Design and Evaluation of AMPER, a Probabilistic Routing Protocol for Mobile Ad Hoc Networks

by

Abdallah W. Jabbour

B.E., Computer and Communications Engineering
American University of Beirut, Lebanon (2002)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2004

Author .....
Department of Civil and Environmental Engineering
May 7, 2004

Certified by .....
John Wroclawski
Research Scientist, Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Certified by .....
George Kocur
Senior Lecturer, Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Heidi Nepf
Chairman, Departmental Committee on Graduate Students

Design and Evaluation of AMPER, a Probabilistic Routing Protocol for Mobile Ad Hoc Networks

by

Abdallah W. Jabbour

B.E., Computer and Communications Engineering

American University of Beirut, Lebanon (2002)

Submitted to the Department of Civil and Environmental Engineering
on May 7, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

An ad hoc network is a group of mobile nodes that autonomously establish connectivity via multi-hop wireless links, without relying on any pre-configured network infrastructure. Traditional ad hoc routing protocols use a large number of routing packets to adapt to network changes, thereby reducing the amount of bandwidth left to carry data. Moreover, they route data packets along a single path from source to destination, which introduces considerable latency for recovery from a link failure along this path. Finally, they often use the minimum hop count as a basis for routing, which does not always guarantee a high throughput. This thesis presents AMPER (**A**d hoc, **M**odular, **P**robabilistic, **E**nhanced **R**outing), an ad hoc routing protocol that minimizes the routing packet overhead, allows the use of alternate paths in the event of a link outage, and employs – without loss of generality – the expected number of transmissions to make forwarding decisions. Following the design of AMPER, *ns*-2 is used to simulate it, evaluate it and compare it to other ad hoc routing protocols.

Thesis Supervisor: John Wroclawski
Title: Research Scientist, Computer Science and Artificial Intelligence Laboratory

Thesis Supervisor: George Kocur
Title: Senior Lecturer, Civil and Environmental Engineering

Acknowledgments

First and foremost, I turn with love, respect and obligation to my parents, who were and are still the reason behind my success. Despite the distance separating us, their prayers and continuous support helped me surmount all the difficulties I faced while a student at MIT.

I would like to thank my supervisor, Mr. John Wroclawski, for taking me on as a student, even though he knew little about me. His comments were always insightful and helped me channel my efforts in the correct direction. I am grateful to my thesis reader, Dr. George Kocur, whom I had the pleasure of interacting with as a student, teaching assistant, and most of all, as a friend. I must not forget Professor Hari Balakrishnan, who made discerning remarks that helped improve the quality of this work.

I greatly benefited from the academic guidance Ms. Dorothy Curtis offered me; I could not have become acquainted with the CSAIL community without her precious help. I am also thankful to James Psota and Alexey Radul, who worked with me on a course paper that constituted the groundwork of this thesis.

Finally, I would like to acknowledge all the other persons I interacted with during the past two years: friends, colleagues, staff members and professors. Their presence made my stay at MIT more fruitful and more enjoyable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In situations where impromptu communication facilities are required, wireless mobile users may be able to communicate through the formation of an ad hoc network. Such a network (illustrated in Figure 1-1) is characterized by a self-organizing and decentralized communication, whereby every node acts as a router by forwarding packets to other nodes using multi-hop paths. Ad hoc networks are commonly encountered in the following contexts:

- Interactive lectures
- Group conferences
- Wireless offices
- Moving vehicles (for dissemination of traffic information)
- Battlefield environments
- Disaster relief situations

Some key factors should be taken into account when building a mobile ad hoc network: bandwidth, power control, network configuration, transmission quality, device

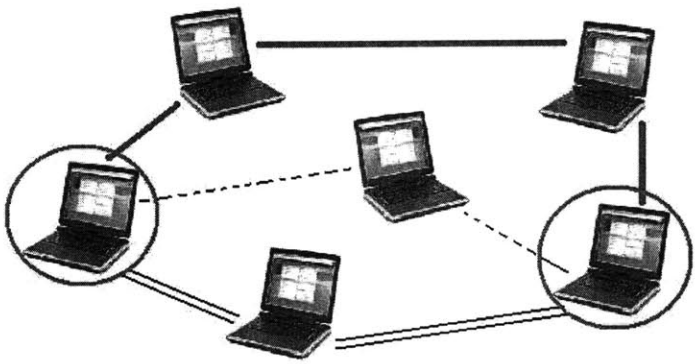

Figure 1-1: An Ad Hoc Network with Six Nodes

discovery, topology maintenance, ad hoc addressing and routing. This thesis solely deals with mobile ad hoc networks from a routing perspective.

1.2 Motivation

Routing is the process by which nodes in a network build a path in order to send information from a source to a destination. Routing in mobile ad hoc networks is not a simple task, due to the dynamic topologies that result from mobility, nodal failures, and unpredictable environmental conditions. In recent years, a variety of routing protocols for mobile ad hoc networks have been developed [9, 11, 12]. One of their drawbacks is that they use a large number of routing packets to adapt to network changes, thereby reducing the amount of bandwidth left to carry data. For example, by periodically updating routing tables, proactive protocols such as DSDV [11] induce a continuous traffic that increases with the number of nodes and the frequency of changes in the topology. On the other hand, reactive protocols such as DSR [9] and AODV [12] often employ some kind of flooding to acquire routes, which is unsuitable for large networks. Moreover, all of these protocols route data packets along a single path from source to destination, which introduces considerable latency for recovery from a link failure along this path. Finally, they use the minimum hop count as a routing metric, which does not always deliver the highest possible throughput [5]. Clearly, the above protocols suffer from inherent shortcomings.

1.3 Proposed Approach

This thesis presents AMPER (**A**d hoc, **M**odular, **P**robabilistic, **E**nhanced **R**outing), an ad hoc routing protocol that minimizes the routing packet overhead, allows the use of alternate paths in the event of a link outage, and employs – without loss of generality – the expected number of transmissions [5] to make forwarding decisions. Our approach is inspired by previous work in the area of probabilistic routing [1, 3, 8, 10, 13, 15, 16], but presents many differences by relaxing the assumption of symmetric traffic, cutting down the number of control packets used, allowing the random routing of data packets, and making the probability entries a function of the loss ratios instead of the hop counts. Following the design of AMPER, we use *ns*-2 to simulate it, evaluate it and compare it to other ad hoc routing protocols.

1.4 Thesis Organization

The thesis proceeds in Chapter 2 with a description of related work. Chapter 3 gives a thorough overview of AMPER, and Chapter 4 describes the environment we use to simulate it. In Chapter 5, we evaluate AMPER by comparing it to three widespread ad hoc routing protocols, namely DSDV [11], DSR [9] and AODV [12]. In Chapter 6, we discuss AMPER's strengths and weaknesses and propose some improvements to overcome its shortcomings. Chapter 7 concludes this thesis.

Chapter 2

Related Work

2.1 Overview

From a routing table structure's perspective, ad hoc routing protocols can be grouped into two categories:

- Non-probabilistic (deterministic) routing protocols, in which the routing table at a node **n** contains <destination, next hop> pairs. The next hop is the neighbor towards which **n** will route its data packets to the corresponding destination.

- Probabilistic routing protocols, in which the routing table at a node **n** contains <destination, next hop, probability value> triplets. In this context, one can distinguish two main routing schemes:

  - Random routing, in which any of **n**'s neighbors is entitled to receive **n**'s data packets, according to the probability value corresponding to that neighbor.

  - Non-random routing, in which **n** exclusively routes its data packets to the neighbor with the highest probability value in the routing table.

We begin this chapter by giving an overview of three deterministic routing protocols, namely DSDV [11], DSR [9] and AODV [12]. We then describe some of the

probabilistic routing protocols we found in the literature. Finally, we motivate the need for a new protocol by exposing the drawbacks of the preceding approaches.

2.2 Destination-Sequenced Distance Vector (DSDV)

DSDV [11] is a loop-free, hop-by-hop distance vector routing protocol. Every DSDV node maintains a routing table that lists all reachable destinations, together with the metric and the next hop to each. Moreover, each routing table entry is tagged with a sequence number originated by the destination station. To maintain the consistency of routing tables in a dynamically varying topology, each node transmits updates either periodically, or right after significant new information becomes available. It does so by broadcasting its own routing table entries to each of its current neighbors. Each node in the network also advertises a monotonically increasing even sequence number for itself, and routes with greater sequence numbers are always preferred for making forwarding decisions. If two routes have equal sequence numbers, the one with the smallest number of hops is used.

When a link to a next hop breaks, any route through that next hop is immediately assigned an infinite metric and an updated sequence number. To describe a broken link, any node other than the destination generates a sequence number that is greater than the last sequence number received from that destination. This newly generated sequence number and an infinite metric are disseminated over the network. To avoid nodes and their neighbors generate conflicting sequence numbers when the network topology changes, the originating nodes produce even sequence numbers for themselves, and neighbors generate odd sequence numbers for the nodes responding to the link changes.

2.3 Dynamic Source Routing (DSR)

Instead of hop-by-hop routing, DSR [9] uses source routing, whereby each forwarded packet carries in its header the complete list of nodes through which it has passed. The

DSR protocol consists of two mechanisms: Route Discovery and Route Maintenance. In Route Discovery, a node **s** wishing to send a packet to some destination **d** locally broadcasts a ROUTE REQUEST packet to its neighbors. The ROUTE REQUEST identifies the destination node to which a route is needed, and also includes a unique identifier chosen by **s**. If a node receives a ROUTE REQUEST for which it is not the target, it adds its own address to the list of nodes in the ROUTE REQUEST packet and locally rebroadcasts it. When the packet reaches its target, the destination node returns to the sender a copy of the entire sequence of hops in a unicast ROUTE REPLY packet, which may be routed along any path. To reduce the overhead of Route Discovery, each node aggressively caches source routes it has learned or overheard.

Route Maintenance is the mechanism by which a sender **s** ensures that a route to a destination **d** remains valid despite changes in the network topology. The sender knows that a hop within a source route is broken when it receives a ROUTE ERROR packet from a certain node along the path. **s** then removes this broken link from its route cache and attempts to use any other route to **d** in its cache. If it does not find one, it can invoke Route Discovery again to find a new route to the destination.

2.4 Ad Hoc On-Demand Distance Vector (AODV)

AODV [12] is a hybrid loop-free hop-by-hop protocol that borrows the broadcast Route Discovery mechanism from DSR, and the destination sequence numbers from DSDV. When a source **s** needs to communicate with a destination **d** for which it has no routing information in its table, it initiates a Path Discovery process by broadcasting a ROUTE REQUEST (RREQ) packet to its neighbors. The RREQ packet contains six fields: the source and destination addresses and sequence numbers, a broadcast ID and a hop count. Each node that forwards the RREQ creates a reverse route for itself back to **s**, by recording the address of the neighbor from which it received the first copy of the RREQ. A node either satisfies the RREQ by sending a ROUTE REPLY (RREP) back to the source, or rebroadcasts the RREQ to its own neighbors after increasing the hop count. Eventually, an RREQ will arrive at a node **y** that possesses

a fresh route to **d**. This node can reply only when it has a route with a sequence number that is greater than or equal to that contained in the RREQ. The RREP generated by **y** contains the number of hops needed to reach **d**, and the most recent destination sequence number seen by **y**. As the RREP travels back to **s**, each node along the path sets up a forward pointer to the node from which the RREP came and records the latest destination sequence number for the requested destination.

When a certain next hop becomes unreachable, the node upstream of the break propagates an unsolicited RREP with a fresh sequence number and an infinite hop count to all active upstream neighbors; this process continues until the source node is notified. When receiving an unsolicited RREP, the sender must acquire a new route to the destination by carrying out a new Path Discovery.

2.5 Probabilistic Routing Protocols

In probabilistic routing protocols, the routing table at a node **n** contains **n**'s neighbors in its columns, all accessible destinations in its rows, and a probability entry corresponding to every destination-neighbor pair. We give below some examples of probabilistic routing protocols for wired and mobile ad hoc networks.

2.5.1 Ant-Based Control (ABC)

ABC [16] was proposed in 1996 as an approach for load balancing in circuit-switched networks. The protocol relies on the operation of routing packets called ants. Ants are periodically launched with random destinations, and they walk according to the probabilities they encounter in the routing tables for their particular destination. When an ant sent from a node **n** arrives at a node **c** through another node **b**, the entry in the routing table of **c** having **n** as a destination and **b** as a next hop sees its probability increased. This is known as reinforcement learning, since more ants sent along the same path will result in a reinforcement of the backward route. Data packets are routed independently of the ants, according to the highest probability value for each destination (probability values are set to be inversely proportional to

paths lengths).

2.5.2 AntNet

AntNet [3] is another probabilistic routing protocol used in wired networks. It shares some similarity with ABC, but it introduces the concept of forward ants and backward ants. Every node **n** contains a routing table with probabilistic entries, and an array of data structures defining a parametrical statistical model for the network delay conditions as seen by **n**. At regular intervals, a forward ant is launched from each node towards a randomly selected destination, in order to discover a minimum hop path and investigate the traffic conditions of the network. The forward ant saves its path in a stack, as well as the time needed to arrive at each intermediate node. When the destination node is reached, the forward ant generates a backward ant, transfers to it all of its memory, and dies. The backward ant takes the opposite path, but is sent on a high priority queue to quickly propagate the information accumulated by the forward ants. During this backward travel, local models of the network status and the routing table of each visited node are modified as a function of the path followed by the backward ants. As in ABC [16], higher probability values correspond to shorter paths, and data packets are routed deterministically according to the highest probability value for each destination.

2.5.3 The Ant-Colony-Based Routing Algorithm (ARA)

ARA [8] borrows many characteristics from DSR [9] and applies the concept of forward and backward ants to mobile ad hoc networks. Specifically, Route Discovery is achieved by sending forward ants to the destination and backward ants to the source. Once the forward and backward ants have established a route between the source and the destination, the flow of data packets through the network serves to maintain that route. ARA prevents loops by letting the nodes recognize duplicate packets based on the source address and sequence number. In the event of a link failure, the protocol tries to send the packet over an alternate link. If it does not find one, the packet is

returned to the previous node for similar processing. A new route discovery process is launched if the packet is eventually returned to the source.

2.5.4 Other Probabilistic Routing Protocols

We have arbitrarily chosen ARA [8] as an example to illustrate probabilistic routing in mobile ad hoc networks. Other protocols abund in the literature, but we have opted not to cover them in depth because they share many characteristics with ABC [16], AntNet [3] and ARA [8]. In PERA [1], forward ants explore routes and record local traffic statistics in an array at each node. Backward ants use the information contained in the forward ants to change the probability distribution at each node. In ANSI [13], proactive ants are used for local link discovery and maintenance, whereas reactive ants are deployed on demand for non-local route discovery. Adaptive-SDR [10] performs node clustering during the initialization stage, and distinguishes between colony ants and local ants. Each node has a colony routing table and a local routing table. Colony ants find routes between clusters, and local ants are confined within a colony. Termite [15] borrows some features from ABC [16] and AODV [12], and uses four types of control packets for route discovery and maintenance.

2.6 Drawbacks and Limitations

2.6.1 Overhead

Any ad hoc routing protocol involves four major types of overhead:

- An algorithmic overhead stemming from the protocol's basic nature.
- An overhead caused by inefficient updates – i.e. sending update packets when nothing significant has changed in the network.
- An overhead caused by the slowness to notice link failures or other network changes – i.e. sending packets over stale routes.

- An implementation overhead due to coding, packet formats, separate routing packets and/or piggybacking.

We will use the following parameters to determine the significance of each type of overhead:

- Bandwidth usage
- Cost of acquiring the wireless channel
- CPU and memory usage

The algorithmic overhead is mainly due to the amount of state used by the protocol, and to the number of messages sent and the manner they are routed. It goes without saying that such an overhead affects bandwidth, channel access cost, and CPU and memory usage altogether. The amount of state maintained in routing tables is $O(N)$ for deterministic routing protocols like DSDV, DSR and AODV, and $O(N \times M)$ for probabilistic routing protocols, where N is the number of nodes in the network and M is the number of neighbors around a node ($M \leq N$). Modern laptops and handheld devices should not suffer from this problem, especially when the network size is relatively small[1]. The algorithm that dictates the sending of messages produces a substantial overhead, but the latter cannot be easily avoided since it relates to the protocol core.

The second type of overhead occurs when a node keeps sending update packets even if little[2] or no change has occured in the network. Although this improves[3] the nodes' accurate knowledge of the current network state, it unneccesarily consumes bandwidth and requires additional cost to access the wireless medium. The solution to this problem involves adapting the update frequency to changing circumstances, in order to balance accuracy and efficiency. We believe that the existing protocols surveyed above deal well with this problem.

[1]Our aim is to design a protocol that behaves well in small networks.

[2]This introduces a tradeoff that is outside the scope of this thesis; there is an interaction between the frequency of changes in the topology and the periodicity of the updates.

[3]or at least does not degrade.

The third type of overhead arises when nodes do not quickly react to network changes, and end up sending packets over stale routes. This again wastes bandwidth and involves additional cost to access the medium. However, this kind of overhead tends to be negligible, mainly because is it only incurred during a short period of time[4].

The implementation overhead can be significant, depending on the choice of routing strategy. If separate control packets are used, they will certainly consume bandwidth, but their most pronounced effect will be to increase the cost needed to gain access to the wireless channel. In fact, the cost spent to acquire the medium in order to transmit a packet is significantly more expensive in terms of power and network utilization than the incremental cost of adding a few bytes to an existing packet[5] [2]. Moreover, protocols that send a large number of routing packets can also increase the probability of packet collisions and may delay data packets in network interface transmission queues. In DSDV [11], the control message overhead grows as $O(N^2)$, where N is the number of nodes in the network. In AODV [12], the broadcast route discovery mechanism prevents the protocol from scaling to large networks. The flooding mechanism used in DSR [9] has the same shortcoming, and the extra overhead caused by the presence of a source route is incurred not only when the packet is originated,but also each time it is forwarded to a next hop. The probabilistic routing protocols surveyed above use a large amount of routing packets, which increases with the number of nodes and the frequency of changes in the topology. If piggybacking onto data packets is used, the bandwidth consumed is practically the same, but the cost of acquiring the wireless channel drops drastically. For all these reasons, it becomes clear that the routing packet overhead is the main source of "avoidable" overhead, which makes us believe that diminishing it will result in performance improvement.

[4]The implicit assumption is that the recovery mechanism works reasonably well.

[5]This is due, in part, to the substantial overhead caused by RTS and CTS packets if they are used.

2.6.2 Slow Recovery from Link Failures

The second drawback resides in the way data packets are forwarded. In DSDV [11], DSR [9] and AODV [12], every forwarding node selects one neighbor only towards a given destination, which implies that data packets are routed along a single path from source to destination. This introduces considerable latency for recovery from a link failure along this path, since the intermediate nodes discovering the failure will then have to find a new route to the destination or notify the source to issue a new routing request. Probabilistic routing protocols do not suffer from this problem, since they can relay packets along alternate routes. However, the probabilistic protocols surveyed in Section 2.5 all launch control packets randomly, but deterministically route data packets according to the highest probability entries for each destination, the reason probably being the avoidance of packet reordering. We believe that random routing deserves further investigation, since it is not examined in any related work we have surveyed.

2.6.3 Unrealistic Probability Update Rules

While the supposition of symmetric links is plausible in modern networks, the probabilistic routing protocols mentioned in the previous section go beyond it by assuming symmetric traffic. When a node **d** receives a packet from a node **s** through a node **n**, **d** will increase the probability of choosing **n** as its next hop towards **s**. This is not realistic because the receipt of a packet by **d** should instead increase the probability of **s** choosing **n** as a next hop to **d**.

2.6.4 Throughput with Minimum Hop Count

Finally, all of the protocols described above use the minimum hop count as a routing metric. De Couto et al. [6] show, using experimental evidence, that there are usually multiple minimum-hop-count paths, many of which have poor throughput. As a result, minimum-hop-count routing often picks marginal links and chooses routes that have significantly less capacity than the best paths that exist in the network.

The shortcomings exposed above lead us to the following question: is it possible to invent an ad hoc routing protocol that minimizes the routing packet overhead, swiftly recovers from link failures, uses a routing metric that reflects network state better than minimum hop count, and – if applicable – realistically updates probability values? The answer turns out to be affirmative.

Chapter 3

Protocol Description

This chapter provides a thorough description of AMPER. Following a brief overview, we state our assumptions and then delve into the actual details of the protocol.

3.1 Overview

Let us recapitulate the goals we set out to achieve with our protocol. First, we would like to minimize the routing packet overhead, keeping valuable bandwidth resources available for data packets. Secondly, we ought to employ a routing metric that reflects network state better than minimum-hop-count. Finally, we intend AMPER to be able to quickly and efficiently shift to alternate routes upon link failure.

To minimize explicit control packet transmissions, AMPER appends internode communication information onto data packets. Whenever a node sends its own or forwards another node's data packet, it prepends an AMPER protocol-level header to it. As packets are sent throughout the network, all nodes promiscuously listen to the AMPER headers in order to learn updated information about the conditions of the network. We use the expected number of transmissions (ETX) [5] as an initial choice of routing metric; however, we have designed AMPER in such a way that we can plug in different metrics as the need arises (i.e., AMPER is not dependent on any particular choice of routing metric). Finally, we use probabilistic routing, whereby packets are forwarded along a given link with a certain probability. This will allow AMPER to

better adapt to alternate routes upon link outages: if a link towards a given neighbor breaks, the sending node can immediately choose some other neighbor(s) to route its packets. In theory, it should also have the desirable side effect of allowing nodes along any path to transmit data from time to time, so that their neighbors remain aware of their existence and state. The probabilistic routing mode is also modular and can easily be flipped from random to non-random and vice-versa.

3.2 Assumptions

Although a node may have several physical network interfaces, it selects a single IP address by which it will be known in the ad hoc network. We assume that each node has sufficient CPU capacity to run AMPER, and is willing to forward packets to other nodes in the network.

MAC protocols such as IEEE 802.11 [4] limit unicast data packet transmission to bi-directional links, due to link-layer acknowledgments and to the required bi-directional exchange of RTS and CTS. Therefore, the assumption of bi-directional links is needed.

Finally, every node should be capable of enabling promiscuous receive mode on its wireless interface hardware, which causes it to deliver every received packet to the network driver software without filtering based on link-layer destination address. Promiscuous mode increases the CPU software overhead and may raise the power consumption of the network interface hardware, but is a valuable feature in AMPER for reasons that will become clear as this chapter unfolds.

3.3 Initialization and Node Joins

The correctness of AMPER does not depend on any explicit activity on initialization or node joins. A node joining the ad hoc network can simply turn on promiscuous receive mode and start running AMPER . If it needs to send packets and has not overheard any information that it finds useful for doing so, it can send an explicit

F	frequency of major state update
f	frequency of minor state update
t_d	destination reachability timeout
t_r	request timeout
t_l	link loss ratio sampling window length
t_n	silent neighbor timeout
m_{ret}	maximum number of request retries

Table 3.1: AMPER parameters

control packet to ask. We have experimented with HELLO packets [12] as a possible optimization, but they have resulted in an overall performance degradation.

3.4 AMPER Parameters

The significant parameters that we varied in our implementation of AMPER are given in Table 3.1. The exact meanings of these parameters will become clear throughout the protocol description. The numerical values we adopted for these parameters will be presented in Chapter 5.

3.5 Node State

Every node **n** maintains the following pieces of information:

- The current time (at node **n** — there is no need for the times that all nodes maintain to be synchronized).
- Timers to determine when **n** will make its next periodic state update. Specifically, these timers go off at the frequencies given by F and f.
- The set of neighbors of **n**. These are the nodes from which **n** has heard something within the silent neighbor timeout t_n, and/or which have heard something from **n** within that time.[1]

[1]The exact definition to use is clear from the context; **n** learns which nodes have heard from it within the timeout because they list it as such in the headers of the packets they transmit — see Section 3.7.

- Loss ratio state:
  - The times at which **n** sent packets (to anyone) within the link loss ratio timeout t_l.
  - For each neighbor **m** of **n**, the times at which **n** heard **m** send a packet (to anyone) within that same timeout t_l.
  - For each neighbor **m** of **n**, the link loss ratio of packets sent from **n** to **m**, as sampled over a window of length t_l.
  - For each neighbor **m** of **n**, the link loss ratio of packets sent from **m** to **n**, as sampled over a window of length t_l[2].
- Routing state:
  - A list of reachable destinations.
  - A list of unreachable destinations, with timestamps of when it was decided that each of them was unreachable.
  - For each reachable destination **d**, the metric value $M_{\mathbf{n}}(\mathbf{d})$ for sending packets from **n** to **d**.
  - For each reachable destination **d** and each neighbor **m** of **n**, the metric value $M_{\mathbf{m}}(\mathbf{d})$ for sending packets from **m** to **d**.
  - For each reachable destination **d** and each neighbor **m** of **n**, the probability $p_{\mathbf{n}}(\mathbf{d}, \mathbf{m})$ of sending a packet destined for **d** to neighbor **m**. Note that $\sum_{\mathbf{m}} p_{\mathbf{n}}(\mathbf{d}, \mathbf{m}) = 1$ for fixed **n** and **d**.
- Node cooperation state:
  - What destinations **n** would like to learn about[3] in order to send data to them.
  - What destinations **n** has heard its neighbors ask about (within the request timeout t_r).

[2]The loss ratio calculations are valid even if the nodes are not transmitting packets continuously.
[3]i.e., discover its neighbors' metric values for

- Which of those destinations **n** itself needs to learn about in order to answer its neighbors.
- What requests for information **n** made within the request timeout t_r, when the last time **n** made each of those requests was, and how many times **n** made each of those requests.
- How far to propagate a request for information; an expanding ring search is implemented on such requests (see Paragraph 3.9.2 for more details).

This state is dynamic, which means that its size can grow and shrink based on the node's knowledge of the current topology. Except for the clock, which we assume is updated continuously and correctly,[4] this state changes only when a node hears a transmission, transmits a packet, or makes periodic updates (see Section 3.8 for details). One can easily see that the maximum (theoretical) amount of state that any node may have to maintain is $N^2 + Nx$, where N is the number of nodes in the ad hoc network and x is the number of times any node has recently sent a packet.

3.6 Routing

When a node **n** wants to send (or forward) a packet to a destination **d**, it checks its state to see whether **d** is reachable.[5] If **d** is determined to be unreachable within the destination reachability timeout t_d, **n** drops the packet. If **d** is neither known to be reachable nor known to be unreachable, **n** puts the packet in a wait queue, records the fact that it wants to find out how to get to **d**, and sends packets to other destinations.[6] If **d** is reachable, **n** looks up the probability distribution corresponding to **d** and routes the packet according to that probability distribution.

In this thesis, we investigate random probabilistic routing, in which every neighbor is entitled to receive the packet with the probability given in the distribution, and

[4]The "periodic update" functions get called when the time comes for them.

[5]Unless, of course, **n** is **d**, in which case it hands the packet up the network stack.

[6]If **n** finds itself with queued packets but no packets on which to piggyback the requests, **n** can send an explicit control packet to make its requests. In particular, this will likely happen on initialization.

	Next Hop Probabilities		
Dest	m_1	m_2	m_3
d_1	0.20	0.30	0.50
d_2	0.90	0.05	0.05
d_3	0.80	0.15	0.05
d_4	0.10	0.75	0.15

Table 3.2: Routing Table for a Node with Three Neighbors and Four Accessible Destinations.

non-random probabilistic routing, in which the packet is always sent to the neighbor with the highest probability value in the routing table. Consider, for example, a node *node* with the routing table shown in Table 3.2. In random mode, it routes a packet for destination d_2 to neighbor m_1 with probability 0.9, and to each of neighbors m_2 and m_3 with probability 0.05. If non-random mode is used, node *node* will route the packets destined to d_2 to neighbor m_2 only.

As mentioned in Section 3.1, the modular structure of AMPER allows us to switch between the two modes if the need arises[7].

3.7 Internode Communication

The motivating philosophy behind AMPER is that control packets have a deleterious effect on network performance. Therefore, as much internode communication as possible is piggybacked on data packets. Specifically, whenever a node **n** sends (or forwards) a packet to a neighbor **m**, **n** prepends an AMPER packet header to said packet[8], which all the neighbors of **n** promiscuously overhear and use to update their state. The recipient **m** of the data packet removes this header and appends its own before forwarding the packet to the next node[9]. The packet header contains:

- Loss ratio information:

[7]This is outside the scope of this thesis, but one could also imagine a "semi-random" mode, in which we would aim to be as random as possible while handling the constraints of TCP.

[8]It is easier to prepend a header than append it in *ns*-2; besides, a prepended header may be better for snooping.

[9]For simplification, we assume that the MAC layer's maximum segment size is bigger than its IP counterpart.

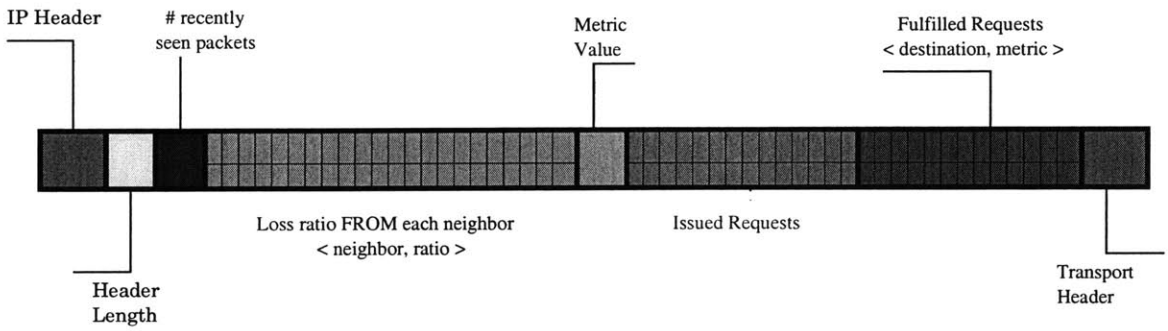

Figure 3-1: Conceptual AMPER Header

  - The number of packets **n** has sent (to anyone) within the link loss ratio window time t_l.
  - For each neighbor **m** of **n**, the sampled link loss ratio for the packets sent from **m** to **n**.

- Routing information:
  - The metric value $M_{\mathbf{n}}(\mathbf{d})$ of **n** for the destination of this packet.

- Node cooperation information:
  - A list of destinations that **n** would like to get information about, but has not asked its neighbors about within the request timeout t_r. This includes destinations that the neighbors of **n** have asked about, but whose requests **n** cannot honor for the lack of said information.
  - If any neighbor of **n** has requested information on any destination $\mathbf{d}' \neq \mathbf{d}$, and **n** has such information but has not yet honored that request, the metric value $M_{\mathbf{n}}(\mathbf{d}')$ of **n** for each such destination $\mathbf{d}'$.

A conceptual view of the AMPER header can be seen in Figure 3-1. The size of this header is variable, so it also includes an appropriate length field. If the wireless links can be assumed to be symmetric, then it may not be necessary to send the loss ratios, which would diminish the header size significantly. We do not experiment with this assumption as it would constitute a loss of generality.

3.8 Node State Update

A node **n** updates its state (described in Section 3.5) under three circumstances: whenever **n** hears a packet transmitted by one of its neighbors, when **n** transmits a packet itself, or as **n** makes periodic updates. The process of such updates is described in detail below.

3.8.1 Updates on Sending

Whenever a node **n** contemplates sending a packet, it updates its state as follows:

- If **n** believes the destination **d** of that packet is unreachable (i.e., **n** has recently found it to be so, within the destination reachability timeout t_d), drop the packet and do not change state.
- If **n** does not know how to reach **d** and is not certain **d** is unreachable, add **d** to the list of destinations **n** would like to learn about, and delay sending this packet (send others instead).
- If **n** attempts to send the packet but the MAC layer is unable to establish contact with the neighbor for which it is intended, remove that neighbor from the neighbor list (with all attendant state clearing and reshuffling of probability values) and try to send the packet again to a different neighbor.

Otherwise:

- Add "now" to the set of times when **n** sent a packet.
- If this transmission makes any requests for information, record them.
- Remove any requests for information honored by this transmission from the list of not yet honored requests for information[10].
- If the neighbors of **n** have made any requests for information that **n** is unable to honor due to the lack of said information, record them.

[10]Handling recovery from lost packets is outside the scope of this thesis – the reliability and recovery mechanisms are not incorporated in AMPER.

3.8.2 Updates on Receiving

Whenever a node **n** hears a packet P transmitted by a node **m** (not necessarily destined for **n**), **n** updates its state as follows:

- If **m** is not on **n**'s neighbor list, put it there and initialize the per-neighbor state to appropriate zero and null values.
- Loss ratio state:
  - Add "now" to the list of times **n** has heard **m** send anything.
  - Set the link loss ratio for packets sent from **n** to **m** to the value reported in the header of P. This is how **n** learns whether **m** has heard anything from it within the silent neighbor timeout t_n.
  - Recalculate the link loss ratio for packets sent from **m** to **n** based on the number of packets **m** has sent (from the header of P) and the number of packets **n** has heard (from **n**'s state).
- Routing state:
  - If the header of P mentions previously unknown destinations, add appropriate null values for them and continue.
  - For each destination **d** for which the header of P has data, update $M_{\mathbf{m}}(\mathbf{d})$ to the value reported in the header of P.
  - For each such destination **d**, update the probability distribution $p_{\mathbf{n}}(\mathbf{d}, *)$ in accordance with the probability distribution update rule.
  - For each such destination **d**, recalculate $M_{\mathbf{n}}(\mathbf{d})$.[11]
  - For each such destination **d**, if **n** wanted to learn about **d** and now has enough state to reasonably send packets to **d**, remove **d** from the list of destinations that **n** would like to get information about.

[11] To do this, it is necessary for M to be local enough to be computed from the metric values of the neighbors, the link loss ratios to them, and the probability distribution for sending packets to them.

- Node cooperation state:
  - If the header of P contains a request for information, record it.
  - If the ultimate destination of P is $\mathbf{n}$ itself, $\mathbf{n}$ treats this as an implicit request for information about itself.[12]

3.8.3 Periodic State Updates

Every node $\mathbf{n}$ periodically updates its state as follows:

- Update the timers that govern $\mathbf{n}$'s next periodic state update.
- If $\mathbf{n}$ has heard nothing from some neighbor $\mathbf{m}$ within the silent neighbor timeout t_n, remove $\mathbf{m}$ from the neighbor list, and delete all its per-neighbor state. Recompute the probability distributions and metric values accordingly.
- Loss ratio state:
  - If any packets $\mathbf{n}$ has sent or heard are not within the link loss ratio timeout t_l, remove their times from the list of recently sent or heard packet times.
- Routing state:
  - For each destination $\mathbf{d}$, update the probability distribution $p_{\mathbf{n}}(\mathbf{d}, *)$ according to the probability distribution update rule.
  - For each destination $\mathbf{d}$ whose probability distribution has changed, recompute the metric value $M_{\mathbf{n}}(\mathbf{d})$.
  - If the declaration of unreachability on some destination has expired, remove that destination from the list of unreachable destinations.
- Node cooperation state:

[12]We have discovered this to be a very good idea, to ensure that the destinations of traffic talk (whether they need to or not) and the network does not lose track of them as they move about.

- If **n** has made an information request that was due to need on **n**'s part [13], has not been answered, but has had its request timeout expire, schedule it to be made again.
- If **n** has made a request for information more than m_{ret} times without success, give up and declare that destination unreachable.
- If a request for information that has been made of **n** has had its request timeout t_r run out, but **n** has not yet managed to honor it, forget about it.
- If **n** has information it would like to acquire or disseminate, but no packets to send on which to piggyback the request or answer, send an explicit control packet containing all the usual header data (including the request or answer in question).

3.9 Remarks on Requests and State Updates

3.9.1 Frequency of Updates

We have found that it is very much worthwhile to perform the node cooperation updates quite often (with frequency f), so that destination requests are propagated reasonably swiftly even when there is little or no traffic to piggyback them on; this proved especially valuable during initialization. On the other hand, we allow the other periodic state updates to occur less frequently (with frequency $F>f$).

3.9.2 Expanding Ring Search

We implement expanding ring search on information requests, which is reminiscent of the DSR Route Discovery mechanism [9]. Whenever a node makes a request, it attaches a "ring size" to it. Every time the node retries the request, it increments the ring size. Every time the request is forwarded, the ring size is decremented. When the ring size reaches zero, the request is not dropped, but it is only forwarded piggybacked

[13] rather than that of one of **n**'s neighbors

on data packets, so extra explicit control packets are not generated for it. However, a control packet is generated if a node needs to honor a request whose ring size has reached zero (but has no data packets for this purpose).

3.9.3 Explicit Control Packets

We believe that the explicit control packets generated for the purpose of propagating and answering information requests should not be terribly burdensome on the network, at least in the random probabilistic routing mode. The reason is that if there is a great deal of traffic around a node, then that node will be called upon at least occasionally to forward some of it, and it can then propagate the requests and answers it cares about. If a node ends up generating a control packet, then that control packet probably doesn't interfere with much data traffic. We have not been able to reliably test this hypothesis.

3.10 Choice of Routing Metric and Probability Rules

In this section, we outline the metric we use for routing, and explain the mapping between the metric values and the probability values in each node's routing table.

3.10.1 Routing Metric

As mentioned in Paragraph 3.8.2, AMPER is modular in the sense it can accomodate a wide variety of metrics. The only condition is that the chosen metric be local enough to be computed from the metric values of a given node's neighbors, the link loss ratios to them, and the probability distribution for sending packets to them.

First off, we avoid using the minimum hop count for the reasons explained in Section 2.6. Choosing the end-to-end delay is possible, but this metric changes with network load, which makes it difficult for good paths to be maintained. Using the product of per-link delivery ratios as a throughput measure fails to account for interference between hops: this metric will pick a lossless two-hop route instead of a

one-hop route with a 1% loss ratio, although the latter has almost twice the throughput. The same inconvenience arises if we select the useful throughput of a path's bottleneck as a metric [5].

In this paper, we choose to experiment with ETX [5], which finds paths with the fewest expected number of transmissions (including retransmissions) required to deliver a packet to its destination. It is clear that the ETX from a source to a destination is the sum of ETX values between the intermediate nodes on the path. ETX accounts for asymmetric link loss ratios, solves the problem of interference between successive hops of multi-hop paths, and is independent of the network load. Although ETX does not perform well with burst losses, it remains a valuable metric to use in AMPER. In the near future, we plan to investigate other metrics that satisfy the "locality" condition, and to draw a comparison between them.

Formally, the ETX from a node $\mathbf{n}$ to its neighbor $\mathbf{m}$ is defined in terms of the forward ($d_{\mathbf{n},\mathbf{m}}$) and reverse ($d_{\mathbf{m},\mathbf{n}}$) delivery ratios by:

$$\mathrm{ETX}_{\mathbf{n}}(\mathbf{m}) = \frac{1}{d_{\mathbf{n},\mathbf{m}} \times d_{\mathbf{m},\mathbf{n}}} \tag{3.1}$$

The ETX value along a path is the sum of the ETX values on the individual links in that path. Under random probabilistic routing, the ETX value from some node $\mathbf{n}$ to a destination $\mathbf{d}$ is given as follows (the sum ranges over the neighbors of $\mathbf{n}$):

$$\mathrm{ETX}_{\mathbf{n}}(\mathbf{d}) = \sum_{\mathbf{m}} p_{\mathbf{n}}(\mathbf{d},\mathbf{m})\,(\mathrm{ETX}_{\mathbf{n}}(\mathbf{m}) + \mathrm{ETX}_{\mathbf{m}}(\mathbf{d})) \tag{3.2}$$

Unlike [5], AMPER need not send dedicated link probe packets to calculate the forward and reverse loss ratios between nodes and their neighbors. As explained in Chapter 3, both can be deduced by using the node state and packet header information.

3.10.2 Metric-to-Probability Mapping

We now describe how the ETX values of a node $\mathbf{n}$'s neighbors and the link loss ratios from $\mathbf{n}$ to them are used to choose the probability distribution with which $\mathbf{n}$ will route packets to those neighbors. This mapping is also modular, since it can easily be changed without affecting the overall AMPER structure.

We choose a mapping that heavily favors neighbors promising a lower workload to deliver the packet. Specifically, a node $\mathbf{n}$ computes, for each neighbor $\mathbf{m}$, the "directed ETX" for that neighbor — the ETX $\mathbf{n}$ would experience if it routed to that neighbor alone. This is given by:

$$\mathrm{DETX}_{\mathbf{n}}(\mathbf{d}, \mathbf{m}) = \mathrm{ETX}_{\mathbf{n}}(\mathbf{m}) + \mathrm{ETX}_{\mathbf{m}}(\mathbf{d}). \tag{3.3}$$

We then subtract 1 from these values, and set the probability of routing to each of $\mathbf{n}$'s neighbors to be proportional to the inverse square of the result for that neighbor.

Let us illustrate this by an example. Suppose a node $\mathbf{n}$ possesses three neighbors $\mathbf{m}_1$, $\mathbf{m}_2$, and $\mathbf{m}_3$ having the delivery loss ratios and ETX values to the destination of interest given below:

$$d_{\mathbf{n},\mathbf{m}_1} = 0.9, \quad d_{\mathbf{n},\mathbf{m}_2} = 0.8, \quad d_{\mathbf{n},\mathbf{m}_3} = 0.95,$$

$$d_{\mathbf{m}_1,\mathbf{n}} = 0.3, \quad d_{\mathbf{m}_2,\mathbf{n}} = 0.9, \quad d_{\mathbf{m}_3,\mathbf{n}} = 0.95.$$

$$\mathrm{ETX}_{\mathbf{m}_1}(\mathbf{d}) = 1.1, \quad \mathrm{ETX}_{\mathbf{m}_2}(\mathbf{d}) = 1.3, \quad \mathrm{ETX}_{\mathbf{m}_3}(\mathbf{d}) = 1.2,$$

Using Equations 3.1 and 3.3 produces the following directed ETX values:

$$\begin{aligned} \mathrm{DETX}_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_1) &\approx 4.804, \\ \mathrm{DETX}_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_2) &\approx 2.689, \\ \mathrm{DETX}_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_3) &\approx 2.308. \end{aligned}$$

Subtracting 1 and taking the inverse squares, the probability values turn out to be

proportional to about

$$0.069, \quad 0.351, \quad 0.584,$$

for a net probability distribution of

$$\begin{aligned} p_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_1) &\approx 0.068, \\ p_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_2) &\approx 0.350, \\ p_{\mathbf{n}}(\mathbf{d}, \mathbf{m}_3) &\approx 0.582. \end{aligned}$$

Observe that the chosen mapping is static — it keeps no record of history information (except what is needed to compute link loss ratios), and does not in any way evolve, but simply translates the directed ETX values into probabilities. We chose a static mapping for ease of implementation — had we chosen a dynamic one, it would have been much more difficult to verify that it was being correctly updated. Moreover, we believe that a dynamic mapping would produce a similar trend for the probability distribution, without any exorbitant differences in probability values.

Having thoroughly described AMPER, we are now in a position to simulate it and evaluate it. However, before proceeding to the actual results, we ought to expose the environment in which we carry out our simulations.

Chapter 4

Simulation Environment

To evaluate AMPER, we use the *ns*-2 network simulator [7], together with the Rice Monarch Project wireless and mobility extensions [14]. *ns*-2 is a discrete event simulator widely used in network protocol research. It is written in C++ and uses OTcl as a command and configuration interface. The Monarch Project wireless and mobility extensions provide arbitrary physical node position and mobility, with a realistic radio propagation model that includes effects such as attenuation, propagation delay, carrier sense, collision and capture effect.

In this chapter, we expose the models we utilize for the physical and data link layers, as well as reasonable movement and traffic scenarios. We also specify the metrics used for the evaluation of AMPER.

To ensure our simulations run smoothly, we randomly generate scenario files (see Sections 4.3 and 4.4) in accordance with our protocol parameters, then simulate all the protocols on exactly the same set of scenarios. This is done in order to avoid that random flukes in setup – such as a network partition in one scenario file – cause a protocol to perform worse than the others.

4.1 Physical Layer Model

We keep the built-in *ns*-2 [7, 14] signal propagation model that combines free space propagation and two-ray ground reflection: when the transmitter is within a certain

reference distance from the receiver, the signal attenuates as $1/r^2$; beyond this distance, it weakens as $1/r^4$. Each mobile node uses an omni-directional antenna with unity gain, and has one or more wireless network interfaces, with all the interfaces of the same type linked together by a single physical channel.

The position of a node at a certain instant is used by the radio propagation model to calculate the propagation delay from a node to another and to determine the power level of a received signal at each node. If the power level is below the carrier sense threshold, the packet is discarded as noise; if it is above the carrier sense threshold but below the receive threshold, the packet is marked in error before being passed to the MAC layer; otherwise, it is simply transmitted to the MAC layer.

4.2 Data Link Layer Model

The link layer implements the IEEE 802.11 Distributed Coordination Function (DCF) [4]. DCF uses an RTS/CTS/DATA/ACK pattern for all unicast packets, and simply sends out DATA for all broadcast packets. The implementation uses both physical and virtual carrier sense for collision avoidance.

An Address Resolution Protocol (ARP) module is used to resolve IP addresses into link layer addresses. If ARP possesses the hardware address for the destination, it writes it into the MAC header of the packet; otherwise, it broadcasts an ARP query and caches the packet temporarily. For each unknown destination hardware address, the ARP buffer can hold a single packet. In case additional packets for the same destination are sent to the ARP module, the earlier buffered packet is dropped.

4.3 Movement Model

The simulations are run in an ad hoc network comprising 50 mobile nodes moving according to the random waypoint model [2, 9] in a flat rectangular topology (1500m $\times$ 300m). In the random waypoint model, each node begins at a random location and remains stationary for a specified pause time. It then selects a random destination

and moves to it at a speed uniformly distributed between 0 and some maximum speed. Upon reaching that destination, the node pauses again for *pausetime* seconds, selects another destination, and repeats the preceding behavior for the duration of the simulation. In our simulations, we investigate pause times of 0, 30, 60, 120, 300, 600 and 900 seconds (where 0s corresponds to continous motion and 900s to no motion). The maximum node speeds we experiment with are 1 and 20 meters per second. The simulations are run for 900 seconds and make use of scenario files with 42 different movement patterns, three for each pair of values of pause time and maximum speed.

4.4 Traffic Model

Data traffic is produced using constant bit rate (CBR) UDP sources, with mobile nodes acting as traffic sources generating 4 packets per second each. We present results with 10, 20, and 30 connections, which we believe are sufficient to illustrate the general trends in protocol performance. The packet sizes investigated are 64 bytes and 256 bytes. Following the Broch *et al.* [2] recommendation, we avoid using TCP sources because of TCP's conforming load property: TCP changes the times at which it sends packets based on its perception of the network's ability to carry packets. This will hinder comparisons between protocols, since the time at which each data packet is originated by its sender and the position of the node when sending the packet would differ between the protocols.

4.5 Metrics of Evaluation

We choose to evaluate AMPER according to the following metrics:

- *Packet delivery ratio*: The fraction of data packets that are successfully delivered to their destinations. This specifically denotes data packets sent by the application layer of a (CBR) source node and received by the application layer at the corresponding (CBR) destination node. For CBR sources, the packet delivery ratio is equivalent to the goodput.

- *Packet delivery latency*: The average end-to-end delay required between originating a data packet and successfully delivering it to its intended destination.

- *Routing packet overhead*: The average number of separate overhead packets (per second) used by the routing protocol. For packets sent over multiple hops, every transmission – whether original or forwarding – is counted.

- *Total bytes of overhead*: The average number of bytes of routing overhead (per second), including the size of all routing overhead packets and the size of any protocol headers added to data packets. For packets sent over multiple hops, every transmission – whether original or forwarding – is counted.

- *Route acquisition latency*: The time needed for the network layer to send the first packet of a flow after receiving it from the application layer of the (CBR) source.

Chapter 5

Simulation Results

Before proceeding to the simulation and evaluation of our protocol, it is worth mentioning that AMPER is very computationally intensive for the nodes to run, which has the downside that running simulations of AMPER consumes a great deal of computer time. We have not tried to design a computationally-efficient routing protocol, as we believe that laptops with modern network cards can handle more or less anything we can throw at them. Of course, some ad hoc applications may be using handheld devices that do not possess substantial memory and CPU power, but this should not be terribly burdensome when the ad hoc network is small[1].

We do not compare AMPER to other probabilistic routing protocols, but rather choose to contrast it with DSDV [11], DSR [9] and AODV [12] for the following reasons:

- The DSDV, DSR and AODV implementations are offered as part of *ns*-2, which makes it easy to simulate them, whereas the evaluation of the probabilistic routing protocols requires writing their implementation from scratch. Besides being extremely time-consuming, the latter is a tremendous hassle since it involves understanding those protocols in great detail[2].

- Previous simulation results of DSDV, DSR and AODV are more detailed and

[1]We should mention again that AMPER is designed to operate in small-sized networks.

[2]This seems like a utopic goal (at least for us); we think the papers in question contain many unclear or questionable issues.

more refined than their probabilistic counterparts, which allows us to better check the accuracy of our own results.

- DSDV, DSR and AODV are popular and widely accepted among the networking community; trying to compete with them is challenging yet rewarding.

The remainder of this chapter is organized as follows: Section 5.1 is a note on AMPER parameter values. In Section 5.2, we present and compare the performance of the four protocols on each of our metrics of evaluation. In Section 5.3, we present the variation of the delivery ratio of each protocol as a function of the size of the packets sent by the CBR sources. In Section 5.4, we present the variation of the delivery ratio of each protocol as a function of the number of connections in the network. In Section 5.5, we show how the overall performance of the protocols changes when the nodes slow down to a maximum speed of one meter per second.

5.1 Values of AMPER Parameters

We varied the values of the AMPER parameters in an effort to maximize the protocol performance. The values of the major parameters that we adopted for the simulations presented below are shown in Table 5.1.

The frequency of major state update, F, reflects the rate at which the neighbor list, the loss ratio state and the routing state are refreshed at each node. This process mainly involves the removal of old neighbors and all associated information, the elimination of destinations with expired declaration of unreachability, and the rebalancing of probability values in the routing tables. F is critical for correct operation of the protocol: if it is set to a value greater than 1.0s, performance degradation becomes noticeable. The value of the frequency of minor state update, f, governs the rate at which the node cooperation state updates are carried out. If f exceeds 0.1s, destination requests will not be propagated reasonably swiftly, even when there is traffic to piggyback them on. Smaller values of F and f make the nodes better aware of the current network state, but may involve a performance tradeoff. In fact,

the nodes will be able to respond quicker to network changes, but if those changes do not happen frequently, smaller values of F and f will consume more bandwidth and will require an additional cost to access the wireless medium. Moreover, very frequent updates may cause packet retransmissions, especially when TCP is used. In addition, it would make the simulations run slower[3].

Unlike the case of F and f, AMPER is not very sensitive to the value of the destination reachability timeout t_d and that of the request timeout t_r. If a certain destination is determined to be unreachable within t_d, the sender drops the packet addressed to this destination. Setting t_d to a value less than 0.5s causes more packet drops, whereas setting it to any value between 2.0s and 4.0s gives approximately the same performance. The value of the request timeout, t_r, affects the number of requests for information that a node can make or listen to before clearing the previous requests. For values of t_r smaller than 0.05s, the nodes start to lag – their ability to follow network changes becomes limited.

AMPER is extremely sensitive to the link loss ratio sampling window length: t_l directly affects the contents of the protocol header and the metric values between nodes, which in turn determines the routing dynamics inside the network. For values of t_l smaller than 1.8s, nodes fail to get an accurate knowledge of the network state and may route along non-optimal paths. For values of t_l bigger than 2.2s, nodes start to behave erratically when their list of neighbors changes or some nearby link breaks.

Finally, AMPER is relatively insensitive to the values of the silent neighbor timeout t_n and the maximum number of request retries m_{ret}. The former determines how long a node should wait before removing a neighbor from which it has not heard anything, and the latter dictates how many times a request for information can be repeated before it is dropped. Values of t_n between 2s and 6s, and a range of 5 to 18 for m_{ret} both seem to deliver a consistent performance.

Since the focus of our efforts was AMPER, we did not vary any parameters for the other three protocols, but rather trusted that their implementers had chosen optimal values to use in *ns*-2.

[3]A 40% decrease in F and f makes the simulations run approximately 3 times slower.

F	frequency of major state update	0.5s
f	frequency of minor state update	0.05s
t_d	destination reachability timeout	2.0s
t_r	request timeout	0.15s
t_l	link loss ratio sampling window length	2.0s
t_n	silent neighbor timeout	3.0s
m_{ret}	maximum number of request retries	10

Table 5.1: Values of AMPER parameters

5.2 Performance Overview

Figures 5-1 through 5-10 give an overview of the performance of the four protocols with respect to the five metrics we tested them on. The simulations are all run at a maximum node speed of 20 m/s, as we believe that faster nodes present a more appropriate challenge for the routing protocols. Each plot presents the metric versus the pause time, and every point in these plots is the average of the values as the connection count is varied over the values 10, 20, and 30, and the packet size is varied over the values 64 bytes and 256 bytes. A more detailed presentation of the effects of the packet size and the number of connections on the delivery ratio is presented in Sections 5.3 and 5.4, and an overview of the effect of reduced maximum node speed follows in Section 5.5.

5.2.1 Delivery Ratio

Figure 5-1 presents the ratio of application packets delivered by each protocol as a function of pause time[4]. DSDV's delivery ratio is quite low at higher rates of mobility[5], but improves as nodes slow down. The majority of the packets are being lost because a stale routing table entry directs them to be forwarded over a broken link. AODV and DSR have consistent trends, and deliver at least 95% of the packets in all cases. AMPER offers excellent performance as well, and outperforms its competitors except at the 30s pause time, where it presents a downward spike. We do not know the cause

[4]We have arbitrarily chosen AMPER's random mode in all comparisons with DSDV, DSR and AODV. Separate graphs will contrast the non-random and random AMPER modes.

[5]This corresponds to lower pause times.

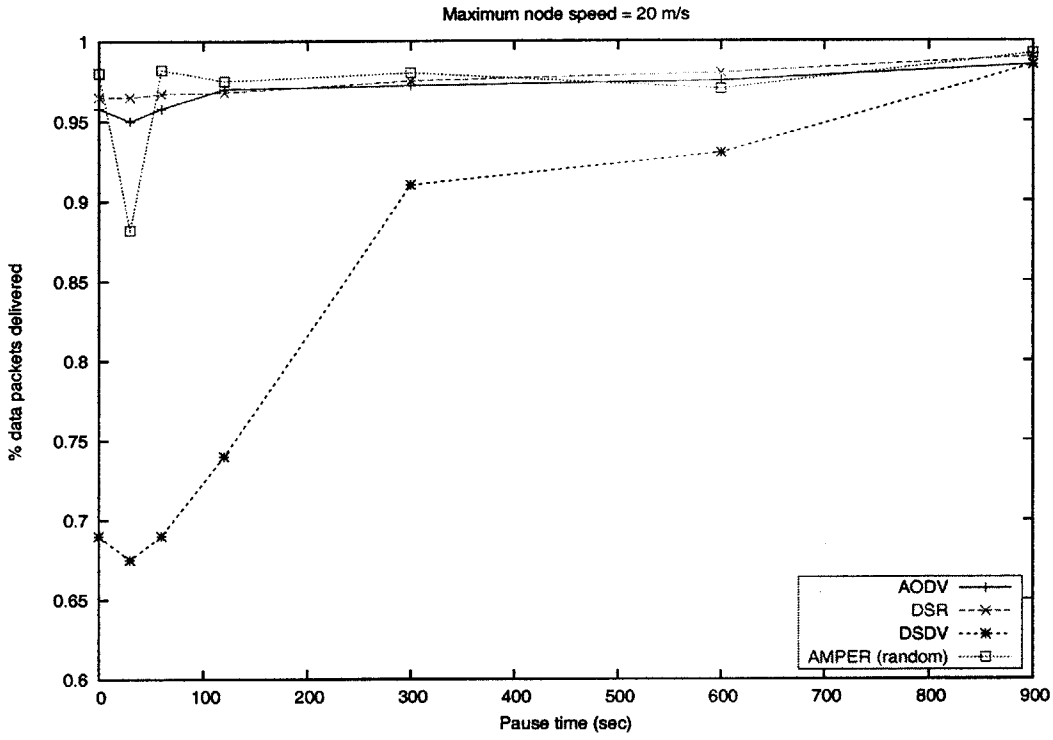

Figure 5-1: Delivery Ratio Overview

of this irregularity; it may be an artifact of the random scenario files or the result of a certain interaction between the AMPER timeouts.

Figure 5-2 presents the ratio of application packets delivered by AMPER as a function of pause time. We investigate both the random and non-random modes, as our ultimate objective is to conclude which one offers better performance. Aside from the spike at the 30s pause time, AMPER delivers at least 95% of its data packets in both modes. The random mode seems to offer a slightly superior performance, except for pause times greater than 600 seconds, where its delivery ratio curve practically coincides with the non-random mode's one.

5.2.2 Delivery Latency

Figure 5-3 presents the average amount of time that successfully delivered packets took to be delivered. DSR's delivery latency is clearly larger than its competitors'; this is mainly due to the cost involved in processing the source route at each DSR node. DSDV, AODV and AMPER all offer delivery latencies that are within 250

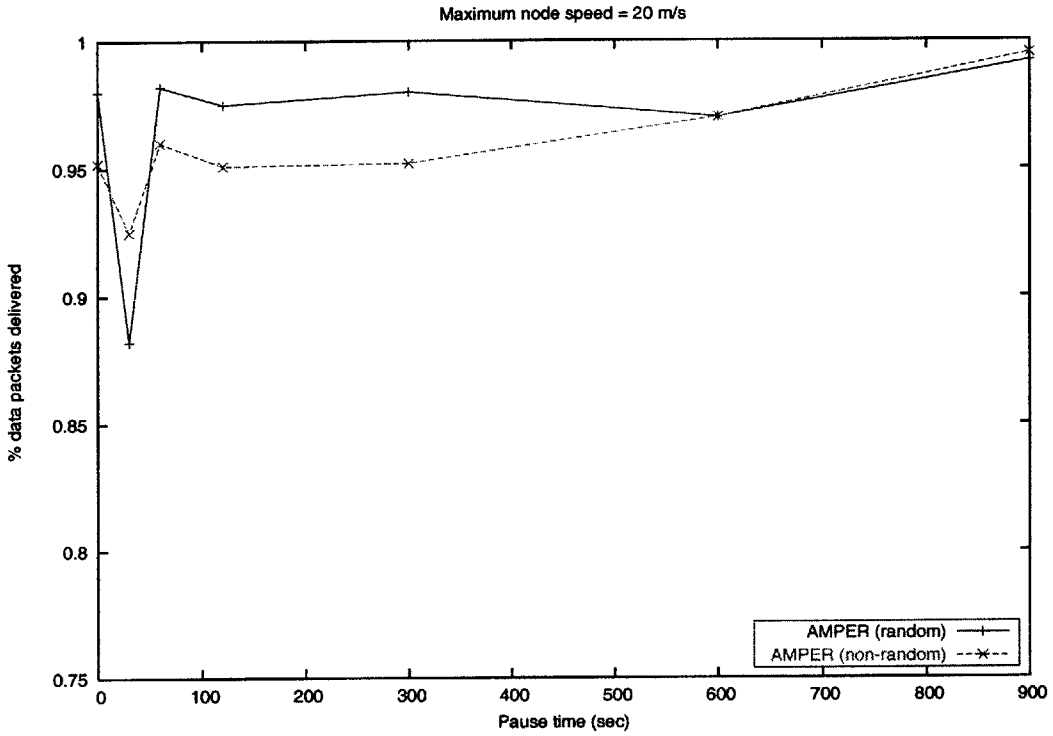

Figure 5-2: AMPER's Delivery Ratio

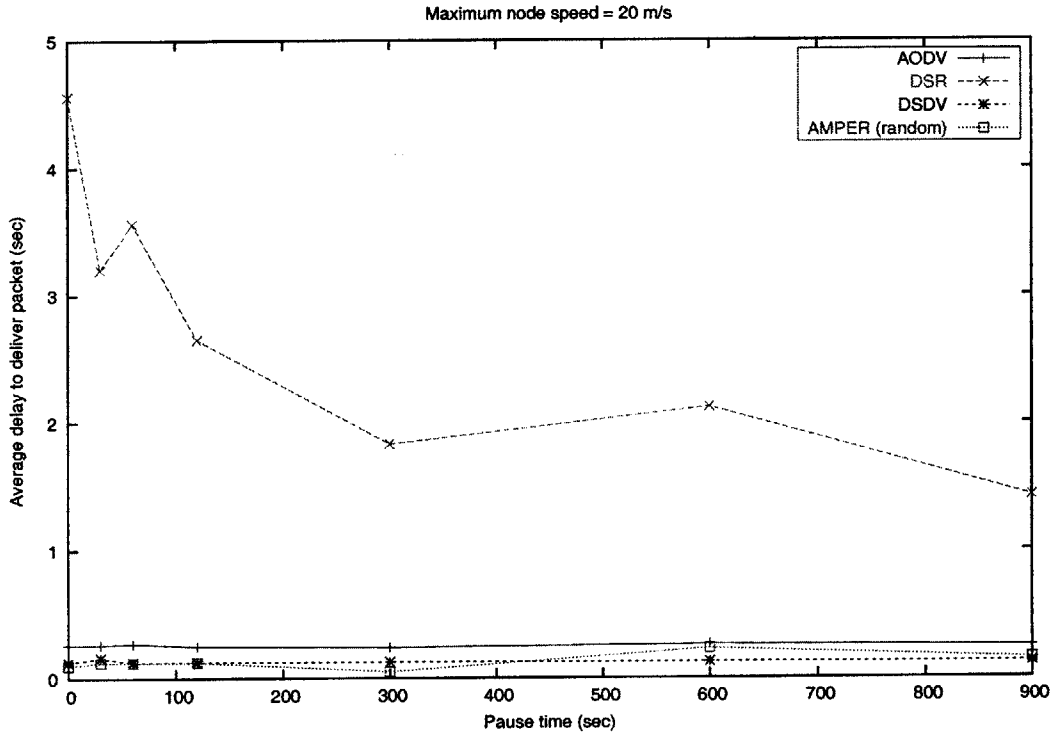

Figure 5-3: Delivery Latency Overview

milliseconds.

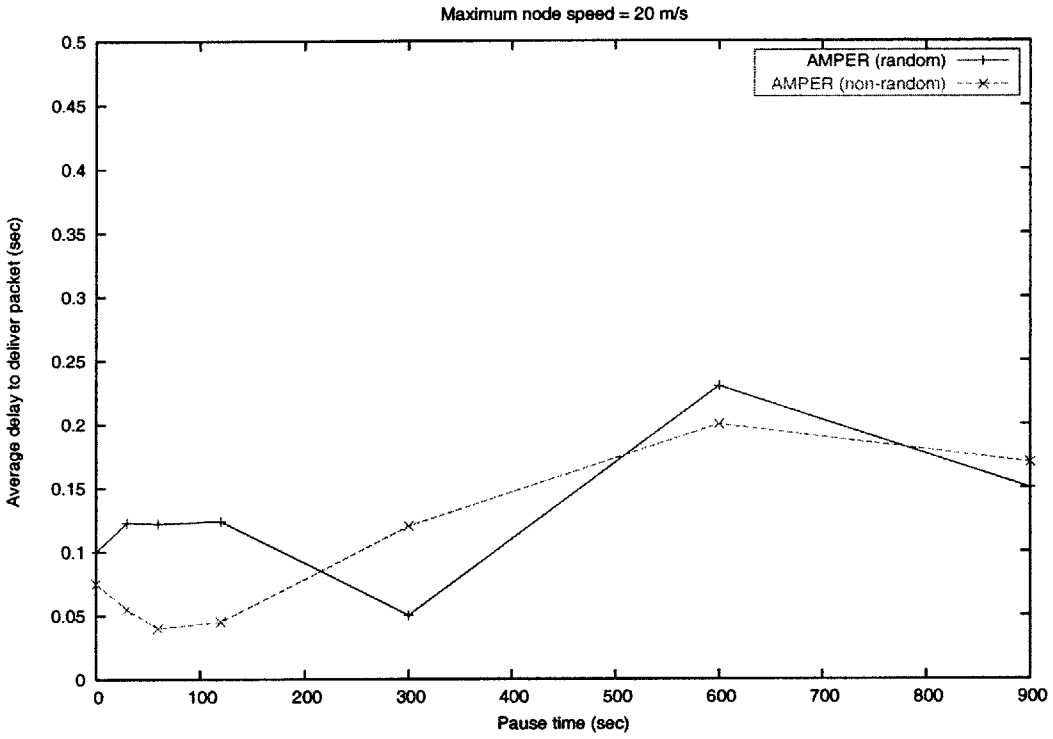

Figure 5-4: AMPER's Delivery Latency

Figure 5-4 presents the average amount of time that successfully delivered AMPER packets took to be delivered. Non-random AMPER peaks at less than 200 milliseconds, and offers lower delivery latencies at higher node mobility rates. The cost of reordering in random AMPER seems to be the cause; however, this effect is not noticeable for pause times greater than 300 seconds. Overall, non-random AMPER appears to be slightly favorable to its random counterpart when the goal is to minimize the delivery latency.

5.2.3 Discovery Latency

Figure 5-5 presents the average amount of time that a connection waited for a route to its destination to be initially discovered. AMPER is competitive with AODV and DSR in this domain, and they all have possess nearly constant discovery latencies that are within 350 ms. The high discovery latency for DSDV is mainly due to the fact that it acts periodically and in response to network changes, rather than in response

to application demand for routes. This explains why its discovery latency oscillates and does not converge when the pause time reaches 900 seconds.

Figure 5-6 presents the average amount of time that a connection waited for a route to its destination to be initially discovered when using AMPER. The random and non-random modes both offer excellent discovery latencies, but the former seems to be slightly better in this domain.

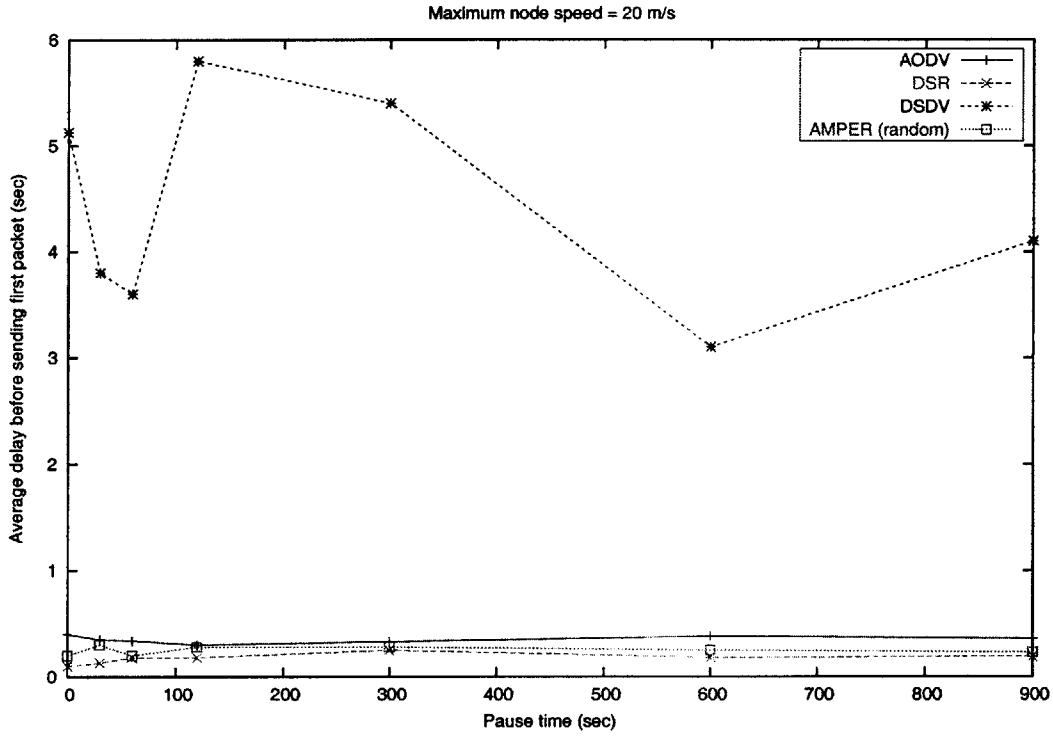

Figure 5-5: Discovery Latency Overview

5.2.4 Routing Overhead

Figures 5-7 through 5-10 present two measures of routing overhead incurred by the protocols we studied. The first is the average number of routing packets each protocol sends per second, and the other is the average number of overhead bytes each protocol transmits per second. The latter includes both the total size of protocol packets and the total size of headers appended to data packets. In both cases, retransmissions of protocol packets are counted separately, as each retransmission costs the network more bandwidth.

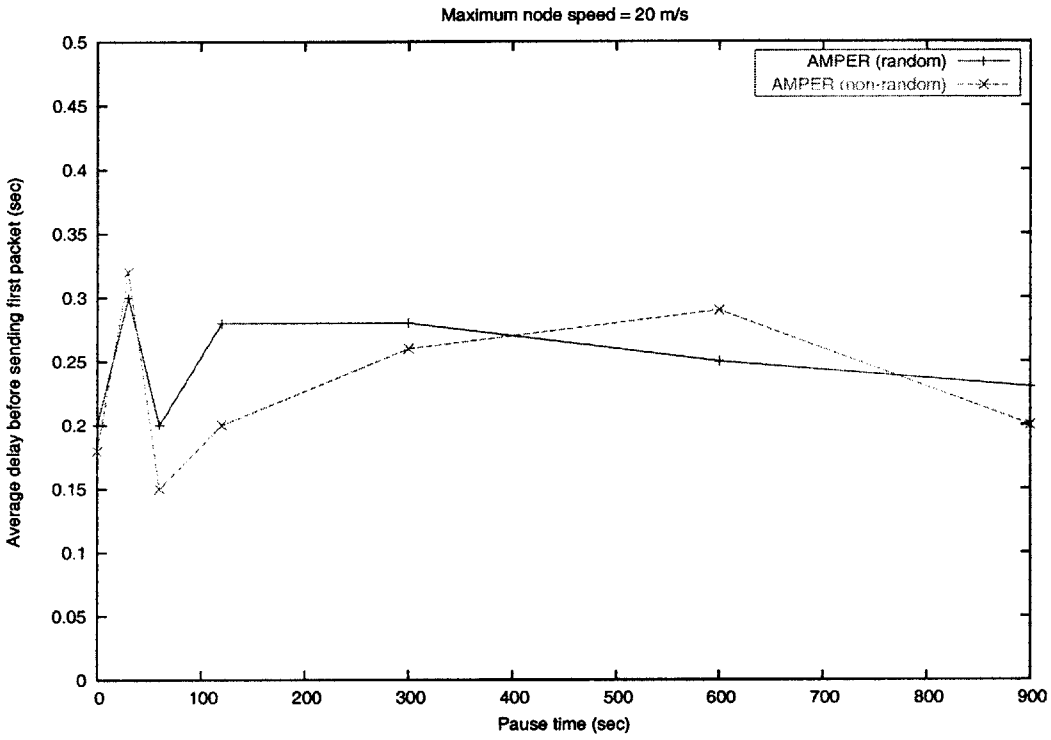

Figure 5-6: AMPER's Discovery Latency

This brings up a detail that should be mentioned: when counting the "size" of protocol-generated packets, we include the IP header as part of the overhead incurred by the protocol, whereas on data packets we do not. The reason is that the IP header would appear on the data packet anyway, but the IP header of the protocol packet would not burden the network if the protocol packet were never transmitted.

Also note that we do not count any RTS/CTS expense that protocol packets may incur. Although such expense is likely to be nontrivial, we choose not to include it for the following reasons:

- It is not very easy to compute from *ns*-2 trace files which packets are RTS/CTS-ed and which are broadcast (the routing protocols we compare AMPER against do both).

- It is difficult to exactly determine how many times an RTS is sent before it is answered.

- It is not immediately clear how much bandwidth an RTS/CTS exchange con-

sumes.

- AMPER broadcasts all the control packets that it sends, so not counting the RTS/CTS expense does not artificially improve our results.

Routing Packet Overhead

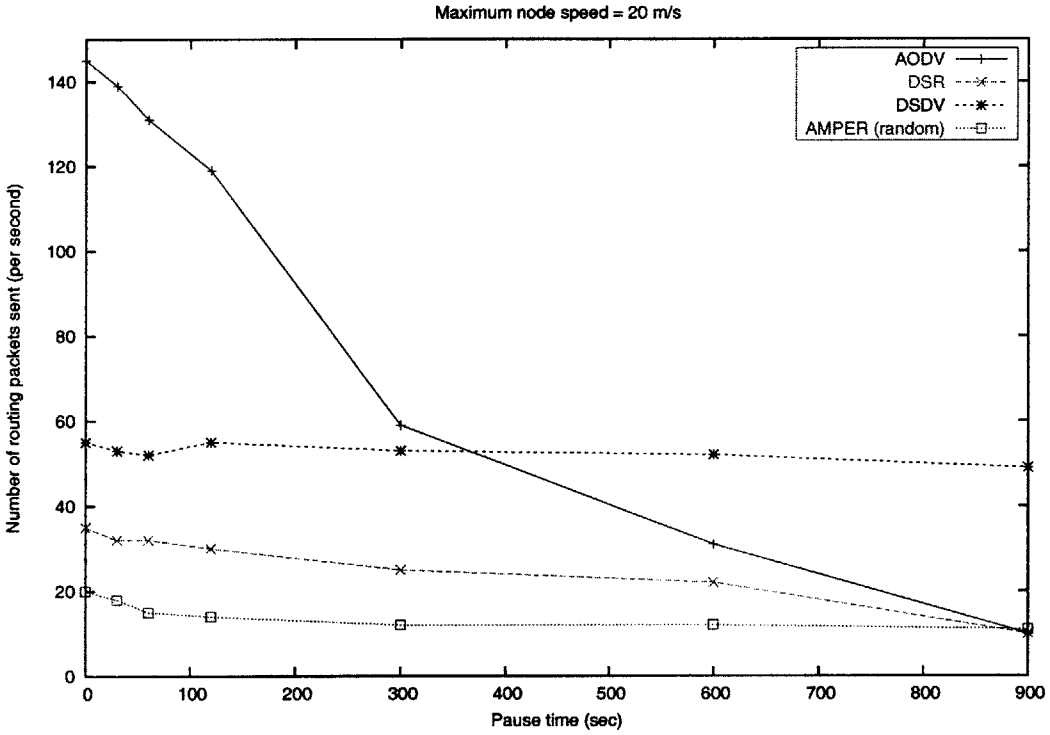

Figure 5-7: Routing Packet Overhead Overview

We have designed AMPER to minimize control packets. Figure 5-7 proves that our protocol indeed sends the lowest number of routing packets regardless of the node mobility rate. At very high speeds, DSR requires twice the number of control packets, and DSDV sends approximately three times that number. Since DSDV is a periodic routing protocol, its routing packet overhead is nearly constant with respect to mobility rate. DSR and AODV are both on-demand protocols, which is why their overhead drops as the mobility rate drops. However, AODV's packet overhead is noticeably higher than DSR's, mainly because AODV makes extensive use of HELLO packets for neighbor discovery.

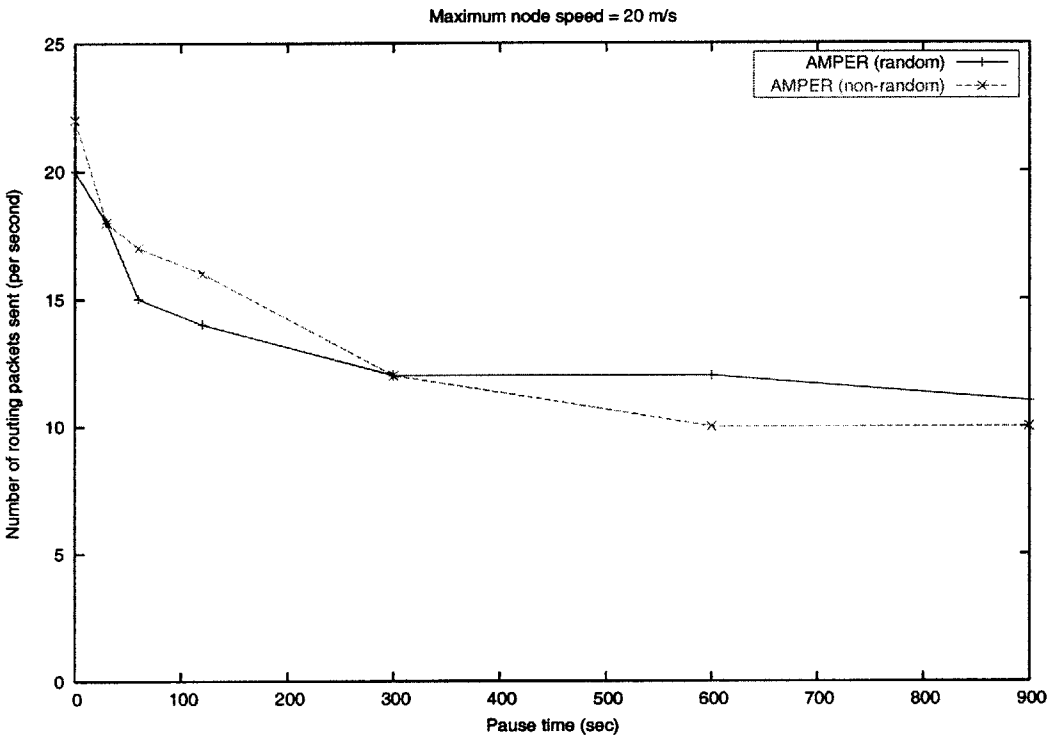

Figure 5-8: AMPER's Routing Packet Overhead

When ran in non-random mode, AMPER produces nearly the same graph as in the random case (Figure 5-8). The number of routing packets sent by our protocol does not seem to depend on the type of probabilistic scheme it uses.

Total Bytes of Overhead

When the routing overhead is measured in bytes and includes the size of the source route header that DSR places in each packet, DSR becomes more expensive than AODV, but still transmits fewer bytes of overhead than DSDV (Figure 5-9). The reason behind AODV's superior performance is its creation of hop-by-hop routing state in each node along a path in order the eliminate the overhead of source routing. Our protocol competes well with DSDV and DSR: although AMPER's header may look large at first glance, we have been able to optimize the usage of header space sufficiently so that the savings on protocol packet transmissions offset the header sizes[6].

[6]A more thorough discussion of AMPER's header size follows in Chapter 6.

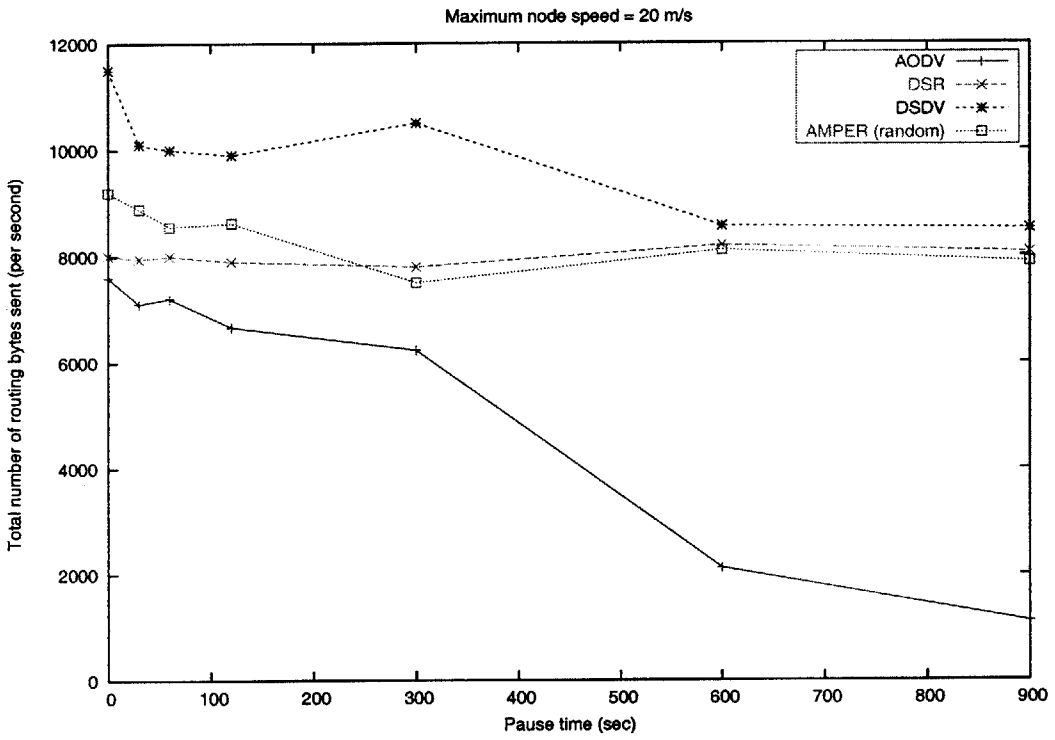

Figure 5-9: Total Overhead Overview

It is worth noting that most of AODV's overhead bytes are carried in many small packets. The cost to acquire the medium to transmit a packet is significantly more expensive in terms of power and network utilization than the incremental cost of adding a few bytes to an existing packet [2]. Consequently, the actual cost of the DSR and AMPER headers is effectively less than that indicated by the number of bytes.

Unsurprisingly, Figure 5-10 shows that the total overhead incurred by AMPER does not depend on the forwarding scheme it uses: AMPER produces approximately the same overhead when operating in random or non-random mode.

5.3 Delivery Ratio Variation with the Packet Size

In Figures 5-11 through 5-15, we show how the performance of the four protocols – as captured by the delivery ratio metric – varies when changing the size of the packets sent by the CBR sources. As expected, bigger packets degrade performance. It also

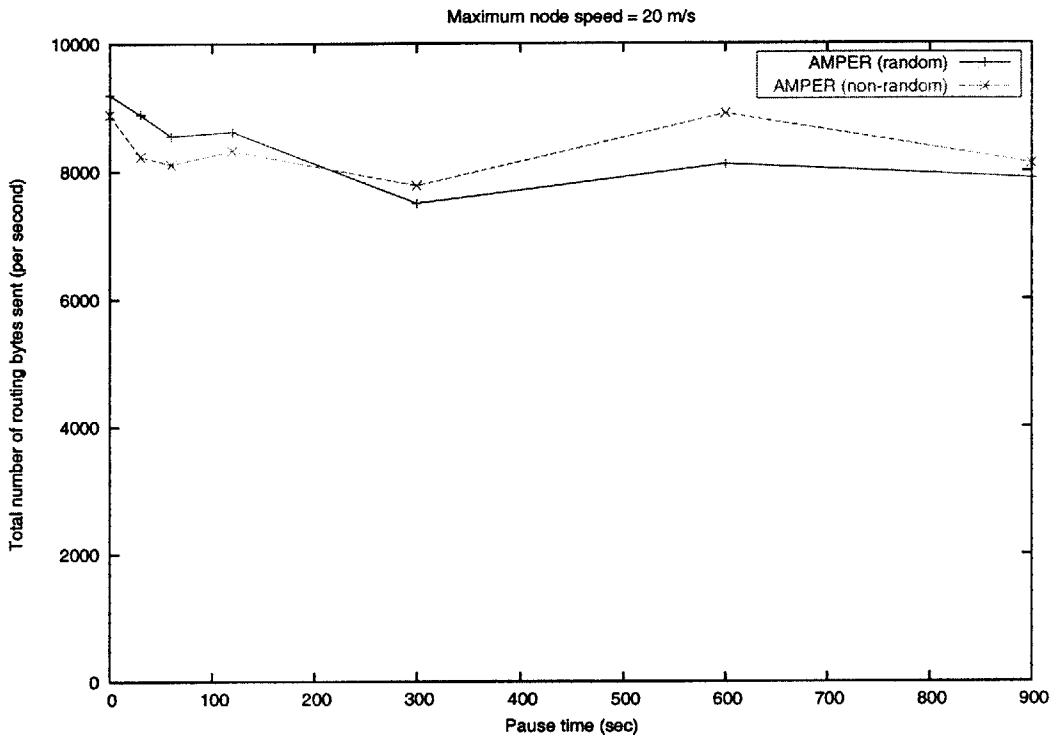

Figure 5-10: AMPER's Total Overhead

appears that all four protocols degrade with roughly equal grace. In the AMPER plots (Figures 5-14 and 5-15), the spike at the 30s pause time reappears. Once again, it may be either an artifact of the randomness in our scenario files, or the result of a interaction between the protocol parameters.

5.4 Delivery Ratio Variation with the Number of Connections

In Figures 5-16 through 5-20, we show how the performance of the four protocols – as captured by the delivery ratio metric – varies when changing the number of connections. The plots show that varying the number of connections instead of the packet size also produces network congestion and performance degradation, the effect becoming seriously pronounced with 30 connections. Here again, the downward spike at the 30s pause time shows itself.

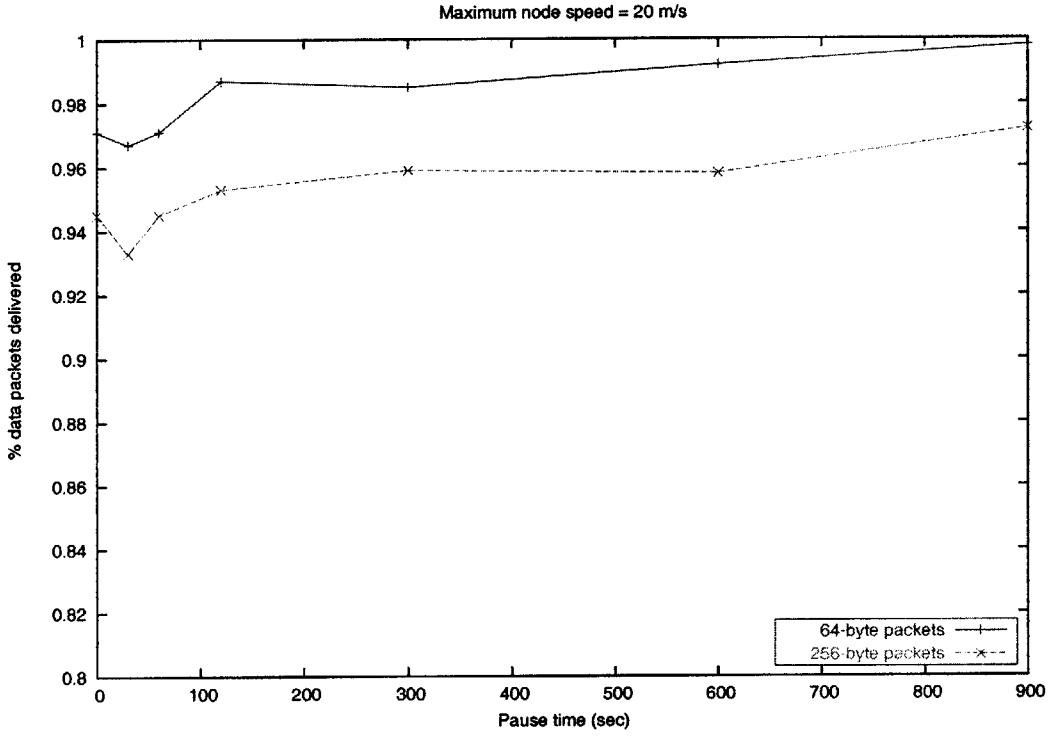

Figure 5-11: AODV – Delivery Ratio Variation with Packet Size

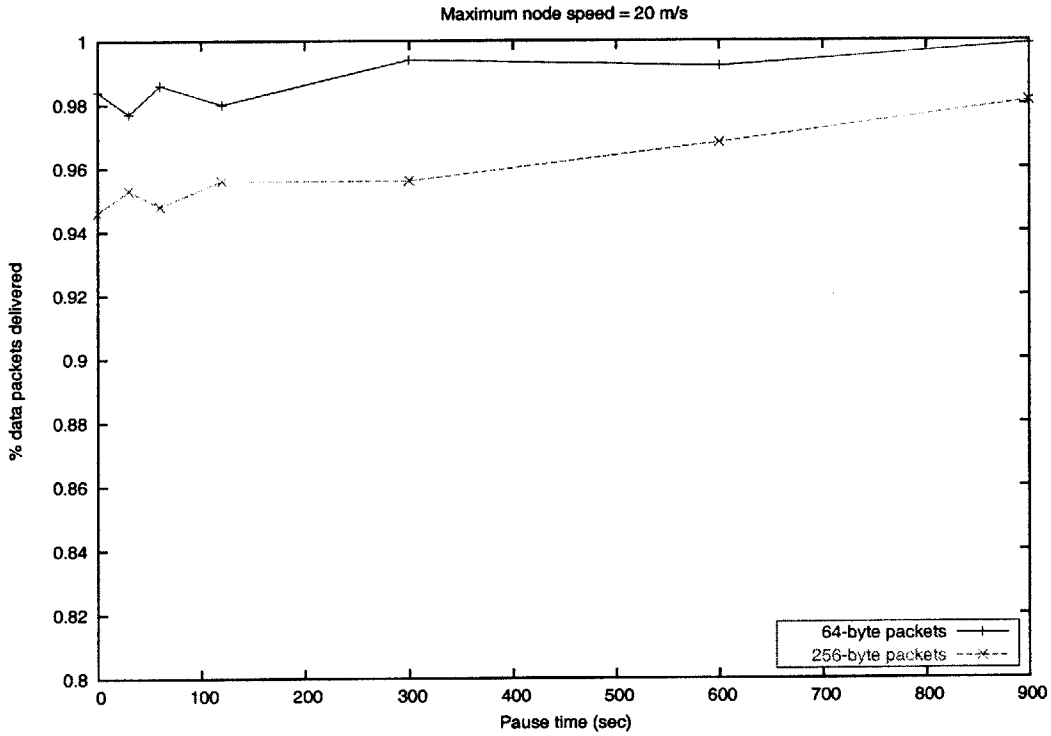

Figure 5-12: DSR – Delivery Ratio Variation with Packet Size

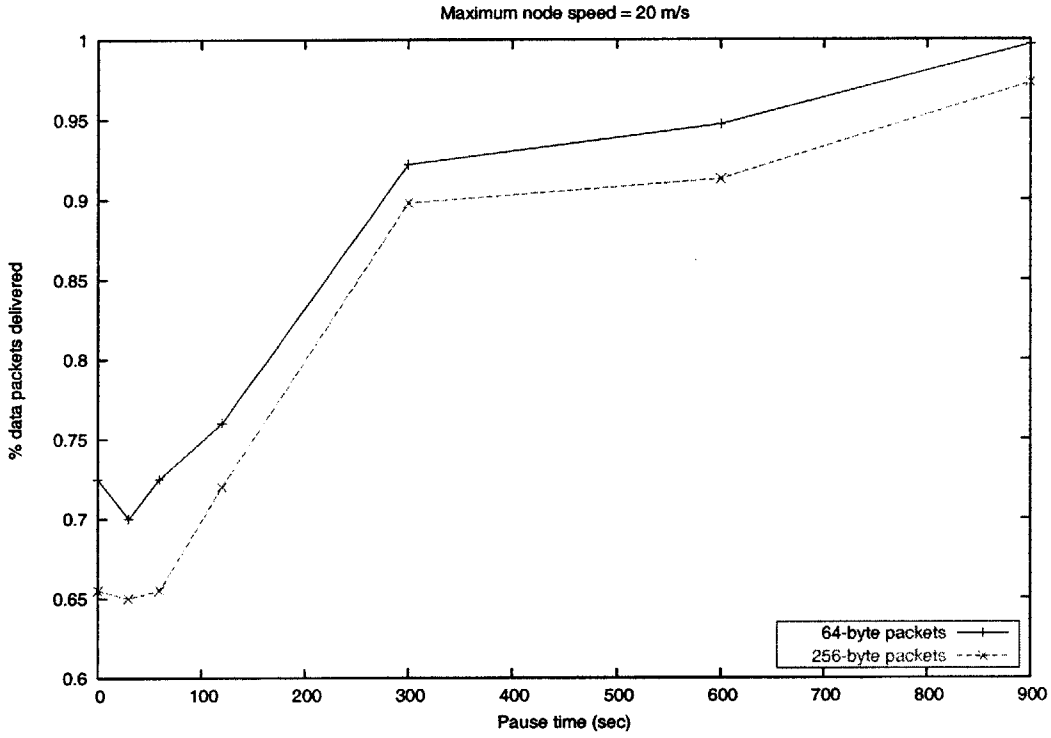

Figure 5-13: DSDV – Delivery Ratio Variation with Packet Size

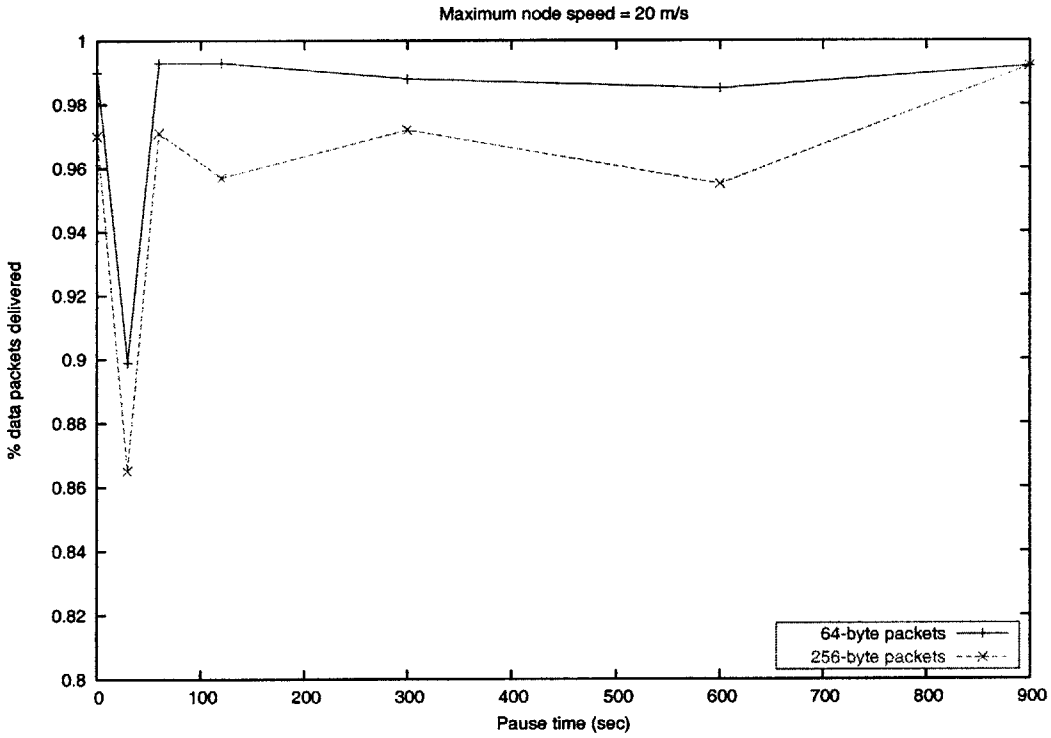

Figure 5-14: Random AMPER – Delivery Ratio Variation with Packet Size

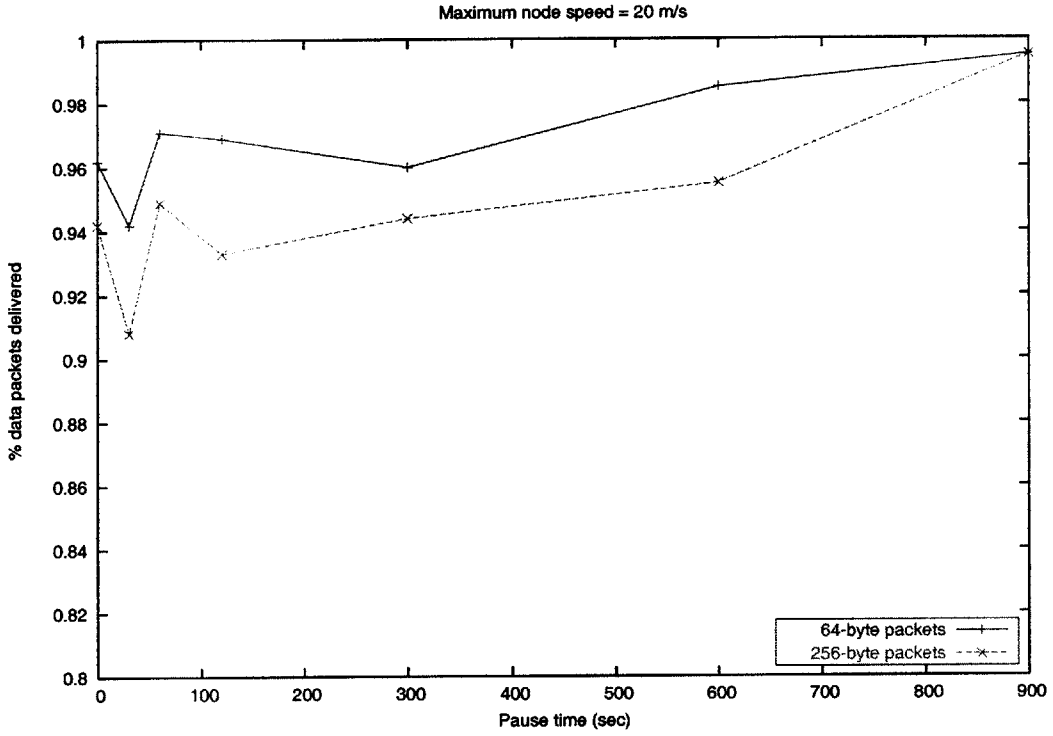

Figure 5-15: Non-Random AMPER – Delivery Ratio Variation with Packet Size

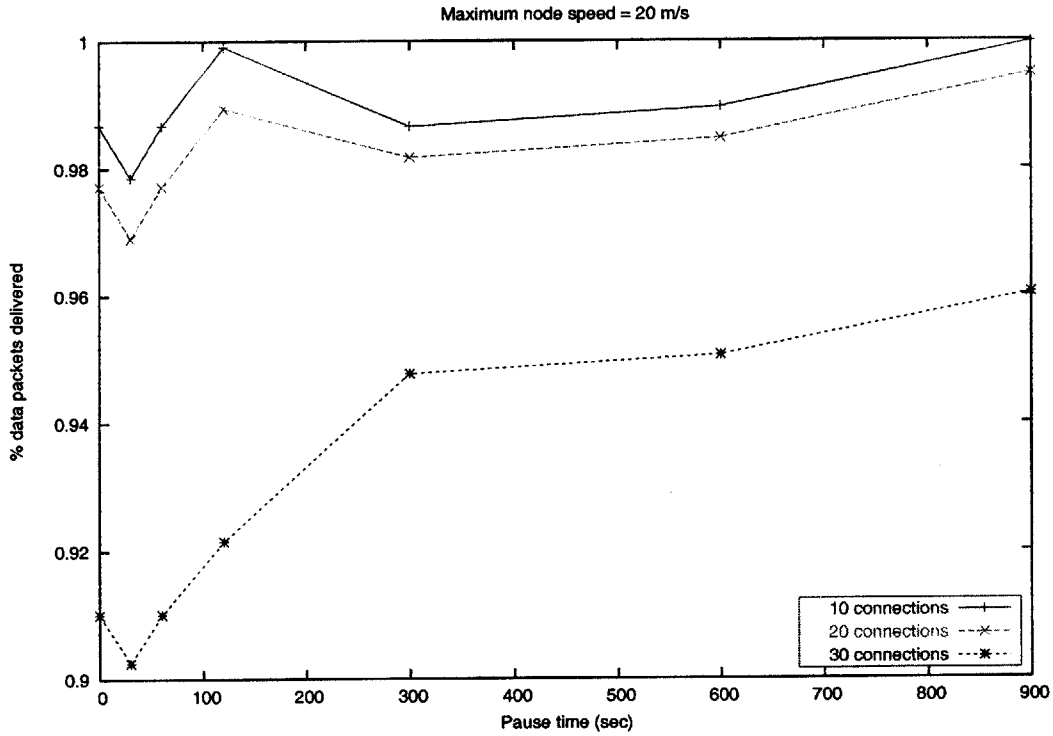

Figure 5-16: AODV – Delivery Ratio Variation with Number of Connections

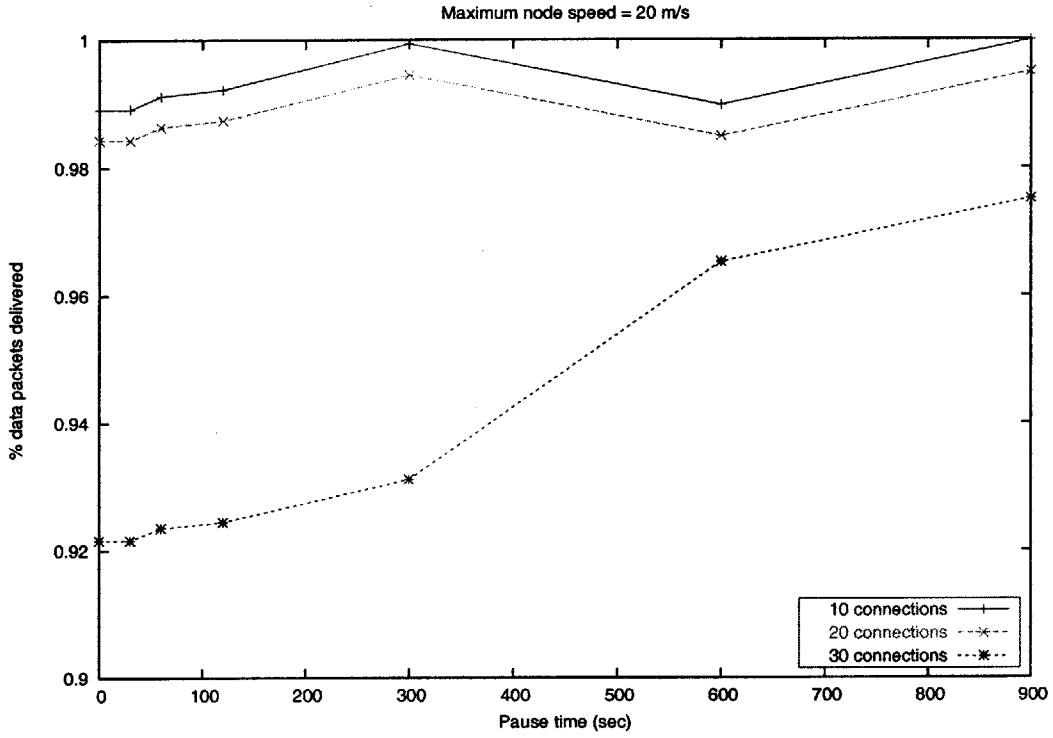

Figure 5-17: DSR – Delivery Ratio Variation with Number of Connections

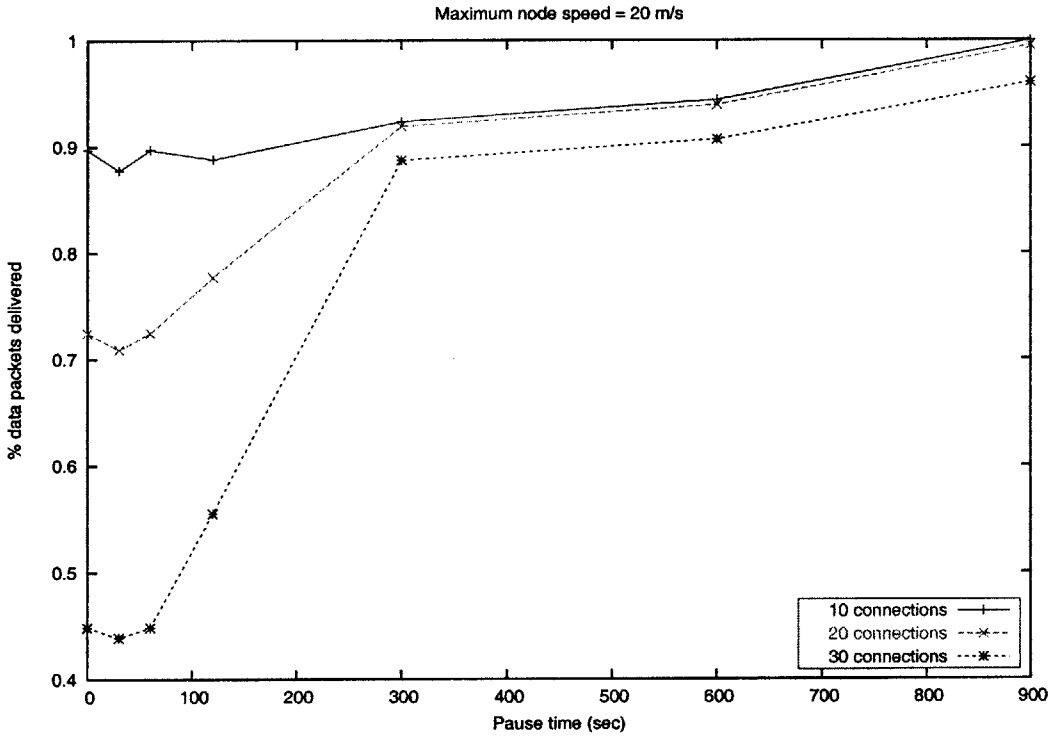

Figure 5-18: DSDV – Delivery Ratio Variation with Number of Connections

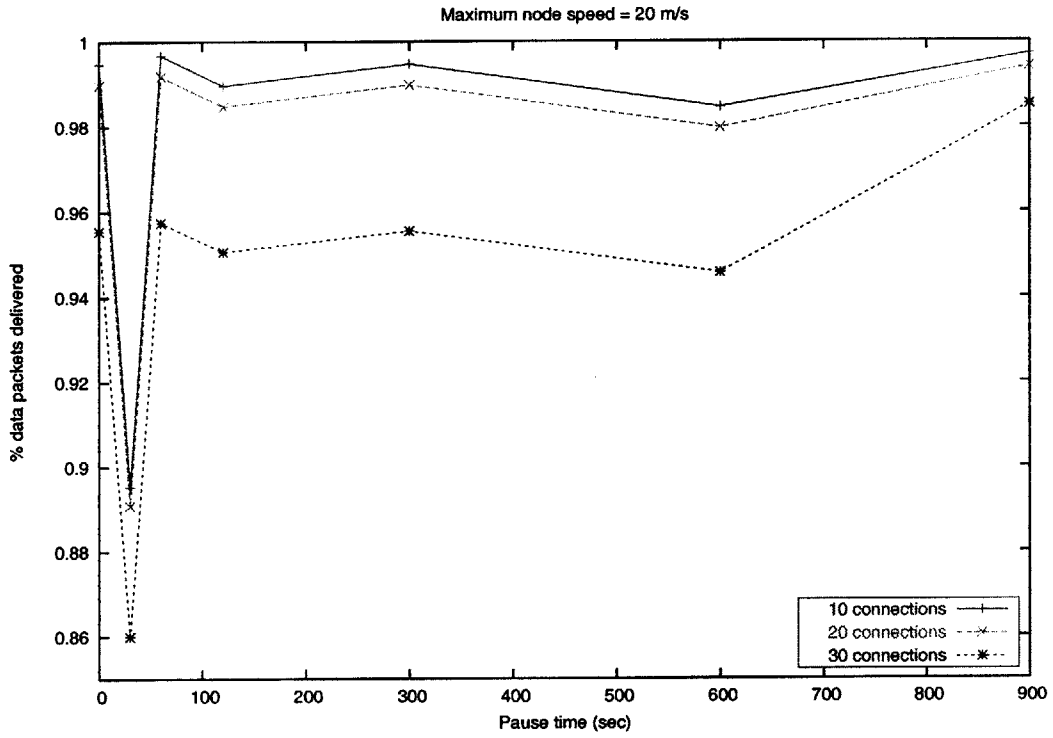

Figure 5-19: Random AMPER – Delivery Ratio Variation with Number of Connections

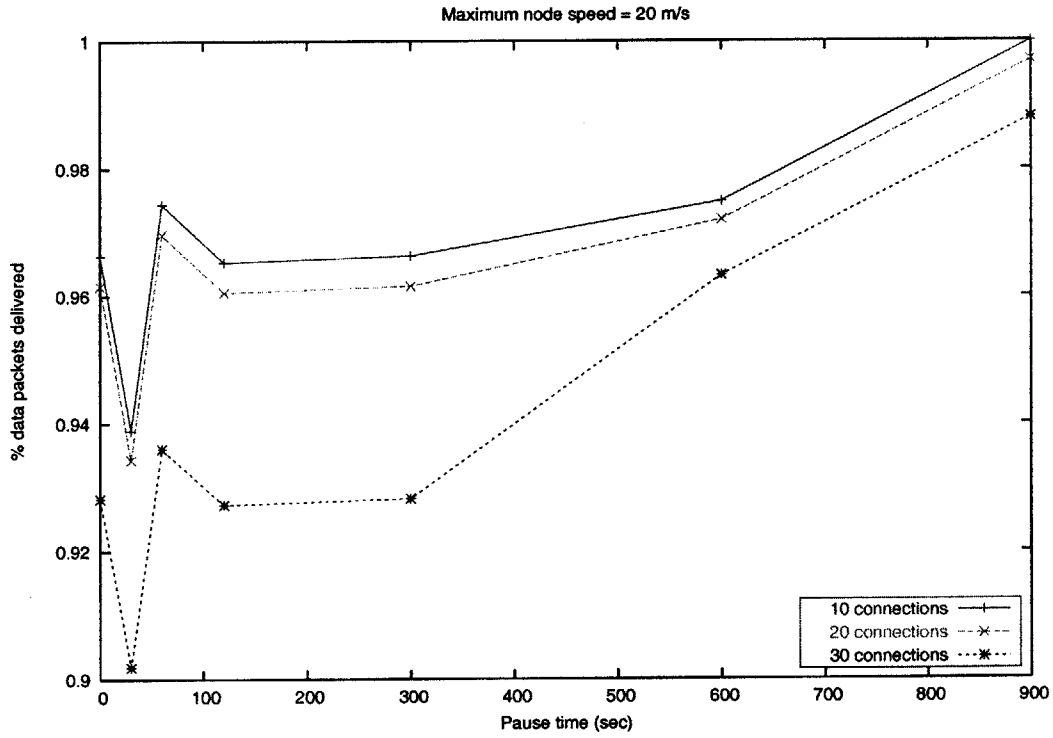

Figure 5-20: Non-Random AMPER – Delivery Ratio Variation with Number of Connections

5.5 Overview with Slow Nodes

Figures 5-21 through 5-30 illustrate the effect of slower nodes on overall protocol performance. Whereas all of our prior results were shown with the maximum node speed set to 20 m/s, these plots compare the protocols under the same metrics when nodes move far more slowly (1 m/s).

5.5.1 Delivery Ratio

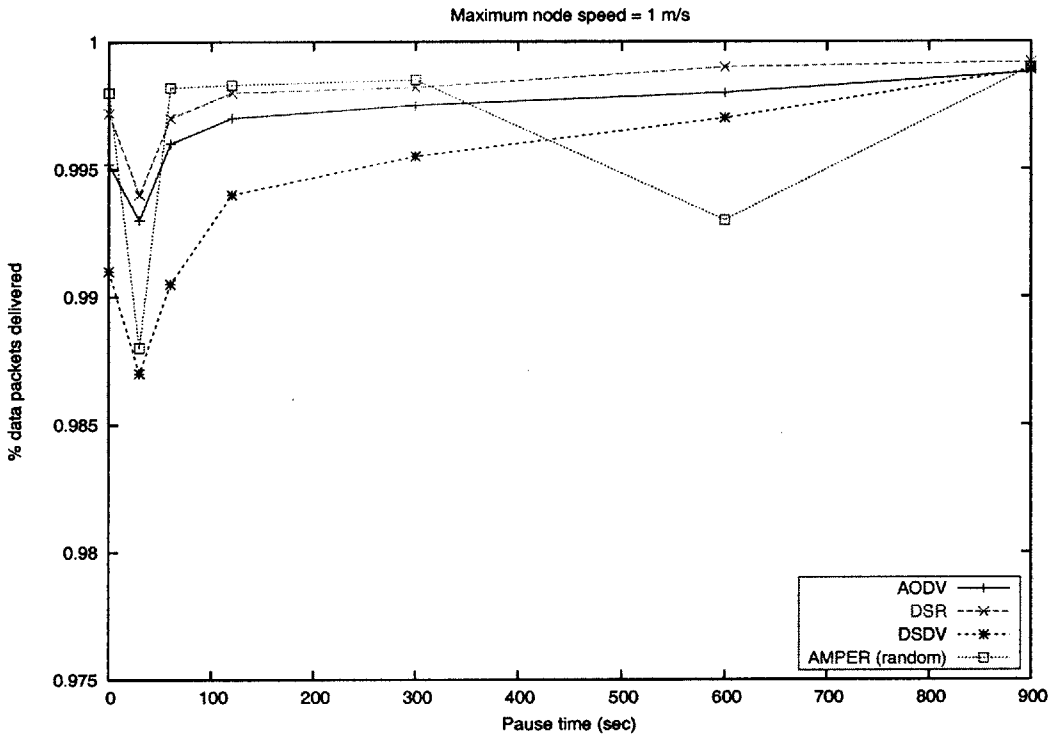

Figure 5-21: Delivery Ratio Overview with Slow Nodes

Figure 5-21 presents the ratio of application packets delivered by each protocol as a function of pause time. Unlike in the 20 m/s scenarios where DSDV had trouble converging, its delivery ratio is remarkably higher and it delivers excellent performance in the 1 m/s scenarios. AODV, DSR and AMPER all see their delivery ratio increase and converge to approximately 99.7%, instead of 95% in the 20 m/s case. AMPER seems to offer superior performance for pause times inferior to 300 seconds.

Figure 5-22 presents the ratio of application packets delivered by AMPER as a

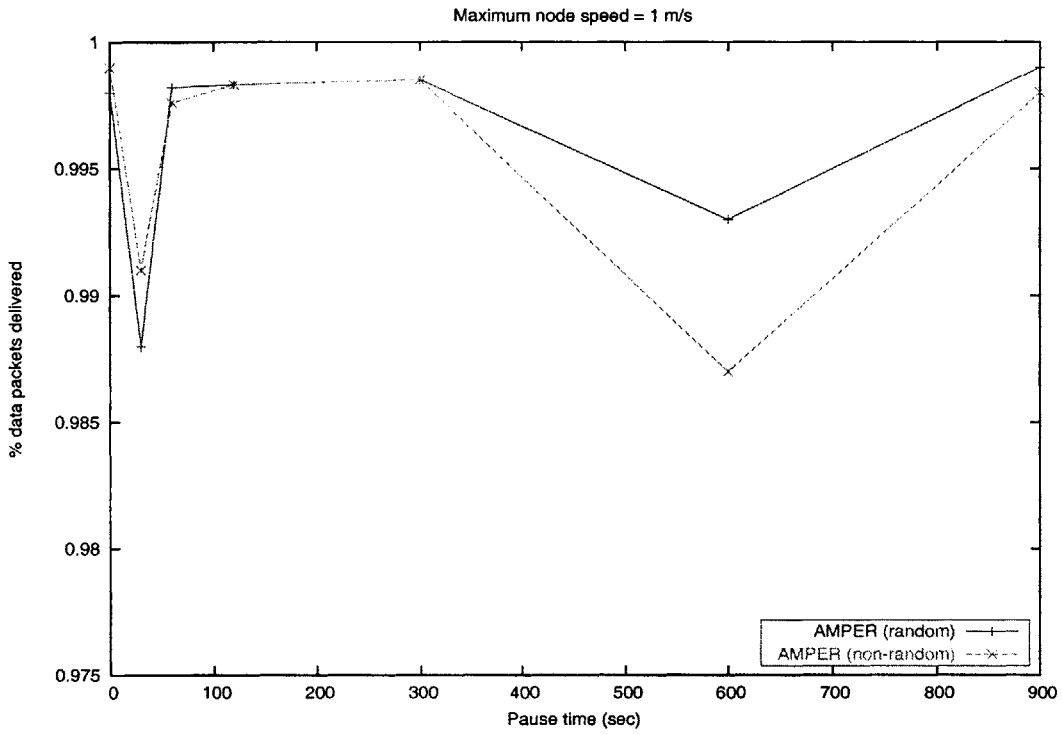

Figure 5-22: AMPER's Delivery Ratio with Slow Nodes

function of pause time. AMPER delivers at least 98.7% of its data packets in both modes. The random mode now seems to offer a superior performance for all pause times.

5.5.2 Delivery Latency

Figure 5-23 presents the average amount of time that successfully delivered packets took to be delivered in the 1 m/s scenarios. DSR's delivery latency is substantially lower than in the 20 m/s case, but it is still larger than its competitors'; this is mainly due to the cost of processing the source route at each DSR node. DSDV, AODV and AMPER still offer delivery latencies inferior to 250 milliseconds. As in the 20 m/s scenarios, the cost of reordering in random mode did not hurt AMPER's delivery latency performance.

Figure 5-24 presents the average amount of time that successfully delivered AMPER packets took to be delivered in the 1 m/s scenarios. Non-random AMPER seems to offer lower delivery latencies for most node mobility rates, the cost of reordering

in random AMPER being probably the reason.

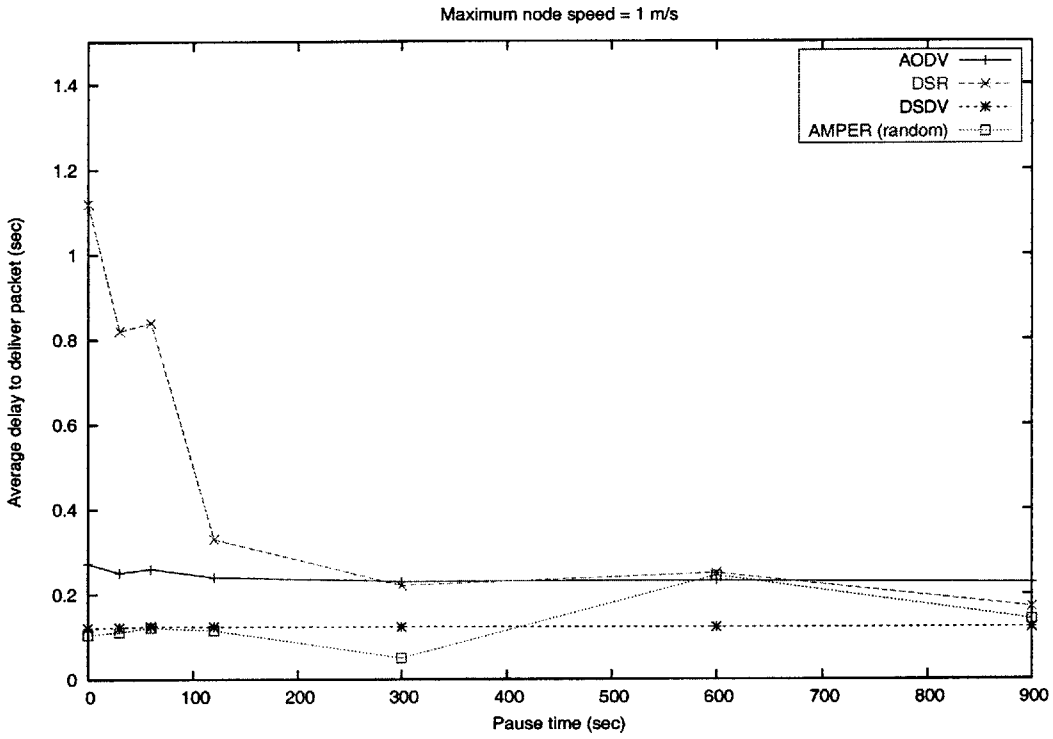

Figure 5-23: Delivery Latency Overview with Slow Nodes

5.5.3 Discovery Latency

Figure 5-25 presents the average amount of time that a connection waited for a route to its destination to be initially discovered in the 1 m/s scenarios. AMPER remains competitive with AODV and DSR in this domain, and they all have possess nearly-constant discovery latencies inferior to 350 ms. DSDV's discovery latency is now considerably lower and converges to approximately 400 milliseconds; this contrasts with its fluctuating behavior in the 20 m/s case.

Figure 5-26 presents the average amount of time that a connection waited for a route to its destination to be initially discovered when using AMPER in the 1 m/s scenarios. The random and non-random modes both offer excellent discovery latencies, but the latter seems to be slightly better in this case.

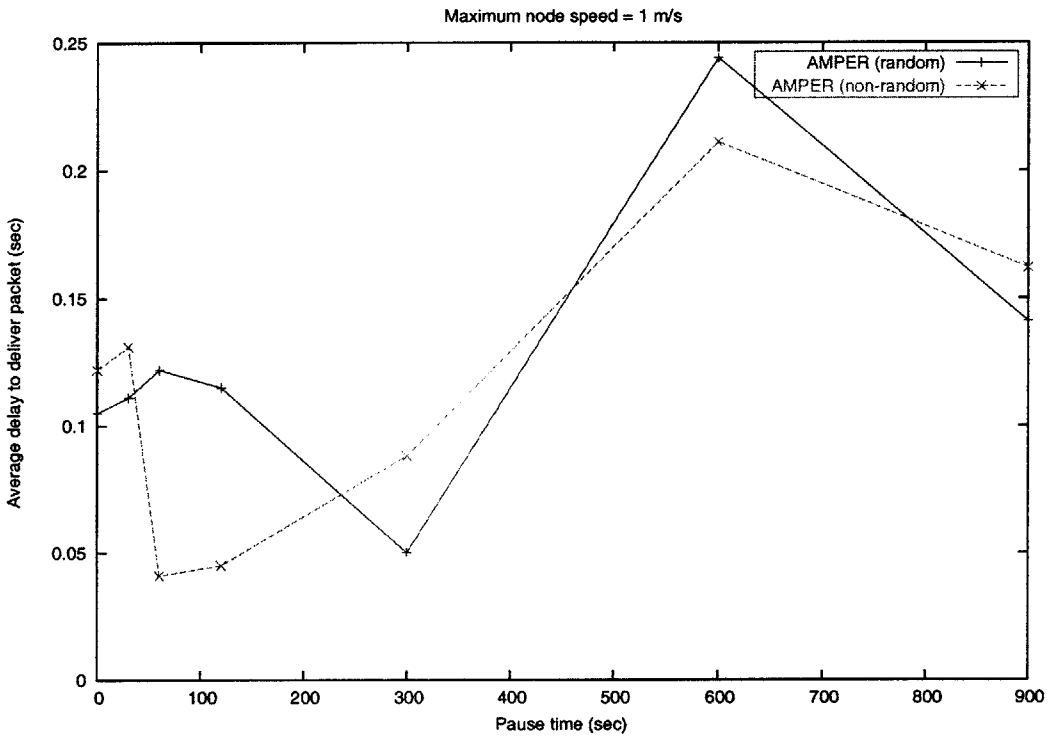

Figure 5-24: AMPER's Delivery Latency with Slow Nodes

5.5.4 Routing Overhead

Routing Packet Overhead

Figure 5-27 shows how all four protocols benefit from the decrease to 1 m/s of the maximum node speed. AMPER still sends the lowest number of routing packets regardless of the node mobility rate. DSDV continues to have an approximately constant overhead, due its periodic nature. DSR and AODV are both on-demand protocols, which is why their overhead drops as the mobility rate drops.

When run in non-random mode, AMPER produces nearly the same graph as in the random case (Figure 5-28). The best combination is to use random mode when the pause time is less than 300 seconds, and non-random mode otherwise.

Total Bytes of Overhead

When the routing overhead is measured in bytes and includes the size of the protocol header that AMPER places in each packet, our protocol performs quite poorly com-

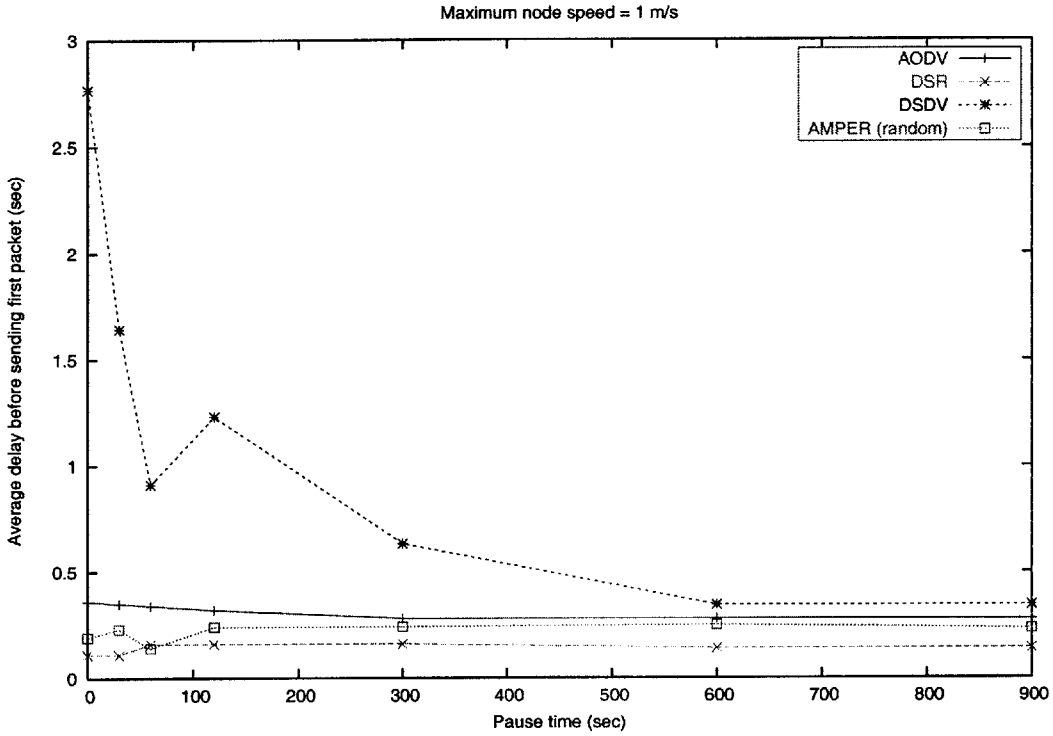

Figure 5-25: Discovery Latency Overview with Slow Nodes

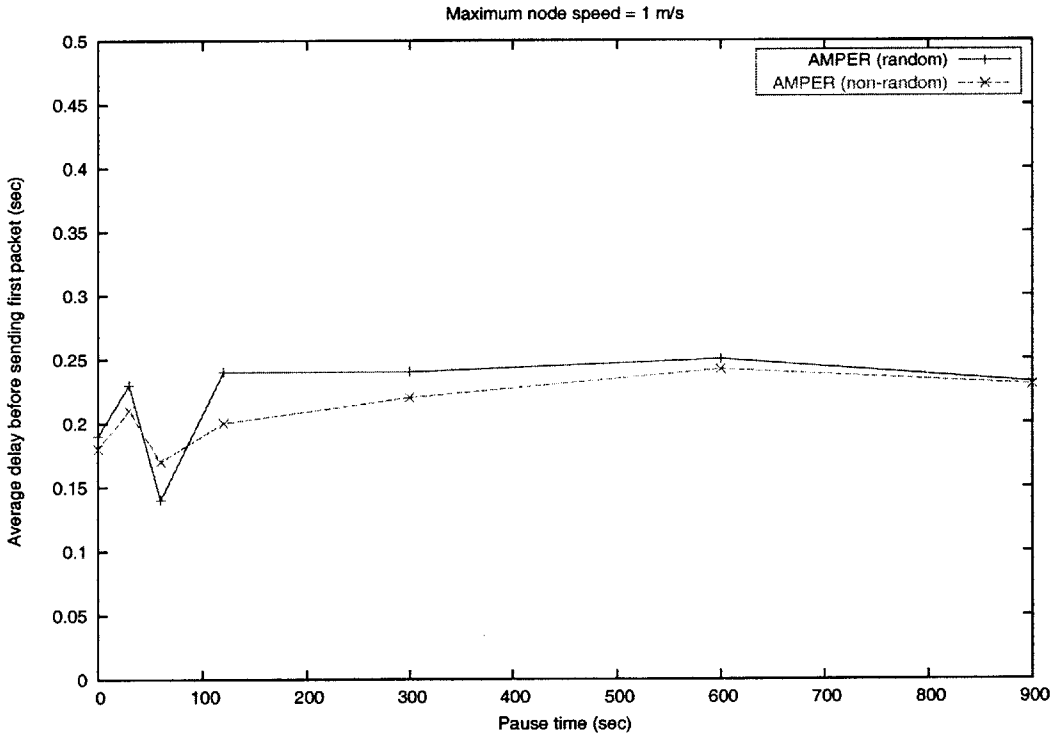

Figure 5-26: AMPER's Discovery Latency with Slow Nodes

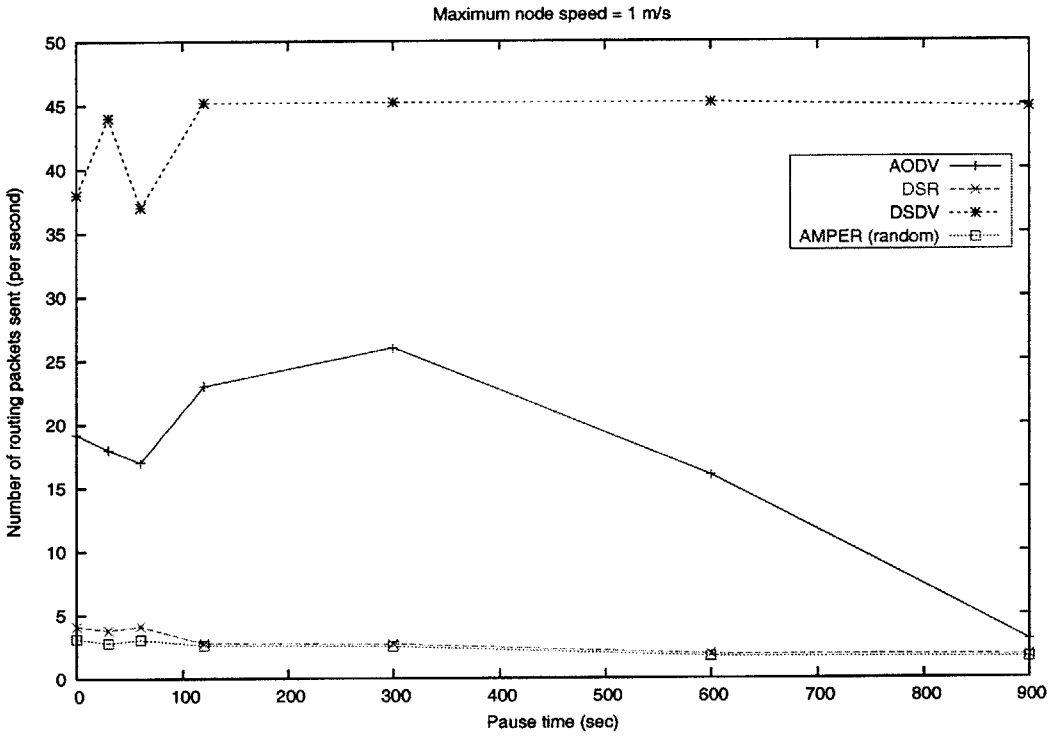

Figure 5-27: Routing Packet Overhead Overview with Slow Nodes

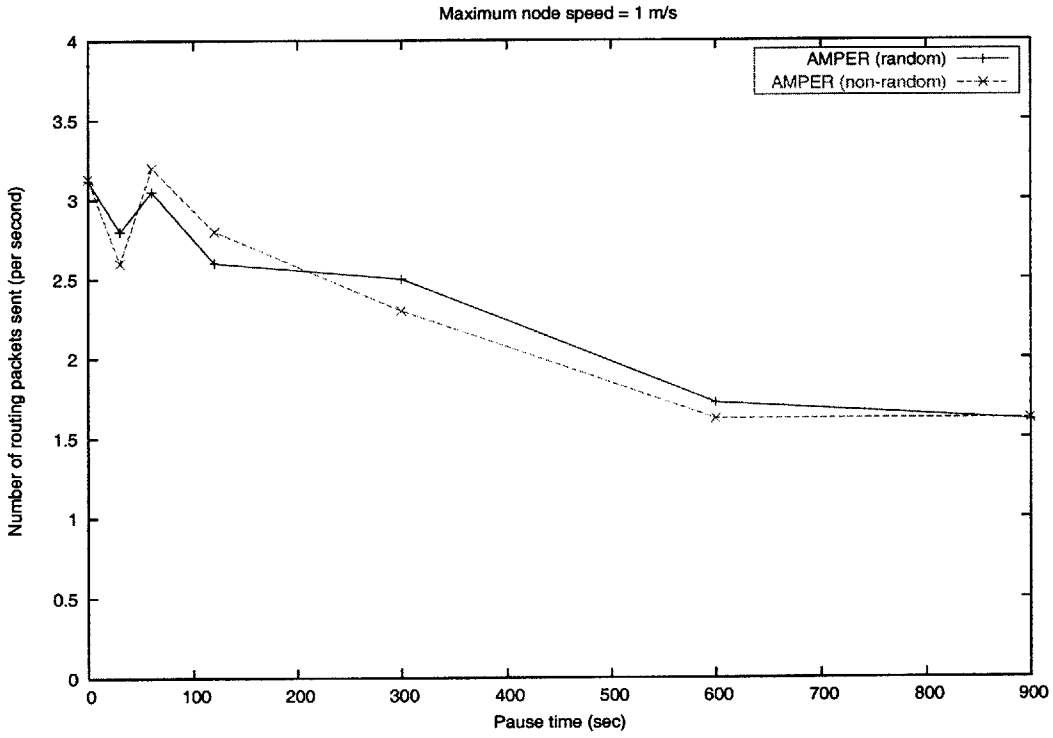

Figure 5-28: AMPER's Routing Packet Overhead with Slow Nodes

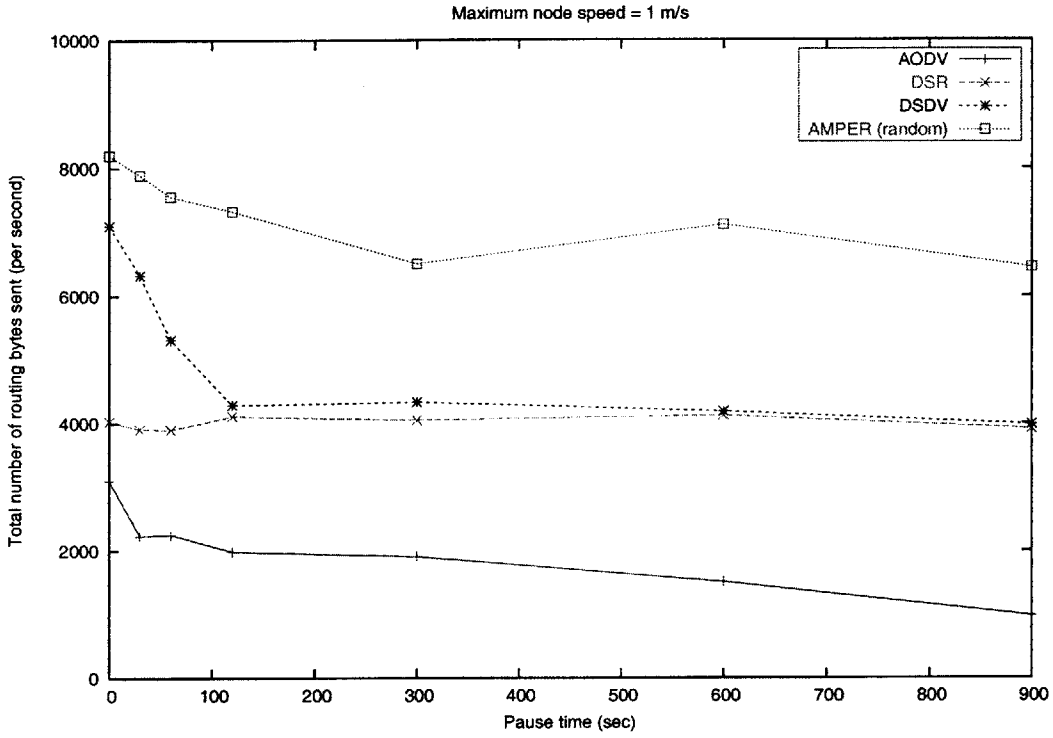

Figure 5-29: Total Overhead Overview with Slow Nodes

pared to the others. Although all four protocols' performance improves, AMPER's competitors benefit more from the nodes slowing down; the savings on packet transmissions cannot hide AMPER's header size anymore. AODV is still the leader in this department, followed by DSR and DSDV.

Figure 5-30 shows that the total overhead incurred by AMPER does not really depend on the forwarding scheme it uses, though random AMPER produces a slighly better performance for intermediate pause times.

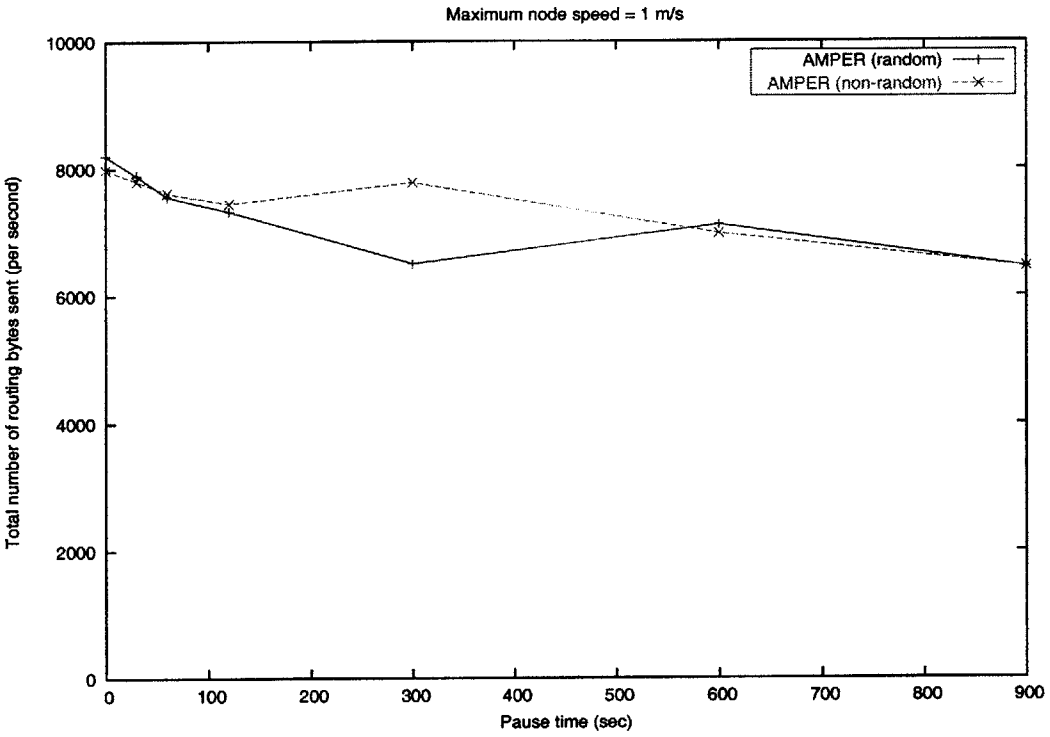

Figure 5-30: AMPER's Total Overhead with Slow Nodes

Chapter 6

Discussion

This chapter discusses some of strengths and weaknesses of AMPER. It is intended to be both a recapitulation of previous notions and a foundation for future work.

6.1 AMPER's Modularity

One of the greatest strengths of AMPER is its flexibility and modularity: although our simulations reflect specific design choices, our framework is built in such a way that switching to other schemes can be done with very little effort. Another merit of modularity is that it allows programmers to work on a certain module while keeping the others unaltered, which is a very useful feature for debugging and optimization purposes. AMPER's modularity features are illustrated in Figure 6-1.

6.1.1 Random and Non-Random Probabilistic Routing

AMPER is a pure probabilistic routing protocol: the routing table at every node contains probability values that dictate the likelihood of sending a data packet to a given destination through a specific neighbor. Two probabilistic routing schemes were exposed in Section 2.1: random and non-random. AMPER's modular framework allows the use of both: given a row of probabilities, a certain function returns the chosen neighbor when random routing is used, while another deals with the non-

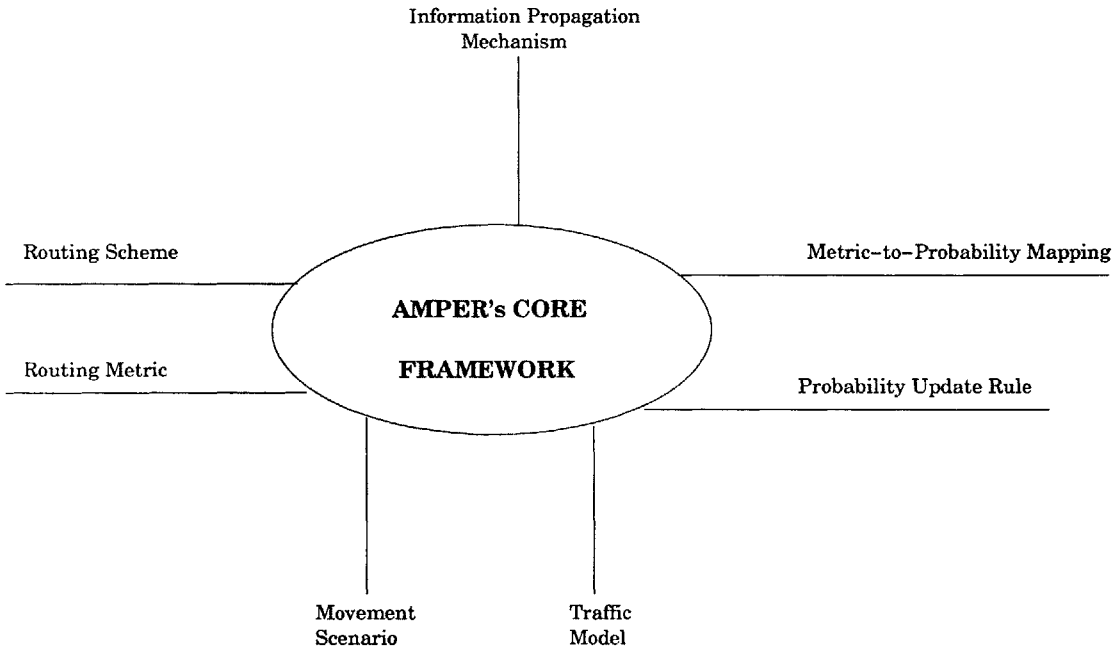

Figure 6-1: Illustration of AMPER's Modularity

random case; a boolean is required to specify the desired scheme (0 = random, 1 = non-random).

6.1.2 Information Propagation Mechanism

As explained in Paragraph 3.9.2, we use the expanding ring search mechanism to propagate information requests: the ring size is incremented for every retry and decremented for every forward. When the ring size reaches zero, the request is only forwarded piggybacked on data packets, to avoid generating explicit control packets for it. We designed our code in a modular fashion that allows the modification of the information propagation mechanism without affecting the other aspects of the protocol. Should need arise, a node could alternatively request information from its immediate neighbors only[1], or send an expanding request after a specific number of unsuccessful non-propagating requests.

[1]this is known as a non-propagating request.

6.1.3 Routing Metric

We picked the expected number of transmissions [5] as a routing metric for the reasons outlined in Paragraph 3.10.1. The ETX metric is attractive because it can be composed, which means that the total ETX from source to destination is the sum of the ETX values along individual links. Moreover, ETX accounts for asymmetric link loss ratios and solves the problem of interference between successive hops on multi-hop paths. AMPER's modular design allows the use of other routing metrics; the only condition is that the chosen metric be local enough to be computed from the metric values of a given node's neighbors, the link loss ratios to them, and the probability distribution for sending packets to them. Some other routing metrics verifying the locality criterion are the path cost, the communication cost, the end-to-end delay, the product of per-link delivery ratios, and the end-to-end throughput.

6.1.4 Metric-to-Probability Mapping

As outlined in Paragraph 3.10.2, we choose a static mapping that translates directed ETX values into probabilities[2] and heavily favors neighbors promising a lower workload to deliver a packet. As with the routing scheme and the routing metric, this mapping is also modular, in the sense it can be modified without affecting the general protocol structure: employing a dedicated function allows the selection of another static or dynamic mapping scheme. We should note that a dynamic mapping – although feasible – requires much more effort because it needs to analyze history information and operate on previous ETX values, which is quite involved in a network with frequent state updates.

6.1.5 Probability Distribution Update Rules

The probability distribution update rules dictate how the probability values in a node's routing table change when a link breaks, a new neighbor is in range (or out of range), a new destination is discovered (or is not reachable anymore), or when

[2]This mapping is consistent across all nodes.

another type of state variation occurs. This involves rebalancing the probability values so that the entries in a row sum up to one and every value in a row is above a predefined minimum threshold. AMPER's modularity makes it possible to modify the probability reshuffling rules without affecting any other component of the framework.

6.1.6 Movement and Traffic Models

The movement and traffic models outlined in Sections 4.3 and 4.4 are completely separate from the other aspects of AMPER; as a matter of fact, they are even independent of the ad hoc routing protocol under investigation. It is possible to change the number of nodes, the topology, the movement pattern, the pause time and the maximum node speed in a scenario file, and run a different simulation on the fly. In a traffic file, we have the flexibility of choosing the transport protocol[3], the number of connections, the packet size and the sending rate. Our simulations are carried out using constant-bit-rate UDP sources. Although constant-bit-rate UDP may not be the most realistic traffic model one could employ, we adopt it for several reasons:

- It is the most widely accepted model among the networking community experts, and is used by the authors of DSDV [11], DSR [9] and AODV [12] to simulate their protocols.

- Due to its conforming load property, TCP changes the times at which it sends packets based on its perception of the network's ability to carry packets. UDP does not possess this peculiarity, and hence can produce accurate relative trends when comparison is needed between different ad hoc routing protocols.

- TCP source nodes generate more traffic when paused at waypoint. UDP sources generate a constant (and therefore predictable) amount of traffic.

[3]or protocols, if a more sophisticated model is desired.

6.2 Header Size Limitations

In Chapter 3, we gave exhaustive details about the AMPER header contents and functionality, but omitted the analysis of its size characteristics.

The header implementation of AMPER makes use of a buffer size B that dictates the length of the arrays of requested destinations and fulfilled requests. Both arrays wrap around when their number of elements increases beyond B; for instance, if $B = 5$ and a node **n** requests a sixth destination, then this last request will overwrite the first. Clearly, the total header size depends on the choice of the parameter B. Moreover, given the fact that a node **n** must calculate the loss ratios to each of its neighbors, the header size will depend as well on the number of neighbors of **n** and, more generally, on the total number of nodes in the network, N.

The total header size S (in bytes) is related to B and N by the following equation:

$$S = 8 + 4 \times \lceil \frac{5N}{16} \rceil + 9B \tag{6.1}$$

A 3-dimensional plot of the header size versus the number of nodes and the buffer size is shown in Figure 6-2. Clearly, the plot has an increasing trend: the header size jumps from 8 bytes (for $B = N = 0$) to 1530 bytes (for $B = 30$ and $N = 1000$). As can be predicted, a large header will result in lengthier computations and degraded performance.

Let us now examine the header size variation with B when the node population is fixed. Figures 6-3 and 6-4 are plots of the header size versus the buffer size when $N = 50$ and $N = 1000$, respectively. The maximum value that B can reach is simply N, and this will theoretically occur when a node is allowed to request all destinations without overwriting a previous request. Both figures show a dramatic increase of the header size (with a slope of 9).

As programmers, we do not have control over the number of nodes in the ad hoc network, which suggests we should try to limit the buffer size, and if possible, choose a specific value of B that will guarantee optimal performance. At first glance, a large buffer size might seem to favor AMPER's correctness and prevent data packet losses.

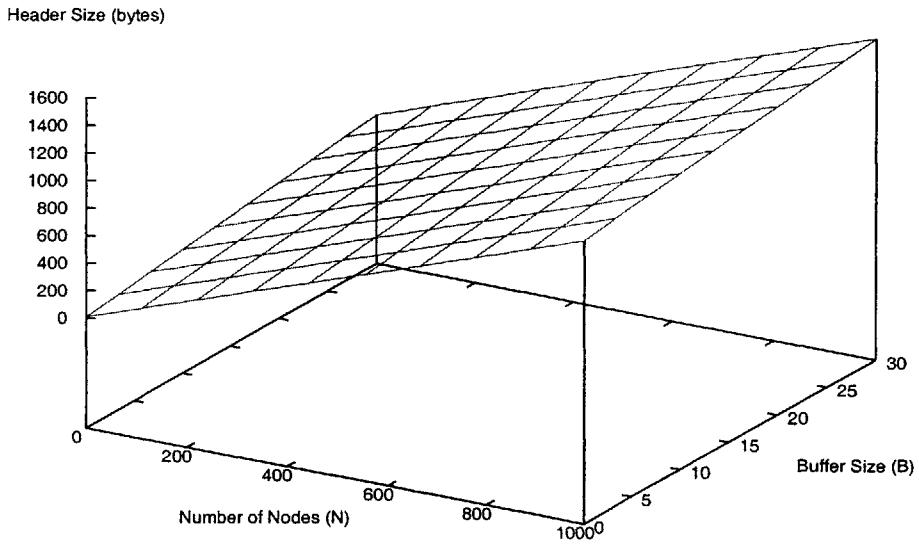

Figure 6-2: Header Size Versus the Number of Nodes and the Buffer Size

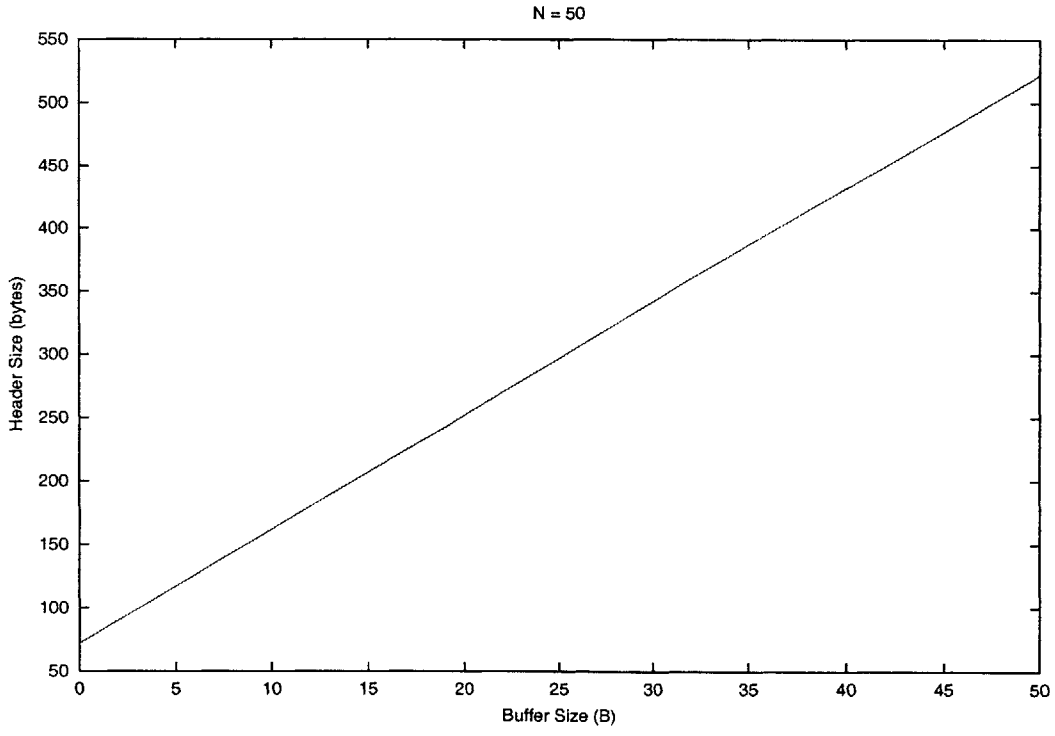

Figure 6-3: Header Size Versus the Buffer Size when N = 50

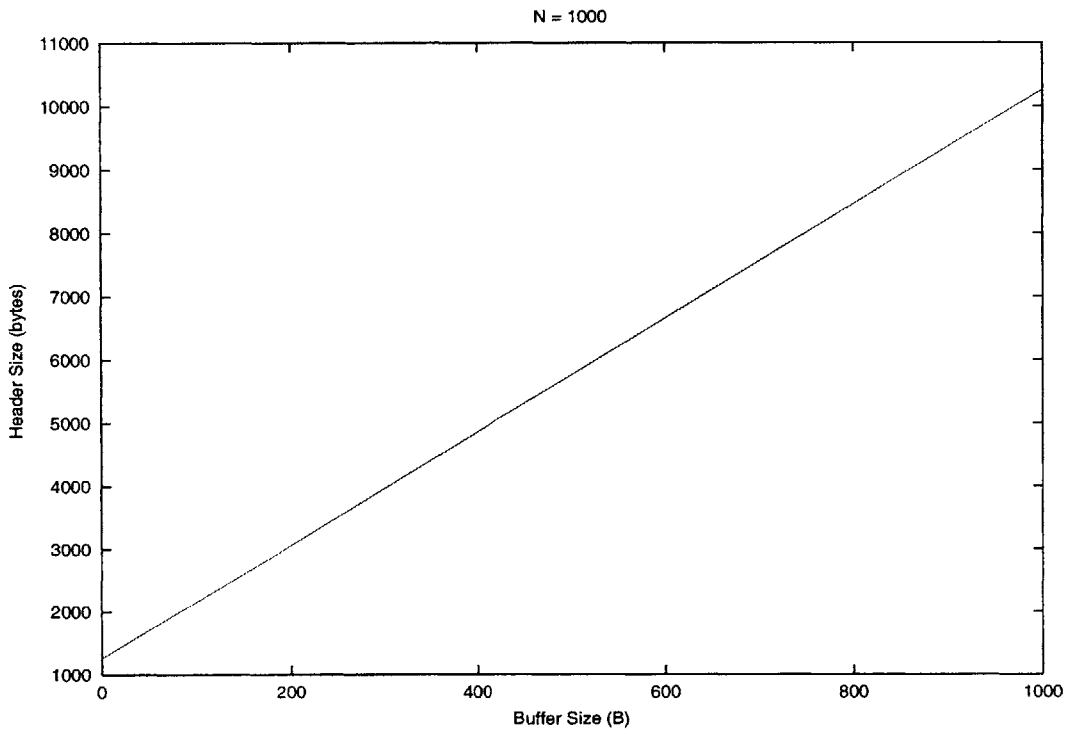

Figure 6-4: Header Size Versus the Buffer Size when N = 1000

This turns out not to be the case, due to the way AMPER timers work: an old array entry will almost always be erased before the corresponding request is serviced. Consequently, a large buffer size will unnecessarily increase the header size without delivering additional performance.

A few simulations reveal, by trial and error, that the optimal value of the buffer size is approximately $B = 0.1N$, which changes Equation 6.1 to the following:

$$S = 8 + 4 \times \lceil \frac{5N}{16} \rceil + 0.9N \tag{6.2}$$

For this optimal value of B, we plot in Figure 6-5 the variation of the header size with the number of nodes in the ad hoc network. This new plot highlights the limitations of our protocol in large networks. All of our simulations were carried out for a node population of 50, because – as mentioned in Section 2.6 – our design choice was to optimize AMPER for small networks. Since the largest packet size that can be accepted at the radio layer is approximately 23000 bytes, the maximum allowable

network size turns out to be approximately $N = 10000$ nodes.

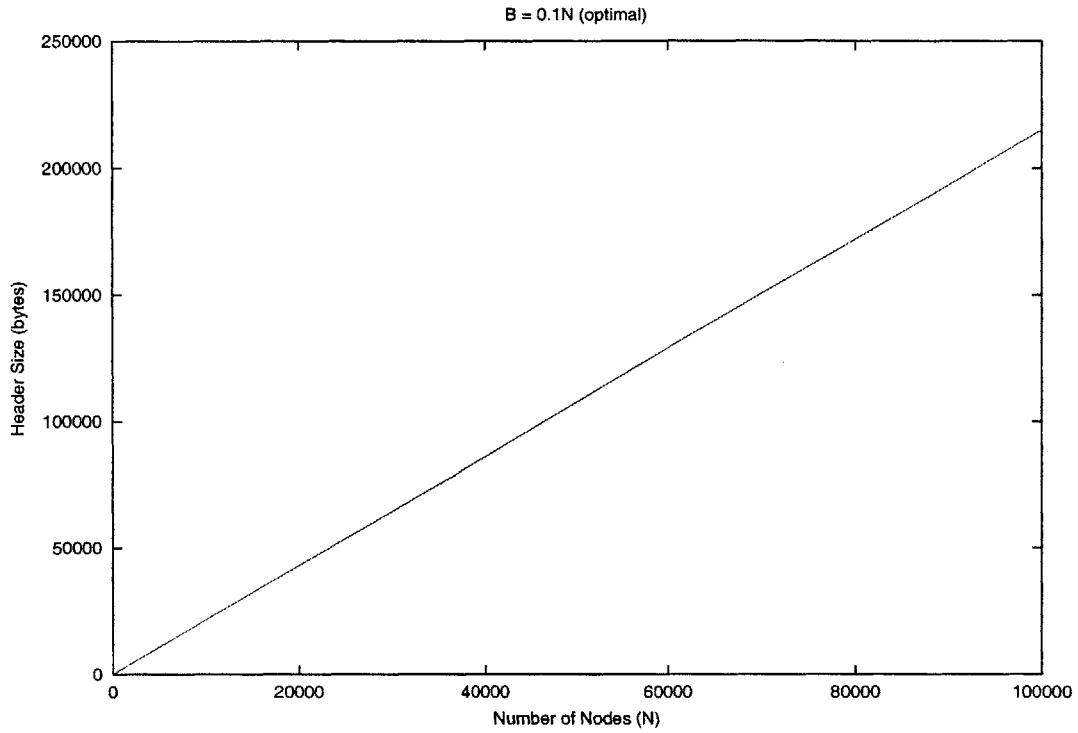

Figure 6-5: Header Size Versus the Number of Nodes when B = 0.1N

Our objective was not to build a protocol that would perform optimally for any network size; none of the protocols we surveyed achieves this utopic goal anyway. We rather wanted our protocol to be suitable for a specific type of applications, and our simulation results prove indeed that AMPER delivers high performance in small-sized networks.

6.3 Amount of State Maintained

As mentioned in Chapter 3, every AMPER node **n** has a routing table containing **n**'s reachable destinations in its rows, **n**'s neighbors in its columns, and a probability value corresponding to every destination-neighbor pair. In theory, **n** could be in range of every other node and could reach all other nodes in the network [4]. The maximum routing table size is therefore $N \times N$, where N is the number of nodes in the ad hoc

[4]including itself, of course.

network. In addition, we have seen that every AMPER node maintains loss ratio state (refer to Section 3.5); if a node **n** is capable of sending x packets within the link loss ratio timeout t_l, then the amount of loss ratio state it has to maintain is Nx. Consequently, the maximum amount of state, M, that may have to be maintained at a given node is:

$$M = N^2 + Nx \tag{6.3}$$

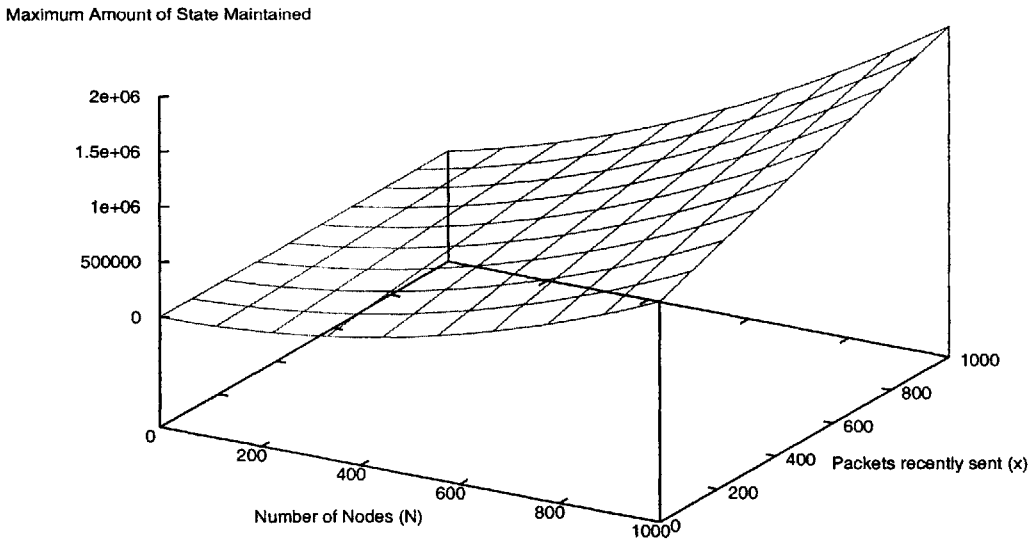

Figure 6-6: Amount of State Maintained versus N and x

The plot of M versus N and x is shown in figure 6-6 and is self-explanatory: the maximum amount of state a node **n** may have to maintain increases tremendously with the size of the network and the number of packets **n** can send within the link loss ratio timeout t_l. In particular, if x exceeds N^2, the amount of state maintained becomes at least $O(N^3)$!

Having no control over the number of nodes in the ad hoc network, we try to limit the value of x so that good performance is achieved within an $O(n^2)$ amount of state.

For this purpose, we create a safety margin by forcing the condition $x < N$. By trial and error, we find that a value of $t_l = N/25$ delivers optimal performance, so we set the rate at which a node sends a packet to $R = x/t_l = 25x/N$, which means that the maximum allowable rate R is 25 packets/s (since $x < N$).

The amount of state maintained at a node does not result in a degraded performance. On the contrary, a node storing considerable state information needs less internode communication in order to send packets to desired destinations. The only drawback is that simulations take much longer to run, mainly because the state update functions will require substantial time to execute. However, it is noteworthy that Equation 6.3 is an expression of the maximum amount of state that a node may need to store; in usual circumstances, this value is considerably lower because the routing tables are not entirely populated.

6.4 Loss Ratios and Link-Layer Retransmissions

The loss ratio calculation mechanism described in Paragraph 3.8.2 overlooks the fact that link-layer retransmissions may hide some losses: a packet that is resent M times appears as received, but the cost incurred by this operation is effectively Mc, where c is the cost of sending the packet once. When making routing decisions, we have assumed that the loss ratio is a good indicator of the quality of a link; this is very true most of the time, but unfortunately not sufficient. To take link-layer retransmissions into account, we propose one of the following approaches:

- Replace the link loss ratio notion by a link quality function. This practically requires reshaping the entire protocol framework including the node state, the header contents, the routing metric characteristics, and the way state updates are conducted.

- Change the loss ratio computations in a way that accounts for link-layer retransmissions, and use the "normalized" loss ratios to calculate the routing metric values as before. For instance, if a node $\mathbf{p}$ sends a packets to a node $\mathbf{q}$ within

the link loss ratio timeout t_l, $\mathbf{q}$ gets b packets, and there are c link-layer retransmissions, a possible formula for the "normalized" loss ratio from $\mathbf{p}$ to $\mathbf{q}$ is:

$$L = \frac{b-a}{a} \times (1 + \frac{c}{d}) \tag{6.4}$$

where d is an integer to be chosen. Using the above example with $a = 10$, $b = 8$, $c = 3$ and $d = 10$, we get a loss ratio of 0.26 (instead of 0.2 when neglecting link-layer retransmissions). The advantage of this approach is that it does not require any modifications to the routing metric formula.

- Keep the loss ratio calculations unchanged, but modify the formula used to obtain the routing metric values[5]. This approach is dependent on the routing metric chosen by the user.

6.5 Control Packets

AMPER uses nearly no control packets; the only case in which it does is when a node wants to acquire or disseminate information but has no packets to send on which to piggyback the request or answer. Such an explicit routing packet is broadcast, and contains all the usual header data, including the request or answer in question. We have found it very convenient (implementation-wise) to use a similar format for data and control packets; the number of bytes we could have saved by using a reduced control packet size instead is small, but this option might have been a possible optimization.

6.6 Energy Consumption and Multicast Routing

Probably the most serious drawback of AMPER is its energy consumption. This is mainly due to the enabling of promiscuous receive mode on the nodes' wireless

[5]for instance, by incorporating a multiplicative factor.

interface hardware, but also to the piggybacking of internode communication on data packets. The predicament is that disabling promiscuous mode would preclude efficient collaborative learning between nodes, whereas avoiding piggybacking would make AMPER's performance similar to DSR's [9].

To mitigate the energy consumption problem, one may unicast control packets instead of broadcasting them; this would be energy-efficient and compatible with sleep mode. Another possibility is to keep broadcasting control packets but to multicast data packets, which would be more energy-efficient than unicasting routing packets.

Multicasting may be achieved by a controlled flooding of data packets to all nodes in the ad hoc network that are within some specified number of hops of the originator; these nodes may then apply destination address filtering[6] to limit a given packet to those nodes subscribed to the packet's indicated multicast destination address. While this mechanism does not support pruning the broadcast tree to conserve network resources, it can be used to distribute information to all nodes subscribed to the destination multicast address. This mechanism may also be useful for sending application level packets to all nodes in a limited range around the sender.

Although desirable, we do not believe that the modularization of the data propagation scheme is feasible [7]; the functions that govern the dissemination of information are scattered throughout the code and tightly intertwined with other blocks.

[6] e.g., in software.

[7] at least using the current framework.

Chapter 7

Conclusion and Future Work

This thesis has presented AMPER, a novel probabilistic protocol for mobile ad hoc networks. AMPER sends a minimal number of routing packets, and replaces them with smart piggybacking; collaborative learning is achieved by having the nodes run in promiscuous mode and ovehear each other's packets and packet headers. Probabilistic routing is used to reduce the latency for recovery from link failures, and routing based on ETX[1] is intended to overcome the degraded throughput that minimum-hop-count suffers from when picking marginal links. One of the most important features of AMPER is its modularity: one can modify the routing scheme (random or non-random), the routing metric (e.g ETX), the metric-to-probability mapping, the probability update rule, the information propagation mechanism, the movement scenario and the traffic model without affecting the overall structure of the protocol.

Our extensive simulations demonstrate that AMPER delivers excellent performance and competes with DSDV, AODV and DSR; AMPER even outperforms its competitors when nodes are running at high speeds. Regardless of node speed, AMPER sends the least amount of routing information in both its random and non-random modes. However, the size of the AMPER header clearly degrades its performance with respect to its competitors, especially when the nodes operate at a maximum speed of 1 m/s. AMPER's random and non-random modes behave very similarly, though performance seems slightly better when the protocol runs in random

[1]ETX is one possible choice of routing metric; it is not a core component of AMPER.

mode.

In the near future, we intend to improve the computational efficiency of AMPER, so that it may be possible to simulate it within a reasonable amount of time[2]. We will also try to shrink the header size, in order to mitigate the performance degradation stemming from the excessive number of bytes of overhead. Moreover, we will try to reduce the protocol's energy consumption, and to come up with a loss ratio calculation approach that accounts for link-layer retransmissions. Finally, we would like to take full advantage of AMPER's modularity, by testing it on other metrics, models and probability mappings and drawing a comparison between them. Our ultimate objective is to deploy AMPER on a testbed and see how it behaves in realistic settings.

[2]as our simulations ran for literally thousands of CPU hours.

Bibliography

[1] John S. Baras and Harsh Mehta. A probabilistic emergent routing algorithm for mobile ad hoc networks. In *Proceedings of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt03)*, INRIA Sophia-Antipolis, France, March 2003.

[2] 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 4th annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'98)*, pages 85–97, Dallas, Texas, 1998.

[3] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[4] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification.* The Institute of Electrical and Electronics Engineers, New York, NY, 1997.

[5] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom'03)*, San Diego, California, September 2003.

[6] Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers, and Robert Morris. Performance of multihop wireless networks: shortest path is not enough.

In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, October 2002. ACM SIGCOMM.

[7] Kevin Fall and Kannan Varadhan. *The ns Manual.* The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC. http://www.isi.edu/nsnam/ns/.

[8] Mesut Güneş, Udo Sorges, and Imed Bouazizi. ARA: the ant-colony based routing algorithm for manets. In *International Conference on Parallel Processing Workshops (ICPPW'02)*, Vancouver, B.C., Canada, August 2001.

[9] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[10] Ioannis N. Kassabalidis, Mohamed A. El-Sharkawi, Robert J. Marks II, Payman Arabshahi, and Andrew A. Gray. Adaptive-sdr: Adaptive swarm-based distributed routing. In *IEEE WCCI 2002, IJCNN 2002 Special Session: Intelligent Signal Processing for Wireless Communications*, Honolulu, Hawaii, May 2002.

[11] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[12] Charles Perkins, Elizabeth M. Royer, and Samir R. Das. Ad hoc on-demand distance vector (AODV) routing. Internet-draft, IETF, February 2003. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-13.txt.

[13] Sundaram Rajagopalan, Chaiporn Jaikaeo, and Chien-Chung Shen. Unicast routing for mobile ad hoc networks with swarm intelligence. Technical Report 2003-07, University of Delaware, May 2003.

[14] The Rice Monarch Project. Wireless and mobility extensions to *ns*-2. http://www.monarch.cs.rice.edu/cmu-ns.html.

[15] Martin Roth and Scott Wicker. Termite: Emergent ad-hoc networking. In *Proceedings of the Second Mediterranean Workshop on Ad-Hoc Networking (Med-Hoc Net)*, Mahdia, Tunisia, June 2003.

[16] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996.