This is to certify that the

thesis entitled

ADAPTIVE RELIABLE MULTICAST
IN WIRELESS LOCAL AREA NETWORKS

presented by

Chiping Tang

has been accepted towards fulfillment
of the requirements for

~~M.S.~~ degree in ~~Computer Science~~

Major professor

Date 5/8/02

ADAPTIVE RELIABLE MULTICAST IN WIRELESS LOCAL AREA NETWORKS

By

Chiping Tang

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science and Engineering

2002

ABSTRACT

ADAPTIVE RELIABLE MULTICAST
IN WIRELESS LOCAL AREA NETWORKS

By

Chiping Tang

Wireless local area networks exhibit substantially different characteristics from their wired counterparts: limited bandwidth, bursty packet loss, and receiver-specific loss patterns. In these environments, a network protocol should constantly adapt its algorithms and parameters in response to the changing channel conditions.

In this research, we propose and implement a reliable multicast protocol that adaptively adjusts parameters in response to channel condition changes. The relevant parameters include the proactive FEC rate, the global NACK suppression flag, and the sending rate. The protocol derives channel condition factors, such as packet loss rate and contention degree, based on feedback from receivers. The protocol is implemented at the application level and it requires no special link layer support. Analysis and simulation results show that the protocol has good overall performance in most cases, in terms of transmission throughput. The factors that degrade the performance are pointed out and discussed.

Our experience of wireless network simulation shows that an accurate channel loss model is critical to protocol performance evaluation. Based on the collected packet traces, we developed a novel channel model that takes into account the correlation of losses among multiple receivers. The model is used for evaluation of our protocol. It could be applied to other studies that involve wireless group communications.

To my wife Qi, and my son Colin

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Philip K. McKinley, for his great support, professional advice, and profound understanding. Without these, I could not have completed this work. I also thank Professors Abdol-Hossein Esfahanian and Sandeep Kulkarni for serving in my thesis committee.

Many other people helped me during the process of finishing this work. I am grateful to Arun P. Mani, Udiyan Padmanabhan, Peng Ge, Seyed Masoud Sadjadi, and other faculty members and graduate students in Software Engineering and Network Systems Laboratory. I also owe a lot of gratitude to my wife Qi, for her selfless support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Multicasting is an efficient method to disseminate information to multiple recipients. In this approach, the sender distributes a packet simultaneously to multiple receivers. Compared to multiple unicasting, in which the sender transmits a separate copy of the packet to each receiver, multicasting typically requires less bandwidth and exhibits lower latency. Due to these advantages, many applications use multicasting for mass information distribution. Examples include multimedia conferencing, video and audio distribution and so forth. In the global Internet, the MBone has been deployed to support large scale multicast multimedia applications [36]. The increasing use of multicast has inspired many research groups to explore efficient approaches to multicast under various conditions and in different types of networks.

Many multicast applications require reliable data distribution. If any data packet is lost or corrupted during the transmission, it should be recovered by either retransmission or other approaches. Example applications include file distribution, distributed simulation and collaborative computing. A major challenge is to find an effective and efficient way to perform error recovery. To reduce congestion-related packet losses, flow control is

also a concern. A large amount of research has addressed this problem in wired networks [3, 9, 11, 17, 19, 21, 23, 30, 32, 38, 42, 49, 53, 60, 61, 66, 67, 69, 74].

In recent years, wireless communication technology has been advancing so rapidly that it has altered our view of computer networks. Combined with advances in portable computing devices, a computer network is no longer a static entity. Network installation has become simpler and faster, since cabling is often not necessary. The network topology may be ad hoc and rapidly changing. Moreover, the network itself may exist only on an as-needed basis. Let us consider an example from a single domain: education. Students may bring their laptop computers to a classroom, or any other meeting location, and form a temporary network instantly. The instructor can introduce her/his laptop to this network and distribute electronic course material. After the session, the participants depart and the network disappears. This flexibility greatly improves the impact of distributed applications. Reliable multicasting is an important component of many such applications, including wireless instruction, collaborative conferencing, and field demonstrations.

However, wireless channels exhibit very different characteristics than their wired counterparts: They usually have limited bandwidth and relatively high and dynamic packet loss rates. These characteristics impose new challenges to communication protocols, and reliable multicast is no exception. A reliable multicast protocol must take into account the loss characteristics of the wireless network. Moreover, since the conditions are dynamic, the protocol should also be adaptive. In this work, we explore efficient algorithms for reliable multicast in one of the most important types of wireless networks: wireless local area networks (WLANs).

1.2 Challenges

In this section, we briefly discuss the technical issues that must be addressed to support reliable multicast in WLANs. A more detailed discussion is included in Chapter 2.

In reliable multicast, data packets are to be delivered without error to all receivers. If a packet is lost or corrupted, and cannot be recovered locally at the receiver, then the sender or another node must retransmit the packet. Therefore, the first challenge is error *detection*. The sender must determine whether or not all receivers have successfully received a packet. Hence, receivers must send feedback to the sender in the form of either positive acknowledgements (ACKs) or negative acknowledgements (NACKs). The overhead of feedback processing could slow down the sender, especially in multicast group with large number of receivers. This is called the ACK (or NACK) implosion problem [15]. Moreover, if uplink and downlink traffic share the same transmission medium, which is true in most local area networks, then this feedback will reduce the available bandwidth for data traffic. To increase scalability, the number of feedback packets should be kept as small as possible.

The second challenge is error *recovery*. After a packet loss is detected, the sender or another node retransmits the packet. A simple retransmission method is to maintain a list of the receivers experiencing the loss and send the lost packet to each of them. However, in the case where many receivers require retransmission, the resulting overhead (at the sender and on the channel) could be overwhelming. Since this retransmission traffic is pure overhead with respect to performance, it should be minimized.

In WLANs, the limited bandwidth, high packet loss rates, and shared transmission medium exacerbate the error detection and error recovery problems. First, the bandwidth of a WLAN is typically an order of magnitude lower than that of its wired counterpart. Combined with the fact that the transmission medium is shared, the feedback and retransmission traffic must be kept even lower to guarantee acceptable data bandwidth. Using NACK-based approaches is one way reduce feedback traffic [66]. However, in WLANs, data packets are propagated in open air, therefore they are vulnerable to various kind of interference, such as radio signals and other noise. The interference cause bit errors in the packet. On the other hand, if a receiver is located far away from the sender, then the received data signals might be too weak and data bits become indiscernible. Both of these factors will produce a CRC error. The link-level network protocol will drop all such packets. As a result, the packet loss rate is high, bursty and location dependent. The high loss rates lead to increased feedback and retransmission traffic.

A number of feedback suppression approaches have been proposed to reduce the feedback traffic. For example, in a *local* NACK suppression scheme, a receiver waits for a random time before sending its NACK, expecting that NACKs from other receivers have already triggered a retransmission. In the case the retransmitted packet arrives before the timeout, the pending NACK is canceled. A *global* NACK suppression scheme [9] can further improve this effect by having receivers send NACKs by way of multicast transmission. A receiver that overhears a NACK that subsumes its own pending NACK can cancel or delay its NACK. These NACK suppression techniques could be adapted to deal with the increased demand for lower feedback and the simultaneously increased possibility of higher feedback in WLANs.

Besides reducing feedback, it is also possible to reduce the overhead of retransmissions. If multiple receivers request the same packet, the sender may choose to multicast a retransmission, instead of unicasting it. In addition, the sender could take advantage of a forward error correction (FEC) encoding algorithm [2, 29, 39, 43, 44, 48, 50, 51, 55, 56, 57, 58, 59, 70]. Specifically, such code can be used to generate parity packets for a given set of data packets. Sending parity packets instead of the original data packets, in response to NACKs, carries the advantage that parity packets can recover different losses at multiple receivers. Using FEC in this manner is called *reactive* FEC. Of course, an FEC protocol can also be used for feedback suppression. In this case, the sender proactively sends some FEC parity packets along with data packets. If the proactive parity packets are sufficient to recover lost data packets, then a receiver does not need to send a NACK. This approach is called *proactive* FEC.

In summary, an efficient reliable multicast protocol on a WLAN must address "traditional" issues, such as feedback suppression and retransmission management. In addition, the protocol should take into account the characteristics of wireless channels, such as low bandwidth, highly variable and bursty losses, and a shared feedback channel. Moreover, the algorithm has to be adaptive so that it can perform well under dynamic conditions.

1.3 Contributions and Structure of Thesis

In this study we focus on the design, implementation and performance evaluation of an adaptive reliable multicast protocol. We use a reactive FEC algorithm to reduce the

number of repair packets sent in response to NACKs. We also apply a proactive FEC algorithm in order to suppress feedback traffic. However, sending too many proactive packets wastes channel bandwidth and reduces throughput. This tradeoff is addressed and explored. To further improve performance, both global and local NACK suppression algorithms are investigated and applied.

Thesis Statement: *Using a combination of proactive and reactive FEC algorithms, in conjunction with local and global NACK suppression, it is possible to develop an efficient and scalable reliable multicast protocol that works well over wireless local area networks.*

The major contributions of this work can be summarized as follows:

1. Analysis of reliable multicast problems over WLANs. In WLANs, channel bandwidth is limited and shared, and packet loss is relatively high and bursty. Conventional techniques for error recovery and flow control do not work well in this environment. We have collected a large number of packet traces from our wireless testbed. From these traces, we induce packet loss properties of wireless channels and the impact of the properties is studied. Several approaches for performance improvement are proposed and compared.

2. Adaptive FEC-based Reliable Multicast (AFRM) protocol design and performance evaluation. AFRM is a reliable multicast protocol that combines proactive and reactive FEC to reduce NACK feedback and repair traffic. The proactive rate is dynamically determined based on the estimation of channel conditions. In the case of low packet loss, the protocol sends few or no proactive FEC parity packets in order to save channel bandwidth. When perceived packet loss turns higher, the sender increases the

proactive rate. The feedback traffic is reduced as a result, and the overall performance drops only slightly. Efforts are made to improve the accuracy and responsiveness of the adaptation. Global and local NACK suppression are applied when appropriate. A rate-based flow control mechanism is included to deal with congestion. We implemented the protocol at the application layer atop UDP/IP sockets and IP multicast. Experimental and simulation studies demonstrate that the protocol exhibits good performance over WLANs.

3. **Development of a general-purpose network simulation framework.** We designed and implemented a network simulation framework. In this framework, network applications built on the socket interface can be ported easily between real and simulated network environments. The application code requires little or no modification. The simulation framework offers simplified low-level network protocol components, including UDP, IP and IEEE 802.11 MAC protocols. Different packet loss patterns can be plugged in at receivers to simulate the wireless network loss. Besides reliable multicasting, this framework can be used for performance evaluation of many protocols in heterogeneous environments.

The remainder of this thesis is organized as follows. In Chapter 2, we present background information related to this study, including reliable multicast, FEC, and WLAN technologies. In Chapter 3, we describe the WLAN reliable multicast problem in detail and evaluate several component methods that might be used in such a protocol, using packet traces and analysis. In Chapter 4, we give a detailed description of AFRM protocol. The discussion includes the parameters and how they are determined, as well as

several implementation issues. In Chapter 5, we describe the experimental environment and simulation framework on which we conducted the performance studies. In Chapter 6, we present related work on reliable multicast, FEC, and WLAN studies. Finally, in Chapter 7 we summarize our work, discussing possible shortcomings and possible directions for future work.

Chapter 2

Background

Three major research areas are directly related to this study: wireless networks, reliable multicasting and forward error correction. In this chapter we present background information on each of those areas.

2.1 Wireless Networks

Wireless communication is perhaps the fastest growing area of communication technology in recent years. The combination of "anytime, anywhere" connectivity and the rapidly dropping prices are behind this popularity. In short, wireless communication is fundamentally changing our view of both telecommunications and computer networking.

The fundamental difference between wired and wireless communication is the way in which signals are propagated. In wired communication, signals are propagated through cables, while in wireless communication they are propagated in open air. As a result, signal losses and distortions are more significant and almost unpredictable in wireless environments, since signals in open air are vulnerable to interference from many sources. Overcoming dynamic, high error rates is a major concern of wireless applications.

Wireless networking is an integration of wireless communication and computer networking. Many wireless networks, especially WLANs, have been installed to provide communication connectivity for wireless computers. Since low-level network protocols usually hide the physical difference between wired and wireless channels, higher-level network protocols and applications usually can execute without modification atop a wireless network. However, due to the different physical channel characteristics, loss-sensitive applications often need to adapt their behavior in response to channel condition changes in order to maintain acceptable performance.

2.1.1 Wireless Loss Characteristics

Packet loss in wireless networks is variable and bursty. Although the loss rate may be high under poor conditions, the loss rate may also be extremely low under good conditions. Henceforth we define a loss as a packet that is not successfully received by a station for any reason. We use the term *burst* to denote a group of consecutive packets that are either all received or all lost. In the all-received case, the group is called a loss-free burst or error-free burst. In the all-lost case, it is called a loss burst or error burst. Figure 2.1 is reproduced from [33]. It depicts packet traces in a GSM network. In the figure, the loss and loss-free burst lengths vary from one to one hundred packets.

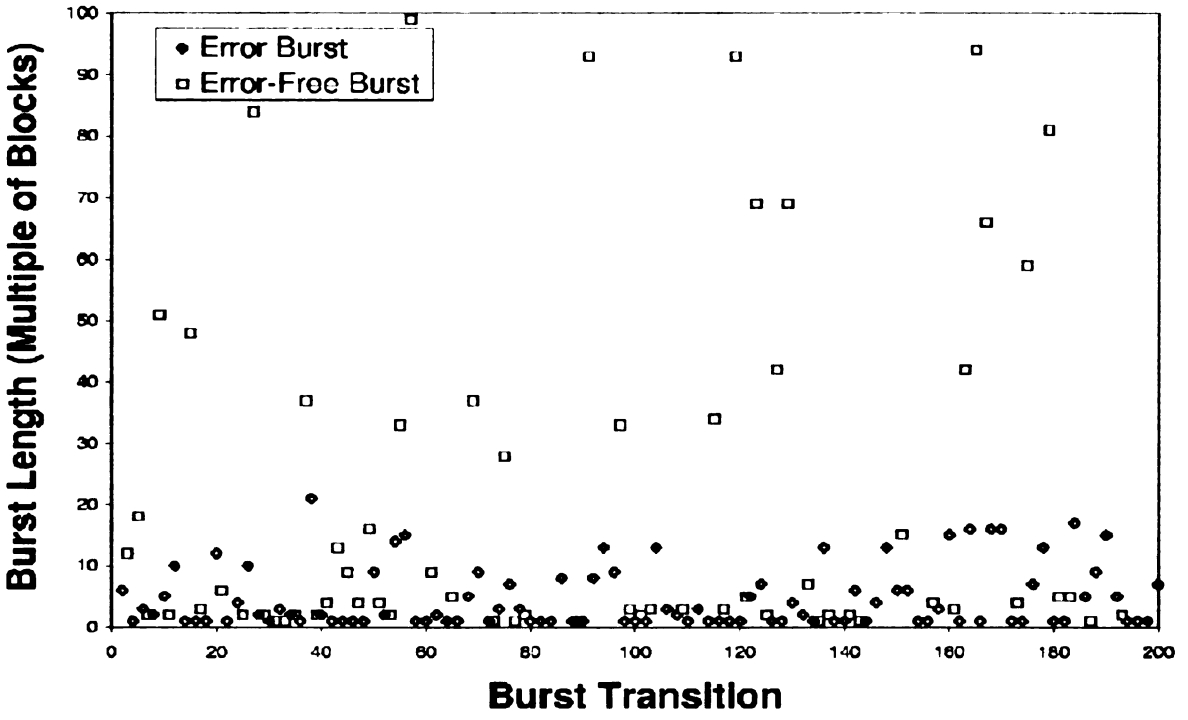


Figure 2.1: Wireless traces in GSM network [33].

The distance between a receiving station and the data source significantly affects the packet loss rate at the station. A wireless propagation error is defined as a loss that is caused by signal fading or shadowing during propagation in the air. Since the signal power continues to drop as the packet propagates in air, receivers that are farther away from the data source are more likely to experience propagation errors. The low level protocol will interpret any packet that contains indiscernible bits as a loss. Figure 2.2, reproduced from [46], illustrates the correlation between packet loss rate and distance. The data was collected in an IEEE 802.11 LAN with 2Mbps bandwidth.

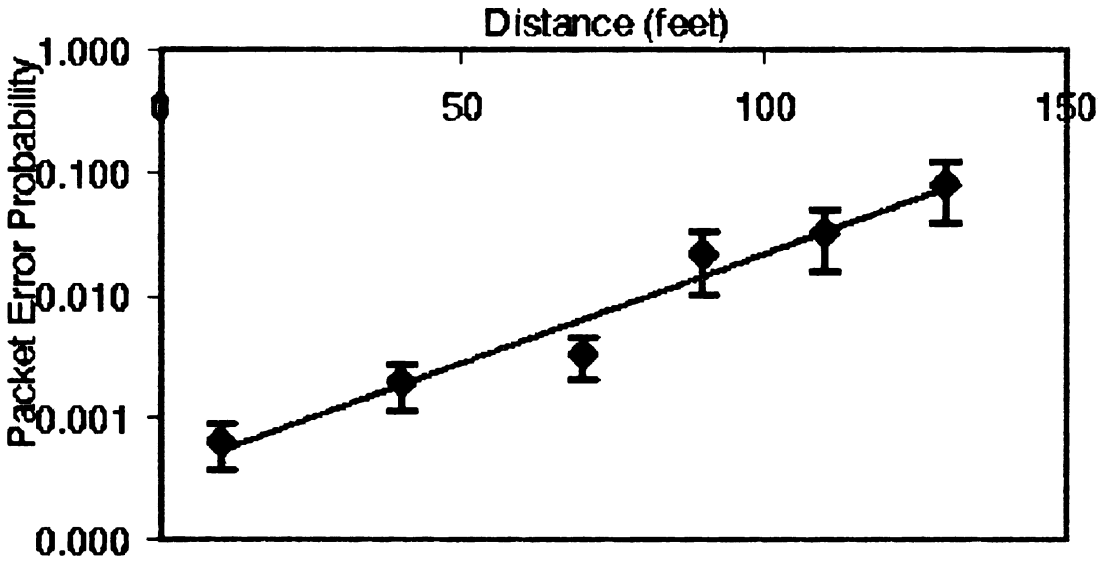


Figure 2.2: Wireless losses vs distance [46].

2.1.2 CSMA/CA

Unlike cellular networks with allocated channels (frequencies), stations in a WLAN share frequency and bandwidth. MAC protocols are needed in such environments to manage contention and collisions. While CSMA/CD [31] is widely used in wired LANs, many wireless MAC protocols are based on the CSMA/CA protocol [31]. The main difference between CSMA/CA and CSMA/CD is the way in which collisions are handled. In CSMA/CD, a station is required to detect immediately whether or not a transmitted frame has collided with another frame. If a collision occurs, the station can promptly schedule a retransmission. The CSMA/CD scheme is not feasible for WLANs, however, since a wireless station is unable to transmit and receive simultaneously on a single radio transceiver. Therefore it is hard for the station to detect if a frame is

successfully transmitted. Instead, a wireless transmitter depends on acknowledgements (ACKs) from other station(s) to determine if a retransmission is needed

The basic CSMA/CA protocol works as follows:

1. When a station is ready to transmit a data frame, it senses the channel.

2. If the channel is idle, the station transmits the frame immediately.

3. Otherwise, it invokes the backoff algorithm. In this algorithm, the station chooses a random number uniformly distributed between 0 and the contention window size. It uses this value as the number of slots it has to wait before next try.

3.1 The station listens on the channel while it is waiting. The slot is counted in only when there is no transmission detected in the slot.

3.2 When the number reaches zero, the station transmits the frame.

4. When a station receives a data frame, it returns an ACK frame to the transmitter after a short and fixed time interval. If the transmitter receives the ACK, the transmission is successfully completed.

5. Otherwise, the transmitter assumes the data frame has been lost and schedules a retransmission.

This basic protocol suffers from the *hidden terminal problem*. As shown in Figure 2.3, station A and station C cannot hear from each other since A is not within C's transmission range and vice versa. When A is transmitting a frame to station B, C cannot detect the signal. Therefore, C could also send a frame to B since it believes the channel is idle. As a consequence, a collision occurs. To solve this problem, an RTS/CTS extension [6, 28] is proposed. In this scheme, a station transmits a short RTS (Request-

To-Send) frame to receiver station before sending a data frame. If the receiver is ready to receive, it returns a CTS (Clear-To-Send) frame. After receiving the CTS, the transmitter starts sending the data frame. Other stations overhearing any of these RTS/CTS frames will wait until the transmission is completed. As a result, the collision rate drops significantly. Although RTS and CTS frame could also collide with others, the chance is much lower since the sizes of these frames are usually far smaller than those of data frames.

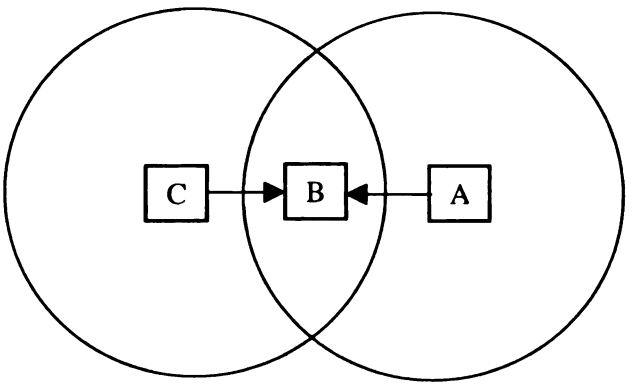


Figure 2.3: Hidden terminal problem.

When a data frame is small, however, the collision probability is unlikely to be higher than that of a RTS frame. In this case, the RTS/CTS control frames become pure overhead. To improve performance, the protocol sets a threshold value. Only those data frames whose size is above the threshold value will be preceded by an RTS/CTS exchanges.

Both the RTS/CTS and ACK/retransmission mechanisms apply only to *unicast* transmissions. In the case of multicast, since there are multiple recipient stations, both approaches become complicated. As a result, multicast packets will experience higher loss in WLANs than unicast packets. Although a number of MAC level protocols have

been proposed to address this problem [37, 63, 64, 65], commercial systems do not yet support them, and all appear to have shortcomings.

2.1.3 The IEEE 802.11 Standard

Presently, there exist two main WLAN standards: IEEE 802.11 [25] and HiperLAN/HiperLAN2 [34, 68]. We discuss only IEEE 802.11, since the wireless devices in the experimental environment used in this study are 802.11 compatible. The IEEE 802.11 standard defines protocols that are necessary to support wireless networking in a local area. Like other IEEE 802 standards, such as 802.3 and 802.5, the primary service of 802.11 is to deliver MSDUs (MAC Service Data Units) between peer LLCs (Logical Link Control). The functions of the 802.11 standard are implemented in network interface cards, their software drivers and wireless access points.

The 802.11 standard provides MAC and physical layer functions for wireless connectivity of fixed and mobile stations. It supports asynchronous and time-bounded delivery services, multicast services, network management services, registration and authentication services, etc. The protocol takes into account the significant differences between wireless and wired LANs in power management, bandwidth, security and addressing. The standard is extensive in its coverage. Only those parts that are directly relevant to this study are discussed here. Please refer to [25] for additional details of the IEEE 802.11 standard.

The 802.11 target environments include indoor buildings and outdoor areas. The standard supports two network topologies: ad hoc (IBSS) and infrastructure (ESS). In ad hoc mode, stations communicate with each other without setting an infrastructure. It is possible that some stations are outside the radio coverage area of a particular station. In

this case a transmitting station needs to dynamically find an intermediate station to relay a packet to the stations that are not directly reachable. In infrastructure mode, an access point is located between wireless stations and the wired network (also referred to as the Distributed System (DS)). Any communication between two wireless stations or between one wireless station and another station in DS is relayed at the access point. Compared with ad hoc mode, the infrastructure mode is a centralized scheme, with the access point running as a bridge between networks. Clearly, the access point is a potential bottleneck in this system. When multiple wireless stations transmit packets simultaneously to stations in the DS, all the packets are delivered to the access point as the first step and forwarded to the DS thereafter. The resource contention at the access point will likely slow down the transmission. Worse yet, in the case of multicast, the access point transmits all packets back to wireless stations while it is forwarding them to DS, even if there is no multicast group member in the wireless network. The presence of this inefficient multicast scheme in the standard is probably due to the fact that the access point does not maintain any multicast group membership information. Despite these disadvantages, we used infrastructure mode in both the experiments and the simulations in this study, since it provides connectivity between wireless stations and wired LAN stations. Furthermore, it is generally more robust than ad hoc mode, in which the movement or crash of a particular wireless station might disconnect other stations and partition the network.

The 802.11 Distributed Coordination Function (DCF) MAC protocol is a CSMA/CA protocol with minor modifications. In this protocol, a node wishing to transmit should always guarantee the channel has been idle for a time equal to DCF Inter-Frame Space

(DIFS) since the last transmission, or a time equal to Extended Inter-Frame Space (EIFS) since the last collision, whichever occurs last. The RTS/CTS exchanges are optional. The time interval between RTS and CTS, CTS and data packet, and between data packet and ACK, is equal to Short Inter-Frame Space (SIFS). The interaction is depicted in Figure 2.4. The protocol uses a binary exponential backoff algorithm for retransmissions. A retransmitting node first doubles the size of the contention window unless it has already reached a maximum value. It then invokes the backoff algorithm before retransmission. It aborts the transmission and discards the packet after several unsuccessful tries.

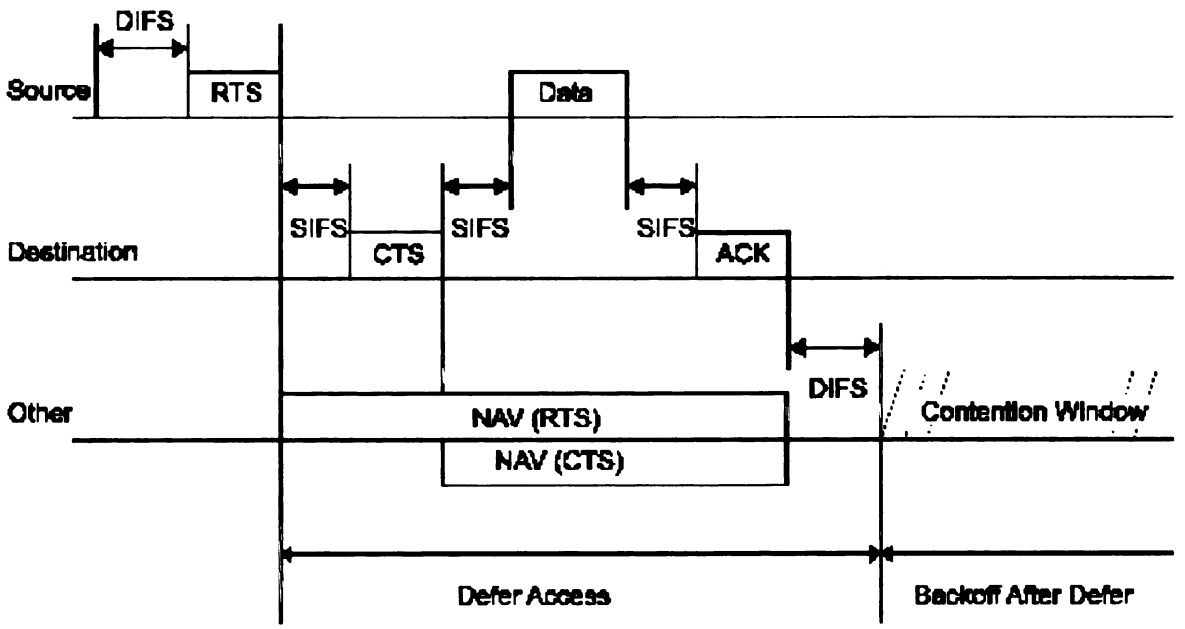


Figure 2.4: RTS/CTS exchange in IEEE 802.11 WLAN [25].

The original 802.11 standard supports two transmission rates: 1Mbps and 2Mbps. More recently, two extensions have been developed that support higher rates, up to 11Mbps in 802.11b and 54Mbps in 802.11a. The protocols differ in physical layer properties, such as signal modulation and operational frequency band. However they

have the same MAC layer protocols and functions. Our experimental devices are 802.11b compatible with 11Mbps maximum transmission rate. We also set our simulation framework to be 802.11b compatible.

2.2 Reliable Multicast

Some multicast applications tolerate a certain degree of data loss. For example, in video distribution, occasional loss or distortion of a video frame imposes little disturbance to human eyes. Hence, it may be unnecessary (even counterproductive) for receivers to attempt to recover the loss. However, in many other applications, any data loss or distortion is absolutely unacceptable. Considering distribution of executable files, the program might be unable to correctly execute, even if a single bit is reversed. An error recovery mechanism is needed in this case. Since applications have different requirements for when and how to do error recovery, reliability is usually provided in high-level protocols. Many protocols are built atop UDP/IP Multicast, which provides unreliable, or best effort, service.

2.2.1 Error Recovery

To guarantee reliability, two mechanisms are needed: error recovery and flow control. Error recovery deals with loss detection and repair packet transmission. Two basic schemes are used for loss detection: sender initiated or receiver initiated [66]. In the former case, the sender pauses after a data packet is transmitted, waiting for all receivers to send back positive acknowledgements (ACKs). This is also called an ACK-based

protocol. If the number of received ACK packets is less than the number of receivers, the sender concludes that the packet has been lost at some receivers and schedules a retransmission.

There are several problems in this approach. First, it is difficult for the sender to determine how long to wait before retransmitting data. Usually the timeout value is based on the round trip time (RTT) between the sender and the slowest receiver. If the RTT is dynamic, however, the timeout value could become too large or too small. If the sender waits too long, transmission is delayed. If the sender times out prematurely, then it may send a repair packet unnecessarily. Second, the sender needs to know the exact number of receivers and needs to maintain status information for each of them. It must handle every join and leave message to track explicit group membership information. In a large multicast group, this overhead could be overwhelming. Third, since every receiver returns one ACK packet for each data packet, the sender may spend a large amount of time processing these ACK packets. This situation is called the *ACK implosion problem* [15]. Furthermore, if the reverse traffic shares the communication channel with data traffic, the available data bandwidth will drop rapidly. In summary, an ACK-based protocol simply does not scale well.

In a receiver-initiated scheme, the sender has no responsibility for loss detection. Its only duty is to assign a unique sequence number to each data packet. Each receiver maintains the sequence number of the last packet received in correct order. When a new packet arrives, the receiver checks the sequence number and compares it with the last one. If there is a gap between these two numbers, the receiver assumes that the intermediate packets have been lost and sends a negative acknowledgement (NACK) to

the sender. Upon receiving a NACK packet, the sender retransmits the requested packets. This NACK-based approach has many advantages over an ACK-based scheme. First, only receivers that have lost packets send NACKs. In many network environments, especially in wired networks, the packet loss rate is fairly low. Therefore, the sender will not need to devote many resources to feedback processing, and more channel bandwidth is available for forward data transmission. Second, the sender need not maintain status information for receivers, but rather adopts a passive "come-and-serve" policy. If a NACK arrives, the requested packets are retransmitted. In the absence of NACKs, the sender assumes that all receivers have successfully received the packet, and no further processing is needed. In short, group membership information is not necessary for the sake of retransmission. Third, there is no need for per-data timeouts, since the sender is told exactly when to retransmit packets.

On the other hand, NACK based protocols have undesirable properties that must be addressed. First, if the last packet in a chain of packets is lost, then the receiver will be delayed in detecting the loss until the next packet arrives sometime in the future. To solve this problem, most NACK-based protocols send periodic "keepalive" or "heartbeat" packets when the sender is in idle [23, 69]. These packets include the sequence number of last data packet having been sent. Receivers detect packet loss by comparing the sequence number in the heartbeat packet with that of the last received data packet. However, if a receiver experiences such poor channel conditions that all packets are lost for a period of time, then neither the receiver nor the sender will be aware of the losses until conditions improve. Second, in this approach, a receiver might request the retransmission of data packets that were sent long before. Indeed, a purely NACK based

protocol requires an infinitely large send buffer to overcome this problem [54]. To solve this problem, many NACK-based protocols adopt a hybrid algorithm in which receivers send periodic ACKs. This approach is a tradeoff between communication efficiency and providing full reliability with finite buffers. Overall, the NACK-based approach is more scalable than the ACK-based approach, and it is adopted in most reliable multicast protocols [17, 23, 32, 69].

After a loss is detected, the sender (or possibly another station [17, 23]) should schedule a retransmission as soon as possible. The repair packets could be sent out either by unicast or multicast. Whether to use unicast or multicast is a tradeoff. The advantage of multicast is that in case other receivers have lost the same packets, one copy of the repair packets is enough to recover from losses at all those receivers. In this manner, the retransmission overhead is reduced. The disadvantage is that, the multicast overhead can be higher than that of unicast. In a local area network with a shared medium, a unicast transmission is in fact a broadcast in terms of channel bandwidth. In this case, unicast and multicast require the same channel capacity, although multicast requires more collective receiver processing because it is delivered to all the receivers (which may be only a subset of the nodes within range of the source). This is called the *exposure problem* [52]. In the case losses are highly correlated among receivers, it is likely that the gain in bandwidth reduction outweighs the overhead of exposure. Therefore, a multicast strategy should be adopted. In other cases a unicast retransmission approach is probably more efficient.

An enhancement to handle the exposure problem is to divide the receivers into multiple multicast groups. Each group has a unique multicast address. When a repair

packet is ready to be sent, the sender chooses a set of multicast groups as the destination, so that the gain in retransmission reduction is greater than the exposure overhead in each of these groups. Since the groups in which no, or very few, receivers require retransmission are excluded from the list, the exposure problem is reduced. This scheme is similar to subcast (subtree multicast [53]), in which packets can be delivered to a subset of receivers in a multicast group. It is more efficient than pure unicast or ordinary multicast. To optimize the algorithm, the receivers are grouped according to their loss tendencies. In the Internet, losses are location dependent. Therefore grouping receivers based on their physical locations or logical network sections is a reasonable approach.

Another enhancement called *local recovery* [17] aims to reduce the feedback-processing load at the sender by allowing stations other than the sender to send repair packets [17, 23]. In a hierarchical network, it is possible to choose a retransmission station that is closer to the requesting receiver than the sender. In this case, the loss recovery time can also be reduced. There are two classes of local recovery approaches: *structured* and *unstructured.* In a structured scheme, receivers are organized into a hierarchy so that each group of several receivers has a dedicated node that is responsible for recovering losses at those receivers. This method can be applied to both ACK-based and NACK-based protocols. The unstructured approach is only applicable to NACK-based protocols. In this case, receivers multicast NACK packets; any node overhearing the request and capable of supplying the missing data can send repair packets.

NACK-based local recovery is often coupled with *NACK suppression* [74]. Since a receiver can overhear NACK packets from other receivers, it has the choice of not sending its own NACK packet that request is subsumed by a NACK from another

receiver. In this case, the receiver sets a timer and waits the repair packets. If the repaired packets arrive before the timer expires, the lost data is recovered and one NACK transmission is saved. Otherwise the receiver goes ahead and sends the pending NACK. In this approach, both the NACKs and repair packets are transmitted by way of multicast. As mentioned above, while multicast is more efficient, it usually incurs additional overhead compared to unicast. This tradeoff should be addressed in either NACK-based local recovery or NACK suppression. In some cases, the subcast approach can be applied to solve this problem.

In summary, the main challenges related to error recovery are feedback implosion and feedback/retransmission exposure. Solutions include NACK suppression, subcast and local recovery. The latter two approaches often require receivers to be organized in hierarchy. In an inter-network, it is natural to group the receivers based on their physical proximity to one another. In a local area network with a shared medium, however, a receiver hierarchy makes little sense. Therefore, NACK suppression is the primary solution in this case.

2.2.2 Flow Control

The second major issue in reliable multicasting is flow control. A sender that transmits too fast might overrun the packet buffer at a receiver or at some intermediate node. Packets that arrive after buffer is full will be dropped and will need to be retransmitted. Flow control regulates the data transmission rate at the sender and avoids data loss caused by buffer overflow.

Flow control approaches in reliable multicast are based on their unicast cousins. As in unicast, multicast flow control methods are classified as window-based or rate-based. In window-based flow control, the sender maintains a send window, and only those packets that fall within the window are permitted to be sent. The sender advances the window when all packets in the window are successfully transmitted. To control transmission rate, the sender adjusts the size of the send window based on congestion conditions. The sender assumes that there is no congestion unless a NACK arrives or an ACK is missing. If feedback indicates packet loss, the sender assumes that a packet buffer at receiver or intermediate node is full and decreases the send window size in order to slow the transmission rate and prevent further loss. In rate-based flow control, the sender uses a timer to control the transmission rate. When the timer fires, the next packet is transmitted. The sender periodically adjusts the timer value based on feedback information. If packet loss is reported, the timer value is incremented to slow down transmission. Otherwise the sender decreases the timer value to accelerate transmission.

It is natural to use window-based flow control with an ACK-based reliable multicast protocol, since the sender regularly receives feedback. On the contrary, rate-based flow control is more appropriate for a NACK-based protocol. One advantage of rate-based flow control is that the traffic is smoother than that under window-based flow control. Specifically, the inter-packet delay in rate-based approach is usually evenly distributed, while in a window-based approach the delay can be close to zero between packets in the same window and large between packets in different windows. However, a rate-based scheme with a NACK-based protocol may suffer from the infrequency of feedback. In particular, it might take a long time for a receiver to detect packet loss when a large

number of consecutive packets are lost due to congestion. Feedback from the receiver is delayed, leading to even more lost packets. Both window-based and rate-based flow control can integrate algorithms such as random early detection (RED) [26] to slow down transmission before severe congestion occurs. The router adopting the RED algorithm drops packets when the buffer consumption exceeds a threshold value, rendering a feedback of packet loss before the buffer is full. Moreover, receivers can send explicit rate requests to the sender to adjust the transmission rate in a more accurate way, as in the RAMP [32] and RMC [69] protocols.

Most flow control approaches use data loss as the primary, or sole, indication of congestion. This assumption is valid in a traditional wired network, where packets are rarely lost during propagation. The situation is very different, however, in a wireless network. Packets frequently are lost or corrupted due to signal fading and shadowing. In this case, slowing down transmission is counterproductive since the packet loss is not caused by congestion. Many strategies have been proposed to differentiate transmission losses from congestion losses. We discuss several approaches in Chapter 6.

2.3 Forward Error Correction

Forward Error Correction (FEC) is an approach to communication error recovery that uses redundant information in the data stream to enable receivers to correct losses without contacting the sender. Two classes of FEC algorithms are bit-level and packet level. In a bit-level method [18], redundant bits are appended to a frame. When some data bits are corrupted during propagation, receivers detect and repair the corruption using the

redundant bits. In a packet-level approach [24], parity packets are generated and sent together with data packets. When some data packets are lost or corrupted during transmission, the parity packets can replace the same number of any lost data packets. In both cases, if the redundant information is sufficient, the lost or corrupted information can be regenerated locally at the receiver without retransmission. Use of FEC can greatly improve performance for loss-sensitive applications when the feedback channel is unavailable or expensive to use.

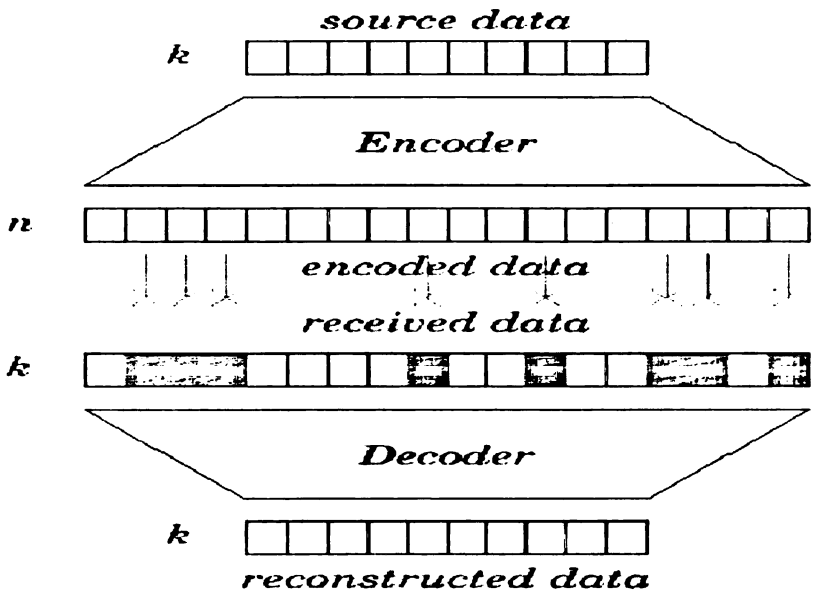


Figure 2.5: Operation of FEC algorithms [56].

FEC algorithms generally work as shown in Figure 2.5. The transmitter applies an encoding algorithm to a set of data bits or packets, transforming them into a larger set of bits or packets. The larger set includes all information in the original set. The transformation guarantees that any subset of the larger set with the same number of elements in the original set is sufficient to regenerate all the original data. After receiving any of such subset, a receiver applies a decoding algorithm to it in order to produce the

original data. Both encoding and decoding algorithms are usually computationally intensive.

FEC has been used for many years in wireless communication, where the error rate is relatively high. Most of this usage is at the bit level, and is supported by hardware. The encoding and decoding algorithms are implemented in special circuits to provide fast processing. On the other hand, packet-level usage is usually integrated in computer network protocols. Since there is unlikely any special hardware support for such applications, its usage is limited by the speed of software encoding at the transmitter and the software decoding at receiver.

Rizzo [56] proposed a software solution for FEC encoding and decoding at packet level. The approach is based on the fact that, in terms of errors, many data communication protocols need to deal only with *erasures*, that is, missing packets in a stream caused by MAC-level CRC errors. Since erasure recovery is much simpler than arbitrary bit error recovery, it is possible to implement such an algorithm in software with reasonable processing speed. In the proposed erasure code, k data sources items are interpreted as coefficients of a polynomial with degree k-1. Obviously such polynomial is completely specified given the values of k different points. If we construct a set with n different points of the polynomial, any subset of k such points are sufficient to reconstruct the polynomial. If we replace "point" by "bit" or a "set of bits", then the algorithm is applicable to FEC encoding and decoding. Experiments in [56] show this scheme exhibits reasonable performance on typical computer hardware. As with many other studies of FEC for wireless communication, we use Rizzo's erasure code in this study.

At the packet level, a single FEC parity packet can recover any lost packet in the *k*-packet group. Similarly it can recover any single packet loss at other receivers. This property is ideal for multicast environments where the loss correlation among different receivers is often low. WLANs are one such environment. For example, suppose that three receivers are sent 10 packets and that receiver1 misses packets 1 and 2, receiver2 misses packets 3, and 4, and receiver3 misses packet 5. In an ordinary ARQ approach, the transmitter would need to retransmit five packets: 1, 2, 3, 4, and 5. However, using an FEC protocol, only two parity packets are required to recover all the losses. Furthermore, the reduction in the number of packets transmitted produces no other communication overhead. It requires only some additional computation for encoding and decoding. Since Rizzo showed that this computation is affordable on ordinary processors, FEC is widely used in multicast communication protocols.

Parity packets can be sent *proactively* with data packets, or *reactively* upon reception of a retransmission request. Both methods possess the aforementioned retransmission-reduction property. In the proactive scheme, no retransmission is needed if the number of proactive parity packets is large enough to recover losses. This property is desirable for communication environment with expensive feedback channel, such as satellite network. However, sending too many unneeded parity packets with the data will waste channel bandwidth and eventually degrade performance. On the other hand, the (proactive or reactive) parity packets themselves can be lost. Therefore, a transmitter usually needs to send more parity packets than are actually needed. It is an open research problem to determine the proper balance between proactive and reactive methods. We address this

problem, along with several other issues on reliable multicasting in WLANs, in this study.

Chapter 3

Issues in Reliable Multicast over Wireless LANs

The goal of this research is to find efficient reliable multicast algorithms for WLANs. To do so, it is first necessary to understand the characteristics and behavior of wireless channels. Thus we began by collecting a large number of packet traces from our experimental environment. Based on these traces, we characterize properties of typical wireless loss patterns and point out factors that affect protocol performance. Next, we studied several component algorithms used in existing reliable multicast protocols by evaluating their behavior and performance in WLAN environments. Based on this experimentation and analysis, we developed a new protocol (see Chapter 4).

In this study, we assume that the primary performance metric is throughput. We further assume there is little other traffic in the WLAN, which is common in many application environments. Therefore, the task is to find an algorithm that maximizes throughput for a given network bandwidth, number of receivers, wireless propagation loss rate and access point buffer size. To enhance scalability, we consider only NACK-based approaches and, because the protocol is intended for local area networks, we assume there is no receiver hierarchy. Finally, we focus mostly on the scenarios where

the sender is located in the wired network, while the receivers are in the wireless network. We believe that this is a typical working environment for reliable multicast protocols in heterogeneous networks. In this study, all the concerns, analyses and protocol optimizations are based on these assumptions.

3.1 WLAN Channel Characteristics

3.1.1 Experimental Environment

The experimental environment, depicted in Figure 3.1, consists of a wired LAN and a wireless LAN. The wired LAN is a 100Mbps Fast Ethernet that connects several high-end workstations. It is extended by a Cisco Aironet WLAN through a Cisco Aironet 340 Series Base Station [12]. The wireless stations include desktop computers and Dell laptop computers, each configured with a Cisco Aironet 340 or 350 Series Wireless Card. The Aironet network is IEEE 802.11b compatible with CSMA/CA access control, operating at 2.4GHz. It is a DSSS (Direct Sequence Spread Spectrum) system that supports raw bit rates of 1Mbps, 2Mbps, 5.5Mbps and 11Mbps. The coverage area is larger for lower bit rates and smaller for higher bit rates. For example, the typical range is 300 feet indoors and 1500 feet outdoors at 1Mbps. At 11Mbps, these distances are 100 feet and 400 feet, respectively. We focus on the highest 11Mbps rate in the experiments since our primary performance metric is throughput.

The high-end workstations are mainly Dell desktop computers with a 1GHz Pentium III CPU and 512MB memory. The laptop computers have either 1GHz Pentium III CPU, or 300MHz CPU, both with 256MB memory. Both the desktop and laptop computers are

configured with either Windows NT or Windows 2000. As mentioned above, in the experiments we located the sender on a wired workstation, while the receivers were located on laptop computers or to desktop computers with wireless network interface cards.

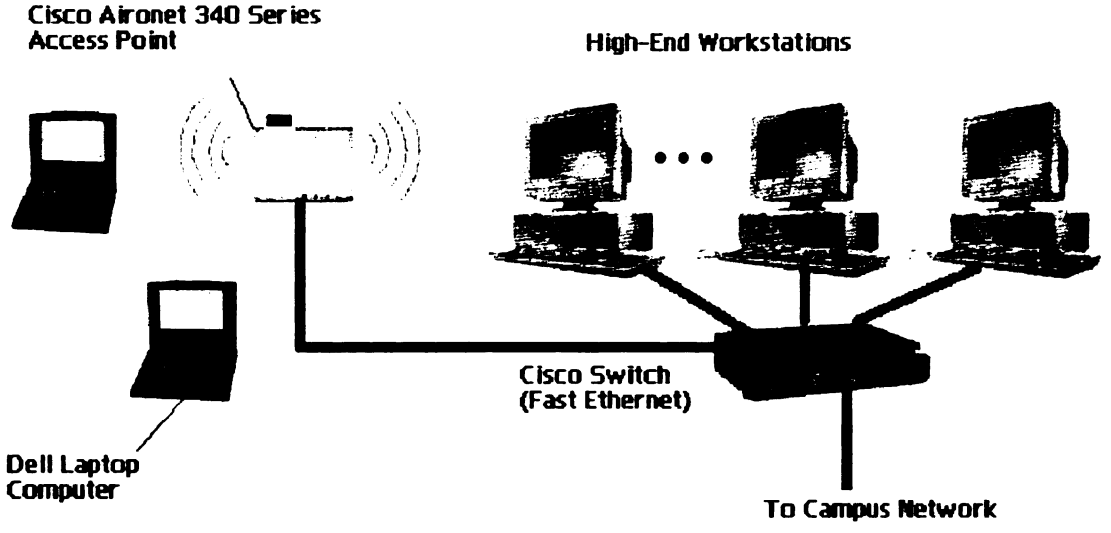


Figure 3.1: Experimental environment.

3.1.2 Packet Trace Collection

We conducted a series of data transmissions under various scenarios and collected packet traces. We built two programs using the Windows socket interface. The sender program transmits data packets at a specified rate to one receiver or to a set of receivers. Each data packet is assigned a unique sequence number. The receiver program checks the sequence numbers of the received packets to detect packet loss. It records the sequence numbers of lost packets, and saves these numbers into packet trace files together with information on corresponding loss burst lengths. From a saved trace, we can calculate the

average error burst length, average error free burst length, standard deviations and burst length distributions. These values reflect the basic characteristics of packet loss patterns.

It is possible that the detected losses are caused by reasons other than wireless propagation errors, since the programs are running at the application level, and the packets are transmitted from a wired sender to wireless receivers via an access point. For example, congestion can cause buffer overflow and packet loss. Congestion can happen at the access point when the data rate of the wired network is greater than the bandwidth of the wireless network. Congestion also happens at receivers if the receiver program is not running fast enough to process packets and empty kernel buffers before new packets arrive. We observed very high packet losses when packets are transmitted without flow control. Such losses are very likely caused by buffer overflow. To reveal wireless channel loss patterns, these "non-propagation" losses should be excluded from the trace files.

Applying flow control is an effective approach to reduce congestion and packet drops. Taking into account the theoretical saturation throughput of approximately 7Mbps in an IEEE 802.11b network, we insert inter-packet delays to limit sending rate at 6Mbps. In case no other traffic exists on the wireless channel, this action should produce very few if any drops due to congestion. To test this hypothesis, we collected packet traces at much lower sending rates, such as 2Mbps. It turns out the traces in this case are very similar to those with 6Mbps sending rate. We conclude that transmission at 6Mbps is almost congestion-free in our environment. Therefore we set sending the rate to 6Mbps in the following experiments since it is close to the optimal throughput of the protocol.

We believe that our receiver programs are able to keep up with such a sending rate. By calculation, the receiver program may take as long as 1.9 milliseconds to process one

received packet with a typical size of 1400 bytes, without delaying the processing of the next packet. This interval is long enough for a 300MHz plus processor to carry out very complex per-packet processing. In fact, very few losses were observed in the experiments when transmitting small files (e.g., 20KB) at 6Mbps, since the error free burst length is usually much greater than 20KB. We observed substantial losses only when the inserted delay between processing of consecutive packets exceeded 2 milliseconds. The result is in accordance with our calculation.

In WLANs, the RTS/CTS/ACK mechanism is utilized to reduce collisions as well as to conduct fast link layer retransmissions. Although the mechanism helps improve the quality of wireless transmissions, it compromises the accuracy of raw propagation loss estimation. Since this mechanism is not adopted for multicast transmissions, we use multicasting to transmit data packets in our experiments. Specifically, we use UDP sockets on top of the IP multicast services. After all these considerations and revisions, we believe it is possible to obtain reasonably accurate wireless loss estimation, even if the trace-collecting programs are running at the application level.

3.1.3 Packet Trace Analysis

Queueing Loss. Although we observed a large amount of packet loss in transmissions without flow control, we could not conclude that the losses were caused by buffer overflow at the access point, without further experimentation. For example, it is possible that the higher sending rate caused more propagation errors. To verify that the losses are congestion-related, we set up two wireless receivers in the following test. One receiver is close to the access point (within 2 meters) while another is farther way (about 10 meters).

The sender is a wired station. The data packets are transmitted first at 6Mbps, then without flow control. Transmissions are grouped in 20 packets. Upon detecting a packet loss, the receiver sends a NACK to request a retransmission. The sender responds by sending enough parity packets to cover the loss, plus additional parity packets to compensate for any losses in the retransmission (see Chapter 4). The number of needed and received parity packets for each group is depicted in Figure 3.2.

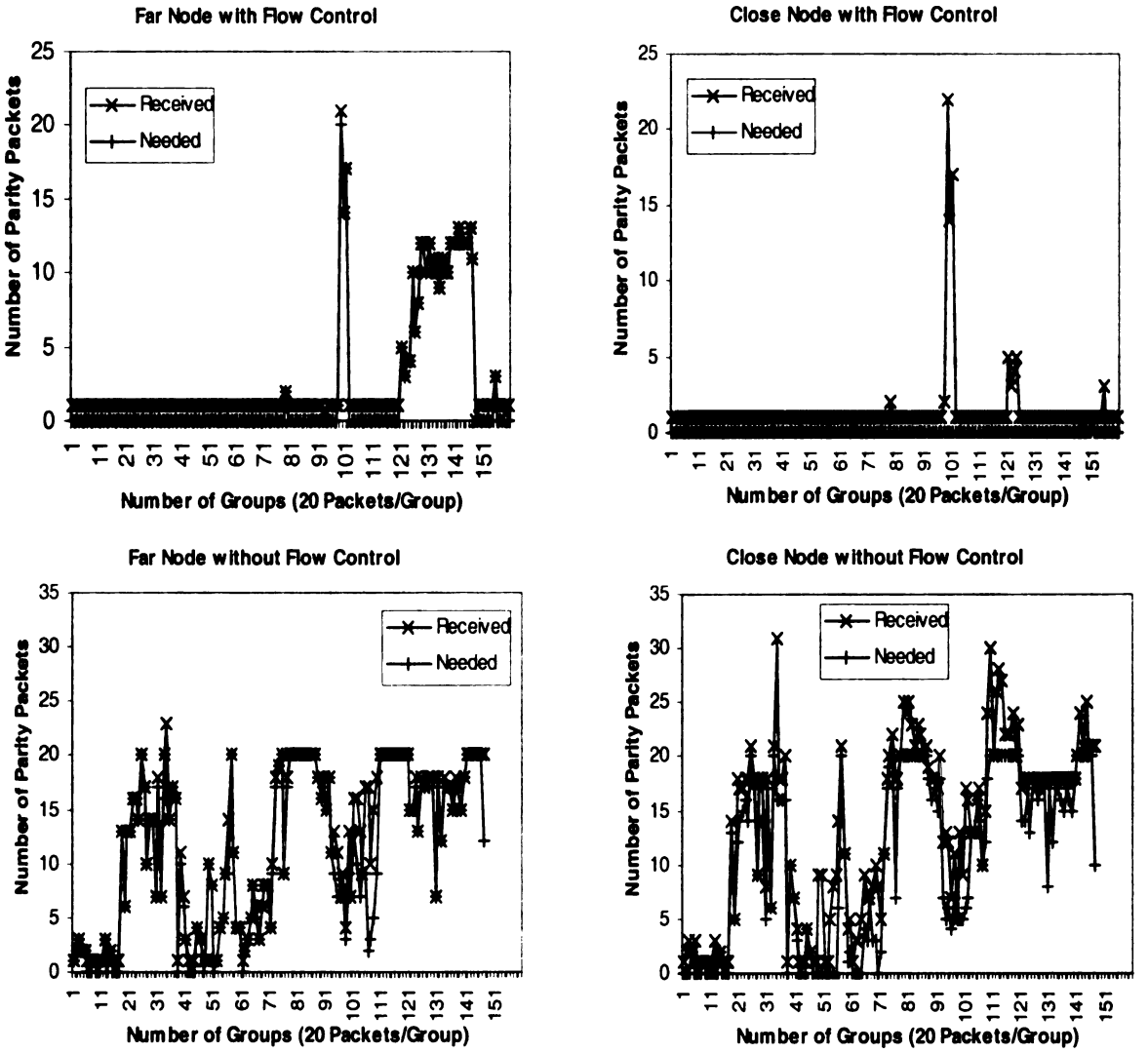
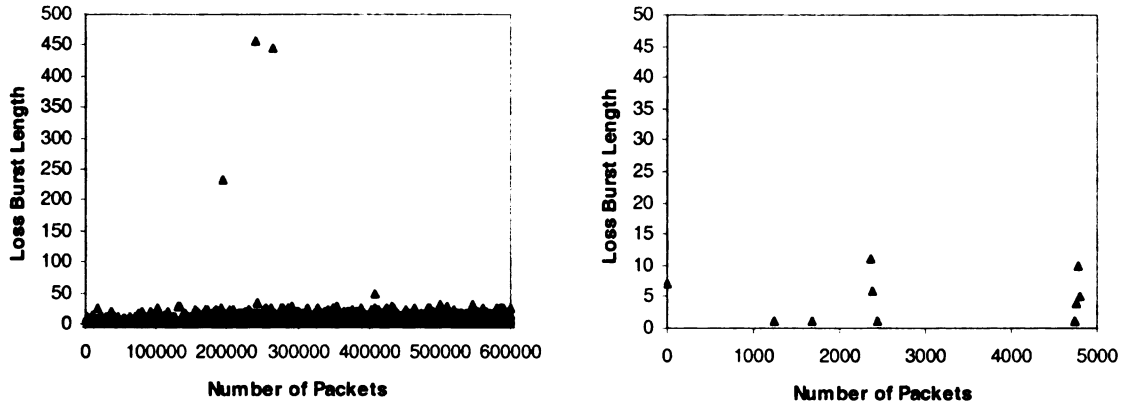


Figure 3.2: Experimental evaluation of queueing loss.

The plots show that stations farther away from the access point experience higher packet losses. This is expected since the signal energy degrades with distance. The poor signal quality at a remote receiver causes more data bits to be indiscernible, and the corresponding packets are dropped. Moreover, it is shown that the packet losses are mostly uncorrelated at two receivers when flow control is applied. Since propagation error is location dependent, this trace (and other similar traces) implies that most of the losses in this case are propagation losses. On the other hand, these two receivers share a large number of packet losses when flow control is not applied, even if they are at different locations. A reasonable explanation is that the shared losses are caused by buffer overflow at the access point. Since the packets are dropped at the access point and not propagated in air at all, receiver locations have no effect on such losses.

Bursty Propagation Loss. We also wanted to understand the nature of packet loss bursts. In the following test, the sender transmits data packets to a wireless receiver close to the access point. The sending rate is set to 6Mbps. We expect that there should be few packet losses since signal degradation is not a major concern in this case. We collected several 20-minute packet traces. The loss patterns are similar. We plot one typical trace in Figure 3.3.



(a) A broad view (b) A detailed view

Figure 3.3: A typical packet trace in WLAN.

As shown in the figure, wireless propagation loss is very bursty. In good states, there are very few packet losses. In a bad state, however, most packets are lost. In Figure 3.3, although the average loss rate is fairly low, the receiver experiences severe losses at certain points in time. The burst could be as long as several hundred packets. However, most bursts are only several packets long and there are many single-packet losses. On the other hand, the error free burst, or the packet distance between two consecutive loss bursts, is usually several hundred packets long. Many error free bursts were even longer than ten thousand packets. Therefore, applications running over wireless networks may experience no packet loss at all during a long time period. But it is also possible that they lose many packets in a loss burst. This dynamic behavior poses a major challenge to many applications, including reliable multicast protocols.

Loss Correlation. A multicast transmission involves multiple stations. Besides studying loss distribution at a single station, it is also meaningful to examine the relationship among losses at multiple stations. In a multicast group, loss patterns vary at different

wireless stations because some losses are location-dependent. On the other hand, all receivers share the data source. Packets are transmitted from a wired station to the wireless network through an access point. Variations such as transmit power level changes at the access point may affect the packet loss probability at all the receivers. Those losses are likely to be correlated. To study the degree of the loss correlation, we set up ten wireless stations as receivers, including laptops and desktops with installed wireless cards, in the following test. The nodes are located at various locations in the range of the access point, from within 2 meters to about 30 meters away. Figure 3.4 shows the receiver locations relative to the SENS Laboratory at Michigan State University. The receiver configurations and the distances to the access point are summarized in Table 3.1. We combine packet traces from different systems and depict them in Figure 3.5. (Only 5 traces are shown for simplicity.)

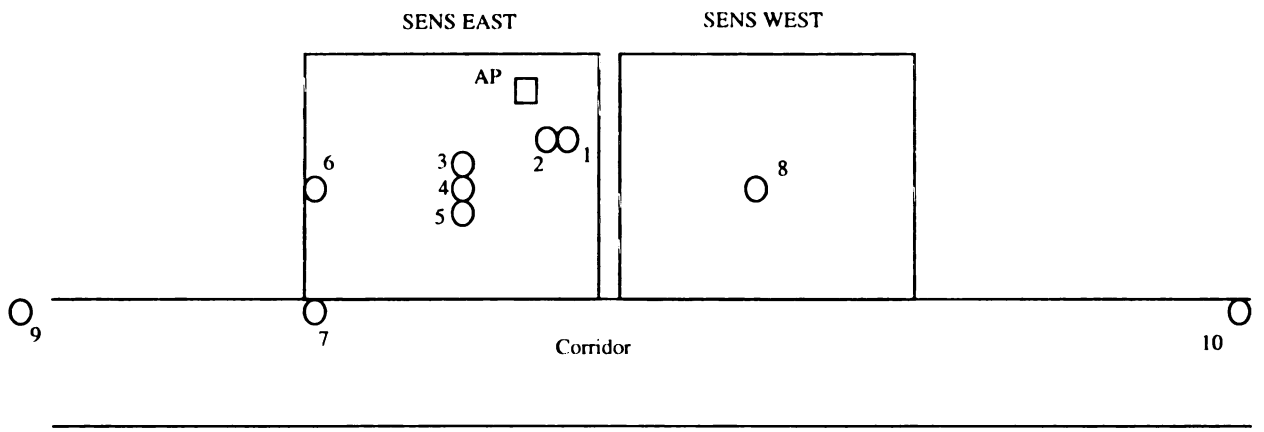


Figure 3.4: Receiver locations.

	node1	node2	node3	node4	node5	node6	node7	node8	node9	node10
Type	D	L	D	D	L	L	L	D	L	L
CPU(MHz)	2*400	1000	1000	1000	300	1000	1000	2*800	1000	1000
Memory(MB)	256	256	512	512	256	256	256	256	256	256
Distance(m)	2	2	4	5	5	10	10	10	20	30

D: Desktop, L:Laptop

Table 3.1: Test machine configurations.

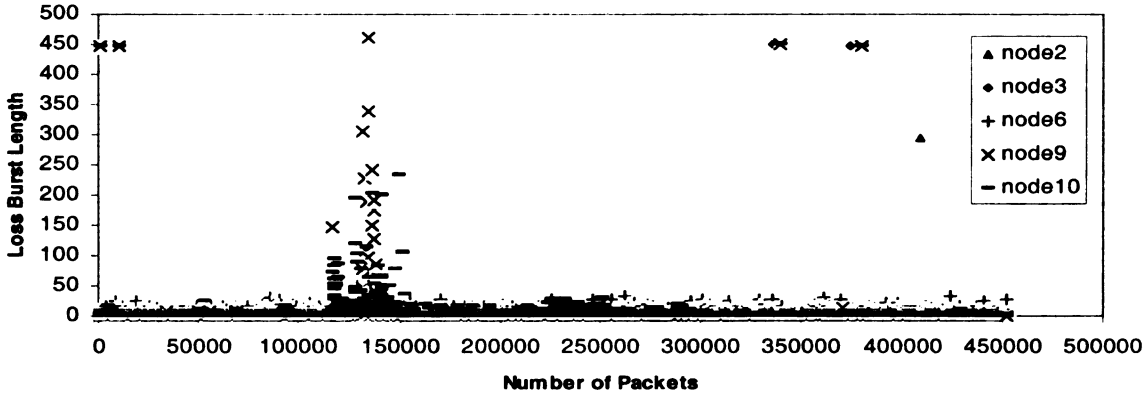


Figure 3.5: Packet traces at multiple stations.

The traces exhibit different packet loss patterns. Basically, the distance from the access point determines the average packet loss rate at a receiver. In Figure 3.5, the receivers that are more than 20 meters away from the access point exhibit poor performance. On the other hand, there is not much difference among receivers that are located within 10 meters from the access point. We plot the average loss rate, average loss burst length and average loss-free burst length in Figure 3.6.

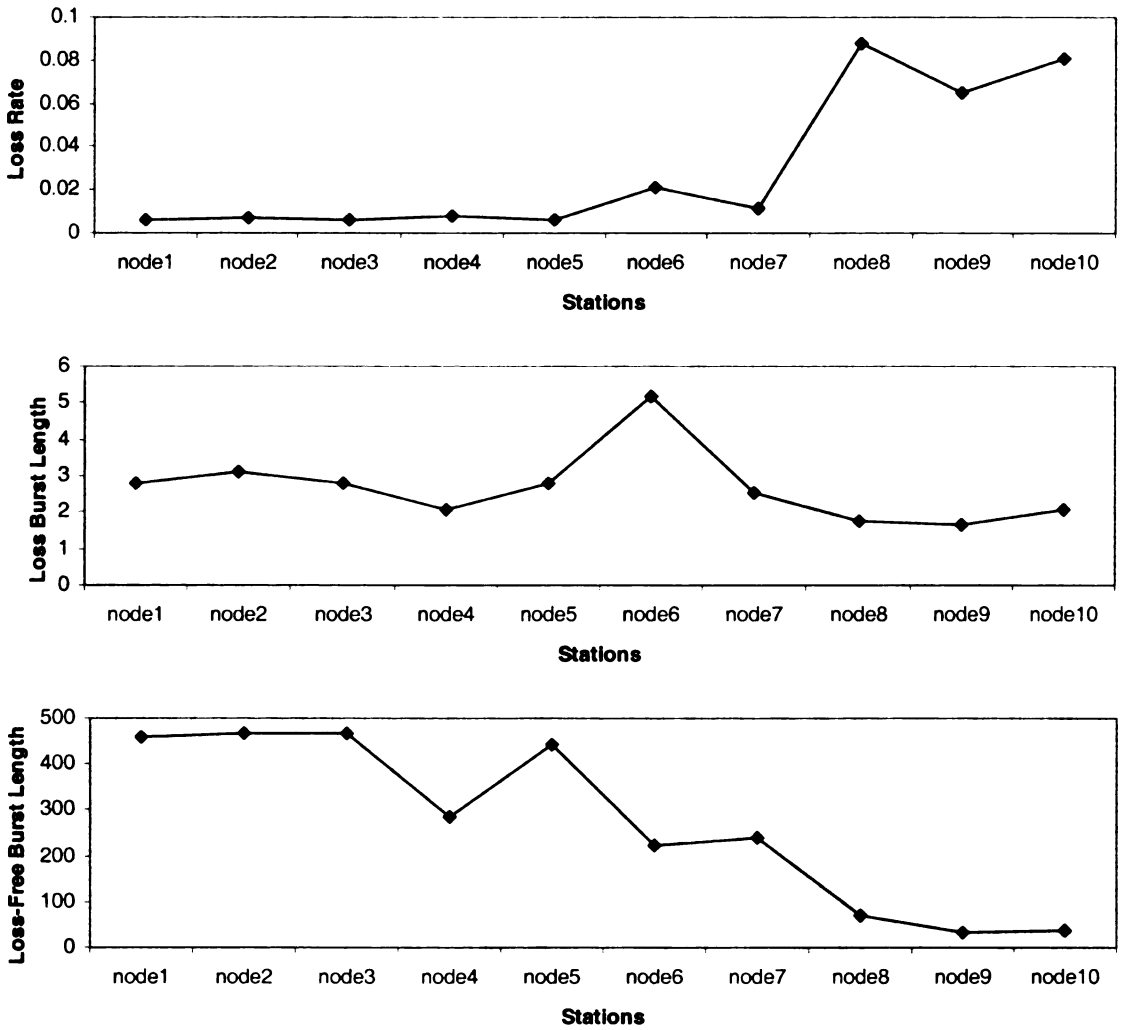


Figure 3.6: Packet trace statistics.

The figures show that loss rate, as well as loss-free burst length, is mainly determined by the receiver distance from the access point. However, loss burst length is affected only by factors in the local environment around a station. It is clear that the average loss rate is basically dependent on loss burst frequency instead of loss burst length.

Next we studied the loss correlation among packet traces. We built a program that compares packet status (received or lost) packet by packet in two traces. Based on the results, we calculated the correlation coefficient of each trace pair and plotted them in

Figure 3.7. For comparison, the correlation coefficient between traces in different test runs is also calculated and shown. This coefficient is near zero (as expected) since the losses are certainly uncorrelated among different tests.

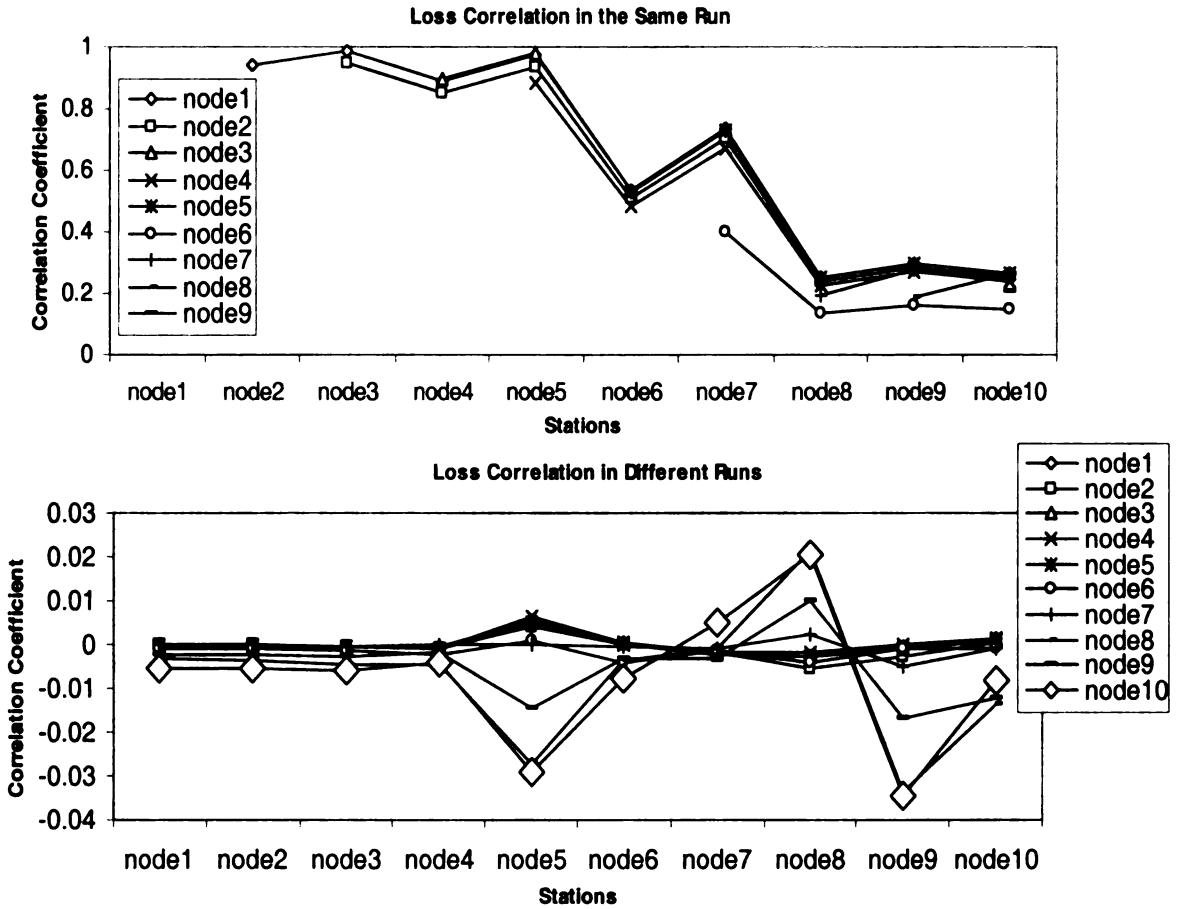


Figure 3.7: Loss correlation of multiple traces.

As shown in the figure, receivers within 10 meters of the access point exhibit high loss correlation. On the other hand, receivers farther away from the access point show lower loss correlation. This is probably due to the fact that signals with higher energy, such as those within short ranges around the access point, are less vulnerable to random interferences than those with lower energy at remote receivers. However, the remote receivers still share a certain amount of loss. In Figure 3.7, the *minimum* value of the

correlation coefficient is approximately 0.13 for traces in the same run. It is still much higher than the maximum coefficient of 0.03 between traces from different runs.

Effect of Reverse Traffic. In reliable multicast protocols, receivers send feedback (either ACKs or NACKs) to the sender. In WLANs, the reverse traffic shares the communication channel with the forward data traffic. Although the CSMA/CA algorithm is used in WLANs to reduce collisions, collisions are unavoidable especially for a large number of receivers. On the other hand, contention for the wireless channel reduces the data transmission rate and decreases the service rate at the access point. Incoming packets will eventually use up the buffer space and cause queueing losses, if the sending rate is fixed. We verified this effect through experimentation. In the following test, we set up a wired sender and a wireless receiver. The sender transmits data packets of 1400 bytes to the wireless network at a rate of 6Mbps. At the same time, another wireless station sends packets of 64 bytes to the sender. The sending rate is approximately 0.3Mbps. Hence, there are roughly as many reverse packets as data packets. In the first 5 minutes, no reverse packets are sent. Then, the wireless station starts to transmit packets by multicasting for 15 minutes. After that, the wireless station stops for 5 minutes, and then transmits packets by unicasting for another 15 minutes. The wired sender keeps sending data packets and terminates 5 minutes later. The wireless receiver records the packet trace, which is depicted in Figure 3.8.

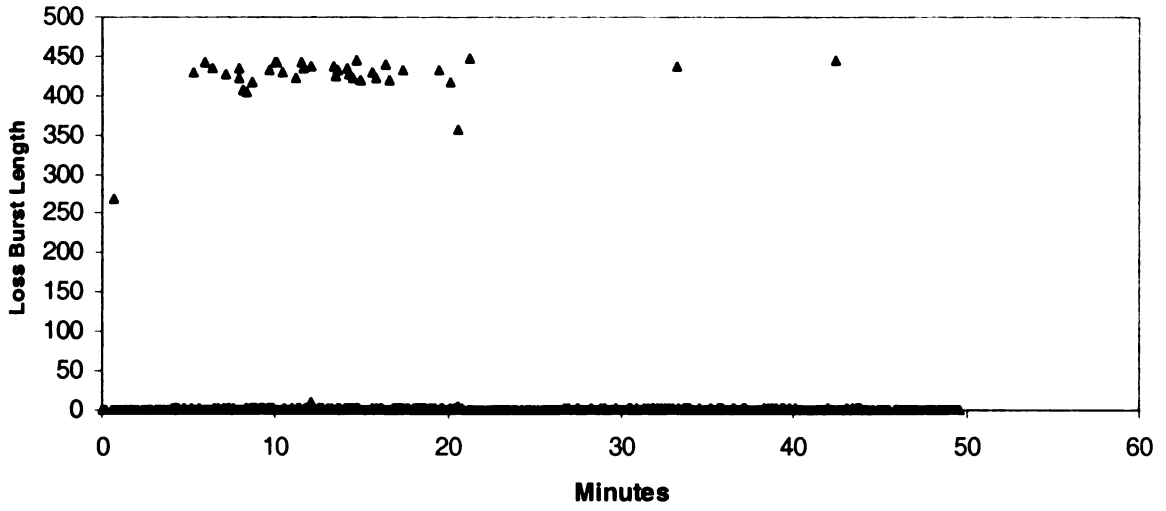


Figure 3.8: The effect of unicasting and multicasting reverse traffic.

In the figure, the packet loss rate clearly increases when the wireless sender starts sending reverse packets. The loss rate drops during the interval when the wireless sender stops sending. As the wireless sender begins transmitting reverse packets by unicasting, the loss rate increases again. However, this time it is much lower than that in the case of multicast reverse traffic. The result is in accordance with the following analysis. In 802.11 WLANs, an access point relays all uplink multicast traffic by forwarding the packets both uplink and downlink. It adopts this strategy because it maintains no multicast group membership information. On the contrary, unicast reverse packet is forwarded either uplink or downlink according to its destination. As a result, data traffic is further slowed down in the case of multicast reverse traffic, and more queueing losses are generated. Therefore, in WLANs multicast feedback produces greater overhead than does unicast feedback. This overhead can directly affect the performance of some optimization approaches like global NACK suppression.

Taking into account the impact of channel contention on the access point service time, it is obvious that there will be more queueing losses generated at the access point if

multiple wireless stations transmit reverse packets simultaneously. In the following test, we set up 3 wireless receivers. A wired sender transmits data packets of 1400 bytes at 6Mbps. After 5 minutes, a wireless station sends reverse packets by unicasting. Then it stops and waits for a while, followed by sending multicast reverse packets. Four more wireless stations join the transmission ensemble after a while. They all send packets by unicasting followed by multicasting, stopping for a while in between. The time unit for both waiting and transmitting is 5 minutes. The sending rate of reverse traffic is approximately 0.1Mbps. As expected, we see in Figure 3.9 that the packet loss rate further increases when more wireless stations start transmitting packets.

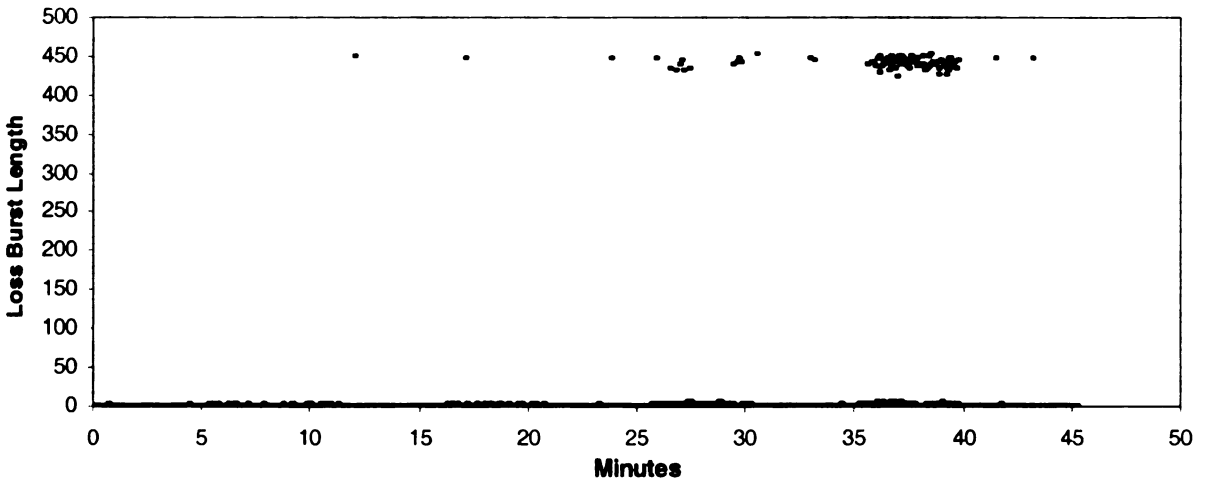


Figure 3.9: The effect of reverse traffic from multiple stations.

3.2 Reverse Traffic and Packet Loss

How to handle reverse traffic is critical for reliable multicast protocols. Reverse traffic is caused by packet loss, but at the same time it also introduces additional packet loss, as

we saw in Section 3.1. In this section, we present a further analysis on reverse traffic as well as its effect on both packet loss and overall protocol performance.

There are several possible causes of packet loss. First, packets can be dropped at a receiver due to buffer overflow. Second, propagation errors such as signal degradation or interference can cause packet corruption. Third, collisions between data traffic and reverse traffic can lead to packet loss. Lastly, a packet can be dropped at the access point if the sending rate is too high or if the service rate (the rate at which an access point forwards data packets into the wireless network) is too low due to contention. In the previous section we show that, considering current computer processing speeds and network bandwidth, a receiver is usually fast enough to complete processing a packet before the next packet arrives. Therefore we only consider the last three cases.

3.2.1 Propagation Loss

Let us consider a multicast group with R receivers. The data are organized in (FEC) groups, each of which contains k packets. Again assume the propagation loss rate is $p(r, g)$ at a particular receiver r, and in an arbitrary packet group g. Then the probability of the receiver transmitting a NACK is $1 - (1 - p(r,g))^k$, since a receiver transmits a NACK whenever a packet loss is detected in the group. The expected total number of NACK packets at group g could be expressed as:

$$Nn(g) = \Sigma_{r=1}^{R} (1 - (1 - p(r,g))^k)$$

The total number of NACK packets during the transmission of the current resource is:

$$Nn = \Sigma_g (Nn(g))$$

Apparently, $Nn(g)$ depends on R, p and k. To reduce the number of NACK packets in a packet group, a protocol should choose smaller data group size k. (It is unlikely that the protocol can control the other two parameters: multicast group size R and propagation loss rate p) However, as far as Nn is concerned, using a larger data group is more effective to reduce the total number of NACK packets. On the other hand, using larger data group demands more buffer space at both the sender and the receivers. Doing so also lengthens per-packet recovery time, and probably increases total transmission latency. The optimal value of k is application dependent.

3.2.2 Collision Loss

Collision loss is caused by collisions between reverse traffic and data traffic. Intuitively, collision loss rate is proportional to NACK density. In other words, the more NACK packets transmitted in a given time interval, the more collisions there might be. On the other hand, a collision does not necessarily result in a packet loss at high layers. The RTS/CTS approach and link-layer retransmission mechanism can recover most collision losses without requiring the intervention of higher layer protocols. However, these mechanisms are helpful only to unicast or uplink multicast traffic (In 802.11 WLANs, the uplink multicast transmission is implemented as a unicast transmission from the wireless sender to the access point, followed by a multicast transmission from the access point). The downlink multicast data traffic, therefore, is more vulnerable to collisions than feedback traffic. In our system, the access point regularly transmits data packets with a particular rate, while receivers irregularly send NACK packets. It is difficult to build an accurate analytical model of this kind of system. To provide a rough

estimate of the collision rate, we adopt the simple model proposed in [8]. The model assumes that all stations are in asymptotic condition, that is, each station always has a packet ready to transmit. All transmissions are conducted by unicasting, which is protected by link layer retransmissions. Although this assumption is not true in our system, the model does take into account the basic collision and contention interaction in an 802.11 WLAN. We describe the model below.

Let us assume that the probability that a station transmits in a randomly chosen slot time is τ, the minimum backoff window size is W, the maximum backoff stage is m, and the collision rate is p_c. Then in [8] it is shown that τ can be expressed as:

$$\tau = 2 / (1 + W + p_c W \Sigma_{i=0}^{m-1} (2p_c)^i)$$

On the other hand, assume that n (equivalent to $Nn(g)$ in section 3.2.1) is the number of other stations that transmit at the same time. The collision rate p_c experienced by this station can be expressed as:

$$p_c = 1 - (1 - \tau)^n$$

We can solve p_c and τ by using numerical techniques. From τ we get the conditional probability p_s that a transmission from a particular station in a slot is successful, and the probability p_{sany} that any transmission in a slot is successful.

$$p_s = \tau * (1 - \tau)^n / (1 - (1 - \tau)^{n+1})$$

$$p_{sany} = (n + 1) * p_s$$

We set W to 31 and m to 4, as they are in the 802.11b standard. We plot the values of τ, p_c, p_s and p_{sany} in Figure 3.10, with varying number of stations. The X-axis shows the value of n, i.e., the total number of stations minus 1.

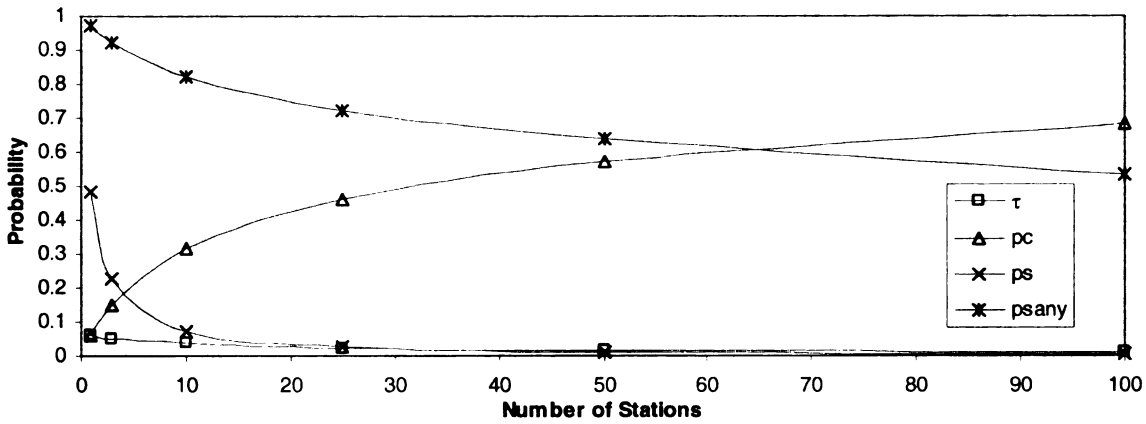


Figure 3.10: Collision rate and transmission rate with asymptotic stations.

The figure shows that the collision rate increases with the number of transmitting stations. On the other hand, the rate of successful transmissions for a particular station drops rapidly as the network becomes moderately large. The drop of the transmission rate is due more to contention rather than to collisions. The collision rate as well as the transmission rate are likely different in our system, however, where the access point has a special role in that it transmits data packets from the wired network to the wireless network. Other wireless stations irregularly send NACK packets. The nodes are not in asymptotic conditions and the NACK packet size is far smaller than the data packet size. Moreover, the maximum backoff stage m for downlink multicast data traffic is 1 instead of the value 4 for unicast data traffic [25]. Thus, we studied the collision rate by using simulation. In the following simulation, a wired sender transmits 1000 packets to the

wireless network. After receiving every 20 packets, each wireless station sends a NACK packet. The data packet size is 1400 bytes and the NACK packet size is 36 bytes. The sending rate is fixed at 6Mbps. The number of collision losses for data packets is depicted in Figure 3.11. The number of total collisions is also shown.

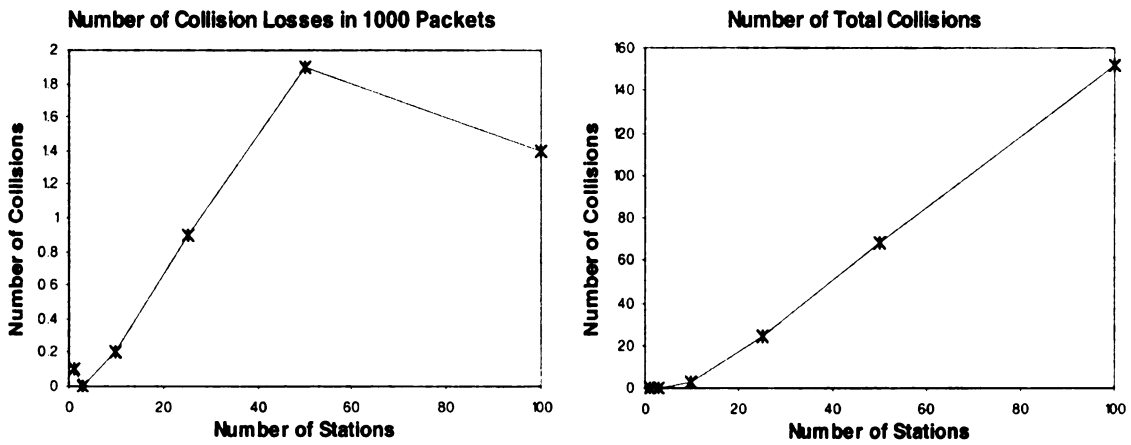


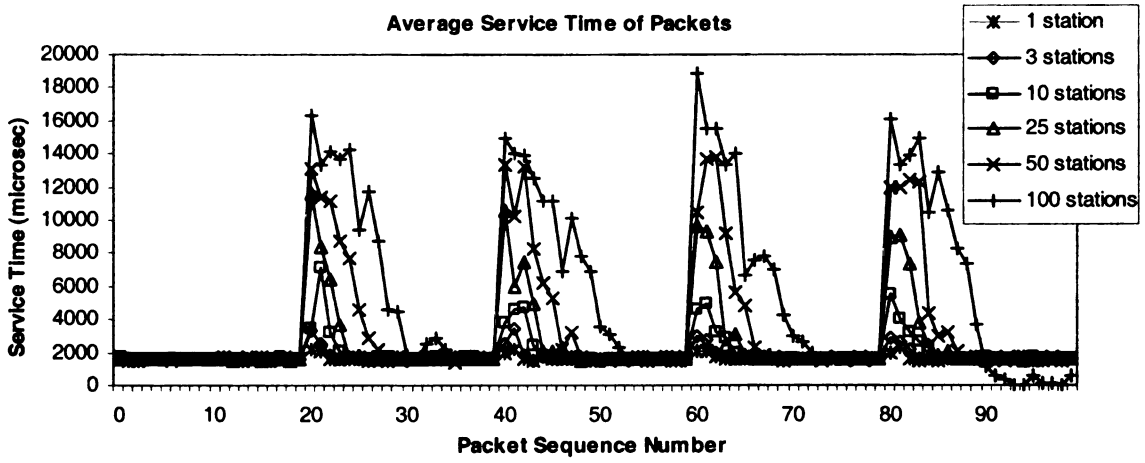
Figure 3.11: Collision rate in simulation.

The figures show that the collision rate in our system is much lower than the rate in an asymptotic system. The collision loss rate, i.e., the rate of data packet losses due to collisions, is even lower. Among 1000 data packets, fewer than two packets on average are lost due to collision, even in a network with 100 NACK-transmitting stations. However, we will see later in this chapter that the collision rate is considerably *higher* when a NACK suppression approach is adopted.

3.2.3 Contention and Queueing Loss

Incoming packets will be dropped at the access point in case that the buffer is full. The degree of buffer utilization depends on the arriving rate and the service rate. In steady state, there are no packet losses, and the average arrival rate is equal to the service rate. As packets get lost in propagation and receivers send NACKs, this balance is broken. In this section we study the effect of reverse traffic on queueing losses.

Reverse traffic contends for the channel with the data traffic. When a larger number of receivers try to access the channel, the chances that the access point can seize the channel become small. Therefore the data transmission is slowed down. As a consequence, the arriving rate becomes higher than the service rate. Eventually it will lead to buffer overflow and packet drop. We study contention and queueing loss though simulation. In the following test, we assume that the buffer capacity is 50 packets, data packet size is 1400 bytes, NACK packet size is 36 bytes, and the sending rate is fixed at 6Mbps. Each receiver transmits a NACK packet after receiving every 20 data packets. We plot the average service time at the access point as well as the number of queueing losses in Figure 3.12.



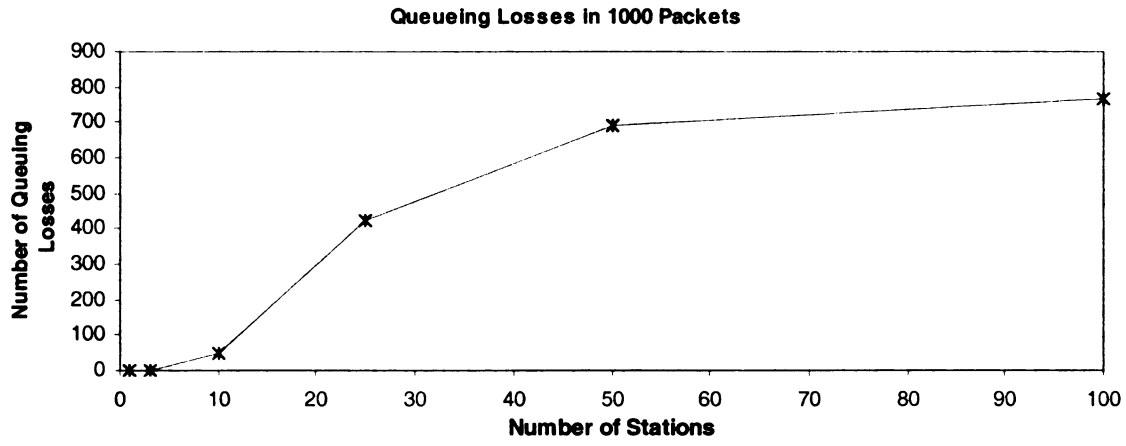


Figure 3.12: Contentions and queueing losses.

In the first figure, the per-packet service time (the time needed for channel access and packet transmission) increases drastically when receivers start transmitting NACKs. It reflects the degree of channel contention. When more stations transmit, the service time becomes longer. This slowing down at the access point causes many packet drops. The second figure shows that more than 50% of packets are dropped due to buffer overflow in a moderately large network. These severe losses will lead to a significant drop in performance. Therefore, NACK suppression as well as flow control is critical to the performance of a reliable multicast protocol.

3.3 Reliable Multicast Strategies

Based on our understanding of packet losses and traffic interactions, we describe and evaluate a number of well-known reliable multicast protocol components in this section. We design our protocol by combining the most efficient algorithms in retransmission and NACK suppression.

3.3.1 Retransmissions

There are two major approaches for loss recovery. One approach is ARQ, and the other is FEC. Upon receiving a NACK packet, the sender either retransmits the requested data packets as in ARQ, or transmits some parity packets as in FEC. We refer to as retransmission approaches. In ARQ, the repair packets are retransmitted data packets, while in FEC they are parity packets. Repair packets can be sent either by unicasting or by multicasting. Although the lack of link layer retransmission is a non-trivial shortcoming of the multicast approach, multicasting is still far more efficient than multiple unicasting in WLAN environments, where all stations share the channel.

Compared to ARQ, the FEC approach is computationally intensive and communication efficient. It incurs larger computational overhead because this approach requires extra computational effort to encode packets at the sender and decode packets at the receiver. It entails smaller communicational overhead because multiple receivers can use the same parity packet to recover different packet losses, which leads to less retransmission traffic. Considering that the computing capability increases faster than network bandwidth does, we favor a communication efficient approach and adopt the FEC approach in our protocol.

3.3.2 NACK Suppression

In previous sections we show that reverse traffic can cause data losses or slow down transmissions. To improve the protocol performance, it is desirable to keep reverse traffic as light as possible. We describe three approaches that can reduce NACK packets:

proactively sending FEC parity packets, local NACK suppression and global NACK suppression.

Proactive FEC. In this approach, the sender proactively sends a certain number of FEC parity packets along with the data packets. In case that some data packets are lost, a receiver can recover the losses without sending retransmission request if enough FEC parity packets have been received.

To determine the maximum gain of the proactive FEC approach, we first assume that the channel prediction is perfect, that is, we know exactly how many packets will be lost in a group. Let k be the number of data packets in a group, and α be the proactive rate, then αk is the number of proactive parity packets in the group. Let $p(r)$ be the average packet loss rate at receiver r. The number of packets received by receiver r is:

$$s_r(p) = (k + \alpha k) * (1 - p(r))$$

If s_r is less than k, that is, the number of received packets is less than required, then the receiver sends a NACK. Thus the total number of NACK packets in the group is:

$$Nn(p) = \Sigma_{r=1}^{R} (1 - B(k + \alpha k, p(r), \alpha k)),$$

where B(n, p, k) is the cumulative binomial probability.

Let $F(x)$ be the overhead of simultaneously sending x NACK packets, $Nl(x)$ be the number of downlink packet drops due to collision and contention caused by x NACK packets, $T(x)$ be the overhead of sending x downlink packets, and $Or(x,y)$ be the total overhead of x NACK packets when requesting y retransmissions. Assume that both α and p are fixed, we can express $Or(x,y)$ as:

$$Or(x,y) = F(x) + T(y + \alpha y) + Or(Nn(p'), \max(k - s_r(p')) * (1 + \alpha))$$

where $p' = p + Nl(x) * (1 - p) / (k + \alpha k)$

Repair packets can also be lost, of course, and such losses will produce more NACKs. Therefore the real overhead is larger. Assume that the additional overhead is a function of the receiver number R, proactive rate α and average loss rate p. The total reverse traffic overhead can be expressed as:

$$Or = Or(Nn(p), max(k - s_r(p)) + h(R, \alpha, p)$$

On the other hand, the overhead of proactively sending packets is straightforward:

$$Op = T(\alpha k)$$

Therefore the total overhead is $O = Or + Op$. When we send fewer proactive parity packets, Op is lower, but Or is likely higher. An optimal algorithm should minimize the total overhead O, specifically, find:

$$\alpha_0 = argmin(O) = argmin(Or(Nn(p), max(k - s_r(p)) + h(R, \alpha, p) + T(\alpha k)) .$$

Let $O(x)$ be the total overhead when the proactive rate α equals to x, we can express the maximum performance gain of the proactive approach over non-proactive or pure-reactive approach as:

$$G = O(0) - O(\alpha_0)$$

As shown in section 3.2, it is possible to obtain an approximation for function F and Nl. On the other hand, T depends only on the packet size and the sending rate. Given $p(r)$, k, and h, we can find the optimal α and maximum gain G through calculation. However, loss rate p is not fixed in WLAN, and the computation is too complicated for an on-the-

fly adaptation. We adopt a simpler approach in our protocol, which dynamically adjusts the proactive rate. This mechanism will be described in the next chapter.

Local NACK Suppression. Local suppression reduces the impact of reverse traffic in two ways. First, it reduces the total number of NACK packets. Second, it reduces the NACK density, that is, the number of NACK packets transmitted during a certain time interval. In this approach, a receiver that requires retransmissions does not transmit NACK packets immediately. Instead, it waits for a random time before sending NACKs. If some repair packets it needs arrive during the waiting time, the receiver adjusts the number of required packets. It cancels the NACK if enough repair packets have been received and no more retransmissions are needed. Otherwise, it transmits the NACK as scheduled. This approach works because the waiting time is random. While a receiver is in waiting, another receiver might have reached its timeout and transmitted a NACK, triggering the sender to transmit the corresponding repair packets. Those repair packets could reach the waiting receiver before its timeout. If the number of received repair packets is large enough to recover the lost packets, the waiting receiver does not need to transmit its NACK. In this case, a NACK transmission is saved. If there are many NACK-sending receivers, the savings could be substantial provided that the waiting time at each receiver is appropriately set. Moreover, even if most receivers still need to send NACKs after waiting, the reverse traffic is spread out into a larger time window. Therefore the NACK density and the related losses are reduced.

We use simulation to study the effect of local NACK suppression. In the following test, we assume that the number of lost packets is uniformly distributed among NACK-

transmitting receivers from 1 to the group size k. The suppression window is set to 100 milliseconds. The data packet size is 1400 bytes, the NACK packet size is 36 bytes, the group size k is 20 and the sending rate is fixed at 6Mbps. The sender transmits 1000 packets. We plot the average number of NACKs sent by each station in Figuire 3.13.

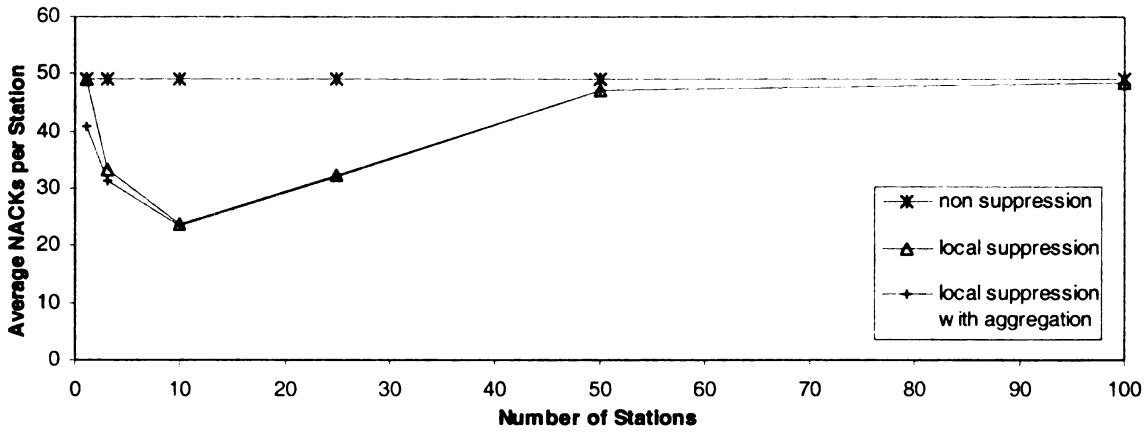


Figure 3.13: The effect of local NACK suppression.

As shown in the figure, the average number of NACKs is indeed reduced by local suppression. However, the suppression does not work well in larger networks. The reason is the contention. Repair packets usually arrive long after sending the corresponding NACK was sent, due to severe channel contention in large networks. Therefore most receivers are still in need of repair packets after waiting, which leads to suppression failures. In some cases, receivers can aggregate retransmission requests in multiple groups and put them into one NACK packet. This usually happens after a long wait. As shown in the figure, the aggregation approach exhibits a little bit better performance. To increase the chance of NACK suppression, we can also set a biased timeout value for each receiver, so that the more demanding receivers (those needing more repair packets) will send their NACKs earlier. The number of corresponding parity packets triggered by

the first NACK is likely to be large enough to recover losses at all receivers. If they are received in time, most NACK transmissions could be suppressed. Based on this strategy, we set the suppression window size at a receiver to

$$W_b = (k - L) / k * 2W$$

where L is the number of required parity packets, and W is the original suppression window size. Obviously, the expected value of W_b equals W, given L is uniformly distributed. We plot the result of this enhanced approach in Figure 3.14, which shows that the biased timeout approach can further reduce the number of NACKs.

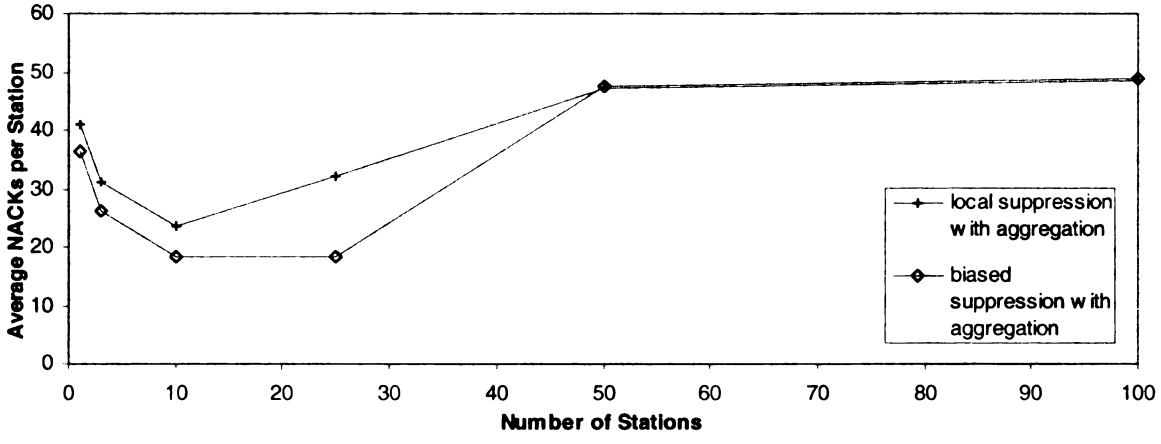


Figure 3.14: The effect of local NACK suppression with biased timeout value.

Simulation results indicate that larger suppression window leads to better suppression performance. We can expect that the average number of NACKs will further drop if the suppression window is set to larger than 100 milliseconds in the above test. However, a long recovery delay is unacceptable for real-time applications. Even for time-insensitive applications such as file distribution, the postponed NACKs could affect the subsequent data transmission. Moreover, a large window demands larger buffer space at the sender,

and increases the total transmission latency if the delay is too long when the transmission is near the end of the resource. Therefore, how to set the suppression window size is a trade-off. It is possible to design an adaptive approach.

To demonstrate the NACK-spread effect, we programmed all the receivers to send NACKs after a random delay without suppression. We compare the number of collisions and drops in this approach with a non-delay approach, using the same number of NACK-transmitting receivers. From Figure 3.15 we see that local NACK suppression can reduce packet drops even if no NACK is actually suppressed, due to the spread effect. However, the collision rate is substantially increased. In non-delay approach, all receivers send NACKs simultaneously, which leads to severe contention and queueing losses. On the other hand, it is likely that the NACK packets will only collide with a small number of data packets at the beginning of the next group, which makes the collision rate low. In local NACK suppression, the NACKs are spread out. The contention degree and the queueing losses are reduced. However, the spread-out NACK packets will collide with more data packets. As a result, the collision rate becomes higher. Overall, the gain in queueing loss reduction is usually larger than the overhead due to collision loss, at least in a moderately large network. Therefore in most cases NACK-spread with no suppression is also beneficial to a reliable multicast protocol.

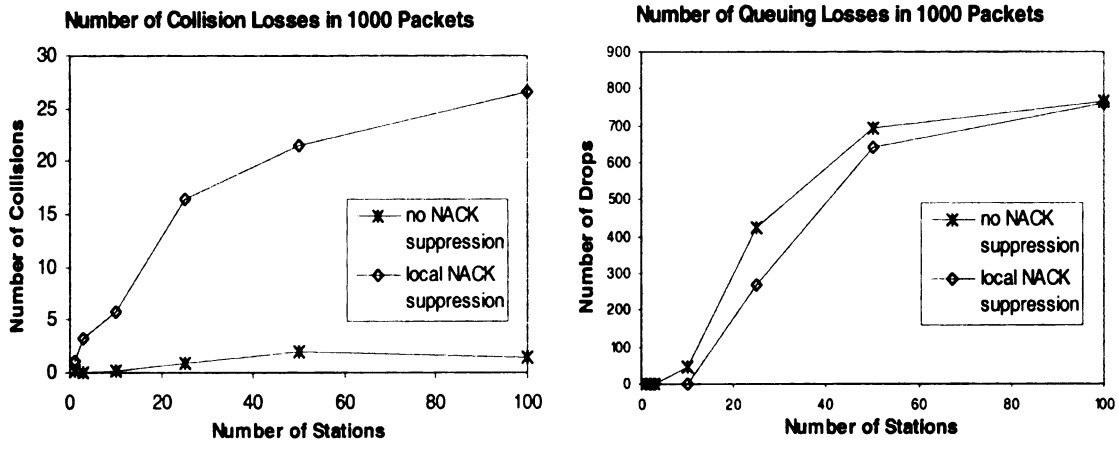


Figure 3.15: The impact of local NACK suppression on collisions and drops.

Global NACK Suppression. Global suppression is more aggressive than local suppression. In this approach, a receiver multicasts NACK packets, so that other receivers as well as the sender can receive it. If a NACK-sending receiver receives a NACK packet from another station, it checks the NACK packet to see whether its own NACK is subsumed. If it is, the receiver postpones sending its NACK, expecting that the sender will respond to the other NACK by multicasting the needed repair packets. In case that the repair packets arrive before the new timeout, the receiver cancels its NACK so that one NACK transmission is saved. Otherwise, it transmits the NACK as usual. The advantage of this approach over the local suppression approach is that a receiver knows sooner whether it should suppress its NACK. In large networks the repair packets might arrive long after the NACK is sent due to heavy contention. As a result, local suppression approach does not work well in this case. On the other hand, multicast NACK packets will certainly arrive earlier than the repair packets. Therefore receivers can further

postpone sending the NACKs, and the probability of NACK suppression is higher than that in a local suppression approach.

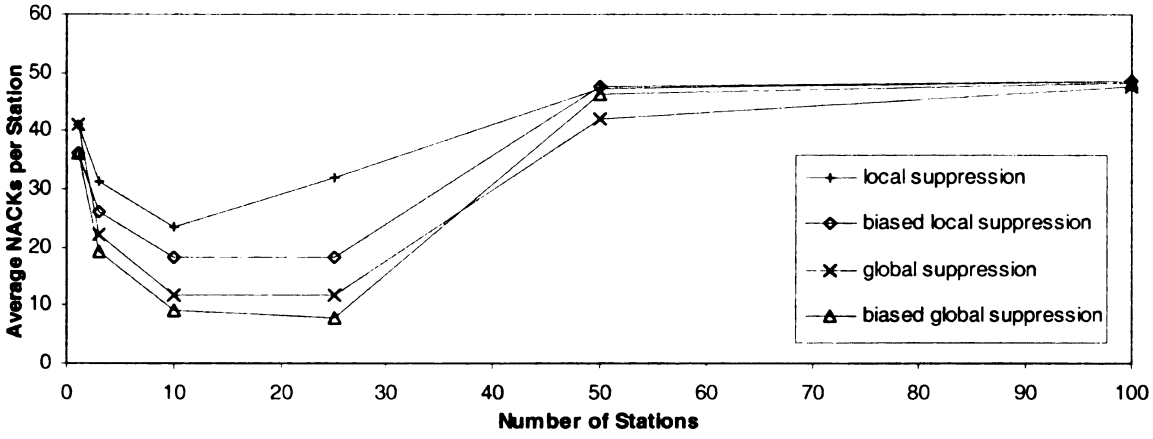


Figure 3.16: Comparing local and global NACK suppressions.

Figure 3.16 shows that global NACK suppression can further reduce the number of NACKs, compared to local suppression. The biased timeout approach is also applicable to global NACK suppression. However, this advantage is not without cost. As shown in Section 3.1, multicast reverse traffic has a more adverse effect on the data traffic in WLANs than does unicast feedback traffic, since the access point has to forward uplink multicast packets both uplink to the wired network and downlink back to the wireless network. Those backward transmissions occupy buffer space and slow down service rate of data packet processing, causing more packet drops due to buffer overflow. We compare the number of packet drops in local and global suppression approaches through simulation. In the following test, receivers transmit NACKs by either unicasting or multicasting after a random delay, regardless of whether or not the NACKs are actually needed. The suppression window size is 100 milliseconds. The sender transmits 1000

data packets. We plot the number of data packet drops of each approach in Figure 3.17. The performance of a non-suppression approach is also shown.

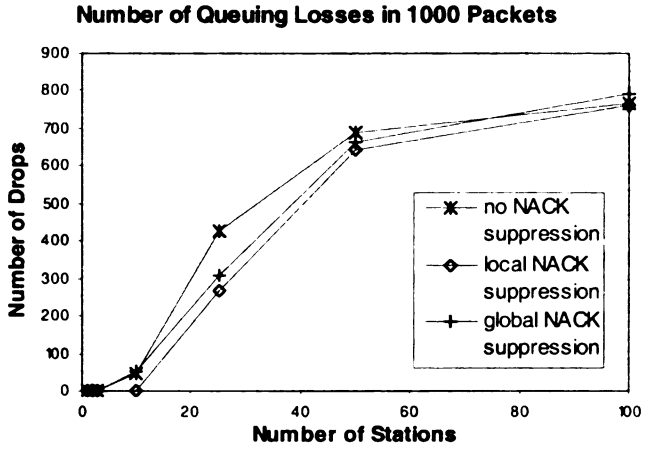


Figure 3.17: Packet drops in local and global suppressions.

The figure illustrates that global suppression causes more queueing losses than local suppression does. However, even without considering NACK reduction, the global suppression approach still outperforms the non-suppression approach due to the NACK-spread effect. Taking into consideration that more NACKs will be suppressed in this approach, it is likely that the gain of suppression is larger than the overhead of extra drops. To compare the overall performance of the NACK suppression approaches, we plot the average loss rate at the access point in Figure 3.18. Both collisions and queueing losses are taken into consideration. Parity packets are counted in as well as data packets.

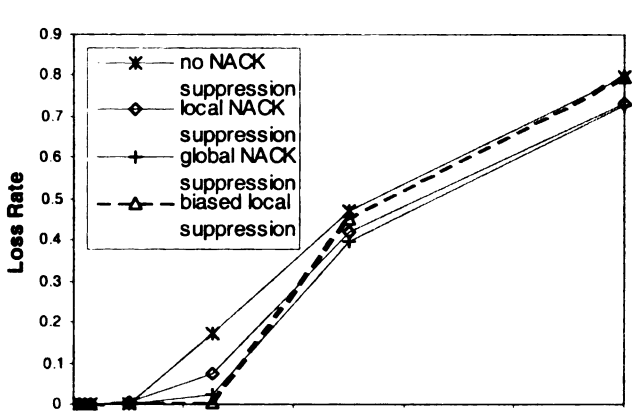


Figure 3.18: Overall performance of NACK suppression approaches.

It is clear that the NACK suppression approaches are capable of reducing packet losses at the access point. The biased local suppression approach exhibits the best performance when the multicast group size is not very large. In larger networks, its performance drops, probably due to the increased collision loss as a result of larger suppression window size variance. On the other hand, the global suppression approach has good overall performance. We point out that, in real situations, it is possible that this approach is not as effective for NACK suppression as in the simulations, especially in medium-sized networks where the simulated suppression has the best effect. In that case, the gain might be unable to offset the overhead of extra drops. Therefore, it might be better to adopt the biased local suppression approach in small or medium-sized networks, while using global NACK suppression in larger networks. In general, both local and global suppression approaches are not sufficient to limit the number of NACKs in large multicast groups in WLANs. The rates of collisions and queueing losses are still too high in that case. A proactive FEC approach is more flexible and effective for NACK suppression in such an environment.

3.4 Summary

In this chapter we studied the characteristics of wireless channels. We confirmed that the wireless propagation loss is bursty and dynamic. The distance between a wireless station and the access point has significant impact on the loss rate, and losses at multiple stations are somehow correlated. Experimental and simulation results indicate that the reverse traffic will cause packet loss due to collisions and contention. Moreover, multicast reverse traffic affects transmission performance more adversely than does unicast reverse traffic.

Several reliable multicast component methods were presented and evaluated in this chapter. We illustrated that a NACK-based protocol with multicast retransmission and FEC encoding should exhibit reasonable performance in WLANs. NACK suppression approaches such as proactive FEC, local NACK suppression and global NACK suppression could further improve performance.

Chapter 4

AFRM Protocol Description

The Adaptive FEC-based Reliable Multicast (AFRM) protocol is designed to be scalable and adaptive. It is well suited to, though not limited to, WLAN environments, where packet losses are relatively high and dynamic. AFRM uses NACK-based error control in order to improve scalability. The responsibility of loss detection is distributed to all stations in the multicast group. Moreover, the sender maintains no membership information. The group is open to any station that is capable of receiving the multicast data packets. Further, the data packets are organized into groups at the sender, and an FEC encoding algorithm is used to generate parity packets. In the case of packet loss, one or more receivers will send retransmission requests. The sender responds to the requests by multicasting a number of parity packets that satisfy the most demanding receiver. Other receivers can utilize those parity packets as well to recover all their data losses. Therefore the total number of retransmissions is reduced.

Although a NACK-based approach produces far fewer feedback packets than does an ACK-based approach, the reverse traffic could still be heavy and significantly affect performance, in cases where the loss rate is high and the multicast group is large. To reduce reverse traffic, AFRM adopts a proactive FEC approach. Since sending excessive proactive parity packets wastes channel bandwidth, the sender adapts the proactive rate

according to the perceived channel conditions. AFRM also adaptively applies the local and global NACK suppression approaches to further improve performance.

In this chapter, we describe the AFRM protocol in detail. First we give an overview of the protocol in Section 4.1. In Section 4.2, we present the protocol strategies and tradeoff considerations. Lastly, we address some implementation issues in Section 4.3.

4.1 Protocol Overview

The AFRM protocol consists of two components: a sender and a receiver. The sender component is required for transmitting packets, while the receiver component is responsible for receiving. Both components are needed at each participating host to support duplex communication. In the following description, we do not distinguish a sender station from the AFRM sender component; neither do we distinguish a receiver station from the AFRM receiver component, unless the omission will cause confusion. Both the sender and receiver components run on top of a best effort multicast service, such as UDP/IP multicast.

In a reliable multicast transmission, a sending application makes a call to the interface function of the AFRM sender, passing down a data block. The sender then fragments the block into packets, prefixing each packet with an AFRM header. In the header, every packet is assigned a sequence number. Those packets are then organized as groups, with group ID and group sequence number added in the header. Next, the sender applies the FEC encoding algorithm to each group, generating FEC parity packets and storing them in a retransmission queue. The sender then multicasts the data packets group by group to

all receivers. After all data packets in a group having been transmitted, the sender checks the current channel condition, based on feedback from earlier transmissions. If the channel is in bad or lossy state, the sender retrieves some parity packets from the retransmission queue and transmits them to the multicast group. If the channel is in a good state, the sender sends fewer or no parity packets. Subsequently, the sender starts processing data packets from the next group and continues transmitting until all data packets are sent.

At a receiver station, the receiving application calls the interface function of the AFRM receiver to receive data. The application will be blocked until the specified number of bytes is received, or the end of a resource is reached. The AFRM receiver listens on a service port that corresponds to a multicast group. When a packet arrives, the receiver checks if the packet has already been received. If so, the packet is discarded. Otherwise, if it is a data packet or a parity packet in the current group, it is inserted into the Data Queue. After all data packets in a group are received, the receiver strips off the packet headers, concatenating the packets into a memory block, and delivering a specified number of bytes in the block to the receiving application. If some data packets are missing, but sufficient parity packets have been received, the receiver applies the FEC decoding algorithm to the packets and generates all the data packets in the group. If the received parity packets are not enough to recover the losses, the receiver schedules a NACK transmission, specifying the group ID and the number of parity packets required. Unless cancelled by one of the NACK suppression methods, the NACK packet is sent out eventually. The receiver continues executing in this manner.

Upon receiving a NACK packet, the sender searches the retransmission queue for parity packets in the specified packet group. Before transmitting those packets, it updates the estimate of the channel condition according to the number of required parity packets in the NACK. The more packet losses, the worse the channel condition is assumed to be. The sender then transmits those parity packets by multicasting. In the case that multiple NACKs for the same group are received within a short time interval, the sender computes a minimum number that is greater than or equal to the number of required parity packets in any NACK. This number is then taken as the number of parity packets needed to be sent. Since some of these parity packets could also be lost, the sender actually schedules more parity packets for transmission. The number of the extra parity packets is calculated according to the current channel condition. After processing the NACK, the sender resumes its normal operation.

Since a NACK packet can also be lost on its way to the sender, a receiver always prepares to retransmit a NACK. It stores each NACK packet in a NACK queue. After a NACK is transmitted, the receiver sets a timer and places the NACK back in the queue. After a timeout, the receiver retrieves the NACK from the queue, checking whether or not any parity packet in that group has recently arrived. If so, and the parity packets are enough to recover the losses, the NACK packet is destroyed since it is no longer needed. If some parity packets have arrived, but the number is insufficient to correct the losses, then the receiver updates the information in the NACK, resets the timer using the same timeout value, and places the NACK back into the queue. If no packet in that group has arrived, the receiver simply increases the timeout value and puts the NACK packet back.

Receivers do not explicitly know whether or not the transmission of the current group is completed, until they receive a packet from a subsequent group. If some packets in the last group are lost, receivers might keep waiting for more packets without sending NACKs. To prevent this situation from occurring, the sender periodically transmits short "keepalive" packets after all data packets have been transmitted. A "keepalive" packet contains the ID of the last group. Upon receiving a "keepalive" packet, receivers compare the ID in the packet with the ID of the last group they have received. In this way they can determine whether or not a packet loss has occurred.

The sender needs to set an appropriate sending rate. To maximize throughput, the sending rate should be set as high as possible. However, as we saw in Chapter 3, a high sending rate will lead to congestion and queueing losses. Therefore, the sender periodically adjusts the rate according to the feedback information it has received.

Several parameters in the AFRM protocol significantly affect the performance of the protocol. Examples include proactive rate, NACK suppression timeout value, sending rate, etc. We evaluate these parameters in the next section.

4.2 Protocol Algorithms

4.2.1 Proactive Rate

How to adapt the proactive rate is a key part of the AFRM protocol. The basic idea is to send more proactive parity packets when the channel is in a poor condition, and fewer when it is in a good condition. The extra packets are expected to compensate for the respective loss rate, so that most receivers will be able to correct their losses without

contacting the sender. However, sending excessive proactive parity packets will waste channel bandwidth and hinder performance. The optimal value of the proactive rate is dependent on the loss rate and the multicast group size. Since the loss rate is dynamic and hard to predict in WLANs, it is almost impossible to always find the optimal value. In the AFRM protocol we use an approximation that is close to the optimal value.

We are aware that the wireless loss locality depends on the packet group size and the error burst length distribution. According to the collected traces, most loss bursts are very short, only 1 or 2 packets long. On the contrary, the error-free bursts are usually longer than several hundred packets. Therefore, the next group is more likely to be error free if there is a loss in the current group. In a large network, however, the random variations in packet loss tend to complement with each other. Hence, the overall packet loss in the multicast group becomes less bursty and more predictable. Therefore it is necessary to consider both the burst length distribution and the network when dynamically setting the proactive rate.

Since it is difficult to make accurate predictions, we need to examine the consequence of an incorrect prediction. For an over-prediction, the gain of the proactive FEC approach is negative due to wasted bandwidth. On the other hand, an under-prediction introduces no extra overhead, but also offers little performance improvement, since many receivers will send NACKs to correct losses. In a large network or in a high loss environment, the overhead of an over-prediction is likely to be less than the penalty of an under-prediction. Therefore, we should be conservative in setting the proactive rate in a low-loss or small-scale network, and we should be more aggressive otherwise.

To appropriately set the proactive rate, the sender needs to estimate the channel condition and the multicast group size. However, an AFRM sender has no direct knowledge of the multicast group size since it does not maintain the group membership information. On the other hand, the sender might be located in a wired network, where it cannot directly measure the wireless channel conditions. The only way to obtain this information is to deduce it from receiver feedback.

For example, the sender can check the source of the NACK packets to estimate the multicast group size. In the case of the channel condition, a natural is to have each receiver calculate the loss rate that it perceives, and periodically send this information to the sender. This approach can lead to accurate channel estimation, but it imposes extra computational load on receivers as well as extra communication load on the network. Since both the computational power of a mobile station and the channel bandwidth of a wireless network are limited resources, this approach might be inappropriate. A revision is to trade the estimation accuracy with the computational and communication overhead. The receivers control what, when and how to send information to the sender. A simple and straightforward implementation is to directly utilize NACKs. From received NACKs, the sender can determine how many packets were lost and which nodes lost packets. Based on this information the sender can make a reasonable estimation of the multicast group size and the current loss rate.

Sometimes a receiver sends no NACK at all, even the channel is in lossy condition, provided that it has received enough proactive parity packets to correct losses. In this case the sender will assume that the channel is in a good state and decrease the proactive rate. The decreasing process is "blind" since there is no feedback. If the proactive rate is

decreased too fast, the receivers will soon receive fewer parity packets than they need and start sending NACKs. If it is too slow, on the other hand, the channel bandwidth will be wasted since many proactive packets are actually unneeded. McKinley and Mani [39] explored different functions for increasing and decreasing the proactive rate, but in separate instances of an adaptive FEC protocol. In the AFRM protocol, we take into account the history of the average loss rate and the multicast group size, enabling us to adapt the rate of change accordingly. If the loss rate is high and the group size is large, we decrease the proactive rate slowly. Otherwise, we decrease the rate faster.

There is one major potential problem in this approach, however. We know that packet losses are not necessarily caused by propagation errors -- congestion and collisions also contribute to packet losses. In this case, sending more proactive packets will only exacerbate the problem. Instead, the sender should reduce its rate of flow. We address this problem in detail in Section 4.2.3, which describes AFRM flow control.

Using NACKs to estimate the channel conditions and adapt the proactive rate involves very little overhead. We designed two algorithms, A1 and A2, which differ in how the proactive rate is adapted. Algorithm A1 aims to suppress as many NACKs as possible by aggressively setting the proactive rate -- it adjusts the proactive rate whenever a NACK is received. On the other hand, algorithm A2 is based on the average loss rate and is more conservative. We provide pseudo-code for these two algorithms in Figure 4.1 and Figure 4.2, respectively.

Algorithm A1

```
a = a_ini          // a is proactive rate
w = 0              // w is the number of group processed after last proactive rate adaptation

[NACK_Receiver]
for each received NACK packet p
        for each group g in p
                m = (number of requested parity packets in g of p)
                n = m – (number of recently retransmitted parity packet in group g)
                if m = k
                        a = a_max
                else
                        a = min(a_max, max(a, (k*a(g)+m)/(k-m)))        // k is the number of
data packet in a group, a(g) is the proactive rate used to transmit data packets in g, a_max is
maximum proactive rate
                send n*(1+a) parity packets

[Data_transmitter]
for each data group g
        send k data packets
        send k*a proactive parity packets
        a[g] = a; w = w + 1

[Update_thread]
do
        if w = W        // W is a threshold for proactive rate decrease
                if no NACK packets received
                        a = max(a_min, a/2)          // a_min is minimum proactive rate
                w = 0
        sleep t seconds
repeat
```

Figure 4.1: AFRM-A1 algorithm.

In algorithm A1, the proactive rate is calculated based on the recent loss rate. If only $(k - m)$ packets are received among $k * (1 + a(g))$ packets transmitted, the success rate is

$$(k - m) / (k * (1 + a(g)))$$

Let us assume that the channel condition does not change during the time of the retransmission. To guarantee that the reactive parity packets are enough to recover the losses, the following inequality must hold:

$$(k - m) / (k * (1 + a(g))) * (1 + a) \geq 1$$

Therefore the proactive rate is set to $(k * a(g) + m) / (k - m)$ to avoid feedback traffic. This is a fairly aggressive algorithm that aims to minimize the reverse traffic. The algorithm should exhibit good performance in high loss environments or in large multicast groups, where the overall loss rate is more stable and a lack of parity packets results in a higher penalty. When the loss rate is low and the multicast group is small, however, it is likely that this algorithm will produce too many useless parity packets, due to the burstiness of wireless losses, the short error burst length, and the large error-free burst length in those environments.

Algorithm A2

```
a = a_ini          // a is proactive rate
w = 0              // w is the revised average error burst length
m = 0              // m is the revised average loss rate in an error burst
lb[] = 0           // lb is an array of packet loss bitmap in recent groups
na = 0;            // na is number of NACK packet received in recent groups

[NACK_receiver]
for each received NACK packet p
        na = na + 1;
        for each group g in p
                update lb[g]
                n = [number of requested parity packet in g of p] – [number of recently
retransmitted parity packet in group g] ;
                if currentgroup - g < w
                        a = m
                else
                        a = 0           // do not send proactive packets
                a = max(a_min, min(a_max, a)
                send n*(1+a) parity packets

[Data_transmitter]
for each data group
        send k data packets
        if currentgroup - lastnackgroup < w
                a = m
        else
                a = 0           // do not send proactive packets
        a = max(a_min, min(a_max, a)
        send k*a proactive parity packets

[Update_thread]
do
        calculate w and m from lb[] and na
        na = 0
        sleep t seconds
repeat
```

Figure 4.2: AFRM-A2 algorithm.

Algorithm A2 is more conservative than A1. It uses a conditional adaptation, whereby a receiver aggregates loss information in the current group and several previous groups into a loss bitmap. Only unreported losses in previous groups are included. (They were not reported in earlier NACKs because the proactive parity packets received in those groups were sufficient to recover the losses locally, or because the NACKs were suppressed.) The receiver attaches the loss bitmap to the current NACK. By combining all the loss bitmaps from multiple receivers, the sender estimates the average of overall error burst length and the average loss rate. If the packet distance between the loss group and the current group is less than the average error burst length, the sender transmits some proactive parity packets with high probability according to the loss rate. Otherwise the sender transmits proactive parity packets with lower probability. To account for the effect of the number of receivers and the actual number of NACK packets, we add a factor to the proactive rate (Empirically the factor is set to $2 * N / R$, where N is the number of NACKs and R is the estimated number of receivers). The underlying observation is, when there are more NACK packets, the chance of an incorrect prediction becomes smaller. In this case, we are able to afford sending more proactive packets.

Algorithm A2 is conservative in the sense that it does not aim to suppress all NACKs, but just some of them. Since the overall error burst is likely to be short in small multicast groups or when the loss rate is low, this algorithm will generate fewer proactive packets than does A1 in those environments. In larger groups or when the loss rate is high, more proactive parity packets will be sent. However, the proactive rate in this case should still be smaller than that of algorithm A1.

4.2.2 NACK Suppression

In chapter 3 we demonstrated how reverse traffic could adversely affect the performance of data transmissions. NACK suppression approaches are intended to reduce the intensity of reverse traffic. Besides proactive FEC approach, we also studied the performance of local and global NACK suppressions in a simulation environment. Although neither is very effective in large-scale networks, they incur lower average overhead than does the proactive FEC approach, whose performance largely depends on the accuracy of channel estimation. We use both local and global NACK suppressions in AFRM to improve performance.

As described in Chapter 3, local NACK suppression can reduce reverse traffic and improve performance. The biased timer approach exhibits better performance provided that the multicast group is not too large. We set up a heuristic threshold value for the multicast group size. In case the estimated size is below the threshold, the biased local suppression approach is utilized. Empirically the threshold is set to 30.

Global NACK suppression also improves performance. However, it also increases the chance of queueing losses. The overall performance depends on the suppression efficiency. It is likely that the gain will offset the overhead in large multicast groups. We activate the global NACK suppression approach when the estimated group size surpasses the threshold value. The sender sets a flag in the AFRM header of each data or parity packet. Receivers check the flag to choose which NACK suppression approach to use.

One shortcoming of both local and global NACK suppression approaches is the resulting long latency between the sending of a packet group and the receiving of feedback packets for that group. This delay will affect the accuracy of channel estimation

at the sender. In large multicast groups, however, packet losses are less bursty. Therefore channel estimation is mainly determined by the average loss rate, which is affected little by the promptness of feedback. Moreover, the expected waiting time for the earliest NACK is shorter in larger multicast groups (This is reflected in the formula $E(T) = W / (n + 1)$, where T is the waiting time of the earliest NACK, W is the suppression window size, n is the number of NACK-sending stations). Furthermore, the suppression gain is likely to be substantial in these environments. On the other hand, those approaches are unlikely to be beneficial to the overall performance in small groups. To solve this problem, we set another threshold value for group size. If the estimated group size is below this threshold, both NACK suppression approaches are disabled. We set the threshold value to 10 empirically.

Another related problem is the long completion time for each packet group. The repair packets might arrive long after the reception of the data packets in the same group, due to the random delay in sending NACKs, as well as round trip latency. In most cases the waiting overlaps with the receiving of the data packets in the next group, so it is not a large problem. However, the waiting in the last packet group of a resource will increase the transmission latency. Since there is no more data traffic, NACK suppression becomes less meaningful. Our solution is to disable NACK suppression in the last one or n packet groups. The value of n depends on the suppression window size and the per-group transmission latency.

4.2.3 Flow Control

Flow control in wireless networks is a challenge. A major problem is how to differentiate queueing losses from wireless propagation losses. Several approaches have been proposed for TCP flow control in wireless environments, including ECN (Explicit Congestion Notification) [4] at the link/network layer and slow start threshold comparison at the transport layer [5]. The ECN approach is also applicable to the AFRM protocol, if a standard cross layer interface is provided. We can also design algorithms such as the slow start threshold comparison approach to differentiate losses. For example, if NACK packets continue to arrive at the sender even if the sending rate has been significantly reduced for a while, those losses are more likely the result of wireless propagation errors. On the other hand, all receivers will experience the same loss if it is caused by congestion at the access point, since the packet is not propagated in the air at all. Such common losses are unlikely for propagation losses. Our approach is to have the receivers include an explicit packet loss bitmap in NACK packets. For small packet groups, the bitmap only requires one 32-bit word per group, an affordable overhead. By comparing the bitmap words in NACKs from different receivers, the sender can differentiate propagation losses from other losses. If the bitmap words indicate that a packet is lost at all the receivers, it is likely that the packet was dropped at the access point due to a collision or congestion. Otherwise, it is likely to be a propagation loss.

As mentioned in Chapter 3, we do not consider buffer overrun at the receivers. Therefore receivers send no explicit rate control requests to the sender [69]. In the AFRM protocol, flow control is applied mainly for congestion avoidance at the access point. We use a rate based flow control approach in this protocol. We set up three threshold values

to separate four phases. Similar to TCP's slow start and congestion avoidance phase, the rate increasing/decreasing speed varies in different phases. The sending rate increases if no NACK is received. Otherwise, the sending rate will decrease if the number of NACKs reaches a threshold, relative to the number of transmitted packets. The pseudo code of the flow control algorithm is provided in Figure 4.3.

Flow Control Algorithm

```
if packetsSent > 2 and nacksReceived > 1 then
        if nacksReceived < packetsSent / threshold0 then
                // no decreasing
        else if nacksReceived < packetsSent / threshold1 then
                delay = delay * (1 + nacksReceived / packetsSent)
        else if nacksReceived < packetsSent / threshold2 then
                delay = delay * 1.75
        else
                if delay * 2 < maxdelay then
                        thresh = delay * 2
                else
                        thresh = maxdelay
                delay = maxdelay
                cdec = (thresh - mindelay) / 25.0
else if delay >= thresh then
        delay = (delay * (1-multi))
else
        delay = (delay - cdec)

if delay < mindelay then
        delay = mindelay
if delay > maxdelay then
        delay = maxdelay

// insert delay between transmissions
```

Figure 4.3: AFRM flow control algorithm.

In the pseudo code, the values of *threshold0*, *threshold1* and *threshold2* are empirically set to 10, 5, and 2 respectively. For preliminary differentiation, we apply the

aforementioned bitmap approach to eliminate packet losses that are definitely caused by wireless propagation errors from the flow control algorithm. Further optimizations are under consideration.

4.3 Implementation

We implemented the AFRM protocol at the application level. The protocol is built on top of the socket interface, specifically, UDP sockets and IP multicast services. Multicast groups are identified by multicast addresses. Membership management is taken care of by underlying IP multicast services.

Although an application level implementation is perhaps less efficient than a kernel level implementation, the application level approach is more flexible and easier to adapt to varying requirements of upper layer applications. At the application level, a protocol can adopt relatively complicated algorithms and use a large amount of memory without causing problems. In a previous work, we implemented a Java version protocol as a component of the Pavilion middleware framework [40]. Recently an extended C version of this protocol has been developed. We use the C version protocol in our experimental and simulation studies, and the following description is based on this version.

4.3.1 Architecture

Sender. An AFRM sender consists of five major components: Application_Interface, encoder, data_transmitter, feedback_processor and update_thread. Their relationship is depicted in Figure 4.4.

The Application_Interface accepts function calls from the upper level applications. An application passes down a data block or resource name that is to be reliably transmitted. The Application_Interface obtains the data block, fragmenting it into packets that are prefixed with AFRM data packet headers, assigning to each a sequence number, group ID and group sequence number. The Application_Interface then places them into the Data Queue and invokes both the encoder and the Data_Transmitter.

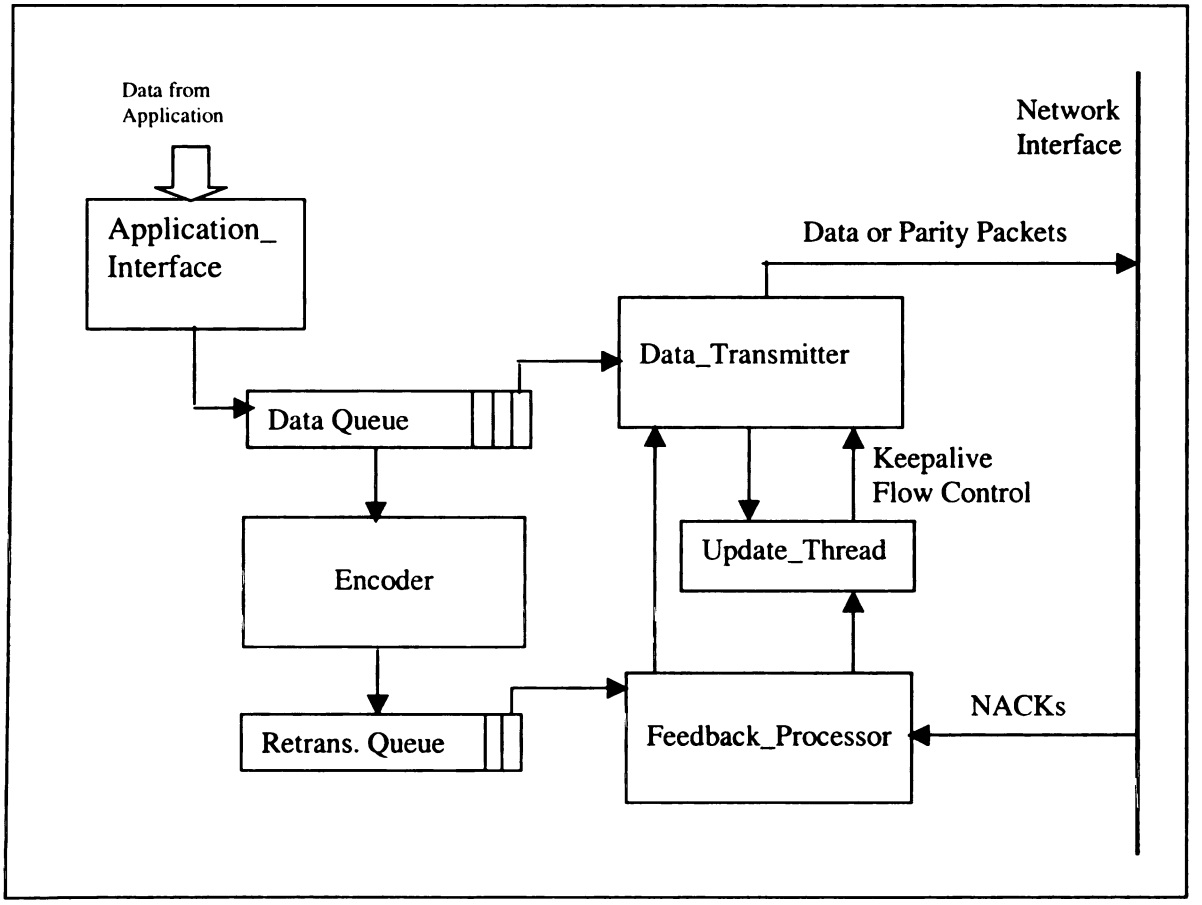


Figure 4.4: The AFRM sender architecture.

The Encoder retrieves data packets from the Data Queue, applying FEC encoding algorithm to each packet group in order to generate parity packets. Each group has the same size n, consisting of k data packets and n-k parity packets. The parity packets are

stored in the Retransmission Queue. To ensure enough packets for decoding, the data packets are also appended to the Retransmission Queue after the parity packets. After encoding one group, the Encoder inserts a special packet into the Data Queue, denoting the end of the group. The Data Queue is implemented as a heap so that all the packets in the queue are sorted according to the group ID and the group sequence number.

The Data_Transmitter retrieves packets from the Data Queue and transmits the packets by multicasting. If it finds a special packet, it searches in the retransmission queue for proactive parity packets. In case the parity packets of the group are not available yet, it suspends until the encoder completes generating those packets. Then it fetches parity packets from the queue and transmits them. The number of those proactive packets depends on the current proactive rate. If a NACK packet is received, the number of requested packets is increased by a times, where a is the current proactive rate. The Data_Transmitter fetches that amount of parity packets from the retransmission queue and transmits them. When a parity packet is pulled out from the head of the Retransmission Queue and transmitted, it is also inserted at the end of the queue. After a data packet is transmitted, it is removed from the Data Queue. The Data_Transmitter always takes the first packet in the Data Queue to transmit. Since packets in Data Queue are sorted in the group ID and the group sequence number, those with smaller group IDs and smaller group sequence numbers will be transmitted earlier. If the Data Queue becomes empty, the Data_Transmitter keeps sending "keepalive" packets, following a binary exponential backoff. The group ID of the last sent group is assigned to those "keepalive" packets. Between transmitting two packets, the Data_Transmitter always inserts a delay according to the current sending rate.

The Feedback_Processor keeps listening on the feedback port. If a NACK packet arrives, the Feedback_Processor updates the loss statistics, searching in the Suppression Queue to see whether or not any NACKs have already received in the same group. If such NACKs are found, it checks the timestamps of those packets and removes those NACK packets considered to be obsolete. Then it subtracts the total number of requested packets in those NACKs from the number in the new NACK. If the number in the new NACK becomes zero or negative, the NACK is dropped. Otherwise, it is timestamped and inserted into both the Data Queue and the Suppression Queue.

The Update_Thread periodically updates some statistics and calculates the values like the average loss rate. It also calculates the new sending rate from the statistics such as the number of received NACKs and the number of transmitted data or parity packets.

Receiver. An AFRM receiver consists of four major components: Data_Receiver, NACK_Sender, Decoder and Application_Interface. Their relationship is depicted in Figure 4.5.

The Data_Receiver loops, receiving packets from a port corresponding to the multicast group. If the received packet is a parity packet, it is inserted into a Recovery Queue. If the received packet is a data packet, it is inserted into both the Recovery Queue and the Data Queue. If the received packet is a keepalive packet, it is discarded. In all above cases the Data_Receiver checks the group ID in the received packet. If the ID is greater than that of the last received group, or if the packet is a "keepalive" packet, the Data_Receiver will count the data and the parity packets in the last received group. If the total number is less than k, then the Data_Receiver will generate a NACK packet, assigning the number of

needed parity packets. The expected dispatch time of that NACK is also set according to the current NACK suppression algorithm. Next, the NACK is inserted into the NACK Queue, and the NACK_Sender is invoked. If the number of the received *data* packets in the group is equal to k, the Application_Interface is invoked and the data packets are delivered. Otherwise, if the total number of the data and parity packets in the group is greater than or equal to k, then the Decoder is invoked.

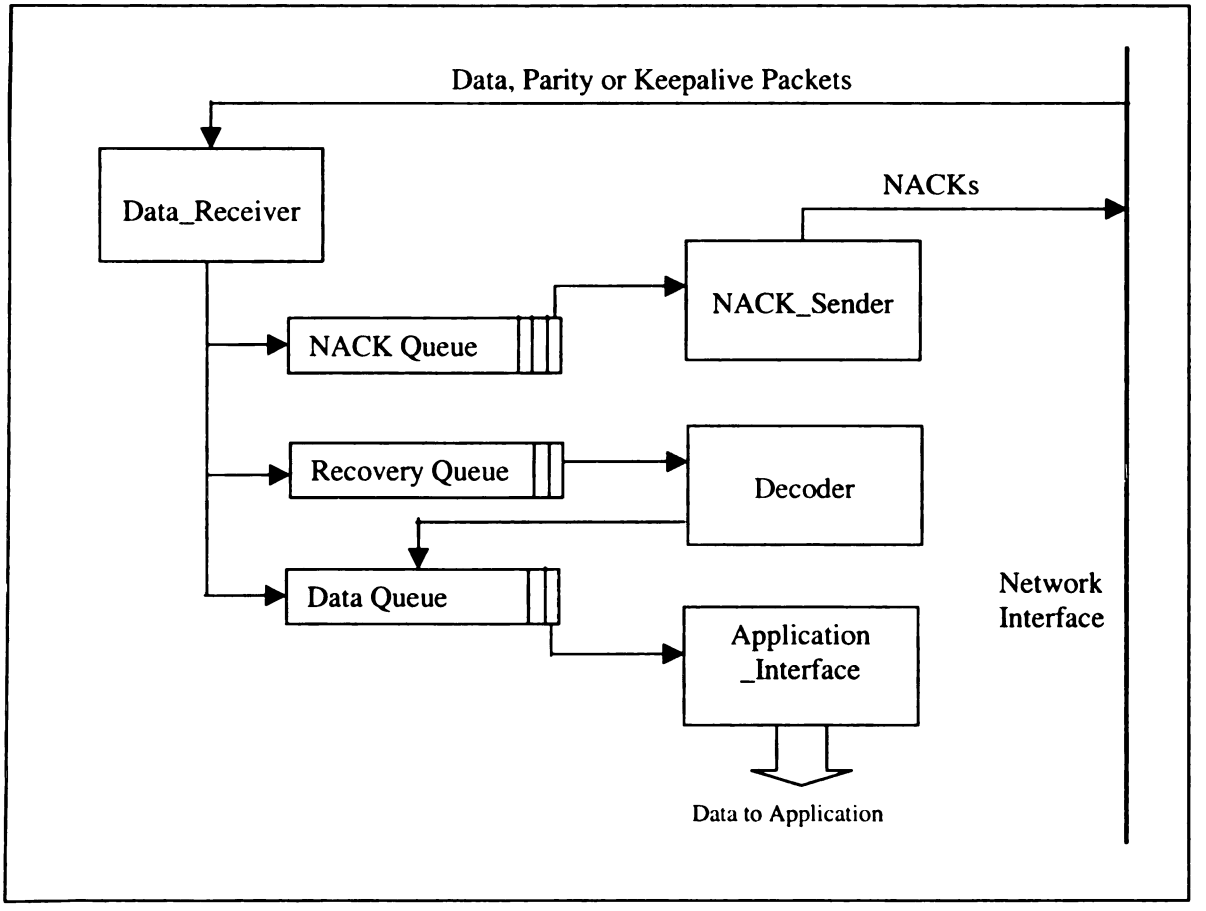


Figure 4.5: The AFRM receiver architecture

The NACK_Sender extracts NACK packets one by one from the NACK Queue. The NACKs in the NACK Queue are sorted according to dispatch time. If the dispatch time of the current NACK is later than the current time, the NACK_Sender suspends itself until

the specified time point is reached. Then the NACK_Sender checks the Data Queue and the Recovery Queue to see whether or not it still needs additional parity packets. If no parity packets are needed now, the current NACK is removed from the NACK Queue and destroyed. If the number of needed parity packets has decreased, the NACK_Sender updates the dispatch time using a linear backoff algorithm and places the updated NACK packet in the NACK queue. Otherwise it transmits the NACK. After transmission, the NACK_Sender updates the dispatch time of the current NACK using exponential backoff and places the new NACK packet in the NACK Queue.

The Decoder fetches packets from the Recovery Queue. If there are at least k packets available in one group, the Decoder applies the FEC decoding algorithm to those packets and generates k data packets. Then the generated data packets are inserted into the Data Queue. If some of those data packets are already in the Data Queue, the new copies are discarded. After decoding, packets in the group are removed from the Recovery Queue and destroyed. The Decoder then invokes the Application_Interface and moves on to the next group.

The Application_Interface accepts function calls from the application. If the application requests a resource or a data block, the Application_Interface removes received data packets from the Data Queue, stripping off the AFRM protocol header and concatenating them into one data block. If the data block is not large enough to satisfy the request, the Application_Interface suspends itself until new data packets are received or generated (by the FEC Decoder). After all the requested data are available, the Application_Interface notifies the application and delivers the data block.

4.3.2 AFRM Packet Format

AFRM stores both packet headers and data bits in the UDP packet payload. There are four kinds of packets: data packet, parity packet, keepalive packet and NACK packet. All those packets share a common header format, although some fields have different meanings in different packets and some packets have extra header fields.

The common header format is shown in Figure 4.6:

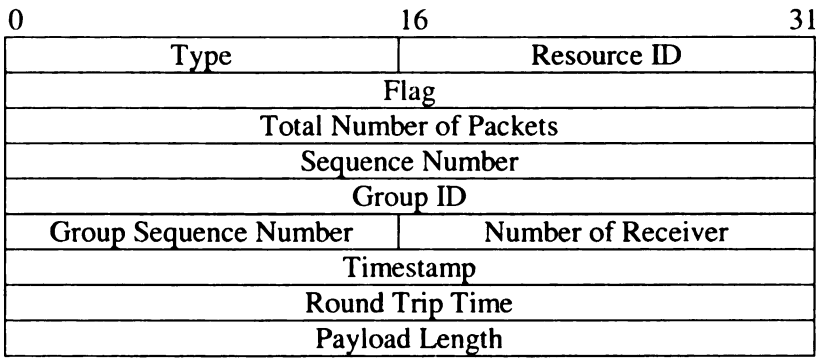


Figure 4.5: The AFRM packet header format.

Type. The 16-bit type field is used to identify the packet type. The possible values are: DATA, PARITY, KEEPALIVE, and NACK.

Resource ID. The resource ID field is a 16-bit unsigned integer that is used to identify the resources transmitted by a particular sender.

Flag. This 32-bit field is used to hold special values, for example, the size of the last packet in a resource in data packets, the NACK suppression switch in data or parity packets, and the packet loss bitmap of the specified group in NACK packets.

Total Number of Packets. This 32-bit field is used to specify the total number of packets in the resource.

Sequence Number. The sequence number field is a 32-bit unsigned integer that is used to identify data packets in the resource. It is assigned –1 for parity packets. In NACK packet, it is the sequence number of the requested packet. If requesting any parity packets, this field is set to –1. In keepalive packets, this field holds the sequence number of the last sent data packet.

Group ID. This is a 32-bit field that is used to identify packet groups in the resource.

Group Sequence Number. This field is a 16-bit unsigned integer. In data and parity packets, it is the packet sequence number in the specified group. In keepalive packets it is set to –1. In NACK packets, it is the number of parity packets required if the sequence number field of the NACK is set to –1.

Number of Receiver. Sender uses this 16-bit field to inform receivers with the estimated multicast group size. Receivers can adapt their NACK-sending behaviors according to this field.

Timestamp. This 32-bit field is used by the sender to timestamp each data or keepalive packet. Receivers will put the timestamp value of the last received data or keepalive packet in NACK packets. This helps the sender to estimate round trip time and set the retransmission suppression window.

Round Trip Time. The sender puts the estimated round trip time in this 32-bit field of each data, parity or keepalive packet. Receivers use this estimation to set NACK retransmission waiting time.

Payload Length. This 32-bit field is used to specify the payload data length in bytes.

Chapter 5

Performance Evaluation

In this chapter we evaluate the performance of the AFRM protocol and compare it with several other reliable multicast protocols. We first study the behavior of the protocol in our experiment environment, where results show it has good performance. Next, we evaluate the protocol in our simulation framework. The framework enables us to vary environment parameters such as the loss characteristics and the number of stations. We test the protocol over a simulated network characterized by various loss patterns, including a uniform model, a bursty model, a synthetic model and real packet traces. The simulated network contains up to 100 wireless stations. In all tests we use throughput as the major performance metric. The results show that the AFRM protocol has better overall performance, in most situations, than other non-proactive and non-adaptive protocols, especially in large multicast groups and in high loss environments.

5.1 Experimental Study

We conducted experimental studies using the testbed described in Chapter 3. The testbed consists of a 100Mbps Ethernet and an 11Mbps Aironet wireless network. The Ethernet is connected to the campus network. The wireless network is set to infrastructure

mode and connected to the Ethernet through an access point. The testbed includes several high-end workstations, each with a 1GHz Pentium III processor and 512MB memory. Some of these workstations are equipped with 11Mbps Aironet 340 Series PCI Wireless Cards and connected to the wireless network, while others are connected to the wired Ethernet. We also use several laptop computers, each with a 1GHz Pentium III processor and 256MB memory. The laptops are connected to the wireless network. All machines in the testbed run Windows 2000 professional version as the operating system. We conducted all our tests during midnights or early mornings when there were no other wireless stations associated with the access point. Therefore in all tests there is minimal other traffic in the wireless network.

5.1.1 Normal Conditions

In this experiment, we study the protocol behavior in "normal" indoor environments. We set up a wired workstation that transmits a 4MB file to a multicast address, on which 1 to 3 wireless receivers are listening. The distance between the wireless stations and the access point varies. For comparison, we implement a non-FEC protocol, a pure-reactive-FEC protocol, and a fixed-proactive-rate-FEC protocol with 5%, 10% and 20% proactive rate. All those protocols are NACK based. In the non-FEC protocol, the sender retransmits the requested data packet upon receiving a NACK. In the pure-reactive protocol, the sender does not send any proactive parity packets and sends only as many parity packets as requested in NACKs. In the fixed-proactive protocol, the proactive rate is specified a priori and applies to both data and (parity packet) retransmissions. In all tests, the data packet size is set to 1400 bytes, including the AFRM header. The group

size parameter k is set to 20, and N is set to 60. The initial sending rate is set to 6Mbps. In the case of NACK suppression, the receiver suppression window size is set to 100 milliseconds. Where applicable, we collected the following data: NACK count, retransmission count, packet loss count, required parity packets count, received parity packets count and proactive rate. We also recorded the total transmission time in order to calculate the throughput.

We noticed that the average loss rate is very low in our indoor environment. From Chapter 3 we see that the loss rate is below 1% in most cases. For the worst station the loss rate is around 8%. In good states there are very few losses. In bad states, however, a station can experience a long burst of losses, which will significantly affect the throughput. We depict such a loss burst from a typical trace in Figure 5.1. The burst is longer than 200 packets.

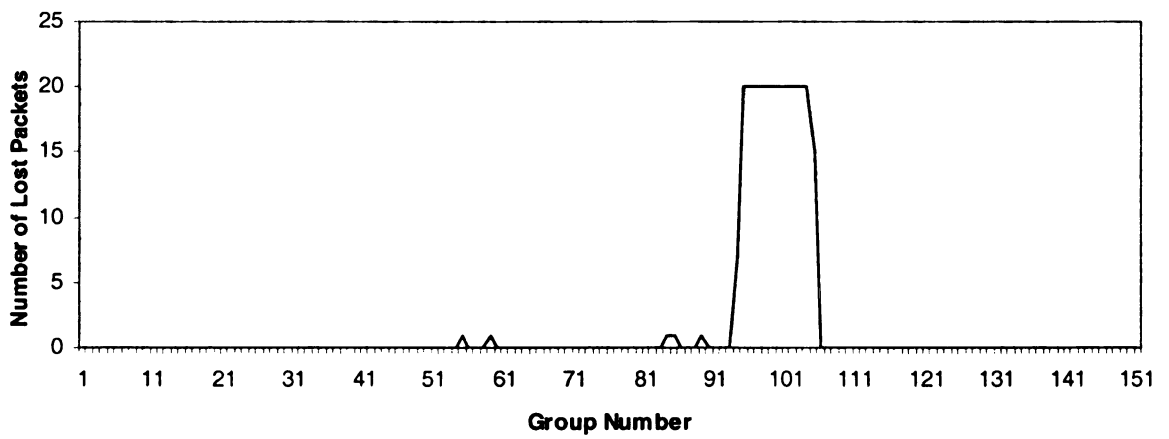


Figure 5.1: A loss burst spans several packet groups.

We compare the throughput of the aforementioned five protocols with the AFRM protocol using two different adaptation algorithms. Since the non-FEC protocol benefits little from NACK suppressions, we turned the suppression flag off in all protocols to

make fair comparison. We conducted each test five times. In calculating throughput, the maximum and minimum values are dropped, and the results are the averages of the remaining values. The throughput is depicted in Figure 5.2.

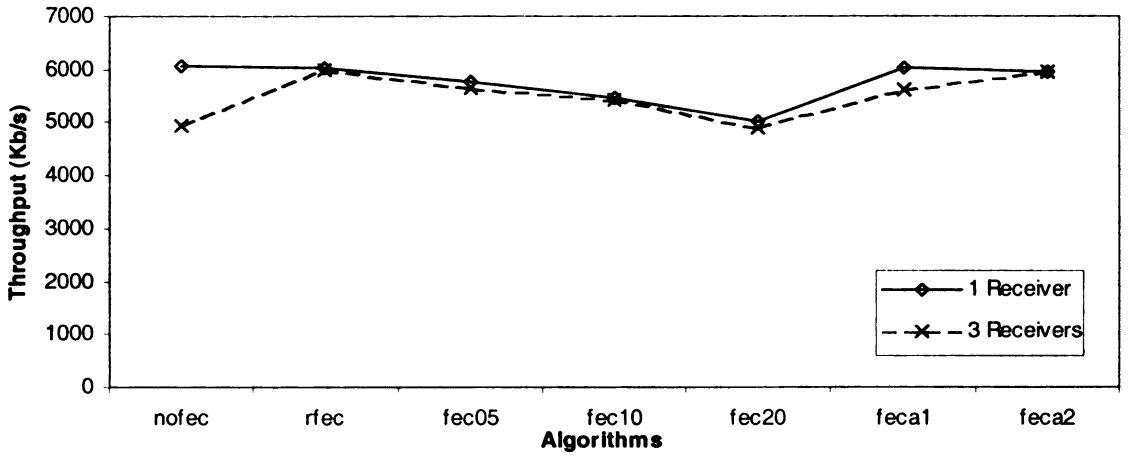


Figure 5.2: Experimental throughput under normal conditions.

In the above figure, "rfec" denotes the pure-reactive-FEC protocol. "feca1" and "feca2" denote the AFRM-A1 and the AFRM-A2 algorithms respectively. The figure shows that in low loss environments all these protocols exhibit fairly good performance. The throughput is as high as 6Mbps, close to the effective limit. When examining the individual protocols, we noticed that the fixed-rate proactive protocols suffer from sending too many unneeded parity packets. On the other hand, the performances of all the protocols drop slightly in the case of 3 receivers, apparently due to the increased reverse traffic.

Wireless losses are dynamic and random, which make it difficult to conduct performance comparisons. This is one of the major reasons that we need a simulation framework. We will describe the simulation approach in section 5.2. On the other hand, the loss rate is significantly higher when a wireless station is farther away from the access

point. If we simulate the higher loss by dropping received packets at a closer station, the perceived loss (simulated loss + real loss) is likely more stationary and suitable for comparisons. Therefore, it makes sense to simulate such losses by using an artificial loss model.

5.1.2 Artificial Loss

In this approach, we modify the receiver code so that it randomly received packets according to a predefined loss rate. Therefore, the loss comprises the controllable artificial loss plus the unpredictable actual loss. For simplicity, the distribution of the artificial loss is initially set to uniform. This is not a particularly accurate model since the real channel loss is not included. However, when we set the artificial loss rate to a large value, say 10%, it is safe to omit the transmission losses, whose rate is usually around 1%. Compared to simulation, the advantage of this approach is that everything except the loss rate is real, including the overheads for communication and computation, as well as congestion and collisions. We use this approach to study the protocol performance in the following tests.

We set the artificial loss rate to 1%, 5%, 10%, 20% respectively and repeat the tests from the previous section. We plot the throughput in Figure 5.3. As shown in the figure, the protocols generate similar throughput when the loss rate is low. The fixed-rate proactive protocols suffer from transmitting excessive proactive packets. When the loss rate is higher, the throughput drops substantially. The proactive FEC protocols become better than the other protocols due to the reduction in NACK packets. The 20% fixed-rate proactive protocol has the best performance in the case of 5% and 10% losses. We can expect that a 30% fixed-rate protocol will achieve higher throughput than the others in

the case of 20% loss. However, a fix-rate protocol is unlikely optimal in all cases in a real wireless LAN since the wireless loss is bursty and dynamic.

The AFRM-A1 protocol has the best overall performance. The AFRM-A2 is the best for low loss conditions, but not as good as some other protocols for heavy losses, due to its conservative policy in sending proactive parity packets. Anyway, both the AFRM protocols outperform others in the case of 20% loss. In multicast groups with 3 receivers, the average performance is generally 5% lower than that for the 1-receiver groups. The performance ranking is basically the same.

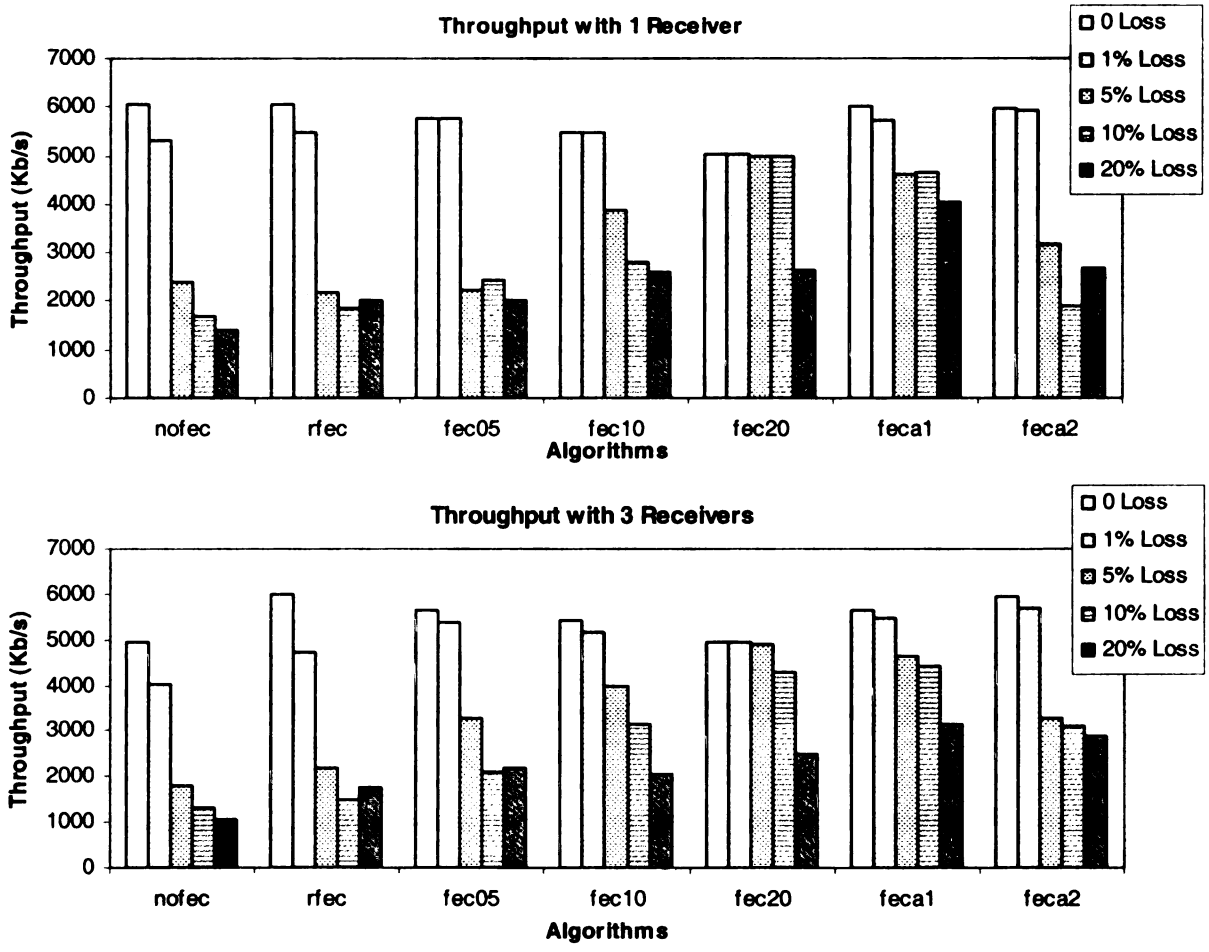


Figure 5.3: Experimental throughput with artificial loss.

Next we study the relationship between the throughput and the number of NACKs, as well as between the throughput and the *goodput* value. The number of NACKs is a metric of reverse traffic. The goodput is calculated by dividing the number of transmitted data packets by the number of total downlink packets. As such, goodput is a measure of the channel efficiency, since sending more parity packets leads to lower goodput. In many application environments, channel bandwidth can be exclusively utilized by a protocol such as AFRM, so the goodput itself is not a major concern. However, goodput reflects the behavior of the protocols and can be used to understand throughput results. We selected the packet traces that produce medium throughput for each protocol, under 20% loss conditions, and calculate the throughput, the number of NACKs and the goodput. The results are depicted in Figure 5.4.

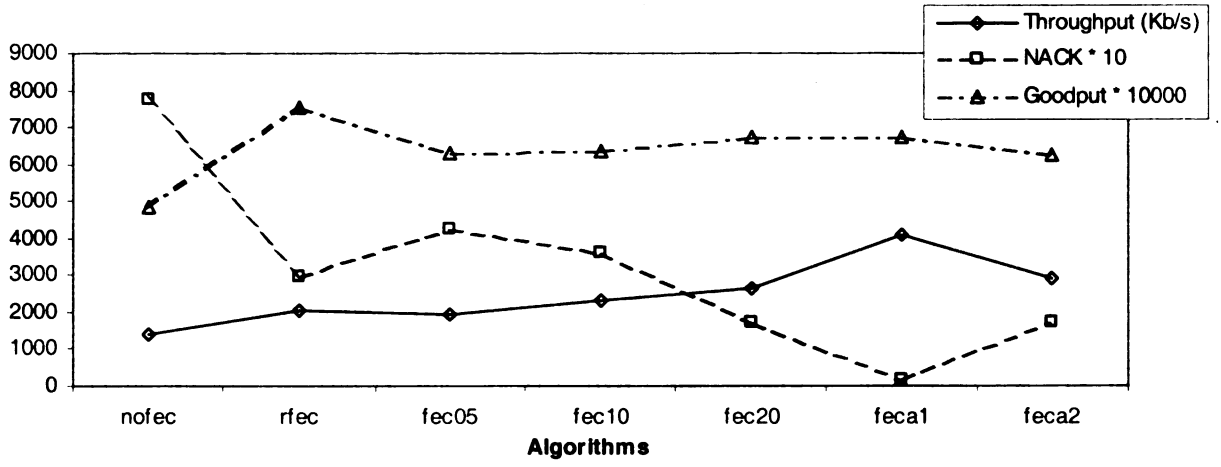


Figure 5.4: Throughput, goodput and total NACKs with 20% loss.

It is clear that the throughput is inversely proportional to the number of NACKs. On the other hand, the goodput has less impact on the throughput and is relatively constant for all the FEC-based approaches. This result implies that aggressively sending proactive parity packets is likely to be beneficial to the throughput in the test environment. Some of

the proactive packets might be useless, but the bandwidth waste could be offset by the reduction of reverse traffic (which in turn reduces retransmissions). The AFRM protocols exhibit higher throughput than the others mainly because they generate fewer NACKs. The AFRM-A1 protocol is better than the AFRM-A2 in this case, since A1 sends proactive packets more aggressively.

Next we examine the proactive rate adaptation of the AFRM protocols. We plot the number of proactive packets, the number of required parity packets and the number of reactive parity packets for every group from a typical run of AFRM-A1 and AFRM-A2. In AFRM-A1, the sender adapts the proactive rate whenever a NACK is received. The algorithm aims to suppress as many NACKs as possible even at the risk of sending too many proactive parity packets. In the test environment, aggressively sending proactive packets is justified since the reverse traffic is more harmful to the performance than is the bandwidth waste. However, this conclusion depends on the implementation of the wireless network MAC protocol and the channel loss pattern. In networks where the reverse traffic has less adverse impact on the performance, or the channel loss is not uniformly distributed, the proactive rate adaptation algorithm should change its parameters. Figure 5.5 shows that the AFRM-A1 algorithm behaves well on the uniform artificial loss model. The number of proactive parity packets being sent is just hovering above the number of parity packets needed. However, we will show later that the performance is not so good on bursty losses. In AFRM-A2, the sender determines whether or not it needs to send proactive packets, according to the loss burst length distribution. In the case that it needs to send, the proactive rate is calculated based on the average loss rate. In this study we do not address further how to find the optimal

parameters for a specific implementation of a wireless LAN. We defer that issue to future investigations.

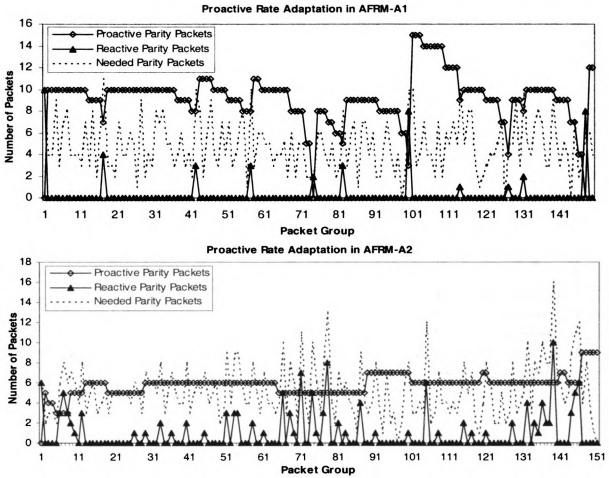


Figure 5.5: Proactive rate adaptation in AFRM algorithms.

To evaluate the performance of the NACK suppression approaches, we tested both local and global suppression for the FEC algorithms in multicast groups with 3 receivers. The suppression window size is set to 100 milliseconds. In Figure 5.6, we plot the average throughput of each protocol under 20% loss conditions.

The figure shows that local NACK suppression improve the performance slightly in cases where the reverse traffic is relatively heavy (rfec, fec05 and fec10) and where the

proactive rate is low. For fec20, feca1 and feca2, however, since most NACKs are already suppressed by proactive packets, local NACK suppression cannot produce enough gains to offset the overhead. Therefore the throughput is lower for those algorithms. Moreover, global NACK suppression will definitely hurt performance in this environment, because the group size is too small. We turn the global NACK suppression flag off until the multicast group becomes larger.

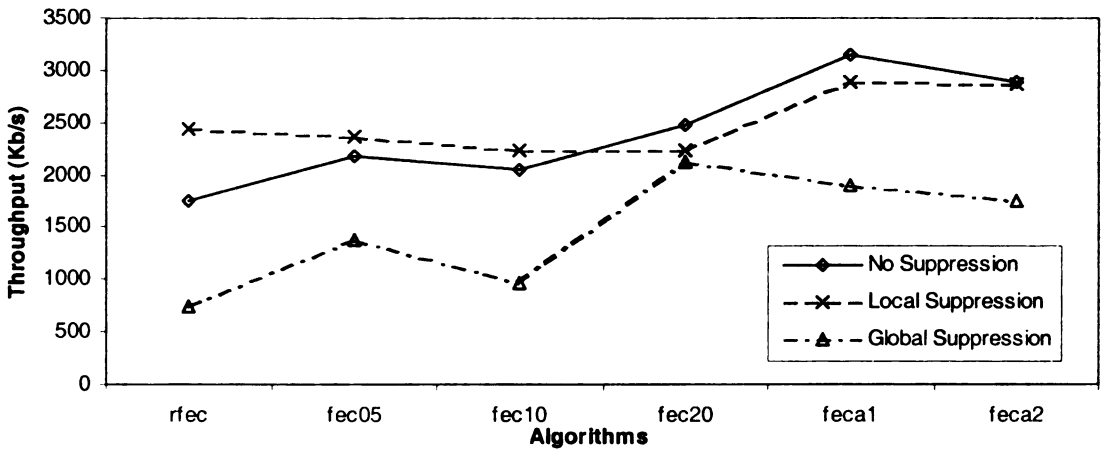


Figure 5.6: Experimental throughput with NACK suppressions.

5.2 Simulation Study

To ensure that a network protocol has a good overall performance, it is necessary to conduct sound and extensive performance evaluations. In wireless LANs, packet losses are bursty and almost unpredictable. Therefore, it is hard to guarantee the fairness in a performance comparison since the channel condition is constantly changing. Moreover, the expense of organizing a large-scale wireless LAN testbed is prohibiting. On the other

hand, the study of the protocol behavior under various loss patterns and network scales is indispensable. To solve this problem, we designed and implemented a network simulation framework that enables us to scrutinize the protocol performance under various conditions.

5.2.1 Simulation Framework

We built a general-purpose network simulation framework atop the CSIM18 simulation package [41]. The framework generates and manages communication objects such as virtual networks, links, packet loss patterns, virtual stations, and virtual processes. It supports the inter-networking concept and provides the basic routing service. The wireless access point is simulated as an object instance of the virtual router, while the wireless channel is simulated as a set of link objects, each of which connects two virtual stations with an associated loss pattern object. Currently four loss models are supported: packet trace, uniform model, bursty model and synthetic model. On each virtual station, a mini operating system is running, which maintains a protocol stack. Simple implementations of network protocols such as UDP, TCP, IP, IEEE 802.3 MAC and IEEE 802.11 MAC are included. They are used to simulate the network traffic processing between the network interface card and the application code. The architecture of the framework is depicted in Figure 5.7.

The framework supports the socket API by over-riding the real socket library. Any network applications built on the socket interface can be easily ported to this simulation platform. For example, let us imagine two application programs that run on different stations and which communicate with each other through a network channel. One application behaves as a client, and the other acts as a server. The server application

listens on a particular socket, waiting for any requests from the clients. When a client application starts, it creates a socket and connects to the server socket by specifying the appropriate address and the port. Subsequently, these two programs transmit data to each other by using send() and receive() socket call on the socket pair.

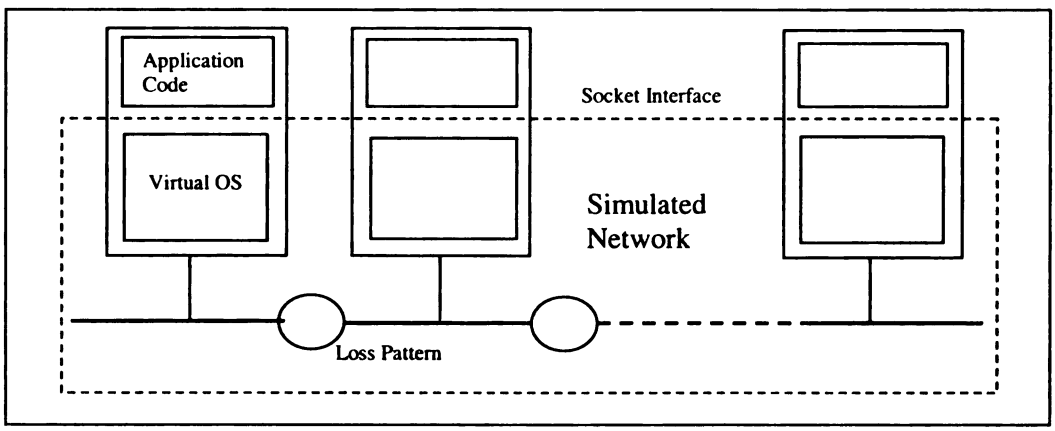


Figure 5.7: Architecture of the simulation framework

To port these two programs to our simulation framework, we first generate a simulated network, which includes at least two virtual stations and a link object that denoting the network channel between these two stations. Next, we relink the client and server application code with the new socket library provided by the framework. When the new code is loaded, a virtual process is created on each of the virtual stations. The client as well as the server code is invoked from one of the virtual processes. Now the client application and the server application begin running on the simulated network.

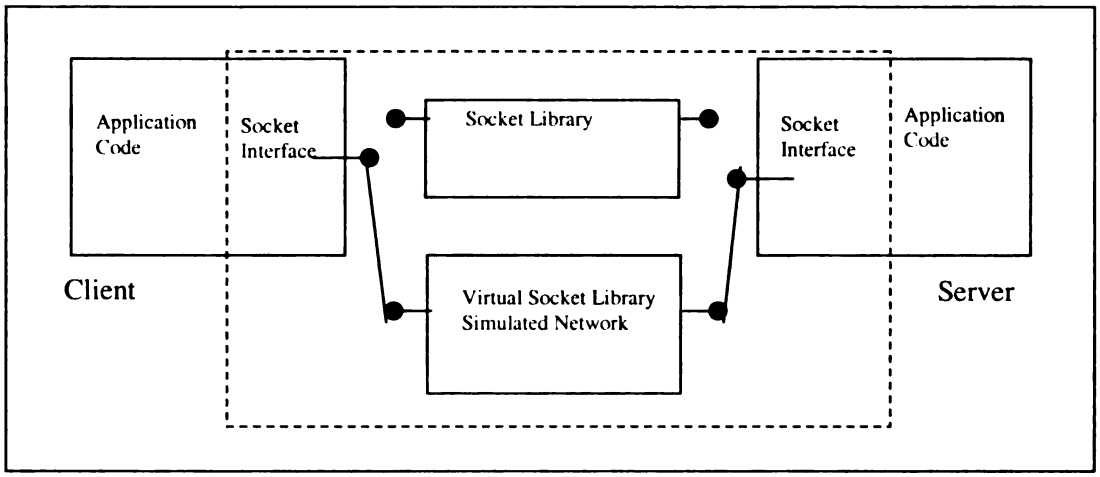


Figure 5.8: Porting applications to the simulated network.

After porting applications to a simulated network, we can study their performance by varying the network configuration and the object parameters. The network scale can be changed by simply adding or deleting virtual stations in the network. To change loss patterns between a wireless station and the access point, we simply change the link object between the two stations by updating the corresponding loss pattern object by specifying new parameters. In next section we describe the loss models in detail.

5.2.2 Loss Models

Packet Trace. This model replays packet losses from the real wireless channel. We collected a large amount of packet traces to study the characteristics of a wireless channel. The procedures and analyses are described in Section 3.1. These traces can be also used to configure loss pattern objects in a simulated network. When generating such networks, we provide a configuration file specifying how many stations are included in the network, the bandwidth and loss patterns for each link, and so forth. The type of a

loss pattern is specified in the file. If the type is a packet trace, the trace file name is given. When a packet is transmitted over a link with a packet trace loss pattern, the simulator looks up the packet trace in the file. The packet will be dropped if the trace indicates a packet loss at the current location. Otherwise it is passed over the channel. After this processing, the simulator updates the current location by moving the pointer to the next packet in the trace file.

Uniform Loss. For each uniform pattern, the loss rate is specified in the configuration file. When a packet is transmitted over a link with this model, the simulator generates a random number between 0 and 1, comparing it with the loss rate of the model. If the random number is smaller than the loss rate, the packet will be dropped, else it will be passed successfully. This model is very simple to implement. However, it does not reflect the real loss distribution in a wireless channel.

Bursty Loss. As we saw in Section 3.1, wireless loss is very bursty. In a loss burst, most of the packets in the channel are lost. In a loss-free burst, few or no packets will be lost. It is natural to use a two-state Markov chain to model the channel behavior. The Gilbert-Elliot model [20] is widely used in the research community. For each channel the model maintains two states. At a specific moment, a channel can be in either the good or the bad state. If the channel is in the good state, the loss rate is very low. If the channel is in the bad state, however, the loss rate is relatively high. A simplified version the model assumes that there is no packet loss at all in the good state and that all packets are lost in the bad state. If a packet loss occurs in the good state, the model immediately switches

the channel to the bad state. Similarly, whenever a packet is successfully transmitted, the model switches the channel to the good state. Assume that the probability of staying in the good state is α, and the probability of staying in the bad state is β. The probability that any packet loss occurs in the good state is $(1-\alpha)$, and the probability that any successful packet delivery occurs in the bad state is $(1-\beta)$. The model is depicted in Figure 5.9. Due to the simplicity for implementation, a number of research works use this version of the model. We also adopt this simplified version in our simulation framework.

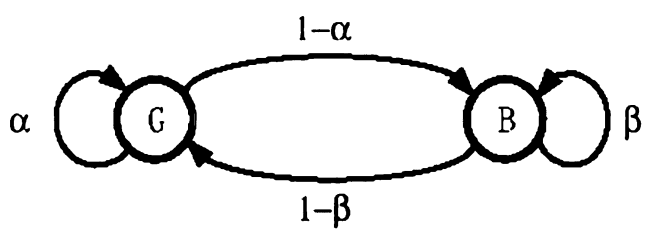


Figure 5.9: Two-state Markov model for simulating packet losses and channel errors.

For each bursty pattern, the transition probabilities between good state and bad state are given in a configuration file. When a packet is transmitted over a link with this model, the simulator generates a random number between 0 and 1, and checks the channel state. If the channel is in the good state, and the random number is smaller than the good-to-bad transition probability, the packet is dropped and the channel switches to the bad state. If the channel is in the bad state, the packet is dropped unless the random number is smaller than the bad-to-good transition probability. The channel switches back to the good state whenever a packet is not dropped.

Synthetic Loss. When generating a simulated network, we need to set a loss model for each link. For a set of links, three approaches can be taken. First, we can set different loss

models for each link. Second, we can use the same loss model, but different instances. Third, we use the same instance for all links. In the first approach, the links will certainly experience independent packet losses. In the second approach, the links share the same loss model. However, the instances are different. Since each instance maintains a separate random number stream as well as unique random number seeds, the links will get different and independent random numbers, which lead to independent packet losses. In the third approach, the links will experience identical packet losses.

In Section 3.1 we saw that packet losses are somehow correlated at different wireless stations. They are neither identical nor totally independent. We will be unable to maintain this relationship when setting loss patterns by using any of the three approaches described above. To solve this problem, we added a synthetic loss model to the simulation framework. In this model, a trace is selected as the base error model. Multiple stations share the base trace with a certain probability. In other cases they generate independent packet losses from the bursty model. Therefore, the correlation feature mentioned above is maintained. We can change the sharing probability to adjust the correlation coefficients.

The synthetic model can be implemented either statically or dynamically. In static implementations, packet losses at each station are calculated a priori. The resulting losses are written to a trace file, whose name is provided in the configuration file. In dynamic implementations, each loss pattern is set to the synthetic model and associated with a base trace file, a sharing probability, and a bursty loss pattern instance. Packet losses at each station are dynamically. For simplicity, we use the synthetic model statically in the framework.

Revised Bursty Loss Model. After analyzing the packet traces we collected in section 3.1, we realized that the simple Gilbert model is probably not a good approximation to the traces. We generate a packet trace based on the simple Gilbert model and compare it with a real trace in Figure 5.10. The good-to-bad transition probability is set to 0.0006, and the bad-to-good transition probability is set to 0.16. Both probabilities are calculated from the real trace.

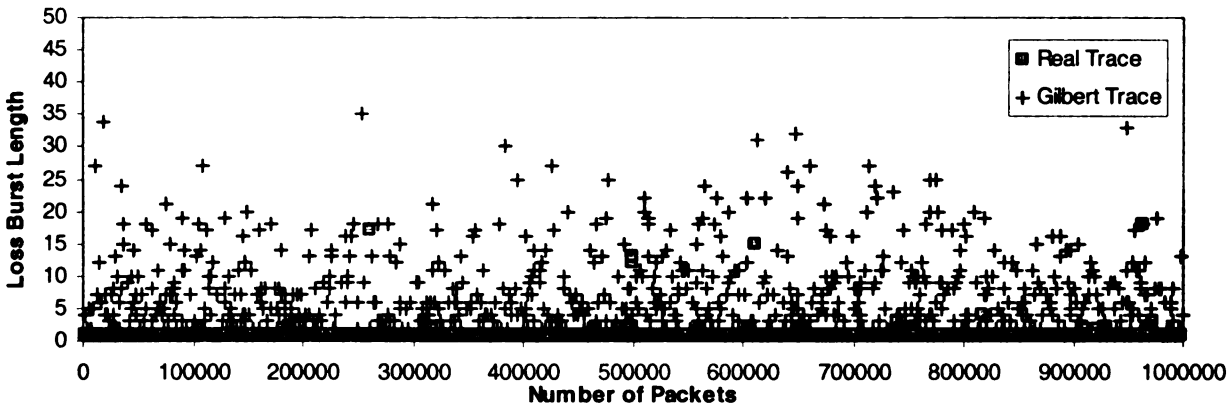


Figure 5.10: Real trace and trace based on Gilbert model.

In the simple Gilbert model, the loss burst length as well as the loss-free burst length follows a geometric distribution. When we look at the packet traces statistics in Figure 5.11, however, it turns out that the exponential distribution is a good approximation to the loss-free burst length. Moreover, the loss burst length distribution is more spread out, like bimodal or Pareto distribution. Therefore we revise the bursty loss model so that the loss-free burst length is exponentially distributed and the loss burst length distribution is in pareto. We use this revised model to generate the synthetic traces as described above. In Figure 5.12, the Pareto parameter a is set to 0.3 and k is set to 100000.

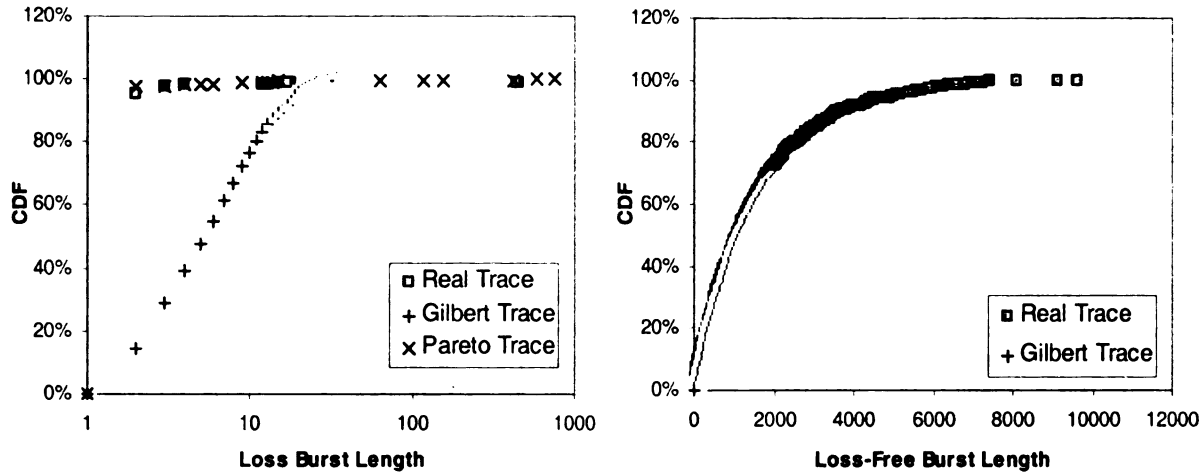


Figure 5.11: Burst length distribution of packet trace.

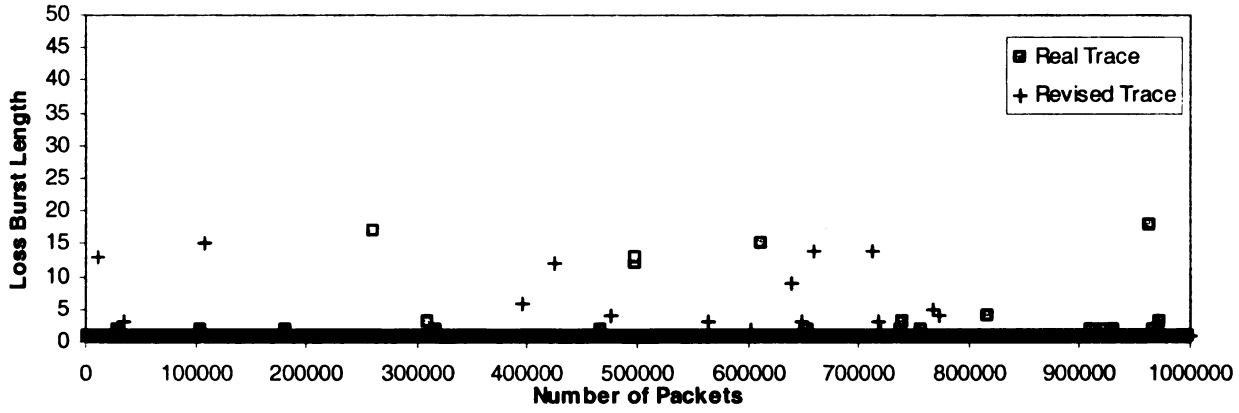


Figure 5.12: Trace based on revised bursty loss model.

5.2.3 802.11 MAC Simulation

Our network simulation framework supports IEEE 802.11 MAC protocols. In IEEE 802.11 Standard [25], there are two versions of MAC protocols: a station version and an access point version. Both are relatively complex. We ignore those parts that are unrelated to the packet transmission in typical scenarios and simplify the MAC protocol

as described in Figure 5.13. The simulation study in Chapter 3 is also based on this implementation.

```
sender() {
        while (true) {
                pkt = fetch();              // fetch packet from data queue
                if (!checkRx())             // check if the channel has been idle for DIFS or EIFS
                        backoff();          // backoff if a transmission within the last DIFS or
EIFS is detected
                while (!transmit(pkt)) {
                        if (!backoff()) { // backoff between retransmissions
                                break;  // too many retranmissions, transmission failed
                        }
                }
        }
}

transmit(pkt) {
        if (is_multicast(pkt) && is_access_point()) {
                dispatch(pkt);              // pkt propagation
                CW =  CWMin; shortRetryCount = longRetryCount = 0; // reset
                return true;
        }
        if (size(pkt) > RTSThreshold) {
                dispatch(RTS);
                wait(SIFS);                 // wait for CTS
                if (!CTS_arrived()) {    // no CTS
                        shortRetryCount ++; incW();     // increase window size
                        return false;
                }
                shortRetryCount = 0;
                wait(SIFS);
        }
        dispatch(pkt);
        wait(SIFS);                         // wait for ACK

        if (!ACK_arrived()) {    // no ACK
                if (size(pkt) > RTSThreshold)    longRetryCount ++;
                else shortRetryCount ++;
                incW();
                return false;
        }
        CW =  CWMin; shortRetryCount = longRetryCount = 0; // reset
        return true;
}
```

```
receiver() {
        while (true) {
                pkt = receive_pkt();      // block if no packet detected
                if (type(pkt) == RTS) {
                        if (ready_to_receive()) {
                                wait(SIFS);
                                dispatch(CTS);
                        }
                } else if (type(pkt) == CTS) {
                        CTS_arrived() = true;
                } else if (type(pkt) == DATA) {
                        if (!is_multicast(pkt) || is_access_point()) {
                                wait(SIFS);
                                dispatch(ACK);// send ACK
                        }
                } else if (type(pkt) == ACK) {
                        ACK_arrived() = true;
                }
        }
}
```

Figure 5.13: Implementation of IEEE 802.11 MAC protocol.

5.2.4 Simulation Results

To evaluate and compare the protocol performances in large multicast groups, and study their behavior over various loss scenarios, we executed simulations similar to the experiments in Section 5.1, but we modeled the wireless channel with different loss patterns and varied the group size. The multicast group size is set to 1, 3, 10, 25, 50 and 100 respectively.

Packet Trace. We collected several packet traces in the experiments described in Chapter 3. We used these traces to simulate packet losses. The throughput is depicted in

Figure 14. The traces from node5 in Table 3.1 are taken in the case of 1-receiver multicast group. The traces from node2, node5 and node 8 are chosen for the case of 3 receivers. All 10 traces are included for tests in a 10-receiver multicast group.

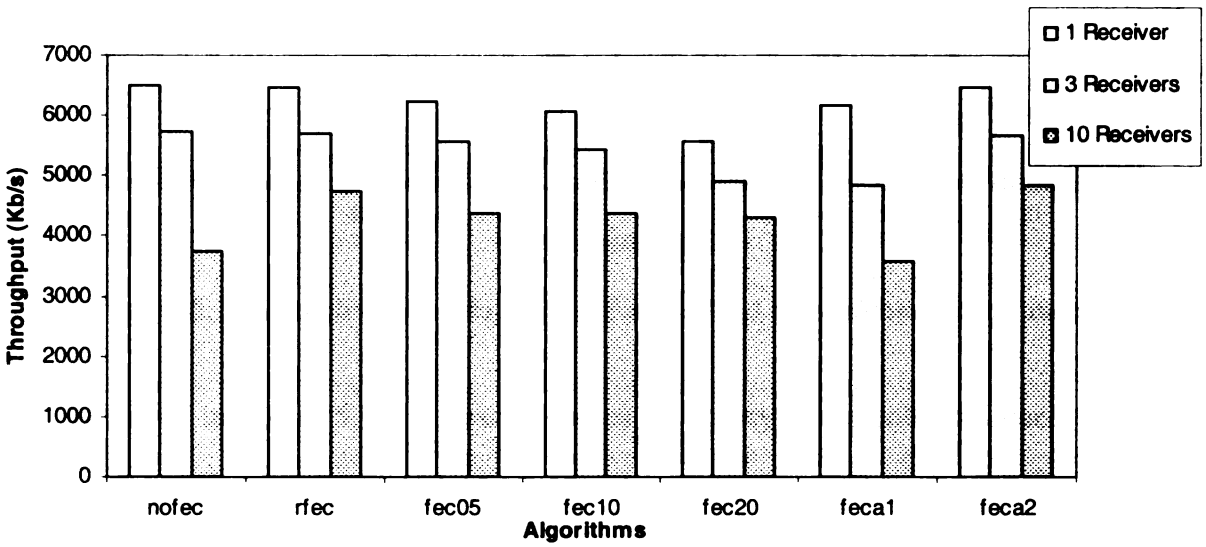


Figure 5.14: Simulation throughput on wireless packet trace.

Figure 5.14 shows that the throughput drops rapidly when the multicast group becomes larger. Among the protocols, pure-reactive FEC and AFRM-A2 have the best performance. Surprisingly AFRM-A1 becomes the worst. Other proactive protocols also suffer. To determine why AFRM-A1 performs so poorly, we plot in Figure 5.15 the number of proactive packets, number of required parity packets and number of reactive parity packets for every group from a typical run of AFRM-A1.

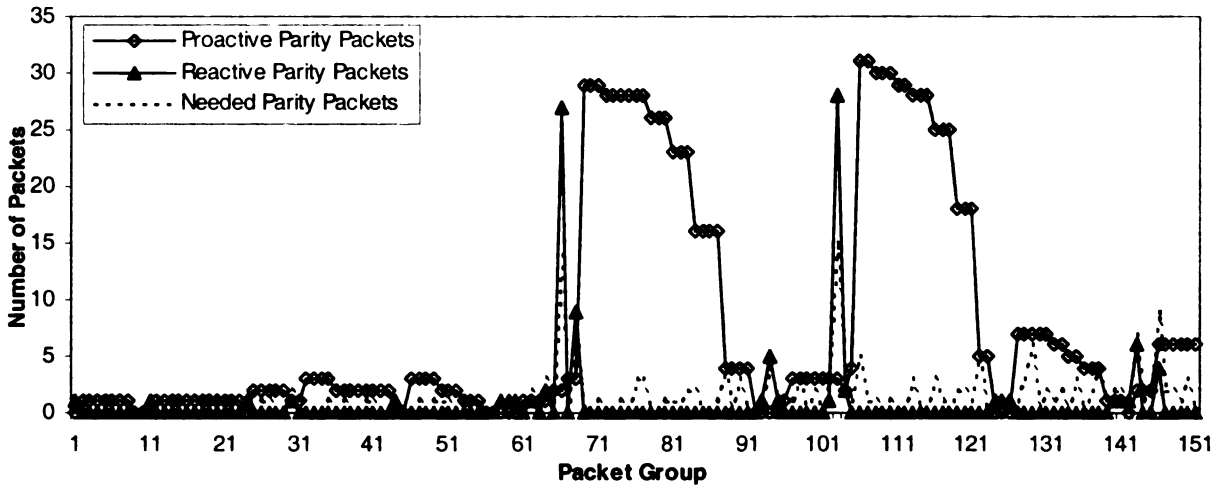


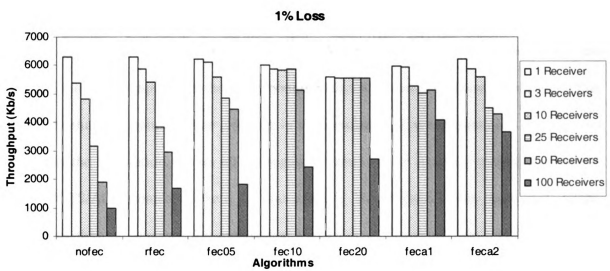
Figure 5.15: Proactive rate adaptation of AFRM-A1 on wireless packet traces.

As shown in Figure 5.15, the packet losses are bursty in the collected traces. In this case, the AFRM-A1 approach is slow to reduce the proactive rate, resulting in the sending of many unnecessary proactive packets following a long error burst. On the other hand, AFRM-A2 checks the average loss burst length and the average loss rate to calculate the proactive rate. In these tests, an AFRM-A2 sender sends few proactive packets. Therefore it exhibits similar performance to that of the pure-reactive FEC protocol.

We want to emphasize the fact that the reverse traffic overhead is lower in the simulation framework than it is in the real test environment. This is because a real access point needs to periodically transmit some management frames besides relaying data packets. This transmission will likely cause more packet drops at the access point. As a result, all protocols have higher throughput in the simulation than they have in the test environment. However, the performance of the proactive protocols does not increase as fast as that of the others due to their redundancy and the overhead of NACK suppression. We expect them to exhibit performance in a real network. On the other hand, excluding

the long loss bursts from the calculation of the proactive rate might be a good idea. We will explore the ideas in our future work.

Uniform Loss. Random loss under a uniform distribution is probably the simplest loss model. We use an independent loss approach. In this approach, each wireless station is assigned a unique random stream. Propagation loss perceived by a station is totally determined by the random stream. It would appear that this model is not very realistic. First, as we showed in Chapter 3, wireless propagation loss is bursty rather than uniform. Second, losses at different stations are somehow correlated rather than independent. On the other hand, packet losses do include congestion losses and collision losses, as well as propagation losses. This model affects only the propagation losses, which are only a portion of packet losses. Given that congestion losses and collision losses are well simulated, this model is perhaps still useful due to its simplicity. We plot the throughput of each protocol in Figure 5.16, for different loss rates.



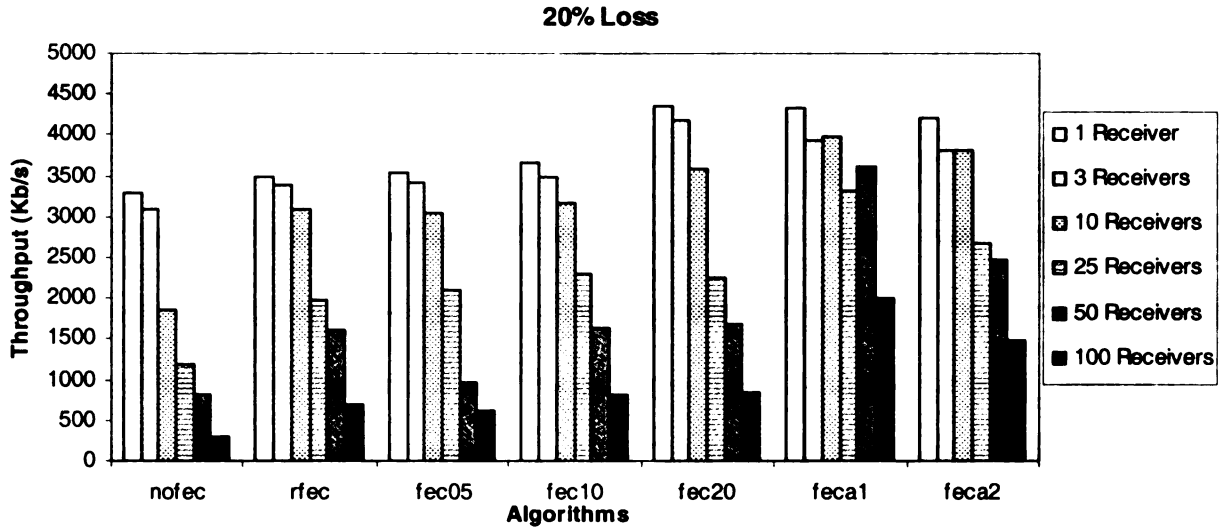


Figure 5.16: Simulation throughput on uniform loss model.

Figure 5.16 shows that the throughput drops rapidly when the multicast group becomes larger or the loss rate becomes higher. Among the protocols, the non-FEC approach drops the fastest. Both AFRM-A1 and AFRM-A2 protocol outperform almost all other protocols at 100 receivers or 20% loss rate. The fixed-rate proactive protocols also demonstrate good performance on the uniform loss model. In the case that the proactive packets are sufficient to recover lost packets, the performance of these approaches is comparable to that of the AFRM protocols. When comparing the simulation throughput with the corresponding results on artificial losses, we noticed that the absolute throughput value is higher in the simulation. The explanation is that the loss rate in the simulation covers all propagation losses in the channel, including even the RTS/CTS/ACK control packets. On the contrary, the artificial losses apply only to the packets that are successfully received at the receiver. Therefore, the actual loss rate is higher in the experiments. On the other hand, the reverse traffic incurs smaller overhead

in the simulation, which implies that the AFRM-A1 protocol may further outperform the other protocols in a real network with a large number of receivers.

Bursty Loss. We use the two-state Markov model to simulate the bursty wireless losses. Similar to the approach in [46], we use the reciprocal of the average burst length as the transition probability between states. For example, if the average error burst length is 10 and the average error-free burst length is 1000, then the bad-to-good transition probability is 0.1 and the good-to-bad transition probability is 0.001. The actual transition probabilities used in our tests are calculated from the average burst length in the packet traces we collected. In Figure 5.17, the good-to-bad transition probability is set to 0.01, while the bad-to-good transition probability is set to 0.2.

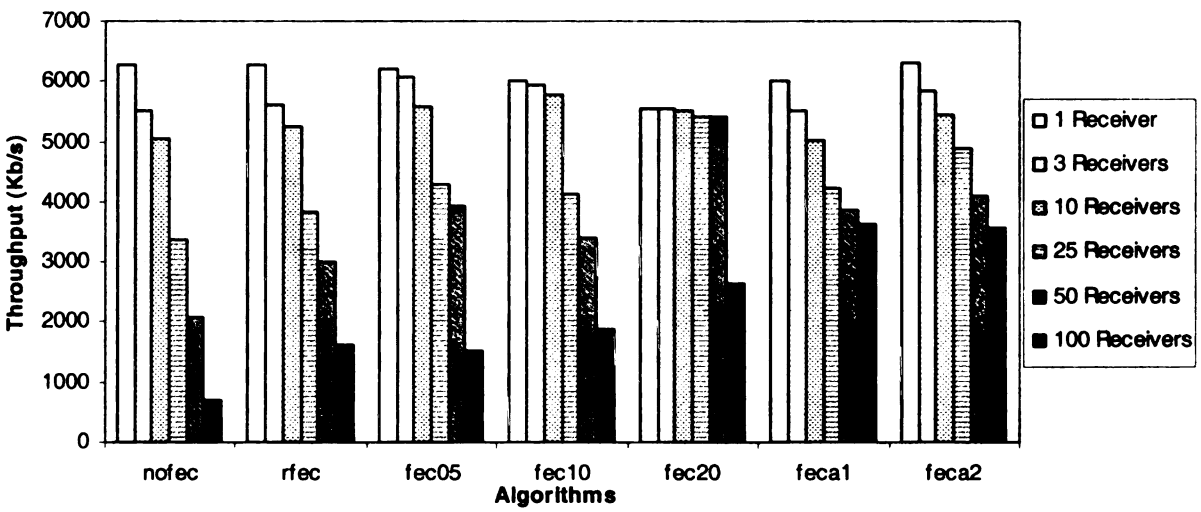


Figure 5.17: Simulation throughput on bursty loss model

The relative protocol performance on bursty losses is similar to that under uniform losses. One exception is that the performance of the AFRM-A1 is slightly lower than that of the AFRM-A2 as well as many other protocols, apparently due to a high proactive rate,

until the multicast group size becomes fairly large. On the other hand, AFRM-A2 outperforms most other protocols by a small amount in large groups.

This model is a better approximation to the real wireless losses than is the uniform loss model. Since we adopt the independent loss approach here again, however, it also does not consider loss correlation among receivers.

Synthetic Loss. As described earlier, the synthetic loss model is based on a combination of the packet traces and the bursty loss model and therefore possesses the advantages of both models. Thus, it may be more realistic than the uniform loss model. Moreover, this model takes into account of the loss correlation at multiple wireless stations. In this test, we generated the synthetic trace using a real trace with a moderate loss rate (around 5%). We plot the throughput for the various protocols in Figure 5.18.

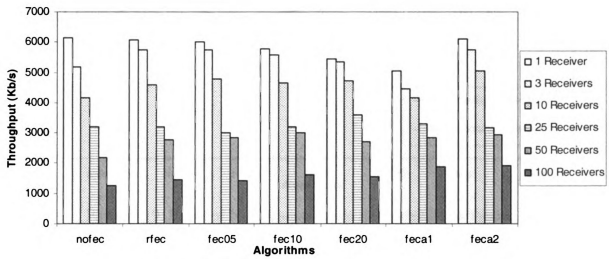


Figure 5.18: Simulation throughput on synthetic loss model.

On the synthetic model, the performance difference among protocols becomes smaller. Since the loss correlation makes the overall loss less "flat," the proactive FEC protocols

are less effective in the NACK suppression approaches as they are on other models. The AFRM protocols have no significantly better performance than others, but neither worse. Once again AFRM-A2 outperforms AFRM-A1. However, if taking into account more reverse traffic overhead, the AFRM-A1 protocol should have better performance in a real network.

Since the packet losses are condensed, the number of receivers that simultaneously send NACKs becomes larger. As a result, the NACK suppression approach should be more effective on this model. To verify this hypothesis, we turn on the NACK suppression flag and compare the performance in Figure 5.19. The number of receivers is 25. The result shows that the NACK suppression approach moderately improves the performance in this case.

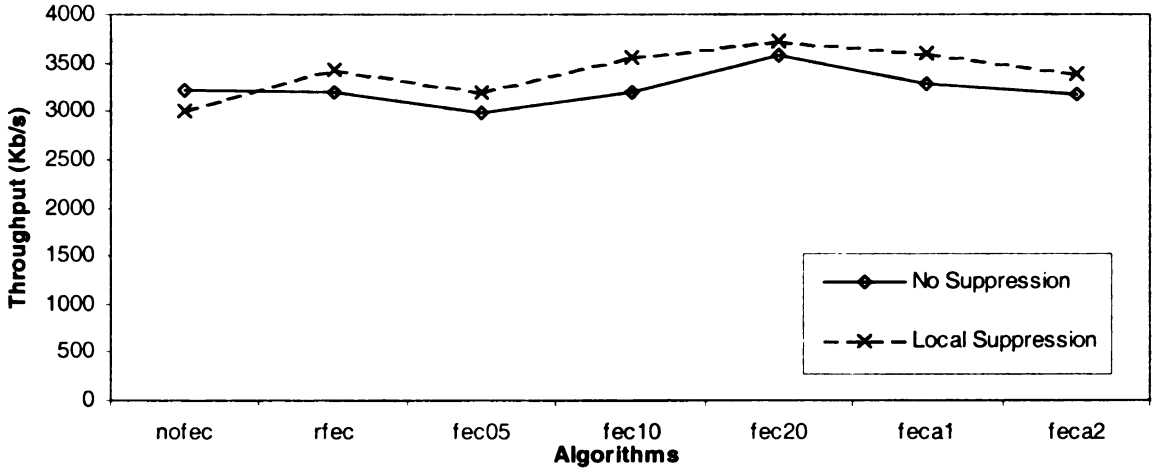


Figure 5.19: Effect of NACK suppression on synthetic loss model.

5.3 Summary

In this chapter we presented the results of both experimental and simulation studies on the performance of the AFRM protocols and other reliable multicast protocols. Although the result of measurement depends on the channel loss model and the network implementation, the AFRM protocols exhibit better overall performance than other protocols, especially when the multicast group is large and the loss rate is high. In the case of low loss rate and small networks, the gain from NACK suppression is also small. Therefore it is likely that the AFRM protocols have slightly lower performance than the others under these conditions. We believe that the protocols can be further optimized so that they will have better performance when there is less reverse traffic. A possible approach is to adopt a hybrid scheme in which the sender transmits proactive parity packets only when both the channel loss rate *and* the number of receivers are larger than threshold values.

Chapter 6

Related Work

As mentioned in Chapter 2, three major research areas are directly related to this study: wireless networks, reliable multicasting and forward error correction. A number of algorithms used in the AFRM protocols, such as proactive FEC and NACK suppression, are based on previous research contributions in the reliable multicasting and FEC areas. The wireless channel loss analysis and the error model design are based on recent research in the area of wireless networks. In this chapter, we discuss the related works in these areas and their relationship to our study.

6.1 Reliable Multicast

A large number of reliable multicast protocols have been proposed. They can be classified into several categories based on their strategies in loss detection, loss recovery and receiver hierarchy.

ACK-based and NACK-based Protocols. The Xpress Transport Protocol (XTP) [61] is an ACK-based reliable multicast protocol. A XTP sender periodically requests receiver status by setting a bit in the outgoing packets. Receivers respond to the request by

returning an ACK packet. The sender aggregates information from the ACKs and retransmits data to the entire multicast group. As in XTP, an AFRM sender controls the receiver behavior by setting flag bits in the outgoing packets.

The Log-Based Receiver-Reliable (LBRM) protocol [23] is a user-level implementation of NACK-based reliable multicast protocol. In LBRM, a log server is set up to manage data retransmissions. A sender transmits data by multicasting to the receivers as well as to the log server. After the log server receives the data, the sender can safely release the data buffers. When a retransmission is needed, the receiver sends a request to the log server and the server handles the retransmission. To facilitate loss detection, LBRM senders periodically transmit heartbeat packets. The heartbeat packets are sent at variable rates, so that the senders can keep the total number of heartbeat messages much smaller than that in fixed-rate protocols. Like LBRM, AFRM is also NACK-based and implemented at the user level. An AFRM sender transmits keepalive packets with binary exponential backoff, which is similar to the transmission of heartbeat packets with variable rates in LBRM.

The Reliable Adaptive Multicast Protocol (RAMP) [32] is a transport layer protocol on top of the IP multicast services. It is NACK-based and is designed to deal with uncorrelated losses among receivers. In this protocol, a receiver returns a NACK by unicasting immediately after a packet loss is detected. The sender responds to the NACK with a unicast retransmission. This approach reduces the cases of unnecessary receiver exposure to retransmissions since most packet losses are uncorrelated. Immediately sending NACKs also decreases recovery time. Based on RAMP, we take the recovery time into consideration when choosing NACK suppression algorithms in AFRM.

The Reliable Multicast Protocol (RMC) [69] is another NACK-based protocol. It supports anonymous group memberships. It uses a combination of window-based and rate-based flow control algorithms to ensure pure NACK-based reliability with limited receiver buffer spaces. Although RMC is designed for use in the OS kernel, the code structure of AFRM sender/receiver is mainly based on that of RMC.

Local Recovery. The Scalable Reliable Multicast (SRM) protocol [17] is an example of unstructured local recovery with NACK suppressions. SRM defines local loss neighborhoods based on the number of hops from the requesting host. A receiver with packet losses transmits a NACK by multicasting, specifying the limit of routing hops or TTL (Time-To-Live) value. Any other receivers overhearing the NACK can retransmit the requested packets if the data is available, or cancel their own NACKs if they experience the same losses. In the cases of retransmission, they send the packets by multicasting with the same number of hops specified. Since both NACKs and retransmissions are sent by multicasting, the recovery traffic is reduced. In AFRM, we borrow from SRM the strategies of global NACK suppression and multicast retransmission.

The Reliable Multicast Transport Protocol (RMTP) [53] is an ACK-based protocol using structured local recovery. In this protocol, each receiver is assigned a designated receiver (DR) to which it periodically sends ACKs. One DR serves multiple receivers and processes ACKs to determine which packets to retransmit. A DR retransmits a particular packet either by unicasting or by multicasting, depending on the number of receivers requesting this packet. RMTP II [67] is an extension of RMTP. It integrates NACK, FEC

and other techniques into the RMTP protocol to improve scalability and real-time performance.

The Tree-based Multicast Transport Protocol (TMTP) [74] is another example of structured local recovery. In this protocol, receivers are grouped into domains. Each domain has a domain manager. TMTP adopts an ACK-based approach to ensure reliability between the sender and any domain manager, as well as between a domain manager and its domain members. A receiver can aggregate several ACKs into one packet to reduce feedbacks. It can also use NACKs to speed up error detection in the case of ACK aggregation.

Active Reliable Multicast (ARM) [38] is a loss recovery approach in which the routers in the multicast tree play active roles in loss recovery. It uses a router as a DR in RMTP or a domain manager in TMTP, integrating routing, feedback aggregation and subcast retransmission in a natural way. It utilizes soft-state storage techniques to improve performance and scalability.

A study of the random delay value in feedback suppression is described in [49]. That study shows that an approach with exponentially distributed random delay leads to lower feedback latency as well as higher feedback suppression efficiency than does an approach with uniformly distributed delay. The proposed approach is scalable with respect to the number of receivers, and it can effectively avoid feedback implosion. Although AFRM uses uniformly distributed random delay, we do benefit from the analysis of this work.

Multiple Multicast Groups and Adaptive Approaches. Multiple multicast channels are utilized in a number of research works to improve loss recovery efficiency. In [30], an

approach is proposed to reduce receiver processing costs as well as network bandwidth consumption in a multicast session. It uses one multicast channel for data transmission and several separate channels for retransmissions. The receivers dynamically join or leave the retransmission channels. The work shows that this approach is efficient in bandwidth saving since the chance of unnecessary receiver exposures to retransmission traffic is reduced. The work also explores the tradeoff between the number of multicast channels and the exposure cost.

In [75], a simple approach is proposed to adapt the number of multicast groups in response to the perceived channel conditions. It utilizes both a centralized and a distributed mechanism to determine the temporal dependence among packet errors. It uses a predefined threshold value to differentiate bursty losses from other losses. In case of bursty loss, it increases the number of multicast groups to maintain prompt and efficient retransmissions.

A similar adaptive approach is proposed in [13]. In that work, an analytical model is built to explore the tradeoffs between unicast and multicast retransmissions. As mentioned in Chapter 2, multicast retransmissions have great advantage over unicast retransmissions in the case of correlated losses. On the other hand, the multicast approaches will waste network bandwidth when losses are less correlated, because in that case a receiver might receive unneeded retransmission packets. This work studies the problem and proposes an approach that chooses the best retransmission method based on the information of the network topology and the number of receivers. An extension of this work [14] introduces an approach that dynamically switches between multicast and unicast retransmissions according to the amount of extra load generated in the network.

Similar to these approaches, AFRM also adapts retransmission strategies in response to the channel condition changes.

Performance Analysis and Comparison. Many reliable multicast protocols exhibit different performance when the underlying environment changes. It is very hard to design a protocol that is optimal under all conditions. To facilitate making tradeoffs in a specific environment, it is desirable to have quantitative performance results for various algorithm candidates. A number of research works focus on the performance analysis and comparison of reliable multicast protocols. In [66], an analytical evaluation of an ACK-based and two NACK-based protocols is described. The major evaluation metric is the host processing time instead of the network bandwidth consumption. The analysis assumes independent losses. The results show that the NACK-based protocols are superior to the ACK-based protocol. In [3], the authors evaluate and compare various reliable multicast protocols such as XTP, MTP [35], SRM, MFTP [42], RMP [11], RMTP, PGM [60], and TMTP. The metrics include reliability, scalability, feedback control, flow control, locus of control, late join/leave and efficiency.

Loss Estimation and Loss Differentiation. A few research works study the problem of loss estimation in a multicast tree. In [9], the authors propose an approach to estimate loss probabilities in the interior of a network. The approach is based on end-to-end measurement. In this approach, a source multicast probes that are recorded at each leaf of the multicast tree. The proposed Maximum Likelihood Estimator (MLE) exploits the inherent correlation between loss observations at different receivers. The results infer the

performance of paths between branch points in the multicast tree from the probe source to the receivers. The analytical model is validated by simulations.

As an example of loss differentiation, the work in [21] proposes an epoch-based approach to differentiate congestion losses from other losses. In this approach, loss notifications from the receivers are examined. If the notifications are transmitted in the same epoch (i.e., a short time interval), they are likely to be indications of congestion losses, and so that they are aggregated. Otherwise they are probably uncorrelated losses, and hence are reported to the sender individually. The classification is smoothed using a running average algorithm on the length of epochs.

6.2 Forward Error Correction

Many reliable multicast protocols use FEC mechanisms to increase the efficiency of error recovery. In [19], the FCAST and ECSRM protocols are proposed. They combine FEC with multicast retransmissions and exhibit better scalability than non-FEC approaches. As another example, a protocol combining active repair service (i.e. local recovery) and FEC is proposed in [58]. That work shows that such an approach improves bandwidth utilization efficiency over both non-FEC repair services and end-to-end FEC approaches. In domains with high loss or a large number of receivers, this approach can reduce bandwidth consumption as well as buffer requirements at repair servers.

FEC+ARQ. FEC approaches are often combined with ARQ in reliable multicast protocols in order to support complete reliability. In [29], the SHARQFEC approach is

proposed. It is a hybrid FEC+ARQ approach that operates in an end-to-end fashion. It utilizes receiver hierarchy to conduct local recovery. FEC is selectively added to regions that experience high losses.

The RMDP protocol [57] is another hybrid FEC+ARQ protocol. Several of its operating parameters are determined based on the conditions of the network. One such parameter controls the rate of proactive parity packets sent along with the data packets. The study shows that the appropriate value of the proactive rate depends on the loss rate. A value between 1.5 and 2.0 makes the probability of NACKs very low. The protocol uses global NACK suppression to reduce the feedback traffic.

Like RMDP, the Collaborative Reliable Multicast (CRM) protocol [43] is another approach using proactive FEC parity packets. It improves the performance at the nodes that are under emission control or are connected by links with low reverse direction capacities. Similar to RMDP and CRM, AFRM also uses proactive parity packets and global NACK suppression to reduce feedback traffic. However, their strategies in parameter determination are substantially different.

Adaptive FEC. A number of adaptive FEC approaches have been proposed. Some of those approaches are based on the adaptation of the amount of redundant information in a frame. Others adapt the FEC encoding parameters n and k at the packet level; examples include ARAM [2], MA_FEC [48] and ARQ/AFEC [59]. Most of these also utilize proactive FEC mechanisms. In MA_FEC, as an example, a table of optimal FEC code is maintained. In response to channel condition changes, the protocol looks up the most

appropriate FEC code in the table and starts using this code in the following transmissions.

In [70], a protocol that adapts FEC proactive rate is proposed. This link layer protocol additively increases the proactive rate when the perceived channel loss rate is high and multiplicatively decreases the proactive rate when the perceived channel loss rate is low. Based on SNR (Signal/Noise Ratio) analysis, the authors claim that an indoor wireless channel is temporarily stationary only during a short time interval. They also pointed out that losses are caused either by distance dependent signal degradation or by random interference, and the number of stations within a cell has little effect on the average loss rate. Since the protocol works at the link layer, the loss notifications can reach the transmitter fast enough to ensure that the channel estimation based on past losses is reasonably accurate for the prediction of future losses. Like this approach, AFRM-A1 also utilizes AIMD mechanisms to adapt the proactive rate. However, AFRM runs at the user level, which leads to relatively inaccurate channel estimations. As a result, AFRM-A2 takes the average error burst length into consideration when calculating the proactive rate.

The W-WBRM protocol [39] is designed as a component of the Pavilion middleware framework. The protocol is implemented at the application level. It also adopts AIMD algorithms for FEC proactive rate adaptation. In this work, the channel condition is estimated using the number of received NACKs. The proactive rate is applied to both data packets and retransmission packets, since the retransmission packets could also be lost. The proactive rate adaptation algorithm in AFRM-A1 is basically the same as that in the W-WBRM protocol, except for some minor modifications. For example, in W-

WBRM the proactive rate is cumulatively increased whenever a NACK packet is received, while the AFRM-A1 protocol recalculates the proactive rate in a non-cumulative manner upon receiving a NACK packet. The formulae for proactive rate calculations are also different.

Performance Analysis and Other Work. A number of research works focus on the performance analysis of FEC algorithms. In [50] and [51], the authors build analytical models for hybrid FEC+ARQ approaches. The performances of those approaches are studied under different loss models, such as spatially or temporally correlated loss models, homogeneous or heterogeneous loss models. The results show that integrating FEC with ARQ either at the same layer or at different layers will improve both efficiency and scalability for multicast transmissions.

The work in [44] studies the group loss probabilities of FEC in shared loss multicast communications. The CDF (Cumulative Distribution Function) for successful deliveries in a multicast tree is presented, as well as the expected value of the packet loss length. Some applications of those formulae are described.

The work in [55] studies the performance of a FEC recovery approach for IP telephony in a queuing system. Using a M/M/1/K model in the probability analysis, the work shows that the simple FEC approach is not appropriate for audio streams in this environment, where packet losses are mainly caused by buffer overflow.

6.3 Wireless Network Studies

Among the huge volume of research works in the field of wireless networking, we select and describe a few that are directly related to our study, including IEEE 802.11 wireless LAN performance analysis, performance of communication protocols over wireless LANs, wireless packet trace analysis and wireless channel modeling.

IEEE 802.11 Wireless LAN Performance Analysis. In [1] and [27], the authors measure the actual throughput of a 2.4GHz product of IEEE 802.11 wireless LAN. They analyze the sources of overhead, categorizing them into the inter-packet gap time, preamble, header fields for the PHY and MAC layer, and control frames such as ACK or RTS/CTS. The impact of the overhead is modeled. Comparisons show that the results of the model closely fit the measurement results. The model indicates that the maximum theoretical MAC layer throughput for 11Mbps bit rate is about 7.43Mbps. The maximum theoretical TCP throughput is about 5.2Mbps. The model is also used to predict the throughput in 5GHz 802.11a networks. The TCP throughput in such networks with 54Mbps bit rate can reach 29Mbps. We use these values to validate our simulation framework.

In [22] and [45], the impacts of the 802.11 MAC protocol parameters on the network performance are studied through simulation. A simplified model of 802.11 DCF is built in [22]. The effects of the backoff procedure, the extended interface spaces (EIFS), and the timing synchronization function (beacon) are studied using this model. Simulation results show that the bandwidth reduction due to EIFS and beacons is less than 10%. In [45], the simulation results are used to help determine both the effective throughput and

the mean packet delay for an offered load with different values of contention window size and number of contending stations. The study shows that the proper choice of CW (Contention Window size) has significant influence on the network performance.

In [7], [8] and [10], the authors derive analytical formulae for 802.11 protocols. In [7] and [8], the CSMA/CA based 802.11 DCF is studied. An analytical model is built to compute both the saturation throughput and the collision rate. The analysis is based on the usage of the discrete-time Markov chain. Both basic access and RTS/CTS access mechanisms are studied. Simulation results show that the analytic model is very accurate. In [10], the derived analytical formula is used to calculate the theoretical upper bound of the 802.11 protocol capacity. The work shows that the protocol could operate far from the theoretical limit depending on the network conditions, and an appropriate tuning of the backoff parameters could make the performance closer to the theoretical limit.

Communication Protocols over Wireless LANs. In [71], [72] and [73], the performance of the standard Internet protocols such as TCP and UDP are studied over wireless networks. In [73], the issues such as the host and interface heterogeneity, TCP bi-directional traffic, and error modeling are explored in a 2.4GHz DSSS wireless LAN. Many performance problems of TCP and UDP over wireless LANs are uncovered, such as the problems of achievable UDP throughput, fast sender and slow receiver, PCMCIA sender, collisions due to bad synchronization, and timer granularity. In [71] and [72], more such problems are discussed and approaches for improvement are suggested and examined. In [72], a mechanism for comprehensive link layer enhancement is proposed.

In [4], a number of approaches for TCP performance improvement over wireless links are described and compared. These approaches are classified into several categories, such as end-to-end protocols, link-layer protocols and split-connection protocols. For each approach, the experimental results on throughput and goodput in both LAN and WAN are presented. The work shows that a reliable TCP-aware link layer protocol will significantly improve the performance of TCP. The approaches such as selective acknowledgements and explicit loss notifications are also effective in performance improvement.

A loss differentiation algorithm for TCP flow control is proposed in [5]. The algorithm aims to improve TCP performance over wireless links. The approach is a pure transport layer solution. It examines the relationship between the slow start threshold value and the number of transmitted packets. A pair of decreased threshold value and an increased packet number indicates wireless losses. The algorithm distinguishes congestion losses from wireless losses in this way so that the TCP flow control algorithm can make a correct decision in sending rate adaptation. In AFRM, we use a different algorithm for loss differentiation, which is based on the observation of queueing loss correlation. However, this kind of approaches that are based on the correlation between sending rate and reported losses could also be integrated into AFRM to improve differentiation effectiveness.

Wireless Trace Analysis and Channel Modeling. In [62], a twelve-week trace of a building-wide wireless LAN is collected. The overall user behavior, overall network traffic, observed throughput, and traffic characteristics from a user point of view are

analyzed based on the collected trace. A number of interesting facts are discovered. For example, the peak throughput is usually caused by a single user/application.

The authors in [46] collect a large number of wireless packet traces in a 2Mbps WaveLAN. Data analysis shows that the packet error rate is not correlated to the transmission rate. The error rate increases with both the packet size and the distance between the sender and the receiver. A two-state Markov model is created for wireless error modeling. Since the resulting trace of the model dose not match the real trace very well, an improved two-state model with hybrid burst length distribution is proposed and evaluated. Results show that the improved model is more accurate than both the uniform loss model and the original two-state model. We adopt a similar approach to build the revised bursty loss model in the simulation framework

In [33], an algorithm is designed to model the time-varying effects on wireless channels. Packet traces in a GSM network are collected. The traces are divided into several stationary components based on the error distributions. The transitions between the components are modeled as a high order Markov process. The work shows that the proposed model is more accurate to represent channel characteristics such as burstiness and error distributions. Artificial traces based on this model are generated and compared to both EED (Even Error Distribution) and Gilbert traces. Results show that the traces of the proposed model are closer to the real traces than those of the other models.

The work in [76] claims that the two-state Markov model might be inadequate to represent some time-varying channels. However, computational complexity makes the models with larger number of states hard to use, and the quality of a channel model is not proportional to the number of states it uses. The work proposes a three-state model to

improve modeling accuracy with limited computational overhead. The model consists of one good state and two bad states. It is applied to the packet delay prediction in an ARQ/queuing system that operates over a fading channel. Results show that the proposed model is more accurate than a two-state model in such environment.

In [47], the performance of a FEC multicast approach is studied on a two-state Gilbert-Elliot model. An analytic model for the FEC protocol is built on the two-state channel model to evaluate the protocol performance. Numerical results show that FEC outperforms ARQ except in the case of very low bit error rates. The analysis also shows that there is no unique best code so that protocols should adapt the FEC code they use in response to the channel condition changes.

The work in [16] proposes an adaptive FEC approach in wireless networks. The FEC approach adapts the ratio of the redundant information in a packet according to the channel conditions. Several fixed or adaptive adaptation policies are proposed. A large number of wireless traces are collected and analyzed. The performances of the adaptation policies are evaluated and compared on the colleted traces.

Chapter 7

Conclusions

Summary. In this study we explored algorithms for reliable multicasting over wireless LANs, and we proposed and evaluated an adaptive FEC-based protocol. First we collected a number of wireless packet traces on an 11Mbps 802.11b wireless LAN. Based on those traces and other analytical models, we identified the major factors that affect the performance of a reliable multicast protocol in such environments. Then we proposed the AFRM protocol, which integrates several component algorithms to achieve high throughput WLANs. The protocol combines proactive FEC and NACK suppression in order to improve performance and the scalability by reducing the adverse effects of reverse traffic. In the protocol, the sender transmits packets by multicasting to all receivers in a multicast group. It also transmits a certain number of proactive FEC parity packets along with the data packets. Upon receiving a NACK, the sender responds by sending the requested number of parity packets, plus additional proactive parity packets. In both cases the number of proactive parity packets is calculated according to the estimated wireless channel conditions. To reduce congestion at the access point, the protocol requires the receivers to report packet loss bitmaps to the sender so that the sender can differentiate wireless propagation losses from queuing losses and apply appropriate congestion control algorithms. To further improve performance, the protocol

adopts both local and global NACK suppression algorithms, activating one of them when it is more effective than the other.

We evaluated the AFRM protocol and compared it with several other reliable multicast protocols through both experimentation and simulation. We studied the protocol behavior in a wireless LAN with up to three receivers. The experiments were conducted in both "normal" indoor conditions and in environments with artificial losses. For the simulations, we built a general-purpose network simulation framework that supports application code reuse. The IEEE 802.11 wireless LAN MAC protocols and multiple wireless channel loss models are also supported. We implemented a simple random loss model, a bursty loss model and a synthetic loss model. The framework also supports packet trace replay. We set up a simulated wireless network with up to 100 receiver stations. The protocol behavior was studied in this environment on all the loss models. Both experimental and simulation results show that the protocol has acceptable performance in small-scale network with low loss rate. Moreover, simulation results indicate that the proposed protocol outperforms the other protocols in large-scale networks or under situations of relatively high loss rate.

Future Work. A number of problems are open to further investigation. First, the channel condition estimation in AFRM can be improved. Due to the bursty and dynamic nature of wireless losses, it is almost impossible to derive a precise formula for loss prediction. A possible improvement is to take into consideration more history information when making the estimation, such as the long-term average loss rate and the performance of the previous proactive rate. Second, depending on the environment, one of the AFRM-A1 and AFRM-A2 algorithms exhibits better performance. A hybrid approach combining

these two algorithms might be able to achieve higher overall performance. Third, a more effective loss differentiation approach will be helpful to improve the flow control algorithm in AFRM protocols. Fourth, an integrated analytical model is needed to improve our understanding of the problem and help design a better adaptation algorithm. Fifth, the effects of other protocol parameters, such as the packet size, the packet group size and the NACK suppression windows size, on the performance of the protocols are worth studying. Finally, a more realistic and comprehensive loss model would be helpful to produce simulation results with higher confidence.

Bibliography

[1] G. Aben. "Throughput Performance of Wireless LANs Operating at 2.4 and 5 GHz". In *11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2000)*, pp. 190-195, London, Sep. 2000.

[2] F. Alagöz, D. Walters, A. Alrustamani, B. Vojcic, and R. Pickholtz. "On the Effects of Adaptive Forward Error Correction Mechanism in Direct Broadcast Satellite Networks". In *Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 1999.

[3] S. V. Appala, and J. R. Austen. "An Evaluation of Reliable Multicast Protocols". In *Proceedings of IEEE Southeastcon '99*. pp. 165-168. 1999.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links". In *Proceedings of ACM SIGCOMM '96*, pp. 256-269, 1996.

[5] Deepak Bansal, Anurag Chandra and Rajeev Shorey. "An Extension of TCP Flow Control Algorithm for Wireless Networks". In *IEEE International Conference on Personal Wireless Communication (ICPWC'99)*, Jaipur, India.

[6] V. Bharghavan et al. "MACAW – A Media Access Protocol for Wireless LAN's". In *ACM SIGCOMM*, pp. 212-225, Aug. 1994.

[7] G. Bianchi. "IEEE 802.11-Saturation Throughput Analysis". In *IEEE Communications Letters*, 2, 12, pp. 318 –320, Dec. 1998.

[8] G. Bianchi. "Performance Analysis of the IEEE 802.11 Distributed Coordination Function". *IEEE Journal on Selected Areas in Communications*, 18, 3, pp. 535-547, March 2000.

[9] R. Caceres, N.G.Duffield, J. Horowitz, and F. Towsley. "Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation". *IEEE Transactions on Information Theory*, Vol. 45, No. 7, pp. 2462-2480, Nov. 1999.

[10] F. Cali, M. Conti, and E. Gregori. "IEEE 802.11 Wireless LAN: Capacity Analysis and Protocol Enhancement". In *Proceedings of IEEE INFOCOM*, 1998.

[11] J. Callahan, and T. Montgomery. "Approaches to Verification and Validation of a Reliable Multicast Protocol". In *Proceeedings of the 1996 International Symposium on Software Testing and Analysis*, San Diego, CA, January 1996.

[12] Cisco Aironet 340 Series Base Station, http://www.cisco.com/univercd/cc/td/doc/pcat/340base.htm#BABFECDE

[13] Carlos de Morais Cordeiro, Judith Kelner e Djamel Sadok. "Establishing a Trade-off Between Unicast and Multicast Retransmission Modes for Reliable Multicast Protocols". *Eighth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems MASCOTS'2000*, August 2000, San Francisco CA, USA.

[14] Carlos de Morais Cordeiro, Judith Kelner, Djamel Sadok e Paulo Cunha. "An Enhanced Reliable Multicast Protocol for Wireless Environments". In *Proceedings of the IEEE Vehicular Technology Conference (VTC'2000).* pp. 975-982, September, Boston, USA, 2000.

[15] Peter Danzig. "Flow Control for Limited Buffer Multicast". In *IEEE Transactions on Software Engineering*, 20(1):1-12, January 1994.

[16] David A. Eckhardt, and Peter Steenkiste. "A Trace-based Evaluation of Adaptive Error Correction for a Wireless Local Area Network". In *Mobile Networks and Applications* 4, 4. (Dec 1999) , pp. 273-287.

[17] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing". In *Proceedings of ACM SIGCOMM'95*, pp 342--356. Cambridge, MA, USA, 28 August--1 September 1995.

[18] F. C. Fujiwara, M. Kasahara, K. Yamashita, and T. Namekawa. "Evaluations of Error-control Techniques in Both Independent and Dependent Error Channles". *IEEE Transaction on Communications*, COM-26(6):785-793, 1978.

[19] J. Gemmell, E. Schooler, and R. Kermode. "A Scalable Multicast Architecture for One-to-many Telepresentations", *IEEE International Conference on Multimedia Computing Systems (ICMCS '98)*, pp. 128-139, 1998.

[20] E. N. Gilbert, "Capacity of a Burst-noise Channel", *Bell System Tech. Journal*, 39:1253-1265, September 1960.

[21] S. Ha, K. Lee, amd V. Bharghavan. "A Simple Mechanism for Improving the Throughput of Reliable Multicast". In *Proceedings of ICCCN '99 (the Eighth International Conference on Computer Communications and Networks),* Boston, MA, October 1999.

[22] A. Heindl, and R. German. "The Impact of Backoff, EIFS, and Beacons on the Performance of IEEE 802.11 Wireless LANs". In *Proceeding of 4th IEEE International Computer Performance and Dependability Symposium*, pp. 103-112, Chicago, IL, March 2000.

[23] Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton. "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation". In *Proceedings of SIGCOMM'95*, August 1995.

[24] C. Huitema. "The Case for Packet Level FEC". In *Proceedings of IFIP 5th International Workshop on Protocols for High Speed Networks* (PfHSN'96), pp. 110--120, INRIA, Sophia Antipolis, FRANCE, October 1996.

[25] IEEE 802.11, 1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology -- Telecommunications and Information Exchange between Systems -- Local and Metropolitan Area Network -- Specific Requirements -- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. http://standards.ieee.org/getieee802/802.11.html

[26] R. Jain, and K. Ramakrishnan. "Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals, and Methodology". In *Proceedings of IEEE Computer Networking Symposium*, Washington, D.C., pp.134-143, April 1988.

[27] A. Kamerman, and G. Aben. "Net Throughput with IEEE 802.11 Wireless LANs". In *IEEE Wireless Communications and Networking Confernce 2000 (WCNC. 2000)*, Vol 2, pp.747 -752, 2000.

[28] P. Karn. "MACA – A New Channel Access Method for Packet Radio". In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pp. 134-140, Apr. 1990.

[29] R. Kermode. "Scoped Hybrid Automatic Repeat Request with Forward Error Correction (SHARQFEC)". *ACM SIGCOMM 98*, September 1998, Vancouver, Canada.

[30] S. K. Kesera, G. Hjalmtysson, D. F. Towsley, and J.F. Kurose. "Scalable Reliable Multicast Using Multiple Multicast Channels". In *IEEE/ACM Trans. Networking*, vol. 8. pp 294-309, Jun. 2000.

[31] L. Kleinrock, and F.A. Tobagi. "Packet Switching in Radio Channels: Part I – Carrier Sense Multiple Access Modes and Their Throughput-Delay Characteristics". In *IEEE Transactions on Communications*, vol. COM-23, no. 12, pp. 1400-1416, 1975.

[32] A. Koifman and S. Zabele. "RAMP: A Reliable Adaptive Multicast Protocol". In *Proceedings of IEEE INFOCOM*, pages 1442--1451, March 1996.

[33] Almudena Konrad, Ben Y. Zhao, Anthony D. Joseph, and Reiner Ludwig. "A Markov-based Channel Model Algorithm for Wireless Networks". In *MSWIM'01*, 2001.

[34] J. Korhonen, "HIPERLAN/2", Department of Computer Science and Engineering, Helsinki University of Technology, Jan. 1999, http://www.tml.hut.fi/Studies/Tik-110.300/1999/Essays/hiperlan2.html

[35] D. R. Kosiur. "IP Multicasting: The Complete Guide to Interactive Corporate Networks". May 1998, John Wiley & Sons.

[36] V. Kumar. "MBone: Interactive Multimedia on the Internet". Indianapolis, IN: New Riders, 1996.

[37] J. Kuri, and S. Kasera. "Reliable Multicast in Multi-access Wireless LANs", In *IEEE INFOCOM'99*, March 1999.

[38] L. Lehman, S. Garland, and D. Tennenhouse. "Active Reliable Multicast". In *IEEE INFOCOM'98*. IEEE, March 1998.

[39] Philip K. McKinley, and Arun P. Mani. "An Experimental Study of Adaptive Forward Error Correction for Wireless Collaborative Computing", In *Proceedings of the 2001 IEEE Symposium on Applications and the Internet (SAINT-2001)*, San Diego-Mission Valley, California, January 2001.

[40] P. K. Mckinley, C. Tang, and A. P. Mani, "A Study of Adaptive Forward Error Correction for Wireless Collaborative Computing", *IEEE Transactions on Parallel and Distributed Systems Special Issue on Mobile Computing*, 2002.

[41] Mesquite Software Inc., "CSIM 18 -- A Low Cost Development Toolkit for Simulation and Modeling", http://www.mesquite.com.

[42] K. Miller, K. Robertson, A. Tweedly, and M. White. "StarBurst Multicast File Transfer Protocol (MFTP) Specification--An Internet Draft". Available from http://www.ietf.org/internet-drafts/draft-miller-mftp-spec-03.txt.

[43] W. L. Miller, R. M. Ollerton, A. Shum, and C. J. Warner. "Proactive FEC-Based Forwarding for the Collaborative Reliable Multicast Protocol". *EUROCOMM 2000. Information Systems for Enhanced Public Safety and Security*. pp. 269 –273, IEEE/AFCEA , 2000.

[44] M. Mosko, and J.J. Garcia-Luna-Aceves. "An Analysis of Packet Loss Correlation in FEC-Enhanced Multicast Trees". In *Proceedings of 8th IEEE International Conference on Network Protocols (ICNP 2000)*, Osaka University Convention Center, Osaka, Japan, November 14 - 17, 2000.

[45] M. Natkaniec and A. Pach. "An Analysis of the Backoff Mechanism Used in IEEE 802.11 Networks". In *Proceeding of ISCC*, 2000.

[46] Giao T. Nguyen, Randy Katz, and Brian Noble. "A Trace-based Approach for Modeling Wireless Channel Behavior". In *Proceedings of the Winter Simulation Conference*, pages 597--604, December 1996.

[47] Neda Nikaein, and Christian Bonnet. "On the Performance of FEC for Multicast Communication on a Fading Channel". In *proceedings of International Conference on Telecommunications ICT*, Acapulco, Mexico, May 2000.

[48] Neda Nikaein, Houda Labiod, and Christian Bonnet. "MA-FEC: a QoS-Based Adaptive FEC for Multicast Communication in Wireless Networks". In *Proceedings of International Conference on Communications ICC*, New Orleans, USA, June 2000.

[49] J. Nonnemacher, and E. Biersack. "Scalable Feedback for Large Groups", *IEEE/ACM Transactions on Networking*, June 1999.

[50] J. Nonnenmacher, E. Biersack, and D. Towsley. "Parity-Based Loss Recovery for Reliable Multicast Transmission". Technical Report 97-17, Dept. of Computer Science, U. Massachusetts, March 1997.

[51] J. Nonnenmacher, M. Lacher, M. Jung, E. Biersack, and G. Carle. "How Bad is Reliable Multicast without Local Recovery?". In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 1998.

[52] Christos Papadopoulos, Guru Parulkar, and George Varghese. "An Error Control Scheme for Large-scale Multicast Applications". In *Proceedings of IEEE INFOCOM*, 1998.

[53] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya. "Reliable Multicast Transport Protocol (RMTP)". *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 407--21, Apr. 1997.

[54] S. Ramakrishnan, and B. N. Jain "A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LAN". In *Proceedings of IEEE INFOCOM*, April 1997.

[55] Victor M. Ramos, Chadi Barakat and Eitan Altman. "Queueing Analysis of Simple FEC Schemes for IP Telephony". In *Proceedings of IEEE INFOCOM*, Anchorage, April, 2001.

[56] L. Rizzo. "On the Feasibility of Software FEC", DEIT Tech Report, http://www.iet.unipi.it/~luigi/softfec.ps, Jan 1997.

[57] L.Rizzo, and L. Vicisano. "RMDP: an FEC-based Reliable Multicast Protocol for Wireless Environments". *ACM Mobile Computing and Communications Review*, **2**, 2, April 1998.

[58] D. Rubenstein, S. Kasera, J. Kurose, and D. Towsley. "Improving Reliable Multicast Using Active Parity Encoding Services (APES)". In *Proceedings of IEEE INFOCOM*, 1999.

[59] A. Shiozaki, K. Okuono, K. Suzuki, and T. Segawa. "A Hybrid ARQ Scheme with Adaptive Forward Error Correction for Satellite Communications". *IEEE Transactions on Communications*, vol. 39, pp. 482-484, April 1991.

[60] T. Speakman, R. Edmonstone, D. Farinacci, S. Lin, A. Tweedly, L. Vicisano, and J. Gemmell. "PGM Reliable Transport Protocol Specification". INTERNET-DRAFT, June 1999.

[61] W. T. Strayer. "Xpress Transport Protocol Specification Revision 4.0". XTP Forum, March 1995.

[62] Diane Tang, and Mary Baker. "Analysis of a Local-Area Wireless Network". In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, USA, Aug. 2000.

[63] K. Tang, and M. Gerla. "MAC Layer Broadcast Support in 802.11 Wireless Networks". In *Proceedings of IEEE MILCOM*, pp. 544-548, Oct. 2000.

[64] K. Tang, and M. Gerla. "MAC Reliable Broadcast in Ad Hoc Networks". In *Proceedings of IEEE MILCOM*, McLean, VA., Oct. 2001.

[65] K. Tang, and M. Gerla. "Random Access MAC for Efficient Broadcast Support in Ad Hoc Networks". In *IEEE WCNC*, Chicago, IL., Sep. 2000.

[66] D. Towsley, J. Kurose, and S. Pingali. "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols". *IEEE Journal on Selected Areas in Communications,* April 1997.

[67] Brian Whetten, and Gursel Taskale. "An Overview of Reliable Multicast Transport Protocol II". In *IEEE Network*, pp.37-47, January/February 2000.

[68] T. Wilkinson. "HIPERLAN – An Air Interface Designed for Multi-Media". Hewlett Packard Laboratories, 1995, http://www.hpl.hp.com/techreports/95/HPL-95-47.pdf.

[69] Robin Wright. "Design and Implementation of a Reliable Multicast Protocol". Master's Thesis, Dept. of Computer Science, Michigan State University, October 1998.

[70] Y. Xu, and T. Zhang. "An Adaptive Redundancy Technique for Wireless Indoor Multicasting". In *Proceedings of IEEE Symposium on Computers and Communications , 2000 (ISCC 2000)*, Antibes-Juan les Pins, France, 4-6 July, 2000.

[71] George Xylomenos and George C. Polyzos. "Internet Protocol Performance over Network with Wireless Links". In *IEEE Network*, 13, 4, pp. 55-63, 1999.

[72] G. Xylomenos and G. C. Polyzos. "Internet Wireless Link Performance". Technical Report #9801, Center for Wireless Communications, University of California, San Diego, La Jolla, CA, USA, January 1998.

[73] G.Xylomenos, and G.Polyzos. "TCP and UDP Performance over a Wireless LAN". In *Proceedings of the IEEE INFOCOM*, 1999.

[74] Rajendra Yavatkar, James Griffioen, and Madhu Sudan. "A Reliable Dissemination Protocol for Interactive Collaborative Applications". In *Proceedings of the ACM Multimedia'95 Conference*, November 1995.

[75] S. Yoon. "Reliable Multicast Considering the Temporal Dependence in Packet Loss". *INET 2001*, Stockholm, Sweden, 2001.

[76] M. Zorzi, and R.R. Rao. "On Channel Modeling for Delay Analysis of Packet Communications over Wireless Links". In *36th Annual Allerton Conference*, Allerton House, Monticello, IL, Sep. 1998.