over standard distance vector routing is the addition of a *sequence number* in each routing table entry. The use of this sequence number prevents routing loops caused by updates being applied out of order; this problem may be common over multihop wireless transmission, since the routing information may spread along many different paths through the network. Each node maintains an *even* sequence number that it includes in each routing update that it sends, and each entry in a node's routing table is tagged with the most recent sequence number it knows for that destination. When a node detects a broken link to a neighbor, the node creates a new routing update for that neighbor as a destination, with an "infinite" metric and the next *odd* sequence number after the even sequence number in its corresponding routing table entry. When a node receives a routing update, for each destination in the update, the node prefers this newly advertised route if the sequence number is greater than in the corresponding entry currently in the node's routing table, or if the sequence numbers are equal and the new metric is lower than in the node's current table entry for that destination; if the sequence number in the update is less than the current sequence number in the table entry, the new update for that destination is ignored.

DSDV sends both periodic routing updates and triggered updates. These updates may be either a "full dump," listing all destinations, or an "incremental" update, listing only destinations for which the route has changed since the last full dump sent by that node. A node in DSDV chooses to send a triggered update when important routing changes should be communicated as soon as possible, although there are multiple interpretations suggested in the published description of DSDV as to which changes should cause a triggered update. One interpretation suggests that the receipt of a new metric for some destination should cause a triggered update, while the alternative interpretation suggests that the receipt of a new sequence number also should cause a triggered update. The latter interpretation has been shown to outperform the former in detailed ad hoc network simulations [26, 86] and is referred to as DSDV-SQ (for sequence number) to distinguish it from the interpretation based only on metrics.

6.2. Assumptions

As a matter of terminology in this chapter, we use the acronym "MAC" to refer to the network Medium Access Control protocol at the link layer, and not to a Message Authentication Code used for authentication.

We assume that all wireless links in the network are bidirectional, since this is necessary for the distributed Bellman-Ford algorithm of distance vector routing to function correctly. Specifically, if a node **A**'s wireless transmissions reach **B**, then **B**'s transmissions would reach **A**. Wireless links are often bidirectional, and many MAC layers require bidirectional frame exchange to avoid collisions [84].

Network physical layer and MAC layer attacks are beyond the scope of this chapter. Use of spread spectrum has been studied for securing the physical layer against jamming [149]. MAC protocols that do not employ some form of carrier sense, such as ALOHA and Slotted ALOHA [1], are less vulnerable to Denial-of-Service attacks, although they generally use the channel less efficiently.

We assume that the wireless network may drop, corrupt, duplicate, or reorder packets. We also assume that the MAC layer contains some level of redundancy to detect randomly corrupted packets; however, this mechanism is not designed to replace cryptographic authentication mechanisms.

The *network diameter* of an ad hoc network is the maximum, across all pairs of nodes in the network, of the length of the optimal route between that pair of nodes. As noted in Section 6.1, standard distance vector routing protocols limit the maximum metric value (and thus the maximum network diameter supported by the protocol). We also limit the maximum network diameter, and we use $m - 1$ to denote this upper bound, such that all routes that can be used by the routing protocol are of length less than m hops. Internal to a node's routing table, the value m can be used to denote the infinity metric in distance vector routing, although in SEAD, entries in the routing table with an infinite metric are not included in routing update messages sent by a node.

We assume that nodes in the ad hoc network may be resource constrained. Thus, in securing our distance vector ad hoc network routing protocol SEAD, we use efficient *one-way hash chains* [103] rather than relying on expensive asymmetric cryptographic operations. Especially on CPU-limited devices, symmetric cryptographic operations (such as block ciphers and hash functions) are three to four orders of magnitude faster than asymmetric operations (such as digital signatures). One-way hash chains are described in Section 5.1.3.

To use one-way hash chains for authentication, we assume some mechanism for a node to distribute the anchor from its generated hash chain. A traditional approach for this key distribution is for a trusted entity to sign public-key certificates for each node; each node can then use its public-key to sign new a hash chain element for itself. Hubaux, Buttyán, and Čapkun bootstrap trust relationships from PGP-like certificates without relying on a trusted public key infrastructure [82]. Alternatively, a trusted node can securely distribute an authenticated hash chain element using only symmetric-key cryptography [79, 148] or non-cryptographic approaches [178].

Since in SEAD, a node uses elements from its one-way hash chain in groups of m, we assume that a node generates its hash chain so that n is divisible by m. When a node first enters the network, or after a node has used most of its available hash chain elements, it can generate a new hash chain, and send the anchor of the new hash chain to a trusted entity or an alternative authentication and distribution service, as described above.

6.3. Attacks

In Section 5.2.2, we presented several general attacks against ad hoc network routing protocol. In this section, we discuss attacks specific to *distance vector* routing; many of the attacks discussed earlier

An attacker can attempt to reduce the amount of routing information available to other nodes, by failing to advertise certain routes or by destroying or discarding routing packets or parts of routing packets. A node failing to advertise a route indicates its unwillingness to forward packets for those destinations. We do not attempt to defend against this attack, since the attacker could also otherwise drop data packets sent to those destinations. A node can drop routing packets it receives, in which case it becomes ignorant of links available to it and fails to pass potentially improved knowledge to its neighbors. This *ignorance attack* has even more limited impact than failing to advertise routes that the node itself knows. Finally, an intruder can jam routing packets; we will disregard such attacks in this chapter, since prevention of such attacks begins at the physical layer.

An attacker can modify an advertisement by changing the destination, metric, or source address (and hence next-hop). For example, an attacker advertising a zero metric for all destinations can cause all nodes around it to route packets for all destinations toward it rather than toward each actual destination. Alternatively, an attacker can modify the source address of the advertisement, thus spreading inaccurate next-hop information.

An attacker can mount a *replay attack* by sending an old advertisement to some node, in an attempt to get that node to update its routing table with stale routes.

6.4. Securing Distance Vector Routing

6.4.1. Basic Design of SEAD

We base the design of our secure routing protocol SEAD on the DSDV-SQ version [26] of the DSDV ad hoc network routing protocol, as described in Section 6.1. In particular, to avoid long-lived routing loops in SEAD, we use destination sequence numbers, as in DSDV; we also use these destination sequence numbers to provide replay protection of routing update messages in SEAD.

We differ from DSDV in that we do not use an average weighted settling time in sending triggered updates. To reduce the number of redundant triggered updates, each node in DSDV tracks, for each destination, the average time between when the node receives the *first* update for some new sequence number for that destination, and when it receives the *best* update for that sequence number for it (with the minimum metric among those received with that sequence number); when deciding to send a triggered update, each DSDV node delays any triggered update for a destination for this average weighted settling time, in the hope of only needing to send one triggered update, with the best metric, for that sequence number.

SEAD does not use such a delay, in order to prevent attacks from nodes that might maliciously not use the delay. Since a node selects the first route it receives with highest sequence number and lowest metric, an attacker could otherwise attempt to cause more traffic to be routed through itself, by avoiding the delay in its own triggered updates. Such an attack could otherwise put the attacker in a position to eavesdrop on, modify, or discard other nodes' packets.

In addition, unlike DSDV, when a node detects that its next-hop link to some destination is broken, the node does not increment the sequence number for that destination in its routing table when it sets the metric in that entry to infinity. Since higher sequence numbers take priority, this node's routing update with this new sequence number must be authenticated, but we did not include a mechanism for authenticating these larger sequence numbers. Instead, the node flags its routing table entry for this destination to not accept any new updates for this same sequence number, effectively preventing the possible routing loop and traditional distance vector "counting to infinity" problem [68, 111] that could otherwise occur in this case.

6.4.2. Metric and Sequence Number Authenticators

In addition to the differences between our SEAD protocol and DSDV-SQ described in Section 6.4.1, the lower bound on each metric in a routing update in SEAD is secured through authentication; in addition, the receiver of SEAD routing information also authenticates the sender (ensures that the routing information originates from the correct sender). We describe the authentication of the lower bound on the distance metric in this section and the neighbor authentication in the following section. Whereas DSDV-SQ (and DSDV) are subject to all of the attacks in Section 6.3, SEAD thus resists those attacks. SEAD is robust against multiple uncoordinated attackers creating incorrect routing state in any other node, even in spite of active attackers or compromised nodes in the network. A description of the detailed security properties provided by the complete SEAD protocol is provided in Section 6.5.1.

One possible approach that could be used for authenticating routing updates in a distance vector routing protocol is for each node to sign each of its routing updates using asymmetric cryptography. However, this approach raises three distinct problems for use in an ad hoc network.

First, an attacker could send a large number of arbitrary forged routing updates to some victim node, such that the victim is forced to spend all of its CPU resources attempting to verify this stream of updates, creating an effective Denial-of-Service attack; this attack would be particularly easy in many ad hoc networks, since ad hoc network nodes tend to have less powerful CPUs than workstations in wired networks. Second, an attacker who has compromised a node can send updates claiming that any other node is a neighbor (metric 1), causing other nodes to incorrectly direct packets for this destination node toward the attacker. Finally, even with no attacker present, the larger signatures and longer signature generation and verification times of asymmetric cryptography would reduce the resources that could otherwise be used for running useful applications and doing useful communication; this problem is more severe in an ad hoc network than in a traditional (i.e., wired and stationary) network due to the limited resources of nodes and links in an ad hoc network, such as available bandwidth, CPU capacity, and battery power (energy).

Instead, in securing routing in SEAD, we use efficient one-way hash chains [103]. The basic operation of a one-way hash chain was described in Section 6.2. Each node in SEAD uses a specific single next element from its hash chain in each routing update that it sends about itself (metric 0). Based on this initial element, the one-way hash chain conceptually provides authentication for the lower bound of the metric in other routing updates for this destination; the authentication provides only a lower bound on the metric, since it does not prevent a malicious node from claiming the *same* metric as the node from which it heard this route. In particular, the one-way hash function provides the property that another node can only increase a metric in a routing update, but cannot decrease it. Due to the properties of the one-way hash function, given any value in the hash chain, an attacker cannot generate any value in the chain that will be used by this node in a future update that it sends about itself (a value to the "left" of the given value in the chain, with larger subscript). Similarly, for each entry in its routing update describing a route to another destination, the hash chain of that destination node allows the metric in that entry to be authenticated by nodes receiving it.

As noted in Section 6.2, we assume that an upper bound can be placed on the diameter of the ad hoc network, and we use $m - 1$ to denote this bound. Thus, within the routing protocol, all metrics in any routing update are less than m. The method used by SEAD for authenticating an entry in a routing update uses the *sequence number* in that entry to determine a contiguous group of m elements from that destination node's hash chain, one element of which must be used to authenticate that routing update. The particular element from this group of elements that must be used to authenticate the entry is determined by the *metric* value being sent in that entry. Specifically, if a node's hash chain is the sequence of values

$$h_n,\ h_{n-1},\ h_{n-2},\ h_{n-3},\ \ldots,\ h_0$$

and n is divisible by m, then for a sequence number i in some routing update entry, an element from the group of elements

$$h_{mi},\ h_{mi-1},\ \ldots,\ h_{m(i-1)+1}$$

from this hash chain is used to authenticate the entry; if the metric value for this entry is j, $0 \leq j < m$, then the value h_{mi-j} here is used to authenticate the routing update entry for that sequence number.

When a node in SEAD sends a routing update, the node includes one hash value with each entry in that update. If the node lists an entry for itself in that update, it sets the address in that entry to its own node address, the metric to 0, the sequence number to its own next sequence number, and the hash value to the first element in the group of its own hash chain elements corresponding to that sequence number. In the example given above for sequence number i, the node sets the hash value in that entry to its h_{mi}. If the node lists an entry for some other destination in that update, it sets the address in that entry to that destination node's address, the metric and sequence number to the values for that destination in its routing table, and the hash value to the hash of the hash value received in the routing update entry from which it learned that route to that destination.

This use of a hash value corresponding to the sequence number and metric in a routing update entry prevents any node from advertising a route to some destination claiming a greater sequence number than that destination's own current sequence number, due to the one-way nature of the hash chain. Likewise, no node can advertise a route better than those for which it has received an advertisement, since the metric in an existing route cannot be decreased.

Nodes receiving any routing update can easily authenticate each entry in the update, given any earlier authentic hash element from the same hash chain, as described in Section 6.2. In order to guard against attacks in which a malicious update claiming a high sequence number attempts to force a receiving node to perform a large number of hash operations in order to authenticate the update, a receiving node may limit the number of hashes it is willing to perform for each such authentication, discarding updates that cannot be authenticated; since DSDV-SQ (and thus SEAD) spreads new routing information across the network, this limit assumes a bound on the number of routing updates about a destination that the receiving node may have missed before any authentic update is received. A similar solution to such an attack would be to have each node tie its own sequence number generation to a loosely synchronized clock value, thus allowing a receiving node to determine if a claimed sequence number in an update could be authentic before performing the implied hashes to confirm that fact.

When a node receives a routing update, for each entry in that update, the node checks the authentication on that entry, using the destination address, sequence number, and metric in the received entry, together with the latest prior authentic hash value received by this node from that destination's hash chain. Based on the sequence number and metric in the received entry and the sequence number and metric of this latest prior authentic hash value for that destination, the node hashes the hash value received in this entry the correct number of times, according to the description above as to which hash value must be used for any given sequence number and metric, to confirm that the resulting value equals the prior authentic hash value. If so, the entry is authentic and the node processes it in the routing algorithm as a normal received routing update entry; otherwise, the node ignores the received entry and does not modify its routing table based on it.

It may be possible for an attacker to modify routing update messages in transit, and such an attacker would be able to prevent certain routes from being advertised; however, such an attacker would also be able to corrupt the entire routing update, which is equivalent to a jamming attack. The protocol can also be secured against modification of the source address for a routing update and against wormhole attacks, by use of other mechanisms at the MAC layer, including mechanisms that rely only on symmetric cryptography [81]. In particular, these MAC layer approaches authenticate the transmitting source of a packet and ensure that this transmitting source is within some distance of the receiver.

6.4.3. Neighbor Authentication

The source of each routing update message in SEAD must also be authenticated, since otherwise, an attacker may be able to create routing loops. Any efficient broadcast authentication mechanism, such as TESLA [145, 147], HORS [160], or TIK [81], can be used to authenticate the neighbor. The drawbacks of these approaches are that they require synchronized clocks, and that they incur either an authentication delay or a relatively high communication overhead.

An alternative approach that does not require time synchronization is to assume a shared secret key among each pair of nodes, and to use the respective key in conjunction with a Message Authentication Code. The sender would include one Message Authentication Code for each neighbor with each routing update. Since SEAD includes periodic neighbor sensing functionality, each node knows the set of neighbors for which it needs to authenticate routing updates. In particular, each node trusts any zero-metric update with a valid authenticator; if a node has received such an update from another node for a recent sequence number, it considers that node a neighbor and computes a Message Authentication Code for it in subsequent updates.

When two nodes first become neighbors, one of the two nodes will transmit a routing update first. That update will cause the receiving node to detect the new neighbor. As a result of hearing this update, the receiving node will send a triggered routing update, allowing the other node to detect the new neighbor.

Table 6.1: Parameters for SEAD performance study

SEAD Parameters

Periodic Route Update Interval	15 seconds
Periodic Updates Missed before Link is Declared Broken	3
Maximum Packets Buffered per Node per Destination	5
Hash Length (ρ)	80 bits

6.5. Evaluation

6.5.1. Security Analysis

Securing a distance vector protocol seems fundamentally harder than securing link-state or on-demand protocols such as DSR [88]. Since distance vector protocols compress the route information into a hop count value and a next hop, it is challenging to verify the correctness of the hop count value. In this section, we discuss some of the security properties of the SEAD protocol.

Using SEAD, given an advertisement for a route with a metric of h hops and a sequence number of s, a malicious node can generate advertisements for h-hop or longer routes with sequence number s, or for arbitrary-length routes with sequence number less than s. Specifically, a malicious node cannot generate an advertisement with sequence number greater than s, nor can it generate an advertisement with sequence number s and metric less than h. A malicious node can generate an advertisement for distance h because it can simply resend the same one-way hash chain element it received from the previous node; a legitimate node would advertise a distance of $h+1$ and generate the authenticator for it by hashing the received authenticator.

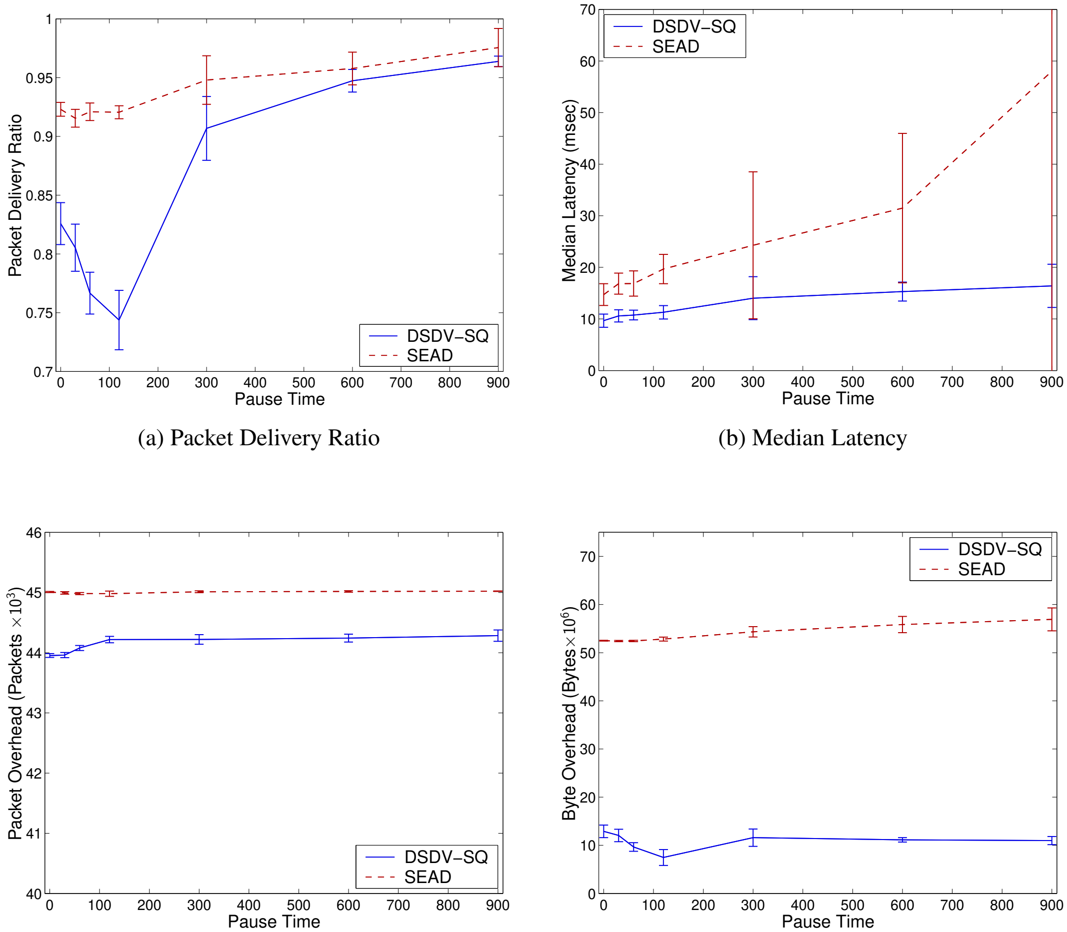
An attacker that has not compromised any node (and hence does not possess any cryptographic keys from a node), cannot successfully send any routing messages, since an uncompromised neighbor node will reject the messages due to the failed neighbor authentication. A repeater can function as a one-node wormhole; this is not addressed by SEAD, though TIK [81] can prevent this attack.

A collection of a number of attackers that have compromised one or more nodes can only redirect the path from a source to a destination through one or more attackers if the length of the best (minimum metric) attacker-free route for which the source receives an advertisement is at least as large as the number of nodes between the destination and the first attacker, plus the number of nodes between the last attacker and the destination.

If each node using SEAD (including attackers) keeps routing tables where the next-hop for a given destination is set to the authenticated source address of the first advertisement received by that node containing the minimum metric for the greatest sequence number, then the next-hop pointers in all nodes' routing tables will describe a route back to the destination.

With SEAD, no routing loop is possible, unless the loop contains one or more attackers. Furthermore, no loop is possible unless no non-attacker node on the loop has received a better advertisement (in terms of sequence number and metric) for this destination than the best advertisement received by some attacker on the loop.

If a collection of attackers form a vertex cut between two groups of nodes in the network [79], the attackers can arbitrarily control the routes between any node in one group and a node in the other group. Since in a vertex cut, any packet between such nodes must physically pass through a node on the vertex cut, no routing protocol can eliminate such attacks.



(a) Packet Delivery Ratio

(b) Median Latency

(c) Packet Overhead

(d) Byte Overhead

Figure 6.1: SEAD performance evaluation results (average over 65 runs)

6.5.2. Simulation Evaluation Methodology

To evaluate the performance impact of our security approach in SEAD without attackers, we modified the DSDV-SQ implementation in our extensions to *ns-2* (Section 2.3.1). Specifically, we increased the size of each routing update to represent the authentication hash value in each table entry. We also removed the settling time and the sequence number changes, as described in Section 6.4.1.

Our simulation parameters were identical to those described in Section 3.2, except that we generated 65 scenarios for each pause time. We evaluated SEAD by comparing it to DSDV-SQ, as described in Section 6.1. We measured performance along four metrics: packet delivery ratio (Section 2.3.3), packet overhead (Section 2.3.3), byte overhead (Section 3.2), and median latency, defined to be the median latency of delivered packets, where latency is calculated as the elapsed time between the application layer passing a packet to the routing layer and that packet first being received at the destination.

6.5.3. Simulation Results

The results of our performance study of SEAD are shown in Figure 6.5.2 as a function of pause time in the random waypoint mobility model. Each figure represents the average over 65 randomly generated runs at each pause time, and the error bars show the 95% confidence intervals; the runs used for SEAD and those for DSDV-SQ were identical. On the right side of each graph (pause time 900), the nodes are stationary, and on the left side of each graph (pause time 0), the nodes are all in continuous motion.

The packet delivery ratios for SEAD and DSDV-SQ are shown in Figure 6.1(a), and the median latency of delivered application-level packets for these simulations is shown in Figure 6.1(b). Surprisingly, SEAD consistently outperforms DSDV-SQ in terms of packet delivery ratio. By not using a weighted settling time delay in sending triggered updates in SEAD, the number of routing advertisements sent by SEAD generally increases relative to DSDV-SQ, allowing nodes to have more up-to-date routing tables.

However, SEAD also increases overhead, both due to this increased number of routing advertisements, and due to the increase in size of each advertisement from the addition of the hash value on each entry for authentication. This increased overhead is shown in Figures 6.1(c) and 6.1(d), which show the number of routing overhead packets and the number of routing overhead bytes, respectively, caused by the two protocols in these same simulations. The vertical scale in Figure 6.1(c) is magnified to show the difference between the two protocols; the vertical scale here ranges only between 40 and 46.

The increased overhead in SEAD causes some congestion in the network in these simulations, as shown in the latency results in Figure 6.1(b). At all pause times, SEAD exhibits higher latency than DSDV-SQ, due to the decreased available network capacity from the increased overhead in SEAD. The rise in latency at higher pause times is due to the nonuniform distribution of nodes in space caused by node motion in the random waypoint model. Although the initial node locations and the locations to which each node moves during the run are uniformally chosen over the space, the straight line path of a node from one location to the next tends to distribute nodes on average closer to the center of the space; at higher pause times, nodes spend most (or all) of the time in their initial uniformally distributed locations. For example over the 65 simulation runs, the average route length used by SEAD at pause time 900 was about 28% longer than at pause time 0 (for DSDV-SQ, the average route length at pause time 900 was about 33% longer than at pause time 0). This increased route length, together with SEAD's increased overhead, created additional congestion at higher pause times in the simulations.

6.6. Related Work

Prior to the publication of our work [78], a number of techniques had been proposed for securing distance-vector protocols and ad hoc network routing protocols.

Kumar [102] discusses attacks against distance vector routing protocols, and describes mechanisms to secure them using Message Authentication Codes. Although these mechanisms ensure the integrity of router-to-router communications, they do not withstand node compromise. In particular, they do not secure the metric in each routing table entry, and thus a compromised router could claim routes of any length to any destination.

Smith et al [177] discuss attacks against distance vector routing protocols, and present countermeasures that provide security. However, their techniques do not apply well in an ad hoc network since they require knowledge of which links are possible, whereas in an ad hoc network, any pair of nodes could be within range and form a link.

Zapata [190] proposes security extensions to AODV, using a new one-way hash chain for each Route Discovery to secure the metric field in an RREQ packet. Our protocol uses a single hash chain for a node's routing information and can therefore authenticate sequence number information, and also minimizes the overhead of authenticating new hash chains.

A number of security protocols have been designed for RIPv2 [8, 111]. These protocols protect the integrity of the packet from modification, but they do not prevent a node from advertising a route that does not actually exist.

Several researchers have proposed the use of *asymmetric cryptography* to secure both wired and ad hoc network routing protocols [96, 143, 170, 190, 192]. However, when the nodes in an ad hoc network are unable to verify asymmetric signatures quickly enough, these protocols may not be suitable and may create Denial-of-Service (DoS) attacks; these protocols also generally require more network bandwidth than does SEAD with its hash values.

Cheung [33] and Hauser et al [65] describe *symmetric-key* approaches to the authentication of updates in link state protocols, but neither work discusses the mechanisms for detecting the status of these links. In wired networks, a common technique for authenticating HELLO packets is to verify that the the incoming network interface is the expected interface and that the IP TTL of the packet is 255. In a wireless network, this technique cannot be used. Heffernan [69] and Basagni et al [10] use shared keys to secure routing communication, which is vulnerable to some single-node compromises. Perrig et al [148] use symmetric primitives to secure routing only between nodes and a trusted base station.

As mentioned in Section 6.2, some researchers have explored the establishment of trust relationships and authenticated keys in ad hoc networks [79, 82, 148, 178].

Marti et al [116] consider the problem of detecting intermediate nodes that do not forward packets. However, their scheme is limited to certain types of network Medium Access Control layers and may trigger false alarms in congested networks.

6.7. Chapter Summary

In this chapter, we have presented the design and evaluation of SEAD, a new secure ad hoc network routing protocol using distance vector routing. Many previous routing protocols for ad hoc networks have been based on distance vector approaches (e.g., [19, 55, 62, 91, 129, 140]), but they have generally assumed a trusted environment. Instead, in designing SEAD, we carefully fit inexpensive cryptographic primitives to each part of the protocol functionality to create an efficient, practical protocol that is robust against multiple uncoordinated attackers creating incorrect routing state in any other node, even in spite of active attackers or compromised nodes in the network. Together with existing approaches for securing the physical layer and MAC layer within the network protocol stack, the SEAD protocol provides a foundation for the secure operation of an ad hoc network.

We base the design of SEAD in part on the DSDV ad hoc network routing protocol [140], and in particular, on the DSDV-SQ version of the protocol, which has been shown to outperform other DSDV versions in previous detailed ad hoc network simulations [26, 86]. For security, we use efficient one-way hash functions and do not use asymmetric cryptographic primitives. Consequently, SEAD is efficient and can be used in networks of computation- and bandwidth-constrained nodes. SEAD actually outperforms DSDV-SQ in terms of packet delivery ratio, although it does create more overhead in the network, both due to an increased number of routing advertisements it sends, and due to the increase in size of each advertisement due to the addition of the hash value on each entry for authentication.

In future work, we plan to also consider mechanisms to detect and expose nodes that advertise routes but do not forward packets, and to merge this work with our other work in securing on-demand routing protocols to create a secure protocol based on ZRP [62]. We are also considering the possibility of extending DSDV to behave like a path-vector routing protocol, allowing the source address of each advertisement to be more readily authenticated.

Chapter 7

Ariadne: A Secure On-Demand Routing Protocol

One metric for measuring the service provided by an ad hoc network routing protocol is according to its ability to deliver packets in the presence of hostile nodes. SEAD (Chapter 6) provides some level of resiliance against a single compromised node, or against multiple compromised nodes that do not collaborate. However, in certain types of network, an attacker may be reasonably able to compromise more than one node. In this chapter, we discuss two contributions to the area of secure routing protocols for ad hoc networks which provide a stepping stone for achieving security even when multiple nodes have been compromised. First, we give a model for the types of attacks possible in such a system, and we describe several new attacks on ad hoc network routing protocols. Second, we present the design and performance evaluation of a new on-demand secure ad hoc network routing protocol, called Ariadne, that withstands node compromise and relies only on highly efficient *symmetric* cryptography. Relative to previous work in securing ad hoc network routing protocols (e.g., [10, 116, 136, 148, 170, 190, 192]), Ariadne provides better service, either by being more secure, more efficient, or more general (e.g., Ariadne does not require trusted hardware and does not require powerful processors). In particular, Pebblenets [10] requires specialized hardware and detection schemes [116] are prone to failure when nodes collude. SRP [136] can be broken without compromising a single node, since SRP relies on most nodes being legitimate; an attacker that can masquerade as many other nodes can severely degrade the security of this approach. Finally, ARAN [170], SAODV [190], and the schemes proposed by Zhou and Haas [192] use asymmetric cryptography, which makes them significantly less efficient. This reduced efficiency exposes them to denial-of-service attacks, as described in Section 7.4.

Ariadne can authenticate routing messages using one of three schemes: shared secrets between each pair of nodes, shared secrets between communicating nodes combined with broadcast authentication, or digital signatures. We primarily discuss here the use of Ariadne with TESLA [147, 148], an efficient broadcast authentication scheme that requires loose time synchronization. Using pairwise shared keys avoids the need for synchronization, but at the cost of higher key setup overhead; broadcast authentication such as TESLA also allows some additional protocol optimizations.

7.1. Assumptions

7.1.1. Network Assumptions

The physical layer of a wireless network is often vulnerable to denial of service attacks such as jamming. Mechanisms such as spread spectrum [149] have been extensively studied as means of providing resistance to physical jamming, and we thus disregard such physical layer attacks here.

We assume that network links are bidirectional; that is, if node **A** is able to transmit to some node **B**, then **B** is able to transmit to **A**. It is possible to use a network with unidirectional links if such links are detected and avoided; such detection may also otherwise be necessary, since many wireless Medium Access Control protocols require bidirectional links, as they require the exchange of several link-layer frames between a source and destination to help avoid collisions [16, 84].

Medium Access Control protocols are also often vulnerable to attack. For example, in IEEE 802.11, an attacker can paralyze nodes in its neighborhood by sending Clear-To-Send (CTS) frames periodically, setting the "Duration" field of each frame greater than or equal to the interval between such frames. Less sophisticated Medium Access Control protocols, such as ALOHA and Slotted ALOHA [1], are not vulnerable to such attacks but have lower efficiency. In this chapter, we disregard attacks on Medium Access Control protocols.

We assume that the network may drop, corrupt, reorder, or duplicate packets in transmission.

When Ariadne is used with a broadcast authentication protocol, we inherit all of its assumptions. For example, when TESLA is used, each node in the network must be able to estimate the end-to-end transmission time to any other node in the network; TESLA permits this value to be chosen adaptively and pessimistically. When this time is chosen to be too large, authentication delay increases, reducing protocol responsiveness; when it is chosen to be too small, authentic packets may be rejected, but security is not compromised.

7.1.2. Node Assumptions

The resources of different ad hoc network nodes may vary greatly, from nodes with very little computational resources, to resource-rich nodes equivalent in functionality to high-performance workstations. To make our results as general as possible, we have designed Ariadne to support nodes with few resources, such as a Palm Pilot or RIM pager.

Most previous work on secure ad hoc network routing relies on *asymmetric* cryptography such as digital signatures [190, 192]. However, computing such signatures on resource-constrained nodes is expensive, and we assume that nodes in the ad hoc network may be so constrained. For example, Brown et al analyze the computation time of digital signature algorithms on various platforms [27]; on a Palm Pilot or RIM pager, a 512-bit RSA [163] signature generation takes 2.4–5.8 seconds and signature verification takes 0.1–0.6 seconds, depending on the public exponent.

When Ariadne uses TESLA for broadcast authentication, we assume that all nodes have loosely synchronized clocks, such that the difference between any two nodes' clocks does not exceed Δ; the value of Δ must be known by all nodes in the network. Accurate time synchronization can be maintained with off-the-shelf hardware based on GPS [35, 183], although the time synchronization signal itself may be subject to attack [54]. We assume that nodes compensate clock drift with periodic re-synchronization. Microcomputer-compensated crystal oscillators [14] can provide subsecond accuracy for several months; if normal crystal oscillators are used, Δ can be chosen to be as large as necessary, though a corresponding reduction in protocol responsiveness will result.

We do not assume trusted hardware such as tamperproof modules. Secure routing with trusted hardware is much simpler, since node compromise is assumed to be impossible.[1]

7.1.3. Security Assumptions and Key Setup

The security of Ariadne relies on the secrecy and authenticity of keys stored in nodes. Ariadne relies on the following keys to be set up, depending on which authentication mechanism is used:

- If **pairwise shared secret keys** are used, we assume a mechanism to set up the necessary $n(n+1)/2$ keys in a network with n nodes.
- If **TESLA** is used, we assume a mechanism to set up shared secret keys between communicating nodes, and to distribute one authentic public TESLA key for each node.
- If **digital signatures** are used, we assume a mechanism distribute one authentic public key for each node.

To set up shared secret keys, we can use a variety of mechanisms: a key distribution center shares a secret key with each node and sets up shared secret keys with communicating nodes, such as in Kerberos [100] or SPINS [148]; bootstrap shared secret keys from a Public Key Infrastructure (PKI) using protocols such as TLS [44]; or pre-load shared secret keys at initialization, possibly through physical contact [178]. Menezes et al discuss several key setup protocols [121].

To set up authentic public keys, we can either embed all public keys at initialization in each node, or assume a PKI and embed the trusted Certification Authority's public key in each node and then use that key to authenticate the public keys of other nodes. Another approach proposed by Hubaux et al [82] bootstraps trust relationships based on PGP-like certificates.

Ariadne also requires that each node have an authentic element from the Route Discovery chain (Section 7.2.6) of every node initiating Route Discoveries. These keys can be set up in the same way as a public key.

Key setup is an expensive operation. Setting up shared secret keys requires authenticity and confidentiality, whereas setting up public keys only requires authenticity. Furthermore, fewer public keys are generally needed, because in a network with n nodes only n public keys are needed, and can potentially be broadcast, whereas $n(n+1)/2$ secret keys need to be set up in the case of pairwise shared secret keys.

We outline here a mechanism to set up these keys without relying on Ariadne, thus avoiding the circular dependency between key setup and a routing protocol. We assume for this a trusted Key Distribution Center (KDC) that either shares a secret key with each node, or uses its private or TESLA key to broadcast authenticated public keys of nodes. In either case, a star-based routing protocol that allows routing between nodes and the trusted entity suffices. To bootstrap authenticated keys between pairs of nodes, the KDC node initiates a Route Discovery with a special, reserved address (not the address of any actual node) as the target of the Discovery. The Route Discovery is processed as in Ariadne (Section 7.2), except that each node receiving the ROUTE REQUEST for the first time also returns a ROUTE REPLY. The KDC can then use each returned route to send encrypted, authenticated keys to each node in the network.

---

[1] In the terms of the attacker classification we present in Section 5.2.1, the strongest attacker in such an environment is Active-0-x. As we discuss in Section 7.2.2, we can thus secure a network with tamperproof hardware through a network-wide shared secret key for all message authentication, with *packet leashes* (Chapter 9) (both implemented within the secure hardware).

7.2. Ariadne

7.2.1. Notation

We use the following notation to describe security protocols and cryptographic operations:

- A, B are principals, such as communicating nodes.
- K_{AB} and K_{BA} denote the secret MAC keys shared between A and B (one key for each direction of communication).
- $\text{MAC}_{K_{AB}}(M)$ denotes the computation of the message authentication code (MAC) of message M with the MAC key K_{AB}, for example using the HMAC algorithm [11].

For notational convenience we assume hash and MAC functions that take a variable number of arguments, simply concatenating them in computing the function.

7.2.2. Design Goals

We aim for resilience against Active-1-x and Active-y-x attackers. Ideally, the probability that the routing protocol delivers messages degrades gracefully when nodes fail or are compromised. Our goal is to design simple and efficient mechanisms achieving high attack robustness. These mechanisms should be sufficiently general to allow application to a wide range of routing protocols.

Defending against an Active-0-x attacker is relatively easy. A network-wide shared secret key limits the attacker to replaying messages. Thus the main attacks remaining are the wormhole and rushing attacks (Section 5.2.2). *Packet leashes* (Chapter 9) can prevent both attacks because they prevent an Active-0-x attacker from retransmitting packets.

Most routing disruption attacks we present in Section 5.2.2 are caused by malicious injection or altering of routing data. To prevent these attacks, each node that interprets routing information must verify the origin and integrity of that data, that is, authenticate the data. Ideally, the initiator of the Route Discovery can verify the origin of each individual data field in the ROUTE REPLY.

We need an authentication mechanism with low computation and communication overhead. An inefficient authentication mechanism could be exploited by an attacker to perform a Denial-of-Service (DoS) attack by flooding nodes with malicious messages, overwhelming them with the cost of verifying authentication. Thus, for point-to-point authentication of a message, we use a message authentication code (MAC) (e.g., HMAC [11]) and a shared key between the two parties. However, setting up the shared keys between the initiator and all the nodes on the path to the target may be expensive. We thus also propose using the TESLA broadcast authentication protocol (Section 5.1.4) for authentication of nodes on the routing path. However, we also discuss MAC authentication with pairwise shared keys, for networks capable of inexpensive key setup, and we discuss digital signatures for authentication, for networks with extremely powerful nodes.

As a general design principle, a node trusts only itself for acquiring information about which nodes in the network are malicious. This approach helps avoid blackmail attacks, where an attacker constructs information to make a legitimate node appear malicious.

In our design, we assume that a sender trusts the destination with which it communicates, for authenticating nodes on the path between them. This assumption is straightforward, as the destination node can control all communication with the sender anyway. However, the destination node can potentially blackmail nodes on the path to the sender. The sender thus needs to keep a separate blacklist for each destination.

In general, ad hoc network routing protocols do not need *secrecy* or *confidentiality*. These properties are required to achieve privacy or anonymity for the sender of messages. Even in the Internet, it is challenging to achieve sender anonymity, and this area is still the subject of active research.

Our protocol does not prevent an attacker from injecting data packets. As we describe in Section 5.2.2, injecting a packet results in a DoS attack only if it floods the network. Since data packets cannot flood the network, we do not explicitly protect against packet injection. However, malicious ROUTE REQUEST messages that flood the network do classify as a DoS attack, and we thus prevent this attack with a separate mechanism that we describe in Section 7.2.6.

7.2.3. Basic Ariadne Route Discovery

We present the design of the Ariadne protocol in three stages: we first present a mechanism that enables the target to verify the authenticity of the ROUTE REQUEST; we then present three alternative mechanisms for authenticating data in ROUTE REQUESTs and ROUTE REPLYs; and finally, we present an efficient per-hop hashing technique to verify that no node is missing from the node list in the REQUEST. In the following discussion we assume that the initiator **S** performs a Route Discovery for target **D**, and that they share the secret keys K_{SD} and K_{DS}, respectively, for message authentication in each direction.

Target authenticates ROUTE REQUESTs. To convince the target of the legitimacy of each field in a ROUTE REQUEST, the initiator simply includes a MAC computed with key K_{SD} over unique data, for example a timestamp. The target can easily verify the authenticity and freshness of the route request using the shared key K_{SD}.

Three techniques for data authentication. In a Route Discovery, the initiator wants to authenticate each individual node in the node list of the ROUTE REPLY. A secondary requirement is that the target can authenticate each node in the node list of the ROUTE REQUEST, so that it will return a ROUTE REPLY only along paths that contain only legitimate nodes. In this section, we present three alternative techniques to achieve node list authentication: the TESLA protocol, digital signatures, and standard MACs.

When Ariadne Route Discovery is used with **TESLA**, each hop authenticates new information in the REQUEST. The target buffers the REPLY until intermediate nodes can release the corresponding TESLA keys. The TESLA security condition is verified at the target, and the target includes a MAC in the REPLY to certify that the security condition was met. TESLA requires each packet sender to choose a τ as the maximum end-to-end delay for a packet. Choices of τ do not affect the security of the protocol, although values that are too small may cause the Route Discovery to fail. Ariadne can choose τ adaptively, by increasing τ when a Discovery fails. In addition, the target of the Discovery could provide feedback in the ROUTE REPLY when τ was chosen too long.

Ariadne Route Discovery using **digital signatures** differs in that no Route Discovery chain element is required (Section 7.2.6). In addition, the MAC list in the REQUEST becomes a signature list, where the data used to compute the MAC is instead used to compute a signature. Rather than computing the target MAC using a Message Authentication Code, a signature is used. Finally, no key list is required in the REPLY.

Ariadne Route Discovery using **MAC**s is most efficient, but requires pairwise shared keys between all nodes. When Ariadne is used in this way, the MAC list in the REQUEST is computed using a key shared between the target and the current node, rather than using the TESLA key of the current node. The MACs are verified at the target and are not returned in the REPLY. As a result, the target MAC is not computed over the MAC list in the REQUEST. In addition, no key list is required in the REPLY.

Per-hop hashing. Authentication of data in routing messages is not sufficient, as an attacker could remove a node from the node list in a REQUEST. We use one-way hash functions to verify that no hop was omitted, and we call this approach *per-hop hashing*. To change or remove a previous hop, an attacker must either hear a REQUEST without that node listed, or must be able to invert the one-way hash function.

Ariadne Route Discovery with TESLA We now describe in detail the version of Ariadne Route Discovery using TESLA broadcast authentication. We assume that every end-to-end communicating source-destination pair of nodes **A** and **B** share the MAC keys K_{AB} and K_{BA}. We also assume that every node has a TESLA one-way key chain, and that all nodes know an authentic key of the TESLA one-way key chain of each other node (for authentication of subsequent keys, as described in Section 5.1.4). Route Discovery has two stages: the initiator floods the network with a ROUTE REQUEST, and the target returns a ROUTE REPLY. To secure the ROUTE REQUEST packet, Ariadne provides the following properties: (1) the target node can authenticate the initiator (using a MAC with a key shared between the initiator and the target); (2) the initiator can authenticate each entry of the path in the ROUTE REPLY (each intermediate node appends a MAC with its TESLA key); and (3) no intermediate node can remove a previous node in the node list in the REQUEST or REPLY (a one-way function prevents a compromised node from removing a node from the node list).

A ROUTE REQUEST packet in Ariadne contains eight fields: $\langle$ROUTE REQUEST, *initiator*, *target*, *id*, *time interval*, *hash chain*, *node list*, *MAC list*$\rangle$. The *initiator* and *target* are set to the address of the initiator and target nodes, respectively. As in DSR, the initiator sets the *id* to an identifier that it has not recently used in initiating a Route Discovery. The *time interval* is the TESLA time interval at the pessimistic expected arrival time of the REQUEST at the target, accounting for clock skew; specifically, given τ, a pessimistic transit time, the time interval could be set to any time interval for which the key is not released within the next $\tau + 2\Delta$ time. The initiator of the REQUEST then initializes the *hash chain* to $\mathrm{MAC}_{K_{SD}}(\textit{initiator, target, id, time interval})$ and the *node list* and *MAC list* to empty lists.

When any node **A** receives a ROUTE REQUEST for which it is not the target, the node checks its local table of $\langle$*initiator*, *id*$\rangle$ values from recent REQUESTs it has received, to determine if it has already seen a REQUEST from this same Route Discovery. If it has, the node discards the packet, as in DSR. The node also checks whether the *time interval* in the REQUEST is valid: that time interval must not be too far in the future, and the key corresponding to it must not have been disclosed yet. If the time interval is not valid, the node discards the packet. Otherwise, the node modifies the REQUEST by appending its own address, **A**, to the *node list* in the REQUEST, replacing the *hash chain* field with $H[\mathbf{A}, \textit{hash chain}]$, and appending a MAC of the entire REQUEST to the *MAC list*. The node uses the TESLA key K_{A_i} to compute the MAC, where i is the index for the time interval specified in the REQUEST. Finally, the node rebroadcasts the modified REQUEST, as in DSR.

When the target node receives the ROUTE REQUEST, it checks the validity of the REQUEST by determining that the keys from the time interval specified have not been disclosed yet, and that the *hash chain* field is equal to

$$H[\eta_n, H[\eta_{n-1}, H[\ldots, H[\eta_1,\\ \mathrm{MAC}_{K_{SD}}(\textit{initiator}, \textit{target}, \textit{id}, \textit{time interval})]\ \ldots]]]$$

where η_i is the node address at position i of the *node list* in the REQUEST, and where n is the number of nodes in the *node list*. If the target node determines that the REQUEST is valid, it returns a ROUTE REPLY to the initiator, containing eight fields: $\langle$ROUTE REPLY, *target*, *initiator*, *time interval*,

$$
\begin{array}{ll}
S: & h_0 = \text{MAC}_{K_{SD}}(\textsc{Route Request}, S, D, id, ti) \\
S \to *: & \langle \textsc{Route Request}, S, D, id, ti, h_0, (), () \rangle \\
\\
A: & h_1 = H[A, h_0] \\
 & M_A = \text{MAC}_{K_{A_{ti}}}(\textsc{Route Request}, S, D, id, ti, h_1, (A), ()) \\
A \to *: & \langle \textsc{Route Request}, S, D, id, ti, \underline{\boldsymbol{h_1}}, (\underline{\boldsymbol{A}}), (\underline{\boldsymbol{M_A}}) \rangle \\
\\
B: & h_2 = H[B, h_1] \\
 & M_B = \text{MAC}_{K_{B_{ti}}}(\textsc{Route Request}, S, D, id, ti, h_2, (A, B), (M_A)) \\
B \to *: & \langle \textsc{Route Request}, S, D, id, ti, \underline{\boldsymbol{h_2}}, (A, \underline{\boldsymbol{B}}), (M_A, \underline{\boldsymbol{M_B}}) \rangle \\
\\
C: & h_3 = H[C, h_2] \\
 & M_C = \text{MAC}_{K_{C_{ti}}}(\textsc{Route Request}, S, D, id, ti, h_3, (A, B, C), (M_A, M_B)) \\
C \to *: & \langle \textsc{Route Request}, S, D, id, ti, \underline{\boldsymbol{h_3}}, (A, B, \underline{\boldsymbol{C}}), (M_A, M_B, \underline{\boldsymbol{M_C}}) \rangle \\
\\
D: & M_D = \text{MAC}_{K_{DS}}(\textsc{Route Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C)) \\
D \to C: & \langle \textsc{Route Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C), \underline{\boldsymbol{M_D}}, () \rangle \\
\\
C \to B: & \langle \textsc{Route Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (\underline{\boldsymbol{K_{C_{ti}}}}) \rangle \\
\\
B \to A: & \langle \textsc{Route Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, \underline{\boldsymbol{K_{B_{ti}}}}) \rangle \\
\\
A \to S: & \langle \textsc{Route Reply}, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, K_{B_{ti}}, \underline{\boldsymbol{K_{A_{ti}}}}) \rangle
\end{array}
$$

Figure 7.1: Route Discovery example in Ariadne. The initiator node **S** is attempting to discover a route to the target node **D**. The bold underlined font indicates changed message fields, relative to the previous message of that type.

node list, *MAC list*, *target MAC*, *key list*⟩. The *target*, *initiator*, *time interval*, *node list*, and *MAC list* fields are set to the corresponding values from the ROUTE REQUEST, the *target MAC* is set to a MAC computed on the preceding fields in the REPLY with the key K_{DS}, and the *key list* is initialized to the empty list. The ROUTE REPLY is then returned to the initiator of the REQUEST along the source route obtained by reversing the sequence of hops in the *node list* of the REQUEST.

A node forwarding a ROUTE REPLY waits until it is able to disclose its key from the time interval specified; it then appends its key from that time interval to the *key list* field in the REPLY and forwards the packet according to the source route indicated in the packet. Waiting delays the return of the ROUTE REPLY but does not consume extra computational power.

When the initiator receives a ROUTE REPLY, it verifies that each key in the key list is valid, that the *target MAC* is valid, and that each MAC in the *MAC list* is valid. If all of these tests succeed, the node accepts the ROUTE REPLY; otherwise, it discards it. Figure 7.1 shows an example of Route Discovery in Ariadne.

7.2.4. Basic Ariadne Route Maintenance

Route Maintenance in Ariadne is based on DSR. A node forwarding a packet to the next hop along the source route returns a ROUTE ERROR to the original sender of the packet if it is unable to deliver the packet to the next hop after a limited number of retransmission attempts. In this section, we discuss mechanisms for securing ROUTE ERRORs, but we do not consider the case of attackers not sending ERRORs (Section 7.2.5).

To prevent unauthorized nodes from sending ERRORs, we require that an ERROR be authenticated by the sender. Each node on the return path to the source forwards the ERROR. If the authentication is delayed, for example when TESLA is used, each node that will be able to authenticate the ERROR buffers it until it can be authenticated.

When using broadcast authentication, such as TESLA, a ROUTE ERROR packet in Ariadne contains six fields: ⟨ROUTE ERROR, *sending address*, *receiving address*, *time interval*, *error MAC*, *recent TESLA key*⟩. The *sending address* is set to the address of the intermediate node encountering the error, and the *receiving address* is set to the intended next hop destination of the packet it was attempting to forward. For example, if node **B** is attempting to forward a packet to the next hop node **C**, if **B** is unable to deliver the packet to **C**, node **B** sends a ROUTE ERROR to the original sender of the packet; the the *sending address* in this example is set to **B**, and the *receiving address* is set to **C**. The *time interval* in the ROUTE ERROR is set to the TESLA time interval at the pessimistic expected arrival time of the ERROR at the destination, and the *error MAC* field is set to the MAC of the preceding fields of the ROUTE ERROR, computed using the sender of the ROUTE ERROR's TESLA key for the time interval specified in the ERROR. The *recent TESLA key* field in the ROUTE ERROR is set to the most recent TESLA key that can be disclosed for the sender of the ERROR. We use TESLA for authenticating ROUTE ERRORs so that forwarding nodes can also authenticate and process the ROUTE ERROR.

When sending a ROUTE ERROR, the destination of the packet is set to the source address of the original packet triggering the ERROR, and the ROUTE ERROR is forwarded toward this node in the same way as a normal data packet; the source route used in sending the ROUTE ERROR packet is obtained by reversing the source route from the header of the packet triggering the ERROR. Each node that is either the destination of the ERROR or forwards the ERROR searches its Route Cache for all routes it has stored that use the ⟨*sending address*, *receiving address*⟩ link indicated by the ERROR. If the node has no such routes in its Cache, it does not process the ROUTE ERROR further (other than forwarding the packet, if it is not the destination of the ERROR). Otherwise, the node checks whether the *time interval* in the ERROR is valid: that time interval must not be too far into the future, and the key corresponding to it must not have been disclosed yet; if the time interval is not valid, the node similarly does not process the ROUTE ERROR further.

If all of the tests above for the ROUTE ERROR succeed, the node checks the authentication on the ERROR, based on the sending node's TESLA key for the time interval indicated in the ERROR. To do so, the node saves the information from the ERROR in memory until it receives a disclosed TESLA key from the sender that allows this. During this time, the node continues to use the routes in its Route Cache without modification from this ERROR. If the sender stops using that route, there will be no need to complete the authentication of the ERROR. Otherwise, each subsequent packet sent along this route by this node will trigger an additional ROUTE ERROR, and once the TESLA time interval used in the first ERROR ends, the *recent TESLA key* field in the next ERROR returned will allow authentication of this first ERROR; alternatively, the node could also explicitly request the needed TESLA key from the sender once the interval ends. Once the ROUTE ERROR has been authenticated, the node removes from its Route Cache all routes using the indicated link, and also discards any saved information for other ERRORs for which, as a result of removing these routes, it then has no corresponding routes in its Route Cache.

To handle the possible memory consumption attack of needing to save information from many pending ROUTE ERRORs, the following technique is quite effective: each node keeps in memory a table containing the information from each ROUTE ERROR awaiting authentication. We manage this table such that the probability that the information from an ERROR is in the table is independent of the time that this node received that ROUTE ERROR.

$$B \to S : \qquad \langle B, C, ti, \mathrm{MAC}_{K_{B_{ti}}}(B, C, ti), K_{ti-1} \rangle$$

Figure 7.2: Route Maintenance in Ariadne. **B** finds a broken link and reports the ROUTE ERROR to the sender **S**. The ERROR is authenticated with TESLA, and includes a previous key, in case the same ERROR had been sent previously, but they key had not been disclosed yet.

When the wireless link capacity is finite, an attacker can inject only a finite number of ROUTE ERRORS within a TESLA time interval plus $2\Delta + \tau$. As a result, the probability of success for our defense against memory consumption attacks for received ROUTE ERRORS in any time interval is given by $p_s = 1 - (y/(x+y))^N$, where N is the number of ROUTE ERRORS that can be held in the node's table, x is the number of authentic ROUTE ERRORS received, and y is the number of ERRORS sent by the attacker. The maintenance of a link therefore follows a geometric distribution, and the expected number of time intervals before success is $(1 - (y/(x+y))^N)^{-1}$. For example, in a network using a 1-second TESLA time interval and an 11 Mbps wireless link, if the size of a ROUTE ERROR packet is 60 bytes, then a node with a 5000-element table receiving just one authentic ROUTE ERROR per second can successfully authenticate and process one of the authentic ROUTE ERRORS within 5.1 seconds on the average, even when an attacker is otherwise flooding the node with malicious ROUTE ERRORS. This 5.1 second recovery time represents a *worst-case* scenario, and minimal node resources are consumed while the node waits to validate one of these ROUTE REQUESTS. Figure 7.2 shows an example of Route Maintenance in Ariadne.

When digital signatures or pairwise shared keys are used, this memory consumption attack is not possible, and the authentication is more straightforward. A ROUTE ERROR need not include a *time interval* or *recent TESLA key*. Furthermore, the *error MAC* is changed to a digital signature when digital signatures are used. When pairwise shared keys are used, the *error MAC* is computed based on the key shared between the original sender of the packet and the sender of the ROUTE ERROR, rather than on the TESLA key of the sender of the ERROR.

7.2.5. Thwarting Effects of Routing Misbehavior

The protocol described so far is vulnerable to an Active-1-1 attacker that happens to be along the discovered route. In particular, we have not presented a means of determining whether intermediate nodes are in fact forwarding packets that they have been requested to forward. Watchdog and pathrater [116] attempt to solve this problem by identifying the attacking nodes and avoiding them in the routes used. Instead, we choose routes based on their prior performance in packet delivery. Introducing mechanisms that penalize specific nodes for routing misbehavior (such as is done in watchdog and pathrater) is subject to a blackmail attack (Section 5.2.1), where a sufficient number of attackers may be able to penalize a well-behaved node.

Our scheme relies on feedback about which packets were successfully delivered. The feedback can be received either through an extra end-to-end network layer message, or by exploiting properties of transport layers, such as TCP with SACK [117]; this feedback approach is somewhat similar that used in IPv6 for Neighbor Unreachability Detection [130]. Stronger properties are obtained when the routing protocol sends such feedback packets along a route equal to the reversed route of the triggering packet; otherwise, a malicious node along one route may drop the acknowledgment for a packet transmitted along a functioning route.

A node with multiple routes to a single destination can assign a fraction of packets that it originates to be sent along each route. When a substantially smaller fraction of packets sent along any particular route are successfully delivered, the node can begin sending a smaller fraction of its overall packets to that destination along that route. However, if the fraction of packets chosen to be sent along a route that appears to be misbehaving were to reach zero, a short-lived jamming attack that is now over could still prevent the future use of that route. To avoid this possible DoS attack, we choose the fraction of packets sent along such a route to be some small but nonzero amount, to allow the occasional monitoring of the route. A packet sent for this purpose can be a normal data packet, or, if all packets are secured using end-to-end encryption, a padded "probe" packet can be used.

Because DSR often returns multiple ROUTE REPLY packets in response to a Route Discovery, the presence of multiple routes to some destination in a node's Route Cache is quite common. Tsirigos and Haas [184] also discuss the use of multiple routes for increasing reliability, although they do not discuss this technique with respect to secure routing protocols.

Malicious nodes can also be avoided during Route Discovery. Each ROUTE REQUEST can include a list of nodes to avoid, and the MAC that forms the initial hash chain element (h_0) is then also computed over that list of nodes. Malicious nodes cannot add or remove nodes from this list without being detected by the target. Choosing which nodes to avoid in this way is beyond the scope of this chapter.

7.2.6. Thwarting Malicious Route Request Floods

An active attacker can attempt to degrade the performance of DSR or other on-demand routing protocols by repeatedly initiating Route Discovery. In this attack, an attacker sends ROUTE REQUEST packets, which the routing protocol floods throughout the network. In basic Ariadne (Sections 7.2.3 and 7.2.4), a ROUTE REQUEST is not authenticated until it reaches its target, thus allowing an Active-1-1 attacker to cause such network-wide floods. (An Active-0-1 can be thwarted by using a network-wide authentication key, as described in Section 7.3.2.)

To protect Ariadne from a flood of ROUTE REQUEST packets, we need a mechanism that enables nodes to instantly authenticate ROUTE REQUESTs, so nodes can filter out forged or excessive REQUEST packets. We introduce *Route Discovery chains*, a mechanism for authenticating Route Discoveries, allowing each node to rate-limit Discoveries initiated by any node.

Route Discovery chains are one-way chains generated, as in TESLA (Section 5.1.4), by choosing a random K_N, and repeatedly computing a one-way hash function H to give $K_i = H^{N-i}[K_N]$. These chains can be used in one of two ways. One approach is to release one key for each Route Discovery. Each ROUTE REQUEST from that Discovery would carry a key from this Route Discovery chain, and duplicates could be suppressed using this value. Because of the flooding nature of Route Discovery, a node that is not partitioned from the network will generally hear each chain element that is used, preventing an attacker from reusing that value in the future. An alternative approach, similar to TESLA, is to dictate a schedule at which Route Discovery chain elements can be used, and to use loosely synchronized clocks to prevent even partitioned nodes from propagating an old ROUTE REQUEST. The latter approach is computationally slightly more expensive, but it is secure against an attacker replaying an old chain element to a formerly partitioned node, causing that node to ignore REQUESTs from the spoofed source for some period of time.

7.2.7. An Optimization for Ariadne

When Ariadne is used with broadcast authentication such as TESLA, additional route caching is possible. In the basic Route Discovery mechanism described in Section 7.2.3, only the initiator of the Discovery can use the route in the REPLY, since the *target MAC* field of the REPLY can only be verified by the initiator. However, if the appropriate data is also broadcast authenticated, any node along a path returned in a REPLY can use that route to reach the target. For example, if TESLA is

Table 7.1: Parameters for Ariadne Simulations

Scenario Parameters

Number of Nodes	50
Maximum Velocity ($v_{\max}$)	20 m/s
Dimensions of Space	1500 m × 300 m
Nominal Radio Range	250 m
Source-Destination Pairs	20
Source Data Pattern (each)	4 packets/second
Application Data Payload Size	512 bytes/packet
Total Application Data Load	327 kbps
Raw Physical Link Bandwidth	2 Mbps

DSR Parameters

Initial ROUTE REQUEST Timeout	2 seconds
Maximum ROUTE REQUEST Timeout	40 seconds
Cache Size	32 routes
Cache Replacement Policy	FIFO

TESLA Parameters

TESLA Time Interval	1 second
Pessimistic End-to-End Propagation Time (τ)	0.2 seconds
Maximum Time Synchronization Error (Δ)	0.1 seconds
Hash Length (ρ)	80 bits

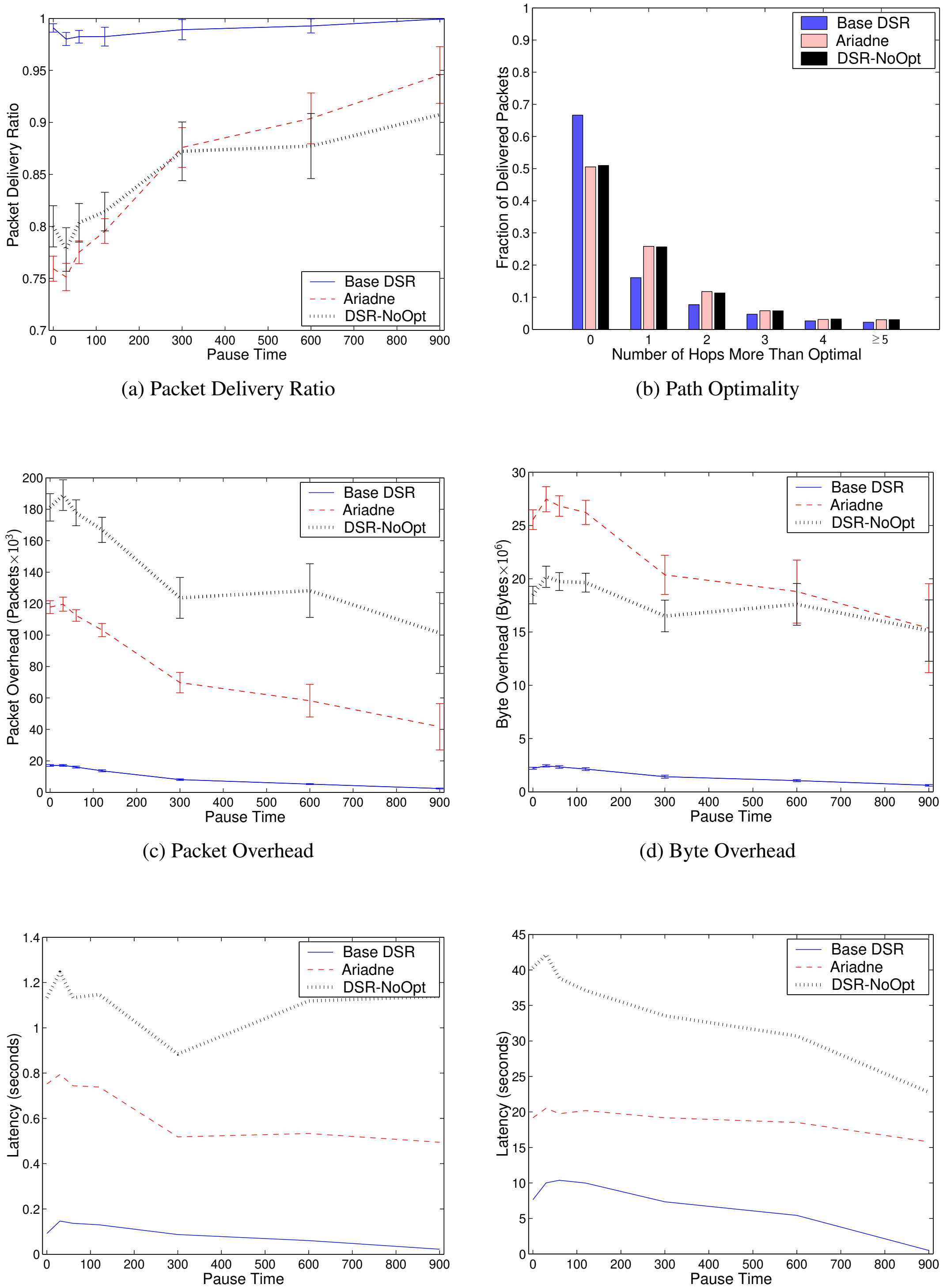
used as the broadcast authentication protocol, a *target authenticator* is placed the packet in addition to the *target MAC*, and is computed using a TESLA key that is not expected to be disclosed until Δ after the last REPLY reaches the initiator (where Δ is the maximum time difference between two nodes). That TESLA key is then disclosed, after appropriate delay, by sending it to the initiator along each path traversed by a REPLY.

7.3. Ariadne Evaluation

7.3.1. Simulation-Based Performance Evaluation

To evaluate the Ariadne without attackers, we used the *ns-2* simulator, with our mobility extensions (Section 2.3.1). The parameters used for our simulation are given in Table 7.1.

We evaluated the version of Ariadne that uses TESLA for broadcast authentication and shared keys only between communicating nodes (without the optimization described in Section 7.2.7). We modeled this version of Ariadne by modifying our *ns-2* DSR model in several ways: we increased the packet sizes to reflect the additional fields necessary for authenticating the packets, and modified the handling of Route Discovery and Maintenance for the additional authentication processing defined in Ariadne; we adjusted retransmission timeouts for ROUTE REQUESTs to compensate for the delay necessary for the disclosure of TESLA keys; and we treated routes learned from Route Discovery in an atomic fashion that did not allow the use of prefixes of routes in the Route Cache. We compare this version of Ariadne versus a recent version of DSR (Chapter 2), which we call simply "Base DSR," and with an unoptimized version of DSR, which we call "DSR-NoOpt." In DSR-NoOpt, we disabled all protocol optimizations not present in Ariadne. By comparing Ariadne with this unoptimized version of DSR, we can examine the performance impact of adding security, independent of the performance impact of the DSR optimizations removed to allow the security changes.



(a) Packet Delivery Ratio

(b) Path Optimality

(c) Packet Overhead

(d) Byte Overhead

(e) Average Latency

(f) 99.99th Percentile Latency

Figure 7.3: Performance results comparing Ariadne with the standard DSR protocol and with a version of DSR with all DSR optimizations not present in Ariadne disabled. Results are based on simulation over 60 runs, and the error bars represent the 95% confidence interval of the mean.

Our simulation parameters were identical to those described in Section 3.2, except that we generated 60 scenarios for each pause time. We computed six metrics for each simulation run: the four metrics from Section 2.3.3, byte overhead (Section 3.2), and *99.99th Percentile Latency*, computed as the 99.99th percentile of the packet delivery latency.

Figure 7.3(a) shows the Packet Delivery Ratio (PDR) for each protocol. Removing the optimizations from Base DSR to produce DSR-NoOpt reduces PDR by an average of 15.2%; adding Ariadne security further reduces PDR by just an additional 0.66% on average, and does not reduce PDR by more than an additional 4% at any pause time. Ariadne delivers fewer packets than DSR-NoOpt at higher levels of mobility for two reasons. First, since Route Discovery operates more slowly, packets are more likely to time out waiting for a ROUTE REPLY, and the route contained in a ROUTE REPLY will have a shorter lifetime. Second, because ROUTE ERRORS cannot be processed until the TESLA key used is disclosed, additional data packets continue to be sent along the broken route for on average half of the TESLA time interval after the ERROR is received.

Surprisingly, Ariadne outperforms DSR-NoOpt at lower levels of mobility. This improved performance results from the average half-second delay (one half the TESLA time interval) that Ariadne introduces between the target receiving a ROUTE REQUEST and sending a ROUTE REPLY. Specifically, when a REQUEST traverses a short-lived link, DSR-NoOpt immediately returns the REPLY, but the new route can be used for only its brief lifetime, contributing additional overhead for forwarding the REPLY and for sending and forwarding the ERROR. In Ariadne, links are tested twice: once when the REQUEST traverses the network, and once when the REPLY is sent along the reverse path. If one of these links breaks between these tests, the REPLY with this route is not received by the initiator. It is this additional route confirmation that allows Ariadne to find more stable routes than DSR-NoOpt.

Figures 7.3(c) and 7.3(d) show the packet and byte overhead, respectively. Ariadne has *consistently lower* packet overhead than DSR-NoOpt, because Ariadne tends to find more stable routes than DSR-NoOpt, reducing the number of ROUTE ERRORS that are sent. This advantage is somewhat countered by the increase in number of ROUTE ERRORS used by Ariadne: since ERROR processing is delayed, more redundant ERRORS are sent. Unfortunately, byte overhead in Ariadne is significantly worse than in either Base DSR or DSR-NoOpt, due to the authentication overhead in ROUTE REQUEST, REPLY, and ERROR packets.

Figure 7.3(b) shows Path Optimality. In Base DSR, the average number of hops along a route used by a packet is 0.6853 hops more than the minimum possible, based on the nominal wireless transmission range of 250 m per hop. In DSR-NoOpt, routes used are on average 0.2705 hops longer than in Base DSR, and in Ariadne, routes used average 0.0044 hops longer than in DSR-NoOpt. DSR-NoOpt performs slightly better than Ariadne because it initiates more Route Discoveries and thus tends to more quickly find shorter routes when they become available than does Ariadne.

Figures 7.3(e) and 7.3(f) show the average and 99.99th percentile latency for the protocols, respectively. Because of the reduced number of broken links that get used in Ariadne relative to DSR-NoOpt, Ariadne generally has better latency than DSR-NoOpt.

7.3.2. Security Analysis

In this section, we discuss how Ariadne resists attacks by certain attacker types, according to the taxonomy we present in Section 5.2.1.

Intuitively, Ariadne Route Discovery is successful when at least one of the REPLYs returned by the target is a working route. Since the target of a Route Discovery returns a route for each of its neighbors, if the first REQUEST from a particular Discovery to reach any neighbor of the target has passed through no malicious nodes, that Discovery will succeed.

To more formally characterize the security offered by Ariadne, we define a *minimum broadcast latency path* between a source and a destination to be any path that forwards a Route Discovery most quickly from the source to the destination. We call a route that only consists of uncompromised nodes an *uncompromised route*. Ariadne prevents compromised nodes from disturbing uncompromised routes. In particular, Ariadne provides two properties assuming reliable broadcast:

- If there exists an uncompromised neighbor of a destination such that the minimum latency path between the initiator of the Discovery and that neighbor is uncompromised, then an uncompromised route from the initiator to the target will be returned in a ROUTE REPLY.
- If at least one REPLY returned as a result of the first property represents a shortest route from the initiator to the target, Ariadne may route packets along one such uncompromised route.

To argue for the correctness of the first property, we note that if the minimum latency path between the initiator and a neighbor of the destination is uncompromised, then the first REQUEST to reach that neighbor comes over an uncompromised route. Since it is the first REQUEST, it will not be filtered by duplicate REQUEST detection, so it will be rebroadcast, and heard by the target. Since the target returns a REPLY for each REQUEST it receives, without performing duplicate detection, a REPLY will be returned. The second property trivially follows from the use of shortest paths and the first property.

Although it may not be possible to achieve reliable broadcast securely or efficiently, we assume that most broadcast packets are received, and hence the properties listed above generally hold.

We now consider Ariadne using our taxonomy of attacks that we present in Section 5.2.1. We list different attacker configurations in increasing strength, and discuss how Ariadne resists these attacks. Ariadne resists many more attacks, but due to space constraints, only a representative sample are discussed here.

Since Ariadne does not attempt to provide anonymous routing, passive attackers can eavesdrop on all routing traffic sent by nodes within range of those attackers. They can also perform traffic analysis on any packets sent or forwarded by nodes within range of the attackers.

When replay protection and a global MAC key are used, an Active-0-x attacker (for $x \geq 1$) can at most perform wormhole and rushing attacks. Packet leashes can prevent these attacks (Chapter 9).

An Active-1-1 attacker may attempt the following attacks:

- Create a gray hole or black hole by removing nodes in a ROUTE REQUEST; however, the per-hop hash mechanism in each REQUEST prevents such tampering. An attacker may fabricate nodes to insert in the accumulated route list of a REQUEST packet, such fabricated nodes would not have known keys at the source, and the REPLY would thus not be authenticated. If the attacker tries to replace the MAC and keys in the reply, such tampering will be detected as a result of the *target MAC* field in the REPLY.
- Create routing loops. Intuitively, the use of source routes prevents loops, since a packet passing through only legitimate nodes will not be forwarded into a loop. An attacker can create a routing loop by modifying the source route each time around the loop; this behavior, however, is no worse than if the attacker were to source packets with period equal to the propagation time around the loop. In particular, if there are n nodes in the routing loop, and a single packet is forwarded around the loop m times, the attacker participates in m forwards, and the total expended effort is mn forwards. Had the attacker instead sourced m packets along n-hop routes, the total attacker effort is m transmissions, and the total network effort is mn forwards, an identical result.

- Flood network with many ROUTE REQUESTs. Since the source address of each REQUEST is authenticated, and since each new Route Discovery needs to carry a new one-way Route Discovery chain value, the compromised node can only produce ROUTE REQUESTs with its own source address. An upper bound on the sending rate can be enforced either by rate limiting of REQUESTs at each node or synchronizing Route Discovery chain elements with time (Section 7.2.6).

- Perform a rushing attack (Section 5.2.2). Rushing attacks can be probabilistically prevented by slightly modifying the Route Discovery protocol [80].

Multiple attackers that have compromised one node (Active-1-x, for $x > 1$) may attempt to construct a wormhole, but append the address and key of the compromised node in each REQUEST forwarded across this wormhole. Packet leashes alone cannot prevent this attack, but packet leashes and GPS can be used in conjunction to ensure that an Active-1-x wormhole attack can be no worse than an Active-1-1 attacker positioned correctly. In particular, if each node forwarding a ROUTE REQUEST includes its alleged GPS coordinates in that REQUEST, then a node can detect if it should be reachable from the previous hop, and if the hop before the previous hop should be able to reach the previous hop. If both of these checks succeed, then the attacker could have placed the compromised node at the position it specified in the packet, and that node would have been able to hear the original REQUEST, append its address, and forward it to the next hop.

Multiple attackers that know all the keys of multiple nodes (an Active-y-x attacker configuration, where $1 < y \leq x$) may perform the following attacks:

- Lengthen the route in the REQUEST by adding other compromised nodes to the route. If the source finds a shorter route, it will likely prefer that route, so the protocol behaves as if the attacker were not there.

- Attempt to force the initiator to repeatedly initiate Route Discoveries. Suppose an Active-y-x attacker had the keys of multiple compromised nodes, and that one such attacker were on the shortest path from the source to the destination. When the attacker receives its first ROUTE REQUEST packet as part of some Discovery, it adds its address and MAC, as normal, but also adds the address of another node it has compromised. When data packets are sent along that route, the attacker replies with a ROUTE ERROR from its first hop to its second hop. In subsequent Route Discoveries, the attacker can use different addresses for the additional address. Since other routes may have been returned as part of any of these Route Discoveries, this attack is not guaranteed to be successful.

  To prevent such starvation, the initiator may include data in the ROUTE REQUEST. To be part of the path, the attacker must forward routing messages, so the initiator can send data to the target. If the attacker alters the data in the ROUTE REQUEST, the destination will detect the alteration (using the shared key and a MAC on the data) and reject that route.

A set of attackers that control a vertex cut of the network (an Active-VC attacker) may perform the following additional attacks:

- Make nodes on one side of the vertex cut believe that any node on the other side is attempting to flood the network. By holding and not propagating ROUTE REQUESTs from a certain node for some time, then initiating many Route Discoveries with the chain values from the old Discoveries, an Active-VC attacker can make that node appear to be flooding the network. When the use of individual elements of a Route Discovery chain are time-synchronized, this attack simply causes the REQUESTs associated with the stale chain elements to be discarded.

- Only forward ROUTE REQUEST and ROUTE REPLY packets. A sender is then unable to successfully deliver packets. This attack is only marginally different from not participating in the protocol at all, differing only in that the sender and some intermediate nodes continue to spend power to send packets, but none of those packets are successfully received.

7.4. Related Work

Prior to the publication of Ariadne [79], several researchers proposed secure routing protocols. Perlman [143] proposed *flooding NPBR*, an on-demand protocol designed for wired networks that floods each packet through the network. Flooding NPBR allocates a fraction of the bandwidth along each link to each node, and uses digital signatures to authenticate all packets. Unfortunately, this protocol has high overhead in terms of the computational resources necessary for digital signature verification and in terms of its bandwidth requirements. Furthermore, estimating and guaranteeing available bandwidth in a wireless environment is difficult [97].

Other wired network protocols have secured periodic routing protocols with *asymmetric* cryptography, such as Kent et al [96], Perlman's *link-state NPBR*, Kumar's secure link-state protocol [102], and Smith et al [176, 177]. However, nodes in an ad hoc network may not have sufficient resources to verify an asymmetric signature; in particular, an attacker can trivially flood a victim with packets containing invalid signatures, but verification can be prohibitively expensive for the victim. In addition, these protocols may suffer in some scenarios because periodic protocols may not be able to cope with high rates of mobility in an ad hoc network. Kumar also discusses threats to both distance-vector protocols and link-state protocols, and describes techniques for securing distance-vector protocols. However, these techniques are vulnerable to the compromise of a single node.

Zhou and Haas [192], Zapata [190], and Dahill et al [170] propose the use of asymmetric cryptography to secure on-demand ad hoc network routing protocols. However, as above, when the nodes in an ad hoc network are generally unable to verify asymmetric signatures quickly enough, or when network bandwidth is insufficient, these protocols may not be suitable.

Cheung [33], Hauser et al [65], and Zhang [191] describe *symmetric*-key approaches to the authentication of link-state updates, but they do not discuss mechanisms for detecting the status of these links. In wired networks, a common technique for authenticating HELLO packets is to verify that the the incoming network interface is the expected interface and that the IP TTL of the packet is 255. In a wireless ad hoc network, this technique cannot be used. Furthermore, these protocols assume the use of periodic routing protocols, which are not always suitable in ad hoc networks. Cheung [33] uses cryptographic mechanisms similar to those used in Ariadne with TESLA, but optimistically integrates routing data before it is authenticated, adversely affecting security.

A number of other researchers have also proposed the use of symmetric schemes for authenticating routing control packets. Heffernan [69] proposes a mechanism requiring shared keys between all communicating routers. This scheme requires $O(n^2)$ keys in an n-node network, and is vulnerable to single-node compromise. Perrig et al [148] use symmetric primitives to secure routing between nodes and a trusted base station. Basagni et al [10] use a network-wide symmetric key to secure routing communication, which is vulnerable to a single node compromise, although they specify the use of secure hardware to limit the damage that can be done by a compromised node. Papadimitratos and Haas [136] present work that secures against non-colluding adversaries, and they do not authenticate intermediate nodes that forward ROUTE REQUESTS, and thus do not handle authorization. Yi et al [189] discuss authorization issues. Our previous work, SEAD (Chapter 6), uses hash chains to authenticate routing updates sent by a distance-vector protocol; however, that approach builds on a periodic protocol, and such protocols tend to have higher overhead than on-demand protocols and may not be suitable in highly mobile networks.

Yi, Naldurg, and Kravets [189] present the notion of security requirements for paths carrying certain traffic, and briefly discuss mechanisms for securing routing, such as network-wide encryption of ROUTE REQUESTs and use of digital signatures.

Routing protocol intrusion detection has also been studied as a mechanism for detecting misbehaving routers [23, 34, 116]. Cheung and Levitt [34] and Bradley et al [23] propose intrusion detection techniques for detecting and identifying routers that send bogus routing update messages. In this chapter, we attempt to authenticate packets before processing them, instead of relying on the delayed reaction of an intrusion detection system. Marti et al [116] consider the problem of detecting intermediate nodes that do not forward packets. Our approach differs in that we choose routes that we have found to work, rather than relying on watching as nodes forward traffic from arbitrary sources.

7.5. Chapter Summary

This chapter has presented the design and evaluation of Ariadne, a new ad hoc network routing protocol that provides security against one compromised node and arbitrary active attackers, and relies only on efficient *symmetric* cryptography. Ariadne operates on-demand, dynamically discovering routes between nodes only as needed; the design is based on the basic operation of the DSR protocol. Rather than generously applying cryptography to an existing protocol to achieve security, however, we carefully re-designed each protocol message and its processing. The security mechanisms we designed are highly efficient and general, so that they should be applicable to securing a wide variety of routing protocols.

Because we did not secure the optimizations of DSR in Ariadne, the resulting protocol is less efficient than the highly optimized version of DSR that runs in a trusted environment. However, we also compared Ariadne to a version of DSR in which we disabled all protocol optimizations not present in Ariadne, allowing us to evaluate and analyze the effect of the optimizations and the security separately. The byte overhead of Ariadne was 26.19% higher than for unoptimized DSR, due to the overhead of the authentication information in Ariadne's routing packets. As explained in our results, however, Ariadne actually performs *better* on some metrics (e.g., 41.7% lower packet overhead) than for unoptimized DSR, and about the same on all other metrics, even though Ariadne must bear the added costs for security not present in unoptimized DSR.

We found that source-routing facilitates securing ad hoc network routing protocols. Source routing empowers the sender to circumvent potentially malicious nodes, and enables the sender to authenticate every node in a ROUTE REPLY. Such fine-grained path control is absent in most distance-vector routing protocols, which makes such protocols more challenging to fully secure.

Chapter 8

Further Efficient Mechanisms for Securing Routing Protocols

Routing protocols are difficult to efficiently secure. An attacker may, for example, attempt to inject forged routing messages into the system, or may attempt to modify legitimate routing messages sent by other nodes. An attacker may also attempt to exploit mechanisms in the routing protocol, such as those intended to quickly spread new routing information, to instead consume large amounts of network and router resources. Even the addition of cryptographic mechanisms to a routing protocol may make the protocol vulnerable to such attacks, since traditional security mechanisms are generally expensive in terms of CPU time; an attacker may be able to cripple several routers simply by flooding each router with large numbers of randomly generated, forged routing messages, which then must be authenticated and rejected by the router, leading to a denial of service by consuming all router CPU time.

Current routing protocols in use in the Internet today, such as the Border Gateway Protocol (BGP) [158] or the Routing Information Protocol (RIP) [111], were originally designed to operate in a trusted environment, assuming no malicious nodes. However, with the growing importance and usage of the Internet, an increasing number of corporations and public services are becoming dependent on the correct functioning of the Internet, and routing protocol security has become a significant issue. The command and control of critical infrastructures (such as the control of the power grid) and the emerging use of the Internet to carry voice traffic are two examples of this trend. The importance of securing Internet routing has also been illustrated by recent BGP misconfigurations [109], in which non-malicious BGP speakers have been able to disrupt Internet routing as a result of incorrect configuration.

Several researchers have proposed secure network routing protocols, but most have used standard digital signatures to authenticate routing update messages [96, 101, 102, 143, 174, 176, 177]. Similarly, in the area of secure multihop wireless *ad hoc network* routing, most researchers use standard digital signatures to authenticate routing messages [170, 190, 192]. Unfortunately, generation and verification of digital signatures is relatively inefficient, and it is thus challenging to design a scalable, efficient, and viable secure routing protocol based on such *asymmetric* cryptography. In particular, the use of asymmetric cryptography exposes

Symmetric cryptographic primitives are much more efficient than asymmetric primitives, but so far, few security mechanisms based on symmetric cryptography have been designed for the requirements of routing protocols. We now discuss the exceptions of which we are aware.

Three mechanisms based on symmetric cryptography have been proposed to secure link state routing updates. Cheung [33] presents an efficient time-based authentication protocol to authenticate link state routing updates. The proposed authentication is optimistic, though, and routers use the routing update before it is authenticated. Hauser, Przygienda, and Tsudik [65] propose to use efficient one-way hash chains to authenticate link state routing updates. Zhang [191] subsequently improves their mechanism and presents a chained Merkle-Winternitz one-time signature [123, 124], similar to our basic MW chains scheme that we present in Section 8.2.4, although our technique is more space-efficient.

Heffernan [69] assumes that neighboring routers share secret keys, and routers use MD5 to authenticate each other's messages. This approach allows BGP to protect itself against the introduction of spoofed TCP segments into the connection stream (TCP resets are of particular concern).

Basagni et al. [10] present a protocol with a network-wide shared key for use in routing, purely based on symmetric cryptography. However, their approach assumes secure hardware to protect the key.

We have previously developed two efficient secure routing protocols based on symmetric cryptography. Our SEAD protocol (Chapter 6) introduces a new efficient mechanism, based on one-way hash chains, to secure distance vector routing updates. Our Ariadne routing protocol (Chapter 7) is a secure on-demand ad hoc network routing protocol using source routing.

In this chapter, we present four new security mechanisms based on efficient *symmetric* cryptographic techniques, that can be applied to strengthen current distance vector and path vector routing protocols or can be incorporated into the design of new secure routing protocols. For securing distance vector protocols, our *hash tree chain* mechanism forces a router to increase the distance (metric) when forwarding a routing table entry. To provide authentication of a received routing update in bounded time, we present a new mechanism, similar to hash chains, that we call *tree-authenticated one-way chains*. For cases in which the maximum metric is large, we present *skipchains*, which provides more efficient initial computation cost and more efficient element verification; this mechanism is based on a new cryptographic mechanism, called MW-chains, which we also present. For securing path vector protocols, our *cumulative authentication* mechanism authenticates the list of routers on the path in a routing update, preventing removal or reordering of the router addresses in the list; the mechanism uses only a single authenticator in the routing update rather than one per router address. We also present a simple mechanism to securely switch one-way chains, by authenticating the next one-way chain using the previous one.

8.1. Assumptions

In designing our mechanisms to build secure routing protocols, we make the following assumptions on node capability and key setup.

8.1.1. Node Assumptions

The computational resources of network nodes vary greatly, from a high-end Internet backbone router to a tiny ad hoc network node. To make our results as general as possible, we design our mechanisms to be extremely lightweight and efficient. This allows our mechanisms to be used on resource-constrained ad hoc network nodes and enables large Internet routers to scale to high bandwidth links. In particular, we design our mechanisms from purely *symmetric* cryptographic functions, such as message authentication codes (MACs) or cryptographic hash functions. In contrast, mechanisms based on asymmetric cryptography are often 3 to 4 orders of magnitude slower than hash functions.

Most previous work on secure Internet and ad hoc network routing relies on *asymmetric* cryptography [96, 102, 143, 174, 176, 177]. However, computing such signatures on resource-constrained nodes is expensive, and such high overhead computations may hinder the routing protocol's scalability to large networks.

We do not assume trusted hardware such as tamperproof modules. Secure routing with trusted hardware is much simpler, since node compromise is assumed to be impossible.

8.1.2. Security Assumptions and Key Setup

We assume a mechanism that enables the secure and authentic distribution of keying material. Most of our mechanisms require the distribution of authentic public values to enable authentication of subsequent values. However, the cumulative authentication mechanism assumes pairwise shared secret keys, or authentic public keys of other nodes if a broadcast authentication system such as TESLA [145, 146] is used.

Digital signatures and a public-key infrastructure may be used to set up the authenticated public values, as well as to establish pairwise shared secret keys if used in conjunction with a key agreement protocol such as Diffie-Hellman [45].

We assume protection against the immediate replay of routing packets. In a wired network or a static wireless network, each router can be configured with a list of possible neighbors; a router that receives an update from a node not on this list can silently discard that update. In mobile wireless networks, such as ad hoc networks, we have developed *packet leashes* which restrict such immediate replay (Chapter 9). In this chapter, we assume that one of these mechanisms is used.

8.2. Mechanisms for Securing Distance Vector Protocols

In this section, we discuss the remaining research challenges in the area of securing distance vector routing. We then present new mechanisms that address these challenges in the context of SEAD (Chapter 6).

The utility of the mechanisms we present is not limited to routing protocols. In particular, the *skipchains* mechanism we present in Section 8.2.5 allows highly efficient generation and verification of elements in long hash chains, giving a constant factor speedup in both generation and verification. Skipchains are thus particularly useful for protocols that use long one-way hash chains, such as TESLA [145, 146] or BiBa [144].

8.2.1. Remaining Challenges in Securing Distance Vector Routing

SEAD is a recent secure distance vector routing protocol we designed, that is particularly efficient because it uses one-way hash chains and no asymmetric cryptography. SEAD is described in Chapter 6. We discuss the remaining research challenges.

Although SEAD does prevent a number of attacks, some attacks and shortcomings remain:

- SEAD does not prevent *same-distance fraud*: that is, a node receiving an advertisement for sequence number s and distance (metric) d can re-advertise the same sequence number s and distance d. Section 8.2.2 presents an approach that prevents this same-distance fraud.

- Another drawback of SEAD is that an attacker can force a victim node to verify a hash chain as long as $O(ks)$, where k is the maximum number of hops and s is the maximum number of sequence numbers represented by a hash chain. Section 8.2.3 describes the tree-authenticated one-way chains mechanism, which bounds this effort by $O(k + \lg s)$. The same scheme prevents the sequence number rushing attack, which we present in Section 8.2.3.

- The overhead to verify authentication values can be large if a node has missed several routing updates. An attacker can exploit this overhead to perform a denial-of-service attack by sending bogus routing updates, forcing a node to spend considerable effort verifying the authenticity. In Section 8.2.4, we introduce a novel authentication scheme that is a hybrid between a one-way chain and a one-time signature which we call an MW-chain. Based on the MW-chain, we introduce in Section 8.2.5 a one-way chain that is very efficient to verify in case of missed routing updates. In a network with maximum diameter k, this approach reduces the verification overhead to $O(c\sqrt[c]{k} + \lg s)$ for arbitrary positive integers c. Finally, we reduce the overhead of setting up a single hash chain from $O(ks)$ to $O(s)$.

We now discuss mechanisms that can solve these remaining challenges. Our mechanisms can be generalized to secure many other distance vector protocols.

8.2.2. Hash Tree Chains for Preventing Same-Distance Fraud

We present an alternative called *hash tree chains* to one-way hash chains for authenticating the distance metric in distance vector protocols, to prevent the same-distance fraud attack introduced above. Our new mechanism forces a node to increase the distance to the destination when sending a routing update. As noted in Section 8.1, we use packet leashes (Chapter 9) to prevent an adversary from replaying a routing update in wireless networks, so that the adversary would be a "stealth node" on that route. The packet leash also provides hop-by-hop authentication, preventing an adversary from impersonating another node.

To prevent same-distance fraud, we need to prevent an attacker from replaying the same hash value (thus without increasing the metric) but replacing the node id with the attacker's node id. We construct a special one-way chain, which we call a hash tree chain, where each element of the chain encodes the node id, thus forcing a node to increase the distance metric if it wants to encode its own id. Each step in this one-way chain contains a collection of values, one or more of which are used to authenticate any particular node. These values are authenticated using a Merkle tree, and the root of that Merkle tree is used to generate the collection of values in the next step. This approach is similar to that used in the HORS signature scheme [160]. These values are authenticated using a Merkle tree, and the root of that Merkle tree is used to generate the collection of values in the next step.

A hash tree chain is a hybrid between a hash tree and a one-way chain. The one-way chain property is used in the same way as in SEAD (to enforce that nodes cannot decrease the distance metric), and the hash tree property is used to authenticate the node id. We construct the hash tree between each pair v_{i-1}, v_i of one-way chain values as follows. From the value v_i, we derive a set of values $b_0, \ldots, b_n$, using a one-way hash function H as $b_j = H[v_i \mid\mid j]$, for each j. We then build a hash tree above those values as described in Section 5.1.2. The root of the tree becomes the previous value of the one-way chain $v_{i-1} = b_{0n}$. Figure 8.1 shows an example. The node with the id 1 forwards the shaded values b'_0, b_1, and b_{23} to the neighboring nodes, which can compute the one-way hash tree chain forward to verify the authenticity of values b'_0, b_1, and b_{23}, and use the value b_{03} to sign their own id when forwarding the route update, thus automatically increasing the distance metric.

We now present two examples of how the hash tree chain can be used: when a single value corresponds to a node, and when a γ-tuple of values corresponds to a node. For notational and analytic convenience, we describe hash tree chains for which the number of values between each hash chain value is a power of two.

In a small network, each value b_j can correspond to a single node; since no two nodes share a single value, an attacker has no way to derive its value from the advertisements of neighboring

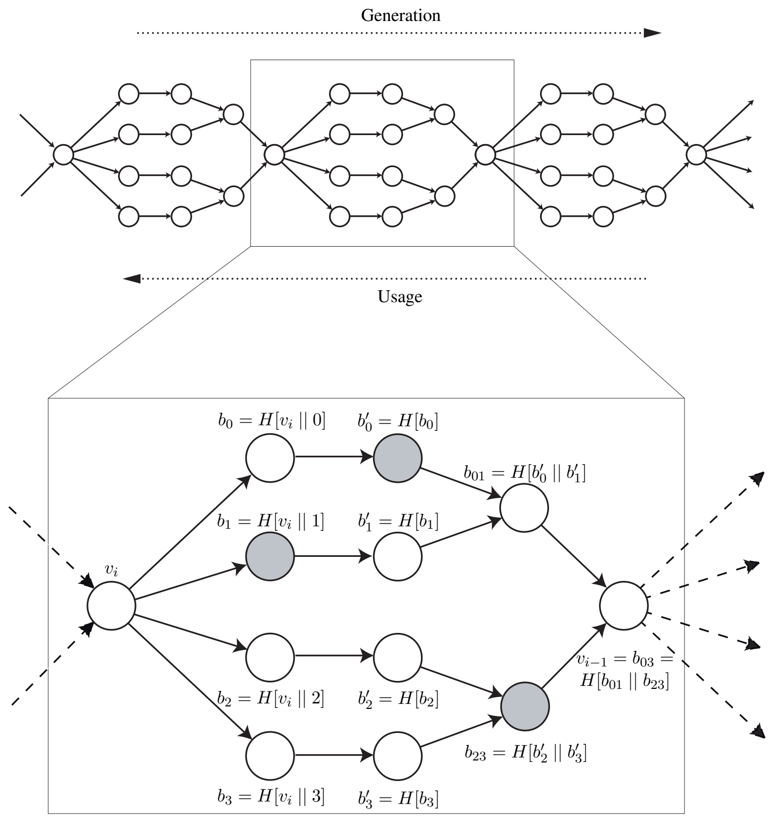


Figure 8.1: Authenticating one distance metric within a sequence of a hash tree chain. In this example, each element b_i stands for one router, so this hash tree chain supports 4 routers.

nodes, and hence it must follow the hash tree chain to the next step in order to provide a valid authenticator.

In larger networks, with n nodes, the $O(n)$ overhead of generating each step of the chain may be too great; as a result, we authenticate each node with a γ-tuple of values. Although two nodes share the same γ-tuple of values, an attacker could learn each of its γ values from different neighbors that advertise the same metric, and could then forge an advertisement without increasing the metric. We show that an attacker's probability of success may be sufficiently small. We also change the encoding of a node id for each update, so that an attacker in a static network cannot continue to forge updates once it finds an appropriate set of values from its neighbors. Consider a hash tree chain with 2^m values in each step (and thus a hash tree of height $m+1$). For example, if each node has a unique node id between 0 and $\binom{2^m}{\gamma} - 1$, then the γ-tuple encodes $x = (\text{node id} + H[\text{sequence number}])$ $\bmod \binom{2^m}{\gamma}$, such that the γ-tuple changes for each sequence number.

We now analyze the security of hash tree chains as the probability that a malicious node can forge an advertisement based on the advertisements from its neighbors. Clearly, if each value corresponds to a single node id, no forgery is possible. We now consider the case in which a pair of values (i.e., $\gamma = 2$) represents each node. For our analysis, we consider a hash tree chain with 2^m values at each step, used in a network with $n = \binom{2^m}{2}$ nodes. We compute the probability that an attacker can claim the same metric after it has heard the same metric advertised from q other nodes. For each of the two values the attacker must produce, there are $2^m - 2$ other nodes that have that particular value. It follows that the attacker has $\binom{(n-1)-(2^m-2)}{q} = \binom{n-2^m+1}{q}$ ways of failing to get a predetermined one of the two values. We now compute the probability that the attacker is unable to obtain either value. Since the set of nodes from which an attacker can receive either value are disjoint, there are $2(2^m - 2)$ nodes that have one of those two values. As a result, the attacker has $\binom{(n-1)-2(2^m-2)}{q} = \binom{n-2^{m+1}+3}{q}$ ways of failing to get either of the two values. Applying the inclusion-exclusion principle, we now compute the number of ways the attacker can fail to obtain both values it needs: $2\binom{n-2^m+1}{q} - \binom{n-2^{m+1}+3}{q}$, of $\binom{n-1}{q}$ possible distributions. The probability of successful defense, then, is

$$\frac{2\binom{n-2^m+1}{q} - \binom{n-2^{m+1}+3}{q}}{\binom{n-1}{q}} \quad .$$

For example, when $m = 8$, then $n = 32640$, and an attacker that hears $q = 3$ advertisements has a 0.000361 (3.61×10^{-4}, or 1.49×2^{-12}) probability of forging a valid authenticator from the three advertisements, without increasing the distance to the destination. In other words, an attacker can decrease its advertised metric by 0.00036 in expectation, or on average once every 2752 rounds. To improve security and reduce route oscillations, we can also require that a node advertise a particular metric for several consecutive sequence numbers before we choose that route. For example, if we require a route to be advertised three consecutive times before we accept it, and if routing updates are sent (and accepted if valid) once per second, then the attacker can successfully send a forged routing update on average only once in over 660 years, given the parameters of n, m, and q above.

To generalize our analysis, we consider the security of the hash tree chain scheme, where a node corresponds to a set of β values. First, we consider the number of ways that an attacker can fail to obtain a specific set of γ different values. There are $\binom{2^m-\gamma}{\beta}$ nodes that do not help the attacker, so there are a total of $\binom{\binom{2^m-\gamma}{\beta}}{q}$ ways to pick q nodes that do not help the attacker.

Let A_i be the set of combinations of nodes that do not include value b_i needed by the attacker. The attacker, then, has $|\cup_{i=1}^{\gamma} A_i|$ ways to fail. We now apply the inclusion-exclusion principle:

$$\begin{aligned}
\left|\bigcup_{i=1}^{\gamma} A_i\right| &= \sum_i |A_i| - \sum_{i_1,i_2} |A_{i_1} \cap A_{i_2}| + \cdots + (-1)^{\gamma+1} \left|\bigcap_{i=1}^{\gamma} A_i\right| \\
&= \sum_{i=1}^{\gamma} (-1)^{i+1} \binom{\gamma}{i} \binom{\binom{2^m-i}{\gamma}}{q}
\end{aligned}$$

Then the probability of a successful defense is

$$\frac{\sum_{i=1}^{\gamma} (-1)^{i+1} \binom{\gamma}{i} \binom{\binom{2^m-i}{\gamma}}{q}}{\binom{\binom{2^m}{\gamma}-1}{q}}$$

We can now use this to analyze variants of the scheme described earlier. In particular, we look for $n > 32000$.

When $m = 6$, $\gamma = 3$. (This represents a four-fold reduction in computation, in exchange for a 17% increase in overhead). Using $q = 3$ as before, an attacker has a 1.675×10^{-3} probability of success; when three consecutive advertisements are required for the same metric before a routing change is made, the attacker succeeds once every 6.74 years.

When $m = 5$, $\gamma = 4$. (This represents a eight-fold reduction in computation, in exchange for a 33% increase in overhead). Using $q = 3$ as before, an attacker has a 8.023×10^{-3} probability of success; when four consecutive advertisements are required for the same metric before a routing change is made, the attacker succeeds once every 7.65 years.

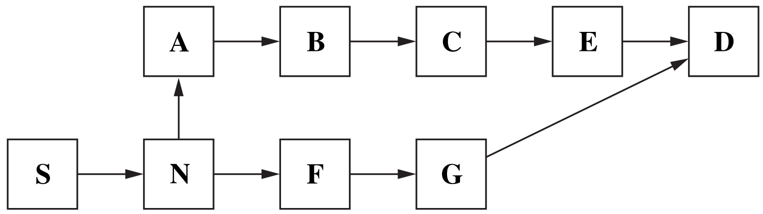


Figure 8.2: A sample network to demonstrate the sequence number rushing attack.

8.2.3. Tree-Authenticated One-Way Chains for Preventing the Sequence Number Rushing Attack

In protocols such as SEAD, a node that has missed a number of sequence numbers may need to perform a large number of hash operations to bring its chain up-to-date. This creates a potential denial-of-service vulnerability: if an attacker knows that a victim missed several recent updates for a destination, the attacker can flood the victim with updates containing recent sequence numbers but bogus authenticators; the victim must then perform many hash operations for each update received in an attempt to verify each update. Alternatively, the attacker can fabricate an update with sequence numbers far in the future, thus requiring each node receiving such an update to perform a large number of hash operations to determine that the update is bogus, although this attack can be somewhat mitigated using loose time synchronization, and rate limiting the use of new sequence numbers.

Another attack is the *sequence number rushing attack*. We explain this attack with an example. Consider the case in which a malicious node A tries to attract traffic flowing from a source S to a destination D through S's neighbor N. Figure 8.2 shows the network setup. Let the attacker A be 4 hops from D, and N be 3 hops from D. If A hears new routing updates from D before they reach N, A can rush the routing update to N. If we use the policy that a node always uses the routing update with the most recent sequence number, N will forward traffic from S to D through A until it hears the routing update with the new sequence number from F which contains a shorter route. To remedy this rushing attack, we adapt a *delayed route update adoption policy*: always use the shortest route from the previous sequence number. For example, when node N hears the first routing update with sequence number i for destination D, it will use the shortest update of sequence number $i-1$. Unfortunately, this approach is still vulnerable to an attack in which A sends two routing updates to N after it hears the update for sequence i: it forges an update with distance 0 of sequence number $i-1$, followed by an update of distance 3 for sequence number i. The tree-authenticated one-way chain mechanism we present in this section prevents A from forging low distance metrics for previous route updates. Together with the delayed route update adoption policy, we can prevent the sequence number rushing attack.

We describe here our efficient tree-authenticated one-way chain mechanism, which has two properties in addition to those of the hash chain in SEAD: first, it bounds the effort to verify an update; and second, it prevents a node with fresh sequence number information from fabricating lower metric authenticators for old sequence numbers.

In our new scheme, we use a new hash chain for each sequence number. A node using this scheme generates a random hash chain root $h_{0,s}$ for each sequence number s, for example by using a PRF $\mathcal{F}$ and a secret master key $\mathcal{X}$ to derive $h_{0,s} = \mathcal{F}(\mathcal{X}, s)$. Given the authentic anchor of this hash chain $h_{k,s} = H^k[h_{0,s}]$ (where k is the maximum metric), any node can authenticate $h_{m,s}$, which is the authenticator for sequence number s and metric m.

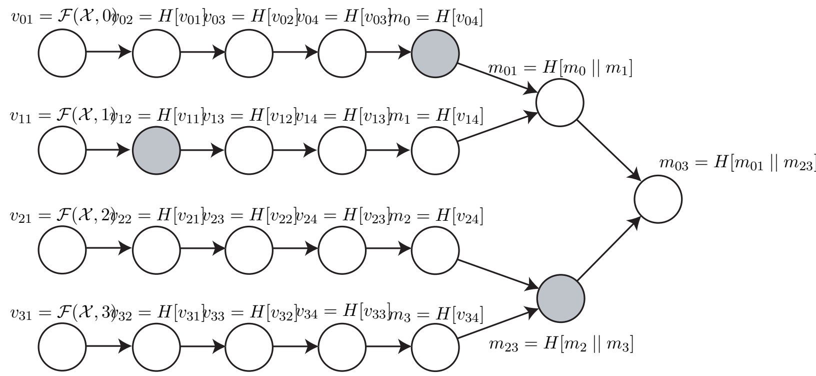


Figure 8.3: Example tree-authenticated one-way chain construction for authenticating a sequence of one-way chains. The instance in this figure allows 4 sequence numbers to be authenticated, and metrics up to 3. The shaded values represent sequence number 1 metric 1.

To allow nodes to authenticate these anchors $h_{k,s}$, each node builds a hash tree, using the hash chain anchors as leaves (Section 5.1.2). When a node sends an update with a new sequence number s, it includes the root of the hash chain $h_{0,s}$, the anchor of the hash chain $h_{k,s}$, and the path to the root of the hash tree. To authenticate any update, the node verifies the anchor by following the path to the root of the hash tree. It then verifies the hash value $h_{m,s}$ by verifying that $h_{k,s} = H^{k-m}[h_{m,s}]$. Since the maximum hash chain length is k and the anchor verification requires $O(\log(s))$ effort, where s is the number of sequence numbers represented by any root, the computation required to verify any update is bounded by $k + \log(s)$.

8.2.4. The MW-Chains Mechanism

In this section, we present a new cryptographic mechanism, which we use in the next section to improve the efficiency of secure network routing and to prevent a class of denial-of-service attacks. This mechanism is an extension to the Merkle-Winternitz one-time signature construction [123]. That construction was subsequently used and extended by Even, Goldreich, and Micali [49], and by Rohatgi [164]. Our extension to this signature construction, which we call a one-way Merkle-Winternitz chain, or MW-chain, provides instant authentication and low storage overhead. This one-way chain contains a list of values, called *heads*, and between any two heads are a set of *signature branches* and a set of *checksum branches*. To achieve low storage overhead, we derive these branches from a single head using a one-way hash function H.

The most basic way to construct an MW-chain is with one signature branch and one checksum branch between each head. Assuming we want one set of branches to sign up to N values, we choose the length of the signature branch and the checksum branch to be N; that is, we choose $\ell_1 = \ell'_1 = N$. A random value that is ρ bits long is chosen as the first head value v_n. Next, the signature and checksum branches are computed using

$$\begin{aligned} s_{1,\ell_1} &= H[v_i \mid\mid \text{“}s\text{”} \mid\mid 1] \\ s_{1,x-1} &= H[s_{1,x}] \\ c_{1,\ell'_1} &= H[v_i \mid\mid \text{“}c\text{”} \mid\mid 1] \\ c_{1,x-1} &= H[c_{1,x}] \end{aligned}$$

for $2 \leq x \leq \ell_1$. Finally, the next head value is $v_{n-1} = H[s_{1,1} \mid\mid c_{1,1}]$. The signature of a value n using this MW-chain is the ordered set $\{s_{1,n}, c_{1,N-n}\}$. An attacker can produce $s_{1,j}$, for $j < n$, but

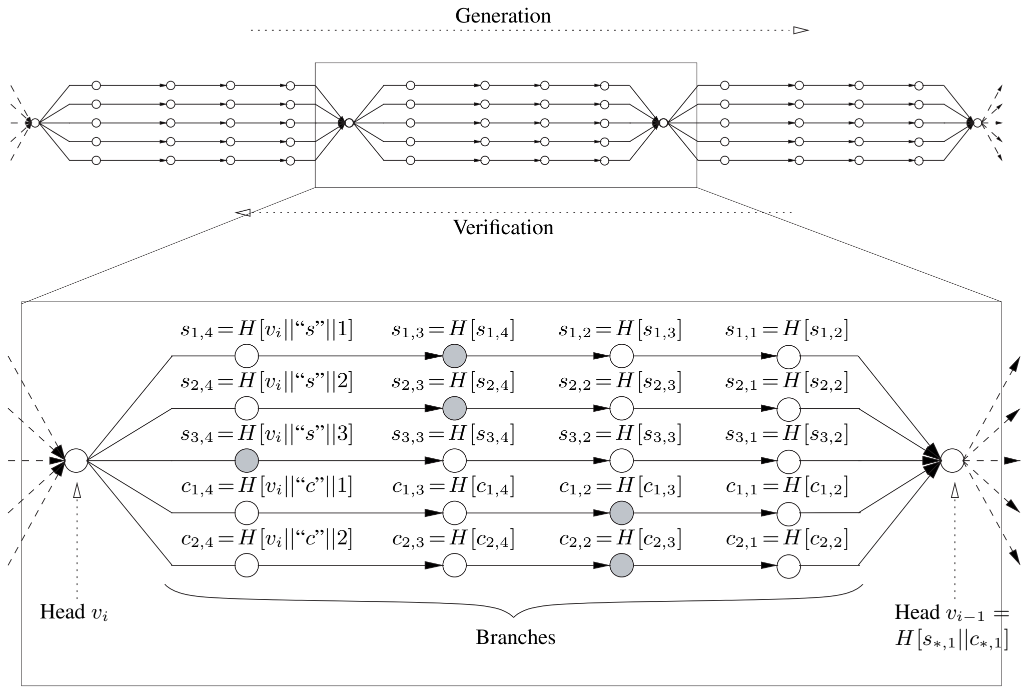


Figure 8.4: An example MW-chain being used to sign the value 58

then cannot produce $c_{1,N-j}$. Similarly, an attacker can produce $c_{1,N-j}$ for $j > n$, but then cannot produce $s_{1,j}$.

More generally, an MW-chain can have m signature branches and m' checksum branches. We call the lengths of the signature branches $\ell_1, \ell_2, \ldots, \ell_m$ and the lengths of the checksum branches $\ell'_1, \ell'_2, \ldots, \ell'_{m'}$. The signature for some value n is the ordered set

$$\{s_{1,n_1}, s_{2,n_2}, \ldots, s_{m,n_m}, c_{1,n'_1}, c_{2,n'_2}, \ldots, c_{m',n'_{m'}}\}$$

where $n_i = \left(\left\lfloor \frac{n}{\prod_{j=1}^{i} \ell_j} \right\rfloor \bmod \ell_i\right) + 1$, and $n'_i = \left(\left\lfloor \frac{\sum_{j=1}^{m} \ell_j - n_j - 1}{\prod_{j=1}^{i} \ell'_j} \right\rfloor \bmod \ell'_i\right) + 1$.

For example, Figure 8.4 shows an example MW-chain being used to sign the value 58. In this example, there are 3 signature chains, each of length 4, and 2 checksum chains, also each of length 4. To sign the value 58 in this case, $n_1 = (58 \bmod 4) + 1 = 3$, $n_2 = \left(\left\lfloor \frac{58}{4} \right\rfloor \bmod 4\right) + 1 = 3$, and $n_3 = \left(\left\lfloor \frac{58}{16} \right\rfloor \bmod 4\right) + 1 = 4$, so $n'_1 = ((12 - (2+2+3)) \bmod 4) + 1 = 2$ and $n'_2 = \left(\left\lfloor \frac{12-(2+2+3)}{4} \right\rfloor \bmod 4\right) + 1 = 2$. The signature is thus the ordered set $\{s_{1,3},\ s_{2,3},\ s_{3,4},\ c_{1,2},\ c_{2,2}\}$.

Every signature chain i can sign $\log_2(\ell_i)$ bits. All signature chains together can sign $b = \sum_{i=1}^{m} \log_2(\ell_i)$ bits. To sign a message M of b bits, the signer splits the message into m chunks $M_1, \ldots, M_m$, each of size $\log_2(\ell)$ bits. The signer adds the values s_{i,M_i+1} to the signature. Note that the first value of a signature chain signs the number 0, the second value a 1, and so on.

To prevent an attacker from forging a message M', where $M_i \geq M'_i, 1 \leq i \leq m$ (because anybody can compute the one-way chain into that direction to know the previous values) the sender uses a checksum chain that moves in the opposite direction of the signature chains. Consequently, an attacker that tries to sign M' as described above would need to invert the checksum chain, which is computationally infeasible. The checksum chains need to be long enough to sign the maximum sum that might occur in the signature chains: $\prod_{i=1}^{m'} \ell'_i \geq 1 + \sum_{i=1}^{m} (\ell_i - 1)$.

The signer computes the checksum of the signature chains by summing all the values that it signed with the signature chains: $s = \sum_{i=1}^{m} M_i$. The signer splits the checksum into m' checksum chunks. The checksum chunks are encoded in reverse in the checksum chains, compared to how the message chunks are encoded n the signature chains. For checksum chunk s_i, the signer adds the value $c_{i,\ell'-s_i}$ to the signature.

Rohatgi [164] proposes a concrete instantiation to sign an 80-bit message: 20 signature chains of length 16, and 3 checksum chains of length 16. Zhang [191] presents a similar mechanism, except that he does not bring the multiple hash chains together into heads. As a result, MW-chains have an advantage in reduced storage overhead.

In retrospect, it may seem that the development of hash tree chains was unnecessary: a node could use an MW-chain to sign its node identifier, thus preventing a node from directly replaying its authenticator. Unfortunately, using MW-chains in this context is not secure, since an attacker receiving several advertisements of equal metric can recover many values of the signature and checksum chains. For example, we performed a Monte Carlo simulation for a scenario in which $n = 32640$ nodes are represented using 5 signature chains of length 8, and 2 checksum chains of length 4, and each attacker hears 3 advertisements. In this case, an attacker was able to forge a valid signature with a probability of 0.196, in contrast to the hash tree chain, where the probability of successful forgery was 3.6×10^{-4}.

8.2.5. Skipchains for Preventing Denial-of-Service Attacks and for Faster Hash Chain Authentication

In Section 8.2.3, we described a mechanism that allows each node to verify a hash chain without needing to perform a large number of hash functions. However, the amount of effort required to verify an element is $O(k + \lg s)$, where k is the length of the hash chain and s is the number of hash chains. The network overhead is $O(\lg s)$, and initial computation cost is $O(ks)$. If the maximum metric is large, this approach may be prohibitively expensive, either in terms of initial computation cost or for element verification.

In this section, we describe an approach which, when combined with the Merkle tree authentication described in Section 8.2.3, has $O(c\sqrt[c]{k} + \lg s)$ verification cost and $O(s\sqrt[c]{k})$ generation cost, at the cost of $O(c + \lg s)$ overhead, where c is any positive integer. We achieve this by creating a *skipchain*, which is a chain that, when followed for one step, skips over many steps in a virtual hash chain. In the most basic version, a skipchain is $\sqrt{k}$ long, and each step in a skipchain represents $\sqrt{k}$ steps in a hash chain, which represents $c = 2$. In general, skipchains can be embedded inside skipchains, allowing values of $c > 2$. Skipchains can also be used in protocols such as TESLA [145, 146] and BiBa [144], to improve the efficiency of following long hash chains.

Each skipchain is represented by an MW-chain capable of signing enough bits to ensure security (for example, 80 bits). Each step in this MW-chain represents m steps in a virtual hash chain. To generate the hash chain (or skipchain) associated with this step, a new head is chosen by hashing the head of this step. The anchor of this hash chain (or skipchain) is computed, and that step in the MW-chain is used to sign this new anchor. For example, if the head of one step in a skipchain is v_i, a node forms $h_{0,i} = H[v_i]$, computes the corresponding anchor (for example $h_{m,i} = H^m(h_{0,i})$, if this is the last level of skipchains). It signs this anchor using v_i, as described in Section 8.2.4.

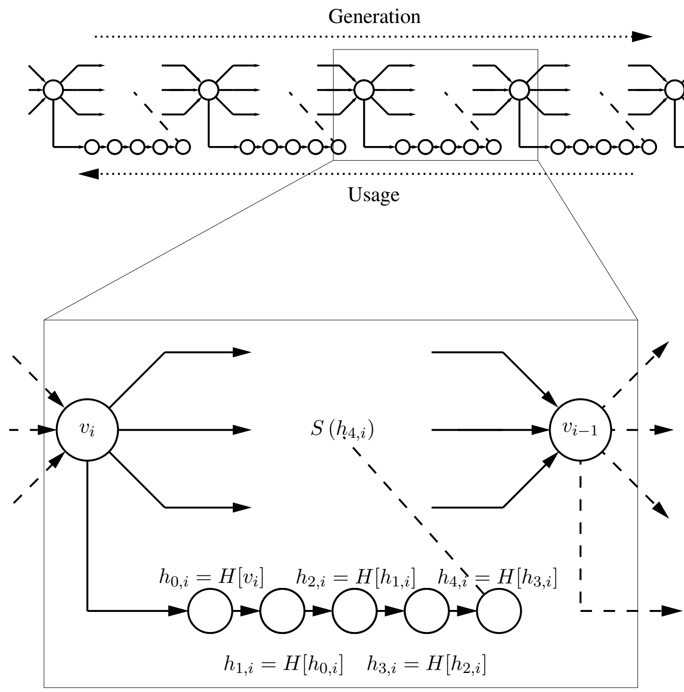


Figure 8.5: One step in a skipchain with $k = 4$

More concretely, we consider the case in which there is one level of skipchain, and each step in the skipchain corresponds to m steps in the virtual hash chain. If the MW-chain is n steps long, then the virtual hash chain is mn steps long. The leftmost element in this virtual hash chain is v_n, from which all chain elements can be derived. An alternative representation is the pair $(h_{0,n}, S_{v_n}(h_{m,n}))$, where $S_{v_n}(h_{m,n})$ represents $h_{m,n}$ signed using v_n. The next element is the pair $(h_{1,n}, S_{v_n}(h_{m,n}))$. The element at position $(m + 1)$ from the left is $(h_{1,n-1}, S_{v_{n-1}}(h_{m,n-1}))$. In general, the xth element from the left is represented by the pair $(h_{x \bmod n,y}, S_{v_y}(h_{m,y}))$, where $y = n - \lfloor \frac{x}{m} \rfloor$.

To verify a hash element, a node follows the hash chain to the anchor, and verifies the signature of the anchor. If there are multiple levels of skipchains (that is, if $c > 2$), the signature is verified recursively: that is, the verification of the signature requires the verification of a signature in a higher level chain. For example, if there are two levels of skipchains ($c = 3$), then the hash chain is followed to its anchor, the second level skipchain signature is checked by following that skipchain to its anchor, and the anchor of that skipchain is verified by verifying the signature in the top-level skipchain.

Skipchains can be generalized to allow skipping over any type of one-way chain that is formed from a single arbitrary head and can be verified using a single anchor. For example, hash tree chains can be used in conjunction with skipchains. This generalized skipchain is generated in the same way as skipchains over hash chains: at the lowest level of skipchain, the head of one step is used to seed the head of the one-way chain, and the anchor of that one-way chain is signed by that step in the skipchain.

Another possible application of such skipchains is to choose the top-level skipchain to represent k steps, where k is the maximum diameter of the network. This would reduce the initial cost of setting up a Merkle tree (as described in Section 8.2.3) from $O(s\sqrt[c]{k})$ to $O(s)$, where s is the number of sequence numbers covered by the tree. This increases overhead and computation cost by $O(1)$ for each update sent and verified, respectively.

Table 8.1: Our mechanisms compared with public key equivalents

	Initialization Computation	Per-Hop Computation	Overhead (Bits)
Hash tree chain	M·120 μs	(M-m)·120 μs	1680
RSA Equivalent (CPU Optimized)	.235 μs	.235 μs +m·401 μs	1024m + 80mlg$_2$ Nη
RSA Equivalent (Minimal Overhead)	7669 μs	η·7669 μs +m·401 μs	1040m
Tree Authenticated One-Way Chains	1.5 μs	3 μs	1600
RSA Equivalent	7669 μs	401 μs	1024
Skipchain	(M/α)·120 μs + α·*step*	(M/α)·120 μs + 2α·*step*	1920
RSA Equivalent	(M/α)·7669 μs + α·*step*	401 μs + 2α·*step*	(M/α)·1024

Notation: M is the maximum metric, m is the metric at a hop, n is the total network size, η is the average number of neighbors, α is the number of hops covered by one skipchain hop, and *step* is the cost of one hash chain step. RSA timings were performed with 1024-bit keys, using OpenSSL [135]. Hash tree chain performance is based on a network of size 32640, roughly the number of ASes in the Internet, and uses hash tree of size 2^6, with 3 values corresponding to each node. CPU optimized RSA equivalent combines all routing table elements using a Merkle tree, amortizing signature costs across all routing table elements. Tree-authenticated one-way chain is of size 2^{20}, and the calculation of initialization cost is amortized over all elements.

8.2.6. Efficiency Evaluation

To evaluate the efficiency of our mechanisms, we implemented generation and verification procedures for the three mechanisms described in this section. For efficiency, our hash function is based on the Rijndael block cipher [41] in the Matyas, Meyer, and Oseas construction [118], with a 128-bit key and a 128- or 192-bit block size, depending on the number of bits to be hashed. With a single block to be hashed, the hash output is the following (with an initialization vector (IV) as the initial key K): $H(x) = E_K(x) \oplus x$. We built our implementation on top of Gladman's implementation [56]. We implemented hash tree chains with 64 leaves, which represents a 64-node network with a single element per node, or a 2016-node network, when using two elements per node. Our skipchain was based on Rohatgi's construction [164] of 20 signature chains of length 16 and 3 checksum chains of length 16.

We ran our tests on a laptop with a Mobile Pentium 4 CPU running at 1.6GHz. Verifying a node in a tree-authenticated one-way chain took 3.08 μs on average, computing one step in a hash tree chain took 120 μs on average, and computing one step of an MW-chain took 145 μs on average. As a result, in a network with maximum metric 16 using skipchains of length 4, the *worst case* verification takes just over one millisecond. Another advantage of our approach is that most of the computation needed for verification can be used for generation; in particular, the worst case authentication plus verification operation takes just 480 μs more than verification alone.

To compare these results to the efficiency of public-key cryptography, we analyzed the functionality provided by each mechanism. A summary of our analysis is shown in Table 8.1. The tree-authenticated one-way chain essentially provides a signature: given a public key (the root value), private values can be authenticated. Tree-authenticated one-way chains are significantly more efficient than existing approaches [190] that authenticate each anchor using RSA.

A hash tree chain uses cryptographic mechanisms to ensure that only nodes authorized to advertise a particular metric can advertise that metric. In particular, only nodes that hear an advertisement with metric m (or lower) can advertise metric $m + 1$. A public-key approach to this problem can be adapted from the solution proposed by Kent et al [96]: each node signs the list of nodes that are allowed to advertise a particular metric. Each routing table element includes a signature chain, with a length equal to the metric, which shows the delegation of authority for advertising particular metrics. A node verifying this chain would need to verify a number of signatures equal to the distance to the destination. In addition, each node needs to run a secure neighbor discovery protocol in order to know which neighbors to authorize. Though such a protocol may be easy to

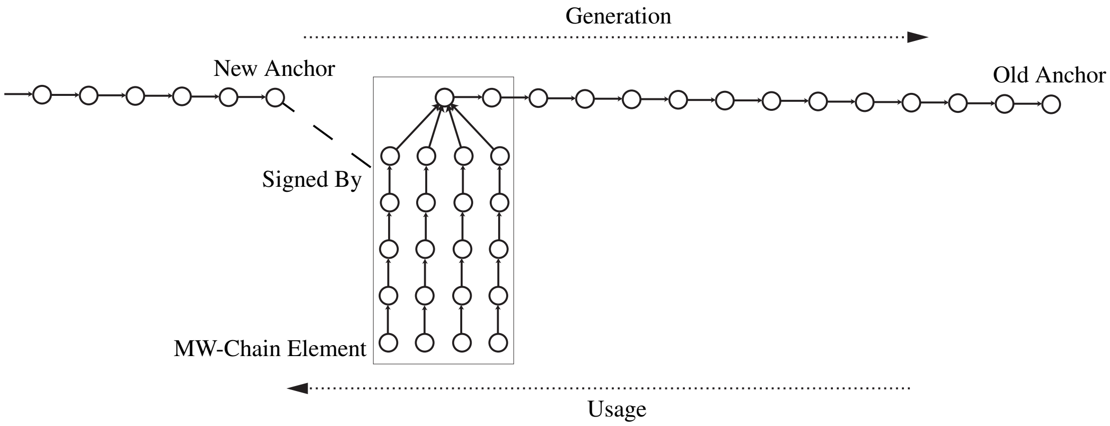


Figure 8.6: Bootstrapping a new hash chain. The new anchor is signed using the MW-chain element at the far left side of the old chain.

design in a wired network or a fixed wireless network, where a list of potential neighbors is easily generated, it could be prohibitively expensive in a mobile wireless environment such as an ad hoc network.

Finally, an alternative to skipchains is signatures. For example, in a network with maximum metric 16 and one step in a skipchain is used to skip over 4 elements, a sender can sign not only the anchor (metric 16 authenticator), it can also sign metric 4, 8, and 12 authenticators. Naturally, when a node sends an advertisement with metric 5, it will not include the signature of the metric 4 authenticator, and in general, a node advertising metric n will not include signatures on any metric m authenticators for $m < n$. Although skipchains may be slower than public key mechanisms on general-purpose processors, they have four advantages: first, they may require less network overhead for long chains; second, signature generation overhead is reduced, especially at the sender; third, they are easier to implement in hardware; and fourth, verification is easily parallelizable.

8.2.7. Bootstrapping New Chains and Trees

As time progresses, the elements of one-way hash chains or hash trees eventually run out and the node needs to securely distribute the anchor of a new chain or the root of a new tree. Recall that the security of our schemes relies on the secure distribution of these initial values, as they are used to authenticate all subsequent values. One solution to this problem is to compute a chain that is long enough to outlast the network; unfortunately, such a computation may be relatively expensive.

An alternative solution is to use the old chain or tree to authenticate the new anchor or root. To achieve this, we place a single MW-chain element at on the left-hand side of each hash chain or hash tree chain, or as the last element of a tree-authenticated one-way chain (as used in Section 8.2.3). When a node comes close to running out of its current chain, it generates a new chain, and uses the MW-chain element from its old chain to form a one-time signature on the anchor of the newly generated chain. It then distributes this new anchor by piggybacking it on several updates around the time the old chain expires. Figure 8.6 shows the use of a skipchain element in a hash chain

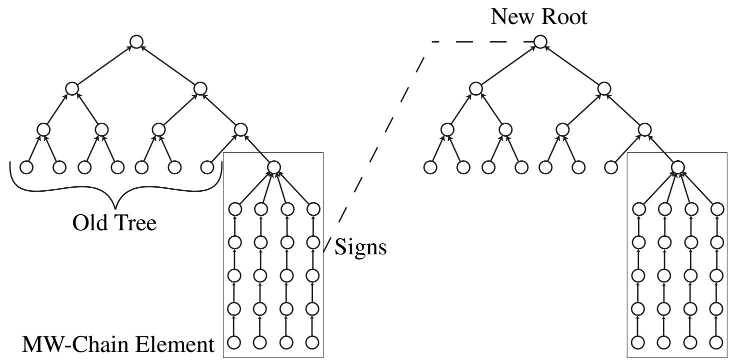


Figure 8.7: Bootstrapping a new tree-authenticated one-way chain. The new root is signed using the MW-chain element embedded in the old tree.

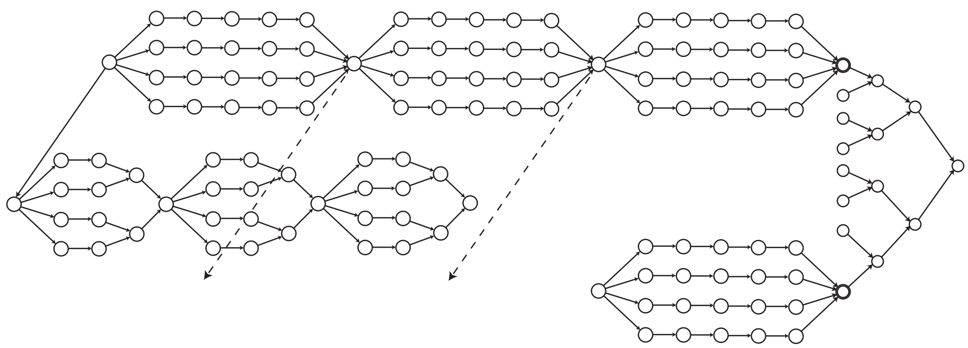


Figure 8.8: The Big Picture. Each leaf (except the bottom leaf) of the hash tree is identical to the top leaf, but the structure is omitted for clarity.

for authenticating a new hash chain anchor, and Figure 8.7 shows the use of a skipchain element in a tree-authenticated one-way chain for authenticating a new tree-authenticated one-way chain root.

This approach can be extended in several ways: each chain could contain several MW-chain elements to allow nodes to more easily reenter the network should they miss an entire chain of another node. Furthermore, a node may choose to piggyback a newly authenticated anchor often, when it first switches to the new chain, and progressively less often as the node consumes that chain. For example, the node may distribute the authentication information whenever the chain element used is a power of 2 from the anchor; this approach reduces overhead while still allowing nodes to rejoin the network after an extended time away.

8.2.8. Combining Our Primitives

Two of our primitives (hash tree chains described in Section 8.2.2 and tree-authenticated one-way chains in Section 8.2.3) protect against specific attacks (namely, same-metric fraud and the rushing attack); in addition, we provide skipchains (Section 8.2.4) for more efficient traversal of long hash chains. In order to prevent both of the above attacks, we can combine our approaches as shown in Figure 8.8. At the highest level, shown on the right side of the figure, we use a tree-authenticated one-way chain, which is a Merkle tree. The root of this Merkle tree is bootstrapped on each node. Each leaf in the Merkle tree is the anchor of another chain, with each leaf representing a single sequence number. In this case, the chains are skipchains built on top of hash tree chains; the chains

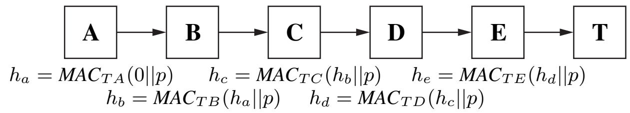


Figure 8.9: Cumulative authentication of packet p to a target **T**

could also be implemented as hash tree chains, or, if same-metric fraud is not a concern, as hash chains. Finally, at the bottom of the figure is an MW-chain element, which is later used for authenticating the root of the next tree-authenticated one-way chain.

8.3. A Mechanism for Securing Path-Vector Protocols

8.3.1. Overview of Path Vector Routing

Path vector protocols are similar to distance vector protocols, except that in place of the metric, each routing update includes a list of routers (or, in the case of BGP, a list of Autonomous Systems) on the route. By default, a path vector protocol will choose a route with the shortest recorded path; policies may also specify specific routers to prefer or to avoid. As a result, a node may wish to authenticate each hop that the routing update has traversed as recorded in the path, and to assure that no hops were removed from that recorded path.

A traditional way to perform this authentication is to have each node insert an authenticator in the packet, and to have the recipient individually verify each authenticator when the packet is received. This approach requires the network overhead of carrying a message authentication code (MAC) for each node in the path. In this section, we present a *cumulative authentication* mechanism that has the property that the message can be authenticated with only a single MAC, together with an ordered list of nodes traversed by the packet.

8.3.2. Cumulative Authentication

First, we describe the cumulative authentication mechanism in the case in which private keys are shared between the authenticating node and each node on the path. Each packet authenticated in this way maintains a *path authenticator* and an *address list*. When the packet traverses a node, the node appends its address to the address list. It authenticates its position in the list by replacing the path authenticator with a MAC computed over the received path authenticator and the packet's immutable fields.

When the packet reaches the receiver, if the path authenticator was originally initialized to a well known value (such as 0), then the receiver can reconstruct an expected final path authenticator value, given the address list. If the reconstructed value matches the received value, then the packet is deemed to be authentic and to have in fact traversed each node in the address list.

Figure 8.9 shows an example of cumulative authentication for a packet p. In addition to updating the path authenticator, each node also appends its own address to address list in the packet. If each node authenticates the packet using a shared MAC with **T**, then **T** can verify the path the packet traversed by verifying the received path authenticator h_e by checking that

$$h_e = MAC_{TE}(MAC_{TD}(MAC_{TC}(MAC_{TB}(MAC_{TA}(0 \,||\, p) \\ ||\, p) \,||\, p) \,||\, p) \,||\, p) \ .$$

Cumulative authentication also resists the removal of previous nodes from the address list. For example, in Figure 8.9, if an attacker **C** wishes to remove **B** from the address list, it must obtain h_a to derive a valid $h_c = MAC_{TC}(h_a||p)$. Since inverting **B**'s MAC is infeasible, an attacker generally must have the cooperation of the node immediately before the node to be removed. This mechanism does not prevent the second node from removing the first node, but since the first node is the source node, this is equivalent to the second node dropping the original packet and originating a packet of the same type to the destination.

Instead of using private, shared keys for authentication, it is also possible to use our cumulative authentication mechanism in the case in which the TESLA broadcast authentication protocol [145, 146] is used for authentication; the authentication can be performed either by the sender of the packet to be authenticated or by each recipient. To perform the authentication at each recipient, as may be desirable with a proactive routing protocol, such as BGP, each node along the path verifies the TESLA "security condition" (that the TESLA keys have not yet been released) and updates a address list and path authenticator as described above using its current TESLA key. The node then buffers the packet for verification. Later, the sender transmits the key, required for the verification of its authentication, to each node to which the sender transmitted the original routing packet. Each node receiving such an authentication packet verifies the authentication information. After the node performs that authentication, it appends its previous TESLA key to the authentication packet and transmits the new authentication packet to each neighbor to which it sent the original routing packet.

In an on-demand protocol, such as Ariadne (Chapter 7) an initiator floods a route request packet when it needs a route to a destination; the initiator may then wish to perform the authentication. In this case, each node along the path updates a address list and path authenticator as described above. When the packet reaches the destination, the destination verifies the TESLA security condition. Alternatively, the destination can include a timestamp, and allow the source to verify the security condition. The destination then adds an authenticator to the path authenticator and address list (and possibly the timestamp), and sends the packet along the reverse of the route along which it came. Each node receiving such a packet includes in the packet a key that allows the original authenticator to be reconstructed. If the end-to-end authentication is also performed using TESLA, the TESLA key used by the destination for authenticating the path authenticator, address list, and timestamp must be sent to the original sender.

8.3.3. Performance Evaluation

To evaluate the performance of cumulative authentication, we examined the overhead reduction resulting from using cumulative authentication together with Ariadne (Chapter 7) We performed 140 simulations, each running over 900 simulated seconds, and examined the number of bytes of overhead transmitted within control packets. When Ariadne was run without cumulative authentication, the *total* overhead across 50 nodes and 126000 simulated seconds was 1997 megabytes, whereas with cumulative authentication the same total overhead was 1491 megabytes. This result represents a 25% reduction in routing overhead.

8.4. Chapter Summary

In this chapter, we have presented four new mechanisms as building blocks for creating secure distance vector and path vector routing protocols. These mechanisms not only protect the routing protocol against standard routing attacks, they are based on highly efficient *symmetric* cryptographic techniques; our mechanisms thus also help to protect the routing protocol against denial of service attacks based for example on simply by flooding large numbers of randomly generated, forged routing messages, which then must be authenticated and rejected by the routers.

For securing distance vector protocols, our *hash tree chain* mechanism forces a router to increase the distance (metric) when forwarding a routing table entry. To provide authentication of a received routing update in bounded time, we presented a new mechanism, similar to hash chains, that we call *tree-authenticated one-way chains*. For cases in which the maximum metric is large, we presented *skipchains*, which provide more efficient initial computation cost and more efficient element verification; this mechanism is based on a new cryptographic mechanism, called MW-chains, which we also presented. For securing path vector protocols, our *cumulative authentication* mechanism authenticates the list of routers on the path in a routing update, preventing removal or reordering of the router addresses in the list; this mechanism uses using only a single authenticator in the routing update rather than one per router address.

As our economy and critical infrastructure increasingly rely on the Internet, securing routing protocols becomes of critical importance. The routing security mechanisms we have described can be applied to conventional routing protocols such as those in use in the Internet today, as well as to specialized routing protocols designed for new environments such as multihop wireless ad hoc networking. Our mechanisms provide a foundation on which efficient secure routing protocols can be designed, and we leave the development of such protocols to future work.

Chapter 9

Packet Leashes: A Defense against Wormhole Attacks

The promise of mobile ad hoc networks to solve challenging real-world problems continues to attract attention from industrial and academic research projects. Applications are emerging and widespread adoption is on the horizon. Most previous ad hoc networking research has focused on problems such as routing and communication, assuming a trusted environment. However, many applications run in untrusted environments and require secure communication and routing. Applications that may require secure communications include emergency response operations, military or police networks, and safety-critical business operations such as oil drilling platforms or mining operations. For example, in emergency response operations such as after a natural disaster like a flood, tornado, hurricane, or earthquake, ad hoc networks could be used for real-time safety feedback; regular communication networks may be damaged, so emergency rescue teams might rely upon ad hoc networks for communication.

Ad hoc networks generally use a wireless radio communication channel. The main advantages of such networks are rapid deployment and low cost of operation, since the nodes and wireless hardware are inexpensive and readily available, and since the network is automatically self-configuring and self-maintaining. However, wireless networks are vulnerable to several attacks. In most wireless networks, an attacker can easily *inject* bogus packets, impersonating another sender. We refer to this attack as a *spoofing* attack. An attacker can also easily *eavesdrop* on communication, record packets, and *replay* the (potentially altered) packets.

In this paper, we define a particularly challenging attack to defend against, which we call a *wormhole* attack, and we present a new, general mechanism for detecting and thus defending against wormhole attacks. In this attack, an attacker records a packet, or individual bits from a packet, at one location in the network, tunnels the packet (possibly selectively) to another location, and replays it there. The wormhole attack can form a serious threat in wireless networks, especially against many ad hoc network routing protocols and location-based wireless security systems. The wormhole places the attacker in a very powerful position, able for example to further exploit any of the attacks mentioned above, allowing the attacker to gain unauthorized access, disrupt routing, or perform a Denial-of-Service (DoS) attack. We introduce the general mechanism of *packet leashes* to detect wormhole attacks, and we present two types of leashes: *geographic leashes* and *temporal leashes*. Finally, we design an efficient authentication protocol, called TIK, for use with temporal leashes. We focus our discussion in this paper on wireless ad hoc networks, but our results are applicable more broadly to other types of networks, such as wireless LANs and cellular networks.

9.1. Problem Statement

In a *wormhole attack*, an attacker receives packets at one point in the network, "tunnels" them to another point in the network, and then replays them into the network from that point. For tunneled distances longer than the normal wireless transmission range of a single hop, it is simple for the attacker to make the tunneled packet arrive sooner than other packets transmitted over a normal multihop route, for example through use of a single long-range directional wireless link or through a direct wired link to a colluding attacker. It is also possible for the attacker to forward each bit over the wormhole directly, without waiting for an entire packet to be received before beginning to tunnel the bits of the packet, in order to minimize delay introduced by the wormhole. Due to the nature of wireless transmission, the attacker can create a wormhole even for packets not addressed to itself, since it can overhear them in wireless transmission and tunnel them to the colluding attacker at the opposite end of the wormhole.

If the attacker performs this tunneling honestly and reliably, no harm is done; the attacker actually provides a useful service in connecting the network more efficiently. However, the wormhole puts the attacker in a very powerful position relative to other nodes in the network, and the attacker could exploit this position in a variety of ways. The attack can also still be performed even if the network communication provides confidentiality and authenticity, and even if the attacker has no cryptographic keys. Furthermore, the attacker is invisible at higher layers; unlike a malicious node in a routing protocol, which can often easily be named, the presence of the wormhole and the two colluding attackers at either endpoint of the wormhole are not visible in the route. As such, the effect of the wormhole on legitimate nodes may even change as nodes move; two legitimate nodes previously connected only by routes through the wormhole and thus possibly unable to communicate, will be able to communicate normally if they come within direct wireless transmission range of each other.

The wormhole attack is particularly dangerous against many ad hoc network routing protocols in which the nodes that hear a packet transmission directly from some node consider themselves to be in range of (and thus a neighbor of) that node. For example, when used against an on-demand routing protocol such as DSR [89] or AODV [141], a powerful application of the wormhole attack can be mounted by tunneling each ROUTE REQUEST packet directly to the destination target node of the REQUEST. When the destination node's neighbors hear this REQUEST packet, they will follow normal routing protocol processing to rebroadcast that copy of the REQUEST and then discard without processing all other received ROUTE REQUEST packets originating from this same Route Discovery. This attack thus prevents any routes other than through the wormhole from being discovered, and if the attacker is near the initiator of the Route Discovery, this attack can even prevent routes more than two hops long from being discovered. Possible ways for the attacker to then exploit the wormhole include discarding rather than forwarding all data packets, thereby creating a permanent Denial-of-Service attack (no other route to the destination can be discovered as long as the attacker maintains the wormhole for ROUTE REQUEST packets), or selectively discarding or modifying certain data packets.

The neighbor discovery mechanisms of periodic (proactive) routing protocols such as DSDV [140], OLSR [154], and TBRPF [13] rely heavily on the reception of broadcast packets as a means for neighbor detection, and are also extremely vulnerable to this attack. For example, OLSR and TBRPF use HELLO packets for neighbor detection, so if an attacker tunnels through a wormhole to a colluding attacker near node B all HELLO packets transmitted by node A, and likewise tunnels back to the first attacker all HELLO packets transmitted by B, then A and B will believe that they are neighbors, which would cause the routing protocol to fail to find routes when they are not actually neighbors.

For DSDV, if each routing advertisement sent by node A or node B were tunneled through a wormhole between colluding attackers near these nodes, as described above, then A and B would believe that they were neighbors. If A and B, however, were not within wireless transmission range of each other, they would be unable to communicate. Furthermore, if the best existing route from A to B were at least $2n + 2$ hops long, then any node within n hops of A would be unable to communicate with B, and any node within n hops of B would be unable to communicate with A. Otherwise, suppose C were within n hops of A, but had a valid route to B. Since A advertises a metric of 1 route to B, C would hear a metric $n + 1$ route to B. C will use that route if it is not within $n + 1$ hops of B, in which case there would be an n-hop route from A to C, and a route of length $n + 1$ from C to B, contradicting the premise that the best real route from A to B is at least $2n + 2$ hops long.

The wormhole attack is also dangerous in other types of wireless networks and applications. One example is any wireless access control system that is based on physical proximity, such as wireless car keys, or proximity and token based access control systems for PCs [39, 98]. In such systems, an attacker could relay the authentication exchanges to gain unauthorized access.

One partial approach for preventing wormhole attacks might be to use a secret method for modulating bits over wireless transmissions; once a node is compromised, however, this approach is likely to fail unless the radio is kept inside tamper-resistant hardware. Another approach, known as RF watermarking, authenticates a wireless transmission by modulating the RF waveform in a way known only to authorized nodes [43]. RF watermarking relies on keeping secret the knowledge of which RF waveform parameters are being modulated; furthermore, if that waveform is exactly captured at the receiving end of the wormhole and exactly replicated at the transmitting end of the wormhole, the signal level of the resulting watermark is independent of the distance it was tunneled. As a result, the watermark may still be intact, even though the packet was made to travel beyond the normal wireless transmission range. Although intrusion detection could be used in some cases to detect a wormhole attacker, it is generally difficult to isolate the attacker in a software-only approach, since the packets sent by the wormhole are identical to the packets sent by legitimate nodes. In contrast to these approaches, the approach we present in this paper, called *packet leashes*, and the specific protocol we present, called TIK, provide a general solution that does not suffer from these problems.

9.2. Assumptions and Notation

The acronym “MAC” may in general stand for “Medium Access Control” protocol or “Message Authentication Code.” To avoid confusion, we use “MAC” in this paper to refer to the network Medium Access Control protocol at the link layer, and we use “HMAC” to refer to a message authentication code used for authentication (HMAC is a particular instance of a message authentication code [11]).

For reasons such as differences in wireless interference, transmit power, or antenna operation, links between nodes in a wireless network may at times successfully work in only one direction; such a *unidirectional* wireless link between between two nodes A and B might allow A to send packets to B but not for B to send packets to A. In many cases, however, wireless links are able to operate as *bidirectional* links. A MAC protocol generally is designed to support operation over unidirectional links or is designed only for bidirectional links; the introduction of our TIK protocol does not affect the capability of the MAC protocol to operate over unidirectional links.

Security attacks on the wireless network’s physical layer are beyond the scope of this paper. Spread spectrum has been studied as a mechanism for securing the physical layer against jamming [149]. Denial-of-Service (DoS) attacks against MAC layer protocols are also beyond the scope of the paper; MAC layer protocols that do not employ some form of carrier sense, such as pure ALOHA and Slotted ALOHA [1], are less vulnerable to DoS attacks, although they tend to use the channel less efficiently.

We assume that the wireless network may drop, corrupt, duplicate, or reorder packets. We also assume that the MAC layer contains some level of redundancy to detect randomly corrupted packets; however, this mechanism is not designed to replace cryptographic authentication mechanisms.

We assume that nodes in the network may be resource constrained. Thus, in providing for wormhole detection, we use efficient *symmetric* cryptography, rather than relying on expensive *asymmetric* cryptographic operations. Especially on CPU-limited devices, symmetric cryptographic operations (such as block ciphers and hash functions) are three to four orders of magnitude faster than asymmetric cryptographic operations (such as digital signatures).

We assume that a node can obtain an authenticated key for any other node. Like public keys in systems using asymmetric cryptography, these keys in our protocol TIK (Section 9.4) are public values (once disclosed), although TIK uses only symmetric (not asymmetric) cryptography. A traditional approach to this authenticated key distribution problem is to build on a public key system for key distribution; a trusted entity can sign public-key certificates for each node, and the nodes can then use their public-key to sign a new (symmetric) key being distributed for use in TIK. Zhou and Haas [192] propose such a public key infrastructure; Hubaux, Buttyán, and Čapkun bootstrap trust relationships from PGP-like certificates without relying on a trusted public key infrastructure [82]; Kong et al [101] propose asymmetric mechanisms for threshold signatures for certificates. Alternatively, a trusted node can securely distribute an authenticated TIK key using only symmetric-key cryptography [148] or non-cryptographic approaches [178].

9.3. Detecting Wormhole Attacks

In this section, we introduce the notion of a *packet leash* as a general mechanism for detecting and thus defending against wormhole attacks. A leash is any information that is added to a packet designed to restrict the packet's maximum allowed transmission distance. We distinguish between *geographical leashes* and *temporal leashes*. A geographical leash ensures that the recipient of the packet is within a certain distance from the sender. A temporal leash ensures that the packet has an upper bound on its lifetime, which restricts the maximum travel distance, since the packet can travel at most at the speed of light. Either type of leash can prevent the wormhole attack, because it allows the receiver of a packet to detect if the packet traveled further than the leash allows.

9.3.1. Geographical Leashes

To construct a geographical leash, in general, each node must know its own location, and all nodes must have loosely synchronized clocks. When sending a packet, the sending node includes in the packet its own location, p_s, and the time at which it sent the packet, t_s; when receiving a packet, the receiving node compares these values to its own location, p_r, and the time at which it received the packet, t_r. If the clocks of the sender and receiver are synchronized to within $\pm\Delta$, and ν is an upper bound on the velocity of any node, then the receiver can compute an upper bound on the distance between the sender and itself, d_{sr}. Specifically, based on the timestamp t_s in the packet, the local receive time t_r, the maximum relative error in location information δ, and the locations of the receiver p_r and the sender p_s, then d_{sr} can be bounded by $d_{sr} \le ||p_s - p_r|| + 2\nu \cdot (t_r - t_s + \Delta) + \delta$. A regular digital signature scheme, e.g., RSA [163], or other authentication technique, can be used to allow a receiver to authenticate the location and timestamp in the received packet.

In certain circumstances, bounding the distance between the sender and receiver, d_{sr}, cannot prevent wormhole attacks; for example, when obstacles prevent communication between two nodes that would otherwise be in transmission range, a distance-based scheme would still allow wormholes between the sender and receiver. A network that uses location information to create a geographical leash could control even these kinds of wormholes. To accomplish this, each node would have a radio propagation model. A receiver could verify that every possible location of the sender (a $\delta + \nu(t_r - t_s + 2\Delta)$ radius around p_s) can reach every possible location of the receiver (a $\delta + \nu(t_r - t_s + 2\Delta)$ radius around p_r).

9.3.2. Temporal Leashes

To construct a temporal leash, in general, all nodes must have tightly synchronized clocks, such that maximum difference between any two nodes' clocks is Δ. The value of the parameter Δ must be known by all nodes in the network, and for temporal leashes, generally must be on the order of a few microseconds or even hundreds of nanoseconds. This level of time synchronization can be achieved now with off-the-shelf hardware based on LORAN-C [126], WWVB [127], or GPS [35, 183]; although such hardware is not currently a common part of wireless network nodes, it can be deployed in networks today and is expected to become more widely utilized in future systems at reduced expense, size, weight, and power consumption. In addition, the time synchronization signal itself in such systems may be subject to certain attacks [17, 54]. Esoteric hardware such as cesium-beam clocks, rubidium clocks, and hydrogen maser clocks, could also be used in special applications today to provide sufficiently accurate time synchronization for months. Although our general requirement for time synchronization is indeed a restriction on the applicability of temporal leashes, for applications that require defense against the wormhole attack, this requirement is justified due to the seriousness of the attack and its potential disruption of the intended functioning of the network.

To use temporal leashes, when sending a packet, the sending node includes in the packet the time at which it sent the packet, t_s; when receiving a packet, the receiving node compares this value to the time at which it received the packet, t_r. The receiver is thus able to detect if the packet traveled too far, based on the claimed transmission time and the speed of light. Alternatively, a temporal leash can be constructed by instead including in the packet an expiration time, after which the receiver should not accept the packet; based on the allowed maximum transmission distance and the speed of light, the sender sets this expiration time in the packet as an offset from the time at which it sends the packet. As with a geographical leash, a regular digital signature scheme or other authentication technique can be used to allow a receiver to authenticate a timestamp or expiration time in the received packet.

9.3.3. Discussion

An advantage of geographical leashes over temporal leashes is that the time synchronization can be much looser. Another advantage of using geographical leashes in conjunction with a signature scheme (i.e., a signature providing non-repudiation), is that an attacker can be caught if it pretends to reside at multiple locations. This use of non-repudiation was also proposed by Sirois and Kent [174]. When a legitimate node overhears the attacker claiming to be in different locations that would only be possible if the attacker could travel at a velocity above the maximum node velocity ν, the legitimate node can use the signed locations to convince other legitimate nodes that the attacker is malicious.

We define $\delta'(t)$ to be a bound on the maximum relative position error when any node determines its own location twice within a period of time t. By definition, $\delta'(t) \leq 2\delta$. In addition, when t is small, $\delta'(t)$ should be small, since the algorithm a node uses to determine its location should be aware of physical speed limits of that node. If some node claims to be at locations p_1 and p_2 at times t_1 and t_2, respectively, that node is an attacker if $\frac{||p_2-p_1||-\delta'(|t_2-t_1|)}{|t_2-t_1|} > \nu$. A legitimate node detecting this from these two packets can also broadcast the two packets to convince other nodes that the first node is indeed an attacker. Each node hearing these messages can check the two signatures, verify the discrepancy in the information, and rebroadcast the information if it has not previously done so. To easily perform duplicate suppression in rebroadcasting this information, each node can maintain a *blacklist*, with each entry in the blacklist containing a node address and the time at which that blacklist entry expires. When a node receives a message showing an attacker's behavior, it checks if that attacker is already listed in its blacklist. If so, it updates the expiration time on its current blacklist entry and discards the new message; otherwise, it adds a new blacklist entry and propagates the message.

A potential problem with leashes using a timestamp in the packet is that in a contention-based MAC protocol, the sender may not know the precise time at which it will transmit a packet it is sending. For example, a sender using the IEEE 802.11 MAC protocol may not know the time a packet will be transmitted until approximately one slot time (20 μs) prior to transmission. Generating an inefficient digital signature, such as RSA with a 1024-bit key, could take three orders of magnitude more time than this slot time (on the order of 10 ms). The sender, however, can use two approaches to hide this signature generation latency: either increase the *minimum* transmission unit to allow computation to overlap with transmission, or use a more efficient signature scheme, such as Schnorr's signature [171], which enables efficient signature generation after pre-processing.

9.4. Temporal Leashes and the TIK Protocol

In this section, we discuss temporal leashes in more detail and present the design and operation of our TIK protocol that implements temporal leashes.

9.4.1. Temporal Leash Construction Details

We now discuss temporal leashes that are implemented with a packet expiration time. We consider a sender who wants to send a packet with a temporal leash, preventing the packet from traveling further than distance L. (All nodes are time synchronized up to a maximum time synchronization error Δ.) Thus, $L > L_{min} = \Delta \cdot c$, where c is the propagation speed of our wireless signal (i.e., the speed of light in air, which is very close to the speed of light in a vacuum). When the sender sends the packet at local time t_s, it needs to set the packet expiration time to $t_e = t_s + L/c - \Delta$. When the receiver receives the packet at local time t_r, it further processes the packet if the temporal leash has not expired (i.e., $t_r < t_e$); otherwise it drops the packet. This assumes that the packet sending and receiving delay are negligible, such that the sender can predict the precise sending time t_s and the receiver can immediately record t_r when the first bit arrives (or derive t_r during reception since the bitrate of transmission is known).

The receiver needs a way to authenticate the expiration time t_e, as otherwise an attacker could easily change that time and wormhole the packet as far as it desires.

In unicast communication, (point-to-point) nodes can use *message authentication codes* for authentication: the sender S and receiver R must share a secret key K, which they use in conjunction with a message authentication code function (for example HMAC [11]) to authenticate messages they exchange. To send a message M to a receiver R, the sender S sends

$$S \rightarrow R: \quad \langle M, \mathrm{MAC}_K(M) \rangle \ ,$$

where the notation $\text{MAC}_K(M)$ represents the message authentication code computed over message M with key K. The packet sent from S to R contains both the intended message M and $\text{MAC}_K(M)$. When R receives this message, it can verify the authenticity of the message by comparing the received HMAC value to the HMAC value that it computes for itself over the received message with the secret key K it shares with the sender S.

However, using message authentication codes in the standard way has two major drawbacks. First, in a network with n nodes, we would need to set up $\frac{n(n-1)}{2}$ keys, one for each pair of nodes. Key setup is an expensive operation, which makes this approach impractical in large networks. Second, this approach cannot efficiently authenticate broadcast packets. To secure a broadcast packet, the sender would need to add to the packet a separate message authentication code for each receiver, making the packet extremely large (and likely exceeding the network's maximum packet size). The need to include separate message authentication codes in the packet could be avoided by having multiple receivers share the same key, but this might allow a subset of colluding receivers to impersonate the sender [29].

Instead, attaching a *digital signature* to each packet could be used to solve the two problems discussed above: each node needs to have only one public-private key pair, and each node needs to know only the public key of every other node. Thus, only n public keys need to be distributed in a network with n nodes. Furthermore, a digital signature provides non-repudiation and authentication for broadcast packets in the same way as for unicast packets.

However, digital signatures have several drawbacks. First, digital signatures are usually based on computationally expensive *asymmetric* cryptography. For example, the popular 1024-bit RSA digital signature algorithm [163], roughly equivalent to use of a 72-bit key in a symmetric encryption algorithm [106], requires about 10 ms on an 800 MHz Pentium III processor for signature generation. Signature verification is more efficient, but still requires about 0.5 ms on a fast workstation. Adding a digital signature to each packet is computationally expensive for the verifier (receiver), but overwhelmingly expensive for the signer (sender). On less powerful CPUs, each digital signature generation and verification takes on the order of seconds [27].

Since many wireless applications rely heavily on broadcast communication, and since setting up $O(n^2)$ keys is expensive, we design the TIK protocol in Section 9.4.2, based on a new protocol for efficient broadcast authentication that simultaneously provides the functionality of a temporal leash.

9.4.2. TIK Protocol Description

Our TIK protocol implements temporal leashes and provides efficient instant authentication for broadcast communication in wireless networks. TIK stands for *TESLA with Instant Key disclosure*, and is an extension of the TESLA broadcast authentication protocol [147]. We contribute the novel observation that a receiver can verify the TESLA *security condition* (that the corresponding key has not yet been disclosed) as it receives the packet (explained below); this fact allows the sender to disclose the key in the same packet, thus motivating the protocol name "TESLA with Instant Key disclosure."

TIK implements a temporal leash and thus enables the receiver to detect a wormhole attack. TIK is based on efficient *symmetric* cryptographic primitives (a message authentication code is a symmetric cryptographic primitive). TIK requires accurate time synchronization between all communicating parties, and requires each communicating node to know just one public value for each sender node, thus enabling scalable key distribution.

We now describe the different stages of the TIK protocol in detail: sender setup, receiver bootstrapping, and sending and verifying authenticated packets.

Sender Setup

The sender uses a pseudo-random function (PRF [58]) $\mathcal{F}$ and a secret master key $\mathcal{X}$ to derive a series of keys $K_0, K_1, \ldots, K_w$, where $K_i = \mathcal{F}_{\mathcal{X}}(i)$. The main advantage of this method of key generation is that the sender can efficiently access the keys in any order. Assuming the PRF is secure, it is computationally intractable for an attacker to find the master secret key $\mathcal{X}$, even if all keys $K_0, K_1, \ldots, K_{w-1}$ are known. Without the secret master key $\mathcal{X}$, it is computationally intractable for an attacker to derive a key K_i that the sender has not yet disclosed. To construct the PRF function $\mathcal{F}$, we can use a pseudo-random permutation, i.e., a block cipher [59], or a message authentication code, such as HMAC [11].

The sender selects a key expiration interval I, and thus determines a schedule with which each of its keys will expire. Specifically, key K_0 expires at time T_0, key K_1 expires at time $T_1 = T_0 + I$, $\ldots$, key K_i expires at time $T_i = T_{i-1} + I = T_0 + i \cdot I$.

The sender constructs the Merkle hash tree we describe in Section 5.1.2 to commit to the keys $K_0, K_1, \ldots, K_{w-1}$. The root of the resulting hash tree is $m_{0,w-1}$, or simply m. The value m commits to all keys and is used to authenticate any leaf key efficiently. As we describe in Section 5.1.2, in a hash tree with $\log_2(w)$ levels, verification requires only $\log_2 w$ hash function computations (in the worst case, not considering buffering), and the authentication information consists of $\log_2 w$ values.

Receiver Bootstrapping

We assume that all nodes have synchronized clocks with a maximum clock synchronization error of Δ. We further assume that each receiver knows every sender's hash tree root m, and the associated parameters T_0 and I. This information is sufficient for the receiver to authenticate any packets from the sender.

Sending and Verifying Authenticated Packets

To achieve secure broadcast authentication, it must not be possible for a receiver to forge authentication information for a packet. When the sender sends a packet P, it estimates an upper bound t_r on the arrival time of the HMAC at the receiver. Based on this arrival time, the sender picks a key K_i that will not have expired when the receiver receives the packet's HMAC ($T_i > t_r + \Delta$). The sender attaches the HMAC to the packet, computed using key K_i, and later discloses the key K_i itself, along with the corresponding tree authentication values (as discussed in Section 5.1.2), after the key has expired.

Because of the time synchronization, the receiver can verify after receiving the packet that the key K_i used to compute the authentication has not yet been disclosed, since the receiver knows the expiration time for each key and the sender only discloses the key after it expires; thus, no attacker can know K_i, and therefore if the packet authentication verifies correctly once the receiver later receives the authentic key K_i, the packet must have originated from the claimed sender. Even another receiver could not have forged a new message with a correct message authentication code, since only the sender knew the key K_i at the time t_r that the receiver received the packet. After the key K_i expires at time T_i, the sender then discloses key K_i (and the corresponding tree authentication values); once the receiver gets the authentic key K_i, it can authenticate all packets that carry a message authentication code computed with K_i. This use of delayed key disclosure and time synchronization for secure broadcast authentication was also used by the TESLA protocol [147].

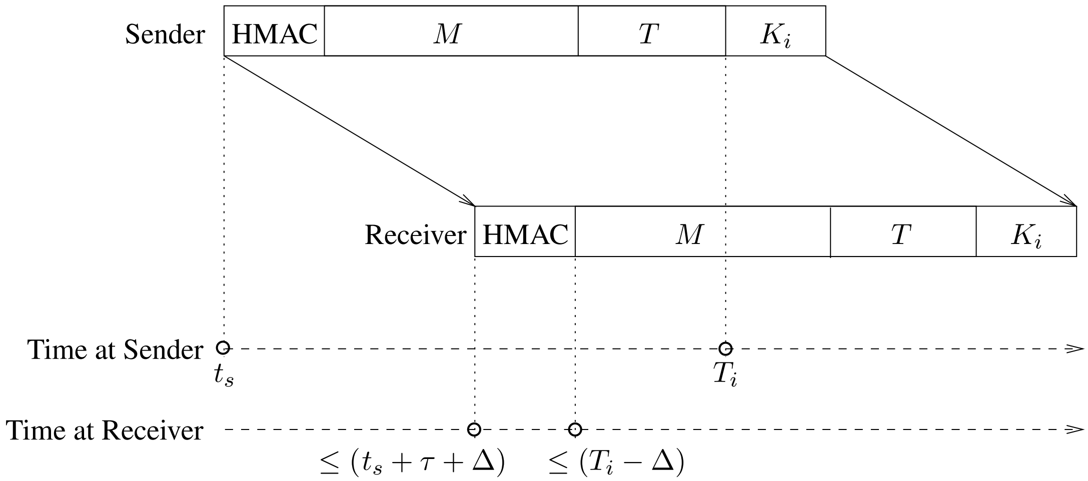


Figure 9.1: Timing of a packet in transmission using TIK

The above protocol has the drawback that message authentication is delayed; the receiver must wait for the key before it can authenticate the packet. We observe that we can remove the authentication delay in an environment in which the nodes are tightly time synchronized. In fact, the sender can even disclose the key in the same packet that carries the corresponding message authentication code.

Figure 9.1 shows the sending and receiving of a TIK packet. The figure shows the sender's and receiver's timelines, which may differ by a value of up to the maximum time synchronization error Δ. The time t_s here is the time at which the sender S begins transmission of the packet, and time T_i is the disclosure time for key K_i. The packet contains four parts: a message authentication code (shown as HMAC in Figure 9.1), a message payload (shown as M), the tree authentication values necessary to authenticate K_i (shown as T), and the key used to generate the message authentication code (shown as K_i). The TIK packet is transmitted by S as

$$S \rightarrow R : \quad \langle \text{MAC}_{K_i}(M), M, T, K_i \rangle \ ,$$

where the destination R may be unicast or broadcast. After the receiver R receives the HMAC value, it verifies that the sender did not yet start sending the corresponding key K_i, based on the time T_i and the synchronized clocks. If the sender did not yet start sending K_i, the receiver verifies that the key K_i at the end of the packet is authentic (using the hash tree root m and the hash tree values T), and then uses K_i to verify the HMAC value in the packet. If all these verifications are successful, the receiver accepts the packet as authentic.

The TIK protocol already provides protection against the wormhole attack, since an attacker who retransmits the packet will most likely delay it long enough that the receiver will reject the packet because the corresponding key has already expired and the sender may have disclosed it. However, we can also add an explicit expiration timestamp to each packet for the temporal leash, and use TIK as the authentication protocol. For example, each packet could include a 64-bit timestamp with nanosecond resolution, allowing over 580 years of use starting from the epoch. Since the entire packet is authenticated, the timestamp is authenticated.

A policy could be set allowing the reception of packets for which the perceived transmission delay, i.e., the arrival time minus the sending timestamp, is less than some threshold. That threshold could be chosen anywhere between $\tau - \Delta$ and $\tau + \Delta$, where the more conservative approach of $\tau - \Delta$

never allows tunnels but rejects some valid packets, and the more liberal approach of $\tau + \Delta$ never rejects valid packets, but may allow tunneling of up to $2c\Delta$ past the actual normal transmission range.

With a GPS-disciplined clock [183], time synchronization to within $\Delta = 183$ ns with probability $1-10^{-10}$ is possible. If a transmitter has a 250 m range, the $\tau - \Delta$ threshold accepts all packets sent less than 140 m and some packets sent between 140 and 250 m; the $\tau + \Delta$ threshold accepts all packets sent less than 250 m but allows tunneling of packets up to 110 m beyond that distance.

9.4.3. MAC Layer Considerations

A TDMA MAC protocol may be able to choose the time at which a frame begins transmission, so that the message authentication code is sent by time $T_i - \frac{r}{c} - 2\Delta$. In this case, the minimum payload length is $\frac{r}{c} + 2\Delta$ times the bit rate of transmission. For additional efficiency, different nodes should have different key disclosure times, and the MAC layer should provide each node with the MAC layer time slot it needs for authenticated delivery.

As mentioned in Section 9.4.2, a CSMA MAC protocol may not be able to control that time at which a frame is sent relative to the key disclosure times. In this case, the minimum payload length needs to be chosen so that a key disclosure time is guaranteed to occur somewhere during the packet's transmission. For example, if the network physical layer is capable of a peak data rate of 100 Mbps and a range of 150 m, and if the key disclosure interval is chosen to be 25 μs and time synchronization is achieved to within 250 ns, then the minimum packet size must be at least 325 bytes. However, if each value in the hash tree is 80 bits long, and the depth of the tree is 31, then the minimum payload size is just 15 bytes.

If a MAC protocol uses a Request-to-Send/Clear-to-Send (RTS/CTS) frame handshake, the minimum packet size can be reduced by carrying the message authentication code inside the RTS frame. In this case, the frame exchange for transmitting a data packet would be

$$\begin{aligned} A \to B: &\quad \langle RTS, \mathrm{MAC}_{K_i}(M) \rangle \\ B \to A: &\quad \langle CTS \rangle \\ A \to B: &\quad \langle DATA, M, \textit{tree values}, K_i \rangle \ . \end{aligned}$$

In particular, instead of having a minimum message size of $\frac{r}{c} + 2\Delta + I$ times the transmission data rate, where I is the duration of a time interval, the minimum message size is just $2\Delta + I - 2t_{turn}$ times the data rate, where t_{turn} is the minimum allowed time between receiving a control frame (i.e., the RTS or CTS) and returning a corresponding frame (the CTS or DATA frame, respectively). This minimum message length includes the length of the CTS, DATA header, payload, and hash tree values.

9.5. Evaluation

9.5.1. TIK Performance

To evaluate the suitability of our work for use in ad hoc networks, we measured computational power and memory currently available in mobile devices. To measure the number of repeated hashes that can be computed per second, we optimized the MD5 hash code from ISI [182] to achieve maximum performance for repeated hashing.

Our optimized version performs 10 million hash function evaluations in 7.544 s on a Pentium III running at 1 GHz, representing a rate of 1.3 million hashes per second; the same number of hashes using this implementation on a Compaq iPaq 3870 PocketPC running Linux took 45 s, representing a rate of 222,000 hashes per second. Repetitive, simple functions like hashes can also be efficiently implemented in hardware; Helion Technology [70] claims a 20k gate ASIC core design (a third the complexity of Bluetooth [5] and less than a third the complexity of IEEE 802.11 [95]) capable of more than 1.9 million hashes per second and a Xilinx FPGA design using 1650 LUTs capable of 1 million hashes per second. In terms of memory consumption, existing handheld devices, such as the iPaq 3870, come equipped with 32 MB of Flash and 64 MB of RAM. Modern notebooks can generally be equipped with hundreds of megabytes of RAM.

A high-end wireless LAN card such as the Proxim Harmony 802.11a [153] has a transmission range potentially as far as 250 m and data rate as high as 108 Mbps. With time synchronization provided by a Trimble Thunderbolt GPS-Disciplined Clock [183], the synchronization error can be as low as 183 ns with probability $1-10^{-10}$. If authentic keys are re-established every day, with a 20-byte minimum packet size and an 80-bit message authentication code length, the tree has depth 33, giving a minimum payload length of 350 bytes (a transmisison time of 25.9 μs) and a time interval of 24.7 μs. Assuming that the node generates each new tree while it is using its current tree, it requires 8 megabytes of storage and needs to perform fewer than 243,000 operations per second to maintain and generate trees. To authenticate a received packet, a node needs to perform only 33 hash functions. To keep up with link-speed, a node needs to verify a packet at most every 25.9 μs, thus requiring 1,273,000 hashes per second, for a total computational requirement of 1,516,000 hashes per second. This can be achieved today in hardware, either by placing two MD5 units on a single FPGA, or with an ASIC. Many laptops today are equiped with at least 1.2 GHz Pentium III CPUs, which should also be able to perform 1.5 million hash operations per second.

Current commodity wireless LAN products such as commonly used IEEE 802.11b cards [2] provide a transmission data rate of 11 Mbps and a range of 250 m. Given the same time synchronization, rekeying interval, minimum packet size, and message authentication code length, the tree has depth 30, giving a minimum payload length of 320 bytes (a transmission time of 232 μs) and a time interval of 231.5 μs. Assuming that the node generates each new tree while it is using its current tree, it requires just 2.6 megabytes of storage and needs to perform just 26,500 operations per second. To authenticate a received packet, a node needs to perform only 30 hash functions. Since any IP packet authenticated using TIK would take at least 232 μs to transmit in this example, TIK can authenticate packets at link-speed using just 13,000 hashes per second, for a total of 39,500 hash functions per second, which is well within the capability of an iPaq, with 82.2% of its CPU time to spare.

In a sensor network such as Hollar et al's weC mote [92, 187], nodes may only be able to achieve time synchronization accurate to 1 s, have a 19.6 kbps link speed, and 20 m range. In this case, the smallest packet that can be authenticated is 4900 bytes; since the weC mote does not have sufficient memory to store this packet, TIK is unusable in such a resource-scarce system. Furthermore, the level of time synchronization in this system is such that TIK could not provide a usable wormhole detection system.

9.5.2. Security Analysis

Packet leashes provide a way for a sender and a receiver to ensure that a wormhole attacker is not causing the signal to propagate farther than the specified normal transmission distance. When geographic leashes are used, nodes also detect tunneling across obstacles such as mountains that are otherwise impenetrable by radio. As with other cryptographic primitives, a malicious receiver can refuse to check the leash, just like a malicious receiver can refuse to check the authentication on a packet. This may allow an attacker to tunnel a packet to another attacker without detection;

however, that second attacker cannot then retransmit the packet as if it were the original sender without then being detected.

A malicious sender can claim a false timestamp or location, causing a legitimate receiver to have mistaken beliefs about whether or not the packet was tunneled. When geographic leashes are used in conjunction with digital signatures, nodes may be able to detect a malicious node and spread that information to other nodes, as discussed in Section 9.3.3. However, this attack is equivalent to the malicious sender sharing its keys with the wormhole attacker, allowing the sending side of the wormhole to place appropriate timestamps or location information on any packets sent by the malicious sender that are then tunneled by the wormhole attacker.

9.5.3. Comparison Between Geographic and Temporal Leashes

Temporal leashes have the advantage of being highly efficient, especially when used with TIK, as described in Section 9.4. Geographic leashes, on the other hand, require a more general broadcast authentication mechanism, which may result in increased computational and network overhead. Location information also may require more bits to represent, further increasing the network overhead.

Geographic leashes have the advantage that they can be used in conjunction with a radio propagation model, thus allowing them to detect tunnels through obstacles. Furthermore, geographic leashes do not require the tight time synchronization that temporal leashes do. In particular, temporal leashes cannot be used if the maximum range is less than $c\Delta$, where c is the speed of light and Δ is maximum clock synchronization error; geographic leashes can be used until the maximum range is less than $2\nu\Delta$, where ν is the maximum movement speed of any node.

To evaluate the practicality of geographic leashes, we consider a radio of range 300 m, maximum movement speed of 50 m/s, a relative positioning error of 3 m, and time synchronization error of 1 ms. Then $t_r - t_s \leq 2$ ms, since the propagation time is at most 1 ms and the time synchronization error is at most 1 ms. Then $d_{sr} \leq ||p_s - p_r|| + 100 \text{ m/s} \cdot 2 \text{ ms} + 3 \text{ m} = ||p_s - p_r|| + 3.2$ m. Since $||p_s - p_r||$ could be as much as 3 m, the effective transmission range of the network interface is reduced by at most 6.2 m.

To compare the effectiveness of geographic leashes and temporal leashes, we compare the distance derived using each approach: $d_{sr} \leq ||p_s - p_r|| + 2\nu \cdot (t_r - t_s + \Delta) + \delta$ for geographic leashes and $d_{sr} \leq c \cdot (t_r - t_s + \Delta)$ for temporal leashes. We use $\frac{d_{\max}}{c}$ to denote the maximum propagation time. Then the maximum error is bounded by $\delta + 2\nu(\frac{d_{\max}}{c} + 2\Delta) + \delta = 2\delta + 4\nu\Delta + 2\nu\frac{d_{\max}}{c}$ for geographic leashes, and by $2c\Delta$ for temporal leashes. Geographic leashes are then more effective when $\delta < c\Delta - 2\nu\Delta - \frac{\nu}{c}d_{\max}$. In general, ν is much smaller than c. Given sufficient computing power and network bandwidth, geographic leashes should be used when $\delta < c\Delta$, and temporal leashes should be used when $\delta \geq c\Delta$.

9.6. Related Work

Radio Frequency (RF) watermarking is another possible approach to providing the security described in this paper. Since we are aware of no published specific details, it is difficult to assess its security. If the radio hardware is kept secret, such as through tamper-resistant modules, some level of security can be provided against compromised nodes; however, if the radio band in which communications are taking place is known, then an attacker can attempt to tunnel the entire signal from one location to another.

It may be possible to modify existing intrusion detection approaches to detect a wormhole attacker; since the packets sent by the wormhole are identical to the packets sent by legitimate nodes, such detection would more easily be achieved jointly with hardware able to specify some sort of direction of arrival information for received packets. To the best of our knowledge, no work has been published regarding the possibility of using intrusion detection systems specifically to detect wormhole attackers.

TESLA generally chooses longer time intervals than TIK does, in order to reduce the amount of computation needed to authenticate a new key. As a result, TESLA is capable of functioning with much looser time synchronization than is required by TIK. Given a sufficient level of time synchronization, TIK provides an advantage over hop-by-hop authentication with TESLA, with respect to latency and packet overhead, but it suffers with respect to byte overhead. In particular, since TIK key disclosure always occurs in the same packet as the data protected, packets can be verified instantly; with TESLA, on the other hand, packets must wait, on average 1.5 time intervals, which is especially significant when packets are authenticated hop-by-hop, as may be required in a multi-hop ad hoc network routing protocol.

The IEEE 802.11i Task Group is designing modifications to IEEE 802.11 [84] to improve security. These modifications generally use a single shared key, or, when multiple keys are used, the keys are used between multiple clients and a single base station. Since base stations are not present in ad hoc networks, and since a single shared key does not prevent any attacks launched from a compromised node, these proposals do not sufficiently address authentication for ad hoc network routing. Furthermore, none of the current proposals within IEEE 802.11i address the wormhole attack.

Other Medium Access Control protocols also specify privacy and authenticity mechanisms. These mechanisms typically use one or more shared keys, allowing compromised nodes to forge packets. Furthermore, to the best of our knowledge, none of these mechanisms protect against wormhole attacks.

9.7. Conclusions

In this paper, we have introduced the *wormhole attack*, a powerful attack that can have serious consequences on many proposed ad hoc network routing protocols; the wormhole attack may also be exploited in other types of networks and applications, such as wireless access control systems based on physical proximity. To detect and defend against the wormhole attack, we introduced *packet leashes*, which may be either *geographic* or *temporal* leashes, to restrict the maximum transmission distance of a packet. Finally, to implement temporal leashes, we presented the design and performance analysis of a novel, efficient protocol, called TIK, which also provides instant authentication of received packets.

TIK requires just n public keys in a network with n nodes, and has relatively modest storage, per packet size, and computation overheads. In particular, a node needs to perform only between 3 and 6 hash function evaluations per time interval to maintain up-to-date key information for itself, and roughly 30 hash functions for each received packet. With commodity hardware such as 11 Mbps wireless links, TIK has computational and memory requirements that are easily satisfiable today; 2.6 megabytes for hash tree storage represents, for example, less than 3% of the standard memory on an Compaq iPaq 3870 with no external memory cards, and since the StrongARM CPU on the iPaq is capable of performing 222,000 symmetric cryptographic operations per second, TIK imposes no more than an 18% load on CPU time, even when flooded with packets at the maximum speed of the wireless network, and normally uses less CPU load than that in normal operation.

When used in conjunction with precise timestamps and tight clock synchronization, TIK can prevent wormhole attacks that cause the signal to travel a distance longer than the nominal range of the radio, or any other range that might be specified. Sufficiently tight clock synchronization can be achieved in a wireless LAN using commercial GPS receivers [183], and wireless MAN technology could be sufficiently time-synchronized using either GPS or LORAN-C [126] radio signals.

A MAC layer protocol using TIK efficiently protects against replay, spoofing, and wormhole attacks, and ensures strong freshness. TIK is implementable with current technologies, and does not require significant additional processing overhead at the MAC layer, since the authentication of each packet can be performed on the host CPU.

Our geographic leashes are less efficient than temporal leashes, since they require broadcast authentication, but they can be used in networks where precise time synchronization is not easily achievable. The dominant factor in the usability of geographic leashes is the ability to accurately measure location; because node movement is very slow relative to the speed of light, the effects of reduced time synchronization accuracy are slight.

Chapter 10

Rushing Attacks and Defense

An *ad hoc network* is a collection of mobile computers (or nodes) that cooperate to forward packets for each other to extend the limited transmission range of each node's wireless network interface. A routing protocol in such a network finds routes between nodes, allowing a packet to be forwarded through other network nodes towards its destination. In contrast to traditional network routing protocols, for example for wired networks, ad hoc network routing protocols must adapt more quickly, since factors such as significant node movement and changing wireless conditions may result in rapid topology change.

This problem of routing in ad hoc networks is an important one, and has been extensively studied. This study has resulted in several mature protocols [37, 90, 134, 139]. Ad hoc networks are targeted at environments where communicating nodes are mobile, or where wired network deployment is not present or not economical. Many of these applications may run in untrusted environments and may therefore require the use of a secure routing protocol. Furthermore, even when the presence of an attacker is not forseen, a secure ad hoc network routing protocol can also provide resilience against misconfigured nodes. In the current Internet, for example, misconfigured routing tables contribute to the majority of routing instabilities [109]. Similarly, a software or hardware failure should cause only the affected node to fail, and not perturb the stability of routing in the remainder of the network. Mission or safety-critical networks can use secure ad hoc routing protocols so that configuration errors, software bugs, or hardware failures do not disturb routing at other nodes. As a result, several secure ad hoc network routing protocols have been proposed [31, 78, 79, 136, 148, 170, 190].

In this paper, we present a new attack, the *rushing attack*, which results in denial-of-service when used against all previously published ad hoc network routing protocols. Specifically, *all* previously published secure on-demand routing protocols are unable to find routes longer than two-hops when subject to this attack.

Because on-demand protocols generally have lower overhead and faster reaction time than other types of routing based on periodic (proactive) mechanisms, on-demand protocols are better suited to many applications of ad hoc networks. To defend this important class of protocols against the rushing attack, we develop a generic secure Route Discovery component, called *Rushing Attack Prevention (RAP)*, that can be applied to any existing on-demand routing protocol to allow that protocol to resist the rushing attack.

Our main contributions in this paper are the presentation of this new attack, the development and analysis of our new secure Route Discovery component that demonstrates that it is possible to secure against the rushing attack, and a general design that uses this component to secure any on-demand Route Discovery mechanism against the rushing attack.

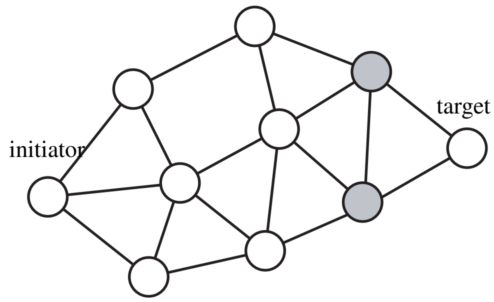


Figure 10.1: Example network illustrating the rushing attack

10.1. The Rushing Attack against Ad Hoc Network Routing Protocols

We introduce here a new attack, which we call *the rushing attack*, that acts as an effective denial-of-service attack against all currently proposed on-demand ad hoc network routing protocols, including protocols that were designed to be secure. In an on-demand protocol, a node needing a route to a destination floods the network with ROUTE REQUEST packets in an attempt to find a route to the destination. To limit the overhead of this flood, each node typically forwards only one ROUTE REQUEST originating from any Route Discovery. In particular, existing on-demand routing protocols, such as AODV [139], DSR [90], LAR [99], Ariadne [79], SAODV [190], ARAN [170], AODV secured with SUCV [31], and SRP [136], only forward the REQUEST that arrives *first* from each Route Discovery. In the rushing attack, the attacker exploits this property of the operation of Route Discovery.

We now describe the rushing attack in terms of its effect on the operation of DSR Route Discovery [87, 88, 90]; other protocols such as AODV [141], Ariadne [79], SAODV [190], and ARAN [170] are vulnerable in the same way. In the network shown in Figure 10.1, the initiator node initiates a Route Discovery for the target node. If the ROUTE REQUESTs for this Discovery forwarded by the attacker are the first to reach each neighbor of the target (shown in gray in the figure), then any route discovered by this Route Discovery will include a hop through the attacker. That is, when a neighbor receives the rushed REQUEST from the attacker, it forwards that REQUEST, and will not forward any further REQUESTs from this Route Discovery. When non-attacking REQUESTs arrive later at these nodes, they will discard those legitimate REQUESTs. As a result, the initiator will be unable to discover any usable routes (i.e., routes that do not include the attacker) containing at least two hops (three nodes).

In general terms, an attacker that can forward ROUTE REQUESTs more quickly than legitimate nodes can do so, can increase the probability that routes that include the attacker will be discovered rather than other valid routes. Whereas the discussion above has used the case of nodes that forward only the *first* ROUTE REQUEST from any Route Discovery, the rushing attack can also be used against any protocol that predictably forwards *any* particular REQUEST for each Route Discovery.

A rushing attacker need not have access to vast resources. On-demand routing protocols delay ROUTE REQUEST forwarding in two ways. First, Medium Access Control (MAC) protocols generally impose delays between when the packet is handed to the network interface for transmission and when the packet is actually transmitted. In a MAC using time division, for example, a node must wait until its time slot to transmit, whereas in a MAC using carrier-sense multiple access, a node generally performs some type of backoff to avoid collisions; protocols like IEEE 802.11 also impose an interframe spacing time before transmission actually begins. Second, even if the MAC layer does not specify a delay, on-demand protocols generally specify a delay between receiving a REQUEST and forwarding it, in order to avoid collisions of the REQUEST packets. In particular, because REQUEST packets are broadcast, and collision detection for broadcast packets is difficult, routing protocols often impose a randomized delay in REQUEST forwarding. An attacker ignoring delays at either the MAC or routing layers will generally be preferred to similarly situated non-attacking nodes. One way to thwart an attacker that rushes in this way is to remove these delays at both the MAC and routing layers, but this approach does not work against all types of rushing attackers and is not general. For example, in a dense network using a CSMA MAC layer, if a node **A** initiates a Route Discovery, and **B** is two hops away from **A**, and **C** and **D** are neighbors of both **A** and **B**, then then **B** will likely not receive the ROUTE REQUEST due to a collision between REQUESTs forwarded by **C** and **D**. In a dense network, such collisions may often prevent the discovery of any nontrivial routes (routes longer than a direct link), which is even more severe than the rushing attack, which prevents the discovery of routes longer than two hops.

Another way that a relatively weak attacker can obtain an advantage in forwarding speed is to keep the network interface transmission queues of nearby nodes full. For example, if each node processes the packets it receives in order, and an inefficient REQUEST authentication mechanism is used, the attacker can keep other nodes busy authenticating REQUESTs containing bogus authentication, thus slowing their ability to forward legitimate REQUESTs. Protocols employing public key techniques are particularly susceptible to these attacks, since they require substantial computation to validate each received REQUEST.

A relatively weak attacker can also achieve faster transit of its REQUEST packets by transmitting them at a higher wireless transmission power level, thus reducing the number of nodes that must forward that REQUEST to arrive at the target. Since packet transit time at each hop is dominated by the processing time at the forwarding node, reducing the path to the target by just one hop is likely to provide a significant latency advantage, thus strengthening the attackers position.

A more powerful rushing attacker may employ a wormhole [81] to rush packets. In this case, the attacker simply forwards all control packets (but not data packets) received at one node (the attacker) to another node in the network (e.g., a second attacker). This forms a tunnel in the network, where packets reaching one end of the tunnel are broadcast out the other end. If the tunnel provides significantly faster transit than legitimate forwarders, nodes near one end of the tunnel generally will be unable to discover working routes to the other end of the tunnel, since it will generally discover routes through the tunnel. In general, a wired tunnel (in which the two attackers have a wired connection between themselves) will provide faster transit than native wireless (multihop) forwarding, since node processing delay in forwarding is much longer than the propagation time.

The rushing attack applies to all proposed on-demand protocols because such protocols must limit the number of packets that any node will transmit in response to a single Route Discovery. Currently proposed protocols choose to forward at most one REQUEST for each Discovery; any protocol that allows an attacker to predict which ROUTE REQUEST(s) will be chosen for forwarding at each hop will be vulnerable to some variant of the rushing attack.

10.2. Assumptions

10.2.1. Network Assumptions

We make the common assumption that most network links are bidirectional. More specifically, we require that the network remain connected when unidirectional links are ignored. Our Secure Neighbor Detection protocol rejects unidirectional links, so underlying routing protocols can assume that the network is free of unidirectional links. If another Secure Neighbor Detection technique is used, and that technique supports unidirectional links, then the ability of our Secure Route

Discovery mechanism to discover and use unidirectional links is limited only by the underlying routing protocol.

Wireless physical layers for sending data from one node to another are often vulnerable to jamming. Mechanisms such as spread spectrum modulation [149], or directional antennas have been extensively studied as means of improving resistance to physical jamming. In addition, an effective jamming attack usually requires additional hardware; in contrast, a rushing attack is much simpler to do because the attacker can use the same hardware as legitimate nodes. An attacker can even remotely break into a legitimate node and perform these attacks. Moreover, the rushing attack allows for far more selective denial-of-service, and is thus harder to detect. Jamming attacks are relatively broad (they deny service to a large number of participants) and are thus also easier to detect. Though a jamming attack is also an important denial-of-service attack, we present mechanisms to defend against the rushing attack because we believe that the rushing attack is more easily performed.

Medium Access Control protocols are also often vulnerable to attack. For example, in IEEE 802.11, an attacker can paralyze nodes in its neighborhood by sending Clear-To-Send (CTS) frames periodically, setting the "Duration" field of each frame equal to the interval between such frames [79]. Less sophisticated Medium Access Control protocols, such as ALOHA and Slotted ALOHA [1], are not vulnerable to such attacks but have lower efficiency. In this paper, we disregard attacks on Medium Access Control protocols.

Prior work has shown that ad hoc network routing in general does not scale well [61]. Most existing simulation of ad hoc network routing protocols consider scenarios of 50 to 500 nodes. In this work, we focus on such medium-sized networks, and will not consider scalability issues; however, we believe that mechanisms such as clustering, which improve the scalability of other on-demand ad hoc network routing protocols, can also improve the scalability of our approach.

10.2.2. Security Assumptions and Key Setup

The protocols discussed in this paper require an instantly-verifiable broadcast signature. However, any signature used should be able to keep up with verification at line speed, to avoid a denial-of-service attack where an attacker floods the victim with bogus messages and thus overwhelms the victim. One example of a protocol which should be fast enough on many nodes is the the HORS broadcast authentication protocol by Reyzin and Reyzin [160], when used in conjunction with a Merkle hash tree [122] to generate one signature over multiple messages. As used in our simulation evaluation, HORS requires an average of $156,760$ hashes per second to sign and verify all messages in a 100 node network, a rate easily achievable even by PDAs. We assume that the keys necessary for broadcast authentication are distributed in advance; a number of techniques for distributing such information have been proposed [9, 79, 82, 178]. To escape the circular dependency of secure routing and key distribution, Hu et al propose a simple routing protocol that discovers a route to a trusted third party, which can in turn bootstrap the initial keys [79].

If a wormhole attack, in which an attacker selectively tunnels packets from one place in the network to another, is considered a possible threat, our Secure Neighbor Detection requires a mechanism to detect such a tunnel between any two legitimate nodes. A number of mechanisms for preventing the wormhole attack, such as TIK, geographical leashes and RF watermarking, have been proposed. Depending on the mechanism used to implement packet leashes, this requirement benefit other parts of the protocol: TIK [81], for example, authenticates each packet in a lightweight manner, thus protecting the more expensive signature verification from a denial-of-service attack. In particular, if a node **A** receives an authenticated packet containing a bogus signature from node **B**, then **A** can lower the priority with which it checks signatures sent by **B**. As a result, an attacker can only cause each node to verify one bogus signature for each node compromised by that attacker.

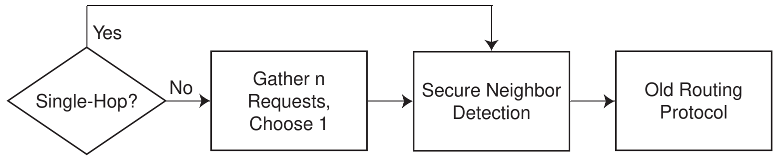


Figure 10.2: Our design to secure a protocol against the rushing attack

We do not assume tamper-proof hardware; the attacker can thus compromise nodes and steal their cryptographic keys. We assume a powerful attacker, which we call *coordinated attacker*. This is an attacker that compromised multiple nodes (and thus knows all their cryptographic keys), with a fast channel to route packets amongst themselves.

10.3. Secure Routing Requirements and Protocol

In this section, we describe a generic protection mechanism that prevents the extremely powerful rushing attack. We also describe a technique to secure any protocol using an on-demand Route Discovery mechanism. Our design is shown in Figure 10.2.

10.3.1. Notation

We use the following notation:

- A or B denote communicating nodes.
- $A : \eta \stackrel{R}{\leftarrow} \{0,1\}^{\ell}$ denotes that node A randomly selects an ℓ-bit long nonce η.
- $A \rightarrow B : \langle M, H(A \mid\mid \eta)\rangle$ means that node A sends B the message M and the hash of A's identifier concatenated with the nonce η.
- $A \rightarrow * : \langle M, \Sigma_M\rangle$ means that node A broadcasts message M with its signature Σ_M.

10.3.2. Secure Neighbor Detection

The functionality of *Neighbor Detection*, in which two nodes detect a bidirectional link between themselves, is present in almost every routing protocol. For example, a node participating in a periodic protocol generally broadcasts advertisements, allowing its neighbors to detect it. Most on-demand routing protocols, on the other hand, perform Neighbor Detection implicitly. In those protocols, a node receiving a ROUTE REQUEST considers itself to be a neighbor of the transmitter. When that node propagates the REQUEST, it claims a link between the transmitter and the recipient. Unfortunately, this does not provide acceptable security; a node receiving a REQUEST can simply replay it. In addition, the address of the previous node is unauthenticated, so an attacker can claim to be any node propagating a REQUEST, and the next hop will trust that information.

Requirements for Secure Neighbor Detection. Two nodes detect themselves as neighbors only if they are within transmission range. The secure Neighbor Detection protocol thus prevents an attacker from: 1) introducing two nodes that are not in transmission range as neighbors; and 2) claiming that it is a neighbor of another node without being able to hear packets directly from that node. From the first requirement it follows that an attacker should not be able to *tunnel* a neighbor solicitation from one compromised node to another uncompromised node. The second requirement

$$\begin{array}{ll}
S: & \eta \stackrel{R}{\leftarrow} \{0,1\}^{\ell} \\
 & M_1 = \langle \text{NEIGHBOR SOLICITATION}, S, \eta \rangle \\
 & \Sigma_{M_1} = \mathsf{Sign}(M_1) \\
S \rightarrow *: & \langle M_1, \Sigma_{M_1} \rangle \\
R: & M_2 = \langle \text{NEIGHBOR REPLY}, R, H(R \,||\, \eta) \rangle \\
 & \Sigma_{M_2} = \mathsf{Sign}(M_2) \\
R \rightarrow S: & \langle M_2, \Sigma_{M_2} \rangle \\
S: & M_3 = \langle \text{NEIGHBOR VERIFICATION}, S, R, H(\eta) \rangle \\
 & \Sigma_{M_3} = \mathsf{Sign}(M_3) \\
S \rightarrow R: & \langle M_3, \Sigma_{M_3} \rangle
\end{array}$$

Figure 10.3: Neighbor detection between S and R. M_2 can be authenticated using a shared key, if available.

demands that a node needs to hear the neighbor solicitation, otherwise it cannot claim to be a neighbor. Finally, the protocol should not introduce a denial-of-service opportunity, for example by flooding nodes with neighbor requests.

A Generic Secure Neighbor Detection Protocol. Our secure Neighbor Detection protocol requires three messages. First, the soliciting node sends a New Neighbor Solicitation packet, either by unicasting that packet to a specific neighbor, or by broadcasting the packet. Next, a node receiving the New Neighbor Solicitation packet sends a New Neighbor Reply packet. Finally, the node initiating the Neighbor Detection sends a Neighbor Verification, which includes broadcast authentication of a timestamp and the link from the source to the destination. If wormhole attacks are considered a possible threat, we build our secure Neighbor Detection protocol above a mechanism, such as packet leashes, which prevent wormholes between legitimate nodes.

To prevent a node from deriving a well-formed Neighbor Reply based on the Neighbor Reply from another node, we include a nonce η in each New Neighbor Solicitation, and require the Neighbor Reply to include the hash of the nonce and the source address. To prevent forgery of New Neighbor Solicitation and New Neighbor Reply packets, we require some form of authentication. If desired, a packet leash can be added to the New Neighbor Solicitation and New Neighbor Reply packets to prevent wormhole attacks. Finally, we rate-limit New Neighbor Solicitations to prevent an attacker from flooding its neighbors. Figure 10.3 shows the full protocol.

Integration with an On-Demand Protocol. In an on-demand protocol, neighbor verification is performed during each Route Discovery. As a result, we can defend against New Neighbor Solicitation floods, by relying on the underlying protocol to defend against ROUTE REQUEST floods; a node responds to any New Neighbor Solicitation presented with a valid REQUEST. If desired, REQUEST flood prevention can be achieved through the use of a hash chain, as in Ariadne [79].

When a node **A** forwards a REQUEST, it includes in that REQUEST a broadcast Neighbor Solicitation. Each node **B** forwarding that REQUEST returns a Neighbor Reply, and piggybacks on the Neighbor Reply a unicast Neighbor Solicitation for **A**. If **A** decides that **B** is a neighbor based on the wormhole prevention mechanism used, **A** returns a signed Neighbor Verification that verifies the link from **A** to **B**. **A** also includes in packet a Neighbor Reply to the unicast Neighbor Solicitation sent by **B**. If **B** decides that **A** is a neighbor based on the wormhole prevention mechanism used, **B** forwards the REQUEST, including the Neighbor Verification for the $A \rightarrow B$ link signed by A, and also including a Neighbor Verification for the $B \rightarrow A$ link signed by itself. **B** need not return a Neighbor Verification, since **A** is likely to hear the forwarded REQUEST, which includes the $B \rightarrow A$ Neighbor Verification. Figure 10.4 shows how **B** forwards a REQUEST from **A**.

$$
\begin{array}{ll}
A: & \eta_A \xleftarrow{R} \{0,1\}^\ell \\
 & M_{1a} = \langle \text{ROUTE REQUEST} \ldots \rangle \\
 & M_{1b} = \langle \text{NEIGHBOR SOLICITATION}, A, \eta_A \rangle \\
 & \Sigma_{M_1} = \mathsf{Sign}(M_{1a} \mid\mid M_{1b}) \\
A \to *: & \langle M_{1a}, M_{1b}, \Sigma_{M_1} \rangle \\
B: & \eta'_B \xleftarrow{R} \{0,1\}^\ell \\
 & M_{2a} = \langle \text{NEIGHBOR REPLY}, B, H(B \mid\mid \eta_A) \rangle \\
 & M_{2b} = \langle \text{NEIGHBOR SOLICITATION}, B, \eta'_B \rangle \\
 & \Sigma_{M_2} = \mathsf{Sign}(M_{2a} \mid\mid M_{2b}) \\
B \to A: & \langle M_{2a}, M_{2b}, \Sigma_{M_2} \rangle \\
A: & M_{3a} = \langle \text{NEIGHBOR VERIFICATION}, A, B, H(\eta_A) \rangle \\
 & M_{3b} = \langle \text{NEIGHBOR REPLY}, A, H(A \mid\mid \eta'_B) \rangle \\
 & \Sigma_{M_3} = \mathsf{Sign}(M_{3a} \mid\mid H(M_{3b})) \\
A \to B: & \langle M_{3a}, M_{3b}, \Sigma_{M_3} \rangle \\
B: & \eta_B \xleftarrow{R} \{0,1\}^\ell \\
 & M_{4a} = \langle \text{ROUTE REQUEST} \ldots \rangle \\
 & M_{4b} = \langle \text{NEIGHBOR SOLICITATION}, B, \eta_B \rangle \\
 & M_{4c} = \langle \text{NEIGHBOR VERIFICATION}, B, A, H(\eta_A) \rangle \\
 & \Sigma_{M_4} = \mathsf{Sign}(H(M_{4a}) \mid\mid H(M_{4b}) \mid\mid M_{4c}) \\
B \to *: & \langle M_{4a}, M_{4b} \Sigma_{M_1} \rangle
\end{array}
$$

Figure 10.4: **B** forwarding the REQUEST from **A**. Σ_{M_2} can be generated using a shared key, if available. The ROUTE REQUESTin M_{4a} includes the bidirectional Neighbor Verification messages M_{3a} and M_{4c}, together with the necessary authenticators ($H(M_{3b})$ and Σ_{M_3}). The use of $H(M_{3b})$ in Σ_{M_3} allows the verification of M_{3a} without needing M_{3b}, which decreases the overhead caused by the REQUEST packet. The same technique is used in creating Σ_{M_4}.

10.3.3. Secure Route Discovery

In this section, we describe how Secure Route Discovery uses Secure Neighbor Discovery to prevent the rushing attack. The intuition behind Secure Route Discovery is to make the forwarding of REQUEST packets less predictable by buffering the first n REQUESTs received, then randomly choosing one of those REQUESTs. However, we need to prevent an attacker from filling too many of these n REQUESTs, since otherwise the attacker could simply rush n copies of a REQUEST, rather than a single REQUEST, and our scheme would once again be vulnerable to the rushing attack.

To limit the number of REQUESTs that traverse an attacker, we exploit the fact that legitimate nodes forward only one REQUEST in any Discovery. First, we require that each REQUEST carry a list of nodes traversed by this REQUEST. Second, we require a bidirectional Neighbor Verification for each link represented by this list of nodes, for a total of two signed Neighbor Verifications per hop. Third, to authenticate the node list, we require each node to authenticate the REQUEST it forwards, though it can piggyback this authentication together with the Neighbor Verification that it signs. Finally, we require buffered REQUESTs be duplicate-suppression-unique: that is, if the route record of any two REQUESTs contain any node A, the route prefix leading up to (and including) A must be the same. These three requirements constrain an attacker to the extent that an attacker that has compromised m nodes can rush at most m REQUESTs.

To prevent replay of old Neighbor Verification messages, each message is tied to a specific Route Discovery. Specifically, when a node **S** sends a Neighbor Verification for the link from **S** to **R**, **S** signs not just **S** and **R** (as in Figure 10.3), but also ties a unique Route Discovery identifier to the Neighbor Verification. For example, in AODV, the RREQ ID and Originator IP Address in an RREQ form a unique identifier; in DSR, the Target Address and Identifier fields from a ROUTE REQUEST, together with the IP Source Address, form a unique identifier. To address wraparound in these Identifier fields, if the nodes in the network have very loosely synchronized clocks (within a few days), the node can include a timeout in addition to this unique identifier. If network nodes have more tightly synchronized clocks (within a few seconds), the node can include a timeout in place of any unique identifier.

In some areas of some networks, a node will not have n distinct paths to the source of the REQUEST. To enable the Discovery of routes to or through such nodes, we allow a node to forward a REQUEST after some time, even if it has not yet received n REQUESTs. In certain cases, however, a fixed timeout allows an attacker to prevent the discovery of a correct route. One way to avoid such an attack is to choose a random timeout between $t_{\min}$ and $t_{\max}$. Alternatively, we can prefer early release when a node has buffered more REQUESTs, for example by choosing a random timeout between $t_{\min} + (n - j)t_{\text{add}}$ and $t_{\max} + (n - j)t_{\text{add}}$, where j is the number of REQUESTs buffered so far. Choosing a timeout when location information is available can provide better properties. If the initiator of each REQUEST includes a timestamp t and its location, intermediate nodes can choose a timeout of t + fixed timeout + propagation speed $\cdot$ distance to initiator. After a node chooses a timeout, either randomly or based on optional location information, the node randomly chooses one received REQUEST for forwarding.

We implement two additional security optimizations to this basic scheme. In general, these optimizations are based on using the property of nonrepudiation to spread information about malicious nodes. First, we require that each REQUEST be signed by the forwarding node. A node detecting an attacker forwarding more than one REQUEST can expose the attacker by flooding the two REQUESTs. Second, if location information is available, and used for example to implement geographic packet leashes, an attacker claiming to be in two places at the same time can be blacklisted in the same way. For example, if each REQUEST includes in the node list location information and time information for each forwarding node, a node can keep a database of previous location information, and find two location claims that significantly exceed the maximum speed achievable by legitimate nodes. In particular, if location information is accurate to δ, and time information is consistent to Δ, and maximum speed is ν, then two locations claimed t time apart is maliciously claimed if the distance between the two locations is greater than $2\delta + \nu(t + 2\Delta)$. Our blacklist mechanisms do not need authentication, since the nonrepudiation of contradicting information can be can be verified by any nodes. We route blacklist information by flooding: contradictory information is rebroadcast by any node that verifies the nonrepudiation and did not have this malicious node on its blacklist. This approach is similar to the blacklist mechanism used by Ariadne [79].

10.3.4. Integrating Secure Route Discovery with DSR

To integrate rushing prevention with DSR [87] or other secure protocols based on DSR, we limit Route Discovery frequency as in Ariadne [79]. Each time a node forwards a ROUTE REQUEST, it first performs a Secure Neighbor Detection exchange with the previous hop. When it forwards the REQUEST, it includes in the REQUEST a bidirectional Neighbor Verification for the previous hop.

As in DSR, the target of a Route Discovery returns a ROUTE REPLY for each distinct ROUTE REQUEST it receives. Each such ROUTE REPLY is sent with a source route selected by reversing the route in the ROUTE REQUEST. This route is likely to work if there are no attackers on the route, since Neighbor Detection only finds bidirectional neighbors.

10.3.5. Integrating Secure Route Discovery with AODV

In AODV [141], as well as other secure protocols based on AODV [31, 170, 190], Route Request (RREQ) packets do not carry a node list. However, in order to filter excessive malicious RREQs, we require each RREQ to carry a node list. Instead of forwarding the first RREQ received, nodes using our Secure Route Discovery randomly select one of the first n RREQs it receives and treats it as the RREQ to forward. More specifically, it places the initiator of the Route Discovery in its routing table using the previous hop of the RREQ selected as the next-hop destination. It then appends its address and authentication information to the node list, and forwards it as in DSR.

Since AODV is a distance-vector protocol, it cannot make use of multiple routes. As a result, the target of a Route Discovery also waits for n RREQ packets before returning a single RREP. The target signs the RREP, and includes in the RREP neighbor authentication for each hop in the chosen path. This authentication allows nodes forwarding the RREP to authenticate the entire path back to the source of the RREP. Each node authenticating this information establishes a route back to the source of the RREP (the target of the RREQ). When this RREP reaches the destination, it will have established a bidirectional route between the initiator and target of the Route Discovery.

Because AODV does not support multiple routes, the security properties of AODV using Secure Neighbor Discovery will be somewhat worse than the properties of DSR using Secure Neighbor Discovery.

10.3.6. Integrating Secure Route Discovery with Secure Ad Hoc Network Routing Protocols

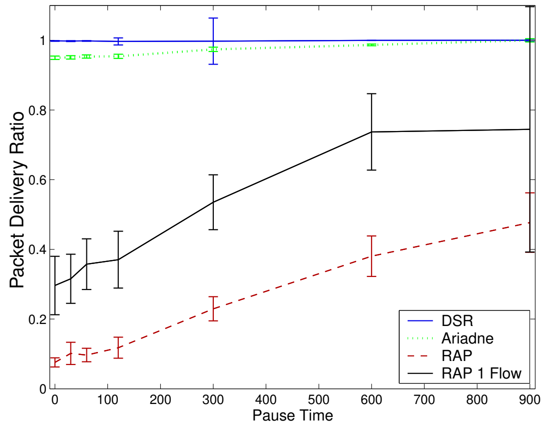
When using our rushing attack prevention together with a secure on-demand routing protocol, a node can first attempt Route Discovery using that secure protocol. If a rushing attacker prevents the discovery of any working routes, the node can then set a flag indicating that it wants to use routing attack prevention, though it must also authenticate that flag to prevent modification. This approach is similar to the principle of *expanding ring search*: first, a node uses a cheaper, but sometimes unsuccessful, search. The node only uses a more expensive search when the cheaper search does not find a route. This optimization provides benefits in two cases: first, when there are no rushing attackers, existing secure routing protocols should be able to find a route. Secondly, a rushing attacker does not have any advantage in one- and two-hop routes.

10.4. Evaluation

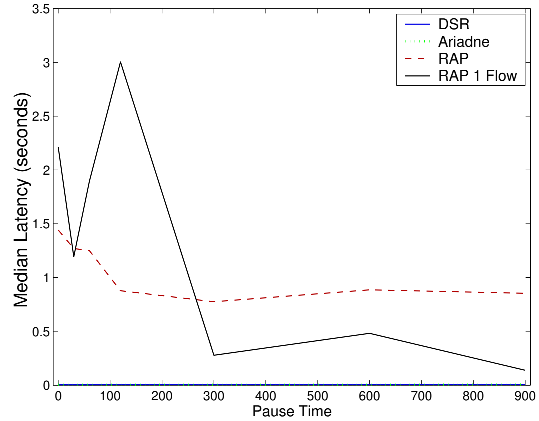
To evaluate our techniques, we analyzed the cost and effectiveness through simulation and analysis. Our simulation was designed to show the cost of our techniques in a non-adversarial environment, whereas our analytical evaluation shows provable bounds on the extent to which an attacker can disrupt a protocol using our techniques.

10.4.1. Simulation Evaluation

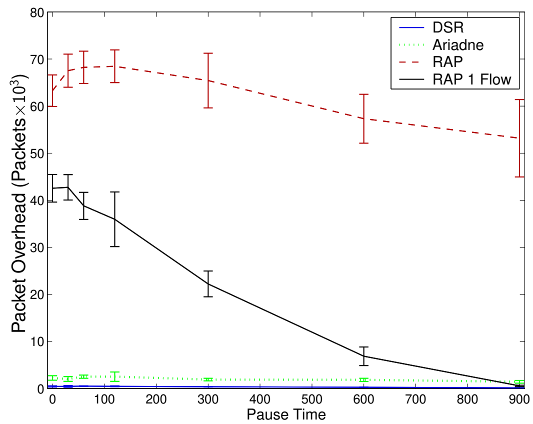
To evaluate the overhead of using our secure neighbor discovery mechanism in a non-adversarial environment, we simulated our scheme using the *ns-2* simulator, using Ariadne [79] as our underlying routing protocol. We call this modified protocol RAP (Rushing Attack Prevention). We did not implement the optimizations described in Section 10.3.6, because our simulations did not include an attacker, so our results would be equivalent to just using Ariadne. We used the original Ariadne source code [128], and modified it to use digital signatures based on HORS and geographical leashes for wormhole protection [81]. We compared our results with Ariadne and DSR in order to determine the added costs of RAP when there are no attackers. However, when a rushing attacker is present, existing on-demand ad hoc network routing protocol would in general be unable to deliver packets over paths longer than two hops (Section 10.1). RAP, on the other hand, would be able to discover



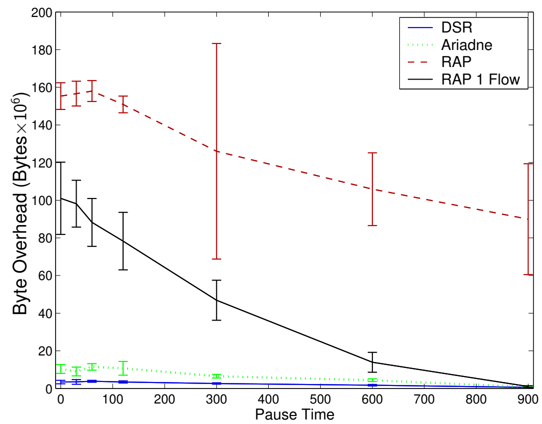
(a) Packet Delivery Ratio



(b) Median Latency



(c) Packet Overhead



(d) Byte Overhead

Figure 10.5: Unoptimized RAP performance evaluation results in non-adversarial environment. Optimized RAP would have same results as Ariadne, except that it would perform better when under attack. Under attack, optimized RAP and Ariadne would perform identically for one- and two-hop routes, but in finding longer routes, RAP should significantly outperform Ariadne, since RAP finds working routes with moderate probability, but Ariadne and DSR can never find routes. "RAP 1 Flow" refers to RAP with the lighter communications pattern of one CBR source. Results based on averages over 50 simulation runs; the error bars represent the 95% confidence interval of the mean.

working paths much of the time, and as a result, would generally outperform existing on-demand routing protocols.

We chose HORS as our broadcast signature, using a time interval of 5 seconds and allowing each node to authenticate up to 20 messages per time interval. We assumed a time synchronization error of 1 second, and used 180 byte signatures. As a result, each public key is 78380 bytes, and each node has an amortized workload of 156760 hash operations per second at each node for generating signatures, as well as verifying all signatures from all nodes. (This level is well within the capability of modern PDAs, and represents around 10% CPU utilization on modern workstations). Our parameters were chosen to provide an 80 bit security level; that is, an attacker must guess 2^{80} signatures to forge one signature in expectation. When signatures were needed at a faster rate than permitted by HORS, we used a multi-signature scheme based on Merkle hash trees [122]. We simulated packet leashes based on optional location information, and waited for 2 REQUEST packets, or a 0.2 seconds fixed timeout plus the distance to the initiator times a propagation speed of 1500 meters per second.

Because a square area is more likely to support multiple routes between a source an a destination, our simulations used 100 nodes in a 1000 m $\times$ 1000 m space moving according to the *random waypoint model* [88]. In this model, each node is randomly placed; at the beginning of the simulation, it waits for a *pause time*, then chooses a velocity uniformly between 0 and 20 meters per second. It then proceeds to a random location at that velocity, and upon arriving, waits for the pause time and repeats. We simulated pause times of 0, 30, 60, 120, 300, 600, and 900 seconds.

We chose a workload of 5 flows, each producing 4 packets per second, using 64-byte packets. This workload was sufficient to cause significant congestion with our scheme, even though normal ad hoc network routing protocols can deliver four or more times the load at lower loss rate; however, secure neighbor discovery incurs significantly higher overhead due to the four-way handshake and speed-of-light delays associated with it. We simulated a link-layer data rate of 2 Mbps.

RAP has significantly worse performance than both Ariadne and DSR because of the added load of the Secure Neighbor Discovery. Figure 10.5(a) shows the Packet Delivery Ratio of the three protocols. DSR delivers between 99.8% and 100% of offered traffic. Ariadne delivers between 95.0% and 100% of offered traffic; a significant improvement over previous simulation results [79]. This suggests that previous simulations used too high a traffic load to fairly evaluate Ariadne in the absence of congestion. Even with this light traffic load, RAP was able to deliver just 7.6% to 47.7% of offered load. This performance is primarily due to congestion. At higher movement speeds (lower pause time), the lower packet delivery ratio is caused by an even higher packet overhead, which results from the on-demand nature of the protocol. We also simulated RAP carrying a lower load of just one flow. At higher pause times, Ariadne with RAP has sufficiently low overhead to deliver between 73.7% and 74.5% of traffic. Even with these pause times, 92.1% of drops were due to MAC-layer congestion, compared to just 4.15% due to the node's inability to find a route. This MAC-layer congestion severely hampers our protocol's ability to deliver application-layer packets.

Figure 10.5(b) shows the median latency of delivered packets. DSR and Ariadne appear to have zero mean latency, since their median latencies of 4.3ms and 3.8ms respectively are significantly lower than the 1050ms median latency of RAP. Two factors contribute to the higher latency of RAP: first, congestion increases the time each node must wait to acquire the medium, and second, if a node receives just one ROUTE REQUEST packet from a Route Discovery, it waits a significant amount of time before forwarding that REQUEST in an attempt to collect enough REQUESTs and choose one at random.

Figures 10.5(c) and 10.5(d) show the Packet Overhead and Byte Overhead of the three protocols. At higher pause times, RAP has more than five times as much overhead when it uses five flows. This indicates that the congestion caused by the protocol significantly reduces the usefulness of the routing protocol packets. When congestion is not an issue, we actually expect that overhead should be less than a factor of five, because nodes can cache information they overhear, thus improving efficiency.

Our performance evaluation shows that in non-adversarial environments, RAP adds significant costs relative to other secure routing protocols. Many of these costs are due to the congestion created at lower bit rates. However, RAP is designed to be used only when necessary (Section 10.3.6), so these higher costs are only incurred when the underlying protocol is otherwise unable to discover a working route. Specifically, RAP incurs *no cost* until the underlying protocol is completely prevented from finding a working route. It then allows that protocol to use a higher cost approach to successfully deliver packets even against a rushing attacker. In the next section, we show how RAP performs under a rushing attack, in which DSR and Ariadne would be unable to find routes containing more than three nodes (two hops).

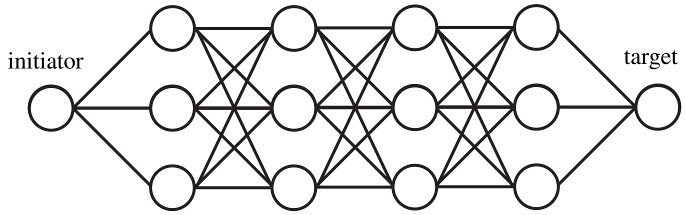


Figure 10.6: Example network topology used in RAP security analysis

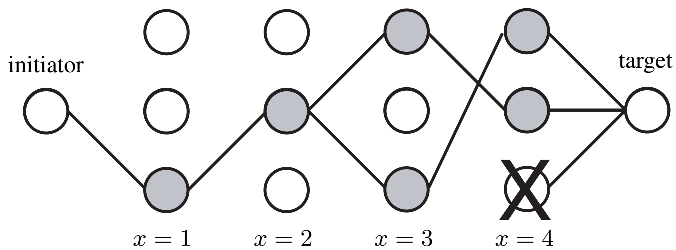


Figure 10.7: An example of a successful Route Discovery. Each gray node chose a valid REQUEST and belonged to a route for which a REPLY was sent. Each line represents a hop in a path chosen by a legitimate REQUEST; the network topology is shown in Figure 10.6.

10.4.2. Security Analysis

This section discusses the security properties achieved with RAP when n distinct routes (both legitimate and attacking) exist between the originator and each other node in the network. (As in Section 10.3.3, two routes are considered distinct if they end in different nodes.) Since routes are required to end in different nodes, an attacker with access to the keys of m compromised nodes can generate at most m distinct, maliciously injected ROUTE REQUESTs for the purpose of denial-of-service.

To analyze the probability of a node subverting a Route Discovery, we assume that the attacker rushes m distinct REQUESTs to each node in the network. As a result, each node needs only $n - m$ additional distinct REQUESTs. We also suppose that the network topology of these legitimate requests is represented by Figure 10.6, such that the ℓ hops from the source to the target form a sequence of tiers, such that the $n-m$ neighbors of the source form the first tier, the $n-m$ neighbors of the target form the last tier, and any two adjacent tiers form a complete, bipartite graph.

We denote the probability of successfully finding a route at tier x given y nodes at that tier to be $S_{x,y}$. In particular, we seek the probability $S_{\ell,n-m}$. Since one-hop neighbors cannot be subverted, $S_{1,y} = 1$ for all $y > 0$. At any other level (that is, when $x \neq 1$), the probability that i of the y neighbors will choose one of the m bogus ROUTE REQUESTs is given by the binomial PDF $\binom{y}{i}\left(\frac{m}{n}\right)^{y-i}\left(\frac{n-m}{n}\right)^{i}$. For example, in Figure 10.7, at $x = 4$, $y = 3$ nodes received a valid ROUTE REPLY, but only $i = 2$ of them forwarded a valid REQUEST.

Each of the i nodes that do not choose bogus REQUESTs chooses one of the REQUESTs it received. Some of these REQUESTs may overlap; the probability of choosing exactly j distinct previous hops is given by $p_{n-m-j}(n-m, i)$, where $p_{r-j}(r, i)$ is the probability that when i balls are thrown into r boxes, exactly $r - j$ boxes are empty (that is, exactly j boxes are full). The solution to the classical occupancy problem [186] gives $p_{r-j}(r, i) = \binom{r}{r-j} \sum_{k=0}^{j} (-1)^k \binom{j}{k} \left(\frac{j-k}{r}\right)^i$. For example, in Figure 10.7, at $x = 4$, $i = 2$ nodes chose $j = 2$ distinct previous hops, and at $x = 3$, $i = 2$ nodes chose $j = 1$ distinct previous hops.

When, at a level $x \neq 1$, i nodes do not choose bogus REQUESTs but instead choose a total of j distinct, legitimate REQUESTs, the probability that the Route Discovery will be successful is $S_{x-1,j}$ by definition. Then

$$\begin{aligned} S_{x,y} &= \sum_{i=1}^{y} \left[\binom{y}{i} \left(\tfrac{m}{n}\right)^{y-i} \left(\tfrac{n-m}{n}\right)^{i} \sum_{j=1}^{\min(i,n-m)} S_{x-1,j} p_{n-m-j}(n-m,i) \right] \\ &= \sum_{i=1}^{y} \left[\binom{y}{i} \left(\tfrac{m}{n}\right)^{y-i} \left(\tfrac{n-m}{n}\right)^{i} \sum_{j=1}^{\min(i,n-m)} \left\{ S_{x-1,j} \binom{n-m}{n-m-j} \sum_{k=0}^{j} (-1)^k \binom{j}{k} \left(\tfrac{j-k}{n-m}\right)^{i} \right\} \right] \end{aligned}$$

For example, when $n = 6$, $m = 2$ and $\ell = 5$, the probability of a successful Route Discovery is 46%.

We now argue that the case above reflects a worst case analysis by analyzing some potential variations. First, the $n - m$ additional incoming nodes could come from earlier tiers (e.g., tiers with lower x). However, since $S_{x,y}$ is monotone decreasing with increasing x and fixed y, the opportunity to choose nodes from earlier tiers only provides a benefit. Second, there may not be as much overlap between the predecessors of the nodes in a single tier; however, this only reduces the number of collisions at the previous tier. Fewer collisions at the previous tier improves performance, since $S_{x,y}$ is monotone increasing with fixed x and increasing y. Third, an attacker can choose to reduce the number of bogus REQUESTs it sends to each node; this has the effect of reducing m, which again increases the probability of success. A final attack allows a powerful attacker to monitor the REQUESTs forwarded by each node legitimate node. Some of these legitimate nodes will have randomly chosen REQUESTs that represent compromised routes. The attacker can then attempt to forward such REQUESTs to nodes that did not hear that REQUEST directly from that node. This attack will be prevented by wormhole detection.

As mentioned in Section 10.3.5, if only one ROUTE REPLY is returned with any discovery, security is somewhat lower. In particular, only one route is returned, and each hop after the first has a $\frac{n-m}{n}$ probability of choosing a nonattacking node under the attacker model used in this section. In a working route, all nodes must forward a nonattacking REQUEST. As a result, the probability of choosing a working route is $\left(\frac{n-m}{n}\right)^{\ell}$, where ℓ is the number of intermediate nodes (excluding the initator and target).

This section presented an extremely conservative security analysis. In particular, an attacker as aggressive as the one described here would need to propagate the ROUTE REQUEST from each Route Discovery from many different locations, subjecting it to the intrusion detection mechanism. A real attacker considering the tradeoff between an improved probability of subversion and an increased probability of being caught is unlikely to use such a powerful attack.

10.5. Related Work

We have already discussed the vulnerability of current secure on-demand ad hoc network routing protocols [31, 79, 136, 170, 190] to the rushing attack in Section 10.1. Perlman's Flooding NPBR [143] routing protocol for wired networks does not suffer from this attack, since the protocol does not depend on the actual path of the flood for routing; rather, it requires that each packet be flooded through the the network.

Other secure routing protocols have been proposed based on periodic (proactive) mechanisms, for wired networks [33, 65, 69, 96, 102, 176, 177] as well as for wireless ad hoc networks [78, 148]. Although these protocols typically are not vulnerable to rushing attacks, such periodic protocols are often less desirable for ad hoc network routing due to their higher overhead and slower adaptivity.

Other areas in secure ad hoc network routing have been explored, such as trust establishment [9, 79, 82, 178], key generation [10], nodes that maliciously do not forward packets [116], and security requirements for forwarding nodes [189]. These areas are beyond the scope of this paper.

Routing protocol intrusion detection has been studied in wired networks as a mechanism for detecting misbehaving routers. Cheung and Levitt [34] and Bradley et al [23] propose intrusion detection techniques for detecting and identifying routers that send bogus routing update messages. In this paper, we describe one invariant of legitimate node behavior, and introduce a distributed mechanism to exclude nodes that have been caught violating that invariant.

10.6. Chapter Summary

In this paper, we have described the *rushing attack*, a novel and powerful attack against on-demand ad hoc network routing protocols. This attack allows an attacker to mount a denial-of-service attack against *all* previously proposed secure on-demand ad hoc network routing protocols. We have also presented RAP (Rushing Attack Prevention), a new protocol that thwarts the rushing attack.

We found that the widely used duplicate suppression technique makes the rushing attack possible, and we designed a new Route Discovery protocol called RAP that replaces the standard mechanism and thwarts the rushing attack. Our approach is generic, so any protocol that relies on duplicate suppression in Route Discovery can use our results to fend off rushing attacks. More importantly, we demonstrated that there are mechanisms that can defend against the rushing attack, even though all previous attempts at secure on-demand ad hoc network routing protocols have been vulnerable.

When integrated with a secure routing protocol, RAP incurs *no cost* unless the underlying secure protocol cannot find valid routes. When RAP is enabled, it incurs higher overhead than do standard Route Discovery techniques, but it can find usable routes when other protocols cannot, thus allowing successful routing and packet delivery when other protocols may fail entirely. We have also shown that existing on-demand routing protocols can be retrofitted using our technique to resist the rushing attack.

Chapter 11

Securing Quality-of-Service Routing in Ad Hoc Networks

In an ad hoc network, some applications may require higher levels of service than can be achieved using best-effort routing, cross-layer interactions (Chapter 4), or other lightweight mechanisms (Chapter 4.5). In addition, unlike wired networks, where overprovisioning is quite straightforward, overprovisioning in a wireless network may not be possible, due to constraints on radio spectrum and power level, or because of noise within that radio spectrum. As a result, carefully choosing paths with sufficient resources may be the only way to provide sufficient resources for many applications.

Though a number of protocols have been proposed for Quality-of-Service routing in ad hoc networks (e.g. [25, 30, 32, 105, 112, 142, 188]), these protocols are intended for operation in a trusted environment, and do not consider the disruptions that can be caused by an attacker. In this chapter, we discuss general mechanisms for securing QoS routing, and applying these to DSR's QoS-guided Route Discovery [25, 112] and the very similar AODV extensions [142].

11.1. QoS-Guided Route Discovery

In many on-demand routing protocols such as DSR and AODV, a node wishing to establish a QoS flow to a destination generally will only be able to route packets along the path that had minimum latency at the time of the Route Discovery. However, a QoS flow may have bandwidth or jitter needs that cannot be satisfied by the path ordinarily returned by Route Discovery. To allow Route Discovery to discover paths satisfying QoS constraints, Maltz introduced QoS-Guided Route Discovery [112], which allows a node to specify QoS metrics that must be satisfied by a path discovered using QoS-Guided Route Discovery. In this section, we review the previous work in QoS-guided Route Discovery.

In QoS-guided Route Discovery, ROUTE REQUEST packets are constrained to paths fulfilling certain requirements. When a node has a cached route, it may either perform a QoS-guided Route Discovery, or it may attempt to establish a new flow along the cached route. If the node chooses to use the cached route and the flow establishment is successful, it is not necessary to perform a QoS-guided Route Discovery, although one may be performed in an attempt to find a better route. The decision about whether or not to perform such a Discovery may be made based on resources available along a cached route or the node's estimate of the probability of successful setup along that route. Alternately, a node may choose to always perform a second search requesting a slightly higher level of resources than is available along the cached route.

To use this QoS-guided Route Discovery mechanism, a node sending a ROUTE REQUEST also inserts an optional QoS Request Header for each type of resource required. Each QoS Request Header indicates the type of resource, the minimum acceptable resource level, and the desired resource level. For example, an audio flow may require at least 2.4 kilobits per second of bandwidth but desire up to 128 kilobits per second.

A node receiving a ROUTE REQUEST containing one or more QoS Request Headers processes each QoS Request Header to determine whether or not the node can provide a new flow with resources at a level at least equal to the minimum requested. If it is unable to provide the minimum requested resource level for any requested resource, the node silently discards the ROUTE REQUEST. If it is unable to provide the desired level specified in any QoS Request Header in the packet, the node modifies the header by setting the desired level equal to the maximum resource level it can provide, and then forwards the ROUTE REQUEST normally. A node able to provide the desired level specified in all QoS Request Headers contained in the packet forwards the ROUTE REQUEST packet normally without modifying the QoS Request Header.

A node that propagates a ROUTE REQUEST containing QoS Request Headers may temporarily reserve the resources specified in the request in order to improve the likelihood that the resources are present when the source begins using this route.

11.2. Mechanisms for Securing QoS Routing

Our key observation is that properties of interest in QoS routing are generally monotone, and that any metrics used with QoS-guided Route Discovery must be monotone. For example, the resources of bandwidth, latency, and jitter all are monotone. In this chapter, we present mechanisms that enforce monotonicity and strict monotonicity in QoS metrics.

11.2.1. Broadcast Authentication for REQUEST Packets

Our mechanisms require the network to arrange some form of broadcast authentication for the immutable fields of REQUEST packets. This authentication can be provided by an efficient, instant broadcast authentication mechanism such as HORS [160]. Alternatively, this authentication can be integrated with a flooding prevention mechanism. In particular, since QoS-Guided Route Discovery requires a flood of the network and hence provides a means for an attacker to perform a Denial-of-Service attack, a secure ad hoc network routing protocol must enforce limits on the frequency at which a node can perform such flooding. Ariadne [79] uses a hash chain to provide this rate-limit. To authenticate the immutable fields of the REQUEST, we replace the hash chain with a MW-chain. A node uses one MW-chain step for each Route Discovery, and uses the value authenticated by the hash chain to authenticate the immutable fields of the REQUEST. For example, if the MW-chain allows the authentication of 2^{80} different values, then an 80-bit one-way hash of the immutable fields of the packet can be encoded as a single value authenticated using this MW-chain.

11.2.2. Enforcing Monotonicity

To ensure monotonicity, the initiator of a QoS-Guided Route Discovery creates a virtual hash chain for each QoS metric requested. This virtual hash chain can be a traditional hash chain, as described in Section 5.1.3, a skiplist (which allow for more efficient authentication of large changes in metric) [73], or a hash tree chain (which prevent same-metric fraud, effectively requiring each forwarder to change the QoS metric) [73].

To generate this hash chain, the initial value (value farthest from the anchor) is chosen to authenticate the maximum level of service requested by this QoS-Guided Route Discovery for this metric. Each step in the hash chain authenticates one quanta, and the hash chain is generated to represent each possible value between the maximum level of service (the initially generated value) and the minimum level of service (the anchor) requested by the QoS-Guided Route Discovery. A ROUTE REQUEST packet includes only the authenticator for the metric currently claimed in that REQUEST Each anchor is included in the Route Discovery, sent with broadcast authentication (as described in Section 11.2.1), allowing each recipient to authenticate each claimed QoS metric.

The quantization of integer or fixed point values is simpler than for floating point values. For example, the smallest step representable with an integer is 1, and the range of a 32-bit signed integer value is 2^{32}, whereas a 32-bit signed floating point value in the IEEE 754 standard can represent a step as small as 2^{-149} with a range of 2^{129}. We overcome this difficulty by using a variable quanta: since n bits can represent at most 2^n values, we conceptually sort all representable values, and correlate one step in a virtual hash chain with one element in this conceptually sorted list. This sorting can be achieved at low cost with proper data representation; for example, postitive floating point values in the IEEE 754 standard can be converted into their integer ranks simply by reading the value in as an unsigned integer of the same length (ignoring non-numeric values).

11.2.3. Limiting Overhead of QoS-Guided Route Discovery

In QoS-guided Route Discovery, a forwarding node does not perform *duplicate suppression* in the same way in standard Route Discovery. In normal Route Discovery, nodes having already forwarded a REQUEST from a Route Discovery ignore further REQUESTs from the same Discovery. In QoS-guided Route Discovery, a node should only ignore a REQUEST if it has forwarded a *better* REQUEST. This raises two problems: first, an intermediate node may not know which tradeoffs are preferred by the source, and second, an attacker can force a node to forward a large number of REQUESTs by playing a single REQUEST multiple times, using progressively better metrics. However, the source knows the tradeoffs it prefers. As a result, a node can include an evaluation function in each REQUEST. This evaluation function can take the form of a function selected from a list, or can be more general, such as has been proposed for Active Networks [180]. Each evaluation function should take the metrics of interest and a maximum value, and return an integer between 0 and the maximum value. A node then can forward an additional REQUEST only when the evaluation function returns a larger value than it did the previous time, thus allowing each node to bound the number of times it forwards a REQUEST from any single Discovery.

11.3. Related Work

A number of distance-vector protocols have been proposed that use hash chains to prevent malicious nodes from reducing the advertised distance to the destination. For example, Zapata and Asokan propose the use of a hash chain in each packet with the anchor authenticated using a public key signature [190]. In this chapter, we extend this approach by enforcing that each node counted is in fact a legitimate node, to resist an attacker attempting to decrease the apparent average of some QoS metric over a path. Furthermore, we use more than one anchor in each packet, and build a Merkle tree over each anchor, which improves authentication efficiency.

11.4. Chapter Summary

In this chapter we present a mechanism for securing QoS metrics that can be discovered using QoS-guided Route Discovery. These metrics all have the property that they are monotone, so we prevent an attacker from reducing monotonely increasing metrics, and prevent an attacker from increasing monotonely decreasing metrics. As a result, an attacker cannot claim a better metric than it has heard. Without secure hardware, it is impractical to force each node to claim a correct metric, but using this technique, an attacker cannot gain an arbitrary advantage over non-attacking routes.

Chapter 12

Thesis Summary and Conclusions

This thesis has presented two types of service improvements in ad hoc network routing: those for trusted environments and those for untrusted environments. In a trusted environment, the work in this thesis covers improvements that help all traffic types, as well improvements that allow on-demand protocols to classify and priortize real-time traffic. We also present mechanisms for securing routing, including efficient mechanisms to secure distance-vector and path-vector protocols, to thwart wormhole attackers, and to secure QoS-guided Route Discovery.

Chapter 2 describes the *link-state cache* which allows a source-routing protocol such as DSR make better use of information gathered through Route Discovery. In our simulations, link-state caches outperformed the previously used path-state caches by a factor of two in packet overhead, while simultaneously improving packet delivery performance. I also provide mechanisms for adapting cache parameters to the observed network behavior. I found that caches using adaptive parameter choices can often outperform statically tuned caches.

In Chapter 3, I presented *implicit source routes* which can remove all the per-packet overhead associated with source routes while still maintaining source control of routes. Our simulations showed a factor of two improvement in byte overhead, with moderate gains in packet delivery ratio and latency, at the cost of a slight increase in packet overhead.

Chapter 4 shows that MAC layer information, particularly utilization information, can be used at higher layers to significantly improve performance. With these optimizations, our simulated network could deliver 30% more traffic with the same packet delivery ratio. Alternatively, our optimizations halved the packet loss rate and decreased packet overhead by 25% in some scenarios. Using MAC layer information to provide congestion notification to TCP also significantly improved TCP fairness. Chapter 4.5 describes the use of physical layer link properties to change routing behavior. In particular, I used the Signal-to-Noise Ratio to modify Route Discovery and Route Maintenance behavior. My modification to Route Discovery allows nodes to discover routes that are likely to have a longer lifetime, whereas my modification to Route Maintenance allows a node to learn about a failing link and discover a new one before the old link breaks. These two modifications, combined with the use of implicit source routes and priority forwarding, can substantially improve the responsiveness of an on-demand protocol to link changes. In particular, following the implementation of these mechanisms in an ad hoc network testbed, we found that audio and video sent over the testbed was received with significantly better quality.

Chapter 6 presents *SEAD*, a secure ad hoc network routing prtoocol based on DSDV. SEAD secures against most attacks by uncoordinated attacker nodes, and uses only efficient (symmetric) cryptographic primitives. SEAD is generally able to deliver a larger fraction of packets than DSDV, though it experiences significantly higher byte overhead and median latency due to the overhead caused by the inclusion of hash values in each advertisement. SEAD provides relatively strong security with low computational overhead, and is relatively simple, making it suitable for use in computationally constrained nodes and environments where there is low mobility.

Chapter 7 presents *Ariadne*, a secure on-demand routing protocol based on DSR. Ariadne provides stronger security properties than SEAD, but can also have higher overhead. Since Ariadne does not secure optimizations to DSR, we compared it to an unoptimized version of DSR. Ariadne performs at least as well as the unoptimized version of DSR across all metrics, and exhibits significantly lower packet overhead.

In Chapter 8, I present several efficient security mechanisms. Several of these mechanisms prevent certain attacks possible against SEAD. When these mechanisms are used, SEAD is no longer vulnerable to *same metric fraud*, wherein a node readvertises the same metric it received, rather than increasing the metric by one. These mechanisms also prevent a number of Denial-of-Service attacks. I present *skiplists*, a general primitive that can be used in many applications of hash chains to reduce the computation required to verify certain hash chain elements. Finally, we describe an optimization for path-state protocols using symmetric primitives, such as Ariadne. This optimization, called *cumulative authentication*, reduces byte overhead by around 25%.

Chapter 9 describes *packet leashes*, which protect against the powerful "wormhole attacker." We describe both *temporal* and *geographical* leashes to limit the distance a packet can travel. We also design an efficient broadcast authentication mechanism to be compatible with the stringent requirements of temporal leashes.

In Chapter 10, I present a mechanism which defends against the powerful *rushing attack*. The rushing attack exploits the predicitability of which REQUEST will be forwarded by an on-demand ad hoc network routing protocol. Since this mechanism deliberately randomizes its responses, it exhibits substantially worse behavior than our other protocols; however, our analytical evaluation shows that our mechanism is quite robust, even in the presence of a large number of attackers.

Finally, Chapter 11 presents mechanisms for securing *QoS-guided Route Discovery* . These mechanisms exploit the monotonicity of many Quality-of-Service specifications (such as latency, jitter, available bandwidth). As in my work for securing distance vector protocols, I use efficient one-way hash chains to to prevent a node from claiming a route better than it actually has. As a result of this work, QoS-Guided Route Discovery can be secured against a malicious node attempting to claim a significantly better route than it actually has.

Within a trusted environment, the work in this thesis has improved service by reducing packet loss, packet overhead, and byte overhead by a factor of two in uncongested environments. In addition, this thesis has enabled trivial packet classification and prioritization, and supports 30% more traffic at the same loss rate (around 5%). Finally, this thesis has substantially reduced the latency and jitter caused by route breakage, enabling the use of audio and video over constantly moving ad hoc networks.

This thesis has also improved service in networks in untrusted environments, by defending against many attacks. The wormhole attack can now be detected, and non-coordinating attackers in distance vector protocols cannot claim lower metrics than they would be able to if they were honest nodes. Ariadne provides mechansims to defend against many types of attacks, even when multiple attackers coordinate to prevent routing. Finally, when QoS-guided Route Discovery is used to find routes meeting certain QoS metrics requirements, uncoordinated attackers cannot claim to have a better path then they would if they were honest nodes with infinite resources.

12.1. Conclusions

In this thesis, I show that it is possible to achieve a higher level of service than previously demonstrated with ad hoc network routing protocols, including the use of real-time applications such as videoconferencing. I also show that interactions with lower layers can substantially improve the performance of routing and transport layers. Finally, I demonstrate that ad hoc network routing can be secured, thus providing service even in a malicious environment.

Bibliography

[1] Norman Abramson. The ALOHA System—Another Alternative for Computer Communications. In *Proceedings of the Fall 1970 AFIPS Computer Conference*, pages 281–285, November 1970.

[2] Agere Systems Inc. Specification sheet for ORiNOCO World PC Card. Allentown, PA. Available at `ftp://ftp.orinocowireless.com/pub/docs/ORINOCO/BROCHURES/US/World%20PC%20Card%20US.pdf`.

[3] Ross Anderson, Charalampos Manifavas, and Chris Sutherland. NetCard – A Practical Electronic Cash System. In *Security Protocols — International Workshop*, edited by Mark Lomas, volume 1189 of *Lecture Notes in Computer Science*, pages 49–57. Springer-Verlag, April 1997.

[4] APE Project. The APE Testbed. Available at `http://apetestbed.sourceforge.net/`.

[5] ARC International. ARC releases BlueForm, a comprehensive solution for Bluetooth systems on a chip. Press Release 6-04-01, Elstree, United Kingdom. Available at `http://www.arccores.com/newsevents/PR/6-04-01-2.htm`, June 4, 2001.

[6] N. Asokan, Gene Tsudik, and Michael Waidner. Server-Supported Signatures. *Journal of Computer Security*, 5(1):91–108, 1997.

[7] Daniel O. Awduche, Lou Berger, Der-Hwa Gan, Tony Li, Vijay Srinivasan, and George Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. Work in progress, draft-ietf-mpls-lsp-tunnel-09.txt, August 2001.

[8] Fred Baker and Randall Atkinson. RIP-2 MD5 Authentication. RFC 2082, January 1997.

[9] Dirk Balfanz, D. K. Smetters, Paul Stewart, and H. Chi Wong. Talking To Strangers: Authentication in Ad-Hoc Wireless Networks. In *Symposium on Network and Distributed Systems Security (NDSS 2002)*, February 2002.

[10] Stefano Basagni, Kris Herrin, Emilia Rosti, and Danilo Bruschi. Secure Pebblenets. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 156–163, October 2001.

[11] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology — CRYPTO "96*, edited by Neal Koblitz, number 1109 in Lecture Notes in Computer Science, pages 1–15. Springer-Verlag, 1996.

[12] Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[13] Bhargav Bellur and Richard G. Ogier. A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM '99), pages 178–186, March 1999.

[14] A. Benjaminson and S. C. Stallings. A Microcomputer-Compensated Crystal Oscillator Using a Dual-Mode Resonator. In *Proceedings of the 43rd Annual Symposium on Frequency Control*, pages 20–26, May 1989.

[15] Dave Beyer, Thane Frivold, Darren Lancaster, John Hight, Mark Lewis, and Fred Templin. Radio Device API. DARPA Global Mobile Information Systems (GloMo) Program, Defense Advanced Research Projects Agency, Arlington, VA.

[16] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A Media Access Protocol for Wireless LAN's. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 212–225, August 1994.

[17] Matt Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[18] Jeff Boleng, William Navidi, and Tracy Camp. Metrics to Enable Adaptive Protocols for Mobile Ad Hoc Networks. In *Proceedings of the International Conference on Wireless Networks* (ICWN'02), pages 293–298, 2002.

[19] Rajendra Boppana and Satyadeva P. Konduru. An Adaptive Distance Vector Routing Algorithm for Mobile, Ad Hoc Networks. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2001), pages 1753–1762, March 2001.

[20] Bob Braden, David Clark, and Scott Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, June 1994.

[21] Bob Braden, David D. Clark, Jon Crowcroft, Bruce Davie, Steve Deering, Deborah Estrin, Sally Floyd, Van Jacobson, Greg Minshall, Craig Partridge, Larry Peterson, K. K. Ramakrishnan, Scott Shenker, John Wroclawski, and Lixia Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309, April 1998.

[22] Bob Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, September 1997.

[23] Kirk A. Bradley, Steven Cheung, Nick Puketza, Biswanath Mukherjee, and Ronald A. Olsson. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 115–124, May 1998.

[24] Eric A. Brewer, Randy H. Katz, Elan Amir, Hari Balakrishnan, Yatin Chawathe, Armando Fox, Steven D. Gribble, Todd Hodes, Giao Nguyen, Venkata N. Padmanabhan, Mark Stemm, Srinivasan Seshan, and Tom Henderson. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications*, 5(5):8–24, October 1998.

[25] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-03.txt, October 1999. Work in progress.

[26] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking* (MobiCom 1998), pages 85–97, October 1998.

[27] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. PGP in Constrained Wireless Devices. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[28] Tracy Camp, Jeff Boleng, and Vanessa Davies. Mobility Models for Ad Hoc Network Simulations. *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2002.

[29] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of INFOCOMM'99*, March 1999.

[30] Derya H. Cansever, Arnold M. Michelson, and Allen H. Levesque. Quality of Service Support in Mobile Ad-Hoc IP Networks. In *Proceedings of the Military Communications Conference* (MILCOM 1999), pages 30–34, October 1999.

[31] Claude Castelluccia and Gabriel Montenegro. Protecting AODV against Impersonation attacks. IETF MANET Mailing List, Message-ID 006601c1eb8c$f279c560$1d1fc7c2@charmette.inrialpes.fr, `https://www1.ietf.org/mail-archive/working-groups/manet/current/msg00186.html`, April 2002.

[32] Shigang Chen and K. Nahrstedt. Distributed Quality-of-Service Routing in Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1488–1505, August 1999.

[33] Steven Cheung. An Efficient Message Authentication Scheme for Link State Routing. In *13th Annual Computer Security Applications Conference*, December 1997.

[34] Steven Cheung and Karl Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection. In *The 1997 New Security Paradigms Workshop*, pages 94–106, September 1998.

[35] Tom Clark. Tom Clark's Totally Accurate Clock FTP Site. Greenbelt, Maryland. Available at `ftp://aleph.gsfc.nasa.gov/GPS/totally.accurate.clock/`.

[36] Thomas Clausen, Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized Link State Routing Protocol. Internet-Draft, draft-ietf-manet-olsr-05.txt, October 2001. Work in progress.

[37] Thomas Clausen, Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized Link State Routing Protocol. Internet-Draft, draft-ietf-manet-olsr-06.txt, September 2001. Work in progress.

[38] Don Coppersmith and Markus Jakobsson. Almost Optimal Hash Sequence Traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '02)*, Lecture Notes in Computer Science, 2002.

[39] Mark Corner and Brian Noble. Zero-Interaction Authentication. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MobiCom 2002)*, pages 1–11, September 2002.

[40] Nicolas Courtois, Louis Goubin, and Jacques Patarin. Flash, a fast multivariate signature algorithm. In *Progress in Cryptology — CT-RSA 2001*, edited by David Naccache, number 2020 in LNCS. Springer-Verlag, April 2001.

[41] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, March 1999.

[42] Eyal de Lara, Dan S. Wallach, and Willy Zwaenepoel. Puppeteer: Component-based Adaptation for Mobile Computing. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[43] Defense Advanced Research Projects Agency. Frequently Asked Questions v4 for BAA 01-01, FCS Communications Technology. Washington, DC. Available at `http://www.darpa.mil/ato/solicit/baa01_01faqv4.doc`, October 2000.

[44] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[45] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, November 1976.

[46] A. Dracinschi and S. Fdida. Efficient Congestion Avoidance Mechanism. In *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks* (LCN'00), November 2000.

[47] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang, and Satish K. Tripathi. Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks. *IEEE Personal Communications*, 4(1):36–45, February 1997.

[48] Donald E. Eastlake. Physical Link Security Type of Service. RFC 1455, May 1993.

[49] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. In *Advances in Cryptology — CRYPTO "89*, edited by Gilles Brassard, pages 263–277. Springer-Verlag, 1989. Lecture Notes in Computer Science Volume 435.

[50] Kevin Fall and Kannan Varadhan, editors. *ns* Notes and Documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. Available from `http://www-mash.cs.berkeley.edu/ns/`.

[51] Sally Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[52] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[53] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[54] Eran Gabber and Avishai Wool. How to Prove Where You Are: Tracking the Location of Customer Equipment. In *Proceedings of the 5th ACM Conference on Computer and Communications Security* (CCS '98), pages 142–149, November 1998.

[55] J.J. Garcia-Luna-Aceves, Chane L. Fullmer, Ewerton Madruga, David Beyer, and Thane Frivold. Wireless Internet Gateways (WINGS). In *Proceedings of the Military Communications Conference* (MILCOM 1997), pages 1271–1276, November 1997.

[56] Brian Gladman. Cryptography Technology: Implementations of AES (Rijndael) in C/C++ and Assembler, June 2002. Available at `http://fp.gladman.plus.com/cryptography_technology/rijndael/`.

[57] Tom Goff, Nael B. Abu-Ghazaleh, Dhananjay S. Phatak, and Ridvan Kahvecioglu. Preemptive Routing in Ad Hoc Networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking* (MobiCom 2001), pages 43–52, July 2001.

[58] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[59] Shafi Goldwasser and Mihir Bellare. Lecture Notes on Cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999, August 1999.

[60] Ajay Chandra V. Gummalla and John O. Limb. Wireless Medium Access Control Protocols. *IEEE Communications Surveys & Tutorials*, Second Quarter 2000. Available at `http://www.comsoc.org/livepubs/surveys/`.

[61] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[62] Zygmunt J. Haas. A Routing Protocol for the Reconfigurable Wireless Network. In *Proceedings of the 1997 IEEE 6th International Conference on Universal Personal Communications Record: Bridging the Way to the 21st Century* (ICUPC '97), volume 2, pages 562–566, October 1997.

[63] Zygmunt J. Haas and Marc R. Pearlman. The Performance of Query Control Schemes for the Zone Routing Protocol. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 167–177, June 1998.

[64] Neil M. Haller. The S/KEY One-Time Password System. In *Proceedings of the 1994 Symposium on Network and Distributed Systems Security* (NDSS '94), pages 151–157, February 1994.

[65] Ralf Hauser, Antoni Przygienda, and Gene Tsudik. Reducing the Cost of Security in Link State Routing. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security* (NDSS '97), pages 93–99, February 1997.

[66] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-Payments based on iKP. Research Report 2791 (# 89269), IBM Research, February 1996.

[67] Ralf Hauser and Gene Tsudik. On Shopping *Incognito*. In *Second USENIX Workshop on Electronic Commerce*, November 1996.

[68] C. Hedrick. Routing Information Protocol. RFC 1058, November 1988.

[69] Andy Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, August 1998.

[70] Helion Technology Ltd. High performance Solutions in Silicon — MD5 core. Cambridge, England. Available at `http://www.heliontech.com/core5.htm`.

[71] Ying-Kwei Ho and Ru-Sheng Liu. On-Demand QoS-Based Routing Protocol for Ad Hoc Mobile Wireless Networks. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications, 2000 (ISCC 2000)*, pages 560–565, July 2000.

[72] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NSS: An NTRU Lattice-Based Signature Scheme. In *Advances in Cryptology — EUROCRYPT '2001*, edited by Birgit Pfitzmann, number 2045 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[73] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Efficient Security Mechanisms for Routing Protocols. In *Proceedings of the 2003 Symposium on Network and Distributed Systems Security* (NDSS '03), February 2003.

[74] Yih-Chun Hu and David B. Johnson. Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (MobiCom 2000), pages 231–242, August 2000.

[75] Yih-Chun Hu and David B. Johnson. Implicit Source Routing in On-Demand Ad Hoc Network Routing. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 1–10, October 2001.

[76] Yih-Chun Hu and David B. Johnson. Ensuring Cache Freshness in Source-Routed Ad Hoc Network Routing Protocols. In *Proceedings of the Workshop on Principles of Mobile Computing* (POMC 2002), pages 25–30. ACM, October 2002.

[77] Yih-Chun Hu, David B. Johnson, and David A. Maltz. Flow State in the Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsrflow-00.txt, February 2001. Work in progress.

[78] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA '02), pages 3–13, June 2002.

[79] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking* (MobiCom 2002), pages 12–23, September 2002.

[80] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols. Technical Report TR01-403, Department of Computer Science, Rice University, June 2002.

[81] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2003). IEEE, April 2003.

[82] Jean-Pierre Hubaux, Levente Buttyán, and Srdjan Čapkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), pages 146–155, October 2001.

[83] P'ei hung Chuang, Hsiao kuang Wu, and Ming kuang Liao. Dynamic QoS Allocation for Multimedia Ad Hoc Wireless Networks. In *Proceedings of the Eighth International Conference on Computer Communications and Networks*, pages 480–485, October 1999.

[84] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[85] IETF RMT Working Group. Reliable Multicast Transport (RMT) Charter. Available at `http://www.ietf.org/html.charters/rmt-charter.html`.

[86] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking* (MobiCom 1999), pages 195–206, August 1999.

[87] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA'94), pages 158–163, December 1994.

[88] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[89] David B. Johnson, David A. Maltz, and Josh Broch. The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, edited by Charles E. Perkins, chapter 5, pages 139–172. Addison-Wesley, 2001.

[90] David B. Johnson, David A. Maltz, Yih-Chun Hu, and Jorjeta G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-07.txt, February 2002. Work in progress.

[91] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[92] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for Smart Dust. In *International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 271–278, August 1999.

[93] Ad Kamerman and Leo Monteban. WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band. *Bell Labs Technical Journal*, pages 118–133, Summer 1997.

[94] Phil Karn. MACA — A New Channel Access Method for Packet Radio. In *Proceedings of the 9th Computer Networking Conference*, pages 134–140, September 1990.

[95] Dean Kawaguchi and Sarosh Vesuna. Symbol Technologies, Inc. Automates Ssystem-To-Gates Design Flow For Wireless LAN ASIC with COSSAP and Behavioral Compiler. Mountain View, California. Available at `http://www.synopsys.com/news/pubs/bctb/sep98/frame_art1.html`, September 1998.

[96] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues. In *Proceedings of the 2000 Symposium on Network and Distributed Systems Security* (NDSS '00), pages 103–116, February 2000.

[97] Minkyong Kim and Brian Noble. Mobile Network Estimation. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking* (MobiCom 2001), pages 298–309, July 2001.

[98] Tim Kindberg, Kan Zhang, and Narendar Shankar. Context Authentication Using Constrained Channels. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, pages 14–21, June 2002.

[99] Young-Bae Ko and Nitin Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking* (MobiCom 1998), pages 66–75, October 1998.

[100] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, September 1993.

[101] Jiejun Konh, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. In *Ninth International Conference on Network Protocols (ICNP '01)*, pages 251–260, November 2001.

[102] B. Kumar. Integration of Security in Network Routing Protocols. *SIGSAC Review*, 11(2):18–25, 1993.

[103] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[104] Tony Larsson. Personal communication, February 8, 2000.

[105] Seoung-Bum Lee and Andrew T. Campbell. INSIGNIA: In-Band Signaling Support for QoS in Mobile Ad Hoc Networks. In *Proceedings of the 5th International Workshop on Mobile Multimedia Communications* (MoMuC 98), October 1998.

[106] Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*, 14(4):255–293, September 2001. Available at `http://www.cryptosavvy.com/`.

[107] Ben Liang. Personal communication, February 4, 2000.

[108] Chunhung Richard Lin and Jain-Shing Liu. QoS Routing in Ad Hoc Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1426–1438, August 1999.

[109] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP Misconfiguration. In *Proceedings of the SIGCOMM '02 Conference on Communications Architectures, Protocols and Applications*, August 2002.

[110] Gary Scott Malkin. RIP Version 2 Protocol Applicability Statement. RFC 1722, November 1994.

[111] Gary Scott Malkin. RIP Version 2. RFC 2453, November 1998.

[112] David A. Maltz. Resource Management in Multi-hop Ad Hoc Networks. Technical Report CMU-CS-00-150, School of Computer Science, Carnegie Mellon University, 1999.

[113] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. The Effects of On-Demand Behavior in Routing Protocols for Multi-Hop Wireless Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1439–1453, aug 1999.

[114] David A. Maltz, Josh Broch, and David B. Johnson. Quantitative Lessons From a Full-Scale Multi-Hop Wireless Ad Hoc Network Testbed. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, September 2000.

[115] Mahesh K. Marina and Samir R. Das. Performance of Route Caching Strategies in Dynamic Source Routing. In *Proceedings of the 2nd Wireless Networking and Mobile Computing* (WNMC), April 2001.

[116] Sergio Marti, T.J. Giuli, Kevin Lai, and Mary Baker. Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (MobiCom 2000), pages 255–265, August 2000.

[117] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[118] Stephen Matyas, Carl Meyer, and Jonathan Oseas. Generating Strong One-Way Functions with Cryptographic Algorithm. 27:5658–5659, 1985.

[119] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley, Reading, Massachusetts, 1996.

[120] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, July 1993.

[121] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[122] Ralph C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–133, April 1980.

[123] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO "87*, edited by Carl Pomerance, pages 369–378, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science Volume 293.

[124] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology — CRYPTO "89*, edited by Gilles Brassard, pages 218–238, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[125] Microsoft Windows NetMeeting 3.0 Resource Kit. Available at `http://www.microsoft.com/windows/NetMeeting/Corp/reskit/`.

[126] David L. Mills. A Computer-Controlled LORAN-C Receiver for Precision Timekeeping. Technical Report 92-3-1, Department of Electrical and Computer Engineering, University of Delaware, March 1992.

[127] David L. Mills. A Precision Radio Clock for WWV Transmissions. Technical Report 97-8-1, Department of Electrical and Computer Engineering, University of Delaware, August 1997.

[128] The Monarch Project. Rice Monarch Project: Mobile Networking Architectures, project home page. Available at `http://www.monarch.cs.rice.edu/`.

[129] Shree Murthy and J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[130] Thomas Narten, Erik Nordmark, and William Allen Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461, December 1998.

[131] National Institute of Standards and Technology. Secure Hash Standard, May 1993. Federal Information Processing Standards (FIPS) Publication 180-1.

[132] Kathleen Nichols, Steven Blake, Fred Baker, and David L. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

[133] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[134] Richard G. Ogier, Fred L. Templin, Bhargav Bellur, and Mark G. Lewis. Topology Broadcast Based on Reverse-Path Forwarding (TBRPF). Internet-Draft, draft-ietf-manet-tbrpf-05.txt, March 2002. Work in progress.

[135] OpenSSL Project team. OpenSSL, May 2000. `http://www.openssl.org/`.

[136] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure Routing for Mobile Ad Hoc Networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, January 2002.

[137] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM '97), pages 1405–1413, April 1997.

[138] Torben Pryds Pedersen. Electronic Payments of Small Amounts. In *Security Protocols — International Workshop*, edited by Mark Lomas, volume 1189 of *Lecture Notes in Computer Science*, pages 59–68. Springer-Verlag, April 1997.

[139] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad Hoc On Demand Distance Vector (AODV) Routing. Internet-Draft, draft-ietf-manet-aodv-10.txt, January 2002. Work in progress.

[140] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994. A revised version of the paper is available from `http://www.cs.umd.edu/projects/mcml/papers/Sigcomm94.ps`.

[141] Charles E. Perkins and Elizabeth M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA'99), pages 90–100, February 1999.

[142] Charles E. Perkins, Elizabeth M. Royer, and Samir R. Das. Quality of Service for Ad hoc On-Demand Distance Vector Routing. Internet-Draft, draft-ietf-manet-aodvqos-00.txt, July 2000. Work in progress.

[143] Radia Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.

[144] Adrian Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 28–37, November 2001.

[145] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and Secure Source Authentication for Multicast. In *Proceedings of the 2001 Symposium on Network and Distributed Systems Security* (NDSS '01), February 2001.

[146] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5 (Summer), 2002.

[147] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[148] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking* (MobiCom 2001), Rome, Italy, July 2001.

[149] Raymond L. Pickholtz, Donald L. Schilling, and Laurence B. Milstein. Theory of Spread Spectrum Communications — A Tutorial. *IEEE Transactions on Communications*, 30(5):855–884, May 1982.

[150] Jon Postel. Transmission Control Protocol: DARPA Internet Program Protocol Specification. RFC 793, September 1981.

[151] Guillaume Poupard and Jacques Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In *Advances in Cryptology — EUROCRYPT '98*, edited by Kaisa Nyberg, number 1403 in Lecture Notes in Computer Science, pages 422–436. Springer-Verlag, 1998.

[152] Guillaume Poupard and Jacques Stern. On The Fly Signatures based on Factoring. In *6th ACM Conference on Computer and Communications Security*, pages 37–45, November 1999.

[153] Proxim, Inc. Data sheet for Proxim Harmony 802.11a CardBus Card. Sunnyvale, CA. Available at `http://www.proxim.com/products/all/harmony/docs/ds/harmony_11a_cardbus.pdf`.

[154] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint Relaying: An Efficient Technique for flooding in Mobile Wireless Networks. Technical Report Research Report RR-3898, INRIA, February 2000.

[155] K. K. Ramakrishnan, Sally Floyd, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 1991.

[156] K. K. Ramakrishnan and Raj Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–313, 1988.

[157] Bhaskaran Raman, Pravin Bhagwat, and Srinivasan Seshan. Arguments for Cross-Layer Optimizations in Bluetooth Scatternets. In *Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*, January 2001.

[158] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[159] Joyce K. Reynolds and Jon Postel. Assigned Numbers. RFC 1700, October 1994.

[160] Leonid Reyzin and Natan Reyzin. Better than Biba: Short One-Time Signatures with Fast Signing and Verifying. In *Information Security and Privacy — 7th Australasian Conference (ACSIP 2002)*, edited by Jennifer Seberry, number 2384 in Lecture Notes in Computer Science. Springer-Verlag, July 2002.

[161] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[162] Ronald L. Rivest and Adi Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Security Protocols — International Workshop*, edited by Mark Lomas, volume 1189 of *Lecture Notes in Computer Science*, pages 69–88. Springer-Verlag, April 1997.

[163] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[164] Pankaj Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security* (CCS '99), pages 93–100, November 1999.

[165] Eric C. Rosen and Yakov Rekhter. BGP/MPLS VPNs. RFC 2547, March 1999.

[166] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol Label Switching Architecture. RFC 3031, January 2001.

[167] Miguel Sanchez. RE: Mobility pattern in a MANET, June 25, 1998. IETF MANET Mailing List, Message-ID: <000a01bda055$d84f9380$11352a9e@msanchez.disca.upv.es>.

[168] Miguel Sanchez. Re: Node Movement Models in Ad hoc, July 15, 1999. IETF MANET Mailing List, Message-ID: <378DC8F6.B01CF351@disca.upv.es>.

[169] Miguel Sanchez. Personal communication, February 1, 2000.

[170] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Elizabeth Royer, and Clay Shields. A Secure Routing Protocol for Ad hoc Networks. In *Proceedings of the 10th IEEE International Conference on Network Protocols* (ICNP '02), November 2002.

[171] Claus P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[172] Adi Shamir and Yael Tauman. Improved Online/Offline Signature Schemes. In *Advances in Cryptology — CRYPTO '2001*, edited by Joe Kilian, number 2139 in Lecture Notes in Computer Science, pages 355–367. Springer-Verlag, 2001.

[173] Prasun Sinha, Raghupathy Sivakumar, and Bharghavan Vaduvur. Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2001), April 2001.

[174] Karen E. Sirois and Stephen T. Kent. Securing the Nimrod Routing Architecture. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security* (NDSS '97), February 1997.

[175] R. Sivakumar, P. Sinha, and V. Bharghavan. CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm. *IEEE Journal on Selected Areas in Communications*, 17(8):1454–1465, August 1999.

[176] Bradley R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Proceedings of Global Internet'96*, pages 81–85, November 1996.

[177] Bradley R. Smith, Shree Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance Vector Routing Protocols. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security* (NDSS '97), pages 85–92, February 1997.

[178] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proceedings of the 7th International Workshop on Security Protocols*, volume 1796 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[179] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, New Jersey, third edition, 1996.

[180] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[181] Chai-Keong Toh. Associativity Based Routing for Ad-Hoc Mobile Networks. *Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems*, 4(2):103–139, March 1997.

[182] Joseph D. Touch. Performance Analysis of MD5. In *Proceedings of the ACM SIGCOMM '95 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 77–86, August 1995.

[183] Trimble Navigation Limited. Data Sheet and Specifications for Trimble Thunderbolt GPS Disciplined Clock. Sunnyvale, California. Available at `http://www.trimble.com/thunderbolt.html`.

[184] Aristotelis Tsirigos and Zygmunt J. Haas. Multipath Routing in Mobile Ad Hoc Networks or How to Route in the Presence of Topological Changes. In *Proceedings of the Military Communications Conference* (MILCOM 2001), pages 878–883, October 2001.

[185] Damla Turgut, Sajal K. Das, and Mainak Chatterjee. Longevity of Routes in Mobile Ad hoc Networks. In *Proceedings of the IEEE 53rd Vehicular Technology Conference* (VTC Spring 2001), 2001.

[186] Richard von Mises. Über Aufteilungs- und Besetzungswahrscheinlichkeiten. *Revue de la Faculté des Sciences de l'Université d'Istanbul*, 4:145—163, 1939.

[187] Alec Woo. CS294-8 Deeply Networked Systems Mote Documentation and Development Information. Berkeley, CA. Available at `http://www.cs.berkeley.edu/~awoo/smartdust/`.

[188] Hannan Xiao, W.K.G. Seah, A. Lo, and K.C. Chua. A Flexible Quality of Service Model for Mobile Ad-Hoc Networks. In *Proceedings of the IEEE 51st Vehicular Technology Conference* (VTC Spring 2000), pages vol.1 445–449, May 2000.

[189] Seung Yi, Prasad Naldurg, and Robin Kravets. Security-Aware Ad hoc Routing for Wireless Networks. Technical Report UIUCDCS-R-2001-2241, Department of Computer Science, University of Illinois at Urbana-Champaign, August 2001.

[190] Manel Guerrero Zapata and N. Asokan. Securing Ad Hoc Routing Protocols. In *Proceedings of the ACM Workshop on Wireless Security* (WiSe), pages 1–10, September 2002.

[191] Kan Zhang. Efficient Protocols for Signing Routing Messages. In *Proceedings of the 1998 Symposium on Network and Distributed Systems Security* (NDSS '98), San Diego, California, March 1998.

[192] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6):24–30, November/December 1999.