DESIGN AND ANALYSIS OF ADAPTIVE ROUTING AND TRANSPORT PROTOCOLS IN MOBILE AD HOC NETWORKS

APPROVED BY SUPERVISING COMMITTEE:

Dr. Rajendra V. Boppana, Supervising Professor

Dr. Thomas Bylander

Dr. Turgay Korkmaz

Dr. Kay A. Robbins

Dr. Parimal Patel, Outside Member

Accepted:

Dean of Graduate Studies

Dedication

This dissertation is dedicated to my parents, Wilson L. Dyer, Sr. and Jane H. Dyer, who taught me the value of education and of hard work and perseverance; and to my wife, Susie Dyer, without whose love, support, and encouragement this work could not have been completed.

DESIGN AND ANALYSIS OF ADAPTIVE ROUTING AND TRANSPORT PROTOCOLS IN MOBILE AD HOC NETWORKS

by

THOMAS DAVID DYER, M.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
December 2002

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Rajendra V. Boppana, for his guidance, encouragement, and patience. Working with Dr. Boppana has been a very rewarding experience, and I am truly indebted to him. I would also like to thank Dr. Kay Robbins, Dr. Thomas Bylander, Dr. Turgay Korkmaz, and Dr. Parimal Patel for serving on my final Ph.D. committee. Their criticism and suggestions have been invaluable.

Finally, I would like to acknowledge the support of my employer, the Southwest Foundation for Biomedical Research. I am especially grateful for the encouragement and support I have received from Dr. John Blangero, Dr. Sarah Williams-Blangero, Dr. Jean MacCluer, and Dr. Bennett Dyke.

December 2002

DESIGN AND ANALYSIS OF ADAPTIVE ROUTING AND TRANSPORT PROTOCOLS IN MOBILE AD HOC NETWORKS

Thomas David Dyer, Ph.D.
The University of Texas at San Antonio, 2002

Supervising Professor: Dr. Rajendra V. Boppana

A mobile ad hoc network (MANET) is a collection of mobile computing devices that communicate using wireless links, forming a multihop network without the use of network infrastructure or centralized administration. In this dissertation, we examine routing and transport protocol performance in MANETs and we explore ways of improving that performance.

It is well known that TCP performance in MANETs suffers from TCP's inability to distinguish packet losses due to congestion from losses caused by mobility-induced route failure. We propose a heuristic in which the TCP sender interprets multiple timeouts for the same packet as an indication that the route to the receiver is broken. Rather than doubling the retransmit timeout interval (RTO) on each consecutive timeout, the sender fixes the RTO. The increased rate of packet retransmissions stimulates route repair, reducing the time taken to repair the broken route and re-establish the flow of TCP packets.

UDP performance in MANETs is also problematic. Using simulations, we show that rapid topology changes prevent UDP from fully utilizing available network capacity. To address this issue, we propose an adaptive unreliable packet delivery service, the Adaptive Datagram Protocol. ADP uses acknowledgements from the receiver to clock the transmission of new packets. The sender buffers packets when the application's sending rate exceeds network capacity. If the buffer becomes full, the sender drops excess packets rather than injecting them into the network where they may cause contention. These features enable ADP to attain higher throughputs and use network resources more efficiently than UDP.

We have conducted extensive simulations over a wide range of network loads to test the applicability of our transport protocol proposals to three MANET routing protocols. The network loads were varied to include FTP file transfers, variable-bit-rate video streams, and HTTP traffic. Our results demonstrate that the fixed-RTO heuristic significantly improves TCP performance for routing protocols which respond well to route stimulation, and in situations where route repairs are prolonged or difficult. Our results also show that, in addition to providing higher throughput and greater efficiency than UDP, ADP works well with TCP to achieve an equitable sharing of available bandwidth.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the use of the Internet and of wireless communications has risen dramatically. Surfing the World Wide Web and chatting on a cellular phone have become commonplace activities. With the advent of wireless networks and the availability of portable computers such as laptops and handheld devices, the day is coming in the not too distant future when applications that require networking can be used “anytime, anywhere” [52].

Wireless networks have been around since the late 1960s when the Aloha packet radio network was first developed [1]. In the last decade, wireless networks have been adapted to enable mobility, leading to their increasing popularity. The most common type of wireless network consists of mobile devices which communicate with base stations that are the gateways to a fixed network infrastructure. As a mobile unit moves about, its link to the wired portion of the network is handed off from one base station to another, thus providing seamless connectivity throughout the network.

Another type of wireless network, called a mobile ad hoc network (MANET), does not rely on fixed infrastructure. Rather, mobile units form a multi-hop wireless network with no fixed routers and no centralized administration. Network nodes function as routers, discovering and maintaining routes to other nodes in the network. Originally developed by the military for use in battlefield situations [37], MANETs are suited for applications ranging from emergency disaster relief to networking laptops in a conference room.

The effort to extend the Internet into the wireless realm is well under way. As the capabilities of mobile devices and the capacity of wireless communications increase, MANET users will want to run many of the same networking applications to which they have become accustomed on the wired Internet. They will

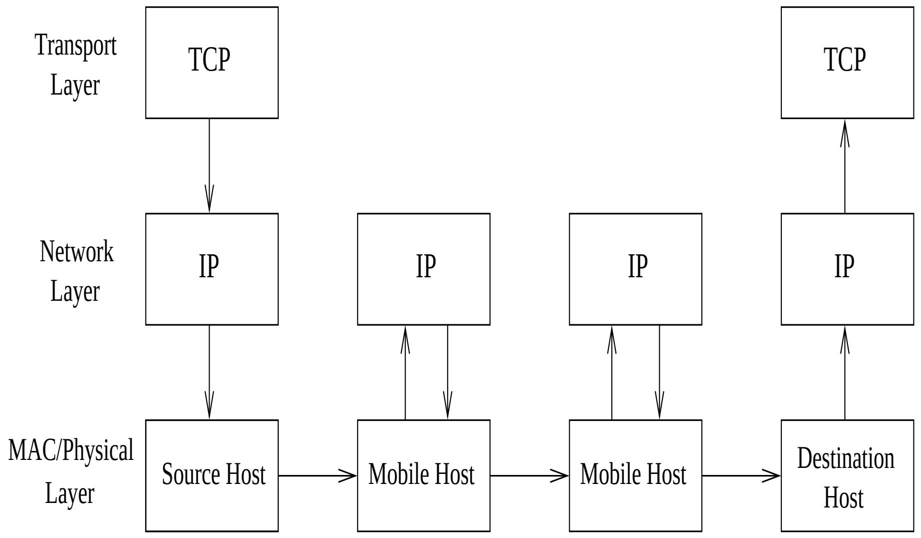


Figure 1.1: Protocol layers in a mobile ad hoc network. TCP = Transmission Control Protocol, IP = Internet Protocol.

expect seamless operation of everything from E-mail and Web browsing to Internet telephony and streaming video. The key to successfully meeting this demand will be the performance of Internet protocols over MANETs.

In the following sections, we give a brief description of some of the elements of Internet Protocol (IP) [56] networking in MANETs. IP networking is typically viewed as consisting of layers as illustrated in Figure 1.1. We start from the ground up, so to speak, beginning with the physical layer and how access to this layer is controlled in MANETs. Next we consider the network layer, the level at which IP is concerned with routing packets from one MANET node to another. Finally we discuss the transport layer, the level at which data can be delivered from an application running on a local host to another application running on a remote host elsewhere in the Internet. At each level, we note the work that has been carried out and touch upon some of the outstanding issues that are topics of ongoing research.

1.1 Wireless MAC layer

Wireless networks can be built using a variety of physical media, for example radio transmissions or infrared signals. Regardless of the physical media employed, network interfaces must follow a Medium Access Control (MAC) protocol. MAC protocols operate at the link layer, the networking layer responsible for transmitting data across a single, direct network link. We use the terms link layer and MAC layer interchangeably, although in a strict layerist view they are not identical.

Wireless links are lossy for a variety of reasons including path loss, fading, noise, and interference. Reliable link-layer protocols are used to hide this lossiness from the higher layers in the network protocol stack [9]. These protocols use forward error correction (FEC) or acknowledgement repeat request (ARQ) mechanisms to implement local error recovery. Hybrid protocols, such as AIRMAIL [4], employ both techniques.

For MANETs, the most commonly used MAC protocol is IEEE 802.11 with Distributed Coordination Function (DCF) [32], which is a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol. As in the Ethernet protocol for wired networks [31], 802.11 devices sense whether the medium is busy, waiting until the link is idle before transmitting. However, because wireless devices have a limited transmission range, not all nodes in a MANET will be in reach of each other. Therefore, a collision avoidance scheme is employed in which two nodes wishing to communicate must successfully complete an exchange of request-to-send (RTS) and clear-to-send (CTS) messages before any data is transmitted. Other nodes that hear this exchange will learn that the channel is going to be busy and for how long.

After a successful packet transmission, the receiving node sends an acknowledgement to the sender. If the sender does not receive this acknowledgement, it retransmits the packet several times before declaring the link to be broken. Thus, 802.11 is able to provide reliable packet transmission provided links are bidirectional. This scheme is used for unicast packet transmissions only. For broadcast transmissions, the wireless device simply waits for an idle channel and then transmits. Hence, broadcasts are not reliable.

1.2 MANET routing protocols

In an IP internetwork, packets are delivered from one network to another via routers. The routers learn about paths through the internetwork by means of IP routing protocols, such as RIP [27], OSPF [50], or BGP [60]. Although these routing protocols were designed to autonomously adapt to changes and failures in the network infrastructure, they are unable to cope with the rapid topology changes possible in a mobile ad hoc network. For this reason, an Internet Engineering Task Force (IETF) working group has been formed to develop a routing framework for IP protocols in MANETs [33].

A number of routing protocols have been proposed and are currently being considered by the MANET working group. These protocols may generally be characterized as *proactive* or *on-demand*. In proactive algorithms, routes are maintained to every node in a network, and routing information is disseminated in periodic routing updates [18, 54]. Because routes are immediately available, proactive protocols exhibit low packet latency. However, this comes at the price of the higher network overhead incurred by the propagation of routing updates, even when there is no current need for this information.

On-demand, or reactive, protocols, on the other hand, discover and maintain routes on an as-needed basis [36, 51, 53]. Because only actively used routes are maintained, the routing overhead may be less than that of a proactive protocol. However, the need for route discovery means that packet latency will increase, perhaps significantly. Hybrid protocols have also been proposed which combine on-demand and proactive approaches. For example, ZRP [26] adopts an approach in which MANETs are divided into zones. A proactive protocol is used for routing packets within a zone, while an on-demand protocol is employed for interzone routing.

A reasonable tactic in MANET routing may be to store multiple routes between a source and a destination. When a route breaks, an alternate route is immediately available. Care must be taken, however, to purge stale routing information, since the use of stale routes wastes network resources, e.g. bandwidth and battery power, and results in poor performance. Various route caching strategies have been studied [30, 48].

A number of performance evaluations of MANET routing protocols have been presented in the literature. Simulation-based performance comparisons have been made of various proactive and on-demand protocols [12, 13, 19, 34, 43]. Using simulation, these studies measured packet latency, throughput, packet delivery fraction, and routing overhead for UDP traffic generated by constant-bit-rate (CBR) sources. The offered traffic, number of nodes, and node speed were varied, and different mobility scenarios were considered.

The design of routing protocols for ad hoc networks is challenging. As pointed out in [52], ad hoc networks might be considered the "acid test" of network protocol design. Potentially, an ad hoc routing

Table 1.1: Expected throughput for 1 TCP connection as a function of the number of hops from sender to receiver. TCP packet size = 1460 bytes.

Number of hops	**Throughput (Mbps)**
1	1.463
2	0.729
3	0.4844
4	0.3399
5	0.2464
6	0.2052
7	0.1981
8	0.1918
9	0.1853
10	0.1824

protocol design might prove to be more scalable than the protocols currently in use in the global Internet. For these reasons, routing protocols for ad hoc networks will continue to be of considerable interest.

1.3 Transport layer protocols

In the Internet, TCP [57] is the de facto standard protocol for reliable packet delivery. Bulk data transfers via FTP, interactive applications like Telnet, and HTTP for Web traffic, all utilize TCP at the transport layer. Therefore, TCP performance is a major determinant of how well these applications perform in wireless networks. At the same time, the use of multimedia and real-time applications requires that MANETs provide good UDP [55] performance. With a realistic mix of applications, TCP connections and UDP flows can both be expected to compete for MANET resources.

In a MANET, the maximum throughput than can be achieved for a transport-layer connection, in the absence of contention from competing traffic, depends on the number of routers (mobile hosts) that must be traversed between sender and receiver. Each link traversal from router to router is called a hop. Table 1.1 shows the expected throughput for a TCP connection as a function of the number of hops. These numbers are taken from a similar table in [29]. In Figure 1.2, we have used this data to estimate the capacity of a single transport-layer connection as it changes over time in a simulated MANET.

The seminal paper by Caceres and Iftode [15] demonstrated that networks that include wireless links

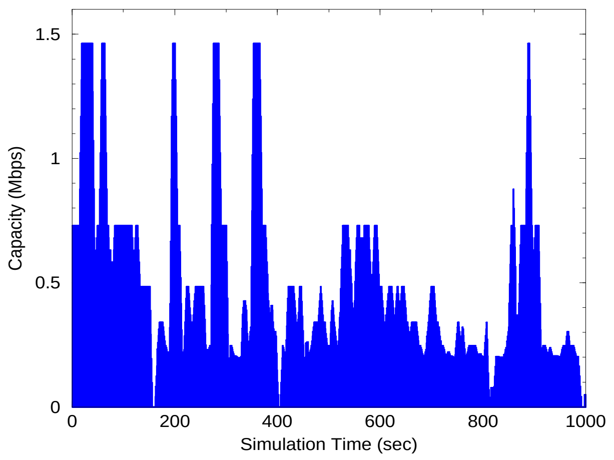


Figure 1.2: Capacity of a single transport-layer connection in a simulated ad hoc network. Estimated using achievable throughput (in the absence of contention) as a function of the number of hops in the shortest path from sender to receiver.

and mobile hosts suffer from delays and packet losses unrelated to network congestion. In MANETs, node mobility can cause routes to break, with the result that packets are delayed in buffers waiting for a new route, or are simply dropped. MANET routing protocols are designed to repair routes quickly, but it is essential that the TCP layer react appropriately to the interruptions caused by route failures. Whenever possible, unnecessary invocation of congestion control must be avoided, as these measures will have a negative impact on TCP performance. Various methods for dealing with this problem have been designed which involve explicit notification of the TCP sender when a link failure has been detected [9, 17, 29]. The sender can then take appropriate steps, such as freezing its state (timers and congestion window) until the route has been re-established. Biaz and Vaidya [11] proposed sender-based heuristics for distinguishing between packet losses due to congestion and losses due to wireless transmission errors. In addition, it has been shown that performance gains may be realized in a wireless environment by using TCP protocol enhancements such as selective acknowledgements [9].

As noted above, MANETs will have to simultaneously support both TCP and UDP traffic. More generally, with network applications sharing scarce MANET resources, quality of service (QoS) issues will grow in importance. Systems such as [42] will be needed to provide service differentiation in MANETs. A discussion of QoS issues in ad hoc wireless networks is given in [16].

A number of studies have analyzed the performance of the TCP protocol over 1-hop wireless links [6, 7, 8, 9, 14, 68]. Using experimental test beds and simulation, these studies measured the increases in TCP throughput that can be attained by using a variety of techniques including split connections, duplicate ACKs, and explicit loss notification. Recent studies have analyzed TCP performance in MANETs. Ahuja et al. [2] measured the TCP Tahoe throughput yielded by different routing protocols, including AODV [53] and DSR [36]. The TCP performance problems caused by route failures in an ad hoc network were addressed by [17, 29], but no comparison was made of different protocols.

To our knowledge, no previous study has presented a performance comparison of MANET routing protocols that includes network traffic from both UDP and TCP flows, nor have existing studies considered multiple TCP connections. Moreover, the published evaluations have not reported performance measurements other than throughput. For example, TCP connect time and routing overhead, both important performance indicators, have not been considered.

In a recent article by Macker et al. [46], it was pointed out that while routing UDP traffic is an important first step in providing Internet services over wireless networks, many applications require the end-to-end reliability and flow control offered by a transport-layer protocol such as TCP. According to the authors, providing transport layer services that are robust and functional in mobile wireless networks is an area of research that is still largely unexplored.

1.4 Contributions of this thesis

This work will make contributions to MANET research in the following areas: routing protocol design, improving TCP performance in MANETs, transport protocol design, and MANET performance analysis.

1.4.1 Routing protocol design

As we have seen, the design of MANET routing protocols is still an active area of research. Many researchers favor on-demand routing protocols, arguing that on-demand algorithms have lower overhead and thus will perform better than proactive protocols. Nevertheless, the proactive approach has the advantage of immediate route availability. Moreover, the routing updates of a proactive algorithm may enable broken routes to be repaired more quickly. It is natural to wonder if a routing method can be devised that successfully combines the best features of on-demand and proactive techniques.

Boppana and Konduru have proposed a new routing protocol, Adaptive Distance Vector (ADV), that is based on a proactive distance vector algorithm [12]. In ADV, however, routes are maintained for active connections only, and routing updates are triggered adaptively in response to varying network conditions. Thus, ADV's routing overhead varies with load and mobility, a characteristic of on-demand algorithms. In that sense, ADV can be considered to be a hybrid algorithm.

In simulations, ADV has been shown to outperform on-demand protocols for UDP traffic in high node mobility scenarios [12]. As part of our dissertation work, we have made some modifications to ADV to tune its TCP performance, and in performance analyses that include TCP as well as UDP traffic, we have shown that ADV performs as well or better than on-demand algorithms.

1.4.2 TCP performance

Improving TCP performance over MANETs is a major goal of our research. We have proposed a sender-based heuristic, called fixed RTO, that is designed to address the performance degradation caused by standard TCP's congestion control response to packet loss. In our method, the TCP retransmit timeout interval (RTO) is fixed when retransmit timeouts are interpreted as having been caused by route failures rather than congestion. As a result, an unnecessary exponential backoff of the RTO is avoided and packet retransmissions serve to periodically probe the network for a repaired route. This technique has been shown to significantly increase TCP throughput for on-demand routing protocols by stimulating their route discovery

We have also investigated the performance impact of a routing protocol's buffer refresh time. This is the maximum amount of time that a packet can remain in a routing-layer queue before it is dropped for lack of a route. We have shown that while a short buffer refresh time is beneficial in terms of packet latency for UDP flows, a longer time is better for TCP traffic. Therefore, we recommend that different buffer refresh times be applied to UDP and TCP packets.

1.4.3 Transport protocol design

An interesting observation that we have made is that TCP, despite the overhead of packet acknowledgements, is able to achieve significantly higher throughput than UDP, a fact that would seem to be at odds with what is observed in wired networks. This disparity is caused primarily by the relatively rapid topology changes in a MANET, particularly in the case of high node mobility. Available bandwidth can be highly variable as the length of the path between sender and receiver changes over the course of a connection. Since the standard UDP protocol does not conform to available network capacity, it will often be the case that a UDP flow is either underutilizing a short route with high bandwidth, or attempting to push too much traffic through a long route with low bandwidth, causing contention and wasting network resources on undelivered packets.

This observation led us to conclude that, in the MANET environment, it is important that a transport layer protocol be adaptive to network conditions in order to efficiently utilize network capacity. We have designed a transport protocol, the Adaptive Datagram Protocol (ADP), that employs a simple ACK-clocking scheme and buffering to provide an unreliable packet delivery service which adapts its flow rate to available network bandwidth.

1.4.4 Performance analysis

Another goal of our research has been to establish a more extensive set of transport layer performance measurements for MANETs. These performance assessments will help to elucidate strengths and weaknesses in existing transport layer protocols, and will serve as benchmarks against which new techniques and proposed protocol modifications can be compared. Of particular interest is the interaction of transport layer protocols, e.g. UDP and TCP. It cannot be assumed that the behavior of these protocols will be the same in MANETs as has been observed in wired networks.

To this end, we have conducted an extensive comparison of combined TCP and UDP performance for three proposed MANET routing protocols, AODV [53], DSR [36], and ADV [12]. Our simulation-based performance analyses have considered multiple transport layer (TCP and ADP) connections, with varying background network loads of UDP traffic. For TCP performance analysis, we have generated traffic from simulated Web (HTTP) connections as well as simulated FTP file transfers. In our analysis of ADP performance, we utilized a variable-bit-rate stream of packets to simulate the flow of frames in a streaming video connection.

1.5 Organization of the dissertation

The rest of this dissertation is organized as follows: Chapter 2 presents background material and discusses related work that has been done on MANET routing protocols and transport layer performance in wireless networks. Chapter 3 describes in detail the Adaptive Distance Vector routing protocol. Chapter 4 describes and evaluates the mechanisms we propose for improving TCP performance in MANETs. Chapter 5 introduces the new Adaptive Datagram Protocol and analyzes its performance for simulated video traffic. Chapter 6 evaluates the performance of our proposed protocol designs for a mix of simulated video and Web traffic. Chapter 7 summarizes the work done and presents our conclusions.

Chapter 2

Background

In this chapter we present background material relevant to our proposed research. First, we describe several of the routing protocols proposed by members of the MANET Working Group. We pay particular attention to the two on-demand protocols, DSR and AODV, which are subjects of the performance analyses presented later in this proposal. We then discuss the IP Transmission Control Protocol (TCP). After reviewing the pertinent details of how TCP operates, we show how the assumptions built into TCP are challenged by the nature of mobile wireless networks, and consider the ramifications for TCP performance in MANETs. We discuss several mechanisms that have been proposed in the literature to deal with these issues and to thereby improve TCP performance. Finally, we take a look at how performance analysis is conducted in MANET research.

2.1 Routing protocols

The design of routing protocols for use in mobile ad hoc networks is challenging and is still an open research area. The protocols that have served the wired Internet well do not translate to the MANET environment. In particular, the existing IP protocols were not designed to cope with the rapid changes in network topology that are possible with node mobility. Other factors, such as power management, should also be taken into account in a successful MANET routing protocol design.

The protocols that are being considered for use in MANETs are varied in their approaches. Although other ways of categorizing them are certainly possible, one way to group these techniques is by their approach to route creation and maintenance. Some protocols take a proactive approach, attempting to identify the best routes at any time between every pair of nodes in the network. That way, the routes will be available if and when they are needed. Other protocols take the point of view that maintaining a route that is unused is wasteful. These methods discover and maintain routes only as needed. Of course, no categorization is perfect, so we have included a third, catch-all group. The protocols in this group may have proactive or on-demand features or both, but they also have characteristics that set them apart from the first two categories. Our classification of routing protocols is summarized in Figure 2.1.

Table 2.1: Classification of MANET routing protocols.

Algorithm Type	**Protocol**
Proactive	Destination Sequenced Distance Vector (DSDV)
	Optimized Link State Routing (OLSR)
On-demand	Dynamic Source Routing (DSR)
	Ad hoc On-demand Distance Vector (AODV)
	Temporally Ordered Routing Algorithm (TORA)
Other	Zone Routing Protocol (ZRP)
	Core-Extraction Distributed Ad hoc Routing (CEDAR)
	Associativity Based Routing (ABR)
	Signal Stability based Adaptivity (SSA)
	Adaptive Distance Vector (ADV)

2.1.1 Proactive protocols

The aim of a proactive routing protocol is to maintain up-to-date routes from every node to every other node in the network. For each destination, a node knows which of its neighbors is the next step, or hop, along the shortest path to that destination. Routing a packet requires only a simple table lookup, hence these protocols are also called table-driven. Changes in network topology are propagated throughout the network in updates in order to maintain a consistent view of the network. Table-driven protocols can be categorized as either distance vector or link state.

Distance vector algorithms are so called because each node maintains, for each destination, the distance to that destination from each of the node's neighbors. The neighbor with the shortest entry in this vector of distances is chosen to be the next hop to the destination. Choosing next hops in this fashion results in the shortest path to any destination. A node derives the information in its distance vector via periodically broadcast updates from its neighbors. This method, the Distributed Bellman-Ford (DBF) algorithm [10], is computationally efficient and straightforward to implement. However, the DBF algorithm is subject to both short-lived and long-lived routing loops because nodes choose their next hops in a distributed manner using information that may be out-of-date. Nevertheless, the simplicity of DBF has made it an attractive choice for implementation, the Routing Information Protocol (RIP) [27] being a well-known example.

In a link state algorithm, each node monitors the status of its link with each of its neighbors. This information is shared periodically with the other nodes in the network. Thus each node acquires a complete description of the network topology, and can apply a shortest-path algorithm to choose its next hop for each destination. Due to propagation delays, the link state information at a particular node may be temporarily out-of-date, possibly resulting in loop formation. Such loops are short-lived, however, disappearing as routing updates traverse the network. Link state algorithms are more complex computationally and require more memory than distance vector algorithms, but they are not subject to the formation of long-lived loops. The Open Shortest Path First (OSPF) routing protocol [50] in the wired Internet is an example of a link state protocol.

Destination Sequenced Distance Vector - DSDV

The Destination Sequenced Distance Vector (DSDV) protocol [54] is based on the distance vector algorithm. DSDV avoids the looping problem of a traditional distance vector method by associating a sequence number with each routing table entry. This sequence number, originally assigned and advertised by the destination node, is used to determine the relative freshness of routing information.

Updates are broadcast periodically to maintain routing table consistency. For each routing table entry included in an update, a node increments the hop count (metric) by one since a node receiving the update will be one hop further from the destination. When a node receives an update, it will update the entries in its routing table for which the corresponding update entry has either a higher sequence number or the same sequence number but a lower metric. If a node does not receive three consecutive periodic updates from a neighbor, the link to this neighbor is considered to be broken. Alternatively, the MAC layer may detect link breakage and report this to the routing agent.

A node increments its own sequence number by two each time it broadcasts a routing update. By convention, these sequence numbers are even. If a node determines that the link to one of its neighbors is broken, it searches its routing table for routes that use the neighbor as the next hop. In any such entries, the metric is set to infinity and one is added to the sequence number. Odd sequence numbers, therefore, denote broken or invalid routes. The use of an odd number ensures that any newer update entry with a valid (not infinite) metric will have a greater sequence number.

To reduce the potentially large volume of network traffic produced by routing updates, DSDV uses two different types of update. Full updates are broadcast periodically and include every entry in the routing table. Smaller, incremental updates include only those routing entries that have changed since the last full update. Incremental updates are triggered when significant changes are made to the routing table. For instance, a route invalidation is considered sufficiently important to trigger an update. Nodes keep track of the weighted average time that routes to a destination fluctuate before an update with the best metric is received. The broadcast of a routing update is delayed by the length of this settling time, which further reduces network traffic by eliminating the broadcast of sub-optimal routes. Still, DSDV has been shown to have very high routing overhead compared to on-demand routing protocols [13]. While the number of routing packets transmitted per second will be smaller for DSDV, the large number of routing entries in each update packet accounts for the higher overhead.

Optimized Link State Routing Protocol - OLSR

The Optimized Link State Routing (OLSR) protocol [18] is an extension of the pure link state algorithm, optimized for use in MANETs. In OLSR, routing overhead is reduced in two ways. First, the flooding of control traffic is minimized by restricting the set of nodes, called multipoint relays (MPR), which relay control packets through the network. Every node in the network selects a MPR set from among its neighbors in such a way that control packets retransmitted by these relay nodes will reach all nodes in its 2-hop neighborhood. Second, the size of control packets is reduced because a node only includes link state information for the members of its MPR Selector set. This is the set of neighbors which have selected the node to be one of their multipoint relays.

Generally speaking, the smaller the MPR sets are, the more optimal the protocol will be. The frequency of the periodic updates can be increased to optimize OSLR's adaptivity to changes in topology. Since only periodic updates are used, OLSR can accommodate high node mobility. The use of MPRs makes OLSR particularly well-suited for use in large, dense networks.

2.1.2 On-demand protocols

In contrast to the methods in the previous section, on-demand protocols are not concerned with identifying and maintaining routes that are not currently in use. In an effort to reduce routing overhead, these techniques have a reactive nature – "we will create no route before it's time." Although discovering routes only when needed will result in higher packet latencies on average, many researchers believe that on-demand protocols inherently have lower overhead and higher throughput than proactive methods, and are hence superior.

Dynamic Source Routing - DSR

The Dynamic Source Routing (DSR) protocol uses source routing to deliver data packets. Routes are stored in a route cache, and each cache entry contains the entire path to be traversed to the destination. When a data packet is originated, the source places the entire path in the packet header. The intermediate nodes along this path simply forward the packet to the next hop specified in the header. Avoiding routing loops is clearly trivial with the use of source routing.

If a source does not have a route to the destination in its cache, it begins a route discovery process by broadcasting a route request (RREQ) packet. Each node receiving the RREQ searches its own route cache for a route to the requested destination. If no route is found, it adds its own address to the hop sequence contained in the RREQ header and broadcasts the RREQ again. A RREQ is tagged with an identification number that each node records so that it will not broadcast the request more than once.

The RREQ propagates through the network until it reaches either the destination or an intermediate node which has a route to the destination in its route cache. The RREQ header contains a record of the hops taken from the source, so this route can be reversed and used to unicast a route reply (RREP) packet back to the source. In the case that bi-directional links cannot be assumed, the RREP is piggybacked on a new request for a route to the source.

If an intermediate node is unable to forward a data packet to the next hop in its source route, it unicasts a route error (RERR) packet back to the source informing it of the broken link. The source removes the broken link from its route cache and all routes containing this hop are truncated at the point of the broken link. Any intermediate node that forwards the RERR will learn of the broken link and remove it from its route cache as well. The source can then attempt to use another route to the destination if one exists in the route cache, or it can initiate a new route discovery.

DSR uses source routing and route caching very aggressively. An intermediate node, upon finding the next hop link to be broken, can use an alternate route to the destination from its own route cache. A source receiving a RERR packet piggybacks the RERR on the following RREQ to help clean up the caches of other nodes which may have the failed link in a cached source route. Nodes are allowed to operate in promiscuous mode, examining the source route in the header of packets not addressed to it. If an intermediate node determines that a shorter route exists through itself, it sends this information back to the source in a RREP. In any case, the intermediate node can use snooping to learn of new source routes and add them to its cache.

There is no mechanism in DSR by which a stale route can be expired, nor is DSR able to choose the freshest route when multiple choices are available in the cache. If stale routes are used, they may cause other caches to become polluted. The use of promiscuous listening coupled with node mobility can result in stale routes polluting caches faster than they can be deleted by route error packets. A detailed discussion of DSR’s stale route problem is given in [48].

Ad hoc On-demand Distance Vector - AODV

The Ad hoc On-demand Distance Vector (AODV) [53] protocol is based upon the distance vector algorithm, and like DSDV, it uses sequence numbers to avoid the formation of long-lived routing loops. Unlike DSDV however, AODV only maintains routes that are in active use. If a source does not have a route to a packet’s intended destination, it buffers the data packet and broadcasts a RREQ in a manner similar to DSR. The source includes the most recent sequence number it has for the destination in the RREQ header. The RREQ is propagated through the network until it reaches the destination or a node with a fresh enough route to the destination, i.e. a route with a higher sequence number than the one in the RREQ. As the RREQ makes its way through the network, the intermediate nodes which forward the RREQ set up a reverse route to the source. This is the path along which the RREP from the destination (or other node with a fresh route) will be unicast back to the source. As the RREP is propagated, the intermediate nodes construct the forward path from the source to the destination. An important feature of AODV is its use of timers to expire routes which have not been used for some period of time. This policy is intended to minimize the stale route problem to which DSR, for example, is subject.

Each node maintains a list of predecessor nodes for each of its routing table entries. This is the set of neighbors which use the node as a next hop to the destination. When a next hop link breakage is detected by the MAC layer, the intermediate node which is unable to forward the data packet drops the packet and sends a RERR to each of its predecessor nodes. Each of these nodes in turn forward it to their predecessors, effectively erasing all routes which use the broken link.

To reduce the number of RREQ broadcasts required for route discovery, AODV uses an expanding ring search in the hope that the destination may be nearby. The time-to-live (TTL) field in the IP packet header is used to limit the number of times the RREQ is re-broadcast. At first only the 1-hop neighbors will receive the RREQ. If the source does not receive a RREP from one of these neighbors within a certain amount of time, it broadcasts another RREQ. This time the RREQ will reach all the nodes that are two hops away. If a reply is still not received, the process continues until the TTL reaches some threshold. At that point the RREQ is simply flooded throughout the network.

Another AODV optimization is local route repair. If an intermediate node which detects a route failure is more than half way from the source to the destination, it initiates a route discovery of its own rather than sending a RERR back to the source. Data packets are buffered during local route repair rather than being dropped.

Temporally Ordered Routing Algorithm - TORA

The Temporally Ordered Routing Algorithm (TORA) [51] is a link reversal protocol designed to operate in a highly dynamic mobile networking environment. Route discovery is source-initiated and multiple paths are provided to any destination. With the exception of short-lived loops, routes are guaranteed to be loop-free. Control messages are localized to a very small number of nodes in the vicinity of a change in network topology.

When a route is created, nodes use a “height” metric to establish a directed acyclic graph (DAG) rooted at the destination. Each link in the route is in either an upstream or a downstream direction depending on the relative height of its endpoints. If node mobility causes the DAG route to break, the height metric will change for some nodes and the direction of some links may be reversed. The height metric depends on the logical time of a link failure, so TORA assumes all nodes have synchronized clocks, perhaps by means of a Global Positioning System.

TORA runs on top of the Internet MANET Encapsulation Protocol (IMEP), which is required to provide reliable, in-order delivery of control messages. TORA is sensitive to routing packet losses and has been shown to perform poorly when compared to DSR and AODV [13, 20].

2.1.3 Other routing protocols

Zone Routing Protocol - ZRP

The Zone Routing Protocol (ZRP) [26] is designed to be used in an ad hoc wireless network consisting of many fast-moving nodes dispersed over a large geographical area. Due to the high degree of node mobility and the potentially large number of destinations, neither a pure proactive or a pure on-demand method will be adequate. The long delay and excessive control traffic during route discovery means an on-demand approach may not be applicable to realtime communication. On the other hand, proactive schemes are not appropriate because they use a lot of network capacity keeping routing information up-to-date. So, ZRP takes a hybrid approach, combining both types of routing.

Each node belongs to a routing zone, which is the set of nodes whose minimum distance (in hops) from the node in question is no greater than a specified number, called the zone radius. Within the routing zone, routes are maintained proactively. Packets destined for a node outside the zone are first propagated to a node on the periphery of the zone, and from there to a peripheral node of the destination's routing zone. Interzone routing is accomplished using an on-demand algorithm. The zone radius is a parameter that is used to adjust ZRP operation to network conditions. However, the zone radius must be chosen at the time the network is set up and cannot be changed. Hence, this decision may have a considerable impact on protocol performance.

Core-Extraction Distributed Ad hoc Routing - CEDAR

The primary focus of the Core-Extraction Distributed Ad hoc Routing (CEDAR) [64] protocol is quality of service routing in small to medium size ad hoc networks with tens to hundreds of nodes. Its goal is robustness rather than optimality. CEDAR identifies a set of nodes which form a core infrastructure for performing route computation. Virtual links are established (via tunnels) between nearby core hosts. Each core host is responsible for routing within its domain and must react to changes in network topology. Route computation is performed on demand by core hosts using local state information only. Quality of service routing is achieved by propagating available bandwidth information for stable links in the core infrastructure. CEDAR adapts quickly to topological changes and satisfies the bandwidth requirements of connection requests with high probability given that admissible routes exist.

Associativity Based Routing - ABR

Associativity Based Routing (ABR) is a loop-free, on-demand protocol [66]. ABR defines a new metric, the degree of association stability, and routes are selected on the basis of this metric. Each node periodically generates a beacon to announce its presence. A node also maintains an associativity table in which it records, for every other node with which it comes in contact, the number of times a beacon has been received from that node. This count, called associativity ticks, is the measure of association stability. A high degree of association stability may indicate low node mobility and vice versa. Paths chosen on the basis of association stability may not be the shortest possible, but they will tend to be the longest-lived routes and therefore be broken less frequently. A drawback of ABR is that the beaconing interval must be short enough so as to accurately reflect the spatial, temporal and connectivity state of the mobile hosts. This beaconing requirement may result in additional power consumption.

Signal Stability based Adaptive Routing - SSA

In Signal Stability based Adaptive Routing (SSA) [21], routes are established on demand and are chosen on the basis of signal strength between nodes. In addition to a routing table, each node maintains a signal stability table in which it records the signal strength of its 1-hop neighbors. Signal strength is obtained from periodic beacons sent by neighboring nodes and each link is classified as either strong or weak. Route requests are forwarded only if they are received over strong channels. The destination sends a route reply to the first arriving route request only since that packet most likely arrived over the shortest and/or least congested path. Link failures are reported to the source which then initiates a new route discovery.

2.2 Transport protocols

Transport layer protocols transform the host-to-host packet delivery service provided by IP into an interprocess communication channel. Adding a level of demultiplexing above the network layer makes it possible for applications to share the network. This basic service is provided by the Internet's User Datagram Protocol (UDP) [55]. In addition, UDP uses a checksum to verify that a message has been correctly transmitted.

Transport layer protocols are also used to turn IP's best-effort level of service into a reliable packet delivery service. IP packets may be dropped, reordered, or duplicated on the way from sender to receiver. By contrast, the Transmission Control Protocol (TCP) [57] enables applications to establish reliable, full-duplex connections. TCP includes a flow control scheme by which the receiver can limit the rate at which the sender transmits data. TCP also implements a congestion control mechanism to keep TCP senders from overloading the network and possibly causing congestion collapse.

2.2.1 TCP fundamentals

In this section, we describe the key components of the TCP protocol that are germane to our work. This is not intended to be a comprehensive description of TCP, for which [65] is an excellent reference.

TCP utilizes a sliding window algorithm to guarantee reliable, in-order packet delivery and to enforce flow control. The sender can have at most a window's worth of outstanding packets, i.e. packets for which no acknowledgement (ACK) of their receipt has yet arrived from the TCP receiver. Since the transmission of new packets must await the acknowledgement of previously transmitted packets, TCP is said to be self-clocking. To implement flow control, the receiver includes an advertised window size in each ACK to inform the sender of the amount of free space left in its buffer. The size of the sender's window (called the congestion window) can never exceed the receiver's advertised window size. We should point out that TCP keeps track of bytes, not packets, so that window size is actually expressed as a number of bytes. For simplicity, however, we will talk about windows in terms of packets.

TCP's acknowledgements are cumulative meaning that the sender is guaranteed that all packets up to and including the packet being ACKed have been successfully delivered. If a packet is delivered out of order, the receiver resends the same ACK it sent previously. When the sender receives this duplicate ACK, it knows that a packet has left the network, but that the left-hand side of its window cannot yet be advanced.

Adjusting the size of the congestion window

If a TCP sender could determine how much network capacity was available for its use at any time, it would know how many packets it could safely have in flight and could set its window size accordingly. In the absence of such knowledge, TCP adopts the strategy of probing the network for additional bandwidth by steadily increasing its window size at the rate of roughly one packet for every window's worth of ACKs. This increase in the rate of transmission continues until a packet drop occurs. Because transmission losses are very infrequent in the wired Internet, a dropped packet is assumed to have occurred when some router's buffer filled up. Hence, packet losses are treated as signs of network congestion. In response to the perceived congestion, the sender immediately cuts its window size in half, and then begins again to steadily increase the size of its window. This process, called additive increase/multiplicative decrease (AI/MD), continually adjusts the size of the sender's window (called the congestion window) over the life of a connection.

When a TCP connection is first established, the sender has no idea what the network capacity might be. Increasing the window size linearly as in AI/MD is likely to take too long, so instead the sender increases the window size by one for every ACK it receives, effectively doubling the number of packets in transit every round trip time (RTT). (The RTT is the interval from packet transmission to receipt of the corresponding ACK.) This is the slow start phase, so named because the original practice was to transmit an entire advertised window's worth of packets all at once, which is likely to overwhelm the routers even if sufficient bandwidth is available. Slow start ends when a packet drop is observed. The congestion window is halved and the sender begins AI/MD.

Detecting and responding to packet loss

TCP has two mechanisms for detecting packet loss. Whenever an ACK arrives for a packet not previously acknowledged (a new ACK), a timer is set for a period of time called the retransmit timeout interval (RTO). If the retransmit timer expires before the next new ACK is received, TCP deduces that packet loss has occurred. The packet following the last consecutively ACKed packet is retransmitted, the retransmit timer is reset, the congestion window size is reduced to 1, and the sender enters slow start. Unlike at the beginning of a connection, the sender now has some idea of what network capacity is available based on the recent history of the congestion window. Before reducing the congestion window size to 1, TCP stores one-half the current window size in a variable called the slow start threshold. This time, TCP will leave the slow start phase and begin AI/MD as soon as the congestion window size reaches this threshold.

In prolonged periods of network congestion, it is possible that the retransmitted packet will not be delivered either. The retransmit timer will expire again and the missing packet will be transmitted once more. To avoid adding to the network congestion, TCP doubles the RTO each time the timer expires. This exponential backoff of the RTO continues until the congestion is alleviated and packet flow is resumed. At that time, the RTO is reset to the current estimate of the RTT plus an additional amount to account for the sample variance of this estimate.

An additional method for detecting packet loss, called fast retransmit, is a heuristic intended to trigger a packet retransmission sooner than the regular retransmit timeout mechanism, which is often coarse-grained. When the sender receives a duplicate ACK, it knows that a packet has been delivered out of order. This suggests that an earlier packet may have been lost, in which case there is no need to wait for the retransmit timer to expire before retransmitting the dropped packet. However, the earlier packet may have only been delayed, so the sender waits until it sees some number of duplicate ACKs before retransmitting the missing packet. This number, let us call it the fast retransmit threshold, is normally set to three.

Collectively, these measures – AI/MD, slow start, and the retransmit timer – comprise TCP's congestion control mechanism. It has been tuned extensively from years of experience in the wired Internet. However, with the advent of wireless networking, it was soon evident that this mechanism does not work well in the wireless environment, and that TCP performance suffers as a consequence. The problem is that, in addition to congestion, packet losses in a wireless network can be caused by transmission errors and by node mobility [15]. Moreover, packet losses tend to be correlated; single, random packet losses are infrequent. A reduction in TCP's sending rate is not the appropriate response to these kinds of losses.

2.2.2 TCP performance in 1-hop wireless networks

A number of schemes were proposed to mitigate the ill effects that noncongestion-related losses have on TCP performance in 1-hop wireless networks. These networks are often characterized by sporadic bursts of bit errors and temporary breaks in connectivity during handoffs. Balakrishnan et al. compared various mechanisms for improving TCP performance across wireless links [9]. These methods fell into one of three groups – split connections, link layer approaches, and end-to-end schemes. In their experiments, the TCP sender resides on the wired network and the receiver is a mobile host communicating with a base station over a wireless link.

Split connections

Split-connection protocols [6, 70] divide the TCP connection into two connections – a wired connection from the sender to the base station and a wireless connection from the base station to the receiver. Loss recovery over the wireless link is separated from that across the wired network, and so is hidden from the sender. Because ACKs can reach the source before the data packet arrives at the mobile host, split-connection protocols violate the end-to-end semantics of TCP acknowledgements. Another disadvantage of these schemes is the overhead of maintaining TCP state at the base station, which tends to make handoffs complicated and slow. While the split-connection approach does insulate the TCP sender from wireless losses, timeouts on the wireless link cause the sender to stall frequently, resulting in poor end-to-end throughput.

Link layer approaches

A link-layer approach evaluated in [9] was the snoop protocol [8]. In the snoop method, a TCP-aware agent at the base station caches the packets sent across the wireless link until they are ACKed so that local recovery is possible in the event of packet loss. By shielding the TCP sender from duplicate ACKs caused by wireless losses, the snoop protocol yielded increases of 10%-30% in throughput compared to a link-layer protocol with no knowledge of TCP. A disadvantage of this or any other technique that requires examination of TCP packet headers is that it will not work when encryption is used.

End-to-end schemes

The end-to-end schemes considered in [9] were selective acknowledgements and the addition of an explicit loss notification (ELN) option to TCP acknowledgements. Selective acknowledgements (SACK) allow the sender to handle multiple losses within a window more efficiently. ELN notifies the sender that a noncongestion-related loss has occurred so the the sender can retransmit the lost packet without invoking congestion control. Two different SACK schemes were considered: a simple version of the SMART proposal [39], and an implementation based on RFC 2018 [49]. Compared to TCP Reno, the RFC 2018 SACK implementation yielded an increase in throughput of approximately 24%, while the fraction of packets successfully delivered (goodput) remained the same. The use of ELN increased throughput by about 25%, and again, goodput was unchanged. The combination of SACK and ELN might be expected to increase throughput even more, but this was reserved for future work.

2.2.3 TCP performance in MANETs

An effective way to deal with wireless transmission losses is to use a reliable link-layer. In MANETs, the IEEE 802.11 MAC protocol provides reliable transmission of packets across wireless links. The MAC layer will retransmit a packet until either an ACK is received, indicating an error-free transmission, or the maximum number of retries is reached, in which case the link is declared to be broken. MANETs are still subject, however, to noncongestion-related losses induced by node mobility. Now, instead of delays during handoffs, we are faced with route failures and the loss of connectivity during route repair.

Since the root of the TCP performance problem in a MANET is its inability to distinguish between losses due to congestion and other types of packet losses, designing a mechanism by which TCP can determine that a loss was not caused by congestion, and thereby avoid congestion control, seems to be an obvious means of solving the problem. If the TCP sender is notified that a route failure has occurred, it can suspend its normal response to packet drops until the route has been reconstructed. Several researchers have taken exactly this approach.

TCP-F: Chandran et al. propose a feedback based scheme they call TCP-Feedback or TCP-F [17]. In this scheme, when an intermediate node detects the disruption of a route due to the mobility of the next host along that route, it explicitly sends a Route Failure Notification (RFN) to the TCP sender. Other nodes that receive the RFN invalidate that particular route and do not forward any packets intended for that destination. Upon receiving the RFN, the source suspends all packet transmissions and freezes its state, including the retransmission timeout interval and the congestion window. Eventually, an intermediate node that has previously forwarded the RFN learns of a new route to the destination. That node then sends a Route Re-establishment Notification (RRN) to the source. When the source receives the RRN, it restores its previous state and resumes transmission. The effect of this scheme was studied by simulating a single TCP connection over which 200-byte packets are transmitted at 12.8 Kbps. Periodic route failures were generated, followed by route re-establishment after some fixed period of time, the route re-establishment delay (RRD). For RRDs in excess of 2 seconds, an increase in throughput of roughly 45% to 75% was reported. For a RRD of less than 1 second, however, essentially no benefit was derived from TCP-F. Three-fold increases in throughput were obtained when the transmission rate was increased to 128 Kbps.

ELFN: Holland et al. advocate the use of explicit link failure notification (ELFN) to significantly improve TCP performance in MANETs. In the ELFN scheme, when the TCP sender is informed of a link failure, it freezes its state (timers and window size) as in TCP-F. There is no route re-establishment notification, however. Instead, the source sends out packets (probes) at regular intervals to determine if a new route is available. Using the *ns-2* network simulator [22], they simulated a wireless network running TCP Reno and the DSR routing protocol. The TCP packet size was 1460 bytes, and the maximum window size was eight packets for both sender and receiver. They employed a random-waypoint network model, in which 30 nodes move toward randomly picked destinations in a 1500m x 300m flat, rectangular area. Upon reaching its destination, a node picks a new destination and continues onward without pausing. A total of 50 different mobility patterns were considered, and the mean speed at which the nodes travel was varied. For a mean node speed of 10 m/s, the throughput of a single TCP connection, averaged over the mobility patterns, was increased by 55% or more. Interestingly, an even greater increase in throughput (close to 100%) was obtained by simply turning off the DSR feature whereby intermediate nodes send out route updates based on the contents of their (often stale) route caches. In other words, avoiding DSR's stale route problem is of greater benefit than explicit notification of route failures.

TCP-BuS: In the TCP-BuS proposal [40], an explicit route disconnection message (ERDN) is generated at an intermediate node upon detection of a route failure. This message is propagated to the source which then stops transmission. Packet transmission is resumed after a partial path has been re-established from the node which detected the route failure to the destination and that information is relayed to the TCP sender in an explicit route successful notification (ERSN). During the course of a TCP connection, packets are buffered at the intermediate nodes along the path from sender to receiver. Nodes upstream from the failed link are able to forward these packets on to the destination once the route has been repaired, relieving the sender from having to retransmit these packets. This scheme is somewhat complex and would seem likely to have trouble with multiple route failures in quick succession, as in a high mobility network.

ATCP: In ATCP [45], a layer between TCP and the routing agent is proposed which, among other things, shields TCP from packet loss that is perceived to be non-congestion related. Upon learning of a route failure (by means of an ICMP *Destination Unreachable* message), ATCP places the TCP sender into *persist mode*, thus avoiding the invocation of congestion control measures. While in persist mode, TCP generates probe packets at exponentially increasing intervals up to a maximum of 60 seconds. Once the route is re-established and an ACK is received for one of the probe packets, TCP moves out of persist mode and resumes packet transmission.

To date, there has not been much reported in the way of evaluating TCP performance over different MANET routing protocols. Ahuja et al. [2] used *ns-2* to conduct a simulation-based comparison of TCP performance over several protocols, including AODV, DSR, and SSA. Only a single source of TCP traffic was simulated in their study. For low node mobility scenarios, the highest throughput was observed for DSR. As node mobility was increased, AODV performance became as good or better than that of DSR. Interestingly, for high node mobility scenarios, the SSA protocol achieved the highest TCP throughput. The authors attributed this to the fact that SSA selects routes on the basis of stability and stable routes experience fewer route failures. They concluded that the frequency of route failures, routing overhead, and delay in route establishment are the important determinants of TCP throughput in an ad hoc network. Like [29], they found that disabling route replies from cache actually improved TCP throughput for DSR by eliminating the effect of stale routes.

2.2.4 UDP performance in MANETs

A method, called a packet spacing protocol (PSP) [38], has been proposed for making wireline UDP flows perform well while being adaptive which is similar in principle to the Adaptive Datagram Protocol. In PSP, the sender initially transmits packets at the highest possible rate and the receiver responds with acknowledgements every round-trip time (RTT) to let the sender know how many packets it has received during the previous RTT. The RTT is determined by the sender and its current value is relayed to the receiver in the header of each PSP packet. From the acknowledgements it receives, the sender can determine whether the current packet rate is too high and can lower the rate by introducing spacing between the packets as they are sent. During the course of a run, if the send rate is below its maximum value and the sender determines that no packets are being lost, the send rate is increased by reducing the packet spacing.

CTP [69] is a configurable transport protocol constructed from independent micro-protocols, which provide the protocol's service attributes and functional components such as packet sequence numbering, positive acknowledgements, window-based or rate-based flow control, and congestion detection and control.

A combination of micro-protocols can be used to make CTP provide a sliding window-based unreliable packet delivery, similar to the one we explored in implementing ADP.

2.3 Performance analysis

Most analyses of TCP performance over wireless networks have been conducted either with an experimental testbed or by simulation. The experimental testbeds were built using real hardware, but in some cases, certain testbed components were emulated in software for better experimental control and accuracy. In the case of MANETs, experimental testbeds can be a bit cumbersome to build, although this has been done. Therefore, most MANET performance evaluations have used simulation.

Another method of studying performance is analytical. For example, the capacity of ad hoc wireless networks has been studied by formulating models of the MAC layer communications, and then applying these models analytically in various traffic scenarios [44]. The models themselves were validated in experimental testbeds and by simulation.

2.3.1 Experimental testbeds

The computer most frequently used in testbeds has been a Pentium-based PC running a Unix variant, such as BSD/OS or Linux, with a BSD version of TCP. 10 Mb/s Ethernet was commonly used for the wired infrastructure in 1-hop wireless setups. IBD ThinkPad laptops were used for mobile hosts in [9]. Wireless links have usually been WaveLAN direct sequence spread spectrum radios, which have a raw signaling bandwidth of 2 Mb/s and a nominal range of 250 m. Wireless links have also been emulated in software. For example, a time-based emulation of low-bandwidth wireless links was used in [14].

Cellular handoffs were studied in [15] by simulating the motion of the mobile host in software. This gave the researchers precise control of handoff event timings, and allowed them to explore the full range of handoff scenarios. Temporary disconnections were also simulated in [14, 25].

A MANET testbed was constructed at Carnegie-Mellon University by the DSR research group [47]. The testbed consisted of five mobile nodes which were laptops in cars traveling at about 25 mph (10 m/s), and two stationary nodes separated by a distance of about 700 meters. The cars moved continuously in a loop with the stationary hosts located at either end of the loop. The area used for the testbed was open to other traffic and had several stop signs, so node speed varied over time in a realistic manner.

2.3.2 Simulation methodology

A discrete event, packet-level routing simulator called MaRS (Maryland Routing Simulator) [3] was used in [19]. MaRS had been used previously to compare link-state and distance-vector routing algorithms for the NSFNET T1 backbone network [63]. The authors extended MaRS to simulate node mobility. Because their study was limited to network-layer details, they did not model link-layer details, such as MAC protocol, interference, and transmission errors, or any physical, radio channel details.

The simulation environment used in [24, 43] was GloMoSim [67]. GloMoSim is a scalable simulation environment for wireless network systems implemented in PARSEC (PARallel Simulation Environment for Complex Systems) [5]. It includes wireless protocols for radio propagation, mobility, MAC, network, transport, and applications. GloMoSim permits several network layers to be modeled together so that their interactions can be studied.

Perhaps the most widely used network simulator is *ns-2* from Lawrence Berkeley National Laboratory (LBNL) [22]. Extensions to *ns-2* from the Monarch Project at Rice University [61] include a set of MANET routing protocols (DSR, AODV, TORA, and others), an implementation of BSD's ARP protocol, and an 802.11 MAC layer and a radio propagation model. Mechanisms are included for modeling node mobility using pre-computed mobility scenarios.

In the Monarch wireless and mobility extensions to *ns-2*, a radio propagation model is employed which combines a free space model where signal power attenuates as $1/r^2$ up to some reference distance (typically 100 m for outdoor low-gain antennas), and a ground reflection model outside the reference distance where the signal falls off as $1/r^4$. The radio propagation model used for the GloMoSim simulations in [43] was a free space model, although the authors did implement the Simulation of Indoor Radio Channel Impulse Response Models (SIRCIM) [59] which considers fading, barriers, foliages, and multipath fading, and hence is a more accurate model. However, SIRCIM was not used in their study because its complexity increased simulation times drastically.

It is also possible to program one's own network simulation. This was done in [17], where the authors viewed the network as a black box that emulates MANET behavior from the transport layer point of view. A fixed number of nodes was assumed between the sender and receiver. The effects of node mobility appeared to the transport protocol in the form of sudden packet losses and delays. The frequency of route failures and the time to re-establish a route were simulation parameters.

2.3.3 Mobility scenarios

Three important elements of a MANET simulation are the number of mobile nodes, the dimensions of the space in which the nodes move about, and the mobility model, i.e. the manner in which the nodes move (speed, direction, and so on). A specific instance of these elements is called a mobility scenario.

Many MANET performance studies specify a square or rectangular, flat space in which the nodes are placed randomly at the start of the simulation. A square field has the advantage that all directions are equivalent due to the spatial symmetry. A rectangular field, on the other hand, forces the use of longer routes than would occur in a square field with the same node density. Commonly used fields are 1000x1000 m, 1500x300 m, and 2200x600 m. For the smaller of the rectangular fields, 50 nodes are normally used, which yields a node density high enough that network partitions occur very infrequently or not at all. 100 nodes is the usual choice for the larger rectangular field.

A commonly used mobility model is the *random waypoint* model [13, 30, 35]. At the beginning of the simulation, all nodes are stationary for an interval of time called the *pause time*. Each node then selects a random direction and moves to that destination at a randomly chosen speed. Once it has arrived at the destination, the node pauses again for *pause time* seconds, selects a new destination, and begins to move again. Nodes repeat this behavior for the duration of the simulation. A pause time of 0 results in continuous node movement, while a pause time equal to the length of the simulation yields a static model, i.e. no movement. Node speeds are usually chosen from a uniform distribution, for example 0-20 m/s.

Together, the mean node speed and the pause time determine the degree of mobility in the random waypoint model. A more general mobility metric was proposed in [34]. They define the mobility measure between any pair of nodes as the time average of their absolute relative speed. The total mobility metric, for a given scenario, is computed by averaging the mobility measure over all node pairs. They found that the number of link failures in a random waypoint scenario was roughly proportional to their mobility metric.

In addition to random scenarios generated using the waypoint mobility model, three “realistic” scenarios were designed and simulated in [34]: conference, event coverage, and disaster area. The capability to model obstructions to radio propagation was added to the simulation. The conference scenario models 50 people attending a seminar session. Mobility is low since only 10% of the nodes are moving at any time. High node density results in relatively high radio interference. This scenario tests the responsiveness of a routing protocol to local change in long-lived routes. The event coverage scenario models 50 highly mobile people, e.g. news reporters or stock brokers, who spontaneously form small clusters as they move around. There are many obstructions in the simulation area, and routes are generally short-lived. This scenario tests how well a routing protocol responds to rapid changes in topology. In the disaster area scenario, rescue team members form ad hoc networks in three geographically separate groups which can intercommunicate only via nodes mounted on fast moving vehicles, e.g. helicopters. The workers move slowly and randomly within each group. This scenario tests the ability of a routing protocol to deal with diverse mobilities and network partitioning events.

Chapter 3

Adaptive Distance Vector

In this chapter, we describe a recently proposed routing protocol for MANETs, called Adaptive Distance Vector (ADV) [12]. We present the results of a performance analysis in which we compare ADV to several other proposed MANET routing protocols. These results greatly extend the performance analysis conducted in [12]. We also describe enhancements to ADV that we have made in order to tune its TCP performance.

ADV is a distance vector algorithm that uses sequence numbers to avoid long-lived loops [27, 54]. As a distance vector algorithm, ADV uses routing updates to learn and maintain routes. However, ADV reduces routing overhead by adapting the size and frequency of updates to network traffic and node mobility. First, routes are maintained only to nodes which are active receivers in order to reduce the number of routing table entries that are included in updates. Second, routing updates are triggered in response to changing network conditions, eliminating the need for periodic full updates such as those used in RIP [27]. Thus, the routing overhead in ADV varies with load and mobility, a characteristic of on-demand routing protocols.

3.1 Adapting the size of routing updates to network load

In a traditional distance vector routing algorithm, each node in a network maintains a route to every other node and includes all of its routing table entries in periodic routing updates. To reduce the size of routing updates, ADV advertises the routes to active receivers only. A node is an *active* receiver if it is the receiver of any currently active connection. Each routing table entry includes a field, *receiver flag*, that indicates whether or not the destination is an active receiver.

Making routes active: When a new connection is to be established, the source node broadcasts an *init-connection* control packet advertising that the destination node of the connection is now an active receiver. When an intermediate node receives this control packet, it turns on the corresponding receiver flag in its routing table and begins advertising its route to the receiver in future updates. The node then rebroadcasts the control packet. Thus the entire network is flooded with the init-connection packet.

When the destination node receives the init-connection packet, it responds, if it is not already an active receiver, by broadcasting a *receiver-alert* control packet with its current sequence number. As nodes process and rebroadcast the receiver-alert packet, they establish reverse routes to the destination. With a pair of broadcasts, all network nodes learn that the destination node has become an active receiver and acquire routes to it.

Because the source node includes its routing table entry for the receiver, with the receiver flag set, in all future updates, other nodes will learn of the new connection even if the init-connection control packet is lost. This method of advertising an active receiver will be slower than the connection-initiation process. However, there is no need to use timers, as in AODV, to ensure that the connection has been established.

Making routes inactive: At the end of a connection, the source node broadcasts an *end-connection* control packet indicating that the connection has been closed. If the destination node has no other active connections, it broadcasts a *non-receiver-alert* control packet indicating that it is no longer an active receiver. Upon receiving one of these control packets, a node turns off the corresponding receiver flag in its routing table and does not advertise its route to the destination in future updates. By means of the connection-initiation and connection-termination processes, ADV varies the number of routes it maintains with the number of open network connections.

Another method for making routes inactive, which does not depend on the propagation of *end-connection* and *non-receiver-alert* control packets, is the use of an active route timeout like the one used in AODV. When a route has not been used for some period of time, say 30 seconds, the destination node is no longer considered an active receiver and that route will not be included in subsequent updates.

Table 3.1: Additional fields in a routing table entry at a node in ADV.

Routing table entry	**Function**
Packets queued	Number of data packets waiting for a route to this destination
Packets handled	Number of data packets forwarded to this destination in the recent past
Receiver flag	Indicates whether this destination is an active receiver or not
Advertisement count	Indicates the number of updates in which this entry should be advertised

3.2 Adapting the frequency of routing updates to network conditions

In order to make the frequency of routing updates adaptive to changing network conditions, several variables are maintained to track network load and node mobility. Some of these variables are defined for each entry in the routing table, while the others are defined at the node-level. We begin this section with a description of these variables. Later we explain how these variables are used in combination to determine when it is time to broadcast a routing update.

3.2.1 Routing table variables

The important variables that are part of each entry in the routing table are described in detail below. These variables are in addition to the usual fields used in distance vector algorithms, i.e. sequence number, metric, and next hop. The additional fields are also shown in Table 3.1 for easier reference.

Packets queued: This variable counts the number of data packets buffered at a node waiting for a route to the destination corresponding to this routing entry. This count is incremented whenever a data packet is buffered because no route to the destination is currently available. It is decremented whenever a data packet is removed from the buffer for transmission once a route has been obtained. A non-zero count indicates an immediate need for a fresh valid route to the destination.

Packets handled: This variable counts the number of data packets forwarded to this destination, including packets originated by a source node. The count is incremented for each packet forwarded to this destination and is halved whenever an update containing an entry for this destination is propagated. A non-zero count

Table 3.2: Node-level routing variables maintained in ADV.

Variable	**Function**
Trigger meter	Sum of trigger values for recent events which indicate the need for an update
Trigger threshold	An adaptive trigger meter level beyond which a node broadcasts a partial update
Neighbor changes	Counts the nodes entering and leaving 1-hop range
Packets buffered	Counts total number of packets queued waiting for routes

indicates that the node is a *forwarding node* to this destination for one or more neighboring nodes. This variable becomes redundant when the packets queued field has a non-zero count.

Receiver flag: This flag is used to indicate whether or not the destination is an active receiver. The routing entry is included in routing updates only if the flag is set.

Advertisement count: If the destination is an active receiver, this variable determines whether or not this routing entry is to be included in the next partial update. (Routes to all active receivers are included in full updates, regardless of the advertisement count.) The advertisement count is incremented if the routing entry has any valid route with either a higher sequence number or a better hop count than the current routing table entry, or if the route has become invalid. The count is decremented (to no less than 0) each time the routing entry is included in an update. The detailed conditions under which the advertisement count is set are given in Table 3.4.

3.2.2 Node-level variables

Each node maintains a single copy of each of the variables described below. These node-level variables are also shown in Table 3.2.

Trigger meter: A node should trigger an update if it has packets waiting in buffers for routes to various destinations, or if it has received an update from a neighbor which indicates a need for fresh routes. Also, a node should advertise, as soon as possible, any fresh valid or invalid route to a destination for which it is a forwarding node. Routing overhead can be reduced considerably if, instead of performing a routing update immediately upon encountering one of the above conditions, a node waits until it sees a sufficient need to trigger an update. Each node tracks events which indicate the need for an update. A value is associated with each such event that is proportional to the urgency with which an update is required. This value is added to a variable called the *trigger meter*. When the trigger meter exceeds some critical value, the *trigger threshold*, a node schedules a routing update immediately.

Trigger threshold: The trigger threshold is used to decide when a partial update needs to be triggered. After processing an update received from one of its neighbors, a node checks to see if the trigger meter has crossed this threshold value. If so, a partial update is immediately scheduled for transmission. The trigger meter is reset to zero after a partial or a full update is scheduled. The trigger threshold changes dynamically based on the recent history of trigger meter values at the time of previous triggered updates. The computation of this threshold value is explained later in this section.

Neighbor changes: In order to measure node mobility, a count is kept of the number of neighbor changes that have taken place during the period of some fixed number of full updates. Neighbor changes consist of other nodes coming into and going out of a node's 1-hop neighborhood. The number of nodes coming into the 1-hop range is determined by the number of times an update or connection control packet is received from a node whose metric was previously greater than 1. The number of nodes going out of the 1-hop range is determined by the number of link breakages that are reported by the MAC layer, plus the number of times an update or connection control packet is received from a node whose metric has changed from 1 to a higher value. If the number of neighbor changes exceeds a preset number, node mobility is categorized as HIGH_SPEED. Otherwise, the network is considered to be LOW_SPEED.

Packets buffered: This variable stores the total number of packets buffered at a node waiting for a route, i.e. it is the node-level counterpart of the *packets queued* field in routing table entries.

3.2.3 Tunable constants

There are several constants that are defined for the ADV protocol. These constants are tunable in the sense that the values of the constants can be chosen so as to optimize the performance of an ADV implementation. The values currently used for ADV simulations yielded the best protocol performance during the development and testing of the original ADV implementation.

Trigger meter constants: There are four constants associated with the trigger meter: TRGMETER_FULL, TRGMETER_HIGH, TRGMETER_MED, and TRGMETER_LOW (given in descending order of their values). These constants are used to update the value of the trigger meter in response to various events (see Section 3.2.5). The values currently assigned to these constants are 50, 20, 8, and 5, respectively.

Packet buffering constants: Two constants, BUFFER_THRESHOLD and BUFFER_TIMEOUT, are associated with packet buffering. BUFFER_THRESHOLD is set at 2; if the number of packets buffered at a node is greater than or equal to BUFFER_THRESHOLD, this is an indication that a routing update is required. The BUFFER_TIMEOUT constant limits the amount of time packets can remain in the queue; packets that have been buffered longer than BUFFER_TIMEOUT seconds are dropped. In our experiments, we have used BUFFER_TIMEOUT values of 1, 5, and 30.

Other constants: The number of neighbor changes at or above which a network is considered to be HIGH_SPEED is currently set at 8. Neighbor changes are tallied over a period of 6 full updates. The minimum time that must elapse between routing updates is currently set at 0.5 second. When a new connection is to be established and a route is already known to the destination, the route is discarded if it has not been used in the last 15 seconds.

3.2.4 Sending routing updates

In this section, we describe the process of sending routing updates. First, we give the structure of a routing update entry (depicted in Table 3.3) and then we explain in detail the processing of a routing update at a node.

A node identifies a destination as an active receiver if the receiver flag is set in the corresponding routing table entry. A full update includes the entries for all active receivers regardless of the need for advertisement. In a partial update, only the entries of the active receivers whose advertisement count is non-zero are

Destination IP address (32 bits)
Sequence number(16) | Metric(8) | Is_receiver(1) | Expected_response(2) | Unused(5)

Table 3.3: Fields in a routing update entry. Total entry size is 8 bytes. Sequence number is the latest known sequence number of the destination. Metric is the hop count to the destination. Is_receiver flag indicates whether or not the destination is an active receiver. Expected_response gives the priority with which a node receiving this update should either respond with a fresh valid route or propagate this routing information further.

included. The conditions under which the advertisement count of a routing entry is increased or decreased are given in Table 3.4. A node always includes the routing table entry for itself in an update regardless of its receiver status. Sequence numbers are used in a manner similar to their usage in other distance vector algorithms [53, 54] except that every update (full and partial) results in a new higher sequence number. Sequence numbers are also updated by source and destination nodes when connection-initiation and connection-termination control packets are broadcast.

Every routing update entry is assigned an expected response value of ZERO (bit sequence 00), LOW (01), MEDIUM (10) or HIGH (11). Expected response values are determined as follows:

- An expected response of HIGH is assigned if there are packets buffered waiting for a route to this destination (packets queued > 0).
- In a HIGH_SPEED network, an expected response of MEDIUM is assigned if the node is a forwarding node to this destination (packets handled > 0).
- In a LOW_SPEED network, an expected response of LOW is assigned if the node is a forwarding node to this destination (packets handled > 0).
- If none of the above criteria apply, an expected response of ZERO is assigned.

The expected response value in an update entry indicates the priority with which a node receiving this update should either respond to this advertised need for a fresh route or propagate the updated routing information further.

To prevent thrashing due to frequent updates, we ensure that at least 500 ms elapse between any two updates triggered by a given node. If the trigger threshold is crossed within 500 ms of a previous update, the newly triggered update is delayed until the minimum time between updates has elapsed. Also, when an update is triggered, a node checks its interface queues for a previous update that is still awaiting transmission. If there is an update pending, then it is merged with the current update thereby avoiding the overhead of an additional update.

Because full updates are triggered when the trigger meter value is high enough, there is no need for periodic full updates. Nevertheless, a mechanism exists in ADV for transmitting full updates at regular intervals if so desired.

3.2.5 Processing received updates

The conditions under which a node updates its routing table upon receiving an update are specified in detail in Table 3.4. In addition, a node copies the Is_receiver flag from an update entry whenever the received sequence number is greater than or equal to the sequence number currently stored in the node's routing table. This serves to identify the active receivers in the event init-connection, receiver-alert, and non-receiver-alert control packets are lost.

As each entry in the received routing update is processed, the trigger meter is increased by a constant amount that is appropriate for the expected response value. Specifically, the trigger meter is increased by TRGMETER_HIGH, TRGMETER_MED, or TRGMETER_LOW for an expected response of HIGH, MEDIUM, or LOW, respectively. The trigger meter remains unchanged for an expected response of ZERO.

Some of the conditions in Table 3.4 also call for increases in the trigger meter. These increases are in addition to those resulting from expected response values. In order to adapt quickly to topology changes in a HIGH_SPEED network, a forwarding node adds TRGMETER_MED to its trigger meter for each update entry with a fresh route or a route invalidation. The trigger meter is increased by TRGMETER_FULL when an active receiver learns that its routing entry is invalid in an update received from a neighbor. This is to ensure that a full update is scheduled immediately after the update has been processed, the intent being to keep one-hop neighbors informed of the presence of an active receiver within their neighborhood so that they can serve as forwarding nodes to that receiver.

My_Entry	Received_Entry	Condition	Action
Valid	Valid	my_seqno < recv_seqno	Update My_Entry with Received_Entry Increment the advertisement count If forwarding node in high-speed network, trigger meter += TRGMETER_MED
		my_seqno == recv_seqno && my_metric < recv_metric + 1	Update My_Entry with Received_Entry Increment the advertisement count
		my_seqno == recv_seqno && my_metric == recv_metric + 1	Decrement the advertisement count Make the source of this update the next hop in My_Entry
		my_seqno > recv_seqno || (my_seqno == recv_seqno && my_metric < recv_metric - 2)	Increment the advertisement count
Valid	Invalid	my_seqno < recv_seqno && my_hop is source of this update	Invalidate My_Entry since I am dependent on this neighbor Increment the advertisement count If forwarding node in high-speed network, trigger meter += TRGMETER_MED
		my_seqno > recv_seqno	Increment the advertisement count
		recv_dest == My_Address && receiver flag == TRUE	Trigger meter += TRGMETER_FULL
Invalid	Valid	my_seqno < recv_seqno	Update My_Entry with Received_Entry Increment the advertisement count If forwarding node in high-speed network, trigger meter += TRGMETER_MED
		my_seqno > recv_seqno	Do nothing
Invalid	Invalid	my_seqno < recv_seqno	Copy recv_seqno into My_Entry
		my_seqno > recv_seqno	Do nothing

Table 3.4: Processing a routing update entry. recv_dest, recv_seqno, and recv_metric indicate the values for the destination, sequence number, and hop count in the received routing update entry. my_seqno, my_metric, and my_hop indicate the destination sequence number, hop count, and the next hop node currently stored in the entry for recv_dest in the routing table at the node which received this update.

After all the update entries have been processed, the accumulated trigger meter value is examined. If it exceeds TRGMETER_FULL, a full update is immediately scheduled for transmission. Otherwise, if the trigger meter has crossed the trigger threshold, a partial update is scheduled.

3.2.6 Processing data packets

When processing a data packet, a node looks in its routing table for a route to the intended destination. If a valid route exists, the data packet is immediately forwarded to the next hop node specified in the routing entry. Otherwise, the packet is buffered until a valid route becomes available. The non-availability of routes indicates that more frequent routing updates are required in order to maintain up-to-date routes. Therefore, when a new packet is buffered, if the number of packets buffered is greater than or equal to BUFFER_THRESHOLD, TRGMETER_MED is added to the trigger meter in order to more quickly trigger an update.

3.2.7 Computing the trigger threshold

A node that is either an active receiver or a forwarding node is called an *active node*. The trigger threshold value for an active node is changed dynamically based on the recent history of trigger meter values recorded at the time of previous partial updates. The trigger threshold value is initially set to TRGMETER_HIGH. Each node keeps track of the number of partial updates it has done since the last full update, the sum of trigger meter values at the time of each partial update, and the time elapsed since the last full update. The trigger threshold value is computed at the time of each full update using the following rules:

- The average trigger meter value per triggered update is computed by dividing the sum of trigger meter values (which is accumulated in a special variable because the trigger meter is reset after every update) by the number of partial updates since the last full update. If no partial updates have been done since the last full update, the average trigger meter value is set to one-half the sum of trigger meter values.

- In order to derive an estimate of the average trigger meter values observed in the recent past, an historical average trigger meter value is maintained and updated at each full update. Denote the newly computed average trigger meter value by t_n, and the current historical average trigger meter value by t_h. Then the new historical average is computed as $t_h = (t_h + t_n)/2$. This is similar to the smoothing function, $(\alpha t_{old} + \beta t_{new})$ used sometimes in, for example, the computation of the estimated round trip time in TCP. Here equal weights of 0.5 are given to α and β in order to adapt to network mobility changes rather quickly.

- Although the above criteria work well, it has been observed that some nodes will benefit from engaging in even more routing activity. To enable this, the number of partial updates done since the most recent full update is compared to the maximum expected number of partial updates given that updates must be no less than 0.5 seconds apart. If the number of partial updates is at least 60% of the maximum number, the trigger threshold is set at t_h. If the number of partial updates is less than 60% but greater than or equal to 30% of the maximum number, the trigger threshold is set at $0.75 * t_h$. Otherwise, the trigger threshold is set at $0.5 * t_h$. In the latter two cases, the trigger threshold is set to a fraction of t_h in order to increase the frequency of partial updates.

For *non-active* nodes, the trigger threshold is set to a constant value that is commensurate with the speed of the network. For low speed networks, the trigger threshold is set to TRGMETER_HIGH, and for high speed networks it is set to TRGMETER_MED.

The idea in computing the trigger threshold differently for different nodes is to ensure that active receiver nodes and nodes with too many buffered packets will engage in a level of routing activity that is adaptive to network conditions, while at the same time preventing other nodes from transmitting more updates than necessary.

3.3 Tradeoffs in ADV update strategy

In traditional distance vector protocols, a node is required to send a routing update whenever the metric for a route is changed in the routing table. This update must occur almost immediately even if it is not yet time for the next periodic update. These triggered updates can cause excessive loads on networks with limited capacity or a large number of nodes. To avoid this overhead, the triggered updates may be delayed by a small random time between 1 and 5 seconds so that if additional route changes occur, the changes will all be consolidated into one triggered update. However, such delaying is beneficial only in static and low speed networks. In ADV, any advertised need for fresh routes can trigger an update. Each of the entries in a routing update has an expected response value, which indicates the necessity of triggering an update. The expected responses are dependent on the mobility and load conditions of the network. Unlike in other DV algorithms, having packets waiting for a route is a sufficient reason for a node to transmit an update. Also, in other DV based protocols periodic full updates are necessary to maintain fresh routes, whereas in ADV full updates occur only as network conditions warrant.

In on-demand protocols, the need for a fresh valid route to an active receiver will immediately result in a route discovery process. Intermediate nodes may broadcast a route request should a route to the receiver become unavailable, as in, for example, local route repair in AODV. Because route replies are unicast, they reach the intended source nodes reliably. In ADV, however, a fresh valid route can only be obtained via routing updates received from neighboring nodes. Hence, obtaining a valid route might take a longer time in ADV than in an on-demand protocol.

3.4 ADV performance analysis

3.4.1 Experimental methods

Simulation environment

For our simulations, we used the *ns-2* network simulator [22] with the wireless and mobility extensions from the Rice University Monarch Project [61]. These extensions include the modeling of an IEEE 802.11 wireless LAN [32]. We used the Monarch implementations of DSDV and DSR, and all parameter values and optimizations used for DSDV and DSR are as described by Broch et al. [13]. The AODV implementation is based on the original Monarch release, with a number of added performance optimizations [20]. ADV was implemented by Boppana and Konduru as described in [12]. Some of the important parameter values used in the various routing protocols are given in Tables 3.5 - 3.8. Link layer notification of broken links was used for ADV, AODV and DSR. This was not done in DSDV since Broch et al. report that link-layer

Table 3.5: Values of various parameters used in the DSDV protocol.

Parameter	Value
Periodic update interval	15 seconds
Minimum time between two triggered updates	1 second
Maximum packets buffered per node per destination	5
Periodic updates missed before link declared broken	3

Table 3.6: Values of various parameters used in the ADV protocol.

Parameter	Value
Minimum time between two triggered updates	0.5 seconds
Maximum packets buffered per node	64
Buffer timeout	1 or 5 seconds
Buffer Threshold	2
TRGMETER_FULL	50
TRGMETER_HIGH	20
TRGMETER_MED	8
TRGMETER_LOW	5
Periodic update interval	∞

feedback increases the number of routing updates [13]. Therefore in DSDV, loss of a neighbor was detected when 3 consecutive periodic updates were missed from that neighbor.

We simulated an ad hoc network with 50 nodes moving in a square 1000m x 1000m field. A node that reaches the edge of the field exits the field and reenters it immediately from the opposite side of the field traveling in the same direction and at the same speed as before. This model simulates a network in which nodes enter and leave over time, but the total number of nodes is constant. For comparison with other performance studies [13, 19, 20, 34, 43], we also used field sizes of 1500m x 300m for a 50-node network and 2600m x 600m for a 100-node network. In those simulations, nodes which reach the field boundary

Table 3.7: Values of various parameters used in the AODV protocol.

Parameter	Value
Active route timeout	50 seconds
Request retries	3
Maximum packets buffered per node	64
Buffer timeout	30 seconds
TTL_START	1
TTL_INCREMENT	2
TTL_THRESHOLD	7

Table 3.8: Values of various parameters used in the DSR protocol.

Parameter	Value
Time between retransmitted requests	0.5 seconds
Maximum packets buffered per node	64
Buffer timeout	30 seconds
Primary route cache size	30 entries
Secondary route cache size	34 entries

bounce back into the field. The node density of the rectangular-field networks is higher than in the square field and network partitions are extremely rare, but the nodes have a tendency to cluster in the middle of the field. The nodes in a square field tend to be more evenly distributed. Despite the lower node density, the square field has very few network partitions.

During a simulation, the nodes move according to a mobility pattern or scenario that was generated in advance using a model of node mobility. Several interesting mobility models are described in [30]. The mobility patterns we used were based on the *random waypoint* model, in which a node travels in a predetermined direction at some speed until it reaches its destination. Upon reaching its destination, a node pauses for a set length of time before moving on to its next destination. To mimic high node mobility, node speeds were uniformly distributed between 0 m/s and 20/ms, yielding a mean node speed of 10 m/s, and the pause time was set to zero. For low-mobility scenarios, the pause time was set to 100 seconds.

We simulated the *steady-state* conditions of a network with various background traffic loads generated by 25, 50, and 100 constant-bit-rate (CBR) connections. We also simulated the *transient* conditions of a network with varying loads from 10 to 50 CBR connections started 10 at a time over a 5 minute period. The CBR packet sizes for all simulations were fixed at 512 bytes. Since routing protocol performance is sensitive to movement patterns, a total of 30 different mobility scenarios were used and each performance metric was averaged over these scenarios.

The steady-state results for the high-mobility, 50-node square field with 25 and 50 connections are given in Section 3.4.2. The transient-state results for the 50-node square field are given in Section 3.4.3 with additional results included in Appendix A. The steady-state results for the 50-node and 100-node rectangular fields and for the 100-node low-mobility network are presented in Appendix A.

Performance metrics

In each simulation run, we measured packet latency, delivery fraction, and throughput. *Packet latency* is the time, in milliseconds, it takes for a data packet to travel from its source to its destination, including queuing and protocol processing delays as well as propagation and transmission delays. The packet *delivery fraction* is the ratio of the number of packets successfully received by the destination to the number of packets transmitted by the source. Data packets that take more than 5 seconds to reach the destination are considered no longer usable and are treated as having been dropped by the destination. *Throughput* is the average rate at which data is delivered from source to destination, measured in Kbits per second.

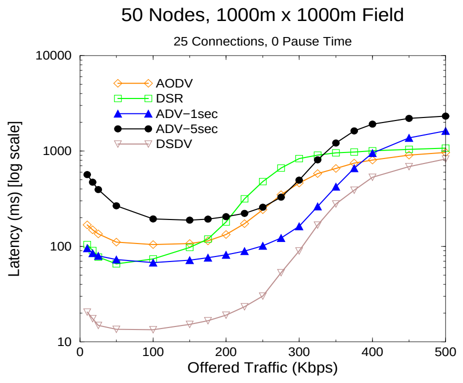
In order to gauge the routing protocol overhead, we measured both the number of routing packets and the number of bytes of routing data transmitted per second at the IP layer. The overhead includes the routing of the background CBR traffic. For DSR, the number of bytes of routing data transmitted includes the routing information carried by data packets.

3.4.2 Steady-state behavior of a high mobility network

In this section, we present the results obtained for a high mobility, 50-node network in the square field. We varied the number of CBR flows, using 25 and 50 connections. In order to measure the steady-state performance of the routing protocols, statistics were gathered for 500 seconds following a warm-up period of 100 seconds.

Packet latency

Figure 3.1 shows the average packet latencies observed for 25 connections. DSDV consistently yields the shortest packet delays. This is expected since DSDV's proactive route updates ensure that a route to any reachable destination is available when needed. Because of the adaptive proactive nature of its route maintenance, ADV with 1-second buffering (labeled ADV-1sec in the performance graphs) enjoys a similar, though smaller, advantage with respect to the on-demand algorithms, which incur packet delays during the route discovery process. Increasing the buffer timeout interval to 5 seconds enables ADV to deliver more packets, as shown in Figure 3.1, but these packets will experience longer delays on average. DSR is able to



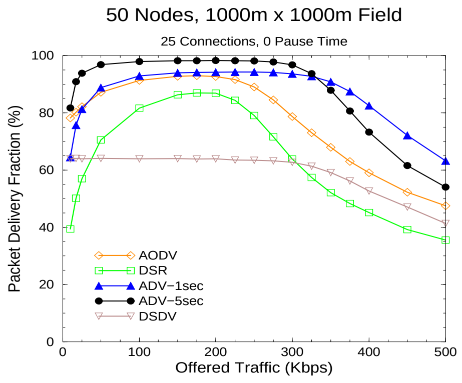


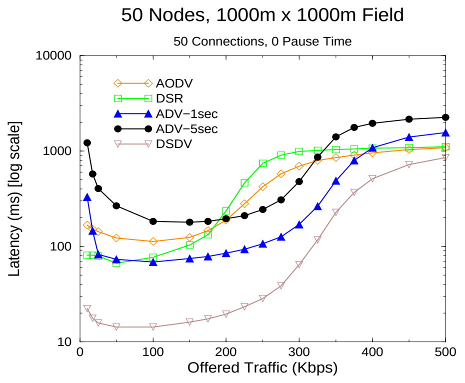
Figure 3.1: Packet latency and delivery fraction for 25 connections in a 50-node network on a 1000m x 1000m field.

snoop packet headers for source routes, reducing its need for route discovery, so DSR yields lower packet latencies than AODV. However, as DSR and AODV approach saturation in the neighborhood of 200 Kbps of offered traffic, packet delays increase much more rapidly for DSR and latencies are lower for AODV.

The packet latencies observed for 50 connections are shown in Figure 3.2. The additional connections have no effect on DSDV latencies because a proactive algorithm maintains all possible routes whether they are in use or not. The larger number of CBR flows has very little impact on ADV latencies except at very low traffic loads, for which increased packet delays were observed. With a low volume of traffic, the frequency of ADV route updates is reduced which in turn leads to higher packet latencies. This effect is magnified as the number of active receivers increases. Average latencies for AODV and DSR were unchanged for very low and very high traffic loads, but packet delays increased more rapidly as saturation was reached between 150 and 250 Kbps of offered traffic. The longer delays result from the increased route discovery overhead required to establish and maintain more connections. The impact of this additional overhead is minimal in a very lightly loaded network and is inconsequential when the routing protocol is well beyond its saturation point.

Delivery fraction

In Figure 3.1, we see that for all the protocols except DSDV, the packet delivery fraction rises rapidly as the offered traffic increases from very low traffic loads to moderate loads. The proactive DSDV maintains a



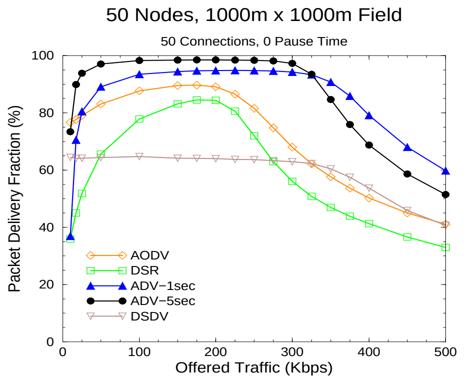


Figure 3.2: Packet latency and delivery fraction for 50 connections in a 50-node network on a 1000m x 1000m field.

fairly constant delivery rate as the network load increases, but the rate is only 60 to 65%. Because DSDV does not consider the link to a neighboring node to have failed until 3 periodic updates from that node have been missed, there can be a significant delay before the link is repaired during which time packets continue to be dropped at the MAC layer. In these simulations, nearly all the DSDV packet drops occurred at the MAC layer as a result of mobility-induced link failure or MAC-layer congestion. There were very few drops due to late packet delivery, and no packets were dropped because of routing-layer buffer overflow.

With 5-second buffering, ADV drops fewer packets in the network, so it is able to deliver a very high fraction of approximately 98%. AODV and ADV with 1-second buffering have comparable delivery rates, peaking at just over 95%. DSR delivery fractions lag the others, reaching a maximum at around 85%. The point at which the various protocols begin to saturate is marked by declining delivery rates. The on-demand protocols saturate at around 200 Kbps of offered traffic, while ADV and DSDV are able to sustain higher loads and do not reach saturation until the offered traffic is in excess of 300 Kbps.

Delivery fractions for DSDV and ADV are unaffected by the larger number of connections in Figure 3.2, except at very high traffic loads for which ADV's delivery rate is slightly lower. Due to the increased routing overhead for 50 connections, the delivery fractions for AODV and DSR are reduced by about 5 percentage points and decline a bit more rapidly once saturation is reached.

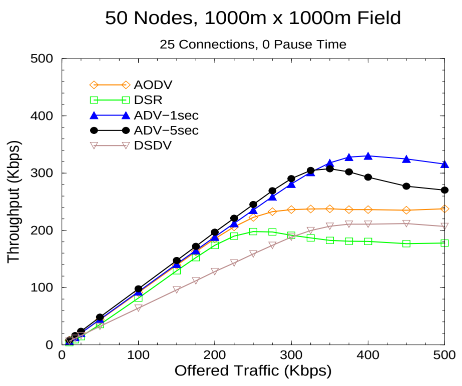


Figure 3.3: Throughput for 25 connections in a 50-node network on a 1000m x 1000m field.

Throughput

In Figure 3.3, we see that ADV, with its high packet delivery rate, yields the highest throughput. The use of a longer, 5-second buffer refresh time gives the best throughput, but as saturation is reached between 300 Kbps and 350 Kbps, the additional buffering results in increased network contention. As a consequence, throughput and packet delivery fraction fall off more rapidly than with a buffer refresh time of 1 second. DSDV gives the lowest throughput at low to moderate traffic loads. DSR throughput peaks at about 200 Kbps, while AODV is able to sustain a higher throughput of nearly 240 Kbps.

Every protocol except AODV experiences a decline in throughput beyond its saturation point, especially 5-second ADV. The large quantity of packets that ADV is buffering at intermediate nodes is a problem at very high loads. Reducing the maximum buffer time to 1 second allows ADV to do a much better job of sustaining its peak throughput. It is noteworthy that AODV is robust to high traffic loads, maintaining its peak throughput as the volume of offered traffic increases.

As shown in Figure 3.4, DSR throughput is not affected by the increase in the number of connections from 25 to 50. The relative performances of ADV and the on-demand protocols remain the same as in the 25-connection case, although throughputs are somewhat lower beyond saturation. In particular, the peak throughput for AODV dropped from about 240 Kbps to around 200 Kbps.

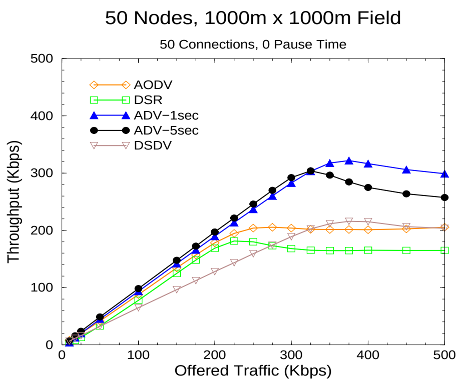
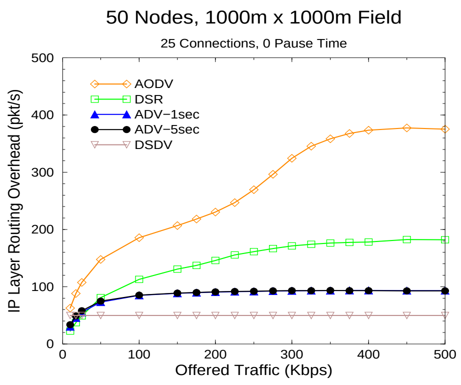


Figure 3.4: Throughput for 50 connections in a 50-node network on a 1000m x 1000m field.



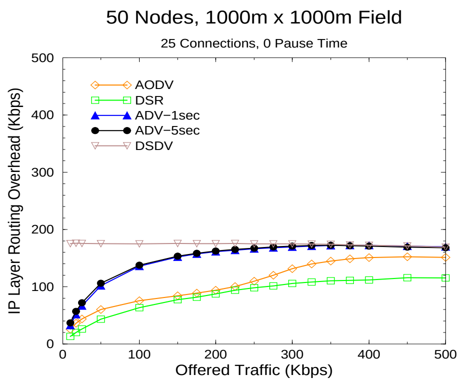
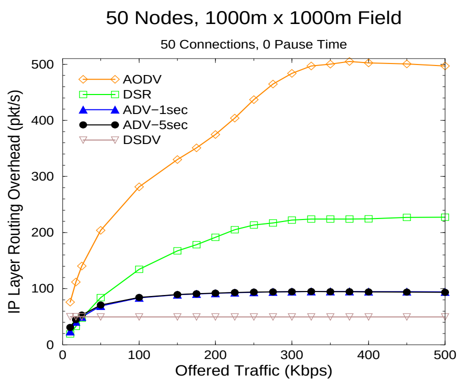


Figure 3.5: IP-layer routing overhead for 25 connections in a 50-node network on a 1000m x 1000m field.

Routing overhead

DSDV uses periodic updates to proactively maintain its routing tables. These updates occur at regular intervals and include routes to all nodes in the network. Thus, the rate at which routing packets are generated is constant with respect to the offered traffic, and if the number of nodes is fixed, the size of these packets is also constant. As shown in Figure 3.5, DSDV generates fewer routing packets than the other protocols except at very low traffic loads. However, because routing information is maintained for every node, DSDV produces the highest volume of routing traffic as measured in bytes.

ADV also uses routing updates, but rather than being periodic, the updates are load-driven and so the number of updates increases with increasing load. The frequency of ADV updates is limited, however, and this maximum rate is reached as the offered traffic exceeds 200 Kbps. Due to the volume of route requests



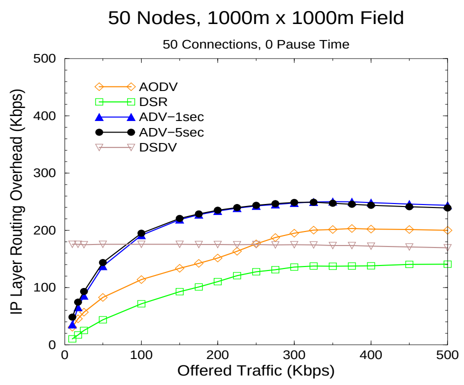


Figure 3.6: IP-layer routing overhead for 50 connections in a 50-node network on a 1000m x 1000m field.

and replies generated during the route discovery and route repair processes, the on-demand protocols generate a large number of routing packets. Compared to ADV, the number of routing packets is nearly twice as high for DSR and almost four times as high for AODV. As noted above, snooping and caching reduce DSR's route discovery overhead compared to AODV.

The picture changes when we consider the volume of routing traffic in bytes. In its attempt to maintain fresh routes to all active receivers, ADV generates larger routing packets than the on-demand protocols, especially at low to moderate loads. However, this view of the routing overhead can be misleading. The cost to acquire the medium to transmit a packet is significantly greater in terms of power and network utilization than the incremental cost of adding bytes to an existing packet. Thus, ADV's low routing overhead in terms of packets is a definite advantage relative to the on-demand protocols.

At the MAC layer, ADV enjoys another advantage in routing overhead. Whereas AODV and DSR both utilize unicast packets to deliver route error and route reply messages, ADV routing information is carried entirely by broadcast packets. Broadcast packets do not incur the added MAC layer overhead of the RTS, CTS, and ACK packets required for unicast packet transmissions. It is their high routing overhead at the MAC layer that causes the on-demand protocols to saturate at lower loads than ADV.

The impact of increasing the number of connections from 25 to 50 is evident when comparing Figures 3.5 and 3.6. In terms of routing packets, DSDV and ADV are unaffected by the increase, although the volume of routing information, as measured in bytes, does rise significantly for ADV. In contrast, the number of

AODV routing packets is 25% higher with 50 connections, while the number of DSR routing packets is up by over 10%. The number of AODV and DSR routing bytes are also much higher, but ADV still produces the largest volume of routing bytes.

ADV vs. DSDV

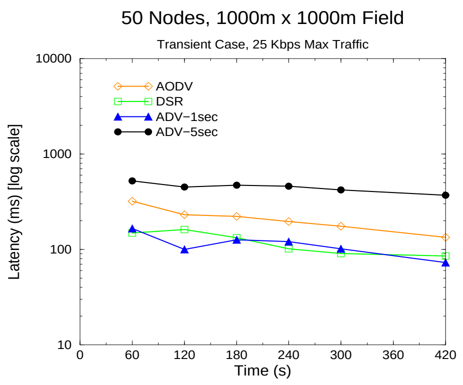
We included DSDV in our performance analysis because we wanted to clearly show the benefits of ADV's adaptive approach to proactive routing. As we have seen, ADV yields significantly higher throughputs and packet delivery fractions than DSDV. At the same time, packet latencies for ADV are comparable to those for the on-demand protocols, DSR and AODV. We believe ADV is clearly the better of the two proactive routing algorithms. Therefore, we did not consider DSDV in subsequent performance analyses.

3.4.3 Transient behavior of a high mobility network

In this section, we examine the transient-state conditions of a high mobility, 50-node network in the 1000m x 1000m square field. Initially, the network traffic is generated by 10 CBR connections. Ten more connections are added every 60 seconds until a total of 50 connections have been started. The simulation continues for two more 60-second intervals during which no additional connections are initiated.

We present two sets of results. In Figures 3.7 - 3.9, the offered traffic is 5 Kbps for each 10 connections, so that the total load is 25 Kbps when all 50 connections have been started. In Figures 3.10 - 3.12, the offered traffic is 20 Kbps for each 10 connections, giving a maximum load of 100 Kbps. Additional transient-state results are included in Appendix A for maximum total loads of 200 Kbps and 300 Kbps.

Referring to Figure 3.7, AODV is quite stable from the outset, showing very little change in packet delivery fraction over time. The delivery fractions for ADV, on the other hand, start out low and do not begin to level off until several 60-second intervals have elapsed. Once 20 connections have been added, ADV with 5-second buffering has the highest delivery fraction, but also the largest packet delays. With a shorter buffer refresh time of 1 second, ADV latencies are as low or lower than those of the on-demand protocols. All three protocols exhibit reduced latencies as connections are added. The packet delays continue to drop (and to a



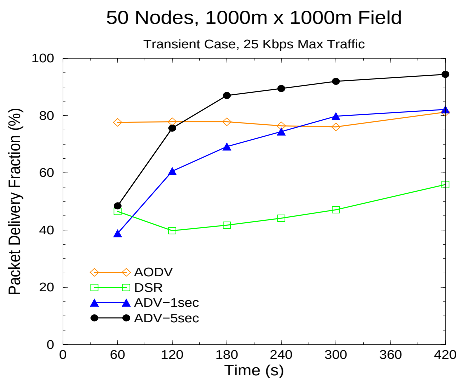


Figure 3.7: Packet latency and delivery fraction for transient case with 25 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

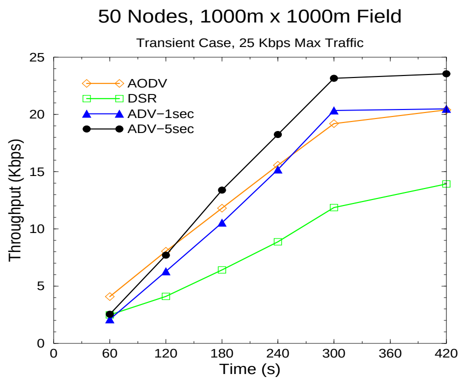
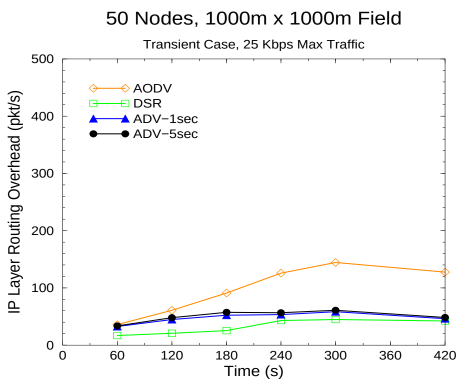


Figure 3.8: Throughput for transient case with 25 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

lesser extent, the delivery fractions continue to increase) over the final two 60-second intervals, suggesting that the protocols have not yet stabilized.

In Figure 3.8, ADV reaches its maximum throughput at the end of the fifth 60-second interval, with almost no change over the remainder of the simulation. On the other hand, the on-demand protocols continue to gain throughput beyond time 300, indicating that AODV and DSR stabilize more slowly than ADV.

The IP-layer routing overhead of all three protocols increases with added connections, although the number of ADV routing packets reaches its maximum once 20 connections have been initiated. Routing overhead declines somewhat for AODV and ADV in the last two 60-second intervals as those protocols stabilize.



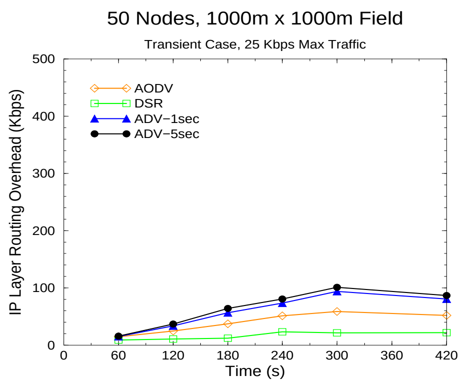
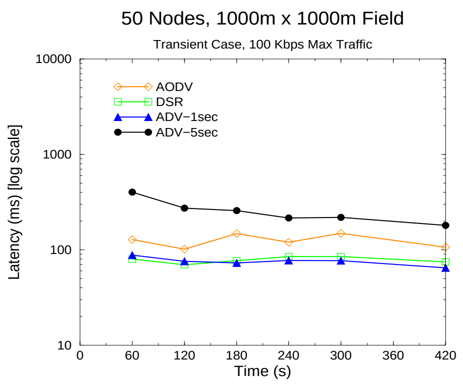


Figure 3.9: IP-layer routing overhead for transient case with 25 Kbps max traffic in a 50-node network on a 1000m x 1000m field.



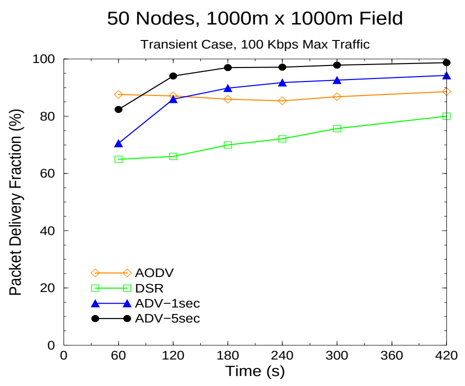


Figure 3.10: Packet latency and delivery fraction for transient case with 100 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

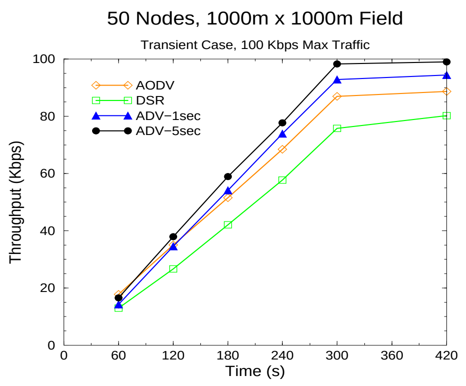
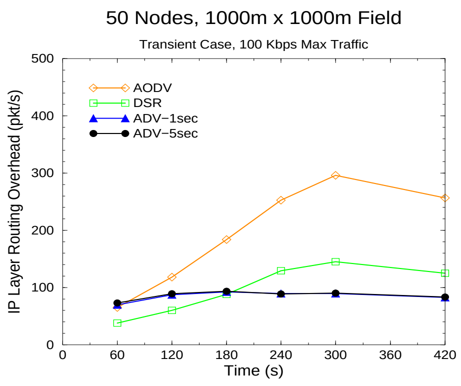


Figure 3.11: Throughput for transient case with 100 Kbps max traffic in a 50-node network on a 1000m x 1000m field.



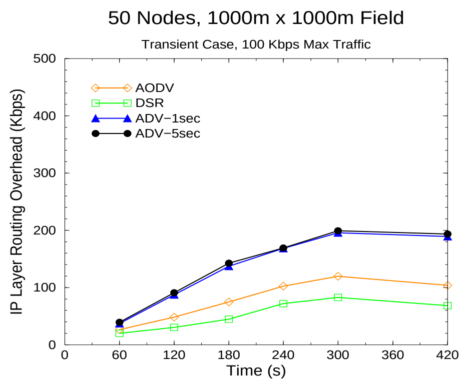


Figure 3.12: IP-layer routing overhead for transient case with 100 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

The story is qualitatively much the same when the maximum traffic load is increased to 100 Kbps. At this higher level of offered traffic, ADV packet delivery fractions stabilize more quickly. As shown in Figures 3.10 and 3.11, ADV with 1-second buffering now outperforms AODV in terms of delivery fraction and throughput.

3.5 Tuning ADV for TCP performance

As we have noted earlier, previous performance analyses of MANET routing protocols have primarily used constant-bit-rate UDP flows for network loads. For this reason, the initial design of ADV was geared toward that type of one-way traffic. A TCP connection, of course, involves a two-way flow – data packets traveling from sender to receiver and acknowledgements traveling in the reverse direction. With this in mind, we modified the ADV connection-initiation process.

Originally, ADV required the intermediate nodes which propagate a *receiver-alert* packet to construct a reverse route to the destination in addition to marking the destination as an active receiver. However, the connection-initiation process is also an opportunity for a node, when it receives an *init-connection packet*, to establish a reverse route to the connection source. If the new connection is a two-way connection, e.g. TCP, the connection source should also be marked as an active receiver. The header prepended to all IP packets includes a field which designates the transport-layer protocol for which a packet is destined. Thus it is straightforward for ADV to determine if a data packet belongs to a TCP flow. Whenever a new route is added to a node's routing table as part of the connection-initiation process, the value of that node's trigger meter is increased to stimulate route updates which will help to propagate the new routing information in the event init-connection packets are lost.

In Section 3.4, we compared the ADV performance attained for two different buffer timeout intervals, 1 second and 5 seconds. Noting the impact of the choice of timeout value, we decided to investigate the effect of the buffer timeout on TCP performance. Based on the results of experiments presented later in Section 4.5, we concluded that using different buffer timeout intervals for TCP (30 seconds) and non-TCP traffic (1 second) improved overall performance.

In the course of our early TCP experiments, we discovered that on occasion route establishment was delayed because ADV attempted to use a stale route, which in turn led to a TCP retransmit timeout. This occurred when the source node had, at an earlier point in time, been a 1-hop neighbor of the destination node and thus a route to the destination, albeit out-of-date, was available for use. The problem is easily solved by ignoring routing table entries for nodes that were not previously active receivers.

Another modification we made to ADV was not strictly TCP-related. When a node receives a route update with a higher sequence number than the one it currently has stored for that destination, or if the distance to the destination is shorter through the source of the route update than the currently stored metric, the routing table is updated with the new information. However, what is the appropriate action to take when the new sequence number and metric do not differ from the stored values? Originally, ADV changed the route's next hop field to point to the node from which the route update was received. The intuition was that the source of the update is known to currently be a 1-hop neighbor, and hence the next hop information is more likely to be useful in the future. However, simulation experience has shown that, in fact, it is counter-productive to do this. Leaving the next hop field alone will generally result in better performance.

3.6 Concluding remarks

The ADV routing protocol starts from a traditional distance vector algorithm, such as that employed in DSDV. Whereas DSDV relies on periodic updates of fixed size to maintain its routing tables, the size and frequency of ADV's route updates are adaptive to network load and node mobility. This adaptivity gives

ADV a significant performance advantage over DSDV in terms of throughput and packet delivery fraction. Although packet latency is higher with ADV than with DSDV, the packet delays for ADV are comparable to the delays for the on-demand protocols.

The routing overhead of the on-demand routing algorithms, which rely on route discovery mechanisms to establish and repair routes, is higher than that of ADV with its proactive route maintenance. ADV's lower overhead results in better performance at moderate to high traffic loads. However, at very low levels of traffic or when the number of connections is small (less than 10), the frequency of ADV route updates is not sufficient to make proactive route maintenance competitive with route discovery. In these cases, AODV and DSR yield superior performance. One of the important design issues in ADV is the tradeoff between the higher overhead of more frequent updates and the increased packet latency that comes with a lower level of routing activity. As it is currently configured, ADV gives its best performance, relative to DSR and AODV, for moderate to high traffic loads in high-mobility networks.

Chapter 4

Improving TCP Performance in Mobile Ad hoc Networks

The performance of transport layer protocols will be a key factor in the successful extension of Internet applications and services to mobile ad hoc networks. The Transmission Control Protocol (TCP) is the de facto standard transport protocol for the Internet, so providing a high level of TCP performance in MANETs is of particular importance. While TCP has been extensively tuned for wireline networks, in its current form TCP does not perform well when used in MANETs. Therefore, one of the main thrusts of this dissertation is to identify ways in which we can increase the performance of TCP in MANETs.

4.1 Techniques for improving TCP performance

We can broadly group techniques for improving TCP performance in MANETs into three categories. This classification is based on which of the layers in the protocol stack are involved, and on what sources of feedback and other pertinent information are used. Our classification scheme is summarized in Table 4.1.

Level	**Type of Feedback**
1	TCP layer only
2	Routing agents at TCP connection endpoints
3	Routing agents at intermediate nodes

Table 4.1: Classification of TCP performance improvement techniques.

- **Level 1: TCP layer.** These mechanisms are implemented in the TCP sender and/or the TCP receiver. No information is required from the routing layer or lower layers in the protocol stack. Existing TCP options, such as selective acknowledgements, are included in this level.
- **Level 2: Routing layer at TCP connection endpoints.** These mechanisms utilize feedback from the routing agents on the hosts where the TCP sender and receiver are running. Such methods may also use the MAC-layer information that is available to these routing agents.
- **Level 3: Routing agents on intermediate nodes.** Mechanisms at this level require feedback supplied by the routing agents running on one or more intermediate nodes along the route established by the TCP connection.

Level 1 solutions have the advantage of being end-to-end mechanisms. The additional complexity of interacting directly with the routing layer or of requiring information from other MANET nodes is avoided. To the extent that level 1 solutions provide comparable benefits to methods requiring information not available at the transport layer, we believe they are to be preferred.

In this section, we elaborate on the framework for TCP performance mechanisms outlined above, and we show where in this hierarchy existing methods and current research fit. We then describe two novel layer 1 techniques that we have designed and implemented.

4.1.1 TCP layer mechanisms

Even though MANET routing protocols are designed to repair broken routes quickly, route failure is often a source of packet delays and packet drops. Significant reordering of packets may occur as well. This suggests that mechanisms designed to avoid unnecessary retransmit timeouts, packet retransmissions, and reductions in the congestion window, may be promising means of improving TCP performance. TCP layer methods will generally require changes to the TCP sender, the TCP receiver, or both. However, existing TCP options may also offer performance benefits, so the use of these mechanisms should be considered, too.

Using existing TCP options

In this section, we discuss two existing TCP options that may be used to improve TCP performance in MANETs, selective acknowledgements and delayed acknowledgements.

Selective acknowledgements: In the Reno version of TCP, a duplicate ACK does not tell the sender which packet or packets are still outstanding, only that one or more packets have been dropped or delayed. The best the sender can do is retransmit at most one missing packet per round trip time (RTT) or risk retransmitting a packet that may already have been received. The selective acknowledgements (SACK) option allows the receiver to specify which non-consecutive packets have been received. Frequent route changes in a MANET can be expected to cause packet losses, and these losses may result in duplicate ACKs that trigger the TCP sender's fast retransmit mechanism. By enabling the sender to more accurately infer which packets are missing, SACK should reduce the number of unnecessary packet retransmissions.

Delayed acknowledgements: In networks where bandwidth is a scarce commodity, it makes sense to reduce traffic whenever possible. The use of delayed ACKs is expected to help by reducing the volume of ACK traffic in normal network conditions. However, the real benefit of delayed ACKs should come when routes break. During route reconstruction, data packets may be sitting in a queue at the source or at an intermediate node. If the route is repaired quickly enough that the data packets have not yet been flushed from the buffer, then all of these packets will arrive at the TCP receiver in quick succession. With delayed ACKs, the receiver sends fewer acknowledgements for these packets. This will, in turn, enable the TCP sender to increase its congestion window more quickly and retransmit fewer packets.

4.1.2 Routing layer mechanisms

These techniques utilize information not directly available to TCP. Level 2 methods do not require any sort of signaling or interaction with intermediate nodes, so they should generally be somewhat less complex than level 3 mechanisms.

Using feedback from routing agents at TCP connection end points

The routing agent running on the host where the TCP sender resides can be an immediate source of feedback to the sender. Likewise, the TCP receiver can get routing layer information directly from the agent running on its host. Here we give some examples of possible routing layer feedback.

Feedback to the TCP sender: When the sender's routing agent is unable to deliver a TCP data packet because the first link in the route to the destination is down, the agent can inform the sender of the route failure immediately. This is a "cheap" form of explicit link failure notification in that there are no route error packets involved. Similarly, when the routing agent learns through an update that the route has been re-established, this information can be passed directly to the sender.

Another useful item of information the routing agent has is the number of packets in the routing agent's send buffer waiting for routes. The number of packets in the MAC interface queue is also of interest. If the number of packets queued is high, the TCP sender may elect to reduce its sending rate until the level of buffering has dropped off.

Routing packets through a MANET is a store-and-forward process, so the number of packets that can profitably be in flight in a MANET is to some degree a function of the path length from sender to receiver. Therefore, the number of hops to the destination may be of interest to the TCP sender if that information can be used to set a reasonable size for the congestion window.

Feedback to the TCP receiver: If the TCP receiver can determine current conditions in the network from the "experience" of the data packets that have arrived recently, it can relay this information back to the TCP sender. Based on the earlier work on the DECbit scheme [58], the addition of Explicit Congestion Notification (ECN) to TCP was proposed for this purpose [23]. If a router along the path to the destination is congested or is close to congestion, it sets the ECN bit in the header of a data packet. The receiver echoes the notification to the sender in an ACK. This mechanism allows TCP to perform congestion control proactively.

Using feedback from routing agents at intermediate nodes

This is the level at which the solutions such as ELFN and TCP-F, described in Section 2.2.3, fit in our hierarchy. These methods rely on route failure (and route re-establishment) notifications from intermediate nodes to reach the sender, which then modifies its behavior accordingly. Another level 3 approach to improving TCP performance might involve using knowledge of routing agent or MAC layer queue lengths at intermediate nodes, for example to adjust TCP's sending rate before congestion occurs.

4.1.3 Proposed TCP sender side options

The TCP sender is designed to adapt to network conditions by tracking the acknowledgements returned by the receiver. The timing of these ACKs and the order in which they are received are clues that the sender uses to infer what is occurring elsewhere in the network. How these clues are interpreted by the sender has a major impact on TCP performance. In MANETs, new heuristics can be used to elicit sender behavior that is conducive to good TCP performance. We have designed and implemented two mechanisms that use sender-based heuristics: the hold-down timer and the fixed RTO.

Hold-down timer

Sometimes the packet delay, as opposed to packet loss, due to a route failure can cause an ACK to arrive so late that the retransmit timer has already expired. In the description of the delayed acknowledgements option above, we noted that if route repair happens quickly, then any data packets that were buffered at the source or at an intermediate node will arrive at the receiver in rapid order. The same thing can happen for ACKs buffered during route repair. Once the route has been re-established, these ACKs will arrive at the TCP sender in quick succession. If the retransmit timer has expired, the packet corresponding to the first of the tardy ACKs will already have been retransmitted. If the ACKs arrive in order, the first one will trigger the retransmission of the packet corresponding to the second tardy ACK, the second one will trigger yet another retransmission, and so on.

In order to address this problem, we designed a method which utilizes a hold-down timer. Figure 4.1 illustrates the technique. When a timeout occurs, the length of this timer is set equal to the retransmit timeout

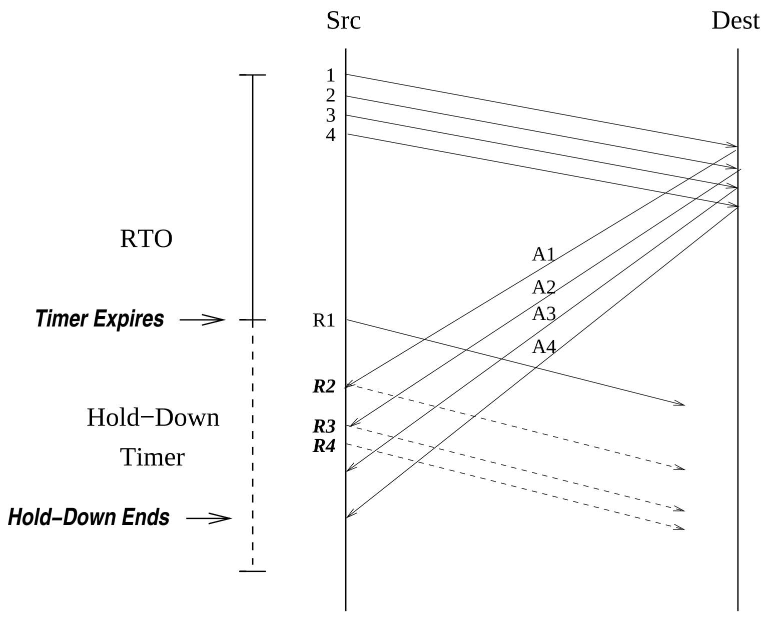


Figure 4.1: TCP data packet/ACK transmission diagram showing tardy ACKs arriving during the hold-down timer interval. The retransmit timer expires before the ACK for packet 1 arrives, so packet 1 is retransmitted (the event labeled **R1**). In normal TCP operation, the arrival of the ACK for packet 1 will trigger the retransmission of packet 2 (event ***R2***), and the arrival of the ACK for packet 2 will trigger the retransmission of packets 3 and 4 (events ***R3*** and ***R4***). With the hold-down timer, the tardy ACK for packet 1 does not cause packet 2 to be retransmitted. Similarly, the late-arriving ACK for packet 2 does not cause packets 3 and 4 to be retransmitted. Instead, since the ACKs for packets 2 - 4 are all received before the hold-down timer expires, no further packet retransmissions occur.

interval (RTO) prior to the timeout. Since the RTO is doubled following a timeout, the hold-down period is one-half of the new timeout period. The first packet in the window is immediately retransmitted as usual. During the hold-down interval, any incoming ACKs are processed in the normal way except that they do not trigger further packet transmissions. If the hold-down timer expires before the first packet in the window has been acknowledged, then TCP reverts to its normal behavior during the second half of the retransmit timeout period, i.e. packets will be retransmitted when the outstanding ACK is received. Otherwise, any packets that are ACKed during the hold-down interval will not have to be retransmitted. If all the packets in the window are ACKed before the hold-down interval ends, the hold-down timer is cancelled and normal TCP operation is resumed.

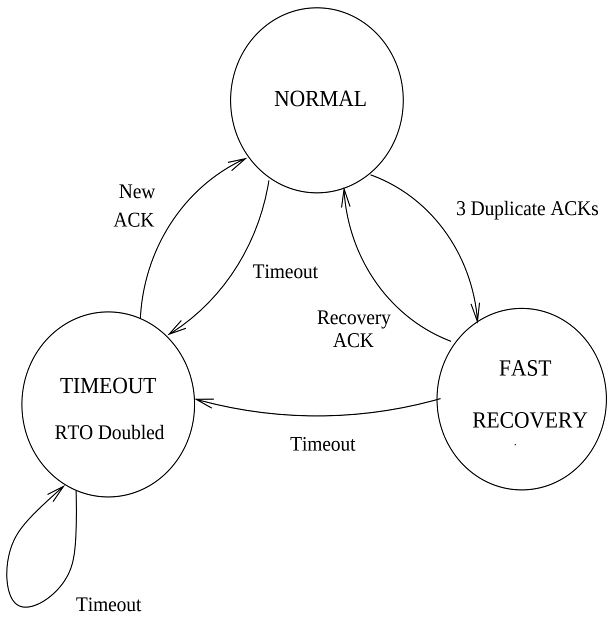


Figure 4.2: Simplified TCP state diagram illustrating normal operation of the TCP Reno protocol after the connection has been established. The RTO is doubled on the first timeout and every consecutive timeout thereafter.

Fixed retransmit timeout interval

In the event of a retransmit timeout, TCP retransmits the oldest unacknowledged packet and doubles the retransmit timeout interval (RTO). This process is repeated until an ACK for the retransmitted packet has been received. This exponential backoff of the RTO enables TCP to handle network congestion gracefully. However, in a MANET, the loss of packets (or ACKs) may be caused by temporary route loss as well as network congestion. Since routes are likely to be broken frequently in high node mobility environments, routing algorithms for MANETs are designed to repair broken routes quickly. To take advantage of this capability, it is intuitive to let a TCP sender retransmit the unacknowledged packet at periodic intervals rather than having to wait increasingly long periods of time between retransmissions.

Therefore, we modified the TCP sender, employing a heuristic to distinguish between route failures and congestion without relying on feedback from other network nodes. When timeouts occur consecutively, i.e. the missing ACK is not received before the second RTO expires, this is taken to be evidence of a route loss. The unacknowledged packet is retransmitted again but the RTO is not doubled a second time. The RTO

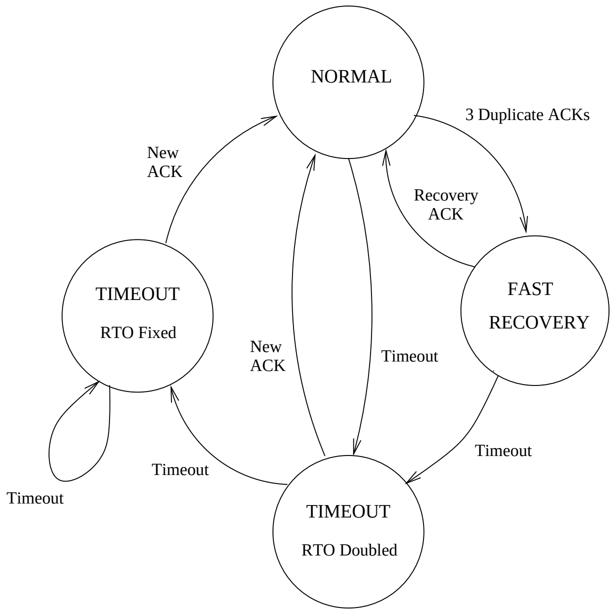


Figure 4.3: Simplified TCP state diagram illustrating the fixed-RTO protocol modification. The RTO is doubled on the first timeout, but remains fixed on succeeding consecutive timeouts.

remains fixed until the route is re-established and the retransmitted packet is acknowledged. The simplified state diagrams shown in Figures 4.2 and 4.3 highlight the change to the sender.

4.2 TCP performance analysis

In this section, we present the results of a simulation-based comparison of TCP performance over three MANET routing protocols: the on-demand DSR and AODV protocols, and the hybrid ADV protocol. The objectives of this study were two-fold. In addition to evaluating the relative performances of these routing protocols, we wanted to test the effectiveness of our proposed sender-side heuristics. We varied the number of TCP connections, over each of which an FTP file transfer was conducted, and we included a background UDP traffic load from multiple constant-bit-rate (CBR) sources.

In the sections which follow, we describe the simulation methods we employed, and we define our performance metrics. We then present the results of the study, and we give a detailed analysis of our observations.

4.2.1 Experimental methods

The simulation environment was as described in Section 3.4.1. All routing protocol parameters were as given in Tables 3.6, 3.7, and 3.8, with the exception that the ADV buffer timeout used was 30 seconds. We simulated a network of 50 nodes moving in a 1000m x 1000m square field. Background traffic loads were generated by 10 and 40 CBR connections, and the CBR packet sizes were fixed at 512 bytes. After a warm-up time of 100 seconds, one or more TCP connections were established over each of which an FTP file transfer was conducted for 900 seconds. The TCP packet size was 1460 bytes, and the maximum size of both the send and receive windows was 8. Since routing protocol performance is sensitive to movement patterns, 50 different mobility patterns (scenarios) were generated and performance results were averaged over these scenarios.

In each simulation run, we measured connect time, throughput, and goodput. *Connect time* is the time it takes to deliver the first TCP packet. Short connect times are important for some types of TCP traffic such as HTTP. *Throughput* is computed as the amount of data transferred by TCP divided by 900 seconds, the time interval from the end of the warm-up period to the end of the simulation. This does not include redundant packet receipts due to unnecessary packet retransmissions and packet replication in the network. *Goodput* is the ratio of TCP packets successfully delivered to the total number of TCP packets transmitted. We measured routing protocol overhead in packets per second and bytes per second at the IP layer as described in Section 3.4.1. We also measured MAC-layer routing overhead, defined as the total number of routing packets transmitted per second at the MAC layer. This includes the RTS, CTS, and ACK packets as well as the IP layer routing packets.

4.2.2 Performance results for 1 TCP connection

We performed a series of five simulation runs. Each simulation run tested a different technique or combination of techniques: TCP Reno, Reno with SACK, Reno with SACK and delayed ACKs, fixed RTO on consecutive timeouts plus SACK and delayed ACKs, and a hold-down timer plus fixed RTO and SACK and delayed ACKs. In each run, a set of performance measurements were made for each of the three routing protocols at each of several background traffic loads from 10 CBR and 40 CBR connections.

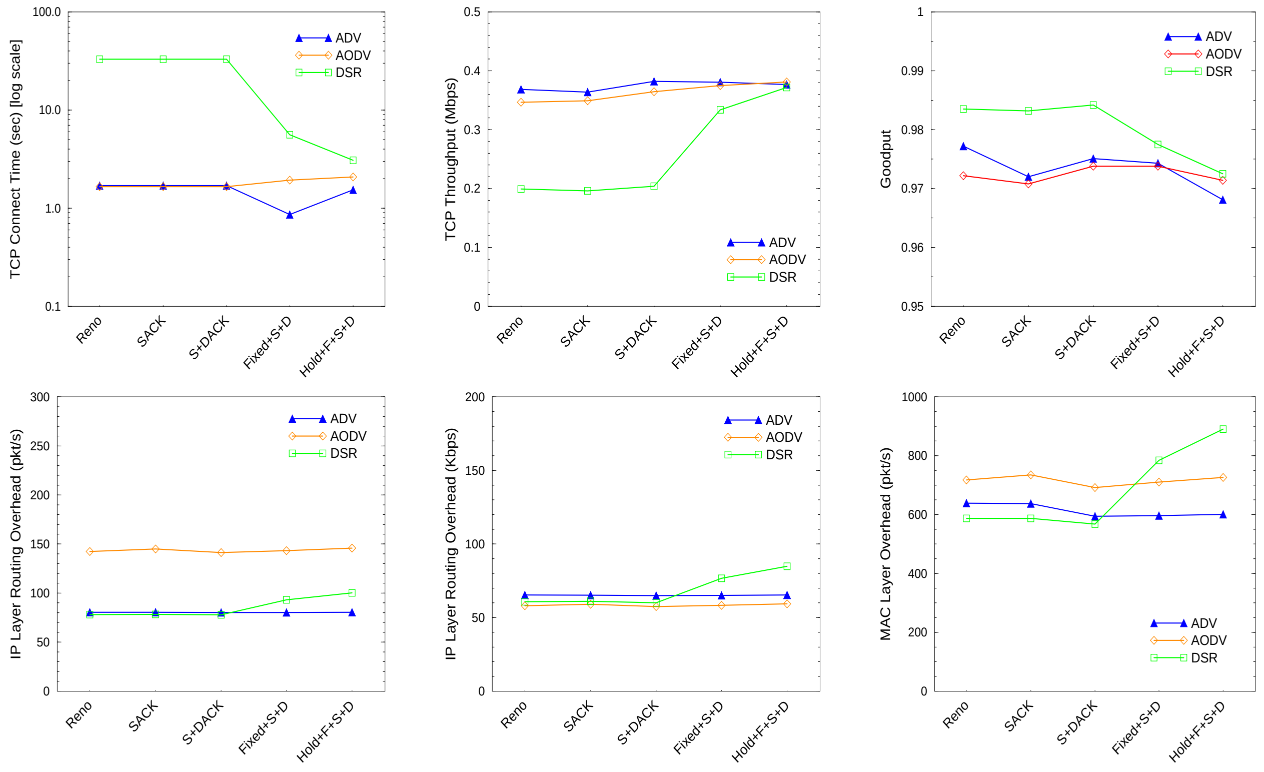


Figure 4.4: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 50 Kbps background load from 10 CBR connections.

Figures 4.4 and 4.5 show the connect times, throughputs, goodputs, and routing overheads, averaged over the 50 scenarios, observed for each of the protocols for a 50 Kbps background traffic load generated by 10 and 40 CBR connections. (Results for background loads of 100 Kbps and 200 Kbps are given in Appendix B.) Tables 4.2 and 4.3 show the throughput results expressed as percent changes relative to the baseline TCP Reno results. To compare the results from any two combinations of techniques, we paired the observed throughputs attained by these methods for each of the 50 scenarios and applied the paired t-test. Differences shown in bold print have 95% confidence intervals that do not include zero.

DSR results

For the TCP Reno baseline with the DSR routing algorithm and a background traffic load of 50 Kbps from 10 CBR connections, the throughputs we observed varied from 0.05 to 0.5 Mbps. This is less than the range of approximately 0 to 0.9 Mbps reported by Holland et al. [29] for a mean node speed of 10 m/s. In

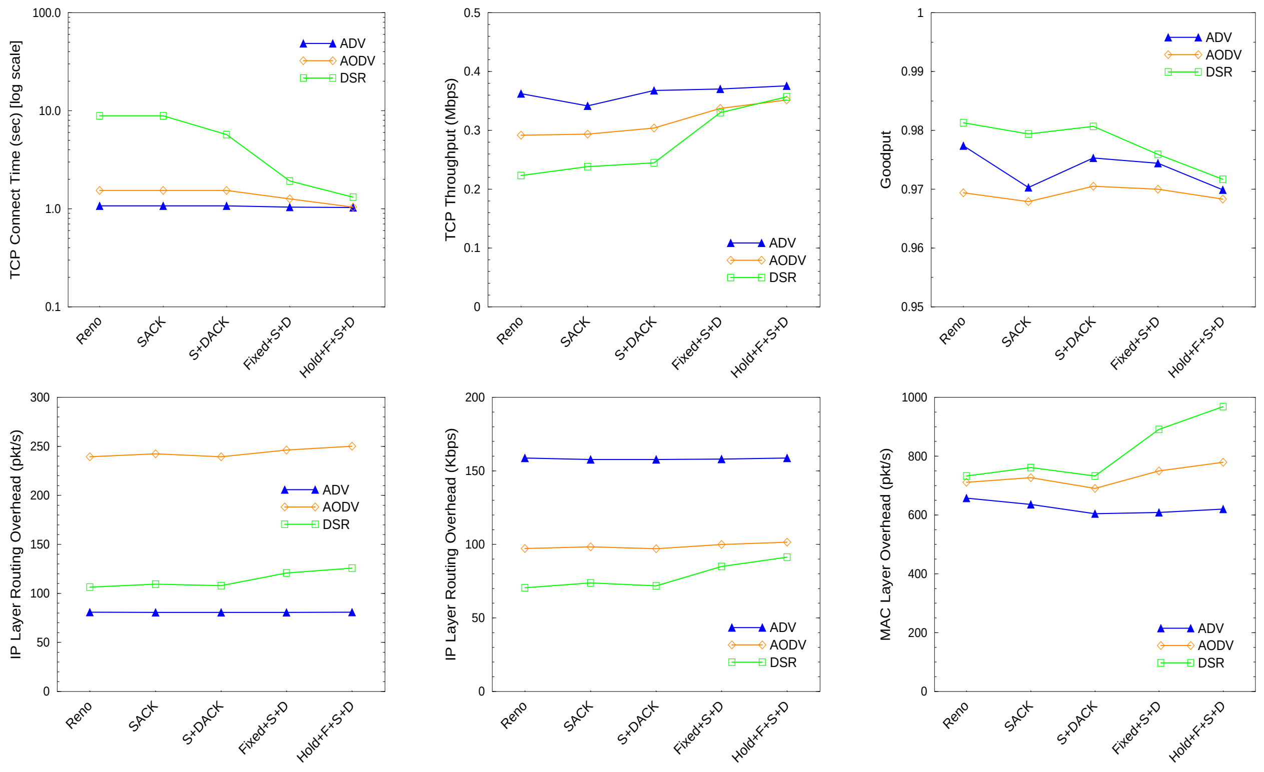


Figure 4.5: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 50 Kbps background load from 40 CBR connections.

Table 4.2: Percent increases in throughput relative to TCP Reno for 50 Kbps 10-CBR background.

Simulation Run	**ADV**	**AODV**	**DSR**
SACK	-1.3	0.7	-1.6
SACK + delayed ACKs	**3.7**	**5.1**	2.3
SACK + fixed RTO + delayed ACKs	**3.3**	**8.1**	**67.4**
SACK + fixed RTO + hold-down timer + delayed ACKs	2.2	**9.9**	**86.4**

Table 4.3: Percent increases in throughput relative to TCP Reno for 50 Kbps 40-CBR background.

Simulation Run	**ADV**	**AODV**	**DSR**
SACK	**-5.7**	0.6	**6.7**
SACK + delayed ACKs	1.6	4.1	**9.6**
SACK + fixed RTO + delayed ACKs	2.2	**15.6**	**47.8**
SACK + fixed RTO + hold-down timer + delayed ACKs	**3.7**	**20.4**	**59.9**

their study, however, there was no competition for bandwidth from background traffic. Furthermore, their mobility scenarios had a higher node density and fewer route failures. For some scenarios, we observed connect times in excess of 3 minutes. Large connect times were also observed by Holland et al., who described a scenario in which throughput was zero over a 2 minute interval.

Referring to the throughput results for the 50 Kbps 10-CBR background shown in Figure 4.4, the most marked change was the increase observed for DSR. In the TCP Reno run, DSR throughput was about 55% that of the other protocols. With a lighter background traffic load, routes in the network are more likely to be stale, and stale routes are troublesome for DSR. The stale route problem is very evident when we consider the average connect times in Figure 4.4. Comparing the base case to the run in which we fixed the retransmit timeout interval in the event of consecutive timeouts, we see that the connect time for DSR dropped dramatically from over 30 seconds to about 5 seconds. At the same time, DSR throughput increased by 67%. The fixed-RTO technique continued to yield a 70-75% gain in DSR throughput as the 10-CBR traffic increased from 50 Kbps to 200 Kbps as shown in Figure 4.7. Significant throughput gains were also observed for the 40-CBR case as shown in Figure 4.5. The goal of fixing the RTO was to reduce the impact of route unavailability. It appears this technique was particularly effective in mitigating the stale route problem for DSR.

Adding the hold-down timer to the other techniques provided an additional increase in throughput. For 10 CBR connections, an additional gain of 20% was observed for all traffic loads, while in the 40-CBR case the gain was around 12% for a 50 Kbps load, decreasing to about 5% for a 200 Kbps load.

AODV results

The fixed-RTO technique, combined with SACK and delayed ACKs, also yielded increased throughput for AODV, although the gains were much smaller than those observed for DSR. For the 10-CBR background, the gain in throughput was approximately 8%. Interestingly, the increase in throughput for the 40-CBR case was about twice as large, nearly 16%. As Figures 4.4 and 4.5 show, the number of AODV routing packets increased nearly 70% when the number of CBR connections was increased from 10 to 40. To the extent that this additional routing traffic results in increased packet delays, we would expect the fixed-RTO technique to be of relatively greater benefit.

In Figures B.1 through B.4 in Appendix B, we see that as the background traffic load was increased, the gain in throughput remained the same, about 8%, for 10 CBR connections, while the gain in the 40-CBR case grew to 48% for a 200 Kbps load.

The hold-down timer yielded only marginal improvements in throughput in most cases. However, in the case of a 200 Kbps 40-CBR load, a gain of 15% in throughput was observed.

ADV results

The increases in throughput that we observed for ADV did not exceed 4%. It appears ADV was performing as well as possible because, although the application of these techniques tended to minimize the performance differences among the protocols, in no case did the other protocols exhibit a higher level of throughput than ADV. Furthermore, in the 40-CBR case, ADV clearly outperforms AODV and DSR regardless of which techniques are used. We observed this same result at higher background traffic loads. As shown in Figure B.2, ADV throughput was 17% to 52% greater than that of the other protocols for a 100 Kbps load and 40 connections.

Goodput and routing overhead

All three protocols exhibited very high goodputs, ranging from 97% to 98%. It is noteworthy that the proposed fixed-RTO technique did not significantly lessen the observed goodputs.

The routing overhead for DSR increased significantly in response to the fixed-RTO and hold-down timer techniques. A 50% increase in the MAC layer overhead was observed for DSR in the 10-CBR case. The increased overhead is an expected result of the large gains in throughput seen for DSR. The routing overheads for ADV and AODV, on the other hand, were not affected by the different techniques.

In earlier experiments not included in this work, we have observed that DSR and AODV seem to have comparable MAC layer overheads for CBR traffic. However for TCP traffic, both AODV and ADV have lower overhead.

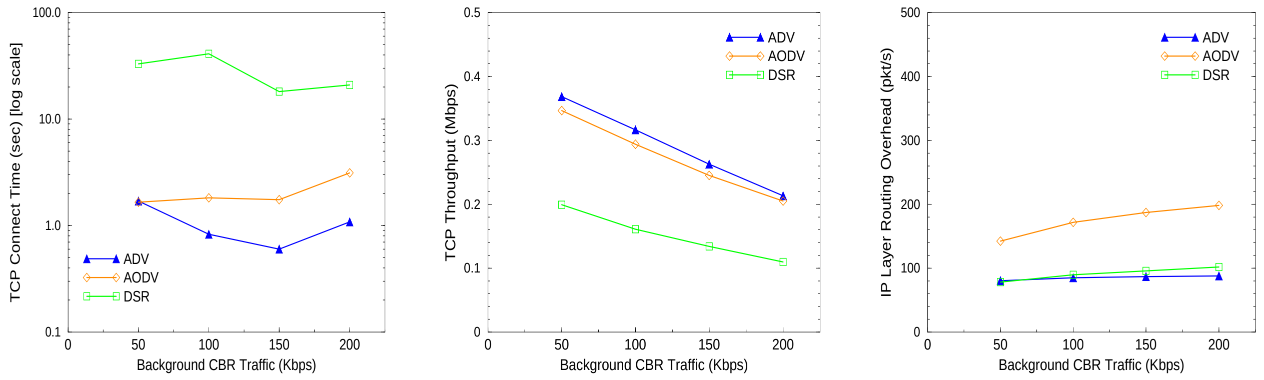


Figure 4.6: Connect times, throughputs, and routing overhead for 1 TCP Reno connection with a 10-CBR background.

4.2.3 Comparison of different TCP options

The use of TCP's selective acknowledgements and delayed acknowledgements options provided modest performance gains, with DSR benefiting the most for a 50 Kbps background traffic load. Throughput gains of 10% to 12% were observed for AODV and DSR at higher traffic loads. Adding the fixed-RTO technique to TCP Reno, along with the SACK and delayed ACK options, yielded the majority of the performance gains that were achieved. While the hold-down timer mechanism provided an additional increase in performance for 1 TCP connection, this benefit was not observed in our initial simulations with multiple TCP connections. Therefore, in the remainder of this chapter we will focus on the performance benefits of combining TCP Reno with fixed RTO, SACK, and delayed ACKs. For convenience, we will refer to this combination as TCP Reno-F.

Figures 4.6 and 4.8 show the connect times, throughputs, goodputs, and routing overheads, averaged over the 50 scenarios, observed for each of the protocols for 1 TCP Reno connection with background traffic loads of 50, 100, 150, and 200 Kbps generated by 10 and 40 CBR connections. Figures 4.7 and 4.9 give the results for 1 TCP Reno-F connection.

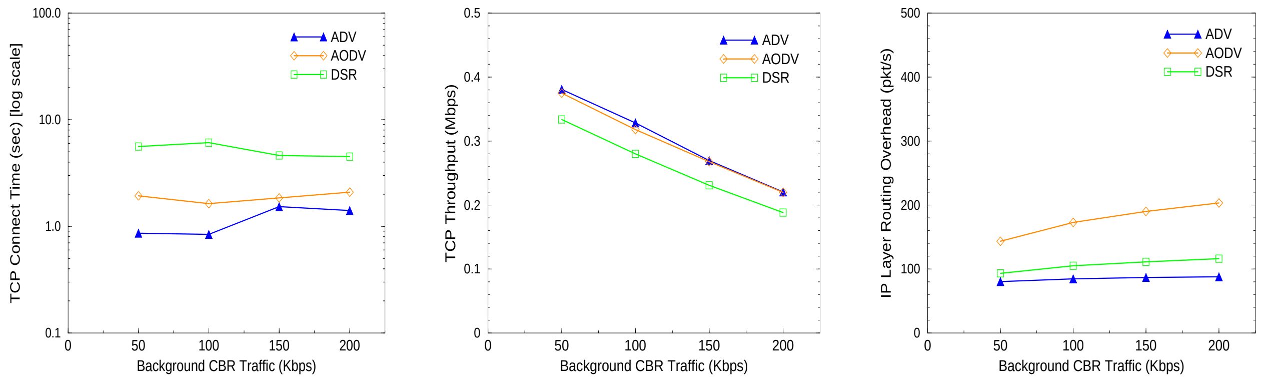


Figure 4.7: Connect times, throughputs, and routing overhead for 1 TCP Reno-F connection with a 10-CBR background.

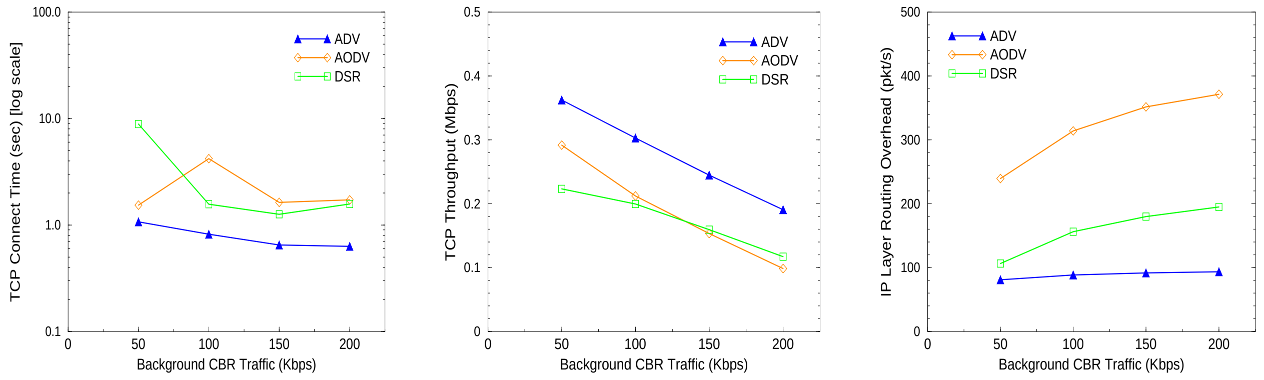


Figure 4.8: Connect times, throughputs, and routing overhead for 1 TCP Reno connection with a 40-CBR background.

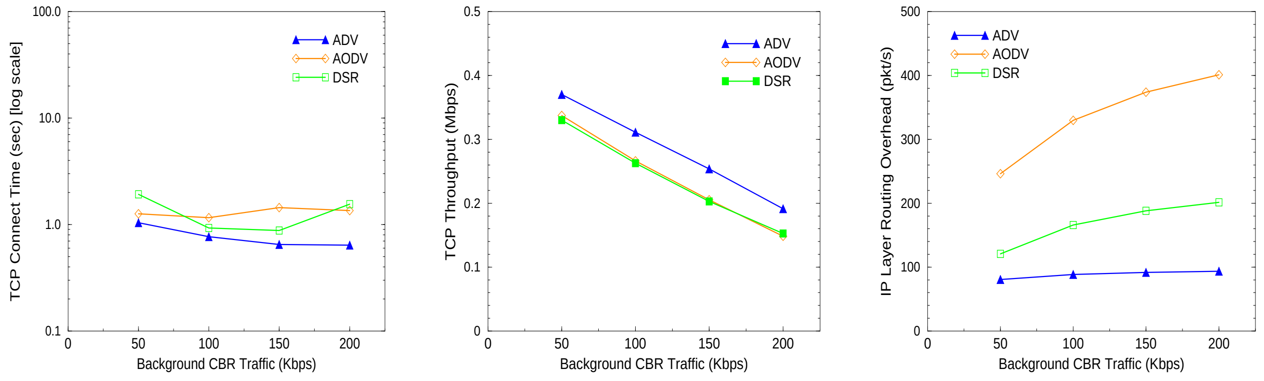
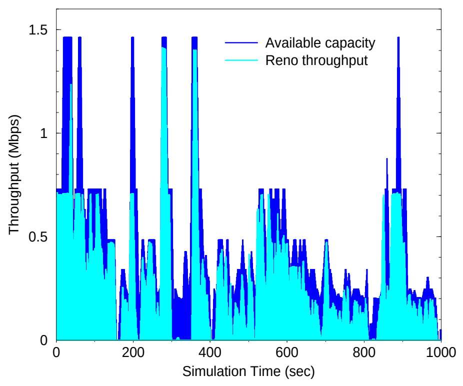


Figure 4.9: Connect times, throughputs, and routing overhead for 1 TCP Reno-F connection with a 40-CBR background.



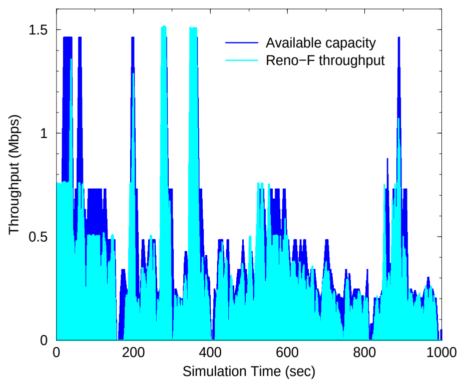


Figure 4.10: Throughput compared to capacity for a TCP Reno and TCP Reno-F connection with the AODV routing protocol and no background traffic. Capacity = 0.486 Mbps. Reno throughput = 0.308 Mbps. Reno-F throughput = 0.361 Mbps.

4.3 TCP Reno-F

The analysis in Section 4.2 has shown that the combination of TCP's selective acknowledgements and delayed acknowledgements options with our proposed sender-side heuristic, fixed RTO, can yield substantial improvements in TCP performance. We call this combination of performance-enhancement techniques the TCP Reno-F protocol. While no TCP variant will succeed in utilizing all the available capacity of a MANET, the Reno-F protocol does a better job of adapting a TCP flow to the changes in a MANET's capacity than does TCP Reno.

In this section, we take a closer look at Reno-F. For this analysis, we utilized two mobility scenarios from Section 4.2 – one for which the TCP Reno performance observed for each routing protocol was average, and another for which the routing protocols yielded the worst Reno performance. Using the average scenario, we show how the fixed-RTO technique works to improve TCP performance. We also discuss various factors which determine the effectiveness of fixed RTO. In particular, we utilize the worst-case scenario to explain why fixed RTO improves the performance of an on-demand protocol such as AODV, but provides little or no benefit when a proactive protocol like ADV is used.

4.3.1 Average case with no background traffic

The throughputs shown in Figure 4.10 were observed for a single TCP connection for the average mobility scenario. Because the benefit of Reno-F is generally greater for an on-demand protocol, AODV was chosen to make this example clearer. To keep the analysis simple, no background network traffic was included and the TCP connection was started at the beginning of the simulation. The expected throughput for this particular connection and mobility scenario was computed on the basis of the hop count between sender and receiver, as explained in Section 1.3. In this example, Reno-F yields a 17% increase in TCP throughput.

Figures 4.11 and 4.12 show the congestion window sizes and route repair times observed for the TCP connection over the course of this scenario. The upper graph shows the congestion window size as a function of time along with two sets of hash marks; the upper (darker) hash marks denote retransmissions and the lower (lighter) hash marks denote transmission of a new packet by the sender. The plotted window size is a 5-second moving average, so non-integral window sizes appear. A 5-second interval was chosen to smooth the plot without losing important detail. The lower graph shows the route repair times observed during the simulation; the horizontal dotted line indicates the average route repair time.

When a route failure interrupts a TCP flow, it is important that the TCP sender learn that the route has been repaired as soon as possible so as to minimize the duration of the interruption. As noted in [45], when multiple consecutive retransmissions of the same packet are lost because the sender and receiver are temporarily disconnected, the doubling of the interval between retransmissions can result in extended periods of inactivity even after the connection has been restored. Referring to Figure 4.11, at approximately time 300, a lengthy route failure causes the TCP Reno sender to stall. The congestion window size falls to its minimum value of 1 and TCP experiences a series of retransmit timeouts which occur at increasingly wider intervals. Packet flow is not re-established for some 50 seconds. In Figure 4.12, the route failure results in more frequent packet retransmissions, enabling the TCP Reno-F sender to recognize the repaired route and resume packet flow quickly.

Thus, an important benefit of the fixed-RTO technique is that it allows the TCP sender to avoid waiting for ever longer periods of time before attempting to retransmit the next unacknowledged packet. Because

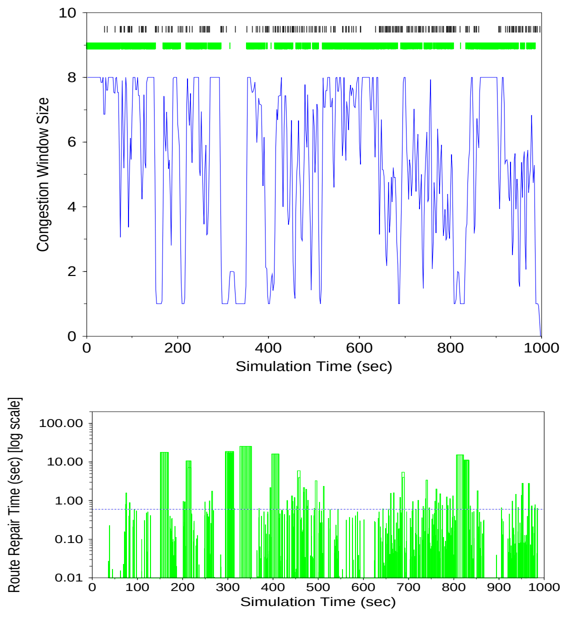


Figure 4.11: AODV congestion window sizes and route repair times for 1 TCP Reno connection with no background traffic load. Average route repair time = 0.597 seconds.

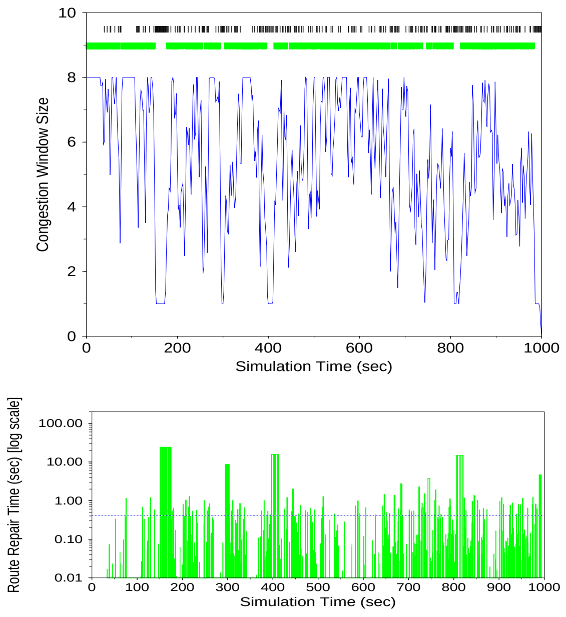


Figure 4.12: AODV congestion window sizes and route repair times for 1 TCP Reno-F connection with no background traffic load. Average route repair time = 0.405 seconds.

Reno-F fixes the interval between retransmissions, the TCP sender is probing the network in a manner similar to the ELFN scheme [29]. But because the probe interval is equal to the retransmit timeout interval, which is based on the estimated RTT, the fixed-RTO method is inherently adaptive.

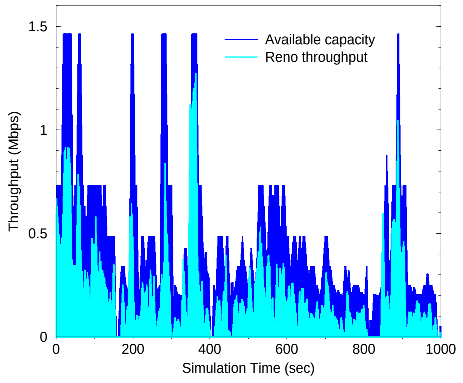
Another benefit of more frequent packet retransmissions is the potential for reducing average route repair time by stimulating the route discovery mechanism of an on-demand protocol. As shown in Figure 4.12, AODV is able to repair routes more quickly with Reno-F. In this example, the average time taken by AODV to re-establish a broken route is reduced by 32% and there are fewer extended route failures. In particular, we see that the route repair delay at time 300 is much shorter as a result of the increased rate of retransmissions.

While TCP throughput is higher overall with Reno-F in this example, there are intervals during the simulation run for which TCP Reno is achieving a higher throughput. It is instructive to consider the reasons for this. Referring to Figure 4.10, consider the interval from about time 80 to 110. In this instance, Reno-F sustains a throughput of approximately 500 Kbps, but Reno is able to reach throughputs of around 700 Kbps for much of the period. Looking at the available capacity shown in Figure 4.10 and referring to Table 1.1, we deduce that the shortest path between the TCP sender and receiver was two hops in length during this period.

In both the Reno and Reno-F runs, AODV is using a 3-hop route at time 80. However, because the precise sequence of route discoveries during the first 80 seconds of the simulation run is different for Reno and Reno-F, these paths are not identical. The route that Reno-F is using at time 80 is quite stable; it does not fail until time 110. The Reno route is unstable, however, and it breaks at time 82. AODV repairs the broken route, replacing it with a new route of the optimal length 2. Although additional failures of the Reno route occur before time 110, in each instance the broken route is replaced by another path of length 2. So, TCP throughput is higher for Reno than for Reno-F during this period of time. This is an example of how the properties of the routing protocol being used can limit the effectiveness of a transport layer protocol. In this instance, the fact that AODV fails to use the shortest possible route available to it in the Reno-F run is a routing protocol issue, and not a reflection on Reno-F.

4.3.2 Average case with background traffic

We take our analysis of this example a step further by adding a 100 Kbps background traffic load from 10 CBR connections. As shown in Figure 4.13, the TCP connection is not able to use as much of the available capacity when other network traffic is present. Nevertheless, Reno-F still provides a throughput gain of about 16%. The reduction in the average route repair delay is more modest than in the no-background case, however, about 17%. This is to be expected because the additional routing activity associated with the background traffic means that more routing information is being propagated through the network via route requests and replies, and this is likely to reduce the route discovery burden for the TCP connection. As a result, route repair delays will be shorter and there will be less room for improvement. For example, when we compare the Reno runs in Figures 4.10 and 4.13, we see that the extended period of almost no throughput from around time 300 to 350 in the no-background case is eliminated when the CBR traffic is added. It should be noted that, although TCP throughput increases are associated with reduced route repair times, it is not possible to predict the magnitude of a throughput increase from the corresponding reduction in route repair delay, and vice versa.



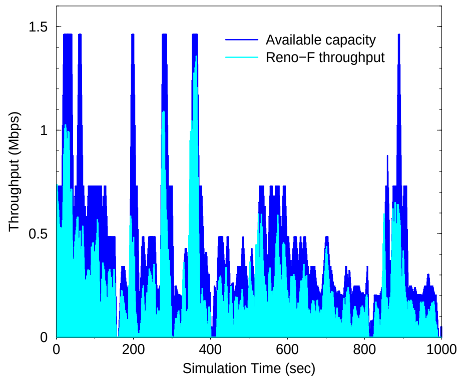


Figure 4.13: Throughput compared to capacity for a TCP Reno and TCP Reno-F connection with the AODV routing protocol and a 100 Kbps background load from 10 CBR connections. Capacity = 0.486 Mbps. Reno throughput = 0.221 Mbps. Reno-F throughput = 0.257 Mbps. Average Reno route repair time = 0.304 seconds. Average Reno-F route repair time = 0.251 seconds.

4.3.3 Worst case

To illustrate how Reno-F can impact the TCP performance of AODV while providing only a modest benefit to ADV, we consider the worst-case mobility scenario. In this scenario, the length of the shortest possible path between the TCP sender and receiver nodes changed fairly frequently and tended to be a bit long, often 5 or 6 hops or more. There was a 50 Kbps background network load from 40 different CBR flows, and the single TCP connection was established after warming up the network for 100 seconds.

In Figure 4.14, at approximately time 375 a route failure occurs causing the congestion window size to drop to its minimum value of one. Retransmit timeouts follow at progressively longer intervals until the maximum of eight backoffs is reached. The upper hash marks indicate that the sender is probing the network at increasingly longer intervals; the absence of hash marks in the lower set indicates the lack of progress by the TCP sender. The route is not successfully re-established until about time 600, and during this period the observed route repair times shown in the lower graph are generally much longer than those before the route failure at time 375. Longer route repair times and fewer attempts to utilize a repaired route before node mobility breaks it again cause the TCP sender to be stuck in retransmission mode as indicated by a congestion window of size 1 and the big gap in the lower hash marks.

With the use of Reno-F in Figure 4.15, packet retransmissions due to timeouts are more frequent, which

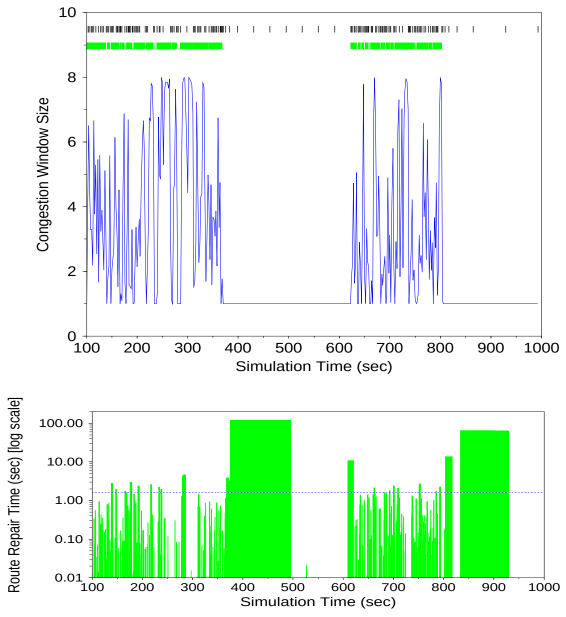


Figure 4.14: AODV congestion window sizes and route repair times for 1 TCP Reno connection with a 50 Kbps background load from 40 CBR connections. Average route repair time = 1.627 seconds. Throughput = 0.0914 Mbps.

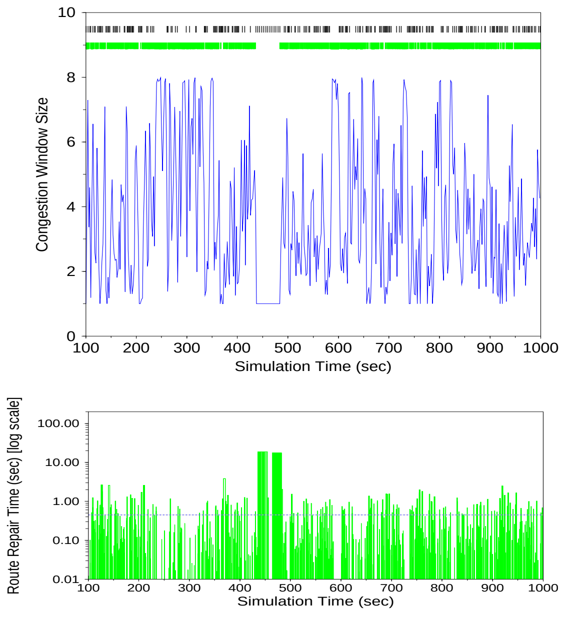


Figure 4.15: AODV congestion window sizes and route repair times for 1 TCP Reno-F connection with a 50 Kbps background load from 40 CBR connections. Average route repair time = 0.446 seconds. Throughput = 0.2023 Mbps.

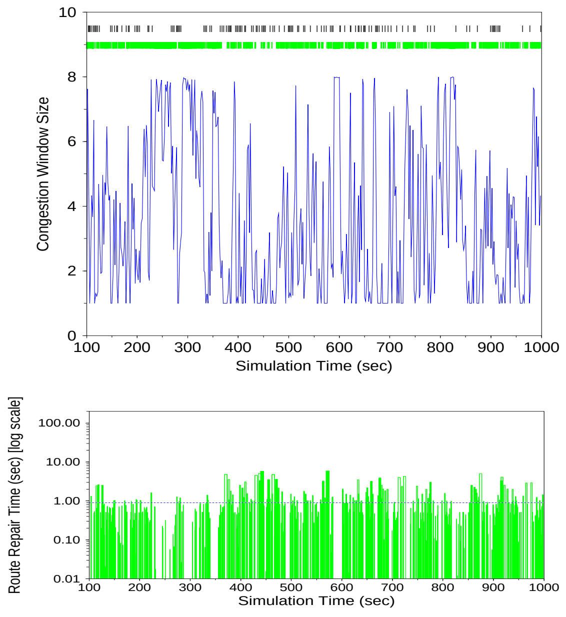


Figure 4.16: ADV congestion window sizes and route repair times for 1 TCP Reno connection with a 50 Kbps background load from 40 CBR connections. Average route repair time = 0.900 seconds. Throughput = 0.1702 Mbps.

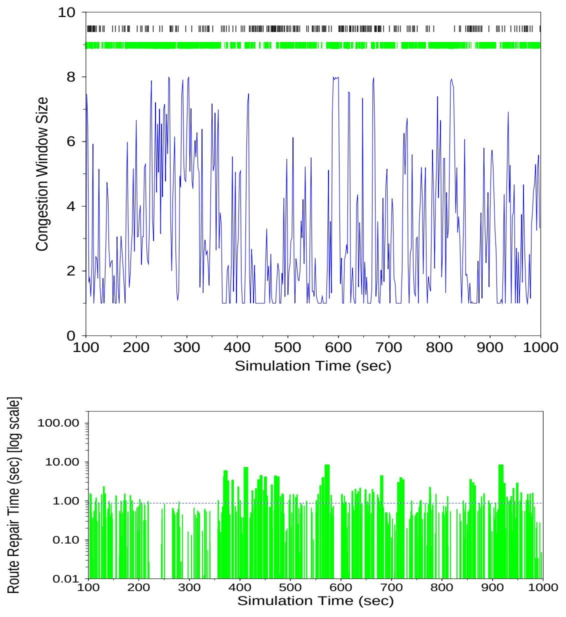


Figure 4.17: ADV congestion window sizes and route repair times for 1 TCP Reno-F connection with a 50 Kbps background load from 40 CBR connections. Average route repair time = 0.879 seconds. Throughput = 0.1710 Mbps.

in turn stimulates AODV to discover new routes to the TCP receiver more frequently, thus reducing route repair time. Looking at the lower graphs, Reno-F reduces the average route repair delay (indicated by horizontal lines) by more than 70% and the maximum route repair delay by more than 80%. So, the TCP sender is able to utilize repaired routes quickly and keep the congestion window open. In contrast, we see in Figure 4.16 that with TCP Reno, ADV is already doing a reasonable job of keeping the congestion window open and route repair times low. On average, ADV's proactive routing mechanism is able to repair routes in about half the time required by AODV. As a consequence, Reno-F produces almost no reduction in route repair time.

In summary, Reno-F, with its fixed-RTO heuristic, works well in situations where route failures are somewhat frequent and of long duration compared to the round-trip times for TCP packets. The performance benefits are greatest when an increased rate of packet retransmissions during route failure will stimulate the routing protocol's route repair mechanism to re-establish broken routes more quickly.

4.4 Analysis of TCP performance for multiple TCP connections

In the case of multiple TCP sources, we considered background traffic loads of 100 Kbps and 200 Kbps from 10 CBR and 40 CBR connections. The sender and receiver nodes were unique for each TCP connection, although in some cases a TCP endpoint was also the endpoint of one or more CBR flows.

4.4.1 Comparison of throughputs for different TCP options

The combined throughputs of 2, 5, and 10 TCP connections with a 100 Kbps background traffic load are shown in Figures 4.18 and 4.19.

As was observed previously for 1 TCP source, the addition of SACK and delayed ACKs to TCP Reno resulted in modest gains (5-10%) in throughput. In most cases, ADV continued to provide the highest throughput. As before, AODV showed decreased throughput relative to ADV and DSR as the number of CBR connections increased from 10 to 40. Because AODV relies on its route discovery process to establish new routes and repair broken routes, the larger number of connections results in considerably more work. At the same time, a larger number of connections, as well as a higher volume of traffic, enables DSR to use

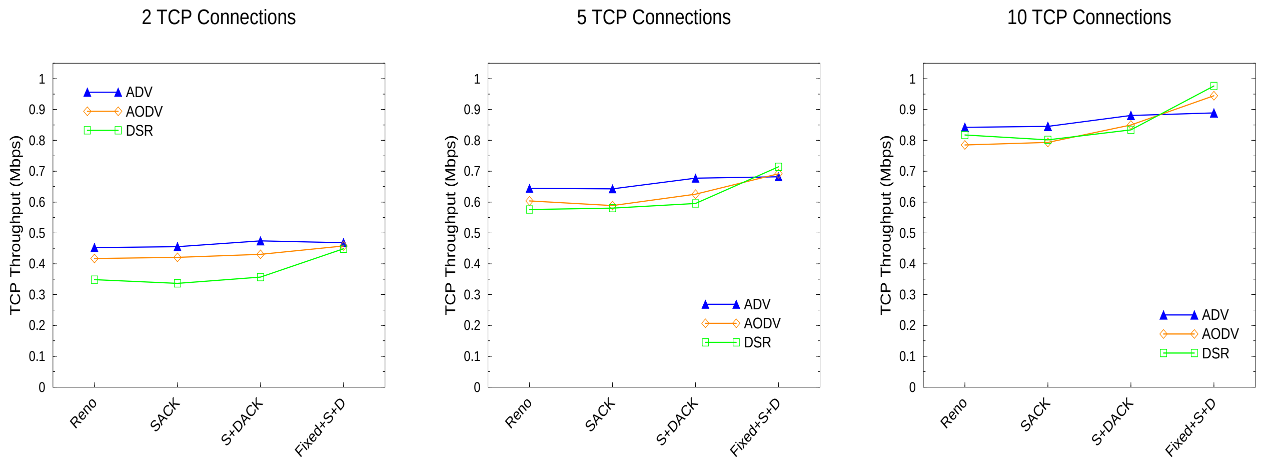


Figure 4.18: Combined throughputs for multiple TCP connections with a 100 Kbps, 10-CBR background.

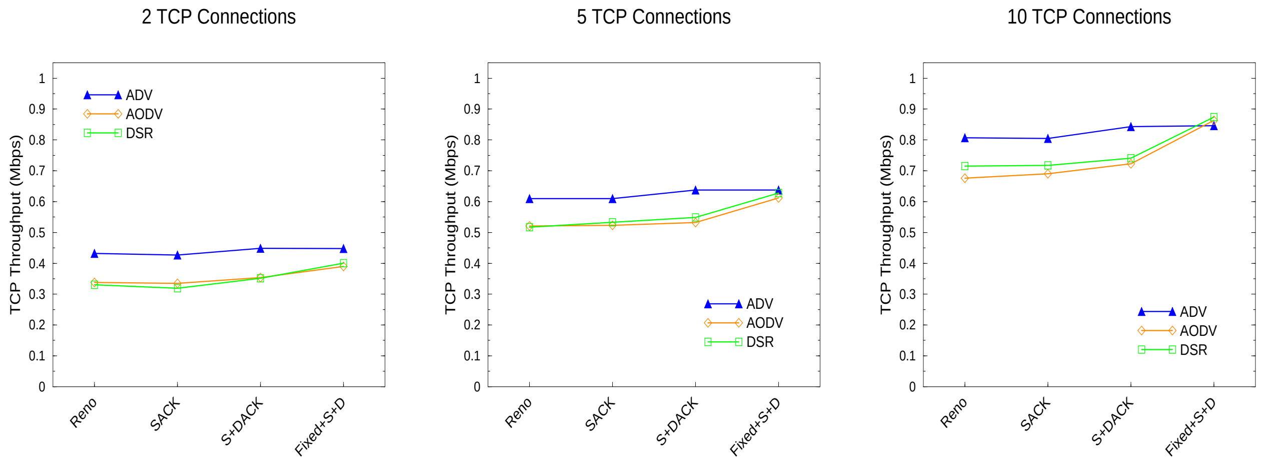


Figure 4.19: Combined throughputs for multiple TCP connections with a 100 Kbps, 40-CBR background.

caching and snooping effectively to reduce this route discovery overhead. For 5 and 10 TCP sources, DSR throughput was observed to be nearly as high or even higher than that of AODV, particularly for a larger number of CBR flows.

When the fixed-RTO technique (Reno-F) was employed, the performance differences of the three protocols tended to be minimized. However, for more than two TCP connections, the benefit of fixing the RTO in response to consecutive timeouts became great enough that AODV and DSR provided greater throughput than did ADV. For 10 TCPs with a 10-CBR background, DSR throughput was 10% higher than ADV throughput. This effect became even more pronounced when the background traffic load was increased to 200 Kbps as can be seen in Figures B.5 and B.6 in Appendix B.

With a 200 Kbps traffic load from 10 CBR flows, AODV and DSR both performed better relative to ADV. As we show later, ADV provided significantly better CBR throughput than the other protocols. Consequently, as the volume of CBR traffic increased, the impact of the background load on TCP throughput was largest for ADV. For the unmodified TCP sender (i.e., no fixed RTO), ADV and AODV throughputs were virtually the same and were higher than the DSR throughput, although this advantage decreased as the number of TCPs was increased. With 40 CBR connections, ADV continued to provide better throughput than AODV and DSR, which showed nearly identical performance.

4.4.2 TCP Reno vs. TCP Reno-F

The connect times, throughputs, goodputs, and routing overheads observed for the various number of TCP connections with a 100 Kbps background load from 10 CBR connections are shown in Figures 4.20 and 4.21. Given the relatively small throughput gains observed when using just the TCP options, we show only the results obtained using TCP Reno and TCP Reno-F (the combination of SACK, delayed ACKS, and fixed RTO). The results for a 100 Kbps 40-CBR background load and for a 200 Kbps load from 10 and 40 CBR connections are given in Appendix B.

For TCP Reno, the goodputs we observed for all three protocols were similar, ranging from about 96% to 98%. Reno-F decreased goodputs by about 2 percentage points.

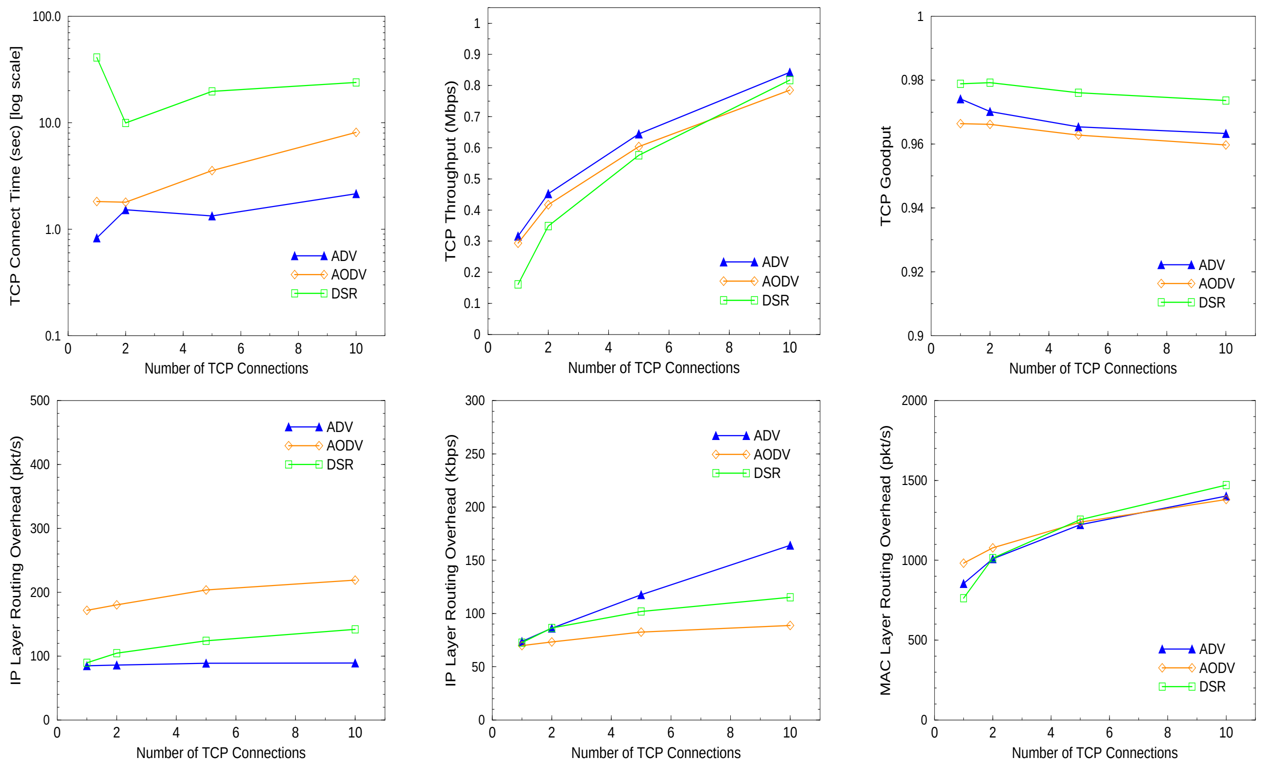


Figure 4.20: Connect times, throughputs, goodputs, and routing overhead for TCP Reno with a 100 Kbps 10-CBR background.

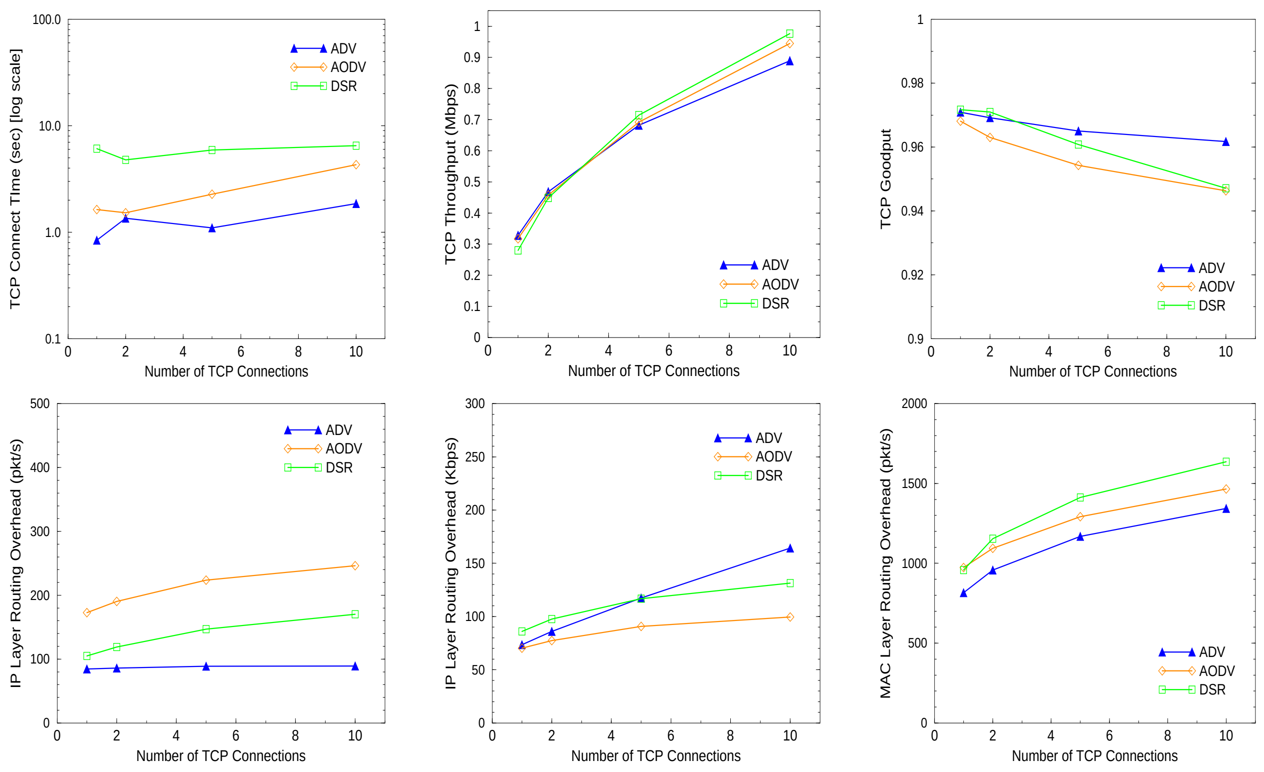


Figure 4.21: Connect times, throughputs, goodputs, and routing overhead for TCP Reno-F with a 100 Kbps 10-CBR background.

AODV generated the highest number of routing packets, followed by DSR. Routing activity for AODV and DSR increased as more TCPs were added, but the rate of increase diminished as the number of connections grew. The proactive ADV generated the fewest routing packets, and ADV routing activity was constant with respect to both the number of TCP connections and the number of CBR connections.

Although the frequency of ADV routing updates remained constant, the amount of routing information contained in the updates did increase with the number of TCPs. When measured in Kbps, ADV routing overhead was quite a bit higher than that of the other protocols. For 1 TCP flow, ADV generated about twice as many routing bytes per second as AODV for 40 CBR connections, again a consequence of ADV's proactive routing. Several researchers have pointed out that the high cost of accessing the medium in a wireless network places a premium on a reduced number of routing packets. Hence, the larger number of ADV routing bytes may not be a concern.

4.4.3 UDP performance

The TCP performance afforded by a routing protocol should be weighed against how well the protocol is able to move non-TCP traffic at the same time. To assess the impact of TCP traffic on non-reactive CBR flows, we measured the average CBR packet latency and the fraction of CBR packets successfully delivered. The results observed for the various number of TCP connections with a 100 Kbps background load from 10 CBR connections are shown in Figures 4.22 and 4.23. The results for a 100 Kbps 40-CBR background load and for a 200 Kbps load from 10 and 40 CBR connections are given in Appendix B.

CBR packet latencies increased as more TCPs were added, but the increases were not enough to indicate saturation. For ADV, the use of a fixed RTO had essentially no effect on CBR latency compared to TCP Reno. For AODV and DSR, however, the increases in TCP throughput with a fixed RTO resulted in increased CBR packet latencies, which were 20% higher for 10 TCP connections. We found that ADV's buffer refresh time (buffer timeout) has a considerable impact on CBR latency. ADV latency was about twice that observed for the other protocols, which had nearly identical latencies, and for 10 TCP connections, ADV latency was slightly greater than 1 second compared to 0.5 second for DSR and AODV. This is in contrast to the results reported by [12] and also shown in Chapter 3, in which ADV latencies were lower than those of AODV and

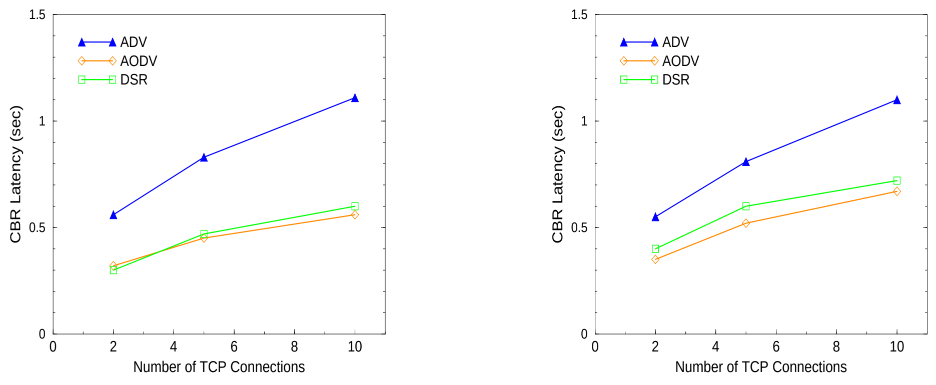


Figure 4.22: CBR packet latencies for TCP Reno and Reno-F with a 100 Kbps background load from 10 CBR connections

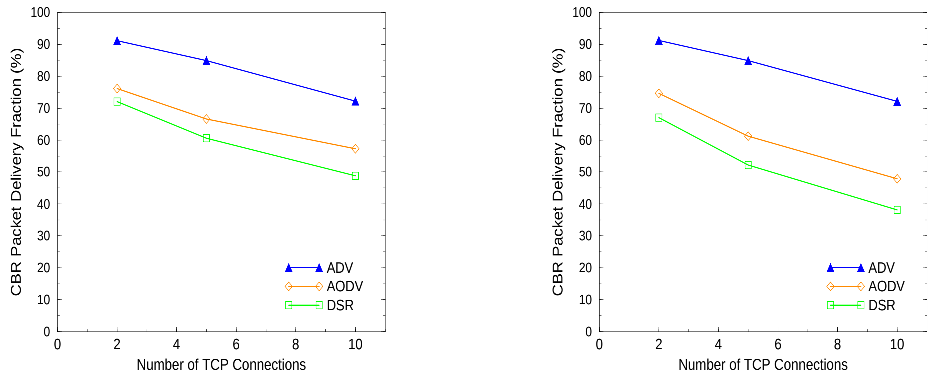


Figure 4.23: CBR packet delivery fractions for TCP Reno and Reno-F with a 100 Kbps background load from 10 CBR connections.

DSR. In that study, however, the CBR packet size was only 64 bytes, and a short buffer refresh time of 1 second was used for ADV.

Compared to the on-demand protocols, ADV did a much better job of handling the background CBR flows in terms of packet delivery fraction. With 10 TCP traffic sources, ADV delivered 70-75% of the CBR packets compared to about 60% for AODV and 50% for DSR. For only 2 TCPs, ADV achieved a delivery fraction above 90% for 10 CBR flows and in excess of 95% for 40 CBR flows. AODV outperformed DSR in all cases, the observed difference increasing with the number of TCP and CBR connections. As in the case of packet latency, the use of a fixed RTO had little or no effect on ADV's delivery fractions compared to TCP Reno. Packet delivery rates were 10-20% lower for AODV and DSR with a fixed RTO, again a result of the higher volume of TCP traffic.

4.4.4 Performance impact of DSR route replies from cache

In their TCP performance experiments, Holland et al. [29] noted that simply turning off DSR's route replies from cache yielded twice as much TCP throughput for a single connection. To confirm this observation, we measured DSR performance for 1 TCP connection with a 100 Kbps 10-CBR background load. The results are given in Table 4.4. We observed a 100% increase in throughput for TCP Reno when DSR's route replies from cache were disabled, nearly the same performance gain as was achieved by using Reno-F with DSR's route replies from cache turned on. It is noteworthy, however, that with route replies from cache turned off, DSR still attained a 14% increase in TCP throughput with Reno-F.

Factoring out the disadvantage that DSR incurs when route replies are made from a route cache with stale entries, we have observed that AODV and DSR derive roughly the same performance benefits from TCP Reno-F. Therefore, in the remainder of this dissertation, we will include only one on-demand routing protocol, AODV, in our performance analyses.

Table 4.4: DSR performance with route replies from cache turned on and off. For 1 TCP connection with a 100 Kbps background load from 10 CBR sources.

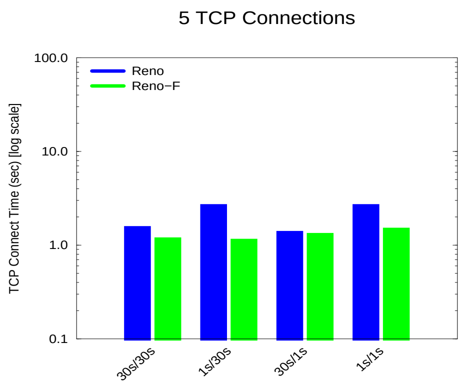
Performance Metric	**Cache Replies On**		**Cache Replies Off**	
	Reno	**Reno-F**	**Reno**	**Reno-F**
TCP throughput (Mbps)	0.1246	0.2567	0.2502	0.2856
CBR packet latency (s)	0.13	0.22	0.35	0.35
CBR delivery fraction (%)	82.79	77.71	84.65	84.56

4.5 Effect of the buffer refresh time on TCP performance

In this section, we assess the impact of ADV's buffer refresh time on TCP performance. The buffer refresh time is the maximum length of time that ADV can buffer a packet while a route is broken or unavailable. The fact that ADV buffers packets at intermediate nodes means that the performance impact of the buffer refresh time is likely to be greater for ADV than for a routing protocol such as AODV which buffers packets at the source node only. In particular, we expected the negative impact on TCP performance of buffering CBR data packets to be significantly greater for ADV, putting ADV at a disadvantage with respect to AODV and DSR. We wanted to minimize this disadvantage if possible.

There are a number of factors to consider in choosing a buffer refresh time. As mentioned above, long buffer times for CBR data packets can have a detrimental effect on TCP performance. Furthermore, if a CBR packet is allowed to stay in the routing-layer buffer too long, it becomes stale and is not useful even if it is eventually delivered. On the other hand, dropping a TCP data packet too quickly may cause frequent and unnecessary timeouts for TCP transmissions. Because the best choice of buffer refresh time might depend on the type of packet being buffered, we decided to investigate whether any performance benefit would accrue from having two buffering times – one for TCP packets, another for non-TCP packets. We tried two different values for each buffer refresh time: 30 seconds (long buffer time) and 1 second (short buffer time).

The results presented in this section were obtained using the simulation environment described in Section 4.2.1. Figures 4.24 and 4.25 present the TCP connect times and throughputs observed for 5 and 10 TCP connections with 100 Kbps of background traffic from 10 CBR sources. We also considered the case of



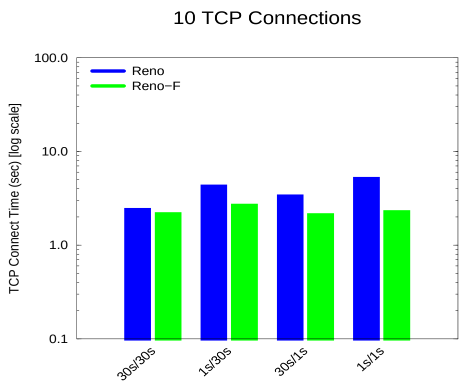


Figure 4.24: Connect time for ADV for 5 and 10 TCP connections with a 100 Kbps background load from 10 CBR sources. In the labels at the bottom of each graph, the buffer refresh times are listed in the order TCP-time/CBR-time. For example, 30s/1s denotes a 30-second TCP refresh time and a 1-second CBR refresh time.

5 and 10 TCP connections with 200 Kbps of background traffic from 10 CBR sources. These additional observations are presented in Appendix B.

TCP connect times were observed to be generally shorter with the longer TCP-packet buffer refresh time of 30 seconds, especially when TCP Reno was used. If a route cannot be established before a packet is dropped from the sender's buffer, a TCP timeout will necessarily occur. Keeping packets buffered for a longer period of time avoids these unnecessary timeouts, and the benefit is greatest in the case of TCP Reno's exponentially increasing timeout intervals.

Buffering TCP packets long enough to avoid unnecessary timeouts can also be beneficial for TCP throughput. For TCP Reno, the use of a 30-second TCP-packet buffer refresh time resulted in throughput gains of up to 8% relative to the 1-second case. The longer refresh time appeared to have the opposite effect for Reno-F, as throughputs were somewhat higher with a 1-second buffer refresh time for TCP packets. However, these increases were 4% or less, indicating that Reno-F throughput is not significantly impacted by the choice of buffer refresh time for TCP packets.

TCP connect times were not significantly affected by the choice of buffer refresh time for CBR packets. TCP throughput, however, was observed to be 8-12% higher with a 1-second CBR-packet buffer time, regardless of the choice of TCP-packet buffer time. When CBR packets do not take up as much buffer space at the source and intermediate nodes, the flow of TCP traffic is improved.

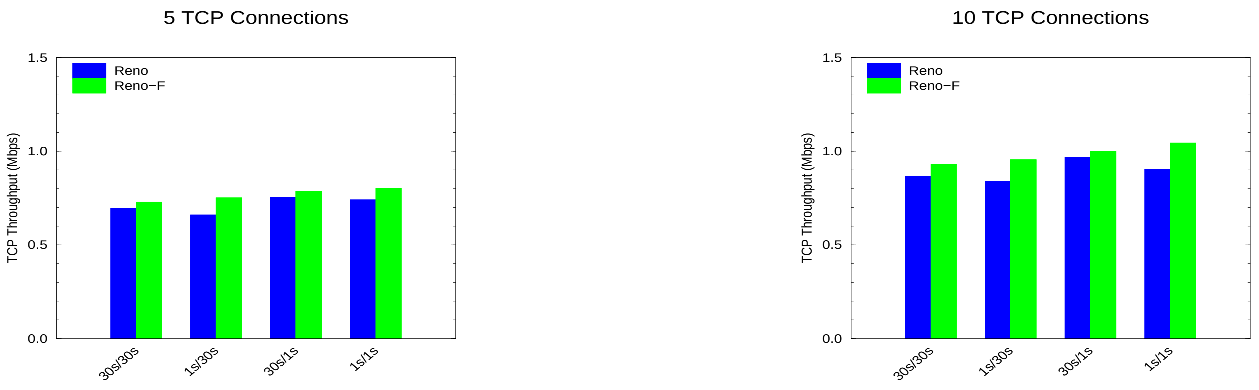


Figure 4.25: Throughput for ADV for 5 and 10 TCP connections with a 100 Kbps background load from 10 CBR sources.

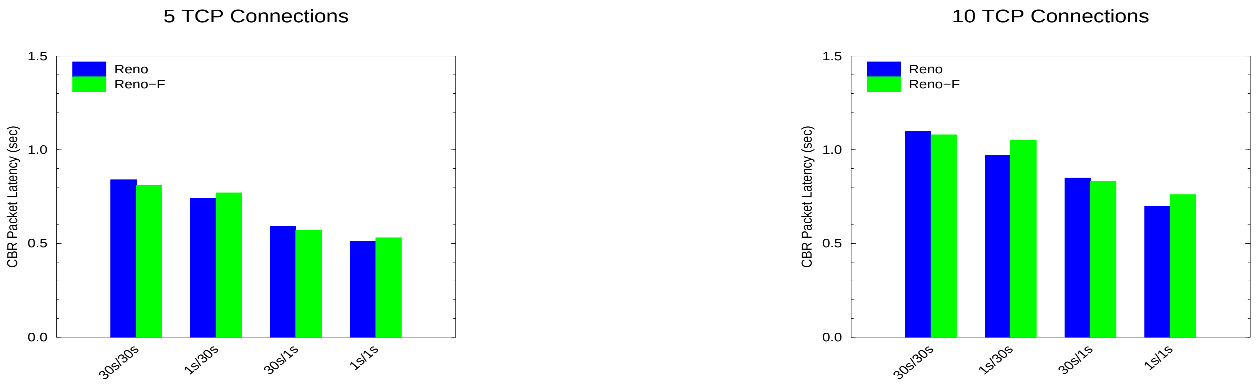
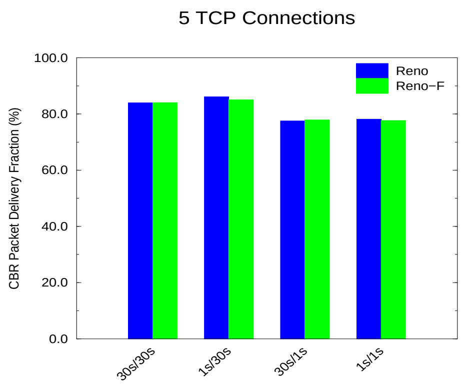


Figure 4.26: CBR packet latency for ADV for 5 and 10 TCP connections with a 100 Kbps background load from 10 CBR sources.

Figures 4.26 and 4.27 present the CBR packet latencies and delivery fractions observed for 5 and 10 TCP connections with 100 Kbps of background CBR traffic. When packets are buffered for longer periods of time, the older packets compete for available bandwidth with more recently transmitted packets. As a result, the average latency of successfully delivered CBR packets can be expected to be larger. With the use of the short 1-second buffer refresh time for CBR packets, we observed decreases in average packet latency as high as 31% compared to the 30-second case. Average packet latencies dropped as much as 18% for TCP Reno with the use of a 1-second TCP-packet refresh. With a longer refresh time, the larger number of TCP packets buffered at intermediate nodes adds to the average packet delay for CBR traffic. This effect was much less pronounced with Reno-F, at most a drop of 8%.



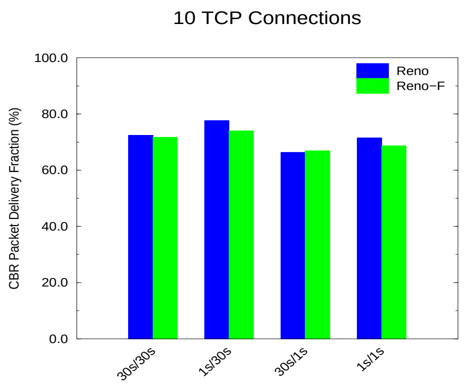


Figure 4.27: CBR packet delivery fraction for ADV for 5 and 10 TCP connections with a 100 Kbps background load from 10 CBR sources.

Since shorter buffer refresh times result in more packet drops, the choice of a 1-second CBR-packet buffer refresh time lowered the packet delivery fraction by as much as 8-10%. Due to shorter queue lengths at intermediate nodes, a shorter TCP-packet refresh time results in a higher delivery fraction for ADV because fewer CBR packets have to be dropped due to staleness caused by late delivery.

Based on the results of this section, we conclude that the best combination of ADV buffer refresh times is 30 seconds for TCP packets and 1 second for all other data packets. The longer TCP-packet buffer time yields shorter connect times and higher TCP throughput, particularly when the TCP Reno protocol is used. At the same time, the shorter buffer time for non-TCP packets results in reduced packet latency and increased TCP throughput.

4.6 Concluding remarks

We began this chapter by categorizing techniques for improving TCP performance in MANETs. We identified three levels at which one might attack the problem, and at each level we gave examples of potential methods for improving TCP performance. The highest level, level 1, is the class of techniques that are purely end-to-end at the transport layer. Such methods do not require information from or interaction with the routing layer or lower layers in the protocol stack, either at the connection endpoints or at intermediate nodes. Level 1 techniques have the advantage of being less complex and therefore easier to implement, requiring fewer modifications of existing network software. If adequate performance gains can be obtained with level 1 techniques, they are preferable.

We have considered several level 1 techniques for improving TCP performance in MANETs. Of these, our proposed TCP sender-side heuristic, the fixed-RTO technique, shows the most promise. We have combined fixed RTO with TCP's selective acknowledgements and delayed acknowledgements options to form the TCP Reno-F protocol. Reno-F works well in a variety of situations; for example, TCP performance benefits accrue with or without the presence of background network traffic. In the remainder of this section, we identify three main cases in which we have observed Reno-F to be helpful.

Reno-F works well with routing protocols having route discovery mechanisms that can be stimulated by an increased rate of packet transmissions, e.g. on-demand algorithms. Route stimulation can be useful in situations where route failures are frequent, or where route repairs are difficult, say due to a sudden, large increase in the number of hops between the TCP sender and receiver.

We have shown in Section 4.5 that, for ADV, the improvements in performance yielded by Reno-F are generally greater when TCP-packet buffering times are short. Short buffer times can increase the rate at which TCP packets are dropped, leading to more frequent TCP retransmit timeouts. Reno-F mitigates the negative performance effect of these timeouts.

Reno-F tends to minimize the TCP performance differences among different MANET routing protocols, although it does not entirely eliminate these differences. In some cases, Reno-F can offset performance problems due to algorithmic shortcomings in a routing protocol. For example, an interesting effect of Reno-F is that it largely negates DSR's well-known problem with stale routes. It should be noted, however, that even when the stale route problem is alleviated by turning off route replies from cache, Reno-F is still able to substantially improve the TCP performance of DSR.

Chapter 5

An Adaptive Datagram Protocol for Mobile Ad Hoc Networks

While TCP may be the most widely used transport protocol, the User Datagram Protocol (UDP) also carries a sizeable amount of Internet traffic. For example, UDP is used in conjunction with the Real-Time Transport Protocol (RTP) [62] to carry streaming video flows. This fact suggests that UDP performance should be expected to have a significant impact on the overall performance that can be achieved in mobile ad hoc networks. Therefore, we examined the UDP performance of currently proposed MANET routing protocols with a view to improving that performance.

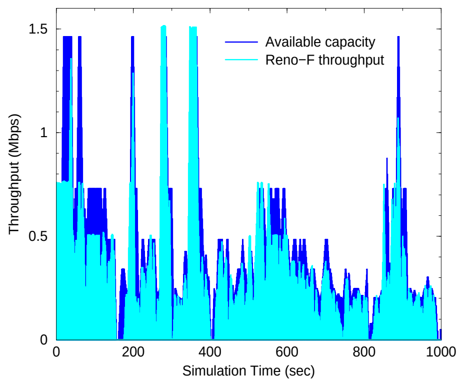
Our results in Chapter 3 and in several previous studies [12, 13, 20, 43] have shown that constant-bit-rate UDP flows reach saturation in the range of 200 to 300 Kbps of combined throughput. However, we saw in Chapter 4 that a comparable number of TCP connections can attain an aggregate throughput of close to 1 Mbps, even in the presence of competing UDP traffic. This performance advantage for TCP over UDP contradicts the results typically seen for wireline networks.

Table 5.1 shows the results of additional simulations with identical settings (packet size, connection endpoints) for TCP and CBR flows. The sending rate for the CBR sources was chosen to match the lowest observed TCP throughput, approximately 1.07 Mbps. These simulations demonstrate that TCP indeed achieves higher throughput than UDP.

Utilizing our average-case mobility scenario from the analysis of TCP Reno-F in Section 4.3, we compared the throughput observed for a TCP Reno-F connection with the throughput attained by a CBR flow

Table 5.1: Transport-layer throughput in Mbps for 10 connections with no background traffic.

Transport Protocol	ADV	AODV
TCP Reno	1.1874	1.0784
TCP Reno-F	1.2746	1.2178
UDP	0.3865	0.4337



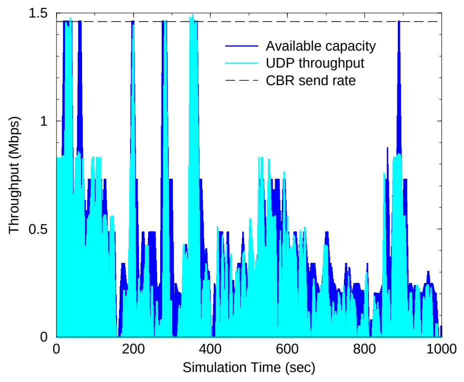
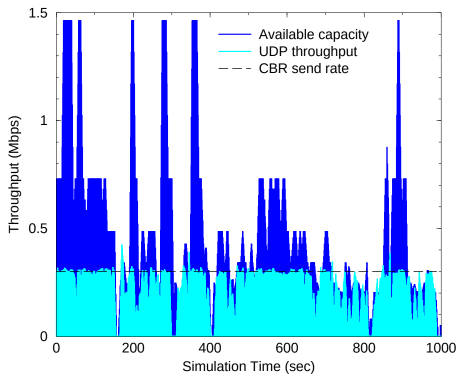


Figure 5.1: Throughput compared to capacity for a TCP Reno-F connection and a 1.46 Mbps CBR flow with AODV and no background traffic load. TCP throughput = 0.361 Mbps, UDP throughput = 0.361 Mbps, UDP delivery fraction = 25%.

with a sending rate equal to 1.46 Mbps, the maximum expected TCP throughput for a 1-hop connection as given in Table 1.1 in Chapter 1. The AODV routing protocol was used and there was no background traffic. Figure 5.1 shows how Reno-F is able to adapt its sending rate and efficiently utilize the available capacity. With the very high sending rate, the CBR flow is able to achieve an equally high throughput. However, because there is insufficient bandwidth to accommodate such a high packet rate, the majority of the CBR packets are dropped in the network and the packet delivery fraction is only 25%. The dropped packets consume network resources (bandwidth and buffer space) that could have been used for other traffic had it been present. In Figure 5.2, we compare the throughput of a 300 Kbps CBR flow and a 500 Kbps CBR flow to the available capacity for the same mobility scenario. Because the two CBR flows have much lower sending rates, the delivery fractions are higher. Nevertheless, UDP's non-adaptivity limits the throughput that can be attained.



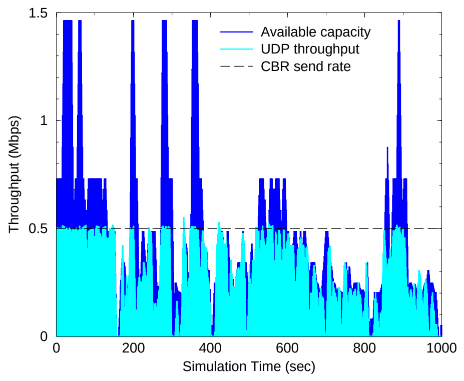


Figure 5.2: Throughput compared to capacity for a 300 Kbps CBR flow and a 500 Kbps CBR flow with AODV and no background traffic load. For the 300 Kbps flow, throughput = 0.232 Mbps, delivery fraction = 77%. For the 500 Kbps flow, throughput = 0.305 Mbps, delivery fraction = 61%.

Based on the observation that TCP outperforms UDP in MANETs, we surmised that adapting the sending rate to the available network bandwidth is essential for a transport-layer protocol to efficiently utilize the capacity of a MANET. To this end, we have designed a new transport-layer protocol, called the Adaptive Datagram Protocol, which is adaptive to network conditions.

5.1 Adaptive Datagram Protocol

Like UDP, the Adaptive Datagram Protocol (ADP) is a best-effort service that does not provide reliable in-order packet delivery. Unlike UDP however, ADP attempts to inject packets into the network at a rate the network can handle. This is accomplished with a simple scheme in which ADP packets are acknowledged by the receiver and these ACKs trigger subsequent packet transmissions. If the application rate temporarily exceeds the available network capacity, the excess packets are buffered up to some limit set by the application. However, if the arrival rate of data packets from the application exceeds ADP's buffering capability, the excess packets are simply dropped from the buffer (oldest packets first) and do not impact network performance. A depiction of the flow of data packets and acknowledgements in ADP is given in Figure 5.3.

Given the success with which TCP is able to conform its flow to changing network capacity, we decided to borrow from TCP's windowing algorithm. Since we do not require reliable packet transmission, that aspect of TCP was not essential. In particular, there is no need for the sender to wait for the acknowledgement

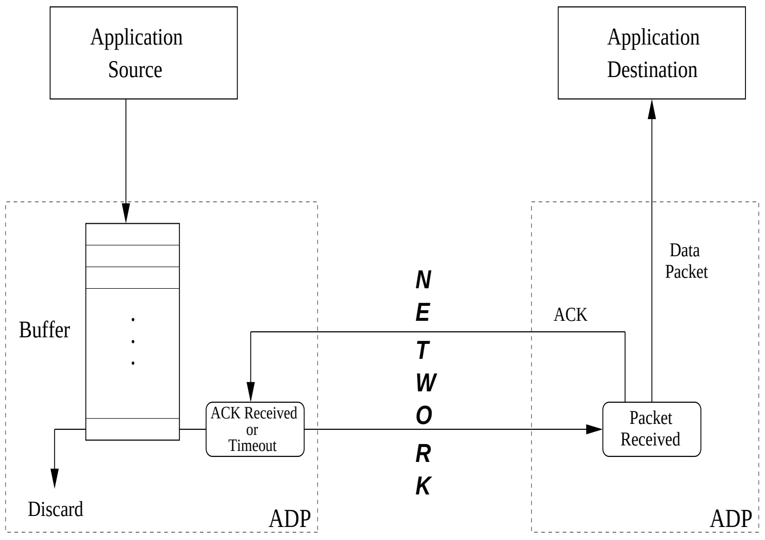


Figure 5.3: ADP packet flow.

of a specific packet as TCP does with its retransmit timer. However, we still needed to retain a timer based on the estimated round-trip time (RTT) to keep the sender from stalling while waiting for the arrival of an ACK.

We derive a smoothed estimate of the RTT in the same way as TCP. A timestamp field in the ADP packet header is used to store the time at which a packet is transmitted and this value is echoed back by the receiver when it acknowledges delivery of the packet. When the ACK is received by the sender, the time elapsed since the packet was transmitted is used to update the current estimate of the RTT. When a packet is transmitted, a timer is set to expire after an interval of time computed exactly the way TCP's retransmit timeout interval is calculated, i.e. the transmit timeout interval equals the estimated RTT plus 4 times the variance in the RTT. The use of the transmit timer guards against the case in which all outstanding ACKs have been lost in the network. Each ADP packet is assigned a sequence number which is used to maintain a log of which recently transmitted packets have been acknowledged. The ADP packet header format is given in Table 5.2.

Table 5.2: Fields in an ADP packet header.

Source port (16 bits) | Destination port (16)
Length (16) | Checksum (16)
Sequence number (32)
Time stamp (32)

5.1.1 Implementing ADP

Windowing algorithm

Initially we used a simple windowing algorithm in which we keep track of the acknowledgement status of recently transmitted packets. Our algorithm differs from the standard sliding window in that since we do not require retransmissions to ensure reliable packet delivery, we cannot use cumulative ACKs to define the left hand side of the window because dropped packets will never be ACKed. Instead, the left hand side points to the oldest packet transmitted, but not yet acknowledged, since the last timeout. The right hand side points to the most recently transmitted packet. The window includes all unacknowledged packets from the left hand side through, and including, the right hand side. The size of the window is started at 1 and is expanded using the additive increase algorithm from TCP. We opted not to employ a slow start phase because, based on our experience with TCP, window sizes in MANETs will generally not be very large. Moreover, since ADP does not wait for specific packets to be acknowledged but will transmit a new packet on the receipt of any ACK (subject to the constraints described below), we were concerned that the multiplicative increase of the window size during slow start might expand the window too aggressively.

When an ACK arrives for a previously unacknowledged packet inside the window, the window size is updated and new packets are transmitted until the number of unacknowledged packets between (and including) the left and right hand sides of the window is equal to the updated window size. The transmit timer is reset each time a packet is transmitted and is not associated with a particular packet. That is, we do not wait for a particular packet to be acknowledged before canceling the transmit timer – the receipt of any ACK is sufficient. If the timer expires before an ACK is received, a new packet is transmitted and the right hand side of the window is reset so that it points to this packet. The left hand side of the window is

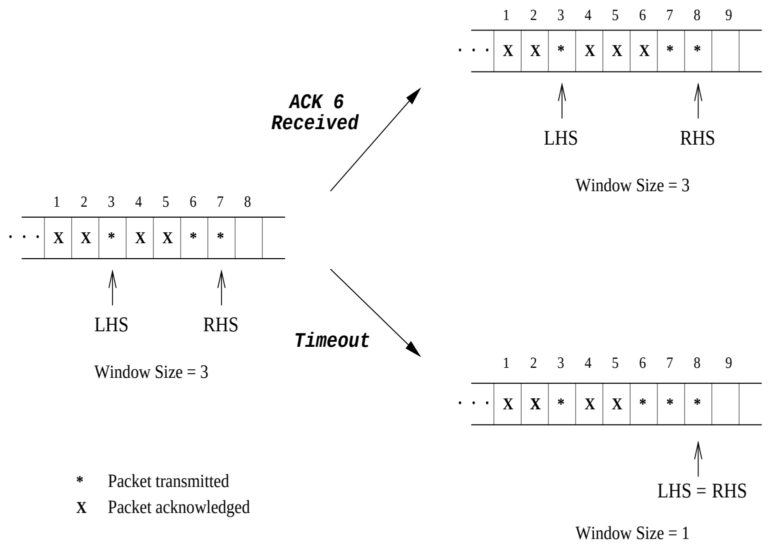


Figure 5.4: ADP congestion window.

advanced so that it also points to the newly sent packet, which returns the window to its starting size of 1. As an optimization, the previous value of the left hand side is remembered so that ACKs which would have fallen in the previous window will still be counted in case they are not really lost but simply delayed. The basic operation of the ADP congestion window is illustrated in Figure 5.4.

ACK-clocking

We also considered the use of a simple ACK-clocking mechanism in which the arrival of each ACK triggers the next packet transmission. There is no window to maintain, so it is not necessary to track the acknowledgement of individual packets. The transmit timer is still required to keep ADP from stalling in case the flow of ACKs is interrupted. It may appear that using any ACK to trigger the sending of a new packet might cause too high an injection rate if the network duplicates packets frequently. However, we have observed a very low rate of packet duplication in our simulations and we do not consider this to be a problem.

Because there is no retransmission of previous packets as in TCP, the fact that there is no window in this scheme does not preclude the possibility of having multiple, distinct packets in flight simultaneously. This is possible because when the transmit timer expires before a tardy ACK arrives, a packet will be injected into the network in addition to the packet that will be sent when the late ACK is received.

Table 5.3: Performance for 1 connection with TCP-like windowing algorithm and a 100 Kbps background load from 10 CBR sources.

Performance Metric	**ADV**	**AODV**
Throughput (Mbps)	0.3054	0.2905
Delivery fraction (%)	86.40	84.07
Network latency (sec)	0.3997	0.2408

Table 5.4: Performance for 1 connection with simple ACK-clocking and a 100 Kbps background load from 10 CBR sources.

Performance Metric	**ADV**	**AODV**
Throughput (Mbps)	0.3146	0.3103
Delivery fraction (%)	99.38	99.24
Network latency (sec)	0.0560	0.0334

Windowing vs. ACK-clocking

We compared the performance attained with the ACK-clocking mechanism to that obtained using the windowing algorithm. Packet buffering was not modeled in this simulation, and an infinite backlog of packets ready to be sent was assumed. Since queuing delay was undefined, packet latency consisted only of the network transmission delay. Although packet sequence numbers are not required for the ACK-clocking scheme, the sequence number field was retained in the packet header. The results are shown in Tables 5.3 and 5.4.

The packet delivery fractions obtained with the windowing method were lower than those observed with ACK-clocking. This is due to the fact that the additive-increase algorithm for adjusting the window size is designed to induce packet drops in an effort to continually probe the network for additional bandwidth. Packet latencies were observed to be higher for the windowing method. Because the windowing algorithm attempts to maximize the number of packets in flight, an in-flight packet will frequently have to sit in a routing layer buffer awaiting transmission. In the ACK-clocking scheme, packets are less likely to incur this delay.

In terms of throughput, the ACK-clocking scheme yielded comparable performance to the windowing algorithm and, being simpler, was adopted for our further work.

Table 5.5: Effect of fixing the transmit timeout interval on throughput (Mbps). For 1 video connection with a 100 Kbps background from 10 CBR sources.

Protocol	**Fixed**	**Not fixed**
ADV, 5 s frame buffer	0.1182	0.1054
ADV, unlimited frame buffer	0.1456	0.1386
AODV, 5 s frame buffer	0.1366	0.1315
AODV, unlimited frame buffer	0.1449	0.1453

Table 5.6: Effect of fixing the transmit timeout interval on frame delivery fraction (%). For 1 video connection with a 100 Kbps background from 10 CBR sources.

Protocol	**Fixed**	**Not fixed**
ADV, 5 s frame buffer	80.25	71.56
ADV, unlimited frame buffer	98.53	93.02
AODV, 5 s frame buffer	93.23	89.78
AODV, unlimited frame buffer	99.09	99.04

5.1.2 Fixed transmit timeout interval

In the TCP Reno congestion control scheme, each time a retransmit timeout occurs, the retransmit timeout interval (RTO) is doubled. The exponential backoff of the RTO continues until either an ACK is received for the retransmitted packet or a maximum number of backoffs have occurred. We followed this practice in our initial design, doubling ADP's transmit timeout interval on consecutive timeouts. This scheme was in use for the comparison of windowing and ACK-clocking above. However, as we saw in Chapter 4, fixing the RTO on consecutive timeouts rather than continuing to double it, can yield gains in TCP performance. So, we explored the possibility that a similar design choice with respect to ADP's transmit timeout interval (TTO) might also result in performance benefits.

We measured the performance of a single video connection using ADP with and without fixing the transmit timeout interval on consecutive timeouts. The results shown in Tables 5.5 and 5.6 were obtained using the experimental setup described in Section 5.2. A background traffic load of 100 Kbps from 10 CBR sources was included. The fixed-TTO version of ADP yielded improved performance when the ADV routing protocol was used, while the gains observed for AODV were modest at best. Since fixing the TTO appears to only help, not hurt, performance, we elected to employ this technique in the ADP design.

5.2 ADP performance analysis

We envision the transport of multimedia traffic, e.g. streaming video, as a potential application of ADP in MANETs. Therefore, in our analysis of ADP performance, we utilized variable-bit-rate (VBR) traffic in order to more realistically simulate a multimedia flow. To facilitate comparisons with the TCP performance results of Chapter 4, we included background CBR traffic in all ADP simulations.

5.2.1 Experimental methods

The simulation environment was as described in Section 3.4.1. We simulated a network of 50 nodes moving in a 1000m x 1000m square field. A background UDP traffic load of 100 Kbps was generated by 10 CBR connections, and the CBR packet sizes were fixed at 512 bytes. After a warm-up time of 100 seconds, one or more simulated video connections were established over each of which VBR traffic flowed for 900 seconds. Performance results were averaged over 10 different mobility scenarios. Based on the results of Section 4.5, we set ADV's UDP-packet buffer refresh time to 1 second. A buffer refresh time of 30 seconds was used for AODV.

5.2.2 Simulation of video traffic

For the foreground traffic, we simulated streaming video using the flow characteristics given for a model of MPEG-coded video traffic [41]. In this model, the compressed video stream consists of frames of three types (Intra-frame, Predictive, and Bidirectional) which occur in the repeated pattern I-B-B-P-B-B-P-B-B-P-B-B-P-B-B-I-... as depicted in Figure 5.5. The sizes of each of these frame types are assumed to be lognormally distributed. For this study, we scaled back the frame-size distribution parameters to give a bit rate that can be handled by a network with 2 Mbps wireless links. Our simulated traffic consisted of 15 packets a second with an overall mean packet size of approximately 1210 bytes. The frame-size distribution parameters we used are shown in Table 5.7.

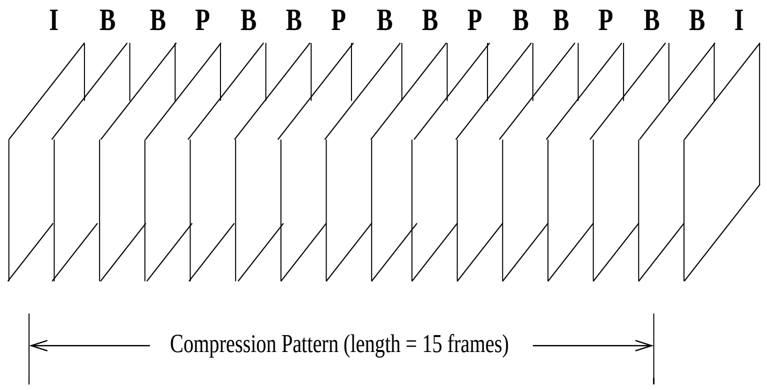

I B B P B B P B B P B B P B B I
Compression Pattern (length = 15 frames)

Figure 5.5: Video frame pattern.

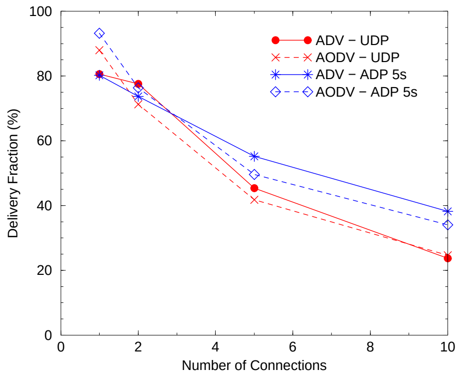
Table 5.7: Frame-size distributions for simulated video stream.

Distribution Parameter	**All Frames**	**I Frames**	**P Frames**	**B Frames**
Mean (bytes)	1211.1	5900.9	1605.8	584.2
Standard deviation (bytes)	309.7	1201.8	1048.9	159.1

5.2.3 Performance results

One of the configurable parameters for ADP is the length of the buffer used to hold video frames. Initially, we used a 75-frame buffer, sufficient to hold 5 seconds of our simulated video traffic. In addition, we varied the number of video flows: 1, 2, 5, and 10. Figure 5.6 presents the frame delivery fraction (which takes into account buffer and network losses) and throughput for ADV and AODV. For comparison, we also show the frame delivery fraction and throughput achieved by standard UDP. Compared to UDP, ADP improves the delivery fraction of one video flow marginally at best. However, as the number of video flows increases to 2, 5, and 10, ADP provides significantly higher performance than UDP; frame delivery fractions were higher by 40-60%. With ADP, most of the frame losses occur due to buffer overflow, and very few frames, about 1.5%, are lost by the network. So, with unlimited buffer lengths, the frame loss will be about 1.5%; however, the buffer lengths will increase continuously for 5 or more video flows because the application generates more packets than the network can handle.

Now let us examine the throughput graph. For an upper bound, we also indicate throughputs achieved



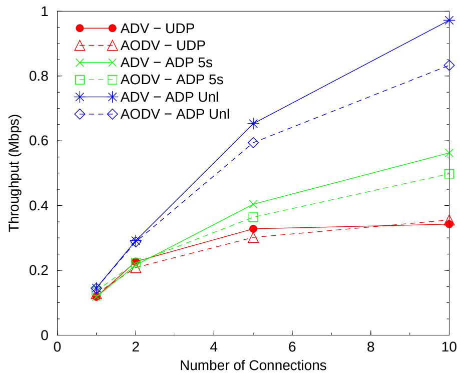


Figure 5.6: Frame delivery fraction and throughput for UDP and ADP with a varying number of video flows and a 100 Kbps background load from 10 CBR sources.

by ADP with infinite frame buffers. ADP gives marginal improvement in throughput for one video flow for both limited and unlimited buffer lengths. However, as the number of flows increases, ADP provides substantially higher throughput. With a 75-frame buffer, ADP achieves 20% more throughput for 5 video flows and 40-60% more throughput for 10 video flows. For two video flows, however, ADP gives about the same throughput as UDP. The primary reason for higher performance by ADP with 5 and 10 video flows is that the network is congested with UDP but not with ADP. With unlimited frame buffers, ADP yields increases in throughput ranging from 50% for 2 video flows to 200% for 10 flows. Finally, it is noteworthy that AODV performs marginally better for 1 or 2 video flows and ADV performs significantly better with 10 flows.

In Figure 5.7, we compare the throughput obtained by ADP connections with unlimited buffering with the throughput obtained by the same number of FTP connections. The FTP throughput numbers were taken from the performance analysis in Chapter 4. For 1 or 2 connections, ADP is underperforming FTP because the rate at which ADP packets can be generated is limited by the application's sending rate, while FTP has an "infinite" backlog of packets ready to send. As the number of connections increases, the throughputs of both ADP and FTP are limited by the capacity of the network. In that case, ADP is doing as well or better than FTP at utilizing the available network bandwidth.

To analyze the performance of ADP further, we conducted several simulations varying ADP frame buffer size. Figures 5.8–5.13 indicate the details of frame losses and video throughputs achieved. For one video

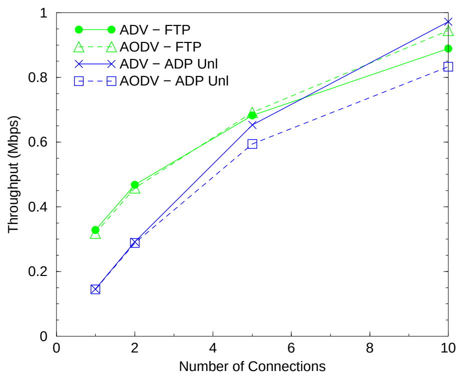


Figure 5.7: Throughput for FTP over TCP Reno-F and video over ADP with unlimited buffering with a varying number of connections and a 100 Kbps background load from 10 CBR sources.

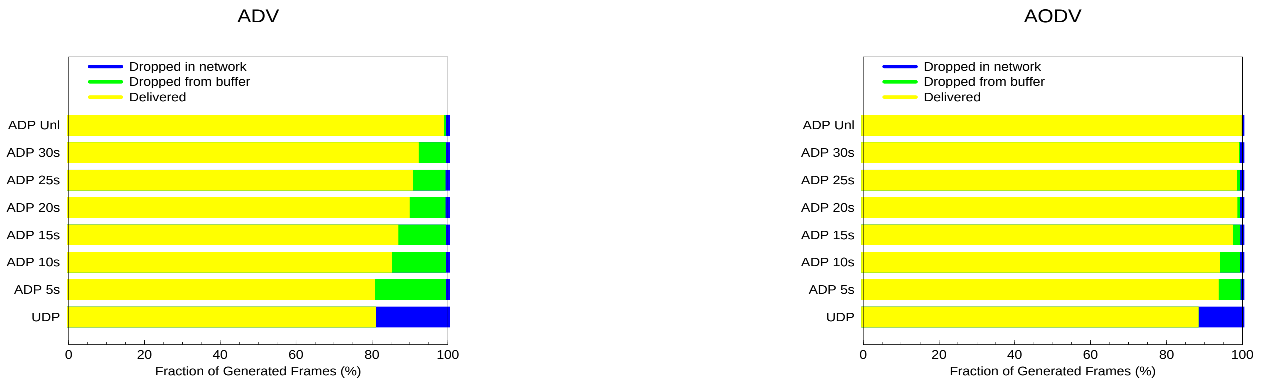


Figure 5.8: Frame delivery and network drop rates for 1 video connection with ADV and AODV.

flow, AODV performs very well; even with a 5-second frame buffer, the frame delivery fraction is around 93%. As the number of video flows increase, ADV performs better than AODV. For 5 and 10 video flows, the frame delivery fraction is rather low. For these cases, the video traffic demands more bandwidth than the network can provide. With UDP, this traffic will congest the network and will interfere with other traffic. In comparison, ADP offers the following benefits: (a) throughput is improved significantly, and (b) most of the excess load is dropped by the source when the frame buffer overflows, which avoids wasting network resources on the traffic that is likely to be undeliverable.

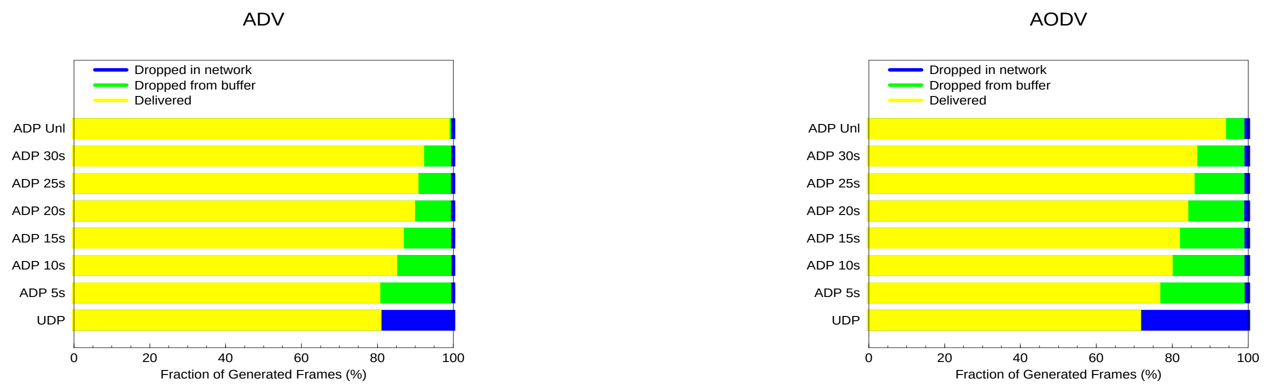


Figure 5.9: Frame delivery and network drop rates for 2 video connections with ADV and AODV.

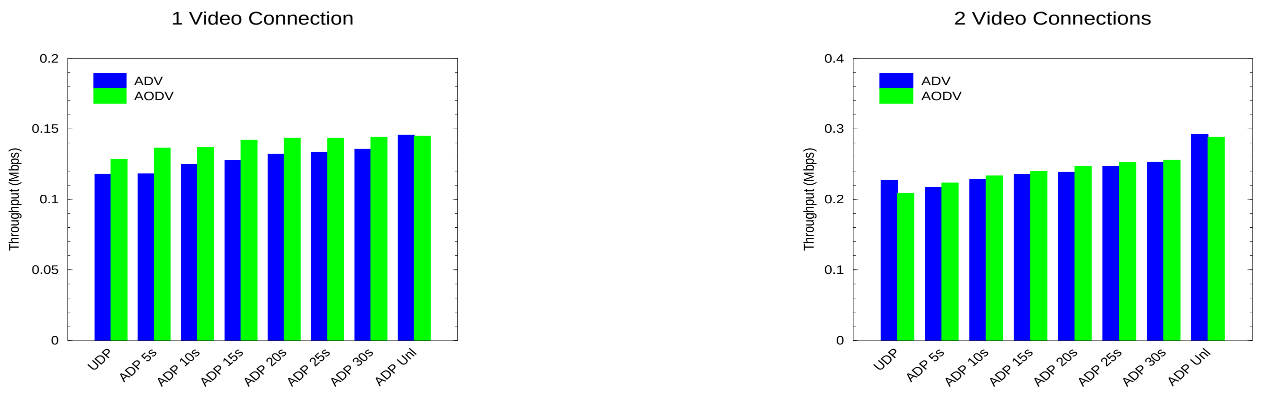


Figure 5.10: Throughput for 1 and 2 video connections with a 100 Kbps background load from 10 CBR sources.

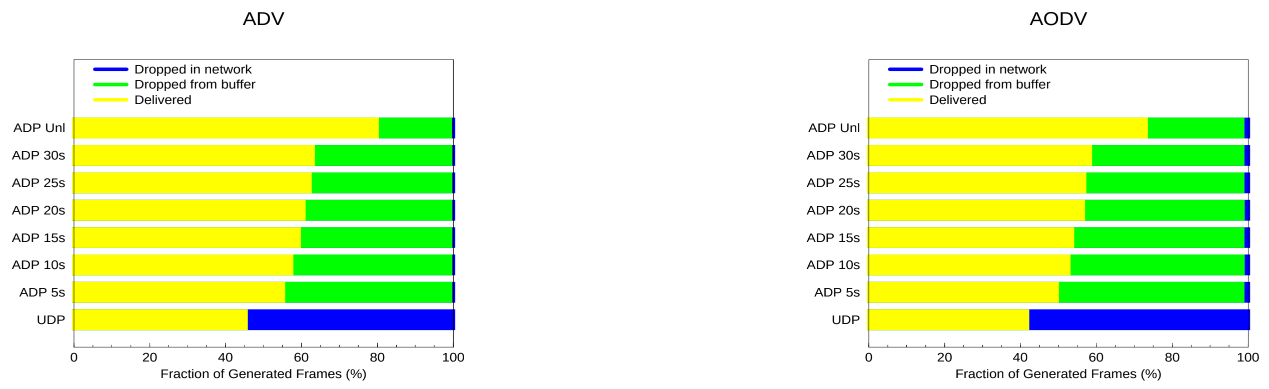


Figure 5.11: Frame delivery and network drop rates for 5 video connections with ADV and AODV.

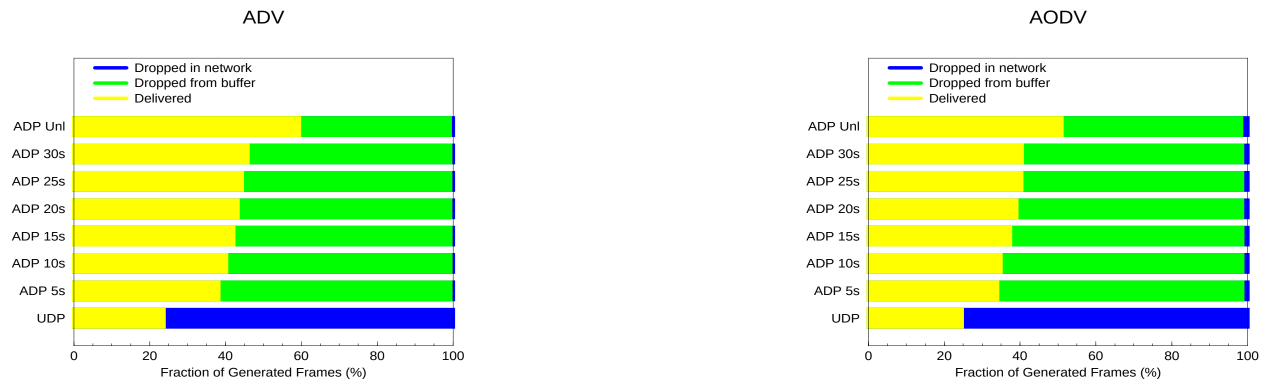


Figure 5.12: Frame delivery and network drop rates for 10 video connections with ADV and AODV.

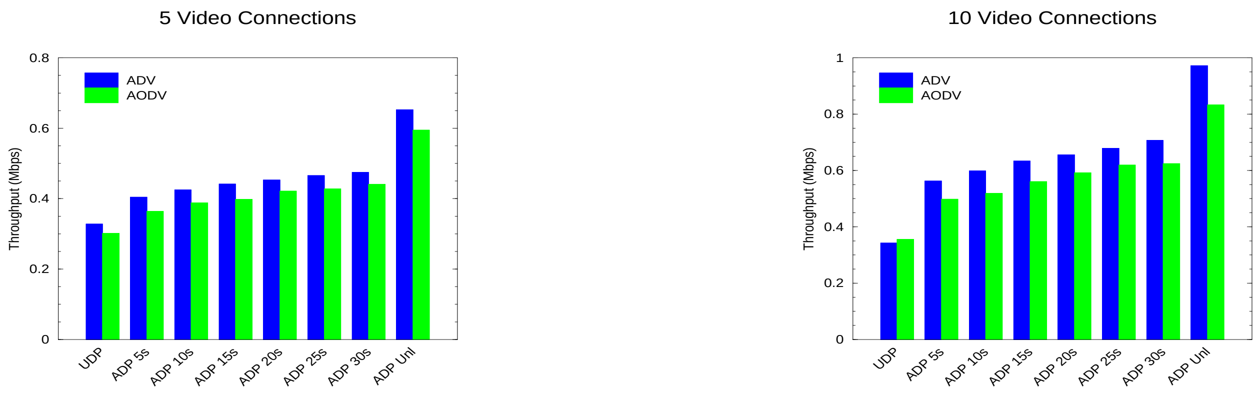


Figure 5.13: Throughput for 5 and 10 video connections with a 100 Kbps background load from 10 CBR sources.

5.2.4 Network resource usage

One interesting observation is that with ADP very few packets are dropped by the network. There are two reasons for this: (a) ADP tries to inject packets into the network at a rate which conforms to the available bandwidth, and (b) ADP tends to send more packets when the distance between sender and receiver is short and fewer packets when the distance is long.

Figure 5.14 shows the throughput, in video frames per second, that we observed for 10 video connections in one of the mobility scenarios from the performance analysis presented in this chapter. In this example, the routing protocol was AODV and a 10-CBR 100 Kbps background load was included. Various performance and MAC-layer statistics are presented for the same simulation run in Table 5.8. The number of packets delivered and throughput are about 50% higher for ADP than UDP. Note that the packets successfully delivered by UDP took more hops between sender and receiver on average than did the packets delivered by ADP.

When we account for the network resources used by dropped packets, the advantage for ADP is even greater. For UDP, there were about 5.6 MBytes transmitted at the MAC layer for every MByte of data delivered. This figure includes the hops taken by packets that were dropped. For ADP, the ratio, which includes the ACK packets, is much smaller at 2.6. If we consider packets rather than bytes, the numbers still favor ADP. There were 4.3 packets transmitted at the MAC layer for every data packet delivered by ADP, while the ratio for UDP was higher at 5.1. ADP is clearly a much more efficient datagram protocol.

5.3 Concluding remarks

We believe that it is fundamentally important that a transport protocol be adaptive to network conditions in an environment as potentially dynamic as a mobile ad hoc network. While the approach adopted by some applications of using an aggressive UDP flow to “blast” packets across the Internet may be workable in the world of fixed, wireline networks, such tactics will not be viable when changes in the network topology and available capacity are rapid. We have shown in this chapter that it is possible to design an adaptive unreliable

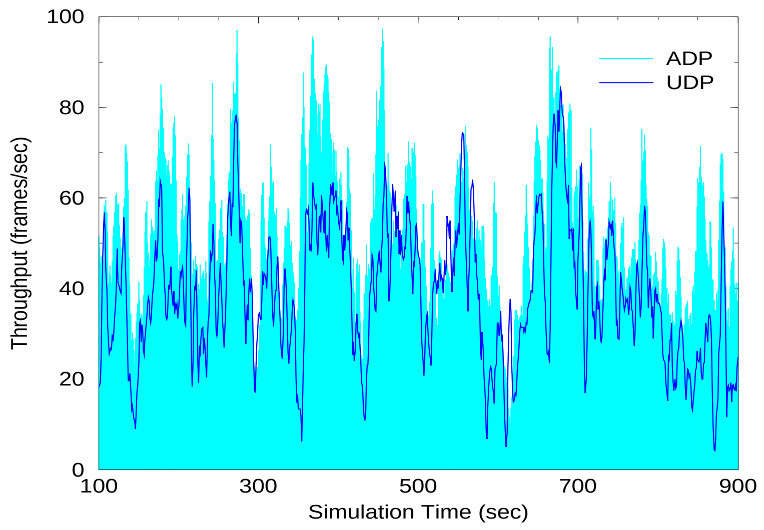


Figure 5.14: Throughput of 10 video connections using ADP and UDP. AODV routing protocol, 10-CBR 100 Kbps background load.

Table 5.8: Performance and MAC-layer statistics for 10 video connections. AODV routing protocol, 10-CBR 100 Kbps background load.

Performance Metric	**UDP**	**ADP**
Throughput (Mbps)	0.3347	0.4992
Packets Delivered	34,901	51,567
Hops Per Delivered Packet	2.40	2.11
Data Packets Transmitted at the MAC Layer (per Delivered Packet)	176,415 (5.1)	115,712 (2.2)
ADP ACKs Transmitted at the MAC Layer	0	106,818
Packets Transmitted at the MAC Layer (per Delivered Packet)	176,415 (5.1)	222,530 (4.3)
MBytes of Data Delivered	39.90	59.51
MBytes Transmitted at the MAC Layer (per Delivered MByte of Data)	222.23 (5.6)	153.09 (2.6)

packet delivery service that achieves high throughputs while being much more efficient than UDP in terms of network resource usage.

The Real-Time Transport Protocol [62] has been designed to use an unreliable datagram service like UDP to move delay-sensitive data while providing the application with the feedback required to adjust its sending rate to match the available network bandwidth. We believe that ADP is a viable substitute for UDP as the underlying transport protocol. Based on the feedback it receives from RTP, an application will be able to choose a combination of sending rate and ADP buffer size that gives a superior level of performance while staying within the application's timing constraints.

In designing ADP, we started with the premise that we needed a congestion control mechanism akin to the one used by TCP. This type of mechanism may be intuitive and is certainly nothing new. However, our insight was that the full complexity of a TCP-like windowing system is not necessary to achieve excellent results. We have shown that a simple ACK-clocking method is sufficient to enable an ADP flow to very efficiently utilize the available network capacity.

A potential criticism of any ACK-based mechanism is that the overhead incurred by the ACK traffic will offset the gains in performance afforded by the mechanism. In this regard, certain design changes could be made to ADP that would likely serve to reduce the number of ACKs without a negative impact on ADP performance. One promising idea that we have not yet investigated is the use of a delayed acknowledgement scheme similar to TCP's delayed ACKs. An 8-bit vector could be added to the ADP packet header in which the receiver gives the status of the 8 packets prior to the packet being acknowledged. In addition to enabling a delayed ACK to serve as acknowledgement of multiple data packets, the "previous-ACK" vector provides redundancy in the event acknowledgements are dropped by the network.

Chapter 6

Multimedia Traffi c in Mobile Ad hoc Networks

The bandwidth available to early implementations of mobile ad hoc networks has been limited. For example, the IEEE 802.11 standard provides link capacities of just 1-2 Mbps. This limited bandwidth is not adequate to support many applications which are typical of the wired Internet. However, newer standards like IEEE 802.11a are specifying bandwidths as high as 55 Mbps, and future technologies will provide even more capacity. This higher capacity will make it feasible for MANETs to carry the traffic from bandwidth-intensive sources such as multimedia applications.

In Chapter 4, we confined our performance analyses to TCP flows from simulated FTP file transfers. When we analyzed the performance of ADP in Chapter 5, we introduced simulated streaming video in order to make the simulated traffic a bit more realistic. In this chapter, we extend our investigation of TCP performance, and Reno-F in particular, to encompass Web traffic. We then combine the simulated Web and video flows to create a multimedia traffic load, and we examine the performance benefits conferred by TCP Reno-F and ADP in this setting.

6.1 Multimedia traffi c

Browsing the World Wide Web is a major activity on the Internet. Web traffic is carried by the Hyper-Text Transfer Protocol (HTTP) which runs on top of TCP. This traffic is characterized by highly variable bit rates and short-lived connections compared to other types of TCP traffic such as FTP file transfers.

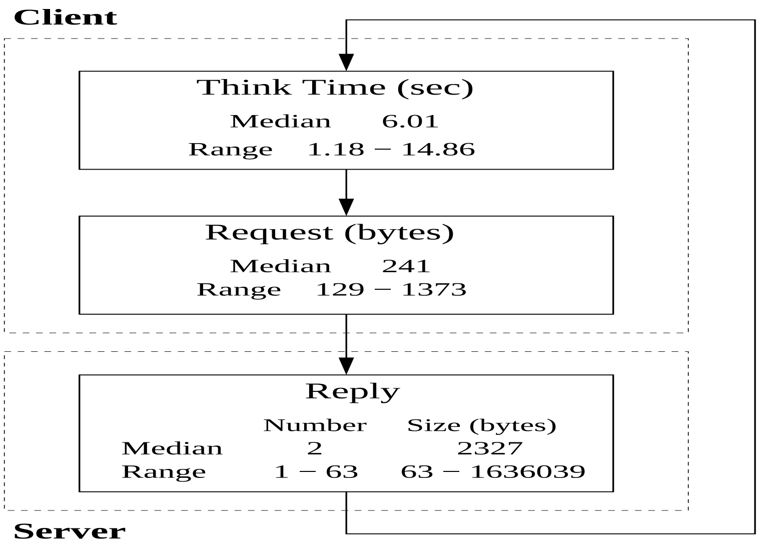


Figure 6.1: Client-server request-reply cycle in a simulated Web session.

Variable bit rates are also a characteristic of streaming video traffic. In this case, video frames are usually injected into the network at a constant rate but the size of these frames can vary dramatically. In Section 5.2.2, we discussed how we have simulated video traffic. In the following section, we consider the simulation of Web traffic.

6.1.1 Simulation of Web traffic

We utilized an HTTP traffic generator [28] to simulate the flow of information between browsers and servers during Web sessions. As depicted in Figure 6.1, each session consists of an alternating sequence of think and HTTP transaction modes. In the think mode, the client thinks for a random period of time and does not generate any network traffic. In an HTTP transaction, the client issues a request, and the server then responds with a random number of replies of variable length. For each transaction, the think time, the size of the request, the number of replies, and the lengths of the replies were drawn from the distributions supplied with the traffic generator. However, to keep the Web sessions short enough so that the majority of the client-server exchanges could be completed within the duration of the simulation, we truncated the think time distribution at 15 seconds. To make the simulations repeatable, we generated the Web sessions in advance and stored them in files which were used as inputs to the simulator.

6.2 Performance analysis

In this section, we present the results of our performance analysis. After describing our experimental setup, we compare the performance of ADV and AODV, with TCP Reno and Reno-F, for a TCP traffic load generated by simulated Web browser sessions. We then extend the performance comparison to include a mixture of the HTTP traffic generated by the Web sessions and multiple simulated video flows carried by UDP and by our proposed Adaptive Datagram Protocol.

6.2.1 Experimental methods

The simulation environment was as described in Section 3.4.1. We simulated a network of 50 nodes moving in a 1000m x 1000m square field. A background UDP traffic load of 100 Kbps was generated by 10 CBR connections, and the CBR packet sizes were fixed at 512 bytes. After a warm-up time of 100 seconds, simulated multimedia (HTTP or video) connections were established and performance data were collected for up to 200 seconds. The performance results were averaged over 50 different mobility scenarios. Based on the results of Section 4.5, we set ADV's buffer refresh time to 1 second for UDP packets and 30 seconds for TCP packets. A buffer refresh time of 30 seconds was used for AODV.

Using the HTTP traffic generator, we simulated 10 Web sessions in which browsers on 10 different mobile nodes issue requests and receive replies from Web servers running on 3 other nodes. In each simulation run, we measured service time, response time, and throughput for the HTTP connections. *Service time* is the time spent in the request-reply phase of a Web session, i.e. the client's think time is not included. If a Web session is not completed prior to the end of a simulation run, the service time for that session is the sum of the time spent in all completed request-reply cycles and any time spent in a request-reply cycle that is still in progress at the end of the simulation. Thus, for an unfinished session, the service time is equal to 200 seconds less the time spent by the client in think mode. For a finished session, the service time is the total time taken to complete the session less the think time. The mean service time is a simple arithmetic mean of the service times for the 10 Web sessions.

Response time is the interval between the sending of a client request and the receipt of the first packet

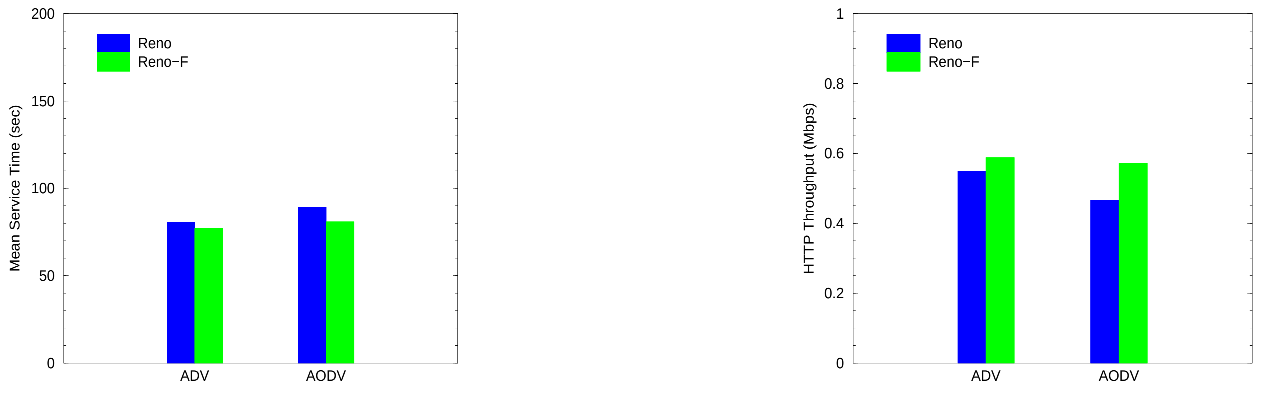


Figure 6.2: Mean service time and throughput for 10 HTTP connections with no CBR background traffic.

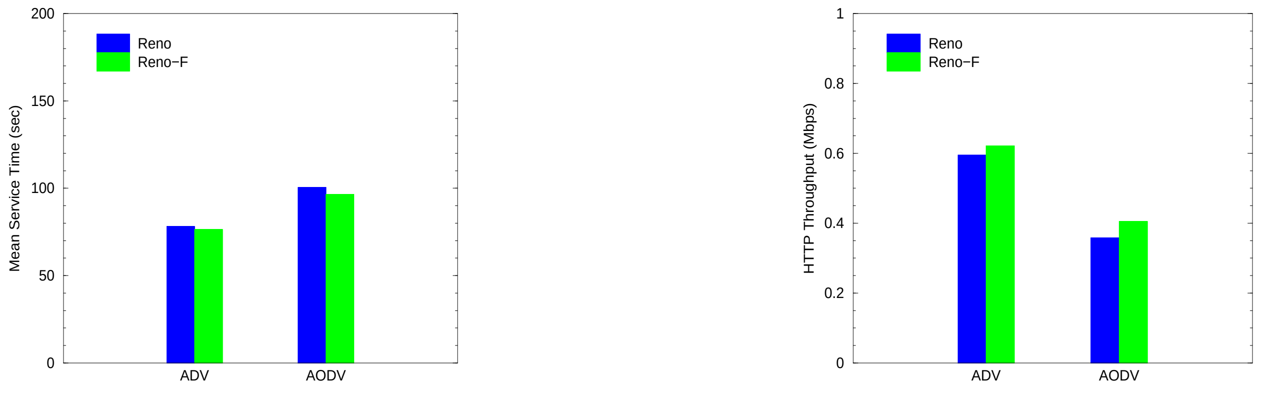


Figure 6.3: Mean service time and throughput for 10 HTTP connections with a 100 Kbps background load from 10 CBR sources.

of the server's reply. The average response time is the arithmetic mean of the response times for all request-reply cycles in the 10 Web sessions. HTTP *throughput* is computed as the total number of TCP bytes delivered for all 10 Web sessions, including the bytes delivered in any request-reply cycles that are still in progress at the end of the simulation, divided by the sum of the session service times. In the simulations which include ADP (UDP) traffic, we measured the throughput for the ADP (UDP) connections.

6.2.2 Performance results for HTTP traffic only

We began by evaluating the performance of the AODV and ADV routing protocols for HTTP traffic without any background network load from CBR traffic. As shown in Figure 6.2, AODV's HTTP performance improved with the use of Reno-F. The average time taken to complete a client-server session was reduced by 9% while HTTP throughput was 23% higher. Reno-F yielded only modest performance gains for ADV, however. This observation is consistent with the results of Chapter 4 where it was shown that, due to its proactive routing algorithm, ADV derives little benefit from the route stimulation of Reno-F.

Next, we added a 100 Kbps background load from 10 CBR sources to the simulations. The observed mean service time and HTTP throughput for the two routing protocols are shown in Figure 6.3. For AODV, the addition of the CBR traffic has a considerable impact on HTTP performance, reducing TCP Reno throughput by 23% and increasing the mean service time by 13%. As in the no-background case, Reno-F had a beneficial effect on the performance of AODV. Mean service time was reduced by 4% and throughput increased by 13%. Once again, Reno-F yielded only a modest improvement in ADV's performance.

Comparing AODV and ADV, we find that with interfering CBR traffic, ADV outperforms AODV by a wide margin. With TCP Reno, HTTP throughput was 66% higher and service times were 21% lower for ADV. With Reno-F, ADV still outperformed AODV by 54% in terms of throughput. As we have seen in Chapter 4, AODV's performance falls off with the higher level of routing activity needed to maintain a greater number of connections. ADV, due to its proactive approach to routing, is able to handle the additional CBR connections without incurring a performance penalty.

Figures 6.4 and 6.5 show the service and response times observed for each of the 10 client-server pairs. There is no evidence that some worst-case Web session is skewing the results and making the AODV performance results look worse. ADV maintained its advantage in service times in every case. As in the case of service times and HTTP throughput, Reno-F lessened the differences in response times for the two routing protocols. ADV yielded shorter response times for every client-server pair, although with Reno-F, its advantage over AODV was smaller. The response time includes the time taken to establish the connection from client to server over which the client's request will be relayed, and the time taken to create a connection in the opposite direction to carry the server's reply back to the client. Thus, the response time is a good indicator of how quickly a routing protocol is able to discover routes between the client and server nodes in an environment with numerous short-lived TCP connections. It is noteworthy that ADV's proactive approach outperforms AODV's route discovery mechanism in this setting.

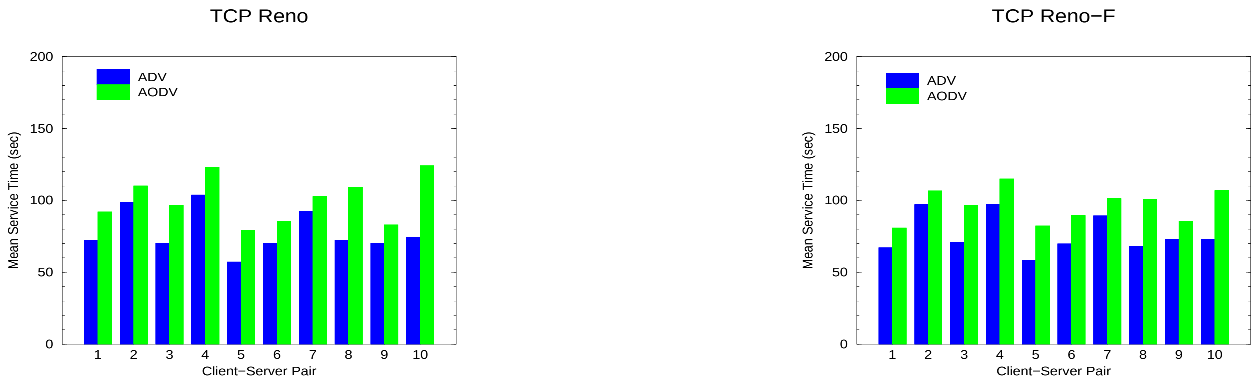


Figure 6.4: Service times for 10 HTTP server-client connections using TCP Reno and TCP Reno-F with a 100 Kbps background load from 10 CBR sources.

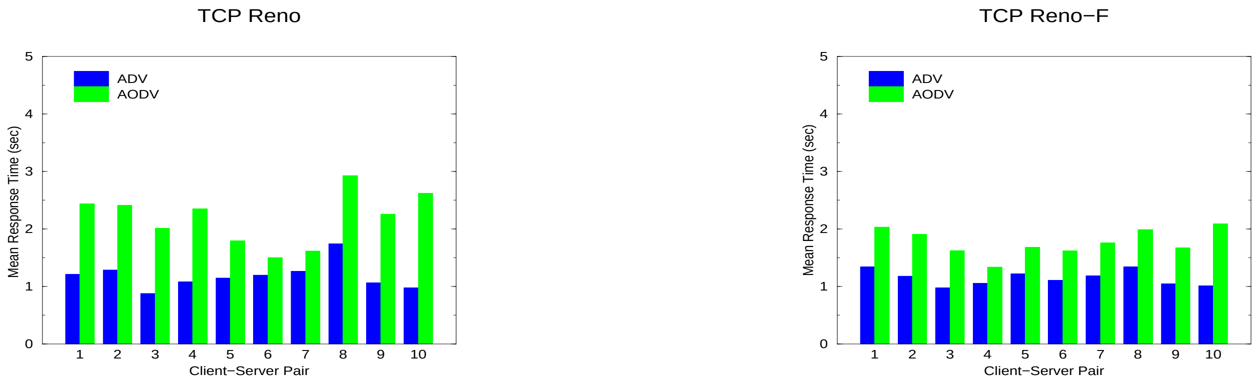
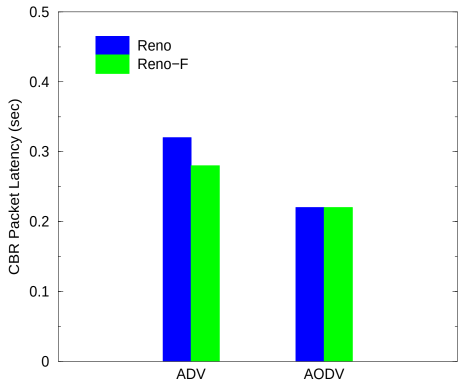


Figure 6.5: Response times for 10 HTTP server-client connections using TCP Reno and TCP Reno-F with a 100 Kbps background load from 10 CBR sources.

In Chapter 4, we observed that, for 10 FTP flows, both ADV and AODV achieved throughputs of nearly 1 Mbps with TCP Reno-F. For 10 HTTP flows, however, even the best-performing ADV obtained only 60% of the FTP throughput. To understand this disparity, consider the differences in FTP and HTTP traffic patterns. The FTP flows were simulated with infinite backlog. Once an FTP connection was established, packets were always ready to be sent. The HTTP flows, on the other hand, consist of a number of request-reply cycles interleaved with periods of think time. New HTTP connections between client and server must be established for every request-reply cycle, which means that an HTTP flow will spend a greater proportion of time in TCP's slow start phase than will a long-lived FTP connection. In the case of our simulated client requests, which are only 1.4 packets in length on average, an HTTP connection will never leave the slow start phase. During slow start, the size of the TCP sender's congestion window is smaller and throughput is lower compared to an established connection that is fully utilizing the bandwidth available to it. Moreover, the rate of TCP retransmit timeouts will be higher for HTTP flows because RTOs occur frequently during connection establishment. For example, with AODV, TCP Reno-F, and background CBR traffic, the number of timeouts per unit of connection time was observed to be over 60% higher for HTTP flows than for FTP flows. Retransmit timeouts cause the TCP sender to re-enter slow start and thus the higher rate of RTOs incurred by HTTP results in lower throughput compared to FTP.

Both protocols were able to handle the background CBR load quite well. Figure 6.6 shows the packet latency and throughput observed for the background CBR flows. The rate of packet loss was low at about 10%, and the average packet latency was in the 200-300 msec range. AODV yielded the lowest CBR packet latency and was not impacted by Reno-F, while for ADV, packet latency was about 12% lower with Reno-F. In nearly every mobility scenario, not all of the Web sessions were completed within 200 seconds, and on average, more request-reply cycles were completed with Reno-F than with Reno. When more request-reply cycles are completed, more periods of client think time occur and total client think time is greater. Thus, Reno-F, in general, minimized the duration for which TCP packets competed for network bandwidth. In addition, the use of delayed acknowledgements in Reno-F resulted in a significant reduction in TCP ACK traffic, as much as 15% for ADV. For ADV, reduced competition from TCP traffic is particularly effective in lowering CBR packet latency because ADV buffers packets at intermediate nodes. In contrast, the low



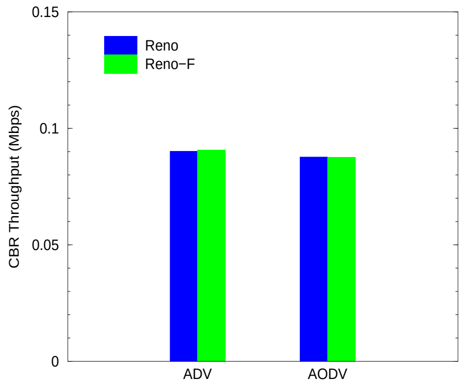


Figure 6.6: CBR packet latency and throughput for 10 HTTP connections with a 100 Kbps background load from 10 CBR sources.

packet latency yielded by AODV with TCP Reno leaves essentially no room for improvement. The reduced interference from HTTP traffic with Reno-F did not impact CBR throughput for either protocol since the network was not congested and the packet delivery rates were nearly as high as they could be even without the competing traffic [12, 20].

6.2.3 Performance results for combined multimedia traffic

We turn our attention now to analyzing the performance of ADV and AODV for a traffic load consisting of video flows as well as HTTP connections. In particular, we want to see how compatible our proposed datagram protocol, ADP, is with TCP Reno and Reno-F. Whereas a non-conforming UDP flow can be expected to steal bandwidth from a TCP connection, ADP's adaptivity should allow ADP and TCP to more equitably share available network capacity.

We measured the combined throughputs of various collections of UDP, ADP, and HTTP connections. The UDP and ADP connections carry simulated video streams as described in Chapter 5. The HTTP connections carry Web session traffic as described earlier in this chapter. The combinations we considered were 10 UDP flows, 10 ADP flows with 5-second buffering, 5 HTTP connections plus 5 UDP flows, and 5 HTTP connections plus 5 ADP flows with 5-second buffering. We also included the combination of 5 HTTP connections plus 5 ADP flows with unlimited buffering as the limiting case.

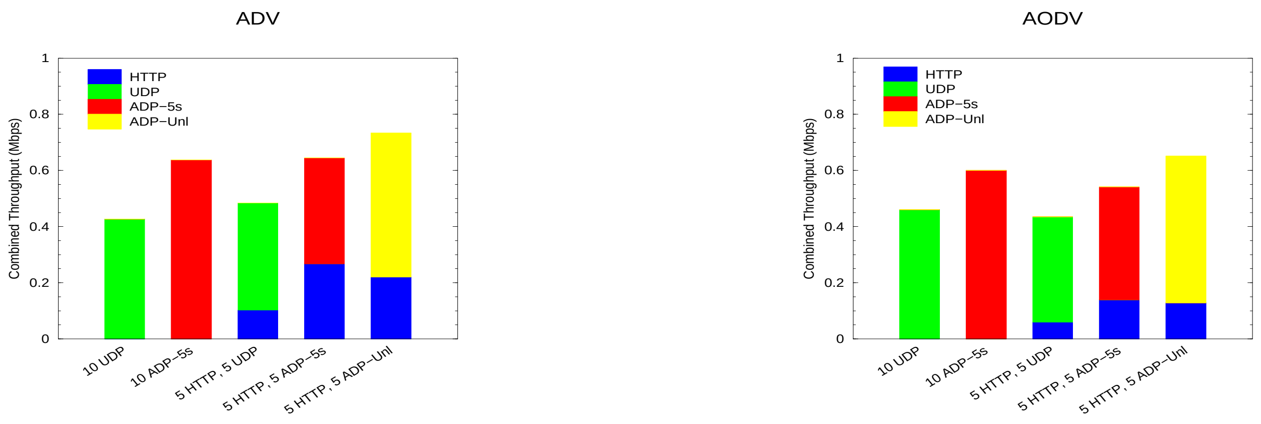


Figure 6.7: Combined throughputs for ADV and AODV for 10 connections with HTTP over TCP Reno and no CBR background traffic.

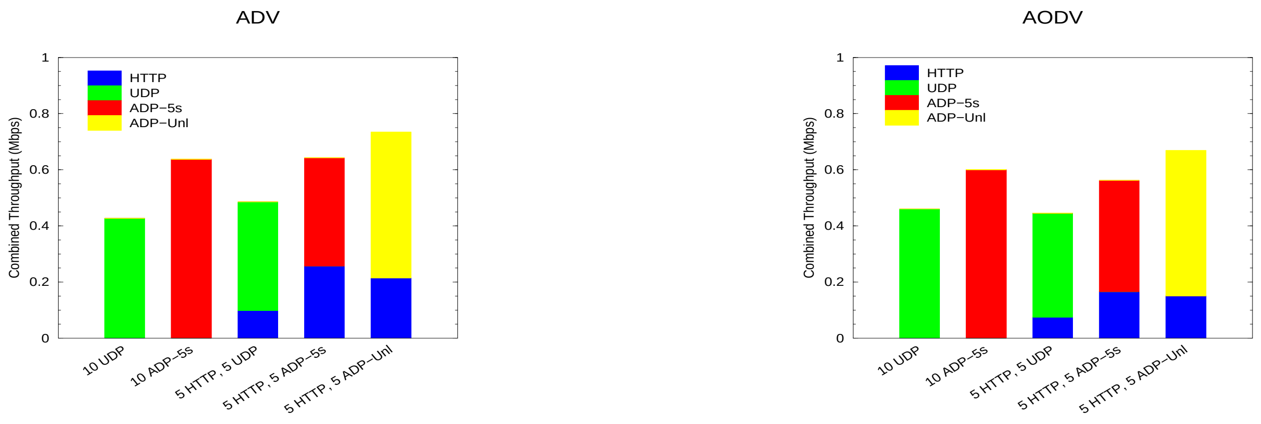


Figure 6.8: Combined throughputs for ADV and AODV for 10 connections with HTTP over TCP Reno-F and no CBR background traffic.

The combined throughputs obtained using TCP Reno and Reno-F, with no interfering CBR traffic, are shown in Figures 6.7 and 6.8. In Figures 6.9 and 6.10, a background load of 100 Kbps from 10 CBR sources has been added. These graphs contain a variety of information which is the basis for the following observations.

UDP and ADP throughput: First, we consider the UDP and ADP connections by themselves. Consistent with the results of Chapter 5, the 10 ADP flows achieved a larger combined throughput than the 10 UDP flows, about 50% larger in the case of ADV. As expected, both the UDP and ADP throughputs were lower in the presence of competing CBR traffic, with the reduction in ADP throughput being the greatest due to

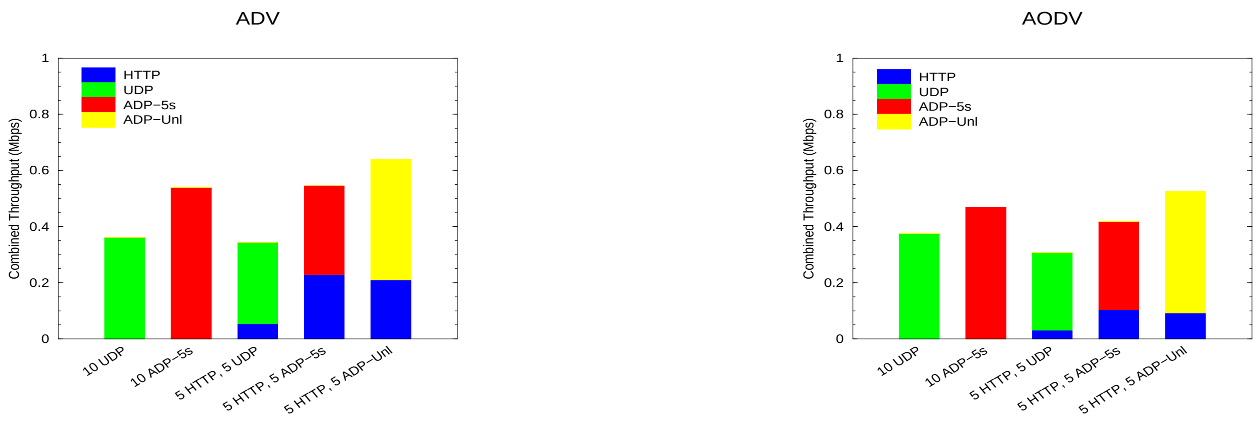


Figure 6.9: Combined throughputs for ADV and AODV for 10 connections with HTTP over TCP Reno and 100 Kbps from 10 CBR sources.

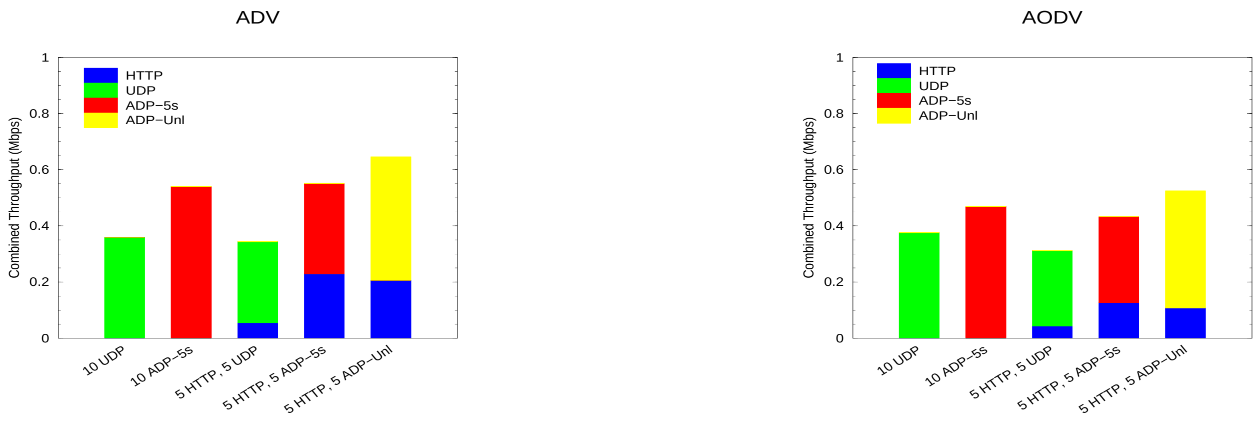


Figure 6.10: Combined throughputs for ADV and AODV for 10 connections with HTTP over TCP Reno-F and 100 Kbps from 10 CBR sources.

ADP's adaptive nature. Being non-TCP protocols, UDP and ADP were, of course, unaffected by the use of Reno-F.

Combined UDP and HTTP throughput: Next, we consider the combination of UDP and HTTP connections. In Figure 6.7, we see that for ADV, the HTTP and UDP flows together obtained a somewhat higher combined throughput than the UDP flows alone. The 5 UDP flows achieved approximately 90% of the throughput observed for 10 UDP connections. Comparing this to the results presented in Figure 5.6, in which the throughput of 5 UDP flows is about 95% of the 10-flow total, we see that the UDP flows were operating at nearly full capacity. Together with the fact that the HTTP flows accounted for only 21% of the combined HTTP/UDP throughput, this is an indication that UDP is consuming bandwidth to the exclusion of HTTP.

For AODV, the 5-UDP throughput was just over 80% of that observed for 10 UDP flows, again in line with expectations based on Figure 5.6. The contribution of the HTTP flows was even smaller than for ADV. In fact, the combined HTTP/UDP throughput was slightly less than was observed for 10 UDP flows. Again, it is apparent that UDP is attempting to monopolize the available bandwidth.

Combined ADP and HTTP throughput: We now consider the combination of HTTP and ADP with 5-second buffering. For ADV, the combined HTTP and ADP flows have captured all the bandwidth consumed by 10 ADP flows alone. With background CBR traffic, the 5 ADP flows obtained roughly 320 Kbps of throughput, which is 54% of the throughput observed for 10 ADP flows. In contrast, the throughput for 5 ADP flows in Figure 5.6 is just over 400 Kbps. At the same time, the HTTP flows achieved nearly 230 Kbps of throughput, which is almost 40% of the throughput shown for 10 HTTP connections using TCP Reno in Figure 6.3. In this case, the ADP flows have adapted to the presence of the HTTP traffic, and HTTP and ADP have established an equitable sharing of the available bandwidth.

With AODV, the sharing between ADP and HTTP is better than between UDP and HTTP, but it still heavily favors ADP. This imbalance is due to AODV's relatively poor HTTP performance. Based on Figure 5.6, the 5 ADP flows have consumed nearly as much bandwidth as possible, and still the combined HTTP/ADP throughput lags that of 10 ADP flows alone.

As was the case in the HTTP-only simulations, Reno-F provided a significant performance boost to the HTTP connections when AODV was the routing protocol. Reno-F had no impact on ADV's HTTP performance, however.

Looking at the combined throughput of HTTP and ADP with unlimited buffering, we see that HTTP obtained less throughput than it did when combined with ADP using 5-second buffering. Unlimited buffering allows ADP to take full advantage of the available capacity and this shifts the balance between ADP and HTTP in ADP's favor.

Figures 6.11 and 6.12 show the impact of combining UDP or ADP flows with HTTP traffic from the

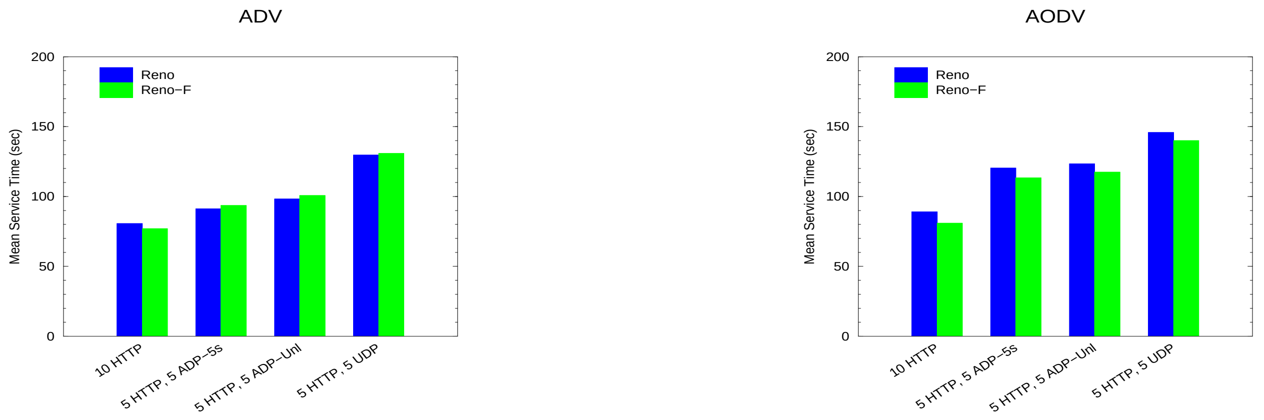


Figure 6.11: Mean HTTP service time for ADV and AODV for 10 connections with no CBR background traffic.

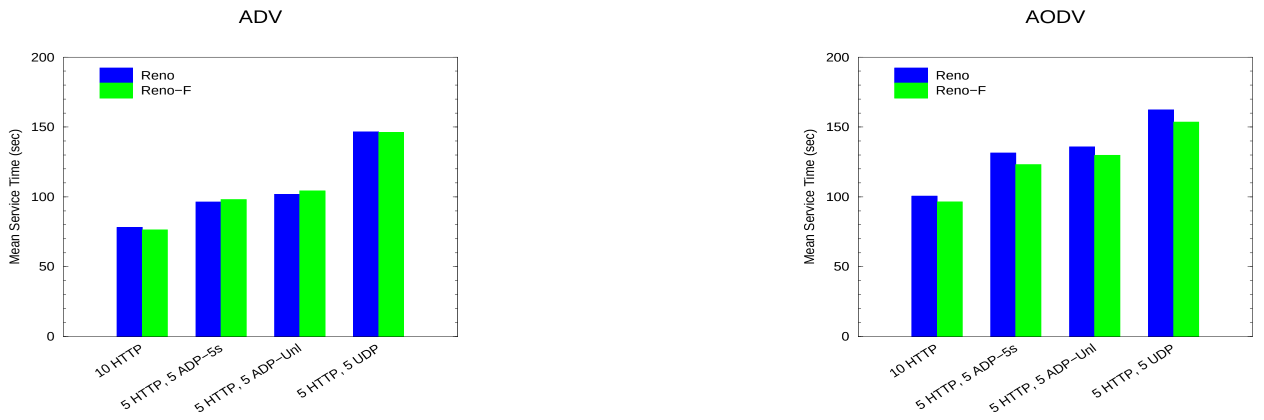


Figure 6.12: Mean HTTP service time for ADV and AODV for 10 connections with 100 Kbps from 10 CBR sources.

perspective of mean service time. When combined with video traffic carried by ADP, the Web sessions took substantially longer to finish, especially for AODV. Nevertheless, due to the inequitable sharing of available bandwidth, service times were by far the highest when the competing traffic was carried by UDP. The use of Reno-F resulted in measurable reductions in mean service time for AODV, but did not benefit ADV.

6.3 Concluding remarks

In Chapter 4, we showed that TCP Reno-F can significantly improve TCP performance in MANETs when the TCP traffic is produced by an FTP file transfer. In that case, only one TCP connection is opened and there are always packets ready to send. We have shown in this chapter that Reno-F also provides a significant boost to TCP performance for the short-lived connections and variable-bit-rate traffic that characterize Web browser sessions. We have also shown that the ADV routing protocol performs quite well in this environment. ADV compares favorably with AODV when the number of connections is small, and clearly outperforms AODV for a larger number of connections and higher volumes of offered traffic.

Our motivation in proposing the Adaptive Datagram Protocol of Chapter 5 was to create an alternative for the UDP protocol that would adapt its sending rate to the available network capacity and thereby achieve high throughputs while utilizing network resources efficiently. We have shown in this chapter that another important benefit of ADP is that it is able to share network capacity equitably with TCP rather than stealing bandwidth from TCP the way UDP does.

Given these results, we believe that the combination of TCP Reno-F and ADP (as the underlying transport protocol for RTP, perhaps) can significantly enhance the ability of a MANET to carry the multimedia traffic we expect to be present in the future.

Chapter 7

Conclusions

Wherever networking technology has led, the Internet has followed. This has been facilitated by the philosophy of Internet protocol design, which seeks to accommodate network heterogeneity. Nevertheless, extending Internet applications and services to mobile users in an ad hoc network is a challenging task. An important, and interesting, part of meeting this challenge is the design of routing and transport layer protocols that perform well in mobile ad hoc networks.

In this chapter, we summarize our work and describe the contributions of this dissertation. We begin with a look at MANET routing protocol design and a discussion of the performance advantages of an adaptive approach to proactive route maintenance. We review the results of our performance comparison of different routing protocols, and we note the work we have done to tune ADV for good performance. Next, we discuss the sender-side approach to improving TCP performance that we have used in our TCP Reno-F protocol, and we summarize the results of our TCP performance analysis, which we believe validate our method. We then consider the various performance benefits of our proposed Adaptive Datagram Protocol, and we discuss how ADP and Reno-F together can improve multimedia performance in MANETs. Finally, we summarize our contributions to MANET performance analysis, and we conclude by noting some possible directions for future research.

7.1 Routing protocol design

Two of the leading proposed MANET routing protocols, DSR and AODV, take an on-demand approach to route maintenance, discovering and repairing routes on a strictly as-needed basis. This approach has been shown to be superior to the purely proactive routing algorithm of the DSDV protocol, in which routes to all possible destinations, whether in use or not, are maintained by means of periodic and triggered updates. In Chapter 3, we examined the advantages and disadvantages of the adaptive, proactive approach to route maintenance taken in the proposed Adaptive Distance Vector (ADV) routing protocol. Using the *ns-2* network simulator with 802.11 wireless LAN extensions, we compared the performances of AODV, DSR, DSDV, and ADV for varying traffic loads generated by CBR connections.

Our performance analysis clearly shows that ADV's adaptive update strategy, in which the frequency and size of route updates depend on the number of connections and volume of traffic, gives much better performance than the traditional distance vector algorithm in DSDV. Furthermore, ADV is able to limit the frequency of route updates, and hence the amount of routing overhead, and still do an excellent job of route maintenance. Its lower routing overhead gives ADV a clear throughput advantage over the on-demand protocols for medium to high levels of offered traffic. At lower traffic loads or when the number of connections is small, the on-demand algorithms do offer superior performance. However, it should be possible to tune ADV to address this shortcoming.

We have made modifications to ADV in order to improve its performance. In particular, we extended the connection-initiation process to facilitate quicker establishment of TCP connections, where the traffic between sender and receiver will be flowing in both directions. An interesting finding of our TCP performance analysis in Chapter 4 is that it is better to buffer UDP packets for short intervals and buffer TCP packets for longer periods of time. So, we have modified ADV to employ two different buffer timeout values, one for TCP traffic and one for non-TCP traffic.

7.2 Improving TCP performance

The performance of TCP in MANETs, and how best to maximize that performance, is an area of research that is as yet largely unexplored. Techniques developed to improve TCP performance in cellular networks and over satellite links are not helpful in multi-hop wireless networks. Researchers have primarily focused on route failure notification schemes designed to mitigate the problems caused by TCP's congestion-control response to the noncongestion-related losses common in mobile, wireless environments. Another idea currently under investigation is to reliably predict impending route failures due to mobility so that alternate routes can be identified before failures actually occur.

We have proposed two techniques for improving TCP performance in MANETs which are based on sender-side heuristics. The first technique, fixed RTO, is designed to address the negative impact on performance resulting from the exponential backoff of TCP's retransmit timeout interval (RTO). In our proposal, when retransmit timeouts occur consecutively, i.e. no ACK is received for the retransmitted packet, we double the RTO on the first timeout only. The RTO is not doubled on succeeding consecutive timeouts, but rather is fixed. The idea is that, in a MANET, consecutive timeouts are likely to be the result of route failure rather than congestion. By increasing the frequency with which packets are retransmitted during route failures, the fixed RTO method is intended to reduce the time taken to re-establish broken routes.

Our second proposal is to introduce a hold-down interval following a retransmit timeout. During this period, any ACKs that arrive are processed in the usual way except that they do not trigger packet retransmissions. The hold-down period ends, and normal TCP processing is resumed, when all the packets outstanding when the timeout occurred have been ACKed or when the hold-down timer expires. The hold-down timer technique is intended to prevent unnecessary packet retransmissions when the outstanding ACKs are simply delayed and arrive too late to avoid a timeout.

We evaluated the proposed TCP techniques using three MANET routing protocols. For this analysis, we varied the number of TCP connections, the volume of background CBR traffic, and the number of CBR connections. In addition to evaluating the effectiveness of our proposed sender-side heuristics, we assessed the performance gains afforded by two existing TCP options, selective acknowledgements and delayed acknowledgements.

TCP Reno-F

The performance analysis presented in Chapter 4 shows that our proposed fixed-RTO mechanism, in combination with TCP's selective acknowledgements and delayed acknowledgements options, yields substantial improvements in TCP performance for the on-demand protocols. We call this combination of fixed RTO and TCP options the TCP Reno-F protocol. Reno-F works well for routing protocols with route maintenance

Table 7.1: Ranking of TCP Reno throughputs obtained using each routing protocol: 1 = highest, 3 = lowest.

#TCP	**#CBR**	**Traffic**	**DSR**	**AODV**	**ADV**
1	10	low	3	2	**1**
1	10	high	3	2	**1**
1	40	low	3	2	**1**
1	40	high	2	3	**1**
10	10	low	2	3	**1**
10	10	high	2	**1**	**1**
10	40	low	2	3	**1**
10	40	high	2	2	**1**

Table 7.2: Ranking of TCP Reno-F throughputs obtained using each routing protocol: 1 = highest, 3 = lowest.

#TCP	**#CBR**	**Traffic**	**DSR**	**AODV**	**ADV**
1	10	low	2	**1**	**1**
1	10	high	2	**1**	**1**
1	40	low	2	2	**1**
1	40	high	2	2	**1**
10	10	low	**1**	2	3
10	10	high	**1**	**1**	2
10	40	low	**1**	2	3
10	40	high	**1**	**1**	2

mechanisms that can be stimulated by increasing the rate of packet retransmissions in response to the consecutive timeouts that occur during route failure. Reno-F is also beneficial in situations where route repair times are extended. For example, a large increase in the number of hops from sender to receiver can result in lengthy route discovery delays for on-demand protocols. In contrast, ADV's proactive route repairs are quick enough on average that Reno-F is able to provide only modest benefits.

Tables 7.1 and 7.2 rank the TCP throughputs obtained with TCP Reno and Reno-F by each of the routing protocols. It is noteworthy that, with Reno-F, both AODV and DSR outperformed ADV as the number of TCP connections was increased.

Hold-down timer

Adding the hold-down timer yielded some limited additional gains in throughput in the 1-TCP connection case. This benefit was not observed for multiple TCPs, however. The hold-down timer is designed to mitigate the negative effect that a train of ACKs, arriving in rapid order after a route has been re-established, can have on the sender's packet retransmission mechanism. If this scenario was less likely to occur with multiple TCP flows, that could explain the hold-down timer's reduced effectiveness. Nevertheless, we believe the gains derived from the hold-down timer for a single TCP connection demonstrates the feasibility of improving TCP performance by the use of sender-based mechanisms.

7.3 Adaptive Datagram Protocol

We observed in our simulations that, in MANETs, TCP is capable of achieving higher throughput than UDP, a departure from normal expectations in a wired network. We believe the explanation for this result is that TCP is able to conform its send rate to the rapid changes in available bandwidth that can occur as a result of changes in path length from sender to receiver, while UDP simply transmits data at a fixed rate. Because the wireless network is a shared medium, the resulting contention at the MAC layer may have a negative impact on UDP throughput over and beyond simple packet loss.

We have designed an adaptive unreliable packet delivery service in which the receiver acknowledges delivered packets and the sender injects new packets into the network at the rate at which it receives these ACKs. Packets are buffered by the sender pending transmission. The size of the buffer is set by the application. If the application generates packets at a faster rate than the network can handle, causing the buffer to fill up, packets are dropped from the buffer (oldest packets first).

For performance reasons, we considered using a windowing algorithm based on TCP's congestion window. However, we discovered that very good performance can be attained using a simple ACK-clocking scheme, in which the arrival of any acknowledgement (as opposed to a specific outstanding ACK) triggers the next packet transmission. A transmit timer similar to TCP's retransmit timer is used to keep the sender from stalling in the event of packet or ACK loss.

In Chapter 5, we showed that this protocol, which we call the Adaptive Datagram Protocol, is able to achieve high throughputs while very efficiently utilizing network resources. We have shown that with unlimited packet buffering and a sufficiently high application sending rate, ADP is able to achieve throughputs as high as those obtained with TCP. ADP's efficiency derives from the fact that when the application rate exceeds network capacity, ADP drops the excess packets from its buffer rather than injecting them into the network where they will consume bandwidth and router queue space, only to be dropped before reaching the receiver. Furthermore, ADP sends more packets when the number of hops between sender and receiver is small, and fewer packets when the the number of hops is large. This reduces the average number of hops taken by successfully delivered packets, and thus reduces the volume of MAC-layer traffic.

The Real-Time Transport Protocol (RTP) is used in the wired Internet to deliver delay-sensitive traffic via an underlying transport protocol, typically UDP, while providing feedback the application can use to adjust its sending rate as necessary to satisfy its latency constraints. We envision ADP as an alternative to UDP as the underlying transport protocol when RTP-based applications are deployed in a mobile ad hoc network.

7.4 Performance analysis

A number of studies have compared the performance of different MANET routing protocols for UDP traffic. There is a lack, however, of such studies for TCP traffic, either by itself or in combination with UDP traffic. This is indicative of the early stage of MANET transport layer protocol research. We believe it is important to conduct these evaluations, because they help to elucidate performance problems and serve as benchmarks for judging the merit of techniques proposed to alleviate those problems.

To our knowledge, ours is the most extensive study of TCP performance over MANET routing protocols to date. Only one study that we are familiar with has considered more than one routing protocol [2]. Ours is the first study to measure performance for multiple TCP connections, and the first to use performance metrics other than file transfer time or throughput. It is also the first performance evaluation to include background UDP flows along with the TCP traffic.

We have also utilized two types of simulated network traffic that, to our knowledge, have not been used in other MANET performance studies. For our analysis of ADP performance in Chapter 5, we simulated variable-bit-rate traffic modeled after the simulated video traffic described in [41]. In Chapter 6, we used an HTTP traffic generator [28] to extend our TCP performance analysis to include the traffic from simulated Web browser sessions. We then combined our simulated video and HTTP flows to create a multimedia workload for testing how fairly ADP and TCP are able to share available network capacity.

7.5 Future research directions

Our research has demonstrated that performance benefits can be derived from making transport protocols more adaptive to the dynamic network conditions in a mobile ad hoc network. This work points the way to several avenues for future research.

In the concluding remarks of Chapter 5, we noted that it should be possible to reduce the volume of ACK traffic in the Adaptive Datagram Protocol, thereby increasing the efficiency of ADP. One option is to implement delayed acknowledgements, whereby ACKs are sent in response to the receipt of two or more data packets rather than for every packet. Because the ADP sender uses a transmit timer, it may not be necessary to implement a timer at the receiver to ensure a maximum delay. Another option is to add an "ACK-vector" to the ADP packet header. The receiver will use this vector to inform the sender of the acknowledgement status of some number of packets with sequence numbers preceding the packet currently being ACKed. The use of an "ACK-vector" can provide redundancy in the event any of the previous ACKs were lost.

We envision ADP as a transport protocol for multimedia traffic, for example streaming video. In this regard, we are interested in pursuing the idea of using ADP instead of UDP as the underlying transport protocol used by the Real-Time Transport Protocol (RTP). We believe an interesting experiment would be to compare the quality of video images transmitted over a MANET using RTP and ADP with that of images carried by RTP and UDP.

We would like to continue our experiments with TCP Reno-F. A concern that might be raised with respect to the fixed-RTO technique is that this departure from standard TCP congestion control may lead to network congestion problems, especially if Reno-F is used in a fixed network. We believe this concern can be addressed by fixing the RTO for the first few consecutive timeouts and then reverting to the standard exponential backoff of the RTO. The bandwidths in today’s Internet are two or more orders of magnitude greater than were available in the 1980’s when TCP’s congestion control mechanisms were first developed. It may be the case that a limited number of packet retransmissions has such a negligible impact these days that we can delay the exponential backoff of the RTO long enough to obtain the performance gains of Reno-F without increasing network congestion measurably.

Another interesting topic for further research is the interplay of node mobility and transport protocol performance. We believe it would be useful to measure transport protocol performance over a wide range of node mobilities. There is an ongoing debate among MANET researchers as to the impact of mobility on performance. It might seem intuitive to expect node mobility, which causes route failure, to always have a negative impact on throughput. In fact, node mobility may increase throughput by virtue of facilitating quicker route repair.

Appendix A

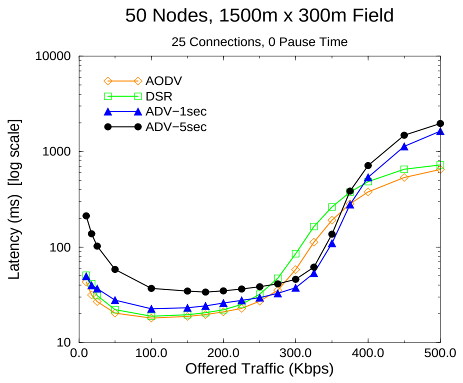
Additional ADV Performance Analysis Results

A.1 Steady state behavior of a high mobility network

In this section, we present additional performance results showing the steady state behavior of a high mobility network.

A.1.1 50-node 1500m x 300m network, 25 connections

Figures A.1 - A.3 give the results for 25 CBR connections in a 50-node network moving in a 1500m x 300m field. The node density in this field is higher than in the 1000m x 1000m square field in the performance analysis of Chapter 3, and there are not sudden, large changes in path lengths like those that can occur when nodes wrap-around in the square field. As a consequence, route failures in the rectangular field are shorter on average and fewer in number, as evidenced in Figure A.3 by the lower level of routing activity for the on-demand protocols compared to the square-field 25-connection case (see Figure 3.5). Because ADV route updates are proactive, the level of routing activity is about the same for the two fields. All three protocols yield higher throughputs than they did in the square field. In particular, DSR's peak throughput improves from just under 200 Kbps to around 300 Kbps. As shown in Figure A.1, AODV and DSR saturate at much higher levels of offered traffic than before, and delivery fractions are very high for all three protocols, with ADV delivering nearly 100% of the CBR packets.



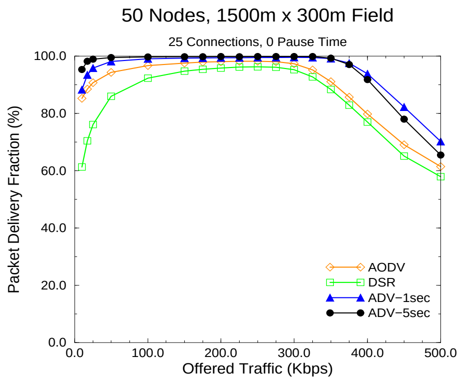


Figure A.1: Packet latency and delivery fraction for 25 connections in a 50-node network on a 1500m x 300m field.

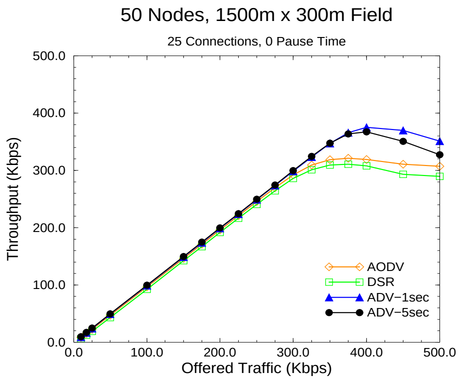
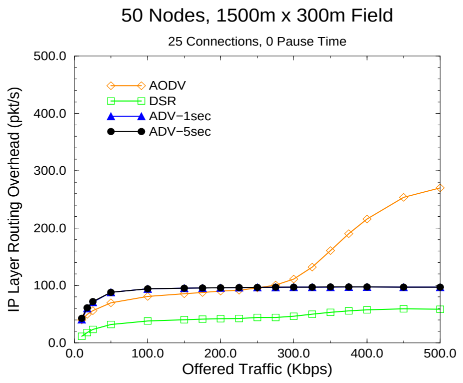


Figure A.2: Throughput for 25 connections in a 50-node network on a 1500m x 300m field.



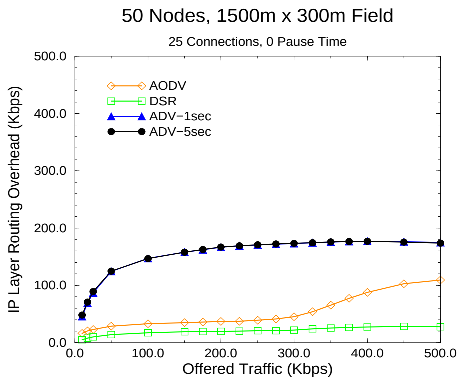
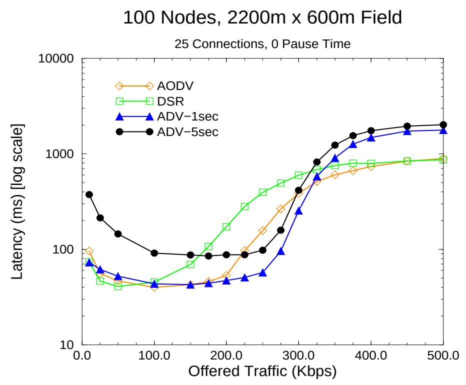


Figure A.3: IP-layer routing overhead for 25 connections in a 50-node network on a 1500m x 300m field.



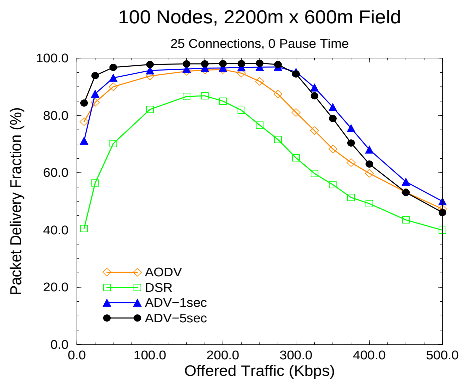


Figure A.4: Packet latency and delivery fraction for 25 connections in a 100-node network on a 2200m x 600m field.

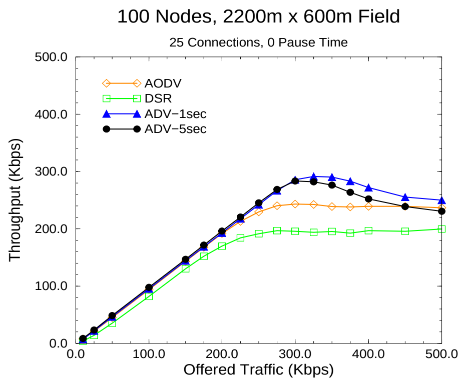
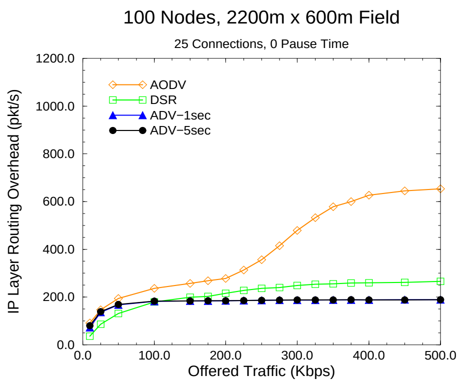


Figure A.5: Throughput for 25 connections in a 100-node network on a 2200m x 600m field.

A.1.2 100-node 2600m x 600m network, 25 connections

Figures A.4 - A.6 give the results observed for 25 CBR connections in a 100-node network moving in a 2600m x 600m field. The node density in this field is roughly half that of the 1500m x 300m field. Comparing Figure A.4 with Figure A.1, we see that the lower node density results in higher packet latencies and lower packet delivery rates, especially for DSR. All three routing protocols saturate at lower levels of offered traffic.



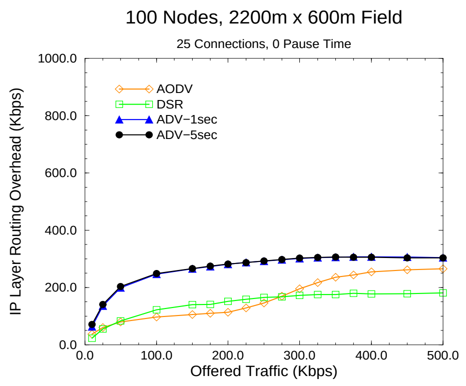
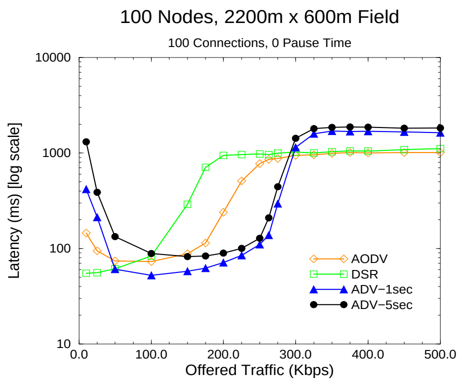


Figure A.6: IP-layer routing overhead for 25 connections in a 100-node network on a 2200m x 600m field.



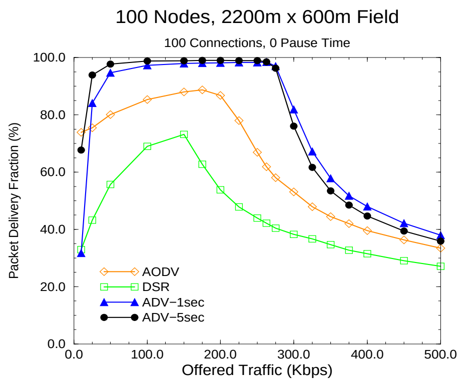


Figure A.7: Packet latency and delivery fraction for 100 connections in a 100-node network on a 2200m x 600m field.

A.1.3 100-node 2600m x 600m network, 100 connections

Figures A.7 - A.9 give the results observed for 100 CBR connections in a 100-node network moving in a 2600m x 600m field. Managing the larger number of connections requires a substantially higher level of routing overhead for all three routing protocols, as shown in Figure A.9. The impact of more connections on packet latency, delivery fraction, and throughput is greatest for the on-demand protocols. Although it saturates at a lower level of offered traffic, ADV is able to achieve a very high packet delivery rate for network loads below its saturation point.

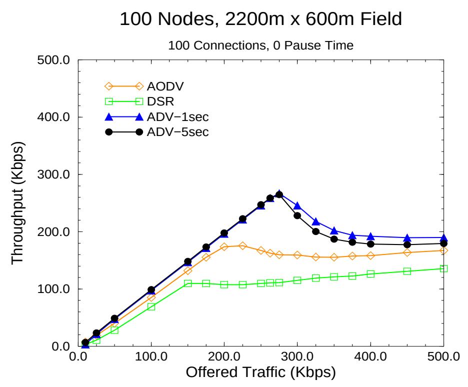
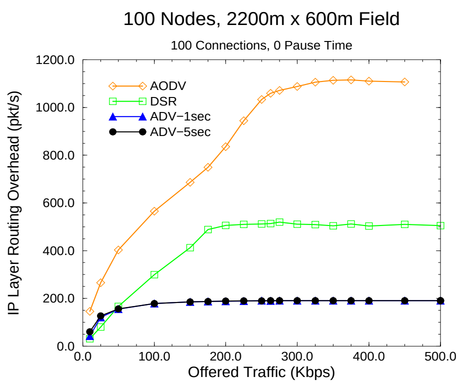


Figure A.8: Throughput for 100 connections in a 100-node network on a 2200m x 600m field.



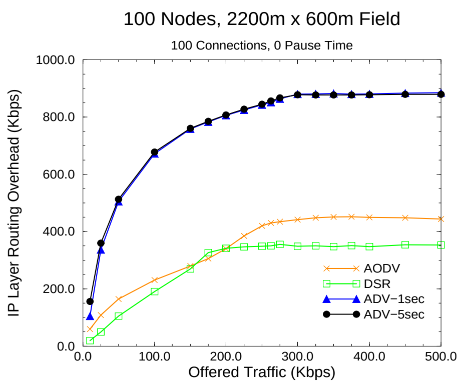
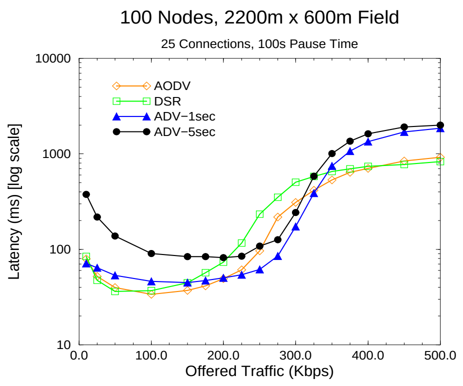


Figure A.9: IP-layer routing overhead for 100 connections in a 100-node network on a 2200m x 600m field.



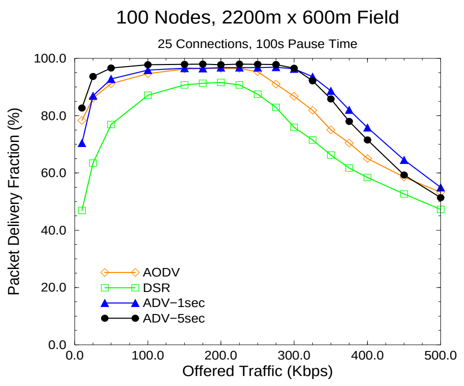


Figure A.10: Packet latency and delivery fraction for 25 connections in a low-mobility 100-node network on a 2200m x 600m field.

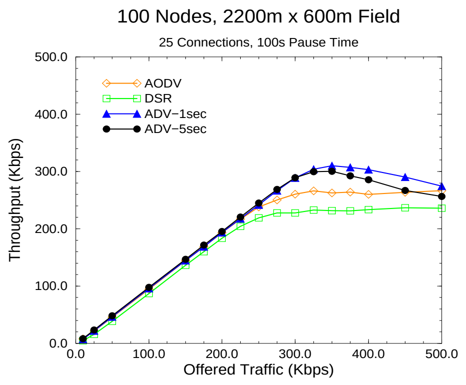


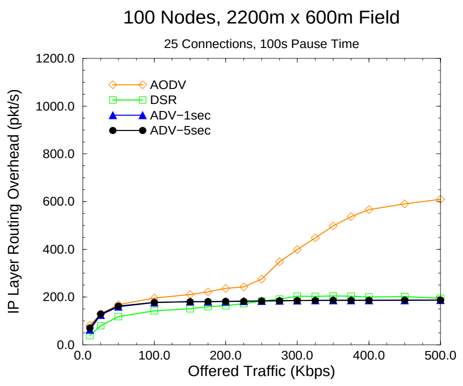
Figure A.11: Throughput for 25 connections in a low-mobility 100-node network on a 2200m x 600m field.

A.2 Steady state behavior of a low mobility network

In this section, we present additional performance results showing the steady state behavior of a low mobility network. Low node mobility was simulated by setting the pause time to 100 seconds. As a result of reduced node mobility, the routing overhead was somewhat reduced for the on-demand routing protocols and all three protocols saturated at slightly higher levels of offered traffic.

A.2.1 100-node 2600m x 600m network, 25 connections

Figures A.10 - A.12 give the results observed for 25 CBR connections in a 100-node network moving in a 2600m x 600m field.



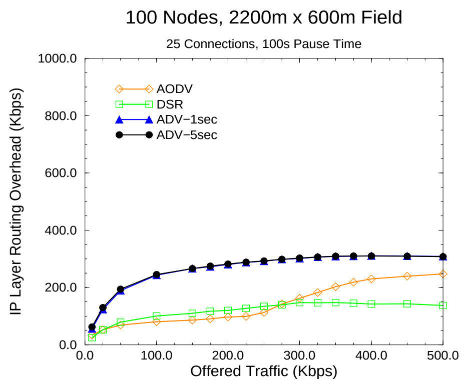
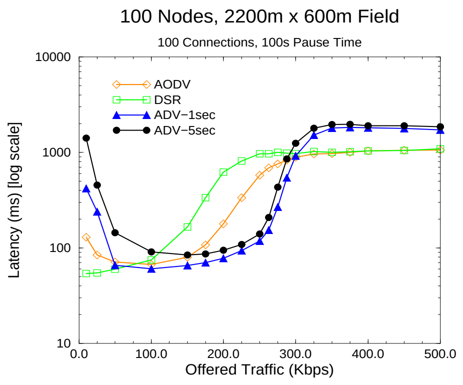


Figure A.12: IP-layer routing overhead for 25 connections in a low-mobility 100-node network on a 2200m x 600m field.



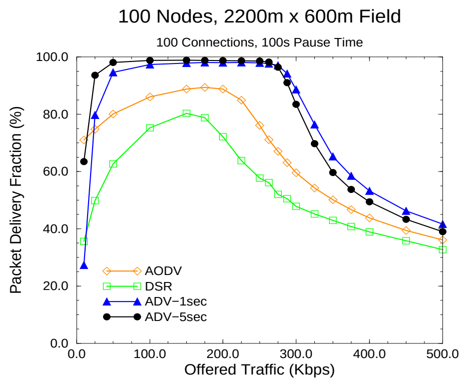


Figure A.13: Packet latency and delivery fraction for 100 connections in a low-mobility 100-node network on a 2200m x 600m field.

A.2.2 100-node 2600m x 600m network, 100 connections

Figures A.13 - A.15 give the results observed for 100 CBR connections in a 100-node network moving in a 2600m x 600m field.

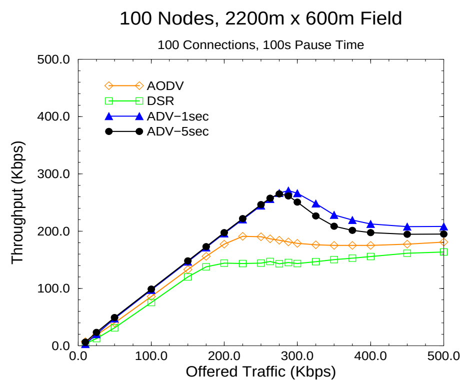
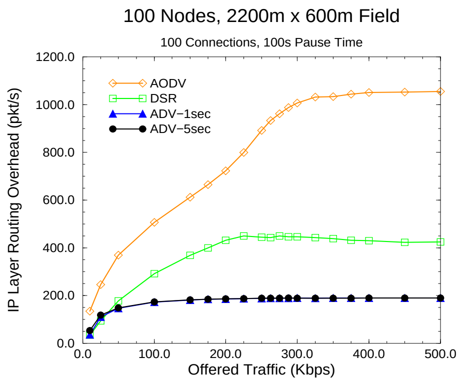


Figure A.14: Throughput for 100 connections in a low-mobility 100-node network on a 2200m x 600m field.



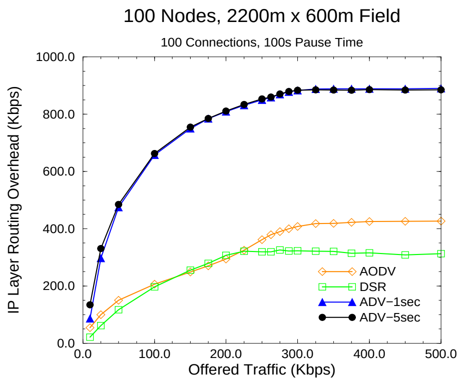
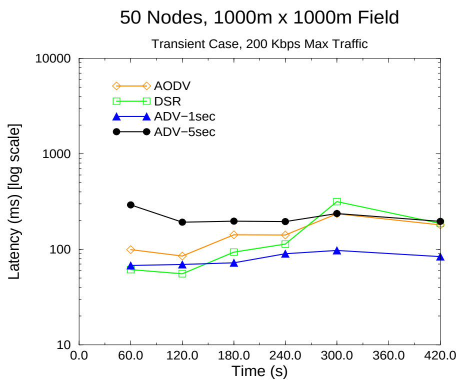


Figure A.15: IP-layer routing overhead for 100 connections in a low-mobility 100-node network on a 2200m x 600m field.



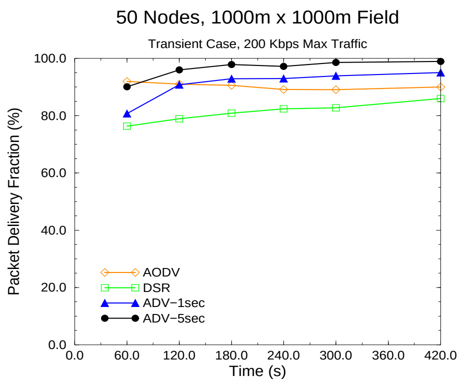


Figure A.16: Packet latency and delivery fraction for transient case with 200 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

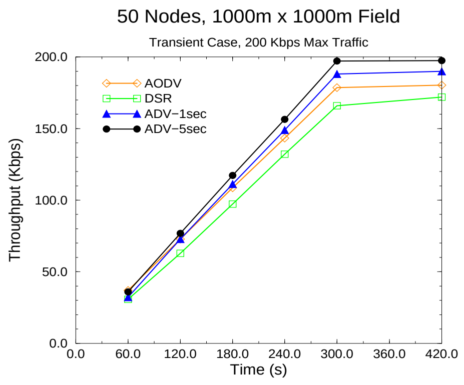
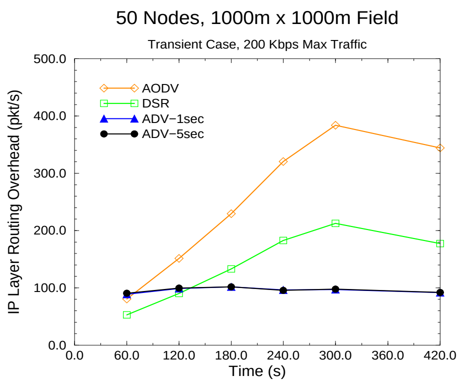


Figure A.17: Throughput for transient case with 200 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

A.3 Transient behavior of a high mobility network

In this section, we present additional performance results showing the transient state behavior of a high mobility, 50-node network in the square field. The maximum offered traffic in Figures A.16 - A.18 is 200 Kbps. In Figures A.19 - A.21, the offered traffic peaks at 300 Kbps.



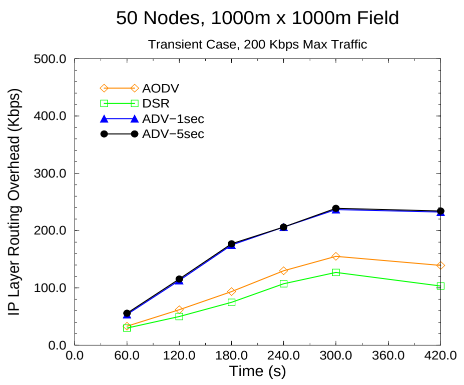
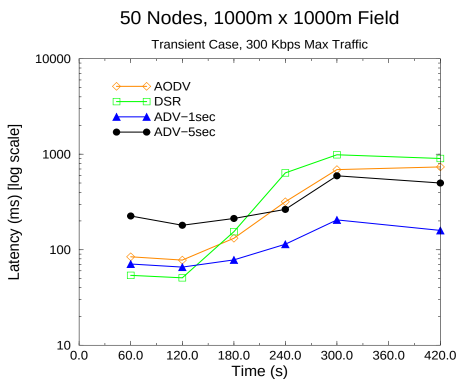


Figure A.18: IP-layer routing overhead for transient case with 200 Kbps max traffic in a 50-node network on a 1000m x 1000m field.



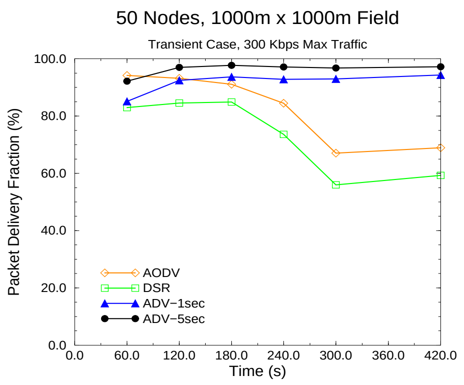


Figure A.19: Packet latency and delivery fraction for transient case with 300 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

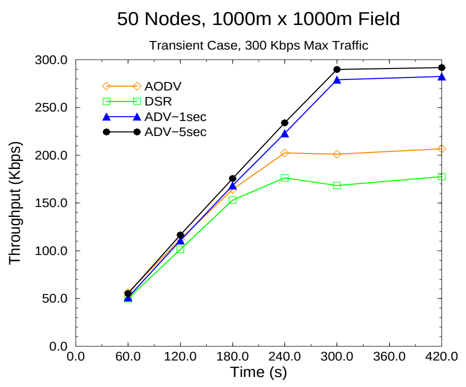
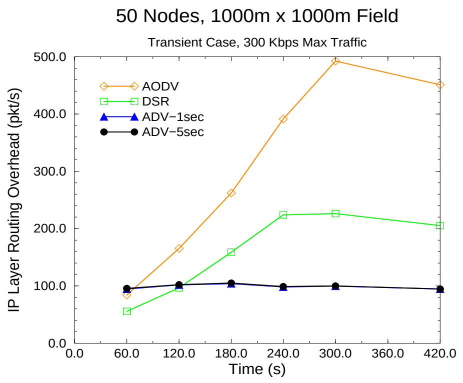


Figure A.20: Throughput for transient case with 300 Kbps max traffic in a 50-node network on a 1000m x 1000m field.



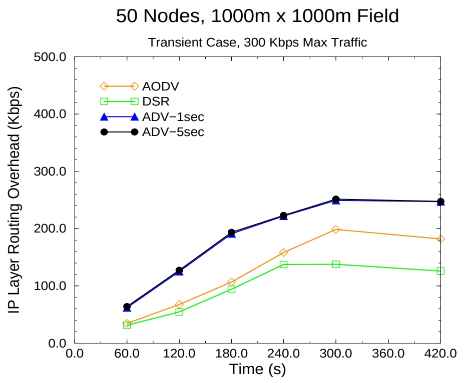


Figure A.21: IP-layer routing overhead for transient case with 300 Kbps max traffic in a 50-node network on a 1000m x 1000m field.

Appendix B

Additional TCP Performance Analysis Results

B.1 Performance results for 1 TCP connection

In this section, we present additional performance results for 1 TCP connection in a 50-node network moving in a 1000m x 1000m field. These simulations included background traffic loads of 100 Kbps and 200 Kbps from 10 and 40 CBR connections.

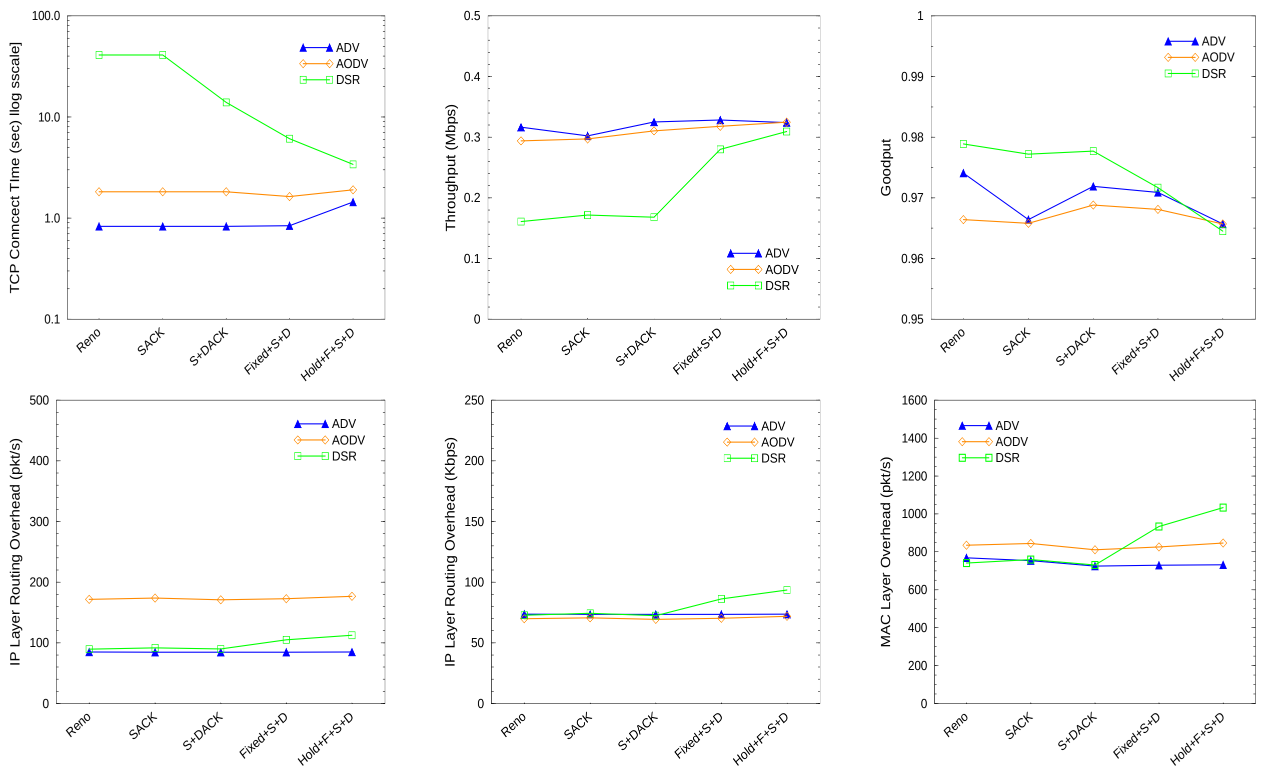


Figure B.1: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 100 Kbps background load from 10 CBR connections.

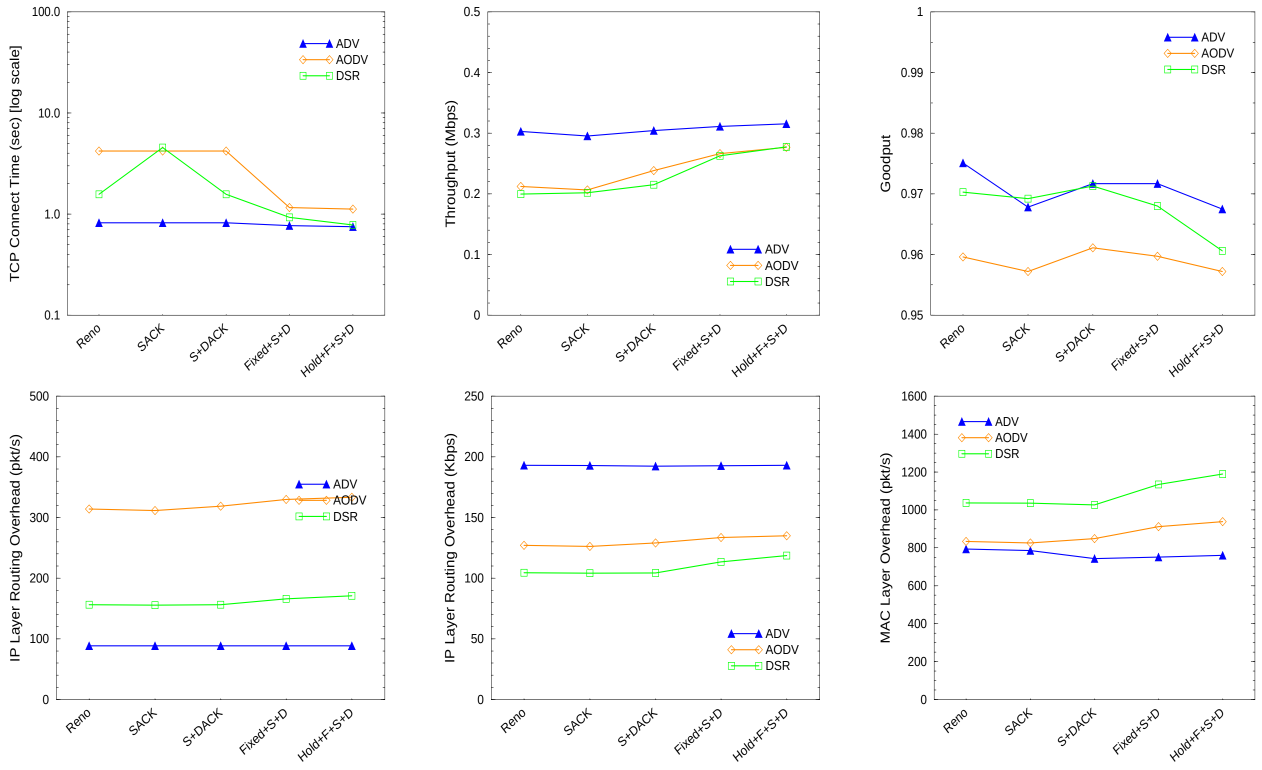


Figure B.2: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 100 Kbps background load from 40 CBR connections.

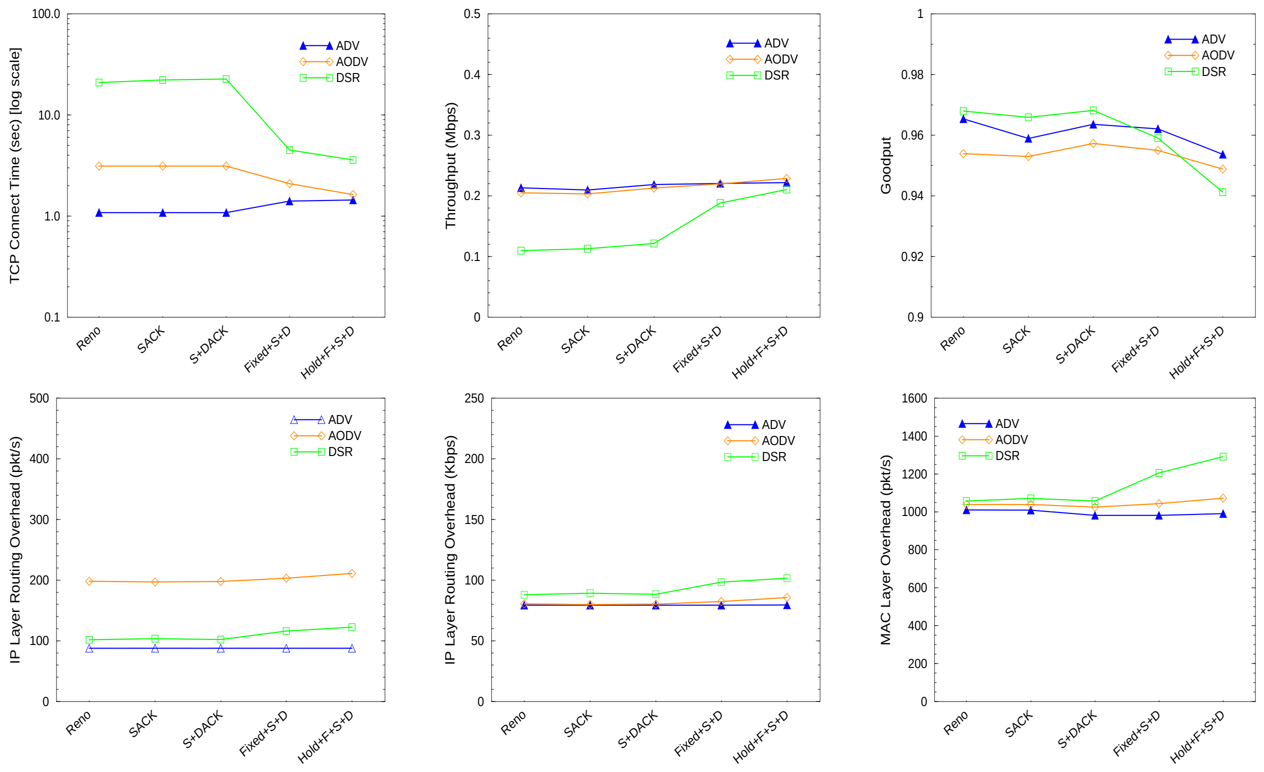


Figure B.3: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 200 Kbps background load from 10 CBR connections.

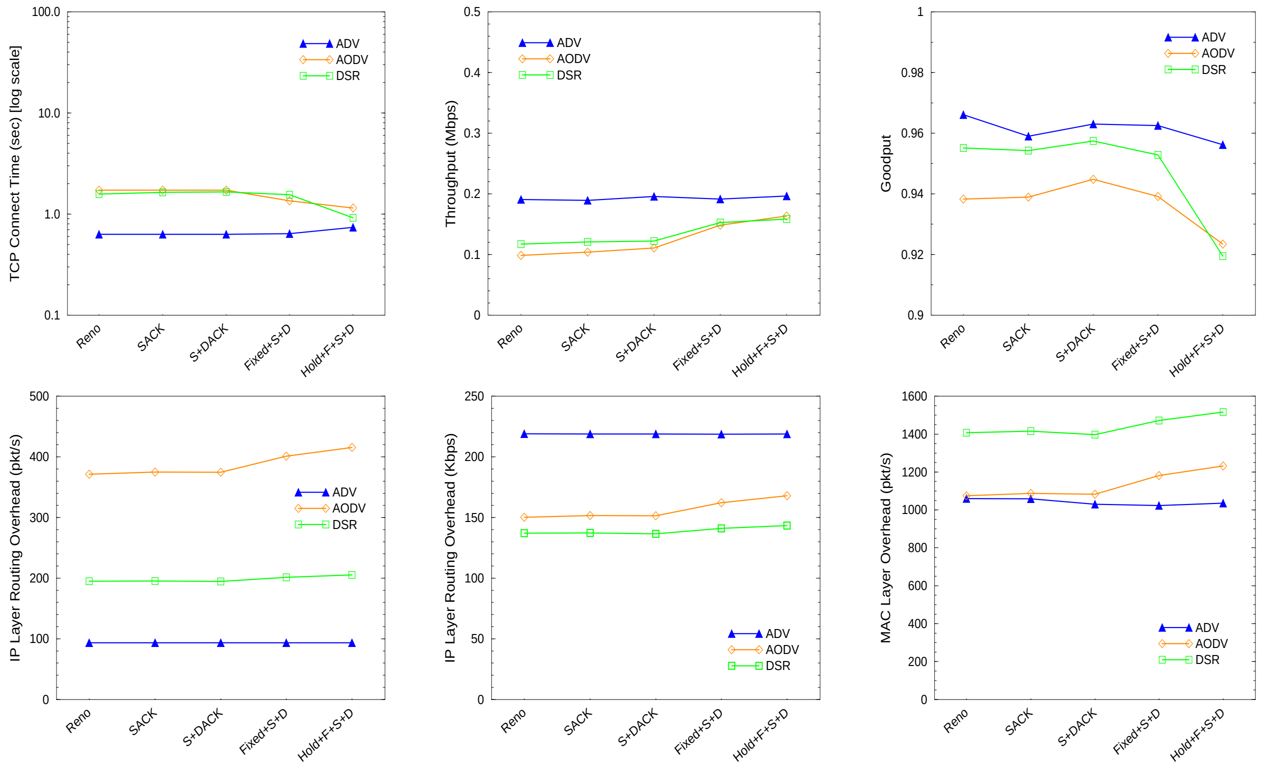


Figure B.4: Connect times, throughputs, goodputs, and routing overhead for 1 TCP connection with a 200 Kbps background load from 40 CBR connections.

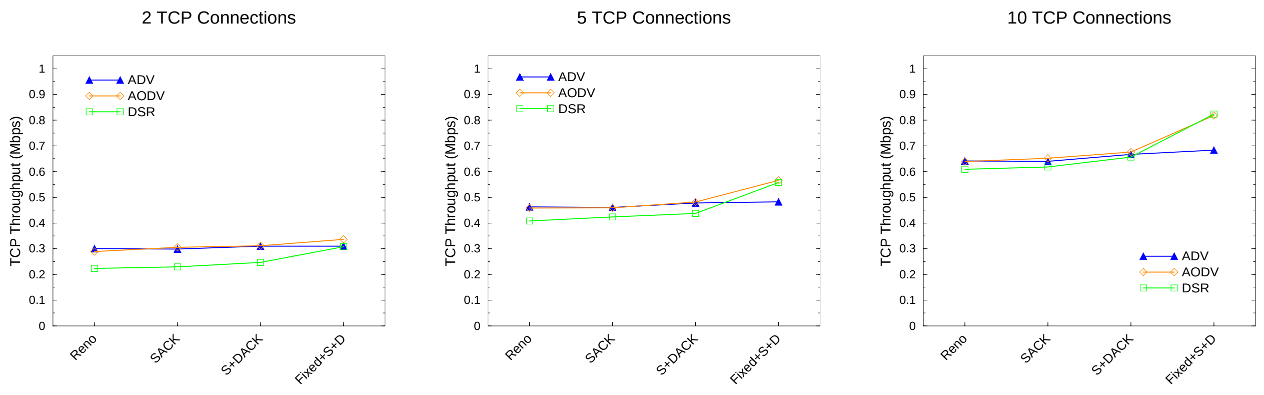


Figure B.5: Combined throughputs for multiple TCP connections with a 200 Kbps, 10-CBR background.

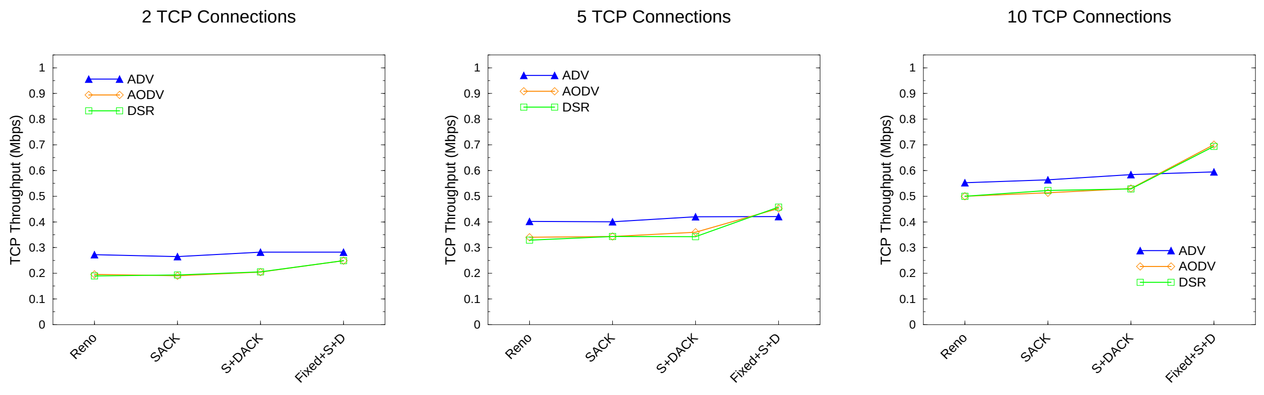


Figure B.6: Combined throughputs for multiple TCP connections with a 200 Kbps, 40-CBR background.

B.2 Performance results for multiple TCP connections

In this section, we present additional performance results for multiple TCP connections in a 50-node network moving in a 1000m x 1000m field. Figures B.5 and B.6 show the TCP throughputs achieved by each routing protocol for each combination of performance-improvement techniques. A background traffic load of 200 Kbps was included in those simulations. Figures B.7 - B.10 present the results obtained for TCP Reno and Reno-F with a 100 Kbps background load from 40 CBR connections.

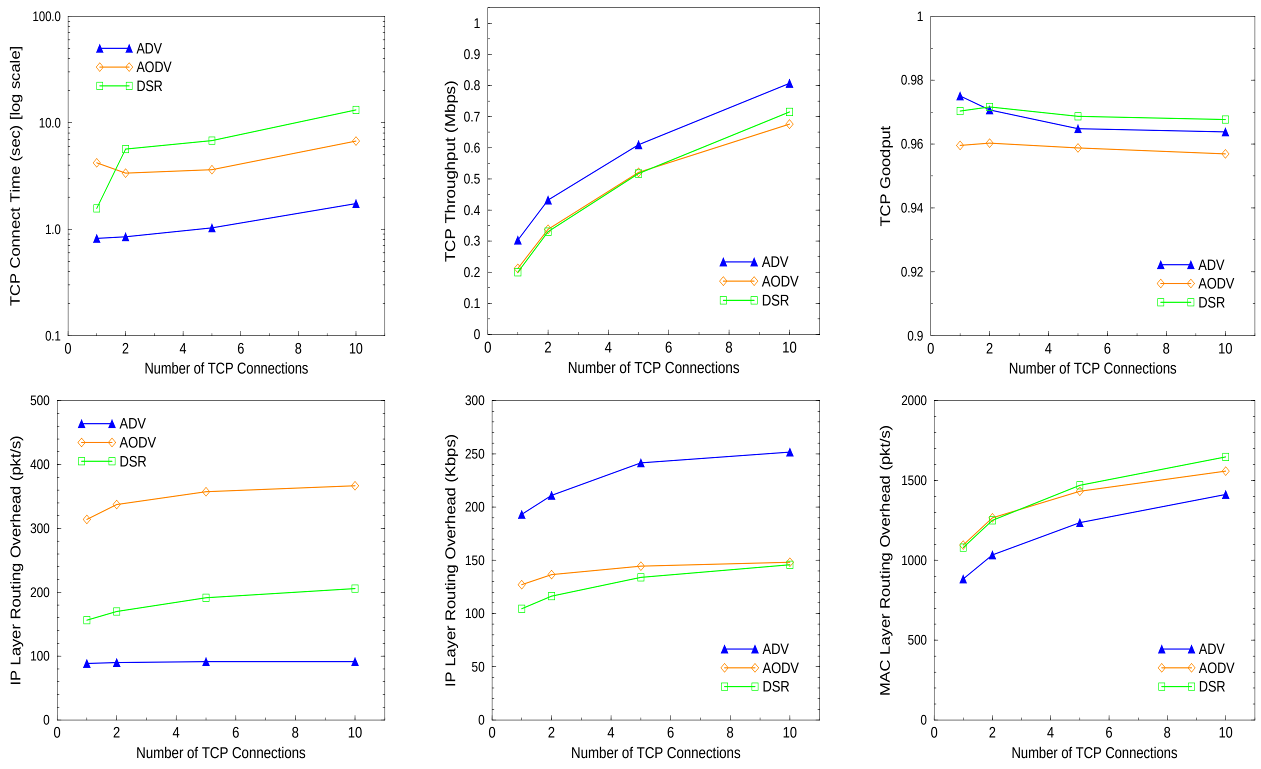


Figure B.7: Connect times, throughputs, goodputs, and routing overhead for TCP Reno with a 100 Kbps 40-CBR background.

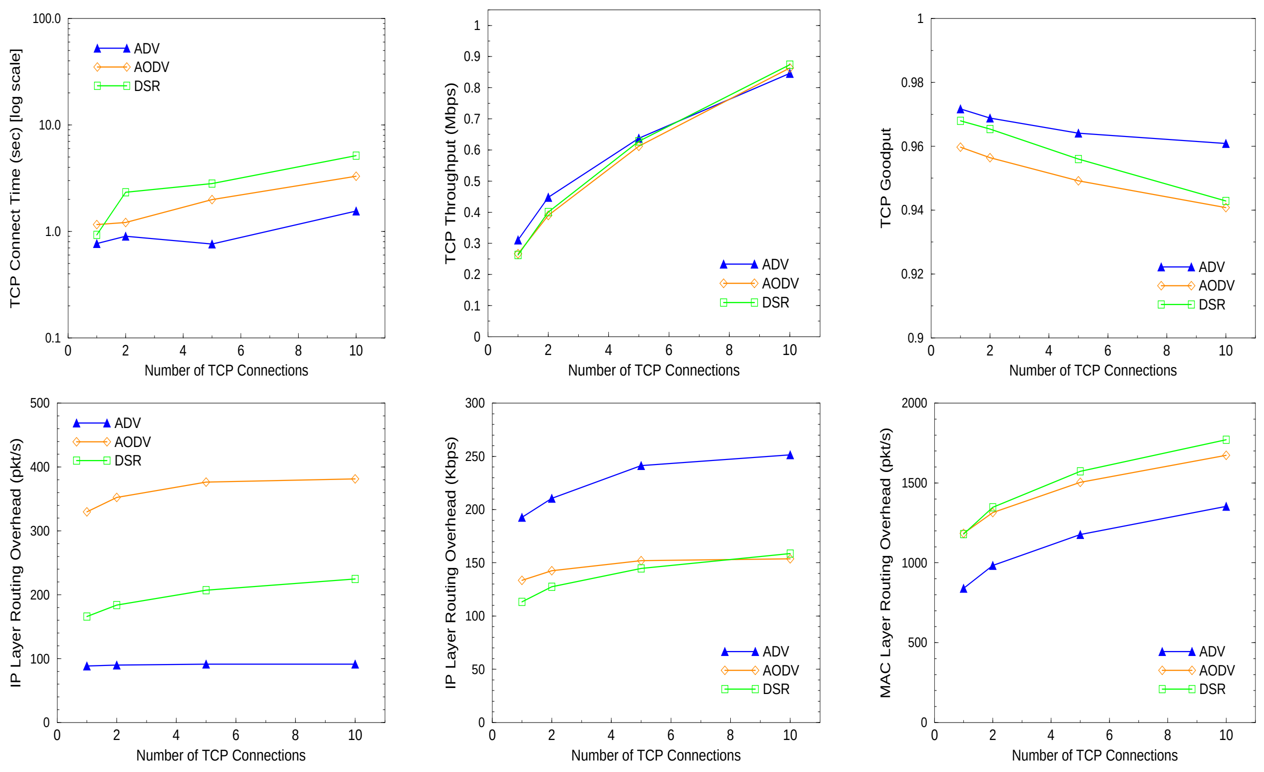


Figure B.8: Connect times, throughputs, goodputs, and routing overhead for TCP Reno-F with a 100 Kbps 40-CBR background.

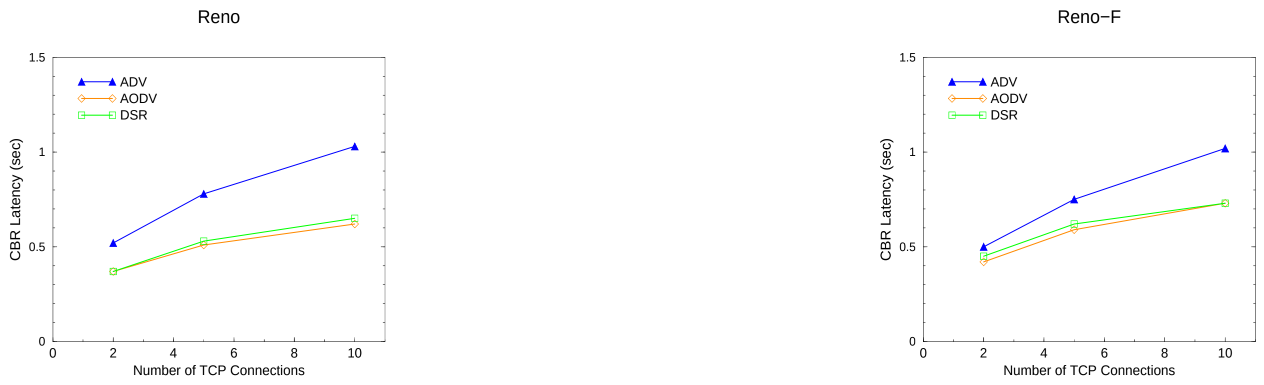


Figure B.9: CBR packet latencies for TCP Reno and Reno-F with a 100 Kbps background load from 40 CBR connections

.

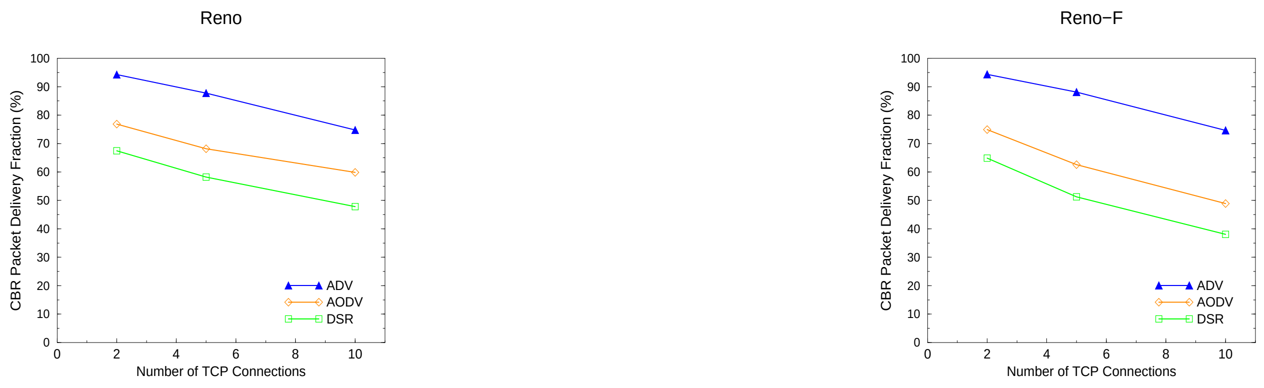


Figure B.10: CBR packet delivery fractions for TCP Reno and Reno-F with a 100 Kbps background load from 40 CBR connections.

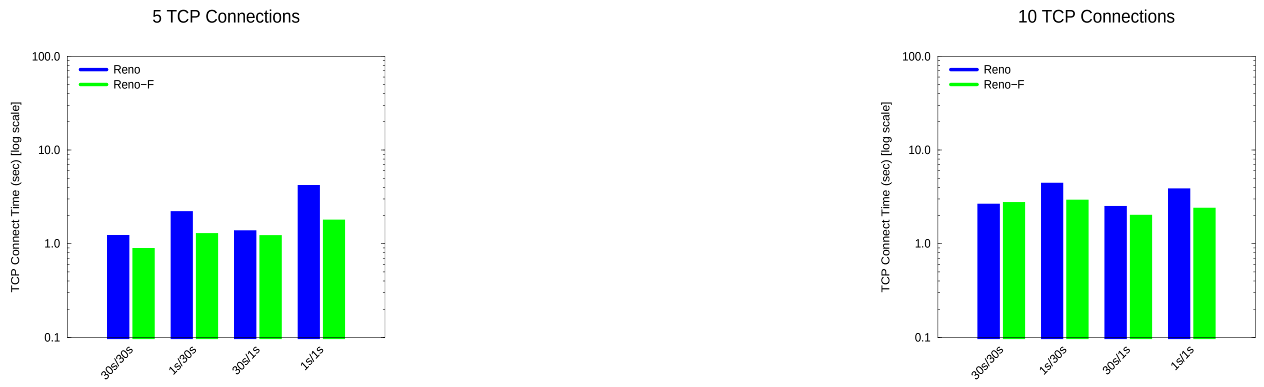


Figure B.11: Connect times for ADV for 5 and 10 TCP connections with a 200 Kbps background load from 10 CBR sources.

B.3 Effect of buffer refresh time on TCP performance

In this section, we present additional buffer refresh time results for multiple TCP connections in a 50-node network moving in a 1000m x 1000m field. These simulations included a 200 Kbps background traffic load from 10 CBR sources.

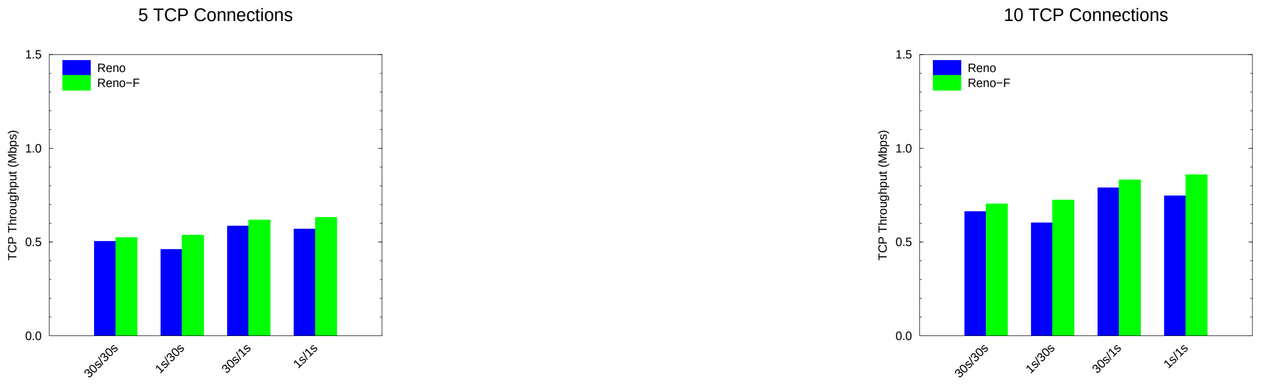


Figure B.12: Throughput for ADV for 5 and 10 TCP connections with a 200 Kbps background load from 10 CBR sources.

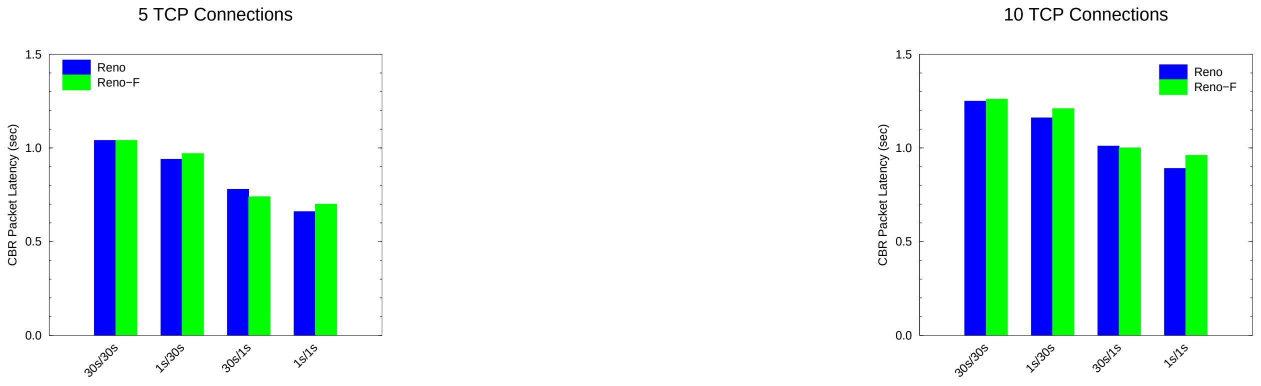


Figure B.13: CBR packet latency for ADV for 5 and 10 TCP connections with a 200 Kbps background load from 10 CBR sources.

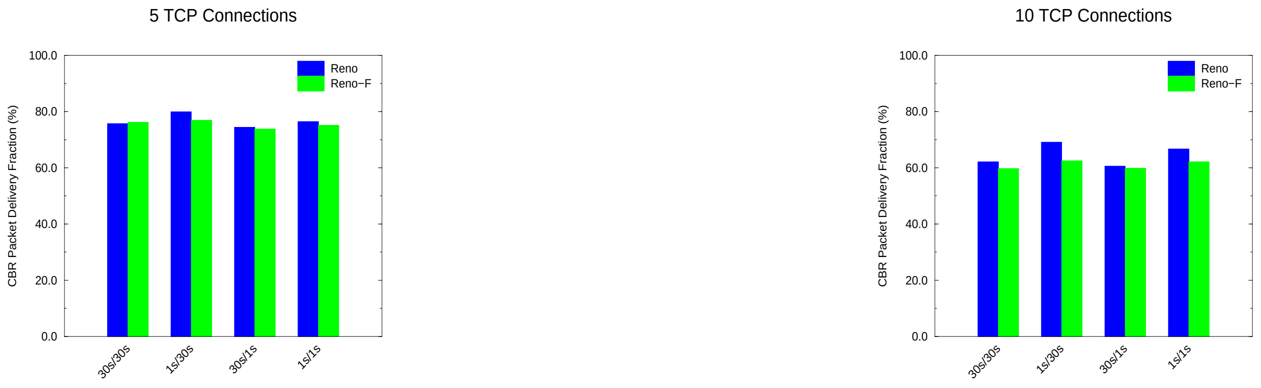


Figure B.14: CBR packet delivery fraction for ADV for 5 and 10 TCP connections with a 200 Kbps background load from 10 CBR sources.

Bibliography

[1] N. Abramson. The ALOHA system – another alternative for computer communication. In *Fall Joint Computer Conference, AFIPS Conference Proceedings*, volume 37, pages 281–285, 1970.

[2] A. Ahuja, S. Agarwal, J. P. Singh, and R. Shorey. Performance of TCP over different routing protocols in mobile ad-hoc networks. *Proc. IEEE Vehicular Technology Conference (VTC 2000)*, May 2000.

[3] C. Alaettinoglu, A. U. Shankar, K. Dussa-Zieger, and I. Matta. Design and implementation of MaRS: A routing testbed. *Journal of Internetworking: Research and Experience*, 5(1):17–41, 1994.

[4] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. AIRMAIL: A link-layer protocol for wireless networks. *Wireless Networks*, 1(1):47–60, Feb. 1995.

[5] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. PARSEC: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, Oct. 1998.

[6] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, pages 136–143, May 1995.

[7] B. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan. Improving performance of TCP over wireless networks. In *Proc. 17th International Conf. on Distributed Computing Systems (ICDCS)*, pages 365–373, May 1997.

[8] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *Wireless Networks*, 1(4):469–481, Dec. 1995.

[9] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Trans. on Networking*, 5(6):756–769, Dec. 1997.

[10] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[11] S. Biaz and N. H. Vaidya. Sender-based heuristics for distinguishing congestion losses from wireless transmission losses. Technical Report 98-013, Dept. of Computer Science, Texas A&M University, June 1998.

[12] R. V. Boppana and S. P. Konduru. An adaptive distance vector routing algorithm for mobile, ad hoc networks. In *Proc. 20th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*, volume 3, pages 1753–1762, Mar. 2001.

[13] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. 4th Annual ACM/IEEE International Conf. on Mobile Computing and Networking (ACM MobiCom ’98)*, pages 85–97, Oct. 1998.

[14] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM SIGCOMM Computer Communication Review*, 27(5):19–43, Oct. 1997.

[15] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[16] S. Chakrabarti and A. Mishra. QoS issues in ad hoc wireless networks. *IEEE Communications Magazine*, 39(2):142–148, Feb. 2001.

[17] K. Chandran, S. Raghunathan, S. Venkatesan and R. Prakash. A feedback based scheme for improving TCP performance in ad-hoc wireless networks. In *Proc. 18th International Conf. on Distributed Computing Systems (ICDCS)*, pages 472–479, May 1998.

[18] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot. Optimized link state routing protocol. IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-06.txt, Sep. 2001.

[19] S. R. Das, R. Castaneda, J. Yan, and R. Sengupta. Comparative performance evaluation of routing protocols for mobile, ad hoc networks. In *Seventh International Conf. on Computer Communication and Networks (IC3N)*, pages 153–161, Oct. 1998.

[20] S. R. Das, C. E. Perkins, and E. M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proc. 19th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2000)*, volume 1, pages 3–12, Mar. 2000.

[21] R. Dube, C. D. Rais, K. Wang, and S. K. Tripathi. Signal stability based adaptive routing (SSA) for ad hoc mobile networks. In *IEEE Personal Communications*, 4(1):36–45, Feb. 1997.

[22] K. Fall and K. Varadhan. ns Manual. The VINT Project. UC Berkeley, LBL, USC/ISI, and Xerox PARC. Available from http://www.isi.edu/nsnam/ns/ns-documentation.html, Apr. 2002.

[23] S. Floyd. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review*, 24(5):8–23, Oct. 1994.

[24] M. Gerla, K. Tang, and R. Bagrodia. TCP performance in wireless multi-hop networks. In *Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 41–50, 1999.

[25] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proc. 19th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2000)*, volume 3, pages 1537–1545, Mar. 2000.

[26] Z. J. Haas, M. R. Pearlman, and P. Samar. The zone routing protocol (ZRP) for ad hoc networks. IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-zone-zrp-04.txt, July 2002.

[27] C. Hedrick. Routing information protocol. RFC 1058, June 1988.

[28] T. R. Henderson, E. Sahouria, S. McCanne, and R. H. Katz. On improving the fairness of TCP congestion avoidance. In *Proc. Global Telecommunications Conference (GLOBECOM 1998)*, volume 1, pages 539–544, 1998.

[29] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proc. 5th Annual ACM/IEEE International Conf. on Mobile Computing and Networking (ACM MobiCom '99)*, pages 219–230, Aug. 1999.

[30] Y.-C Hu and D. B. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *Proc. 6th Annual ACM/IEEE International Conf. on Mobile Computing and Networking (ACM MobiCom '00)*, pages 231–242, Aug. 2000.

[31] IEEE Computer Society CSMA/CD (Ethernet) Standards Committee. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer (PHY) specification. IEEE Standard 802.3, 1993.

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

[33] IETF MANET Working Group Charter. http://www.ietf.org/html.charters/manet-charter.html.

[34] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks. In *Proc. 5th Annual ACM/IEEE International Conf. on Mobile Computing and Networking (ACM MobiCom '99)*, pages 195–206, Aug. 1999.

[35] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, edited by T. Imielinski and H. Korth, chapter 5, pages 158–163, Kluwer-Academic Publishers, 1996.

[36] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks (DSR). IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt, Feb. 2002.

[37] J. Jubin and J. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, Jan. 1987.

[38] A. Kapadia, A. Feng, and W.-C. Feng. The effects of inter-packet spacing on the delivery of multimedia content. In *Proc. 21st International Conf. on Distributed Computing Systems (ICDCS)*, pages 665–672, Apr. 2001.

[39] S. Keshav and S. P. Morgan. SMART retransmission: performance with overload and random losses. In *Proc. 16th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '97)*, volume 3, pages 1131–1138, 1997.

[40] D. Kim, C.-K. Toh, Y. Choi. TCP-BuS: Improving TCP performance in wireless ad hoc networks. *Journal of Communications and Networks*, 3(2):175–186, June 2001.

[41] M. Krunz and H. Hughes. A traffic model for MPEG-coded VBR streams. In *ACM SIGMETRICS Performance Evaluation Review*, 23(1):47–55, May 1995.

[42] S.-B. Lee, G.-S. Ahn, and A. Campbell. Improving UDP and TCP performance in mobile ad hoc networks with INSIGNIA. *IEEE Communications Magazine*, 39(6):156–165, June 2001.

[43] S.-J. Lee, M. Gerla, and C.-K. Toh. A simulation study of table-driven and on-demand routing protocols for mobile ad hoc networks. *IEEE Network*, 13(4):48–54, Aug. 1999.

[44] J. Li, C. Blake, D. S. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *Proc. 7th Annual ACM/IEEE International Conf. on Mobile Computing and Networking (ACM MobiCom '01)*, pages, 61–60, July 2001.

[45] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1300–1315, July 2001.

[46] J. P. Macker, V. D. Park, and M. S. Corson. Mobile and wireless internet services: Putting the pieces together. *IEEE Communications Magazine*, 39(6):148–155, June 2001.

[47] D. A. Maltz, J. Broch, and D. B. Johnson. Experiences designing and building a multi-hop wireless ad hoc network testbed. Technical Report CMU-CS-99-116, School of Computer Science, Carnegie-Mellon University, Mar. 1999.

[48] M. K. Marina and S. R. Das. Performance of route caching strategies in dynamic source routing. In *Proc. of the International Workshop on Wireless Networks and Mobile Computing (WNMC) in conjunction with the International Conf. on Distributed Computing Systems (ICDCS)*, pages 425–432, Apr. 2001.

[49] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, Oct. 1996.

[50] J. Moy. OSPF Version 2. RFC 1583, Mar. 1994.

[51] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. 16th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '97)*, volume 3, pages 1405–1413, 1997.

[52] C. E. Perkins, ed., *Ad Hoc Networking*. Addison-Wesley, Reading, MA, 2001.

[53] C. E. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (AODV) routing. IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-11.txt, June 2002.

[54] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance vector (DSDV) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, Oct. 1994.

[55] J. Postel. User Datagram Protocol. RFC 768, Aug. 1980.

[56] J. Postel. Internet Protocol. RFC 791, Sep. 1981.

[57] J. Postel. Transmission Control Protocol. RFC 793, Sep. 1981.

[58] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[59] T. S. Rappaport, S. Y. Seidel, and K. Takamizawa. Statistical channel impulse response models for factory and open plan building radio communications system design. *IEEE Transactions on Communications*, 39(5):794–807, May 1991.

[60] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, Mar. 1995.

[61] Rice University Monarch Project. Wireless and mobility extensions to ns-2. Available from http://www.monarch.cs.rice.edu/cmu-ns.html, Oct. 1999.

[62] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, Jan 1996.

[63] A. U. Shankar, C. Alaettinoglu, K. Dussa-Zieger, and I. Matta. Transient and steady-state performance of routing protocols: Distance-vector versus link-state. *Journal of Internetworking: Research and Experience*, 6:59–87, 1995.

[64] R. Sivakumar, P. Sinha, and V. Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications*, 17(8):1454–1465, Aug. 1999.

[65] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, MA, 1994.

[66] C.-K. Toh. Long-lived ad hoc routing based on the concept of associativity. IETF Internet Draft. http://www.ietf.org/proceeding/99nov/I-D/draft-ietf-manet-longlived-adhoc-routing-00.txt, Mar. 1999.

[67] UCLA Parallel Computing Laboratory. Global Mobile Information Systems Simulation Library (GloMoSim). Available from http://pcl.cs.ucla.edu/projects/glomosim.

[68] N. Vaidya, M. Mehta, C. Perkins, and G. Montenegro. Delayed duplicate acknowledgements: a TCP-unaware approach to improve performance of TCP over wireless. Technical Report 99-003, Dept. of Computer Science, Texas A&M University, Feb. 1999.

[69] G. T. Wong, M. A. Hiltunen, and R. D. Schlichting. A configurable and extensible transport protocol. In *Proc. 20th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*, volume 1, pages 319–328, Mar. 2001.

[70] R. Yavatkar and N. Bhagwat. Improving end-to-end performance of TCP over mobile internetworks. In *Proc. Workshop on Mobile Computing Systems and Applications*, pages 146–152, Dec. 1994.

Vita

Thomas Dyer was born in Texas City, Texas on February 5, 1953, the son of Wilson L. Dyer, Jr. and Jane H. Dyer. After graduating from Texas City High School in January 1971, he attended the University of Texas at Austin from 1971 to 1973. He completed his undergraduate education at the University of Houston at Clear Lake, where he obtained a Bachelor of Science degree in Mathematical Sciences in December 1977. He later pursued graduate studies at the University of Texas at San Antonio and earned a Master of Science degree in Computer Science at UTSA in May 1987.

Tom and his wife, Susie, reside in Leon Valley, Texas. They have four children and, at last count, four grandchildren. He has been employed since 1988 as a Senior Systems Analyst in the Department of Genetics at the Southwest Foundation for Biomedical Research in San Antonio, Texas.

Publications in Computer Science

T. D. Dyer and R. V. Boppana. Analysis of TCP and UDP traffic in MANETs. In *Proc. 3rd IEEE Workshop on Wireless LANs (WLAN '01)*, Sep. 2001.

T. D. Dyer and R. V. Boppana. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. In *Proc. 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, pages 56–66, Oct. 2001.

T. D. Dyer and R. V. Boppana. Assessing the impact of buffer refresh time and local route repair on routing protocol performance in mobile ad hoc networks. In *Proc. 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, volume XV, pages 219–224, July 2002.

T. D. Dyer and R. V. Boppana. Routing HTTP traffic in a mobile ad hoc network. To appear in *Military Communications Conference (MILCOM 2002)*, Oct. 2002.

T. D. Dyer and R. V. Boppana. On routing Web and multimedia traffic in mobile ad hoc networks. To appear in *Hawaii International Conference on System Sciences (HICSS '03)*, Jan. 2003.