LOCAL ROUTE RECOVERY IN MOBILE AD HOC NETWORKS

BY

CIGDEM SENGUL

B.Sc., Istanbul Technical University, 2000

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Sceince
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2003

Urbana, Illinois

Abstract

On-demand routing protocols for ad hoc networks reduce the cost of routing in high mobility environments. However, route discovery in on-demand routing is typically performed via network-wide flooding, which consumes a substantial amount of bandwidth. Therefore, it is essential to reduce the frequency of route discoveries to achieve efficient communication. In this thesis, we present a technique called by-pass recovery that aims to reduce the frequency of route request floods due to topological changes. Specifically, when a broken link is detected, a node patches the effected route using neighborhood information and thereby by-passes the broken link. We implemented a prototype of our approach based on source routing. Simulation studies show that SLR (Source Routing with Local Recovery) achieves efficient and effective local recovery while maintaining acceptable overhead.

Dedicated to my parents, Sait and Nural Sengul

Acknowledgments

I would like to thank my advisor, Dr. Robin Kravets, for providing me invaluable direction. Her support and encouragement made this thesis possible. I would also like to thank Casey Carter for discussions and comments.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

An ad hoc network is formed by a group of autonomous wireless devices without depending on any fixed infrastructure. Each node communicates directly with its neighbors and functions as a router that forwards packets for nodes that are not within transmission range of the sender. Ad hoc networks provide fast and easy deployment due to their self-organizing and infrastructureless nature. Such networks are ideal for situations where building an infrastructure is time-consuming or costly, if not impossible. Mobile ad hoc networks (MANETs) are viewed as suitable systems which can support specific applications, including networks for developing nations, emergency disaster relief, pervasive computing and military applications. Ad hoc networks can also be useful in conferences where people participating in the conference can form temporary networks without the presence of fixed base stations and system administrators.

Although ad hoc networks provide numerous advantages, they have certain characteristics that impose new implementation challenges, especially in routing. Maintaining communication in ad hoc networks requires effective routing mechanisms in the presence of dynamic topology, low bandwidth and limited battery power. Furthermore, radio link related problems such as signal fading, interference, multipath, exposed and hidden terminal, etc., can further complicate the task of a routing protocol in ad hoc networks. Changes in network topology may cause route failures and require discovery of new routes. However, route discovery can be expensive, often resulting in a network-wide flood. Therefore, the challenge to routing in dynamic ad hoc networks stems from the need to maintain routes while minimizing overhead from such maintenance.

Routing protocols for ad hoc networks can be categorized as proactive or reactive (on-demand) based on when routes are discovered. Proactive protocols [28] maintain consistent and up-to-date routing information regardless of the presence of traffic, thus consume valuable resources such as bandwidth and power even if the network is idle. On-demand routing has been shown to reduce routing overhead in highly mobile environments by only maintaining actively used routes [33, 4, 27, 29, 16]. Although on-demand routing protocols only initiate route discovery when a route is needed, such discovery is typically performed via network-wide flooding. Since flooding consumes a substantial amount of bandwidth and energy, it is essential to reduce the frequency of route discoveries, and so network flooding.

To overcome performance problems from frequent route discovery attempts, hybrid protocols incorporate both reactive and proactive protocol characteristics [11, 15]. Zone Routing Protocol (ZRP) [11] and Cluster Based Routing Protocol (CBRP) [15] are examples of hybrid protocols that divide the network into zones/clusters and utilize reactive protocols for inter-zone communication and proactive protocols for intra-zone communication. While hybrid protocols do not waste resources by flooding the network for each route request, it is difficult to balance the cost of exchanging routing information periodically (i.e., proactivity) and network-wide flooding for route discovery (i.e., reactivity) [32].

Other protocols reduce the frequency of flooding by allowing an intermediate node to initiate a limited route discovery in the event of a route failure [29, 36] or employing local error recovery mechanisms [34, 3, 27]. However, protocols using either limited broadcast or local recovery have focused on reducing packet drops and not on utilizing the bandwidth efficiently during route recovery. Multipath routing protocols cache multiple routes to a destination in a single route discovery [23, 19, 18, 26, 31]. However, in the presence of mobility, mutipath protocols incur additional packet drops and delay due to their dependency on potentially stale routes from caches. The goal of our research is to improve local recovery to reduce the frequency of route request floods while maintaining acceptable overhead.

This thesis describes a technique called by-pass route recovery that uses local query messages to acquire information about the nodes in the neighborhood of a broken route. The goal is to find a patch between one of the neighbors and a node along the route to the destination, essentially by-passing the broken link. By-pass route recovery is supported by having all nodes snoop ongoing communication to track their neighbors. By tying local recovery to up-to-date information about a node's neighborhood, the chance of recovering a broken route increases compared to using potentially stale routes from caches. While by-pass route recovery imposes some overhead to in terms of local recovery messages, these messages only impact nodes in the one-hop neighborhood of the repairing node. The key benefit of by-pass route recovery comes from this localization, which allows scaling to large networks.

We demonstrate the effectiveness of by-pass route recovery in the context of SLR (Source Routing with Local Recovery), a DSR-based on-demand routing protocol. In SLR, efficient and effective local route recovery is achieved by combining route recovery with an improved route invalidation mechanism that closely ties route validity to up-to-date neighbor information. Results of extensive simulations show that SLR enables quick recovery of broken routes, increasing the packet delivery ratio while maintaining acceptable overhead.

1.1 Abbreviations

ABR	Associativity-Based Routing
ACK	Acknowledgment
AODV	Ad Hoc On-Demand Distance Vector
AODV-BR	AODV with Backup Routes
AOMDV	Ad Hoc On-Demand Multipath Distance Vector
CBR	Constant Bit Rate
CBRP	Cluster Based Routing Protocol
CHAMP	Caching and Multipath Routing Protocol
CTS	Clear to Send
DAG	Directed Acyclic Graph
DCF	Distributed Coordination Function
DSR	Dynamic Source Routing
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
LAR	Location-Aided Routing
MAC	Media Access Control
MANET	Mobile Ad Hoc Network
NSR	Neighborhood Aware Source Routing
RTS	Request to Send
SMR	Split Multipath Routing
SLR	Source Routing with Local Recovery
TORA	Temporally-Ordered Routing Algorithm
TTL	Time to Live
WAR	Witness Aided Routing
ZRP	Zone Routing Protocol

Chapter 2

Routing in Ad Hoc Networks

Wireless communication is becoming increasingly popular due to the technological advances in portable and wireless communication devices. There are two distinct approaches for enabling communication between wireless hosts. The first approach is supported by an infrastructured network, with fixed and wired gateways. The gateways in such networks are called base stations. A mobile node within an infrastructured network connects and communicates to the nearest base station in its communication range. As a mobile node moves out of the range of one base station, a "handoff" occurs to a new base station, and the mobile node is able to continue its communication without any disruption.

The second approach is to form an infrastructureless network, commonly known as an ad hoc network, among devices that want to communicate with each other. Infrasturetureless networks do not have fixed routers or base stations. All users participating in such a network should be willing to colloboratively forward data packets to maintain an operational network. Therefore, there is no need for handoff and location management in the sense of an infrastructured network. However, in some sense, these problems still exist in ad hoc networks where, for example, a node employs location management for neighbor discovery.

Due to the limited transmission range of wireless network devices, nodes that are not in range of each other communicate via intermediate nodes that act as routers. Figure

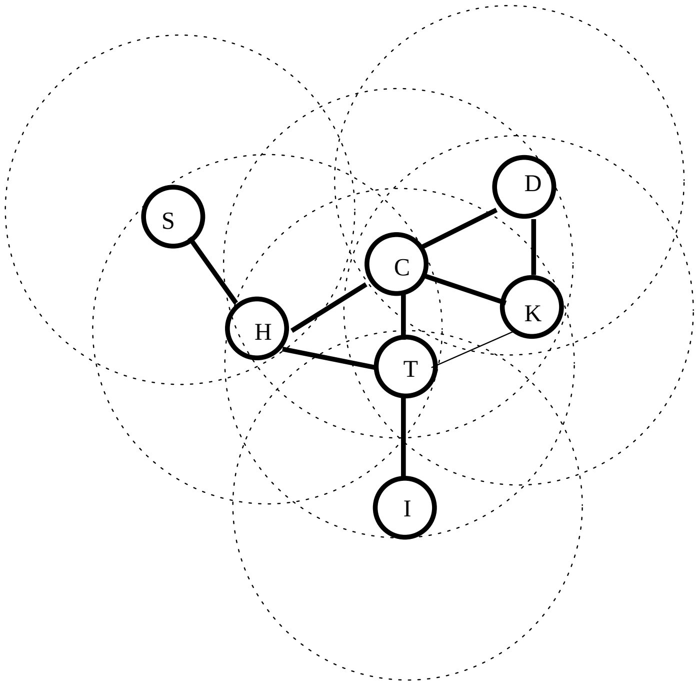


Figure 2.1 Example ad hoc network

2.1 shows an example of an ad hoc network with seven nodes. In this network, although Node S and Node I are not in transmission range, communication between these nodes can be established via Nodes H and T, which will act as routers and forward packets between Node S and Node I.

Since a message can traverse several hops before it reaches its destination, a routing protocol is necessary to provide the establishment and maintenance of routes. In mobile ad hoc networks, the routers are free to move randomly and organize themselves arbitrarily, thus the network's topology may change rapidly and unpredictably. The dynamic topology of such networks impose challenges on routing protocols. It is crucial for the protocol to have the ability to detect broken routes fast enough and immediately react to those changes. Additionally, an ad hoc routing protocol should incur low routing overhead, since typical nodes in ad hoc networks have limited CPU capacity, storage capacity, battery power and bandwidth.

There has been extensive research on routing protocols for ad hoc networks using conventional routing protocols as underlying algorithms. Therefore, it is essential to understand the basic operation of traditional routing protocols like flooding, link-state and distance-vector, which are described next.

2.1 Conventional Routing Algorithms

2.1.1 Flooding

Many routing protocols in wireless networks use flooding (e.g., DSDV [28], AODV [29] and DSR [16]) to distribute control information (i.e. a control message is sent from the source node to all nodes in the network). Flooding operates as follows. The source node sends the message to its neighbors, which in turn relay this message to their neighbors until all nodes in the network have received the message.

Flooding obviously generates a vast number of duplicate packets and some sort of sequence number is necessary to keep track of which packets have been flooded. Optimizations such as Gossip-based routing [10] have been proposed to reduce the overhead from flooding in wireless networks, where each node forwards a message with some probability.

2.1.2 Link-State Routing

In link-state routing [35], each node periodically collects information about its neighbors, such as delay, and floods this information to all other nodes in the network. Once a node has accumulated a full set of link-state packets, it can maintain a view of the complete topology of the network. A shortest path algorithm is used to constuct routes to destinations.

2.1.3 Distance-Vector Routing

In distance-vector routing [35], a node periodically monitors its distance to its neighbors (e.g. in terms of delay) and instead of flooding the whole network broadcasts this information only to its neighbors. Receiving nodes use this information to update their routing tables, using a shortest path algorithm. A node performs *triggered updates* as well as *periodic updates* when a change occurs in its routing table.

Compared to link-state routing, distance-vector routing provides each node an aggregate view of the network, without causing too much overhead in the network. However, distance-vector routing has a serious drawback, known in the literature as the “count-to-infinity” problem. In particular, distance-vector routing takes a large number of update messages to detect that a node is unreachable.

2.2 Proactive vs. Reactive Routing Protocols

Routing protocols for mobile ad hoc networks can be categorized into three classes: *Proactive, Reactive and Hybrid.* Protocols differ based on how they handle *Route Discovery* and *Route Maintenance.* Route discovery sets up initial routes or searches for new routes when the old ones break. Route maintenance manages accurate information about existing routes and also supports error recovery, which is initiated when a broken route is detected.

In proactive protocols, routing information is exchanged among neighbors periodically or each time a change occurs in network topology. Essentially, proactive protocols continuously perform route discovery and route maintenance by frequently updating reachability information. While protocols in this category have the advantage that routes are immediately available when requested, they suffer from high control overhead. Specifically, proactive protocols maintain routes regardless of the presence of traffic, thus consume valuable resources such as bandwidth and power even if the network is idle.

Hybrid protocols aim to reduce the control overhead by balancing proactivity and reactivity. However, all current protocols rely on proactively acquiring at least one-hop neighborhood information. In comparison, on-demand routing protocols reduce routing overhead by tying route discovery to network communication [33, 4]. Therefore the remainder of this thesis focuses on on-demand routing.

2.3 Local Recovery in On-Demand Ad Hoc Routing Protocols

On-demand protocols initiate a route discovery only when a new route is needed for initial route set-up or due to a broken route. However, on-demand protocols rely on flooding for route discovery, which causes high routing overhead and interference with ongoing traffic. While there exists techniques for reducing the cost of the initial route set-up, flooding may still be needed to find the destination. On the other hand, repairing broken routes is a good target for optimization since any repaired route alleviates the need for full route discovery from the source. However, it is important to ensure that any route recovery technique costs less in terms of message overhead, delay and potentially even power, than route set-up via flooding. Additionally, if the route repair selects a broken route, the protocol will have to fall back on flooding and any benefits from route recovery will be lost and the flow will experience extra delay.

Several protocols implement solutions to the flooding problem in on-demand routing by providing more efficient error recovery and route discovery mechanisms. These protocols can be categorized into three main classes:

- Limited broadcast: Route discovery is initiated by intermediate nodes. The broadcast range is limited and does not flood the whole network [29, 36, 17].
- Multipath routing: Multiple routes are discovered and cached in a single route discovery [19, 23, 26, 31, 18].
- Local error recovery mechanisms: Route errors are handled locally at an intermediate node instead of end-to-end error recovery at the sender [27, 3, 34].

Table 2.1 provides a taxonomy of local recovery protocols. The rest of the section discusses protocols in these three classes in more detail.

2.3.1 Limited Broadcast Approaches

Protocols such as AODV [29] and ABR [36] provide route recovery by allowing intermediate nodes to initiate a search to replace a failed route. In AODV, an intermediate node attempts to repair a path if it is no further than "maximum repair" hops away from the destination when a link break occurs. To repair the route, the node broadcasts a route request message with a limited time-to-live. ABR (Associativity-Based Routing) employs a similar technique but provides route recovery with routes that tend to be more long-lived by basing route decisions on a measure of next-hop mobility. We believe both methods are too bandwidth consuming, since even with limited broadcast, flooding scheme can deliver the "localized" query messages to a large number of nodes, leading to high routing overhead. Other mechanisms that try to localize query flooding to a limited region of the network also exist [17, 5]. However, these mechanisms either require location information (e.g., obtained using GPS) [17] or fine tuning of parameters to determine the query region [5].

2.3.2 Multipath Routing Approaches

Multipath routing mechanisms [23, 19, 18, 26, 31] discover and cache multiple routes with a single route discovery. When a broken route is detected, it is expected that other routes will be available from the cache and a new route discovery due to a broken route is only needed when all cached routes to a destination break.

Although multipath routing protocols reduce the number of route discovery attempts in certain situations, they may suffer in the presence of mobility, incurring additional packet drops and delay. This is mainly due to the fact that the multiple routes are cached during the route discovery phase. If a significant amount of time has passed between route discovery and route recovery, it is likely that the cached routes will be invalid due to frequent topology changes. Without any mechanism to keep the caches up-to-date, even if multiple routes are discovered, a route discovery attempt may be inevitable.

2.3.3 Local Error Recovery Mechanisms

To enhance both regular and multipath routing protocols, local error recovery mechanisms can provide more robust route recovery during route failures in mobile environments. In AODV-BR [18], nodes snoop route reply messages to create alternate next hops to the destination. When a node detects a broken route, it performs a one hop broadcast to its immediate neighbors, which unicast the packet to their next hop if they have an entry to the destination. However, the alternate routes may be stale since they were only populated during route discovery. A similar methodology is used in WAR

(Witness-Aided Routing) [3] that designates nodes in the neighborhood of an ongoing communication as witness nodes, which buffer packets for the flow and deliver the packets themselves to the next hop if a failure occurs. This requires witness nodes to maintain state and dedicate storage for communication in which they are not involved. In both AODV-BR and WAR, the same packet may be received by the destination several times and cause unnecessary overhead.

NSR (Neighborhood-Aware Source Routing) [34] uses link-state information to enable relay nodes to repair broken routes. NSR maintains current two-hop neighborhood information for all nodes via HELLO messages. While NSR can be considered a "partial multipath" routing protocol since it proactively stores multiple routes to some nodes, the HELLO messages are very expensive.

DSR can be considered an "implicit multipath" routing protocol due to the potential of caching multiple routes. DSR also provides a route salvaging option that enables intermediate nodes to recover from route failures locally by searching for alternate routes in their caches. Although such salvaging may reduce packet drops due to route failures, since nodes immediately send a route error back to the source, salvaging in DSR does not achieve any reduction in the frequency of route discoveries. A recently proposed protocol, CHAMP [37], uses a distributed salvaging algorithm where all nodes temporarily cache packets before forwarding. When a node receives a route error for a forwarded packet and if the failed packet still exists in its "packet cache", the node salvages the packet with an alternate route from its route cache. Therefore, local recovery is achieved by incurring additional storage overhead in relay nodes. Additionally, trying to recover from the failure at all upstream nodes using stale route cache information may incur extra delay when it is best to initiate a route discovery.

TORA [27] is an early routing algorithm that decouples network flooding from the rate of topological changes and thereby localizes the reaction to route failures. However,

	Limited Broadcast	Multipath Routing	Localized Error Correction
AODV	Yes	No	No
ABR	Yes	No	No
LAR	Yes	No	No
AOMDV	No	Yes	No
SMR	No	Yes	No
AODV-BR	No	Yes	Yes. Neighbors forward failed packets via an alternate in their routing table.
NSR	No	Yes. Multiple partial paths are stored as link state information.	Yes. Link state information for 2-hop neighbors.
WAR	No	No	Witness nodes
DSR	No	Yes/No. Multiple routes are implicitly stored in route caches.	DSR route salvaging
CHAMP	No	Yes	Distributed route salvaging
TORA	No	Yes. Routed to destination form DAGs, which provide multiple route information.	Yes. Link reversal mechanism.

Table 2.1 Classification of Route Maintenance Mechanisms in On-Demand Routing Protocols

TORA uses a link-reversal algorithm that relies on synchronized clocks. Such synchronization may be difficult to achieve among nodes in an ad hoc network and therefore limits TORA's applicability. Another drawback with TORA is the potential for instability when multiple nodes concurrently detect link failures and re-build routes based on each other.

Although each of these local error recovery mechanisms has limitations, we believe that localization of recovery is necessary for the scalability of ad hoc routing protocols. To this end, we present by-pass route recovery, which achieves effective local route recovery without using proactive HELLO messages.

2.3.4 Route Repair with Local Recovery

Another important issue with local recovery, other than the actual mechanism to repair a broken route, is to determine which actively used routes are affected by a link failure and if they should be repaired along with the broken route. For routing protocols that maintain routing information in intermediate nodes, it is challanging to determine active routes (i.e., the routes in use) when routing information is not frequently updated. While a proactive approach fixes all routes in routing tables regardless of whether they are in use, a reactive approach only fixes the broken route and not possible soon-to-fail routes. Therefore, a reactive approach may incur additional overhead by initiating multiple route repairs for the same link failure, whereas a proactive approach may create unnecessary overhead by repairing unused routes. In AODV, the repairing node may elect to repair all routes that are affected by the broken link before a data packet is received or wait until a data packet is received. In other words, the node can act proactively or reactively. NSR and WAR are reactive and repair routes only when they fail. We believe there is a need for a mechanism that provides a balance between proactivity and reactivity. SLR, similar to DSR, identifies some actively used routes that should be repaired by checking the interface queues for packets with routes that contain the broken link.

Although the main focus of this thesis is to provide local recovery only when link failures occur, a second approach preemptively activates route repair when the signal strength of a link goes below a certain threshold (i.e. route recovery is initiated before the link actually breaks). Preemptive route recovery may enable a smooth handoff to a new route for connections by switching to a new route before a link breaks. With this method, it is possible to avoid packet loss if the link-breakage prediction algorithm allows enough time to find an alternate route. We will implement the preemptive approach as part of future work.

Chapter 3

By-pass Route Recovery

The goal of any local recovery mechanism should be to repair broken routes in a way that reduces control overhead and chooses valid routes. Therefore, a routing algorithm with local route recovery should possess the following characteristics to enable efficient route recovery for link or node failure.

- Repair with cached routes when available
- Repair with local route recovery when cached routes are not available
- Repair all active routes affected by broken links
- Utilize bandwidth efficiently

Utilization of both route caches and local route recovery mechanisms is essential for providing robust recovery in ad hoc networks. The benefit of using route caches is two-fold. First, when a link failure occurs, an alternate route may be immediately available. Second, using route caches can provide reduction in the control overhead that is required to repair a route. However, in high mobility environments route caches may contain stale routes. Therefore, route recovery should not entirely rely on route caches and local error recovery mechanisms should take over when route caches do not provide useful information.

Although the main concern of local recovery is to repair the broken route as fast as possible, it is important to alleviate the effects of a broken link on future transmissions by repairing all active routes that use the broken link. Finally, a local recovery mechanism should use bandwidth efficiently by incurring minimum overhead.

By-pass route recovery follows these guidelines to achieve efficient and effective route recovery. Although by-pass route recovery is not limited to any routing protocol, the specifics of the mechanism depends on the characteristics of the underlying on-demand routing protocol. In this section, we describe how by-pass routing can be integrated into a source routing protocol.

When a node detects a broken route, it searches its route cache for an alternate route to the destination. If a route exists, the node patches the broken route with the alternate route. If the node is not able to repair the route from its route cache, it initiates by-pass route recovery by querying its immediate neighbors to see if they have a link to any of the nodes on the downstream route to the destination. As replies arrive, the node patches the routes affected by the link failure with the received connectivity information. When the patched packets reach the destination, the new route information is added to an enhanced route error packet and sent back to the source to inform it about the broken link and successful route change.

Local repair of broken routes may result in an increase in route lengths to destinations. In by-pass routing, although the sources are informed of repair information, they are not forced to use a longer route if they have a shorter route cached. Specifically, by-pass route recovery aims to reduce the frequency of route discoveries, while allowing the node to use shorter routes when possible.

An illustration of local route recovery using caches is shown in Figure 3.1. Initially, the flow from Node S to Node D uses the route “S-J-K-L-M-N-P-R-D”. When the link breaks between Node M and Node N, Node M detects the failure and attempts to patch

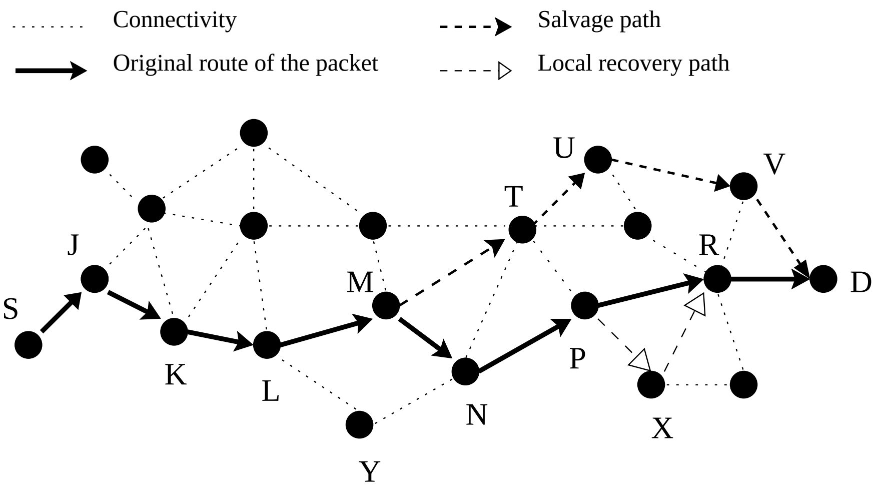


Figure 3.1 Error recovery example

Node M: Cache state

M→L→Y→N→P→R→D

M→T→U→V→D

Patch process

S→J→K→**L**→**M**→**L**→Y→N→P→R→D

New Route: S→J→K→L→M→T→U→V→D

a) Route patching with alternate cached route

Node P: Query replies

P→X, P→X→R

Patch process and new route

S→J→K→L→M→N→**P**→**X**→**R**→D

b) By-pass local recovery

Figure 3.2 Illustration of route recovery

the route by using an alternate route from its cache to destination D (see Figure 3.2). When Node M finds a route without loops, Node M salvages the packet with the "S-J-K-L-M-T-U-V-D" route.

Figure 3.1 also illustrates an example of by-pass recovery. Again, Node S initially uses the route "S-J-K-L-M-N-P-R-D". When the link between Node P and Node R fails and Node P does not have an alternate route in its cache to the destination, Node P triggers a local query to its immediate neighbors. The neighbors reply if they have active links to any of the downstream nodes on the broken route. In Figure 3.1, Node X reports its connectivity with Node R to Node P. Node P patches the route accordingly and the packet is first forwarded to Node X and then to Node R to reach the destination (see Figure 3.2).

Chapter 4

Source Routing with Local Recovery (SLR) for Ad Hoc Networks

To evaluate the benefits of "by-pass" recovery in a full protocol, we implemented a prototype, Source Routing with Local Recovery (SLR) as an extension to DSR. Although SLR uses DSR as the underlying protocol to demonstrate the feasibility and effectiveness of local recovery with by-pass routing, SLR implements a different route selection and maintenance scheme. The essential aspect that distinguishes SLR from DSR is *error handling*. On a route failure, DSR reports the error to the original sender immediately, even if it salvages the failing packet. This places the entire responsibility on the sender, regardless of where the error occurs. SLR uses a three level error handling mechanism (salvaging using route caches, local recovery, and error reporting) and informs the sender of a need for route discovery only when salvaging and local recovery fails. Although SLR makes use of source routes in packet headers, the ideas presented in this paper can be applied to other routing protocols. For example, for AODV, the node detecting a route failure can query neighbors if they have connectivity to either the other end of the broken link or the destination. We plan to evaluate the effect of by-pass routing on other protocols in the future.

In this section, we give a brief overview of relevant aspects of DSR and describe the operation of SLR in detail.

4.1 Overview of DSR Protocol

DSR is an on-demand routing protocol that only establishes routes to destinations for active flows. When a node wants to find a route to another node, it broadcasts a *Route Request* message to all its neighbors. Each node that receives a route request that it has not seen before appends its own address to the source route in the packet and re-broadcasts the packet. In this way, the route request is flooded throughout the network. When a node that knows a route to the destination or the destination itself receives the route request, a *Route Reply* is returned to the source.

In DSR, when a node detects a broken link to a neighbor, the corresponding route entries are deleted from the routing table and the source nodes that are actively using that link are informed of the link failure.

Several optimizations for route maintenance have been proposed by the authors of DSR [1]. SLR utilizes all described optimizations, which are as follows:

- Route Salvaging: The node that detects a link failure may salvage the data packet by replacing the source route in the header with an alternate route from its route cache. The node also sends a route error packet to the original sender, identifying the link over which the packet could not be forwarded. To prevent the possibility of the infinite looping of a packet, a *salvage count* is used to limit the number of times a packet can be salvaged.

- Increased Spreading of Route Error Messages: A source node receiving an error packet propagates this error with the next route request packet.

- Automatic Route Shortening: When a node overhears a packet and it is not the intended next-hop destination for the packet but exists later on the packet's source route, it sends the shorter route as a gratuitous route reply back to the source.

- Snooping: Finally, a node can snoop data packets by operating in promiscuous mode. Snooping is used to learn additional routes to each destination and to check for route error messages.

4.2 Operation of SLR

SLR uses three mechanisms that work together to allow efficient recovery from route failures:

- MAC Neighborhood Cache: This mechanism is responsible for determining the state of links to neighboring nodes.
- Route Cache: Recently discovered routes are cached to avoid expensive route discovery.
- Error Recovery:
  - Route Salvaging: Any relay node can use an alternate route from its route cache to salvage a route that contains a broken link.
  - Local Error Recovery: Any relay node triggers by-pass route recovery if route salvaging is not possible to fix the route failure. A broken route is repaired utilizing link-state information collected from immediate neighbors.

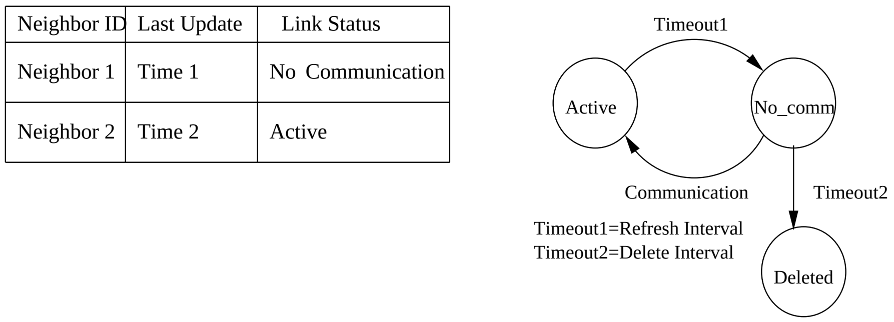
4.2.1 MAC Neighborhood Caches

In SLR, the MAC neighborhood cache provides recent link and node status information. A node may infer its active neighbor set from the information exchanged as a part of link sensing and record this information in its MAC neighborhood cache. Each node maintains neighborhood information by caching a list of neighbors with a *last-update* timestamp, indicating the time the node last heard from a particular neighbor.

To maintain the most recent neighborhood information, a node updates its MAC neigborhood cache when a control or a data packet is received from any neighbor and if a cache entry has not been updated within a given *refresh interval.* Cache invalidation is a two stage process. If the refresh interval expires without any sign from a neighbor, the neighbor's *link status* is set to *no communication.* Once in *no communication* state, if there is no communication from the corresponding neighbor during the *delete interval*, the neighbor is deleted from the MAC neighborhood cache. On any activity, the link status is marked as *active* (see Figure 4.1). The rationale behind this two-stage process is that it provides a second chance for nodes that have not been in active communication recently. The need for a two-stage process becomes clear when we discuss the utilization of MAC neighborhood caches.

In SLR, link sensing is accomplished through passive inference. In other words, nodes operate in promiscuous mode and eavesdrop on messages to update MAC neighborhood caches accordingly. Another commonly used method for neighbor detection is a periodic exchange of link state advertisements (e.g. HELLO messages). However, compared to passive inference, HELLO messages are more energy and bandwidth consuming and therefore not used in SLR.

To successfully populate the MAC neighborhood caches, some knowledge of the underlying MAC protocol is useful. While SLR is not limited to any specific MAC protocol, SLR was implemented based on IEEE 802.11 MAC [6], which uses RTS (Request-to-send) and CTS (Clear-to-send) messages to provide a form of *virtual carrier sensing* and channel reservation to reduce the impact of the well-known *hidden terminal* problem. After the RTS/CTS exchange, data packet transmission should be followed by an ACK. A node



Neighbor ID	Last Update	Link Status
Neighbor 1	Time 1	No Communication
Neighbor 2	Time 2	Active

Figure 4.1 MAC Neighborhood cache and cache update process

listens for RTS/CTS/DATA/ACK messages to infer a neighbor's existence and updates its MAC neighborhood cache.

While most neighbor information can be acquired by simply snooping packets, IEEE 802.11 has one behavior that requires additional state to determine the originator of a packet. In IEEE 802.11, while RTS and DATA messages carry both sender and receiver information, CTS and ACK messages only contain receiver information. When a node overhears a CTS (ACK) message, the node checks if it is the destination and if it has recently sent the corresponding RTS (DATA) message. However, without any additional mechanism, a node that is not the sender of the RTS or DATA message cannot determine the originator of CTS or ACK message simply by looking at the message. Due to this problem, a node can cache only senders but not the receivers on a given route. To utilize CTS and ACK messages, all nodes in SLR follow the same method as the actual senders of RTS and DATA messages. Basically, a node that has overheard an RTS (DATA) message records the identity of the receiver and the sender of this message. When the node overhears the CTS (ACK) message, it checks if the recorded sender information matches the receiver information of the CTS (ACK) message. If there is a match, the receiver of the RTS (DATA) message is a neighbor and should be in the MAC neighborhood cache. Using this method, MAC neighborhood caches represent the current neighborhood of a node more accurately (see Figure 4.2). However, there also exists edge cases when a node cannot discover an actively communicating neighbor. This situation occurs when a node is only able to snoop a CTS (ACK) message but not the corresponding RTS (DATA) message. Figure 4.3 illustrates such a case. This case only affects the optimization and not the correctness of MAC neighborhood caches. Also note that, in Figure 4.3 although RTS and CTS messages are ordered in such a way that Node C overhears one RTS message and then a CTS message using the proposed mechanism, it does not mistakenly assume Node E is an active neighbor[1].

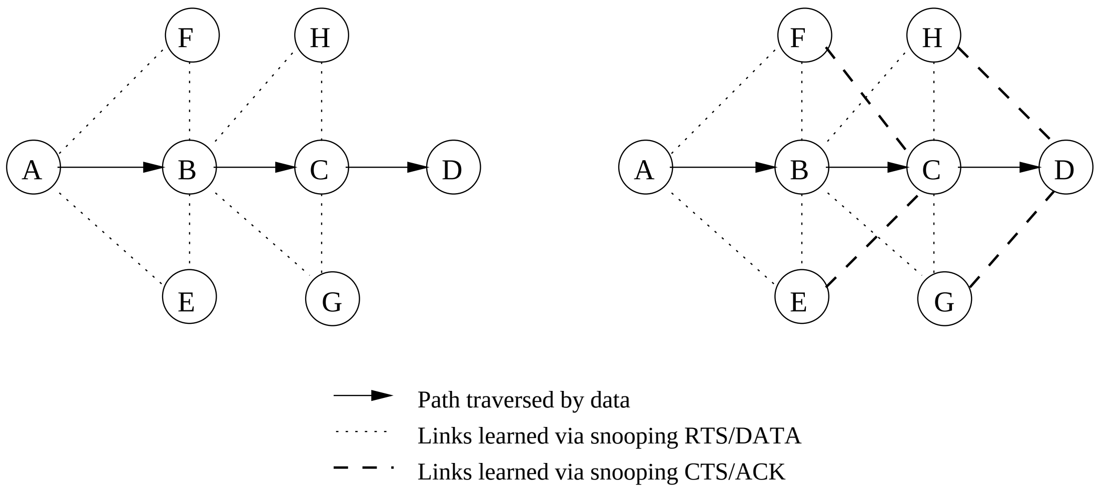

F
H
A
B
C
D
E
G
F
H
A
B
C
D
E
G
Path traversed by data
Links learned via snooping RTS/DATA
Links learned via snooping CTS/ACK

Figure 4.2 Link-state information learned from RTS/CTS/DATA/ACK messages

In SLR, MAC neighborhood caches are used to enhance route selection. A node searching for a route in its route cache checks if the next hop of the route exists in its MAC neighborhood cache. More specifically, a node can identify a stale route if the next hop is not in its neighborhood. Therefore, the delete interval of MAC caches serves to determine the validity of routes in route caches. The reason for choosing the delete interval as a timeout for route caches is to utilize routes even when link status is *no*

[1]It is assumed that the effective carrier-sense range of radios is at most 500m, so that the sequence of RTS/CTS messages is possible to occur as shown in Figure 4.3.

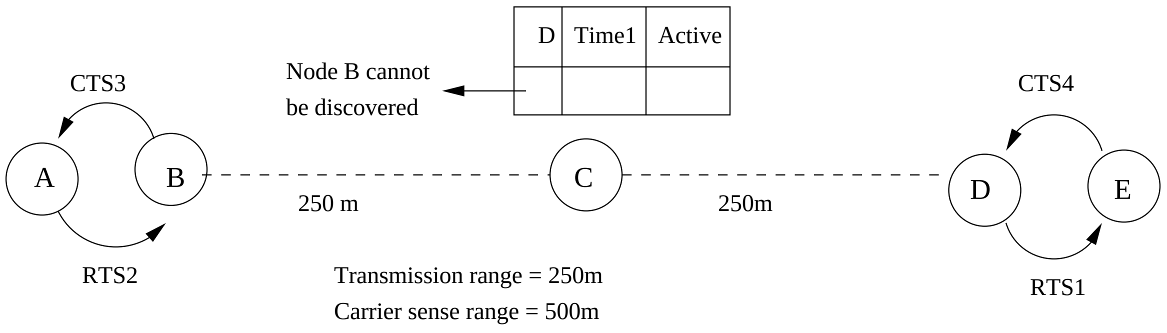


Figure 4.3 An example of when a node cannot discover an active neighbor

communication and thereby give a chance to route caches when there is no communication in the network.

MAC neighborhood caches are also important during local recovery, where a node queries its neighbors to see if they have connectivity to any of the downstream nodes on the broken route to the destination. In this case, the status of the links should be taken into account and only nodes that have active links to any of the nodes in the query message should reply, since query replies must carry the most recent connectivity information to facilitate efficient local recovery.

4.2.2 Route Caches

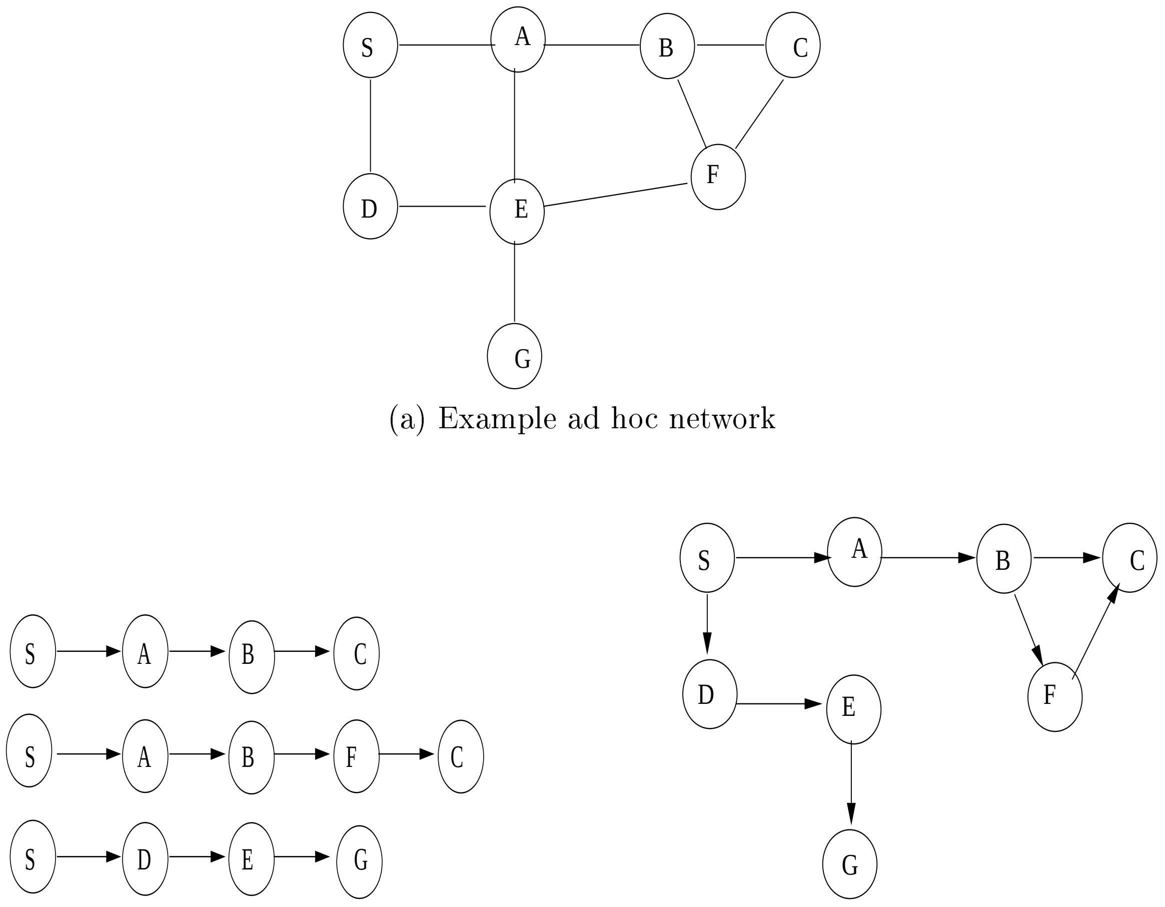
Route caching is an important part of any on-demand routing protocol. In on-demand routing, a newly discovered route should be cached so that it may be reused the next time it is requested. A cache hit with a valid route saves bandwidth and energy consumption by eliminating the message overhead from route discovery and similarly reduces end-to-end delay since the packet can be sent immediately without waiting to find a route. On the other hand, route caching introduces the problem of effective strategies for managing the structure and the contents of the cache since node mobility can change network topology and possibly invalidate cached routes. The use of an invalid cached route incurs additional bandwidth, energy consumption and delay. Propagating data messages or route replies with stale routes can also pollute other caches when intermediate nodes promiscuously listen to all packets and cache route information. Therefore, careful design choices must be made about the cache *structure* and cache *timeout* mechanisms. However determining the optimal timeout interval for route caching is a complex problem and an active area of research [20, 12, 21].

Since DSR is the underlying routing protocol for SLR, the rest of the section discusses caching strategies proposed for DSR. In DSR, as mentioned before, routes are learned via explicit route replies or, snooping data and route reply packets destined to other nodes. Two alternatives, *Path cache* and *Link cache*(see Figure 4.4), have been proposed for implementing route caches in DSR [12]. Path caches store each complete per-destination route information. Link caches utilize the potential information from learned routes more efficiently by creating a unified graph from all links in all routes. Although path caches are simple to implement and easily guarantee loop-free routes, link caches provide nodes a more detailed view of the current network topology. Link caches with an adaptive timeout mechanism (i.e., a link's timeout is chosen according to its stability) have been shown to be more accurate than other types of link caches and path caches [16, 21, 12]. However, generally link caches require more complex route search algorithms (such as Dijktra's shortest path algorithm) compared to linear search algorithms. On the other hand, the performance evaluations in [21, 12] of route expiry mechanisms are only based on link caches (i.e. none of the path caches employ a timeout mechanism). Motivated by [24], we believe that path caches will also benefit from a timer-based expiry approach so that stale routes are not used in response to route queries or to salvage broken routes.

It must be also noted that there might be cases when stale routes exist in route caches even if a timeout mechanism is employed. For example, in [24], Marina and Das point out that there is no way to determine the freshness of routes in DSR when intermediate nodes promiscuously listen to all packets and cache route information. Even if a route error message erases invalid caches, an "in-flight" data packet that carries the same stale route would put the route back in caches. *Epoch numbers* [14] are proposed to prevent the re-learning of a stale link after having heard the link has broken. However SLR does not employ such a numbering mechanism, since one of the main goals of SLR is to reduce the reliance on route caches. SLR utilizes a local recovery strategy when caches contain obselete information, which enables more effective information correction in route caches as well as error recovery. For this reason, the implementation of SLR uses a simple path cache known as Path-Gen-34 [12] instead of more complex data structures such as link caches. We will investigate the effect of link caches on local recovery as a part of our future work.

Path-Gen-34 is composed of two separate caches: a *primary cache* and a *secondary cache*. Routes learned as a result of route discoveries are inserted into the primary cache. Routes learned opportunistically (e.g., by snooping) are inserted into the secondary cache. On a route search, both primary and secondary caches are searched for the shortest path. When there is a cache hit in the secondary cache (i.e., the shortest path exists in the secondary cache), the route is promoted to the primary cache. Each cache runs an independent replacement strategy based on a round-robin scheme. The division of the cache into primary and secondary caches prevents opportunistic routes from competing for cache space with routes of known value to the node.

The original Path-Gen-34 does not employ a timeout mechanism, so routes in the caches are not guaranteed to be fresh. However, SLR uses MAC neighborhood information to determine the validity of routes. Basically, a source selects a route from its route cache as the source route of the flow if and only if the next hop exists in its MAC neighborhood cache. With this method, even if stale route information enters the route caches, it is not utilized unless the next hop is in the neighborhood of the source.



(a) Example ad hoc network

(b) Path cache

(c) Link cache

Figure 4.4 Path Cache and Link Cache Data Structures for a Node S

Figure 4.4 illustrates an ad hoc network, where the link cache and the path cache is formed after Node S discovered the following routes: “S-A-B-C”, “S-A-B-F-C”, “S-D-E-G”. A problem that occurs with the use of timeout mechanisms in caches is the potential loss of useful information, which occurs when invalidating unused routes. In Figure 4.4, Node S chooses the shorter path “S-A-B-C” between two possible routes to Node C. Since the route “S-A-B-F-C” is never used, the links B-F and F-C will expire from the link caches of the source and intermediate nodes. After the expiration of these links, if the link between Nodes B and C fails, the link cache cannot provide an alternate route and a route discovery is necessary. However with by-pass routing, Node F overhears data from source Node S to destination Node C, and so caches Node B and Node C as its neighbors. Therefore, Node B can discover an alternate path to Node C by doing a local query to its neighbors.

4.2.3 Route Recovery

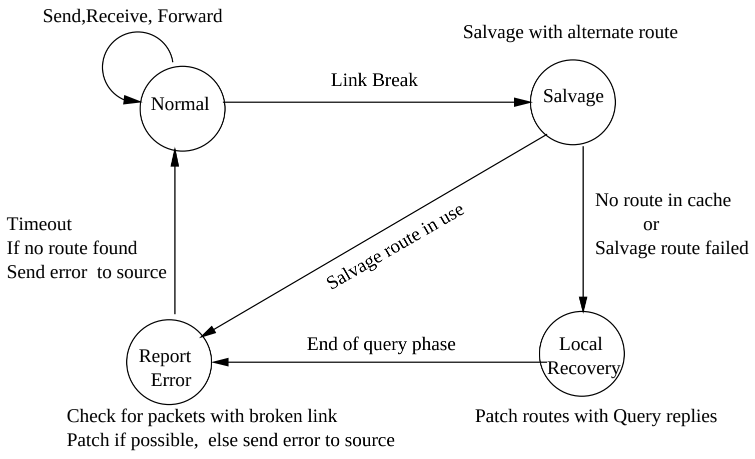


Figure 4.5 Protocol State Diagram

In SLR, error recovery is a three stage process (see Figure 4.2.3). When a node detects a broken route, it first tries to recover from the failure by salvaging the packet using its route cache. This attempt is recorded in a “fail-record table” as <*source, dest, route, method=salvaging*>. When salvaging fails or is not possible, the node buffers the packet and all packets that are affected by the broken link in its interface queue in a “fail-packet buffer”. The next step is to query neighbors for connectivity information to downstream nodes on the broken routes. As query replies arrive, the node patches the broken routes and sends packets out using the new routes.

The rest of this section describes route salvaging and local recovery in detail.

4.2.3.1 Route Salvaging

While DSR uses route salvaging as an optimization for route maintenance, a node still sends an error message back to the source to indicate the broken link. The motivation for route salvaging in DSR is to provide the necessary time for a source to complete a new route discovery, while reducing the number of packets dropped due to link failures. However, salvaging is a temporary solution and the source is expected to find a new route.

In DSR, a node salvages a packet by simply replacing the source route with an alternate route from its cache, and thereby loses the first (successful) part of the source route. The main goal of SLR is to provide the source with a stable route as well as to inform the source of the route failure due to the broken link. To provide such information, a node searches complete loop-free routes to the destination to salvage a packet, preserving the first part of the route (i.e., from the source to the node dectecting the route failure). More specifically, in SLR, the node detecting the route failure looks for an alternate route in its cache to patch the broken route (see Figure 3.2).

In SLR, a packet can be salvaged only once. This is important to reduce the use of route caches for route recovery. Failure of the salvaged route serves as a warning for the node detecting the failure to perform local recovery instead of relying on its route cache.

If the salvaged packet arrives at its destination, the destination in turn sends back an enhanced route error message to the source to indicate the salvaged route as an alternate to the broken route. When forwarding this information, the repairing node snoops on the packet to update the fail-record with the repair information.

If new packets that use the broken link in their source routes arrive at the node that detected the route failure, the repairing node continues salvaging packets with alternate routes. On the other hand, if no repair information is available (i.e., no acknowlegment is

```
Fail-Buffer Check
For each packet p in fail-buffer
    IF salvage == enabled and !p.salvaged
        IF Patch(p.route, salvage route)
            Send(p, salvage route)
        ELSE
            p.route = salvage route or Drop(p)
            SendRateLimited(ERROR, p.source)
    ELSE
        Drop(p)
        SendRateLimited(ERROR, p.source)
```

Figure 4.6 Fail-Buffer Check Algorithm

received from the destination) the node should prevent packets from using the failed link and send back error messages to sources trying to use the recently broken link. In this case, the node has two options about the fate of the packet: either replace the packet's source route with an alternate route in its route cache, if it exists, as DSR salvaging does, or drop the packet. SLR uses the first option, which is more aggressive, to give one last chance to the packet (see Figure 4.6). Specifically, before forwarding a packet, a node checks its fail-record table for recent failures and salvages accordingly. If the source does not switch to a salvaged route, the repairing node should send error messages back to the source in a rate-limited fashion until the fail-record expires (see Figure 4.7).

4.2.3.2 Local Recovery

A node performs local error recovery under the following conditions:

- The node does not have an alternate path to perform salvaging,
- the packet has already been salvaged,

```
Forward(Packet p)
/* FailTable update */
      IF p == ERROR-enhanced
        IF FailTable.contains("p.dest-p.src")
          patch = GetPatch(p.newRoute, p.BadLink)
          FailTable.updateTable(patch, p.BadLink)
     /* Avoiding route failures */
      IF FailTable.contains(node, p.nextHop)
        IF FailTable.existPatch("node-p.nextHop")
          IF Patch(p.route, patch)
            Forward(p)
          ELSE
            Put(FailBuffer, p)
        ELSE
          Put(FailBuffer, p)
```

Figure 4.7 Changes to Packet Forwarding

```
Recv(Packet p)
      IF p.salvaged or p.patched
        DeadLink(p.BadLink.from, p.BadLink.to)
        ERROR-Enhanced =p.route + p.BadLink
        SendRateLimited(ERROR-Enhanced, p.src)
```

Figure 4.8 Changes to Packet Reception

- salvaging is disabled.

When one of the above conditions is satisfied, the node detecting the route failure sends a one-hop broadcast message to its neighbors, querying if they are neighbors with any of the nodes in the failed route. The local recovery is recorded in the “fail-record” table as *<source,dest, route, method=by-pass routing>*.

The one-hop broadcast packet includes aggregate information of all the active failed routes affected by the link failure. More specifically, on a link failure, a node searches its interface queue for packets that carry a source route that contains the broken link. The failing packet and all packets in the interface queue that need to use the broken link are stored in a “fail-packet buffer”. A list of all nodes existing on such soon-to-fail routes is sent with the query message. This is done to find alternate paths for all effected active connections. Thus, with by-pass route recovery, a reply message can fix multiple active routes.

The nodes that receive the local query message search their MAC-neighborhood caches for a neighbor listed in the query message. The node includes all such neighbors in its reply message (see Figure 4.9). To avoid query reply storms, nodes use a random backoff algorithm and send a query reply only if they have not overheard another node’s reply to the same query.

When the querying node receives a reply, it checks its “fail-packet buffer” for failed packets and repairs as many broken routes as possible with new link-state information. For flows that have switched to a new route, one packet is marked as “patched” to warn the destination of the broken link and the route change (see Figure 4.10). The node also updates its route cache with the new connectivity information.

When patched packets arrive at their destinations, enhanced route error messages are sent to the source that carry both the broken link and the repair information. The

```
LocalQuery-Recv(Query Q)
FOR $i$ = 0 to *length(Q)*
        IF Q[i] == id or
            MACNeighborhoodCache.contains(Q[i])
          QueryReply += Q[i]
          *replyCount++*
          Put(QueryReplyBuffer, QueryReply)
          BackoffSend(QueryReply)
```

Figure 4.9 Replying to local queries

```
QueryReply-Recv(Qreply)
        FOR each node $i$ in Qreply
          patch[$i$] = id + sender(Qreply) + node[$i$]
        FOR each packet p in FailBuffer
          IF Patch(p.route, patch[$i$])
            *mark**(p, patched)
            Send(p, p.newRoute)
        UpdateRouteCache(patch[$i$])

        * p is marked, if a marked packet has not been sent for
     the same flow recently
```

Figure 4.10 Processing query replies

fail-record entries corresponding to the repaired routes are updated with the repair information. The records for repaired routes are not deleted from the fail-record table until they expire. The repairing node uses these entries to identify if the source node switched to the new route successfully. If a repairing node does not receive any query reply messages or enhanced route error messages indicating the repaired routes are indeed an alternate to the failing routes before a timeout occurs, a route error message is sent back to the source (see Figure 4.6).

Chapter 5

Simulation Study

The effectiveness of local recovery via by-pass routing can be evaluated by its impact on the following properties:

- *Packet delivery ratio:* The ratio of data packets delivered to the destinations to those generated by the sources. Packet delivery ratio is important since it quantifies packet loss rate and characterizes the maximum throughput the network can support.

- *Average end-to-end delay:* The difference between the time a packet was sent by the sender and the time it was received at the destination, including all possible delays due to buffering during route discovery, queueing delays, retransmission delays at the MAC layer, propagation times and delays incurred by local route recovery.

- *Goodput:* The overhead per byte, capturing the effect of routing overhead and of route length. Goodput is calculated as the ratio of the number of data bytes received to the overhead of local query messages and enhanced route error messages, DSR control overhead of route requests, route replies and route errors and the overhead of forwarding data along each hop.

- *Routing overhead:* The ratio of local recovery and DSR control overhead to the amount of data received at the destinations in bytes. Different than goodput, routing overhead does not include the effect of hop count (i.e., the byte overhead of forwarded data messages is omitted). Each transmission of a packet counts as one transmission. This metric determines the scalability of the protocol.

- *Average hop count:* The average number of hops a data packet traverses to reach its destination.

5.1 Simulation Environment

To evaluate the performance improvements of by-pass routing, we compare SLR with DSR. We compare the following schemes:

- *DSR (cache):* Dynamic source routing.
- *DSR (nocache):* DSR without intermediate nodes replying to route requests from route caches. Salvaging is also disabled.
- *SLR (cache):* Source routing with local recovery.
- *SLR (nocache):* SLR without salvaging and propagation of route replies from route caches.

We implemented SLR in the ns-2 [2] network simulator using the CMU [1] wireless extension. Our simulation results represent an average of five runs with identical traffic models, but different randomly generated network topologies. We use the *random way-point* [4] mobility model, with a speed uniformly distributed between 0-20 m/sec. We will study the performance of SLR with different mobility models as a future work, since the random waypoint model has been shown to have limitations [38]. Table 5.1 lists the SLR parameters.

Table 5.1 Parameters used in SLR simulation

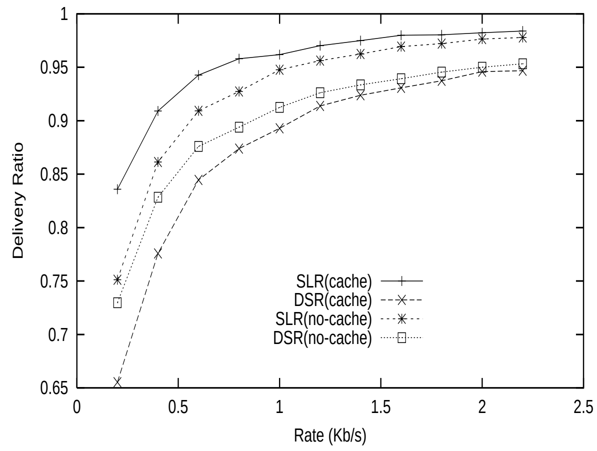
Fail-record: Table size (number of entries)	34
Fail-record: Timeout (s)	1.0
Fail-buffer: Packet timeout (s)	0.02
MAC Neighborhood: Refresh interval (s)	0.05
MAC Neighborhood: Cache timer (s)	3.0

5.2 Impact of Traffic Load

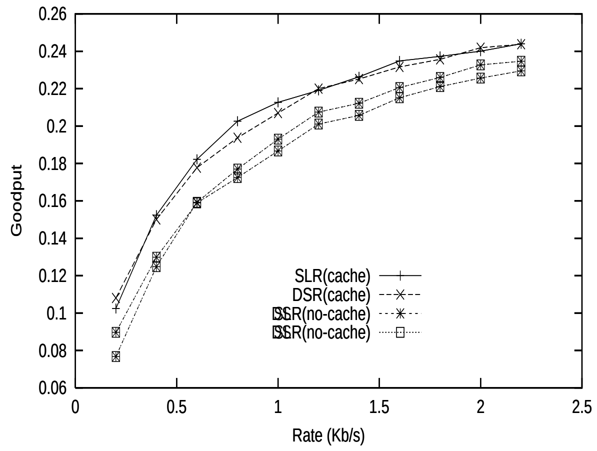
To evaluate the impact of traffic load, we simulate SLR in 1500mx500m network with 60 nodes with long-lived CBR connections at different transmission rates between 0.2Kb/s-2.2Kb/s. All data packets are 128 Bytes. There are 20 connections started randomly between 20s and 25s. Each simulation runs for 600s. A pause time of 60s was used to achieve moderate mobility in the network.

We compare packet delivery ratio, goodput, routing overhead and delay of DSR and SLR with and without the benefit of route caches. Figure 5.1(a) shows the packet delivery ratio as the traffic load changes. Both SLR (cache) and SLR (nocache) achieve high packet delivery ratios compared to DSR (cache) and DSR (nocache). SLR (cache) shows the best performance by increasing the delivery ratio between 3-19%. With SLR (nocache), the increase in delivery ratio is lower in low traffic loads. As the network load, and so amount of communication, increases, the nodes provide more robust recovery with more frequently updated MAC neighborhood caches. We see from Figure 5.1(a), the packet delivery ratio of DSR(cache) is lower than DSR (nocache) and SLR. This indicates that SLR uses route caches more effectively than DSR by tying route validity to up-to-date MAC neighborhood information.

Figure 5.1(b) shows that in terms of goodput, SLR performs very close to DSR. The main reason for comparable performance is due to forwarding more data messages over longer routes in SLR (see Table 5.2). On the other hand, Figure 5.2(a) shows that SLR



(a)



(b)

Figure 5.1 Delivery Ratio and goodput vs. traffic load, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

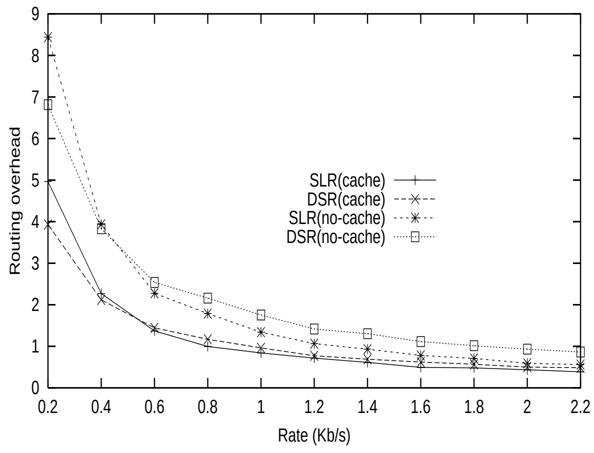
Table 5.2 Average Hop count with different traffic load

	SLR(cache)	DSR(cache)	SLR(nocache)	DSR(nocache)
Hops	3.427	3.144	3.324	3.160

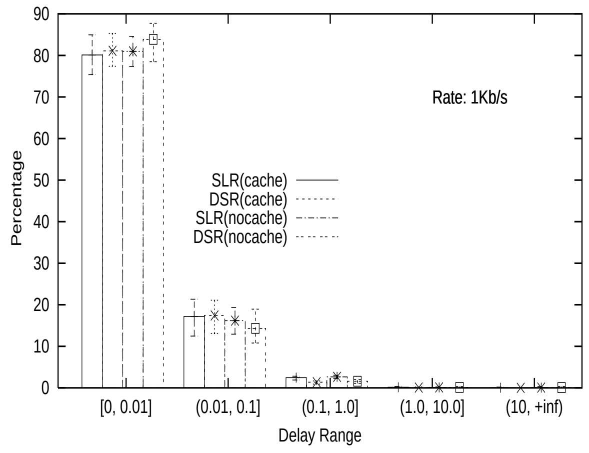
incurs around the same amount of routing overhead as DSR, by replacing route request, route reply and DSR error messages with local query and enhanced route error messages. Therefore, SLR performs significantly better in terms of throughput, by incurring local route recovery overhead that facilitates efficient communication. Another observation is that although SLR uses longer routes, it does not incur significantly higher delays and delivers most of the packets in less than 0.01 seconds. This conforms to our discussion in previous sections that SLR reduces the delays due to route discovery (see Figure 5.2(b)).

5.3 Impact of Mobility

We evaluate the impact of mobility in a 1500mx500m network with 60 nodes and 20 CBR connections with transmission rates of 4Kb/s. The data packet size is chosen as



(a)

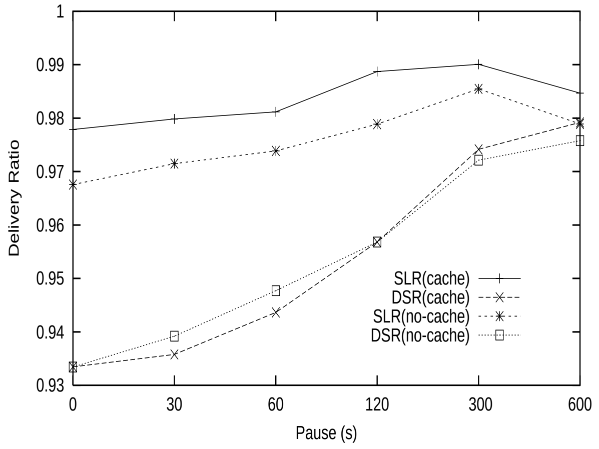


(b)

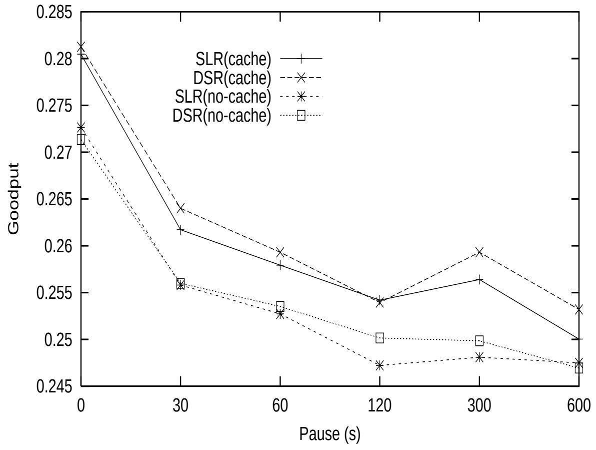
Figure 5.2 Routing overhead and delay vs. traffic load, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

256Bytes. We change mobility rate by setting different values of pause times as 0, 30, 60, 120, 300 and 600 simulation seconds.

As shown in Figure 5.3(a), the packet delivery ratio of SLR (cache) and SLR (nocache) are higher than both types of DSR. While the performance improvement is significantly higher in high mobility, DSR catches up with SLR in low mobility scenarios as the number of broken routes decrease. We see the same effects of routing overhead and hop count on goodput, which indicates that SLR, by localizing the reaction to topological changes, improves the throughput with acceptable overhead. Specifically, Figure 5.4(a) shows that SLR outperforms DSR in terms of overhead. These results match the results shown in Figure 5.2(a) when the traffic load increases in the network, but are plotted in a different scale. The difference in hop count between SLR and DSR shows that SLR does not achieve better goodput due to using longer routes.

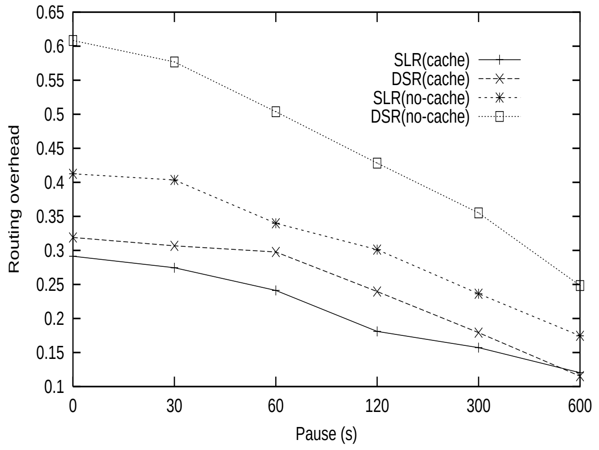


(a)

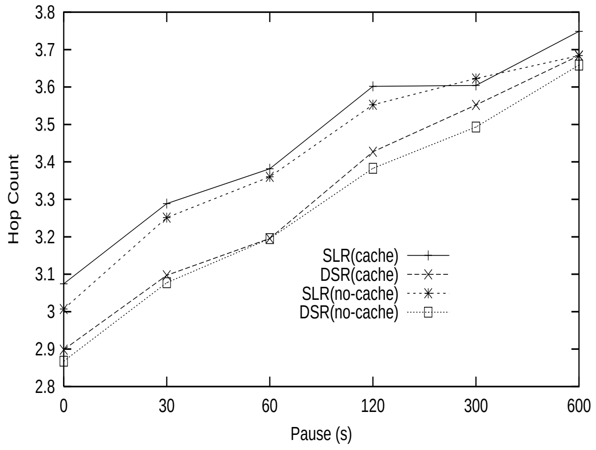


(b)

Figure 5.3 Delivery Ratio and Goodput vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s



(a)



(b)

Figure 5.4 Routing overhead and hop count vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

Chapter 6

Conclusions

On-demand routing protocols are attractive for mobile ad hoc networks. However, their effectiveness is limited by the use of flooding to discover new routes. In this paper, we propose by-pass route recovery that reduces the need to perform route discovery for broken routes. Our primary concern is to provide robustness to route failures.

We implemented a prototype of by-pass route recovery based on source routing. Simulations show that SLR (Source Routing with Local Recovery) achieves significantly higher throughput while maintaining acceptable overhead. The results verify that local recovery is the right approach for route recovery in ad hoc networks. We are currently working on performance evaluations on larger networks. Our initial results show that by-pass routing enables efficient communication by localizing the reaction to topology changes and so provides scalable route recovery.

Our future plan is to investigate the benefits of by-pass routing with other on-demand protocols. We also plan to evaluate the effectiveness our approach by comparing its performance with hybrid and local recovery protocols. Additionally, we will look into other issues related to local recovery such as utilizing link caches in SLR's implementation and further evaluate our scheme with more realistic mobility models. Our future research goal is to study by-pass routing in power-saving ad hoc networks. We believe the advantages gained through by-pass routing due to its localized behavior will enable better power management.

References

[1] *CMU Monarch Extensions to ns.*, http://www.monarch.cs.cmu.edu edition.

[2] *The VINT project, The network simulator - ns2*, available at http://www.isi.edu/nsnam/ns/ edition.

[3] I. D. Aron and S. K. S. Gupta. Analytical comparison of local and end-to-end recovery for reactive protocols for mobile ad hoc networks. *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 69–76, August 2000.

[4] J. Broch, D. Maltz, D. Johnson, J.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '98)*, pages 85–97, October 1998.

[5] R. Castaneda and S. R. Das. Query localization techniques for on-demand routing protocols in ad hoc networks. *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Ad Hoc Networking and Computing,MOBICOM '99*, pages 186–194, August 1999.

[6] IEEE 802 LAN/MAN Standards Committee. Wireless LAN medium access control MAC and physical layer (PHY) specifications. IEEE Standard 802.11, 1999.

[7] IEEE 802 LAN/MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999 edition edition, 1999.

[8] S. R. Das, R. Castenada, J. Yan, and R. Sengupta. Comparative performance evaluation of routing protocols for mobile, ad hoc networks. *Proceedings of Seventh International Conference on Computer Communication (IC3N)*, pages 153–161, October 1998.

[9] S. R. Das, C. E. Perkins, and E. M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. *Proceedings of IEEE Conference on Computer Communications (INFOCOM '00)*, pages 3–12, March 2000.

[10] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE INFOCOM*, 2002.

[11] Z. J. Haas and M. R. Pearlman. Determining the optimal path configuration for the zone routing protocol. *Journal of Selected Areas in Communication*, 17:1395–1414, 1999.

[12] J.-C. Hu and D. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. *Proceedings of IEEE/ACM MOBICOM '00*, pages 231–242, August 2000.

[13] Y-C Hu and D. B. Johnson. Implicit source routes for on demand ad hoc network routing. *Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc '01)*, 2001.

[14] Y.-C. Hu and D. B. Johnson. Ensuring cache freshness in on-demand ad hoc network routing protocols. *Proceedings of Principles of Mobile Computing 2002 (POMC '02)*, October 2002.

[15] M. Jiang, J. Li, and Y. C. Tay. *Cluster Based Routing Protocol*, draft-ietf-manet-cbrp-spec-01.txt edition, August 1999.

[16] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. G. Jetcheva. *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)*, draft-ietf-manet-dsr-07.txt edition, August 2002.

[17] Y.-B. Ko and N. H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Ad Hoc Networking and Computing, Mobicom '98*, August 1998.

[18] S.-J. Lee and M. Gerla. Backup routing in ad hoc networks. *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2000.

[19] S.-J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. *Proceedings of the International Conference on Communication (ICC)*, 2001.

[20] B. Liang and Z. J. Haas. Optimizing route cache lifetime in ad hoc networks. *Proceedings of IEEE INFOCOM '03*, 2003.

[21] W. Lou and Y. Fang. Predictive caching starategy for on-demand routing protocols in wireless ad hoc networks. *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC '01)*, pages 671–679, November 2001.

[22] D. A. Maltz, J. Broch, J. Jetcheva, and D. B. Johnson. The effects of on-demand behavior in routing protocols for multihop wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, pages 1439–1453, August 1999.

[23] M. K. Marina and S. R. Das. On demand multipath distance vector routing in ad hoc networks. *Proceedings of IEEE International Conference of Network Protocols (ICNP)*, 2001.

[24] M. K. Marina and S. R. Das. Performance of route caching strategies in dynamic source routing. *Proceedings of the 2nd Wireless Networking and Mobile Computing (WNMC)*, April 2001.

[25] A. Nasipuri, Robert Castaneda, and S. R. Das. Performance of multipath routing for on-demand protocols in mobile ad hoc networks. *Mobile Networks and Applications*, 2000.

[26] A. Nasipuri and S. R. Das. On demand multipath routing for mobile ad hoc networks. *Proceedings of 8th Annual IEEE International Conference on Computer Communications and Networks (ICCCN)*, 1999.

[27] V. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. *Proceedings of IEEE INFOCOM '97*, April 1997.

[28] C. Perkins and P. Bhagvat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *Proceedings of ACM SIGCOMM '94*, pages 234–244, August 1997.

[29] C. E .Perkins, E. M. Royer, and S. R. Das. *Ad Hoc On-Demand Distance Vector (AODV) Routing*, draft-ietf-manet-aodv-11.txt edition, June 2002.

[30] L. Qin and T. Kunz. Proactive route maintenance in dsr. *ACM SigMobile Mobile Computing and Communications Review*, July 2002.

[31] J. Raju and J. J. Garcia-Luna-Aceves. A new approach to on-demand loop-free multipath routing. *Proceedings of the 8th Internation Conference on Computer Communications and Networks (ICCCN)*, pages 522–527, October 1999.

[32] V. Ramasubramanian, Z. J. Haas, and E. G. Sirer. SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks. *ACM Symposium on Mobile Ad Hoc Networking andi Computing (MobiHOC)*, 2003.

[33] E. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6:46–55, April 1999.

[34] M. Spohn and J. J. Garcia-Luna-Aceves. Neighborhood aware source routing. *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC '01)*, 2001.

[35] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, New Jersey, 3rd edition, 1996.

[36] C.-K. Toh. Associativity-based routing for ad hoc mobile networks. *Wireless Personal Communications Journal, Special Issue on Mobile Networking & Computing Systems*, 4:103–109, 1997.

[37] A. Valera, W. K. G. Seah, and SV Rao. Cooperative packet caching and shortest multipath routing in mobile ad hoc networks. *Proceedings of IEEE INFOCOM '03*, 2003.

[38] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. *IEEE INFOCOM*, 2003.