# EVALUATION AND IMPLEMENTATION OF PROTOCOLS IN THE LOCAL AREA NETWORK TESTBED ENVIRONMENT

Benjamin Lewis Barnett, III

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

TR-89-35 November 1989

## Abstract

This thesis presents the results of the Local Area Network Testbed experience. The design of the testbed, the results of several experiments and the results of a formal protocol analysis are included. Three data link layer protocols for Carrier Sense Multiple Access with Collision Detection (CSMA/CD) bus networks were implemented in the testbed. The performance of these three protocols under several different artificial workloads is compared. The three protocols were the commercially available Ethernet, the Enet II protocol proposed by Molloy, and the Virtual Time CSMA/CD (VTCSMA/CD) protocol proposed by Molle. The three protocols represent three fundamentally different approaches to handling collisions among users of a broadcast channel. Ethernet randomizes retransmission attempts for the conflicting packets in an attempt to minimize the likelihood of successive collisions. Enet II uses a probabilistic algorithm to schedule the retransmissions of conflicting packets to resolve the collision. VTCSMA/CD uses a technique which reduces the initial likelihood of collisions. Enet II is shown to have significantly better variance of delay than Ethernet. VTCSMA/CD has the best variance of delay of the three protocols due to the success of its collision avoidance method.

The implementation of Enet II demonstrates that techniques usually reserved for slotted networks can be beneficially employed on their unslotted counterparts. To investigate the adaptation of slotted protocols to unslotted use, a well known slotted Collision Resolution Protocol (CRP), the Gallager First-Come, First-Served (FCFS) protocol, is adapted to unslotted operation and proven to have bounded delay. A second adaptation of the protocol which responds to collisions differently is shown to deadlock. Deadlock detection and recovery methods are presented. A new CRP based on the deadlock recovery method and using information about the location of colliding stations is proposed.

**Keywords**: Broadcast Bus Protocols, Protocol performance measurement, Protocol Verification.

# EVALUATION AND IMPLEMENTATION OF PROTOCOLS IN THE LOCAL AREA NETWORK TESTBED ENVIRONMENT

by

BENJAMIN LEWIS BARNETT III, B.S., M.S.C.S.

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December, 1989

## ACKNOWLEDGEMENTS

I am grateful for the advice and support of the members of my committee. Special thanks go to my supervisor, Michael Molloy, for his perseverence and inspiration.

This dissertation and the work it describes owe more than I am able to express to my parents. Their support and confidence during this effort were continuous and unwavering.

I'm very grateful to Pradeep Jain for many helpful discussions.

Thanks to Chris Edmondson-Yurkanan, who kept an eye on me.

Many thanks to the Unix and Shop staff of the University of Texas Department of Computer Sciences; Fletcher Mattox, James Johnson, Ron Vasey, Pat Horne and C. W. Branch provided valuable help during the course of the experiments. Thanks to Boyd Merworth for invaluable help with text formatting.

Thanks to Brad Blumenthal, who kept me sane through the ordeal, and to my officemates, A. T. Campbell, Sampath Rangarajan, K. R. Subramanian, and Ramakrishna Thurimella.

My deepest gratitude goes to Rebekah Lane, who gave me the impetus for the final push, and who put up with me while it was going on.

This research was supported in part by grant #MCS8122039 from the National Science Foundation.

Benjamin Lewis Barnett III

The University of Texas at Austin
August, 1989

# ABSTRACT

This thesis presents the results of the Local Area Network Testbed experience. The design of the testbed, the results of several experiments and the results of a formal protocol analysis are included. Three data link layer protocols for Carrier Sense Multiple Access with Collision Detection (CSMA/CD) bus networks were implemented in the testbed. The performance of these three protocols under several different artificial workloads is compared. The three protocols were the commercially available Ethernet, the Enet II protocol proposed by Molloy, and the Virtual Time CSMA/CD (VTCSMA/CD) protocol proposed by Molle. The three protocols represent three fundamentally different approaches to handling collisions among users of a broadcast channel. Ethernet randomizes retransmission attempts for the conflicting packets in an attempt to minimize the likelihood of successive collisions. Enet II uses a probabilistic algorithm to schedule the retransmissions of conflicting packets to resolve the collision. VTCSMA/CD uses a technique which reduces the initial likelihood of collisions. Enet II is shown to have significantly better variance of delay than Ethernet. VTCSMA/CD has the best variance of delay of the three protocols due to the success of its collision avoidance method.

The implementation of Enet II demonstrates that techniques usually reserved for slotted networks can be beneficially employed on their unslotted counterparts. To investigate the adaptation of slotted protocols to unslotted use, a well known slotted Collision Resolution Protocol (CRP), the Gallager First-Come, First-Served (FCFS) protocol, is adapted to unslotted operation and proven to have bounded delay. A second adaptation of the protocol which responds to collisions differently is shown to deadlock. Deadlock detection and recovery methods are presented. A new CRP based on the deadlock recovery method and using information about the location of colliding stations is proposed.

## Table of Contents

## List of Tables

# List of Figures

# Chapter 1

# Introduction

This dissertation presents results in three main areas; the construction and use of performance testbeds for Local Area Networks (LANs), the implementation and measurement of the performance of two previously unimplemented protocols, and the adaptation of slotted collision resolution algorithms for use on unslotted networks. Few implementation and performance measurement studies of data link level protocols for LANs exist, primarily due to the early success of the Ethernet system [MB76]. Some proposed protocols have been shown analytically to have desirable delay and throughput characteristics. However, analytic models of such protocols often contain simplifying assumptions which do not represent actual conditions in networks. For this reason, it is vital that the research community expand empirical studies of actual implementations of proposed protocols. Likewise, little attention has been paid to the possibility of adapting collision resolution techniques for use on the unslotted contention bus common to LANs, in part due to the difficulty in ensuring the correctness of such adapted protocols. Many protocols have been formulated for slotted networks because it is less difficult to analyze the correctness and performance of slotted protocols. Results presented here demonstrate that there is a correspondence between one slotted collision resolution protocol and an unslotted adaptation of the protocol. This presents the possibility of designing protocols for slotted networks and deriving correct unslotted versions for use on simpler (from a hardware viewpoint) unslotted networks.

The Local Area Network Testbed (LANT) is a dedicated network of computers equipped with network interfaces and load generation and performance measurement software which is used to implement examples of Collision Resolution and Collision Avoidance Protocols (CRPs and CAPs, respectively) for LANs and to compare these protocols to the Ethernet protocol. The Ethernet protocol employs a random backoff scheme for collision handling. The Enet II protocol [Moll85] and the Virtual Time Carrier Sense Multiple Access with Collision Detection (VTCSMA/CD) [MK85] protocol are also implemented and measured. The Enet II protocol uses a probabilistic method for subdividing the set of colliding stations until a subset is found that contains only one active host; at this point a successful transmission will result. This effect is similar to that of slotted CRPs. The VTCSMA/CD protocol avoids collisions by using a second clock to spread packet transmission attempts with respect to the end of busy periods on the network. In the measurements presented here, both protocols display significant advantages over Ethernet when loads are heavy. This result had been predicted, but could only be verified by implementation and comparison using a system such as the LANT.

The performance measurement results for Enet II demonstrate the success of resolution techniques usually associated with slotted systems on an unslotted network. Other results on adapting slotted CRPs to work on unslotted networks are also presented. A specification for the FCFS algorithm [Gall78] on an unslotted network is presented and formally proven to have bounded delay in networks where identical arrival times are not allowed. A version of the protocol which makes different assumptions about the treatment of collisions is formally demonstrated to deadlock. Methods of detecting and resolving such deadlocks are presented. These methods of handling deadlock are applicable to the initial unslotted version of the protocol, which can also deadlock if time is kept as an integer value. A new unslotted CRP related to the deadlock resolution method is presented. This protocol takes advantage of positional information about stations to reduce the length of collision clear time and idle steps, thus reducing the overall time for resolving collisions.

This chapter contains an overview on the area of local networking, describing the explosive growth in the area in recent years; some background information on the evaluation of network performance; a summary of the results presented here; and an outline of the organization of the dissertation.

## 1.1. Overview

It would be a gross understatement to say that LAN use has grown rapidly in the years since the introduction of Ethernet. Growth in the LAN market has had an undeniably explosive character. In the earliest reported traffic characterization study of Ethernet [SH80] the average utilization level was under 1%, and the principal uses for the network were file transfers, remote database queries, and terminal access to timesharing hosts. Since that time, LAN connectivity has gone from being something of a luxury to forming a basic foundation for computational activity of all kinds. As evidence for this claim, consider the fact that nearly all of the popular workstations on the market today come from the manufacturer with an Ethernet interface already installed and operating system software to support its use. The current extreme of this line of developments is the diskless workstation, which depends on the network for all aspects of its operation except CPU cycles and physical memory access; a diskless workstation boots over the network from a remote machine which serves as a bootstrap server, has its virtual memory paging space and file storage on a remote file server, depends on other remote servers for communication namespace resolution, timekeeping, print spooling, system utilities, and electronic mail service. Network congestion leads to a decrease in response time for the user of such a workstation, and a network outage often means the complete suspension of computing activities.

The increasing importance of network connectivity has led to rapid and sometimes poorly planned growth in individual network installations as well. LANs tend to grow organically as first one working group and then another acquires equipment and a need for connections to remote resources. There are many difficulties in keeping a large network functioning, such as ensuring that the specification limits for segment length and total host count are maintained, detecting malfunctioning hardware, managing congestion, and even maintaining the physical integrity of the network cable. (For a discussion of experiences during the evolution of the University of Texas network, see [Moll86].) Manufacturers have responded to such problems with various network management tools ranging from special purpose computers for network monitoring and diagnosis [Net87] to bridges and routers for further subdividing LANs. This indicates that many installations are reaching the design limits of the current generation of LAN protocols.

The impact of the growth in LAN usage on research in the area is twofold. First, building tools to manage the complexity of current networks requires a well developed understanding of the behavior of the underlying protocols and their response to a wide variety of traffic loads. Second, as installations grow toward the design limits of current networks and congestion becomes a more frequent problem, the question of how the next generation of networks can address these limitations arises. The answers to this question may encompass more efficient protocols for networks similar to existing ones as well as networks based on different hardware configurations for which existing protocols may be inappropriate. Good examples of the latter case are the 100 Mbps networks now being planned. In these networks, the higher bandwidth and longer propagation times make the ratio of packet transmission time to collision detect time too low for CSMA/CD protocols to be practicable. Thus, it is evident that there is a critical need for greater understanding of the performance of existing protocols and for the development of new protocols to remedy the deficiencies of the current generation of algorithms and meet the needs of the next generation of network hardware.

A large number of alternatives to the Ethernet protocol have been proposed and analyzed in the literature, but very few have actually been implemented and empirically compared to the popular commercial network protocol. The reasons for this are fourfold.

First, Ethernet provided more bandwidth than early installations could effectively utilize. As noted above, one of the largest installations existing in 1980 experienced a utilization of only one or two percent on the average. The measurements of normal traffic on the UT network presented in Chapter 4 convincingly demonstrate that this is no longer the case. So many installations have suffered from such severe congestion on specification-compliant networks that a whole new market has sprung up for devices to further subdivide networks in an intelligent way.

Second, actual implementation of alternatives in a fashion that would provide meaningful data for comparison is difficult and expensive. From the very beginning, the functions of the Ethernet protocol were built into the network interface hardware, making it impossible to instrument them for performance measurements or to implement alternative protocols without expensive, specially fabricated hardware. In the meantime, this situation has grown only worse, not better, particularly in view of the growing practice of standardizing protocols for a particular type of network before the first interface is ever built and tested, let alone subjected to a real network environment.

Third, until recently the bottleneck in network performance in relatively small installations was clearly in the higher level protocols such as TCP. However, since 1980, many local network installations have expanded to the point where bandwidth is becoming a scarce commodity, while at the same time, the efficiency of higher level protocols has improved. In a recent study, Jacobson reports a TCP sender-receiver pair generating 8.9 megabits per second throughput using an experimental version of the BSD networking software [Jaco88].

Fourth, many of the more interesting alternative protocols are defined in terms of a time slotted medium, making their use in the unsynchronized (unslotted) CSMA/CD environment problematic. Though these protocols have many desirable performance properties, implementation of the algorithms without explicit slotting is very difficult.

## 1.2. Background

The question is often asked, " Why bother with network [protocols/performance]? Ethernet works fine." The preceding section explains that while Ethernet may have " worked fine" in the past, computing and communications needs are not static. Understanding the performance of network protocols can help us manage their use more effectively and guide the design of protocols with better performance characteristics. As Ferrari notes in the preface to his book, *Computer Systems Performance Evaluation* [Ferr78], most engineering fields do not consider performance evaluation to be a subject separate from other design considerations; rather, it is an integrated aspect of system design. Succinct and accurate evaluations of the performance of computer systems are vital to the entire computer community, including system designers, purchasing agents, systems programmers, systems administrators and computer users. As the discussion in the previous section makes clear, this is even more important in the rapidly growing and changing arena of computer networking. Often it is the case that competing products or protocols will offer much the same functionality, leaving the performance of the alternatives as the only basis for choosing between them. A particularly apt example is the choice between CSMA/CD protocols and ring protocols; the functionality of the two alternatives is similar, but the performance characteristics of the protocols make them appropriate for applications with differing requirements.

Formal analyses of the performance of many varieties of CSMA and CSMA/CD protocols have been published, e.g. [CL83], [FW81], [KT75], [Lam80], [MB76], [MSV87], [TK85] and [TH80]. In order to keep the analyses tractable, many assumptions must be made concerning the behavior of packet sources, the characteristics of the protocols, and the behavior of the medium. In each of the analyses, some or all of the following assumptions are made:

- slotted communication medium
- infinite population of stations
- well behaved aggregate packet arrival rates (i.e., arrivals combined with retransmissions form a poisson process)
- zero cost acknowledgements
- fixed propagation delays (the star topology assumption)
- fixed collision detect times
- fixed or limited packet sizes

Each of these assumptions affects the accuracy of modeling and does not necessarily reflect conditions on real networks. The Xerox PARC Ethernet experiments [SH80] demonstrated that in actual use, arrivals are not exponential in nature, and packet sizes display a distinctly bimodal distribution. The latter point has been addressed in some of the analyses [TH80], but the former has not. It is also clearly the case that there is never an infinite population of stations.

Many protocols have been proposed for communication on Local Area Networks, but few have been implemented. The protocols range from variants on the Ethernet backoff scheme such as [FW81], [AS84], and [TT77], to collision resolution approaches such as [Moll84], [Molle83], [Mass80], [Gall78], [Cape79], and [TV82], to collision minimization strategies in [MK85], [TH80], and [KK81]. The performance of these protocols has either been approximated using the techniques mentioned above or in some cases ignored, but has never actually been observed.

Of these protocols only Ethernet is in widespread use. While the analytic models and simulation studies of other protocols may give a feel for their performance, these studies can only approximate conditions of actual use. The true test of a new network protocol is in implementation and empirical evaluation. The LANT provides an environment where these protocols can be implemented, debugged, and tested with a minimum of effort. Often an elegantly designed protocol may have little utility due to implementation difficulties. The LANT provides an environment in which these difficulties can be discovered and possibly remedied. The LANT configuration is capable of generating heavy loads on a 10 Mbps Ethernet, providing a rare opportunity to observe the behavior of the Ethernet protocol and other protocols under severe loads. In addition, the experiment configuration software allows a wide range of load conditions to be specified. While heavy loading is an interesting condition under which to study Ethernet, for other protocols interesting behavior may occur under other conditions. While the difficulties in accurately modeling and simulating protocol behavior and performance remain, direct testing provides the only accurate comparison between proposed approaches.

There are two approaches to measuring traffic on local networks. The most prevalent approach, represented by [SH80], [Nabi84], [AKKPC86], [Abra87] and the various commercial network monitoring products, measures the overall throughput characteristics of the network from the point of view of a promiscuous receiver. This type of measurement provides information on the overall efficiency and stability of a network protocol. An alternate approach, seen in [Gons85], [BMK88] and in the present work, is to monitor the performance of the protocol from the point of view of the individual *user*, in this case, the host attached to the network. From this perspective, the quantities of interest are the throughput achieved by the host and the delay experienced due to the access mechanism. The latter approach is obviously the more difficult, requiring instrumentation of the network interface itself. At the data link layer, where functionality resides in a hardware network interface, this approach is impossible unless the interface has been specifically designed to support such measurements.

In the testbed described here measuring the performance of the network as viewed by the user is facilitated by the existence of Ethernet interfaces which implemented significant portions of the protocol as interrupt service routines in an operating system device driver. A driver for this device was available with the 4.3BSD UNIX† operating system, which allowed modifications to be made to instrument the transmission process and to implement other protocols. The LANT provided a dedicated group of machines with sufficient computing power to drive such implementations. This combination allowed the implementation of several interesting protocols in a uniform environment without incurring the expense and effort involved in the design and fabrication of custom hardware interfaces for each of the protocols.

## 1.3. Results

Results are presented on the performance of two alternatives to Ethernet, the Enet II protocol and the VTCSMA/CD protocol. The Enet II protocol uses probabilistic techniques to resolve collisions on an asynchronous CSMA/CD bus. VTCSMA/CD is a collision avoidance protocol which uses a *virtual* clock running only during idle periods at a rate higher than real time to determine when transmissions will be attempted. These protocols take two different approaches to the multiple access problem for broadcast bus networks; Enet II is a true collision resolution protocol in the style of the tree and window protocols, while VTCSMA/CD reduces the probability of collisions initially, consequently avoiding the additional delay of rescheduling them for later transmission. These two protocols were implemented and tested on the LANT. Ethernet was also tested on the same equipment, allowing a comparison of the popular commercial network and two alternatives. Ethernet takes a third approach to collision handling by randomly rescheduling the transmission time of packets involved in collisions.

† UNIX is a trademark of Bell Laboratories.

The experiments revealed a consistent relationship among the three protocols independent of the load driving the network. While Ethernet enjoyed a slight advantage in peak throughput, both Enet II and VTCSMA/CD achieved higher stable throughputs under continuous overload conditions. Enet II also displayed a great improvement over Ethernet in the variance of the delay experienced by packets in their transmission attempts even though both protocols experienced similar collision rates at all offered load levels. VTCSMA/CD experienced the lowest variance of delay and the lowest average delay for most offered loads due to a much lower collision rate. These results strongly demonstrate the advantages of these two protocols. Enet II reduces the variation in delay by imposing a regular control structure on the retransmission of packets involved in collisions. VTCSMA/CD achieves the reduction by greatly reducing the rate of initial collisions.

The Enet II protocol incorporates a technique which can be used for unslotted implementations of protocols such as the Gallager First-Come, First-Served (FCFS) protocol [Gall78]. The FCFS protocol is a slotted CRP that uses packet arrival times as the criterion for subdivision of the set of colliding stations. The most difficult problem in making the transition from slotted to unslotted media occurs in protocols like FCFS which must be able to determine the end of an idle step. Unless clocks at all the stations in the network are perfectly synchronized, timers cannot be used. Enet II avoids this problem by causing intentional collisions, which are detectable by all stations, to signal the end of the idle step in the algorithm. There are other complications in the adaptation from slotted to unslotted operation, not the least of which is guaranteeing the correct behavior of the resulting unslotted protocols. An extension to the formal system of [JL88] that allows more natural modeling of collision detection is presented. Results concerning the behavior of the broadcast bus in the presence of collisions are developed. These results are used to prove that delay is bounded for an asynchronous version of the FCFS protocol. A demonstration is also given that deadlocks can occur in implementations that take an aggressive approach to interpreting network events. The difficulty of recovering from such deadlocks as opposed to treating network events more conservatively and thus avoiding the deadlocks is also considered.

These results lay a groundwork for investigating several interesting issues in protocol design. Time slotted protocols are generally easier to prove correct and their performance is easier to determine than analogous asynchronous protocols. However, the asynchronous environment of CSMA/CD networks is attractive due to its simplicity and efficiency under a wide range of loads. The results on adapting slotted protocols to unslotted media presented here demonstrate that in the case of the FCFS protocol there is a correspondence between the correct slotted protocol and a correct asynchronous implementations if certain guidelines are followed. This observation could allow protocols to be designed in the less hostile slotted paradigm, and then converted to operate in more practical unslotted networks.

## 1.4. Summary of Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 discusses the previous research results in the area of local network measurement and broadcast protocol verification. Chapter 3 details the design and implementation of the Local Area Network Testbed. This chapter includes descriptions of both the hardware and the software for network monitoring, experiment configuration, experiment management, and the implementation of the Ethernet, Enet II and VTCSMA/CD protocols. Chapter 4 discusses the implementation of the transmission monitoring instrumentation and the design of the measurement experiments. This chapter includes results from a traffic characterization study of the University of Texas campus ethernet using LANT software. Chapter 5 discusses the results of the performance studies of the three protocols. Chapter 6 presents results on the correctness of an asynchronous version of the Gallager FCFS protocol and discusses the application of the conversion and analysis techniques to other slotted protocols. Chapter 7 is a summary of the results presented in the preceding chapters. Appendices are included containing programming information and instruction for the use of the various software packages which comprise the LANT suite.

# Chapter 2

# Previous Research

The topics presented in this dissertation fall into three main categories; first, the design and construction of the LANT; second, the measurement and comparison of data link level protocols for LANs; and third, the adaptation of slotted collision resolution algorithms for use on unslotted networks. This chapter reviews the relevant literature in these areas.

Several experiments similar to the LANT experiments have been performed. In all cases, these experiments were devoted to the investigation of a particular protocol, not to the development of generally useful tools for network performance experimentation. The LANT environment consists of a set of tools which may be applied to many protocols. It also provides greater load generating flexibility than any of the previous systems.

In Chapter 6, results are presented on the correct adaptation of a slotted Collision Resolution Protocol (CRP) for use on unslotted networks. In order to prove the correctness of the adapted CRP, a model was needed which captures essential characteristics of unslotted broadcast protocols such as the propagation of signals and relative orderings of network events. Though the literature on modeling point-to-point networks for verification purposes is rich, very little research exists on such modeling for broadcast networks due to the essential part that timing relationships play in the correct operation of such protocols.

## 2.1. Measurement Studies

Since the existence of a testbed usually implies the existence of measurements and vice versa, the literature on these two areas coincides. Measurement studies relevant to the current work have been carried out by three organizations, the Xerox Palo Alto Research Center (PARC), the National Bureau of Standards (NBS), and the Digital Equipment Corporation (DEC). The Xerox PARC experiments and the DEC experiments considered various aspects of the performance of the Ethernet Protocol. The National Bureau of Standards experiments investigated the performance of an NBS developed CSMA/CD protocol.

### 2.1.1. Xerox PARC Experiments

Two sets of experiments have been conducted on the Ethernet protocol at the Xerox PARC. The first set was conducted by Shoch and Hupp [SH80] on a 3 Mbps experimental network. The second set was conducted by Gonsalves [Gons85] on both 3 and 10 Mbps networks, using a modified version of the software written for the earlier experiments.

#### Shoch and Hupp

This work confirms the throughput capacity, stability and fairness of Ethernet under high loads and characterizes network traffic under normal conditions. No attempt was made to measure the delay suffered by packets as a result of queueing on the transmitting hosts. The experimental environment consisted of a 550 meter cable with 120 hosts connected. Important observations on the normal operating load of the installation were collected. These observations showed that the average utilization of the network ranged from 0.60% to 0.84%, with a one second peak utilization of 37%. Figure 2.1 shows the observed packet length distribution. Approximately 80% of the observed packets were short, consisting of terminal traffic and acknowledgments, with most of the remainder being large packets associated with file transfers. The large packets accounted for the majority of actual bytes transmitted. The observations also indicated that 99.18% packets were transmitted successfully on the first try, 0.79% were delayed due to deference, and 0.03% suffered collisions. The measured interpacket arrival time is shown in Figure 2.2. The interpacket delay of normal traffic did not strongly resemble an exponential distribution. The study also pointed

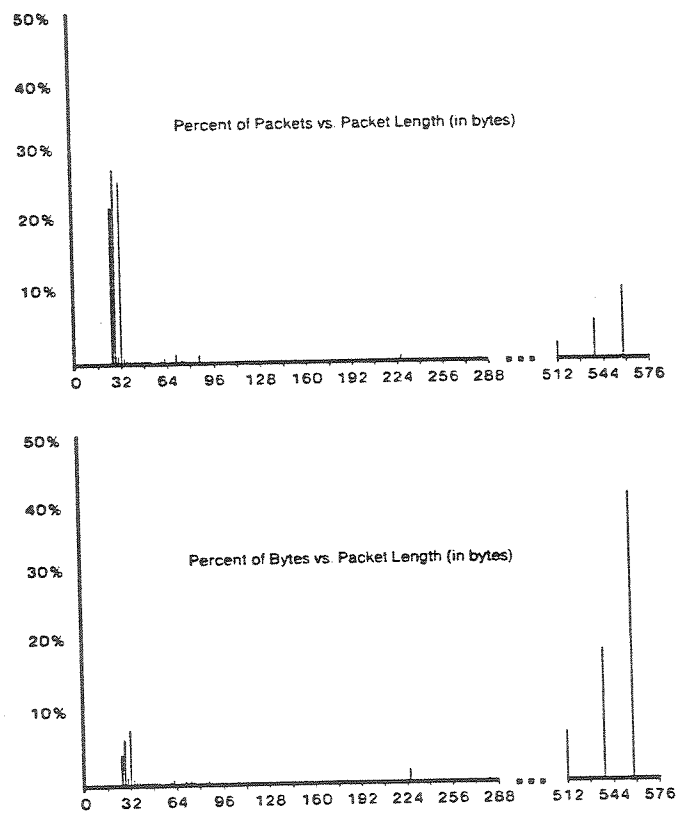


**Figure 2.1:** Measured Ethernet packet length distribution ([SH80], Figure 4).

out the influence that some specific applications (transactions with servers and a multi-player game were cited) had on the distribution.

The network was also subjected to heavy loads using artificial traffic generators. The experiment management software searched the network for idle hosts and loaded them with generator processes. Each generator produced a specified fraction of the available capacity. Though the artificial load experiments were conducted during low utilization periods, they were not completely free from the influence of extraneous network traffic. Figure 2.3 shows the effects on utilization of increasing the number of continuously queued hosts. This

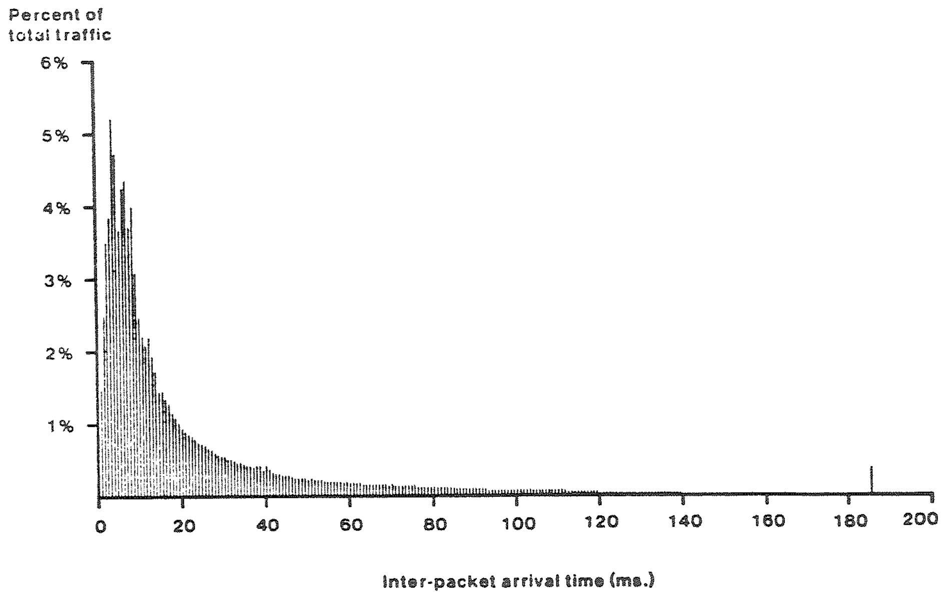


Figure 2.2: Measured distribution of interpacket arrival times for Ethernet ([SH80], Figure 6).

result indicates that once more than five hosts were active, the network was sensitive to the total offered load, but not to the configuration producing the load. A peak utilization of 96% was observed for packets of 512 bytes, with little apparent degradation as offered load increased beyond 100%. The artificial load generation software was constructed on top of the PUP protocol suite, and was thus restricted to a maximum packet size of 512 bytes.

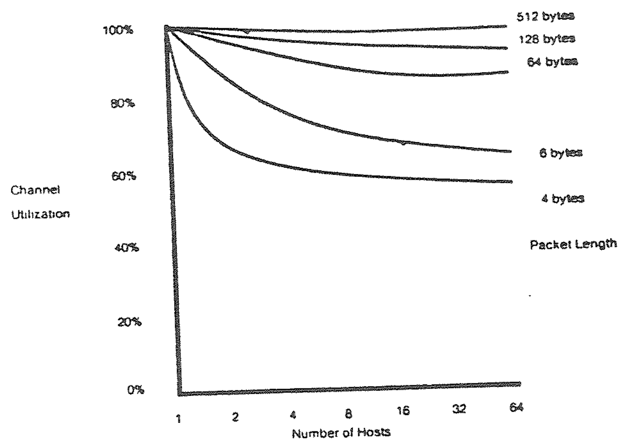


**Figure 2.3:** Measured utilization of Ethernet with continuously queued sources ([SH80], Figure 10(b)).

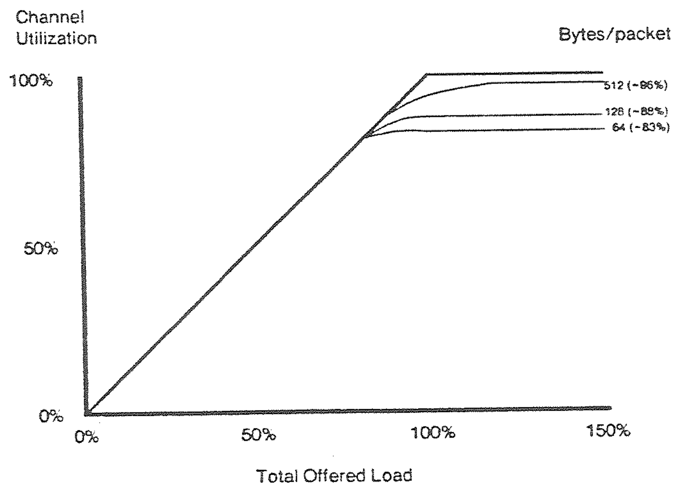


**Figure 2.4:** Measured utilization of Ethernet under high load ([SH80], Figure 9).

**Gonsalves**

Recently, the results of Shoch and Hupp were extended to include packet delay measurements and throughput results for the production 10 Mbps version of Ethernet. Using the same scheme as Shoch and Hupp, Gonsalves produced traffic consisting of fixed length packets with uniformly distributed generation times. On the 10 Mbps Ethernet, Gonsalves found that small packets were more susceptible to delay, with most packets suffering delays of up to 10 times the packet transmission time even with an offered load of only 19% of capacity. At high loads, delays of up to 0.5 seconds were observed, with 25% of the small packets transmitted suffering delays of greater than 15 times the packet transmission time. Under the same load conditions, 25% of the large packets suffered delays greater than 10 times the packet transmission time. Figure 2.5 shows the measured throughput versus offered load for Gonsalves' experiments. Figure 2.6 shows the average delay versus throughput.

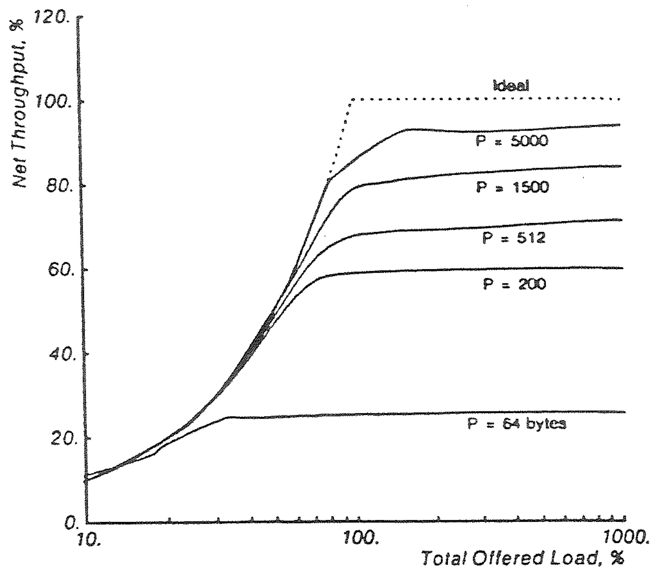


Figure 2.5: Measured utilization of Ethernet under high load ([Gons85], Figure 4.8).

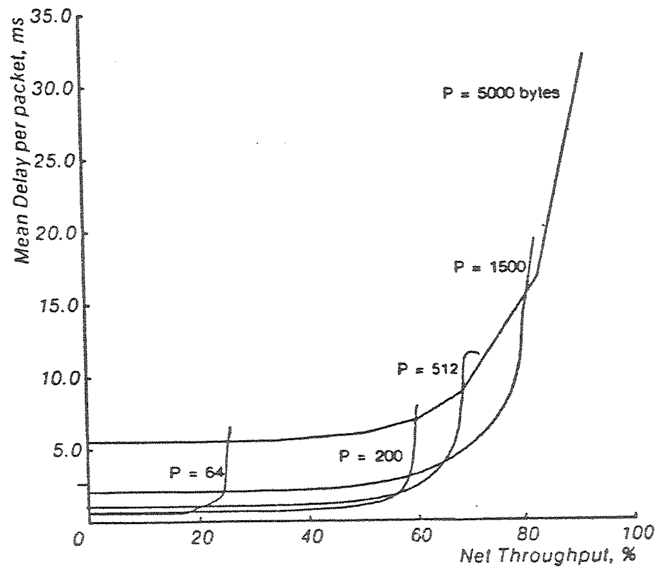


Figure 2.6: Measured delay of Ethernet under high load ([Gons85], Figure 4.9).

### 2.1.2. National Bureau of Standards Experiments

Amer [Amer82] described the design of a Local Area Network performance measurement center for the National Bureau of Standards NBSNET. NBSNET is a 1 Mbps bus network employing a CSMA/CD protocol similar to the Ethernet protocol. The measurement center consisted of software for monitoring network traffic both for throughput and delay statistics, and special hardware devices for generating artificial loads. The NBS Measurement Center has been used for several studies [Stok83].

A similar facility for measuring a campus Ethernet exists at the University of Delaware [AK85]. The results of an eight week observation of network traffic is reported in [AKKPC86]. 104 million packets containing 10 billion bytes of data were observed during the study. It was concluded that the traffic on the observed network was not as strongly bimodal as that observed in other studies due to the fact that there was no heavily utilized file server on the network. The packet arrival distribution did not resemble a poisson arrival process. The observed peak to average ratio of arrival rate was from 5 to 1 to 10 to 1. Finally, it was observed that network servers have a much greater influence on local network traffic than user behavior.

### 2.1.3. Digital Equipment Corporation Experiments

Two groups within DEC have produced studies on Ethernet Behavior. Boggs, et. al. of the DEC Western Research Laboratory performed artificial load generation studies intended to dispel persistent misconceptions among the user community concerning the limitations of Ethernet. Lewis conducted a study of Ethernet usage at a large DEC facility using a DEC built network monitoring device.

#### Boggs, Mogul, and Kent

Experiments aimed at dismissing persistent myths concerning the limitations of Ethernet are described in [BMK88]. Boggs, et. al., argue that many theoretical results have been "taken out of context," leading to inaccurate beliefs in the networking community about the performance of Ethernet. As an example, they cite the widely held belief that Ethernet saturates at an offered load of 37% of capacity. While this is true for a load composed strictly of minimum sized packets, throughput remains near 100% over a wide range of offered loads for packets of greater length.

These experiments were conducted on a network composed of twenty-four prototype Titan RISC workstations. The computers were attached to four DELNI multiport repeaters, which were spaced at 1000 foot intervals for most of the experiments. Measurements were also made with equal clusters of computers at either end of the cable. The measurements included the bit rate, the packet rate, the transmission delay (which includes queueing delay due to collisions and the actual transmission time of the packet), and the excess delay. Excess delay is the measured average delay minus the " ideal delay" for a packet to be transmitted in a system with a given number of hosts.

Boggs, et. al. observed that increasing the number of hosts participating in the experiments or decreasing the packet size decreased the efficiency of the network. The decrease due to increasing the number of hosts was particularly evident in experiments where the load consisted of small packets. Figure 2.7 shows the measured utilization versus the number of hosts for various packet sizes. Experiments conducted with a bimodal distribution of packet lengths showed a decrease in peak utilization and the same decline in efficiency with increasing number of hosts. Plots of the standard deviation of the bit rate show that the Ethernet backoff scheme is inherently unfair for networks with a small number of hosts, but that the fairness improves with the number of hosts. The packet rate for the experimental network showed a peak and decline as the number of hosts increased for packets of length smaller than 512 bytes. For larger packets, the packet rate showed a slight but linear decrease as the number of hosts increased. Delay was found to increase linearly with the number of hosts participating in the experiment, as did the standard deviation of delay. The worst case excess delay (for maximum sized packets and 24 hosts) was about 3% of the measured delay, corresponding to the 97% utilization observed under the same circumstances.

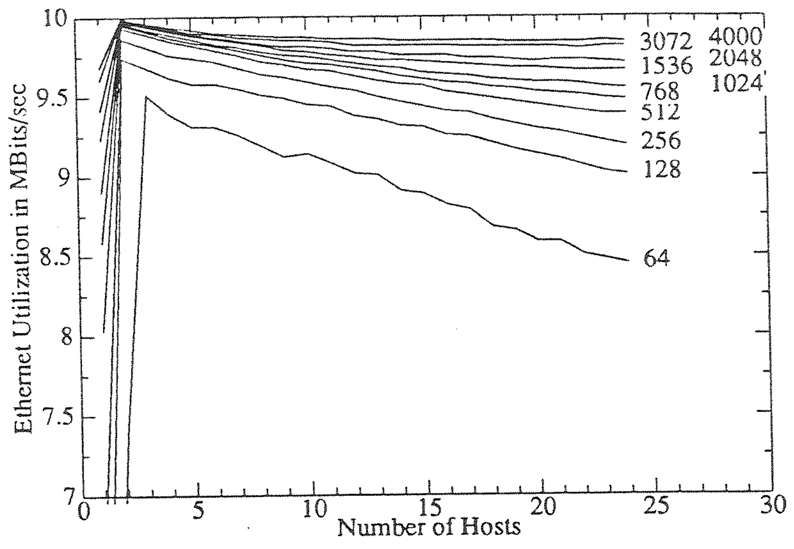


Figure 2.7: Measured bit rate versus number of hosts for Ethernet ([BMK88], Figure 3-3).

## Lewis

Recently published figures [Lewi89] detail the utilization of a large Ethernet over a twenty seven hour period. The Ethernet at the DEC Spit Brook Road, New Hampshire facility was monitored using a DEC monitoring device. The network consisted of 467 computers and twenty-three terminal servers with 1528 terminal lines. The network was normally partitioned by two Lanbridge 100's, but for this experiment, the bridges were removed. The resulting network consisted of a 500 meter backbone with eight segments attached with repeaters. Lewis reports normal utilization between 15% and 20%, with occasional peaks of 50%. Usage was observed to increase when employees arrived for work, before lunch, after lunch, and before quitting time. With the bridges restored, the utilization was normally 8% to 17%, with peaks of 35%. The intent of the study was strictly to evaluate the utilization of the network, so no artificial loads were measured. Additionally, traffic was broken down by protocol, indicating that most of the packets were generated by the terminal servers. No breakdown in terms of packet length was discussed. It is interesting to contrast the differences in both the size of the network measured and the average utilization between this study and the original Xerox study. The differences reflect the tremendous growth of network installations and applications in the intervening decade.

## 2.2. Broadcast Protocol Verification

The literature of protocol verification is rich with methodologies for the verification of point to point communication protocols. Unfortunately, these methodologies are not particularly useful for modeling and analyzing the properties of broadcast protocols in which the position of stations on the network, the propagation of signals, the relative position of signals on the medium, and the relative timing of events may be important.

While there have been attempts to formally analyze broadcast protocols (in particular, approaches using finite state machines [Frat83, TBF83]), the attempts have had an unsatisfactory level of representation or have not been general enough. However, in recent work by Jain and Lam [JL87][JL88][JL89], a model which captures the interesting behavior of broadcast channels in a simple yet powerful way is presented. A small extension to this model to handle collision detection in a natural manner is proposed here and the resulting model is used in the discussion of slotted and unslotted protocol behavior in Chapter 6. The model is described in greater detail there.

# Chapter 3
# The Local Area Network Testbed

This chapter discusses the hardware and software that make up the LANT. It is of greatest interest to those interested in the design and implementation of network testbeds. Readers whose interest is in the protocol performance aspects of this thesis may wish to restrict their attention to Sections 3.7 and 3.8 which discuss the load generation process and the protocol implementations. Section 3.1 discusses the design goals for the testbed. Section 3.2 discusses the DEC VAX† computers and network hardware underlying the system. The remaining sections are devoted to the various software components used in the LANT. The network monitoring program ILMON and the associated utilities are discussed in Section 3.3. The Experiment Configuration Package (ECP) is discussed in section 3.4. The Experiment Execution Package (EXP) is discussed in Section 3.5. The Experiment Analysis Package (EAP) is discussed in Section 3.6. The system software that implements the measurement functions is discussed in section 3.7. The protocols and their implementations are discussed in Section 3.8. A simulator for the Ethernet protocol is discussed in Section 3.9. Section 3.10 contains a summary of the chapter.

† DEC and VAX are trademarks of the Digital Equipment Corporation

## 3.1. Testbed Design Goals

The LANT provides an environment where experiments can be conducted to discover the performance behavior of various local area network communications protocols. In this environment, such experiments are repeatable and easy to configure. The system accommodates the addition of new protocols as they are proposed. It provides extensive monitoring capabilities, including passive monitoring of production networks and source queuing delay monitoring on testbed hosts.

The primary goal in the design of the Local Area Network Testbed was to create an environment where performance experiments could be run in a repeatable and reliable fashion. Many of the previous performance measurements on local area networks have been done on production LANs operating under real workloads [SH80][Lewi89]. Much has been learned about typical network traffic in this way, but because of the variable nature of day to day workloads, these measurements are not repeatable. Even when artificial workloads were generated, the networks were not dedicated to the experiments.

The testbed was designed to facilitate the investigation of alternative communications protocols. Many such protocols have been suggested and analyzed mathematically or simulated, but very few have been implemented and tested on actual networks. Repeatability is especially important when investigating the performance of alternative protocols. The experiment control software uses the standard Unix *ioctl* message passing mechanism for communication with the instrumented network interface device drivers. This provides a single software interface to the (necessarily) device dependent instrumentation code for the underlying network technology. Though the protocol implementation or even the interface hardware may change from experiment to experiment, the experiment control software remains the same.

LANT allows the user to configure experiments quickly and easily. The experiment is described at a relatively high level of abstraction; the user need not think about the details of writing processes to generate loads or the details of different protocols. Each experiment description consists of a list of probability distributions and an assignment of those distributions to the generation of packet lengths and interpacket intervals on each load generation machine. This method of specifying load generation allows a high degree of flexibility in the kinds of experiments that can be created. It is possible to design an experiment that models any observed network behavior closely. The inclusion of a user-defined, discrete distribution allows the behavior of a wide range of computer activities such as file service, terminal interaction with mainframes, or workstation use to be modeled.

Each host in the testbed network is furnished with monitoring software for measuring the queueing delays incurred by each packet due to traffic conditions and the operation of the protocol. Delay is an important characteristic of protocols which has been estimated and simulated, but has seldom been actually measured. LANT software can also monitor the traffic on the network passively. Quantities such as total throughput and frequency of source/destination pairs are recorded. This capability is used to monitor the performance of production networks to help characterize actual network traffic loads.

## 3.2. Testbed Hardware

The testbed hardware allows changes to the basic ethernet protocol in software to implement the various new protocols. Five independent load generators were used in the experiments. If a single source transmits to a single destination, no collisions can occur, and 100% throughput is possible. Similarly, because of the process activity and interface hardware architectures which require significant time to change modes from receiving to transmitting, a symmetric A to B and B to A load will tend to minimize collisions. The only way to simulate a more random load (i.e. a larger user population) is by having at least three processors actively communicating with each other. The choice of five computers was motivated by a combination of economic considerations and the results presented by Shoch and Hupp. The results for utilization under continuously queued packet sources presented in [SH80] indicated that the effect of the number of transmitting hosts on utilization was minimal for more than 5 hosts.

The experimental network testbed consists of five VAX 11/750 machines [Moll83]. Each machine has a minimal configuration of 2 megabytes of main memory, 131 megabytes of fixed disk storage, a 10 megabyte removable media disk and a console. A single 3Com Ethernet board is installed in each machine with an additional Interlan board for the gateway. The gateway between the experimental network and the regular departmental network

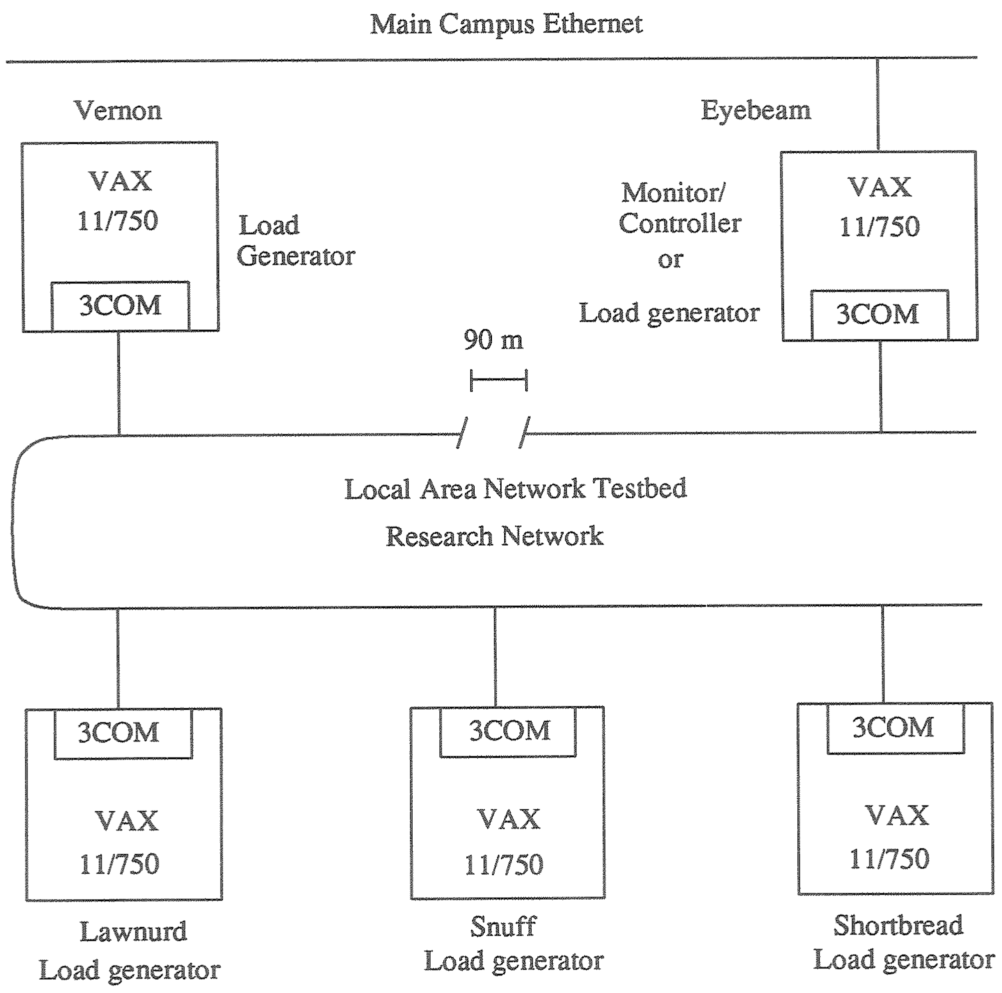


**Figure 3.1:** Layout of LANT stations.

was necessary to allow experiments to run in isolation from the normal traffic of the main network and to provide access to a stable environment for the development of the testbed software.

The inherent speed of the hardware (10Mbps) requires a relatively powerful processor to generate appropriate loads. The measurements made at the University of Delaware [MC83], indicate that throughputs of 875,000 bytes per second (7Mbps) are possible on a dedicated Unibus for small packets (250 bytes). This indicates that a testbed with five load generators could saturate a 10 Mbps network. Special techniques, such as retransmitting controller buffers without reloading, can be used to increase the load capacity of such machines. The LANT load generation machines are equipped with 3Com 3C300 Unibus Ethernet controllers [MC83]. These network interfaces, an early product offering by 3Com, perform the Ethernet backoff algorithm in the device driver rather than in hardware or microcode on the interface board. The design is uniquely suited for use in a performance measurement system such as LANT. Many of the proposed protocols can be implemented for these boards by modifying the packet transmission and collision handling routines in the Unix device driver for the interface.

Implementation of the Virtual Time CSMA/CD protocol requires hardware modification of the 3Com interface. The protocol requires that a "virtual clock" run only when the network is idle. In order to implement this feature, it is necessary to modify the interface boards so that the carrier sense signal is readable by the device driver. This modification is detailed in Section 3.7.4.

## 3.3. The ILMON Network Monitoring Package

ILMON (InterLan MONitor — the software was specific to the Interlan NI-1010 Ethernet interface) is a network monitoring package for Unix systems which performs promiscuous reception of network traffic and presents the user with reports on the observed traffic in various formats [BM87]. Packet reception may be conditioned on many quantities, including hardware source and destination addresses, packet length, packet type, and the error status of the packet. There are also tools for consolidating data from several monitoring sessions and for formatting the data for graphical presentation.

ILMON runs on a DEC VAX 11/750, which is connected to the main UT Campus Ethernet and an experimental network with Interlan NI1010 Unibus Ethernet Controllers. It uses the promiscuous reception and receive-on-error modes provided by the NI1010 to monitor network traffic. The 4.3BSD UNIX device driver for the NI1010 was augmented to provide various monitoring modes which the user selects by constructing a *filter* for the monitoring session. A filter identifies which of the available monitoring functions are to be active during a session and under what conditions a packet should be included in the collected information.

ILMON was originally conceived as the passive monitor for the Local Area Network Testbed. In that capacity it collects information on various experimental protocols. In practice, the package was capable of providing useful information only when low loads were generated and was thus not an important tool in gathering data on the LANT experiments. However, the program proved invaluable in two other tasks. First, ILMON was used in debugging the experimental protocol implementations. Second, the package was used to characterize the traffic on the main campus network to aid in the design of the artificial traffic loads.

The remainder of this section details the design, implementation, and use of ILMON. Section 3.3.1 discusses the organization of the system, including the method of configuring monitoring sessions using filters, the types of reports produced, and the practical limitations of the system. Section 3.3.2 discusses the implementation of the device driver extensions and the user interface for ILMON. Section 3.3.3 gives some results on the efficiency of ILMON. A user's manual for ILMON is found in Appendix C.

### 3.3.1. ILMON Design

The design of ILMON centers on using filters to specify the monitoring activities which will occur during a monitoring session. The user specifies a filter with a command interpreter which is loosely based on the UNIX *ifconfig* utility. The filter itself is a data structure containing flags indicating which of the possible monitoring activities is enabled for a session. There are also fields in the structure for various comparison quantities and for storage for the data collected during a session. Appendix A contains a listing of the C structure definition used for filters. Many of the capabilities of popular commercial network monitors are incorporated into this framework.

## The Filter Mechanism

In general usage, the word *filter* suggests the desire to exclude or "filter out" something. Though ILMON also uses filters in accomplishing other tasks, the principal purpose of a filter is to exclude uninteresting packets from the data recorded during a monitoring session. Due to memory constraints and the large disparity between mass storage access times and the typical interval between network events, it is not practical to record all packets in their entirety for later examination. Thus, it is necessary to be selective during the collection of data. In ILMON, filters specify a predicate which a packet must satisfy before information about that packet is included in the collected data. The filter also determines which of several different data collection modes is in use during a monitoring session, and provides storage for the collected data.

### Filter predicates

ILMON allows packets to be filtered by any combination of several criteria. To be collected packets must satisfy the conjunction of all the criteria specified in the filter used for a session. The criteria include packet status, station addresses, length ranges and packet types.

The user may choose to include only packets received in error, only valid packets, or all packets. The Interlan NI1010 normally does not pass error packets to the device driver. However, it is possible to run the interface in "receive-on-error" mode, in which case error packets are not automatically discarded by the interface.

ILMON allows the user to filter packets on Ethernet source and destination addresses. To do so, a list of source addresses is specified and the source address of each incoming packet is checked against the list. If the source address does not match any of the addresses in the list, the packet is discarded. If a match is found, the packet is logged or summarized. A similar list is specified for destination addresses. If either list is empty, no checking on the corresponding address is done. This type of checking is useful for monitoring the traffic between two stations, or monitoring the network output of a troublesome host.

The user may specify a constant between the minimum and maximum allowable packet lengths and one of the relational operators >, <, or =. Packets are included which have lengths greater than, less than, or equal to the constant respectively. It is also possible to specify a range of lengths. In this case, packets are included in the collected data only if their length falls within the range specified.

Packets may be filtered according to type. The possible types are shown in Table 3.1. These types are taken from the Network Information Center's "Assigned Numbers" report [RP85]. Type information is often useful in characterizing the applications which contribute to network load.

### Collected quantities

Once a packet has satisfied the filter predicate, the storage fields of the filter are updated with information about the packet. A number of bits in the filter flag are used to specify what sort of information is kept during a monitoring session. Certain quantities are kept for every session, regardless of the filter flag value. These quantities are:

1) the number of errors which occurred during the session

2) the number of collision fragments observed during the session

3) the number of packets lost during the session.

These quantities are taken from the NI1010's onboard statistics registers. The total reported for number of lost packets by the interface is actually a count of the number of times one *or more* packets were lost before the successful reception of a packet, so estimates of actual packet loss using this value are not completely accurate. In particular, any loss rate above 50% will be reported as 50% — every packet received will have the "lost" flag set.

For short periods of time, it is possible to log the headers of packets as they arrive for later examination. Memory constraints limit the number of headers that can be retained at any given time to approximately 5000. Given an average network utilization of around 1 Mbps, this capacity allows roughly 15 seconds of header logging. From ILMON it is possible to retrieve and store filters, thus extending the duration of a packet logging session. However, performing this action causes some packets to be missed in the meantime. The information stored is the arrival time of the packet, the ethernet header, and the IP header, if

Table 3.1: ILMON – Recognized Packet Types

| Type | Code | Description |
|---|---|---|
| ETHERTYPE_PUP | 0x200 | PUP protocol |
| ETHERTYPE_PUPAT | 0x201 | PUP address transfer |
| ETHERTYPE_EXP | 0x400 | LANT experiment packet |
| ETHERTYPE_IP | 0x800 | IP protocol |
| ETHERTYPE_ARP | 0x806 | Addr. resolution protocol |
| ETHERTYPE_X75 | 0x801 | X.75 Internet |
| ETHERTYPE_NBS | 0x802 | NBS Internet |
| ETHERTYPE_ECMA | 0x803 | ECMA Internet |
| ETHERTYPE_CHAOS | 0x804 | Chaosnet |
| ETHERTYPE_X25 | 0x805 | X25 Level 3 |
| ETHERTYPE_NSCOM | 0x807 | XNS Compatibility |
| ETHERTYPE_SYMP | 0x81C | Symbolics Private |
| ETHERTYPE_DMOP | 0x6001 | DEC MOP Dump/Load Assistance |
| ETHERTYPE_DRCS | 0x6002 | DEUNA Remote Console Server |
| ETHERTYPE_DNET | 0x6003 | DECNET |
| ETHERTYPE_LAT | 0x6004 | DEC LAT |
| ETHERTYPE_CRVLN | 0x8003 | Cronus VLN |
| ETHERTYPE_CRDIR | 0x8004 | Cronus Direct |
| ETHERTYPE_NEST | 0x8006 | Nestar |
| ETHERTYPE_EXCL | 0x8010 | Excelan |
| ETHERTYPE_RARP | 0x8035 | Reverse ARP |
| ETHERTYPE_DLB | 0x8038 | DEC Lanbridge 100 |
| ETHERTYPE_LOOP | 0x9000 | Loopback support |
| ETHERTYPE_BRIB2 | 0x9001 | Bridge IB2 |
| ETHERTYPE_BR | 0x9002 | Bridge Status |

the packet has type IP.

Histograms of the number of packets and bytes originating from or intended for each hardware address on the network may be collected. Since maintaining the data structures for such a histograms incurs a large amount of processing overhead, many packets may be lost while collecting data in this mode.

A histogram on the size of observed packets may be collected. This information (the number of packets of each size) is also used to calculate the throughput for the network and total number of bytes observed in the monitoring session. A histogram on the types of the packets observed may also be collected. The recognized types are shown in Table 3.1.

### 3.3.2. Limitations of ILMON

There are several practical limitations to ILMON's capabilities. ILMON is not able to log collision fragments due to the fact that the Interlan controller filters them out automatically. This feature of the controller is not programmable in the way that the filtering of error packets is. Being able to examine the fragments themselves is a desirable capability in fault diagnosis tasks. In addition to this hardware limitation, the efficiency of the ILMON software is less than perfect. The instrumentation of the controller device driver introduced extra code into the time-critical receive interrupt routine. This means than when monitoring is in progress, some packets are not processed before they are overwritten in the controller's buffers. A quantification of this packet loss appears in section 3.3.5.

There are also useful features which were considered but not included in the design of ILMON. Among these are real-time display of collected data and a more flexible scheme for specifying the "filter predicate." ILMON is not written for a particular workstation or graphics device, and thus does not incorporate the visual display of collected data in real time. In light of this fact, the further reduction of efficiency which would have been incurred by retrieving and displaying the data as it was collected was deemed counterproductive. As previously stated, the various conditions specifiable in the filter are connected by logical *and*. This was convenient and sufficient for the author's purposes, but may be expanded in the future to incorporate grouping and disjunction.

### 3.3.3. Implementation

ILMON is implemented as an instrumented version of the 4.3BSD UNIX device driver for the Interlan NI1010 Ethernet controller and a front end which allows the various monitoring options to be selected by the user. There are also several programs for displaying the filters collected and for consolidating and reducing the data.

#### Driver instrumentation

Monitor functions are activated using the UNIX *ioctl* mechanism to communicate between the user application and the device driver. Code was added to the *ilioctl* routine in the Interlan driver to pass filters in and out of the kernel, to set and clear promiscuous mode, and to reset and retrieve the on-board statistics registers. Whenever the interface is in promiscuous mode (as indicated by a bit in the flag field of the *ifnet* structure for the interface) code in the receive interrupt routine *ilrint* is executed to test packets against the filter predicate. If a packet satisfies the predicate, the filters counters are updated to reflect the reception of the packet, and the headers are stored if packet header logging is enabled. Finally, the address of the packet is checked. If the destination is the monitor station or the broadcast address, the packet is enqueued for the proper higher level protocol. Otherwise, the packet is discarded.

#### User Interface

The ILMON package user interface is composed of several programs. The program called ILMON is an interactive front end, loosely based on *ifconfig*, which allows full control of the monitoring functions. *Timedmon* is a non-interactive program which allows monitoring sessions of specified duration to run at a specified time. Finally, there are several programs for displaying and analyzing data collected by ILMON.

ILMON allows the user to specify, run, and save the results of monitoring sessions. The filter specification process is menu-driven, presenting the user with a list of the possible monitor functions and predicate conditions. Additionally, the user may define filter templates, which, can be saved, reloaded and used in monitoring sessions at later times. These templates are also used in the non-interactive program *timedmon*. Several other utility functions are available in ILMON, such as retrieving the Interlan's onboard statistics, reading the interface flags from the *ifnet* structure, reading and setting the driver state flags from the *ilsoftc* structure, and reading the interface board's control registers. Many of these functions are left over from an earlier program used to debug the added *ioctl* code in the device driver.

ILMON can be viewed as needing constant supervision. *Timedmon* automates the monitoring process, taking all the information it needs to run a monitor session (or a series of monitor sessions) as command line arguments. *Timedmon* requires that a filter template have been previously defined and saved in ILMON. The syntax for *timedmon* is then

`timedmon -i`*interface #* `-f`*filter* `-d`*duration* `-s`*start time* `-o`*output file* `-r`*repetitions*

*Interface #* is the unit number for the interface to be monitored. *Filter* is the name of a file containing a filter template defined in ILMON which will be used for the monitoring session. *Duration* is a string of the form hh:mm:ss specifying how many hours, minutes and seconds the monitoring session is to last. *Start time* is the time (in 24 hour notation) at which the monitoring session is to begin. *Output file* is a name to be used for saving the resulting data, or a prefix from which file names will be built should multiple sessions be requested. *Repetitions* is the number of times the monitoring task should be repeated. This program allows monitoring session requests to be set up ahead of time and left in the background to wait for their specified starting time. It also provides the ability to run sessions for a specified period of time, a function not provided in ILMON.

## Data examination and processing

Data analysis functions are provided by the Experiment Analysis Package (EAP), a part of the Local Area Network Testbed software. EAP provides some frequently used plots as well as the ability to specify general plots on any field of the filter's data. The standard plots available are: packet length histogram, a raw plot of network utilization data, and an averaged plot of network utilization data. Definition of the axes and labeling is under user control. The general plotting facility allows the user to choose quantities derived from filter data such as throughput, utilization, packet size, packet loss, etc. for the axes of a plot. This facility is easily extensible. EAP works on single filters, or can consolidate the data from many filters for plotting. Many of the figures for in this document were generated using EAP.

There are two other programs which are useful for viewing and analyzing filters. *Prfilter* simply reads in a filter and prints out the contents in human-readable form. *Logsum* produces a summary of filters containing logged packet headers. The information produced includes the timestamps of the first and last packets logged, the number of packets logged, the throughput, the average inter-arrival interval for the logged packets, and a histogram of the inter-arrival time distribution.

### 3.3.4. Performance

The extra processing in the network interface device driver required by ILMON for collecting data often causes the interface to miss some packets. The penalty thus incurred is investigated for packet histogram collection and header logging.

#### Packet loss for various monitoring functions

Figure 3.2 shows the percentage of packets lost during the collection of packet length histograms and packet header logging. Packet length histogram collection is the minimal monitoring task performed by ILMON, requiring only that one array element be incremented, along with the overhead incurred by any of the monitor functions. This overhead consists of several flag comparisons and at most two Ethernet address comparisons to determine whether a packet should be discarded or passed on to the higher level protocols. Header logging additionally requires that the packet header be copied into a log structure, and may require additional copying depending on the packet type. The data plotted in Figure 3.2 is averaged from one minute samples, with bins of 100,000 bits and, where possible, with 20 samples per bin. No users were logged in to the monitor computer while most of the samples were collected, though the computer was operating in multi-user mode. Packet loss for histogram collection remains under 10% for throughput levels up to 1.2 Mbps. At higher throughput, more serious degradation occurs. However, since throughput levels greater than 1.2 Mbps for one minute intervals were rarely observed, complete sets of

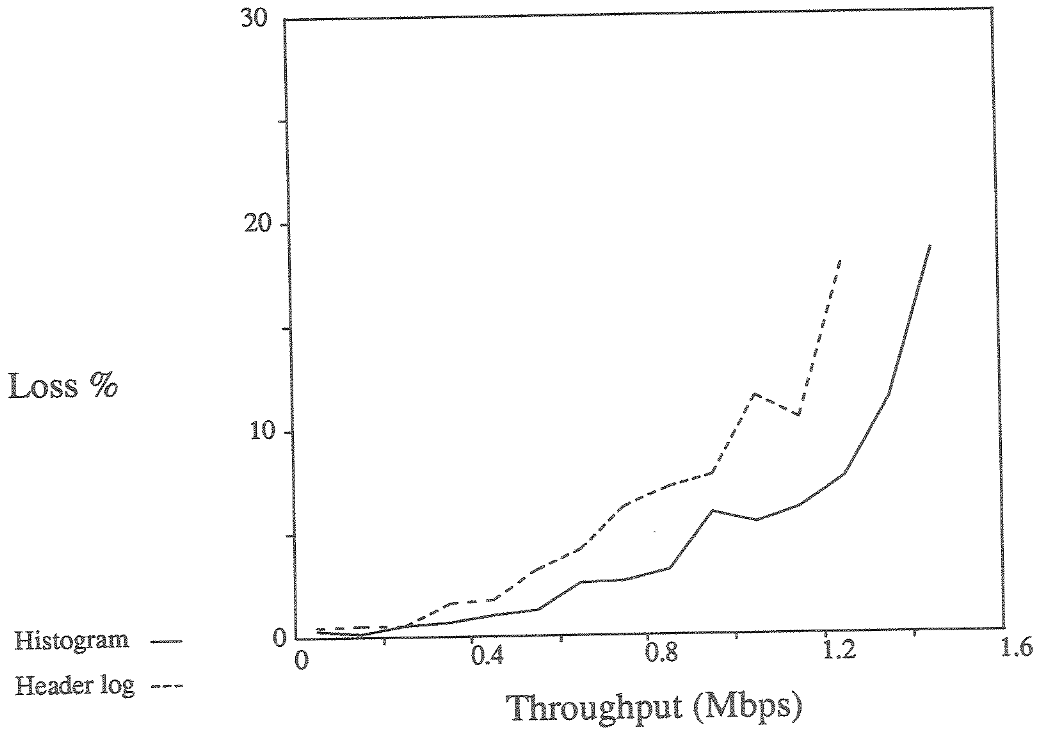


**Figure 3.2.** Efficiency of packet length histogram collection and packet header logging. Averaged from observations of the UT Campus network.

samples for these levels were not collected. Trials of LANT artificial load generation software indicate that the effective resolution of the Interlan's loss reporting mechanism (50% reported packet loss) is reached at throughput levels of approximately 3.0 Mbps. The loss percentage for header logging was, as expected, consistently greater than for histogram collection, exceeding 10% at a throughput of about 1.0 Mbps.

## 3.4. The Experiment Configuration Package

The Experiment Configuration Package (ECP) was written to provide a means for specifying parameters for experiments. The program allows the user to specify an experiment in terms of load generating processes residing on the various nodes of the network and the probability distributions which govern the length of packets and delay between packets. By specifying the behavior of the load generators in terms of distributions, repeatability is ensured. [Ferr78]

Logically, an experiment runs on a set of nodes. There is one monitor node, and the remainder of the participating nodes (perhaps including the monitor node) are load generating nodes. The monitor node initially performs some control and synchronization tasks, and then waits while the load generation is in progress. The load generating nodes take active part in producing traffic on the network by generating packets according to an experiment description prepared in advance. The load generating nodes also maintain local logs of packets produced and the queueing delay suffered by each packet. When the experiment has ended, the monitor node collects the local logs from the load generators. The ECP allows the user to specify the load produced by the load generation nodes during an experiment. A user's manual for the ECP is found in Appendix D.

### 3.4.1. Data Structures

The following scheme for representing experiment descriptions ensures a high degree of flexibility while keeping the configuration process simple. Experiment descriptions conceptually have three parts: a list of distributions for packet size and interarrival time, a topology, and a protocol selection. These three parts determine the behavior of the load generators and the network during an experiment; altering any of the three effectively creates a different experiment.

With this scheme, it is possible to run experiments with different protocols using the same packet size, time distribution and topology. It is also possible to run experiments using the same protocol and packet types with different topologies. Finally, it is also possible to evaluate the sensitivity of a protocol to packet size and average load for a given topology.

The behavior of the processes which generate packets during an experiment is governed by probability distributions for the length of packets generated and the delay between packets. These distributions are chosen from the distribution list which contains a number of records, each specifying a probability distribution taken from a standard list, along with the appropriate parameters. The available distributions are exponential, geometric, and binomial. It is also possible to specify a discrete or continuous general distribution by entering a set of x,y coordinate pairs. One distribution will be associated with a process for the generation of packet lengths, and another distribution will be used for the generation of delays between packets.

The distribution list is referred to by the topology description of an experiment. This description contains a record for each node in the network consisting of an two indexes into the distribution list, one for packet length generation and the other for interpacket delay generation. This scheme allows a wide range of packet generation behaviors to be specified in a uniform way.

The protocol selection determines which protocol will run during the experiment. The selection is made from a list of currently available alternative protocols. Since implementing and installing a new protocol requires a great deal of effort, each protocol is coded and compiled into a separate operating system kernel. This approach makes the inclusion of different network architectures, such as token rings, relatively painless. To the software, a new architecture would be nothing more than another protocol selection.

### 3.4.2. Files

The experiment description is stored in two files, the header file and the configuration file. Breaking the information into two files facilitates editing experiment descriptions, allowing a series of experiments to be defined using the same distribution list with multiple topology definitions or vice versa.

The header file contains general information concerning the experiment, the distribution list, and the kernel selection. The general information includes the name of the experiment, a comment on its purpose, and several general housekeeping items such as creation and last modification dates, flags indicating the completeness of the description, and information concerning the storage of the description.

The configuration file contains the topology entries for all processes to be created by the experiment. These records consist of a node number, the indices into the distribution list for interpacket delay and packet length, and indices into a table of seeds for the random number generation associated with the two distributions. The exact structure of the file is documented in Appendix B.

### 3.4.3. Software

The Experiment Configuration Package (ECP) allows a user to create and edit the experiment specifications discussed above. This front end was written using Curses, a Unix package for terminal independent screen manipulation. In order to allow the ECP to be used on as many different types of terminal as possible, it is menu oriented and contains only simple graphic aids for the experiment creation process. The structure of each screen used in the program is maintained separately from the program code. This allows the appearance of the screens to be altered without recompilation of the front end.

A section of the ECP is dedicated to the creation of each of the three parts of the experiment description. In addition, each part can be edited after creation, either to correct mistakes or to create new experiments. It is also possible to dump any or all sections of a description in printable form to a text file for later output and to update the list of available protocols in the protocol selection section.

## 3.5. The Experiment Execution Package

The Experiment Execution Package (EXP) was written to take the description generated using the ECP and run the described experiment. An experiment is implemented in terms of a process model. The processes fall into two classifications: processes involved in generating traffic during the experiment and processes involved in creating and managing the progress of an experiment. In the following sections, these processes will be described in high-level, functional terms.

The programs which comprise the EXP form a distributed application. Communication during experiments is carried out through polling, since the network connecting the machines is running under an experimental protocol. The processes can be thought of as falling into two categories. The first is data generating processes. The load generating processes fall into this category. An experiment consists of an instantiation of the load generating process running on one or more of the load generating nodes. The second category is experiment management processes. The experiment management processes are the supervisor, on the monitor node, and its agents, the local controllers, on the load generating nodes. In the following sections, each program will be discussed individually, then the flow of control among the programs during an experiment will be described.

The EXP data structures are used to the translate an experiment description into input parameters for the various processes that will be used to generate packets during the experiment. There is one supervisor record which contains the information from the header file. Using this structure, the supervisor creates records for each of the local control processes containing the distribution list, and a process record for the load generator the local controller is to create. These process records are distributed to the load generating processes upon their creation by the local controller.

### 3.5.1. Supervisor

The supervisor handles overall control of the experiment. The experimenter first chooses an experiment description and specifies a duration for the experiment. No other user intervention is necessary. The supervisor makes contact with a previously initiated local control process on each of the load generating nodes. These processes can be thought of as local agents of the supervisor. For each of these local control processes, the monitor extracts the pertinent information from the description and transmits it to the controller. When this activity is completed, the supervisor waits for a signal from each of the controllers. This signal indicates that the controller has received the descriptions of the load generating processes successfully and is ready to create them. When a signal has been received from each of the controllers, the supervisor sends a ‘go’ signal to each. At that point, the supervisor terminates. Collection of the local logs is accomplished by a Unix shell script which copies each log into the appropriate experiment directory.

### 3.5.2. Local Control Processes

A local controller resides on each load generating node and performs duties specified by the supervisor. After being initiated, the controller waits to be contacted by the supervisor. Upon being contacted, the controller receives the description of the load generation process it is to create and manage. When this description has been successfully received and copied into the device driver, the controller sends a signal to the supervisor indicating that it is ready to proceed. At some later time, the controller receives a ‘go’ signal from the supervisor. The controller then initiates the load generator and terminates.

### 3.5.3. Load Generator

The experiment description specifies the distributions the load generators will follow to generate packets and the duration of the experiment. Each load generator remains quiescent until it receives an activation signal from the local controller. The load generator then generates two random numbers, according to distributions passed to it for packet length and interpacket delay. Using the generated length, a packet is built, and after the generated delay has elapsed, the packet is submitted for transmission. This task is repeated until a specified amount of time has elapsed.

### 3.5.4. Control Flow

The actions of these processes can be grouped into three phases of an experiment run. These phases are initialization, load generation and data acquisition, and cleanup. The actions of the various processes during each phase are represented in table 3.2.

The first phase consists of the translation of an experiment description into information relevant to each load generating node, and the transmission of this information to the proper local controller. This translation process is carried out by the supervisor, which also synchronizes with each local controller and transmits the information. The initialization phase ends when the supervisor sends the 'go' message to the local controllers, and the controllers initiate the load generation processes.

The second phase consists of the actual generation of packets at the load generating nodes and the logging of the resulting network activity. This phase lasts until the load generating processes shut down.

The last phase of an experiment run occurs when the load generators shut down. After a brief interval, a shell script copies the local logs into the experiment directory on the monitor node. These logs include statistics on the number and distribution of collisions observed and on the queueing delays suffered by the packets. The successful retrieval of the local logs marks the end of an experiment run.

## 3.6. The Experiment Analysis Package

The Experiment Analysis Package (EAP) is one of a set of software tools to access and reduce the collected data. This program is briefly described in Section 3.3.3. The EAP is able to process both data filters from the ILMON network monitoring program and logs generated by the Experiment Execution Package. The program can produce summaries of single experiment logs or can combine logs from a number of experiments to produce curves of many varieties. The log files contain the following data: queueing and interpacket delay histograms, packet length histogram, collision histogram, total packet count, and duration of experiment. From this data, the program calculates and allows the user to display the throughput, offered load, average queueing delay, average interpacket delay,

**Table 3.2: Process actions during various experiment phases**

| **Phase** | **Process** | | |
|---|---|---|---|
| | Supervisor | Controller | Load Generator |
| Initialize | Select experiment | | |
| | Open connection to local control processes | | |
| | Send expt. description | Rcv expt. description | |
| | | Pass params to ld gen | Rcv params |
| | Send synch. signal | Rcv synch. signal | |
| Generate | Send 'go' | Rcv 'go' | |
| | | Send 'start' to load generators | Rcv 'start' |
| | | Terminate | Gen. pkts. |
| | | | Terminate |
| Cleanup | Retrieve logs | | |
| | Terminate | | |

variance of delay, average packet size, total collisions, collisions per packet, collision rate, total packets and packet rate. Any of these quantities may be displayed on the x or y axis of a line graph or scatter plot. Raw data or the average of a set of observations can be displayed. The program produces input files for a program called *fplot* (written by Dr. Michael Molloy) which produces files readable by Pic, the Unix picture description language [Kern82]. The figures in Chapters 4 and 5 were produced with these programs.

## 3.7. System Software

The testbed hosts ran the 4.3 BSD version of Unix. System software was modified in several ways. The modifications were necessary to implement the load generation and delay monitoring functions of the testbed. Changes to the clock handling routines are discussed in Chapter 4.

Of the many interesting challenges presented by building a testbed from off the shelf components, two of the most difficult are generating adequate throughput from each machine (particularly in a testbed with a small number of machines) and providing accurate timing information. Minnich reported maximum process to process throughput for TCP/IP on a 4.1BSD Unix VAX 11/780 using the 3Com controller to be 355 kilobits per second [MC83]. This figure suggests that any load generation technique built on top of TCP/IP would not be able to generate interesting loads with the equipment available for the LANT. It was therefore necessary to modify the network device driver for the network controller to provide both adequate generated throughput as well as delay and throughput measurements for the artificially generated traffic loads.

The artificial load generation code is included in the device driver. Throughput for each generator is boosted by bypassing the normal transmission process and using the onboard packet buffers of the 3Com controller as a packet library. The 3Com has 32K of memory which is used as 16 2K packet buffers. The normal process for transmitting a packet is to copy the packet from main memory into the high end of one of these buffers, write an offset to the beginning of the packet in the first word of the buffer, and then initiate transmission by writing the buffer index into the transmit control register. Since the contents of artificially generated packets is irrelevant, it was possible to preload the buffers with one Ethernet header after another. By filling fifteen of the sixteen buffers with headers in this way, it is possible to have a header in the proper location for transmitting a packet of any legal size with no copying of data from main memory onto the board. Consequently, to generate a packet of a given size, it is only necessary to look up the buffer number and offset of the proper header in a table, write the offset to the first word of the buffer, and write the buffer number to the control register. This technique allows a single host to generate almost 60% of network capacity for maximum sized packets.

Modifications were also necessary for timing purposes. The clock handling routine and clock interrupt frequency were adjusted to provide interval timers and timestamps of adequate resolution for the traffic measurements. During experiment execution, histograms of the per-packet delay and packet length are collected . Delay is measured from the time at which the packet is submitted for transmission to the time at which the transmission is successfully completed. This is accomplished by taking a timestamp immediately before the transmit control register of the controller is written with the transmit command and another immediately after the "transmit done" bit in the transmit control register is set. A histogram of the number of collisions suffered by packets is also collected.

## 3.8. Protocol Implementations

Three protocols were investigated using the LANT, the Ethernet protocol, the Enet II protocol and the VTCSMA/CD protocol. The Ethernet experiments used the device driver supplied with 4.3 BSD Unix for the 3Com Ethernet controller. The other two protocols were implemented by modifying this device driver.

### 3.8.1. Ethernet Implementation

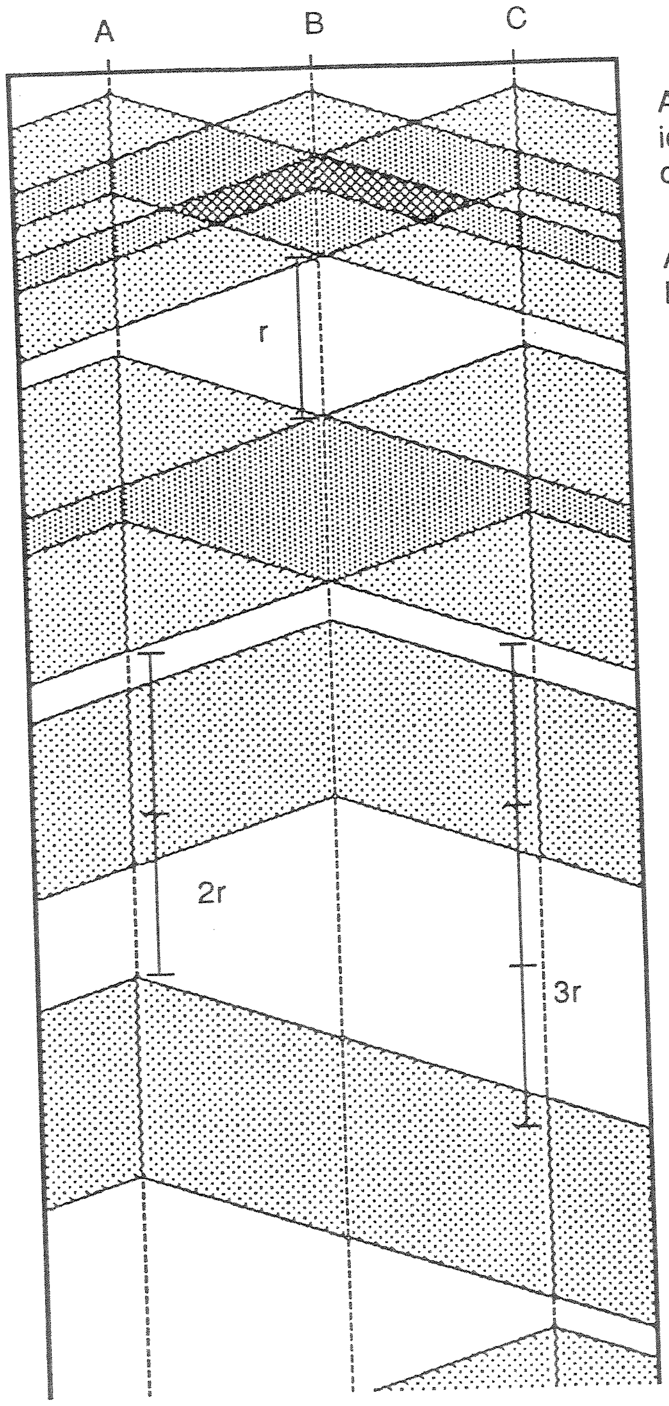
Ethernet is one of the classic examples of a CSMA/CD bus network protocol. Ethernet uses a 1-persistent CSMA/CD access strategy. A ready station which senses the network busy when attempting to transmit will transmit with probability 1 when it senses the network idle at the end of the transmission in progress. This strategy results in low latency when traffic is light, but causes some decrease in throughput under heavy loads when compared to the nonpersistent and p-persistent versions of CSMA/CD.

When a station has a packet ready to transmit, it first checks to see whether the ether is currently busy (the *carrier sense*). If not, the packet transmission begins immediately. If so, the transmission is deferred until the ether is idle. If no other stations are attempting to transmit, the packet is successfully transmitted.

Because the stations are spatially distributed, there may be a non-negligible propagation delay between stations. Hence, it is possible for two stations to begin transmission more or less simultaneously and for these transmissions to interfere with each other, corrupting the data in transit. This interference is typically referred to as a *collision*. If a station at one end of the network begins transmission at time $t$, it is possible for a station at the opposite end of the network to begin transmission as late as $t + \alpha - \varepsilon$, where $\alpha$ is the propagation delay of the network and $\varepsilon$ is some arbitrarily small constant. The second station will detect the collision immediately, but the first station will only see the interference at $t + 2\alpha - \varepsilon$. However, if no interference is detected by $t + 2\alpha$, then the station is said to have captured the ether, and the transmission is guaranteed to be successful. For a formal discussion of this effect, see Theorem 6.2.

The second component of CSMA/CD is collision detection. Early protocols, such as Aloha [Abra70] discovered collisions after the complete packet had been transmitted, thus wasting the entire transmission time of the packet. The efficiency of a CSMA protocol is greatly increased by detecting the interference immediately and aborting the transmission as quickly as possible. Ethernet interfaces achieve this behavior by continuously comparing the signal on the ether to the signal they are transmitting and aborting their transmission when a difference is detected. A 32 to 48 bit burst of noise is then transmitted to insure that all stations recognize the collision. (This is referred to as the collision consensus enforcement policy.) An example of the operation of Ethernet collision handling is given in Figure 3.3. The figure is a time/space diagram, with time advancing downward, and the horizontal axis representing the length of the network cable. The shaded areas indicate the propagation of signals from the stations. Darker areas indicate that signals from more than one station are overlapping. The quantity $r$ refers to the slot time for the network.

The Ethernet specification [DIX82][IEEE82] places restrictions on several aspects of the network. Signal attenuation limits the effective length of a single segment of Ethernet, but repeaters which regenerate the signal are allowed and may be used to set up a tree configuration of segments. The maximum allowed round trip propagation delay is 464 bit times. This is a delay limit, rather than a cable length limit; devices such as repeaters induce some latency of their own. The specification states that the longest path between two transceivers is to be 1500 meters. In order for collisions to be uniformly detectable, the minimum packet size is 512 bits. To simplify the design of interfaces and to maintain quick response for interactive traffic (small packets), the maximum packet size is 1536 bytes.



A, B, and C sense the ether idle and transmit, causing a collision.

A and C choose to wait 0 slots. B chooses to wait 1 slot.

A and C begin transmission after the minimum packet spacing, and collide again. B senses the channel busy and defers.

A chooses to wait 2 slots. C chooses to wait 3 slots. B transmits when the channel goes idle and completes successfully.

A senses the channel idle after 2 slots and transmits.

C senses the channel busy and defers. A transmits successfully.

C transmits.

**Figure 3.3:** Ethernet collision handling example.

The experiments on Ethernet performance were performed using the Ethernet implementation supplied with the 4.3BSD Unix distribution (/sys/vaxif/if_ec.c). This implementation adheres to the Ethernet spec in all but one important particular. The collision interrupt service routine uses only five bits of the backoff mask to generate the backoff interval, rather than the ten bits specified. The truncated backoff interval means that the interval from which the backoff is randomly chosen increases for the first five collisions suffered by a packet to 32 slot times, but for all subsequent collisions the interval remains 32 slot times. A correct implementation should increase the interval for the first ten collisions to 1024 slot times. Communication with Keith Bostic of BSD [Bost88] indicates that this was an oversight in the implementation. Unfortunately, this problem was discovered only after the measurements were completed. The truncated backoff scheme implementation improves delay at low loads by shortening the average retransmission delay. However, under heavier loads redistributing retransmissions over a shorter interval also increases the chances of the retransmissions suffering subsequent collisions. The truncated backoff algorithm partially explains the instability observed in Ethernet in the measurements presented in Chapter 5.

### 3.8.2. Enet II Implementation

The Enet II protocol performs collision resolution in the unslotted CSMA/CD environment and incorporates techniques that facilitate the implementation of slotted protocols in this environment. The difficulty with implementing most of the slotted collision resolution protocols on a CSMA/CD bus is that the algorithms must be able to determine when an idle 'slot' has occurred. In an unslotted medium, this determination is difficult. To understand how Enet II accomplishes this, it is necessary to note two facts. First, the Ethernet specification [DIX82] and the IEEE 802.3 standard [IEEE82] set a maximum length for cables in CSMA/CD bus networks. Using this limit, a protocol can assume a fixed maximum propagation delay, $a$, for the network. A station listening to the ether for $r = 2a$ after a message completes is guaranteed to hear any station which began transmitting after the message completed transmission. Second, collision detection and the subsequent 'jam' (collision consensus enforcement) do not waste very much bandwidth. It is therefore practical to use intentional collisions to gain information about the state of other transmitting stations during the collision resolution period.

In Enet II, a station can be in one of three states; inactive, active, or deferred. Inactive stations either have no packets to send or have just successfully transmitted a packet. Active stations are trying to transmit a packet that may have been involved in a collision. Deferred stations have attempted a transmission but are waiting for active stations to complete their transmissions. In the algorithm statement that follows, $r$ represents twice the propagation delay of the network. The protocol we describe here is referred to as "Naive Enet II," and is described in Figure 3.4. The protocol can be improved by adding counters as described in [Moll85] to implement an unslotted Capetanakis protocol.

```
Algorithm Statement

Inactive → active (new packet available for transmission)
    Sense channel
    Wait until idle for $3r$, then transmit

Active
    Successful transmission → inactive
    If a collision occurs while transmitting, flip a coin
    If heads is flipped, try to transmit again; active
    If tails is flipped, monitor the channel
        If channel idle for $r$, transmit; active
        If successful transmission observed, transmit; active
        If collision observed within $r$ → deferred

Deferred
    Monitor the channel
        If idle for $2r$, transmit; active
        Otherwise, repeat deferred action
```

**Figure 3.4.** Statement of the Enet II algorithm.

The transmit by active stations after the channel is observed idle for $r$ produces a guaranteed collision which signals that no stations flipped heads, so stations should flip their coins again. Collisions observed within $r$ of the original collision are viewed as actual collisions, while collisions observed between $r$ and $2r$ after the original collision are viewed as signals that an idle step has just ended. The initial $3r$ wait serves to allow active and deferred stations to finish their transmissions before any new packets are transmitted. The operation of the protocol is demonstrated pictorially in Figure 3.5.

In two instances the testbed hardware and software made it impractical to implement the Enet II protocol exactly. The Enet II specification calls for timing of events in multiples of $r$, the round trip propagation delay of the network. For IEEE 802.3, $r$ is approximately 50 microseconds. As detailed in Section 4.1.1, increasing the clock interrupt rate to meet the 50 microsecond requirement affected the performance of the computers to an unacceptable degree. A clock resolution of 200 microseconds was finally implemented. As a result, the specified idle periods of $r$, $2r$, and $3r$ for the various protocol states are approximately four times as long as they theoretically could be. Throughput and delay characteristics of the resulting protocol are degraded, but not necessarily by a factor of four.

The second problem is also related to the idle monitoring periods, and is an effect of the packet reception process of the network controller used by the load generators. In order to monitor network events, a promiscuous receive request is issued to the controller. Any packet which subsequently appears on the network is reported to the device driver, regardless of the packet's intended destination. If a packet was received during the monitored interval, it is indicated by a bit in the receive control register of the controller. A collision is indicated by the reception of a packet smaller than the minimum required packet size. An idle interval is indicated by the absence of the "receive done" bit in the receive control register of the controller. The problem occurs when the controller receives a packet whose transmission time is greater than the monitored interval. Since the "receive done" bit is not set until the transmission is complete, it appears that the ether was idle when the interval expires. This means that the protocol will occasionally mistake a successful transmission for an idle period. The result of this situation is that some packets will join the transmission process sooner than they should, and that some deferred packets will retransmit earlier than they should. Very few packets reach the deferred state, so it is believed that the performance degradation thus caused is negligible. In order to correctly implement this portion of the algorithm, it would have been necessary for the interface to have a register bit which

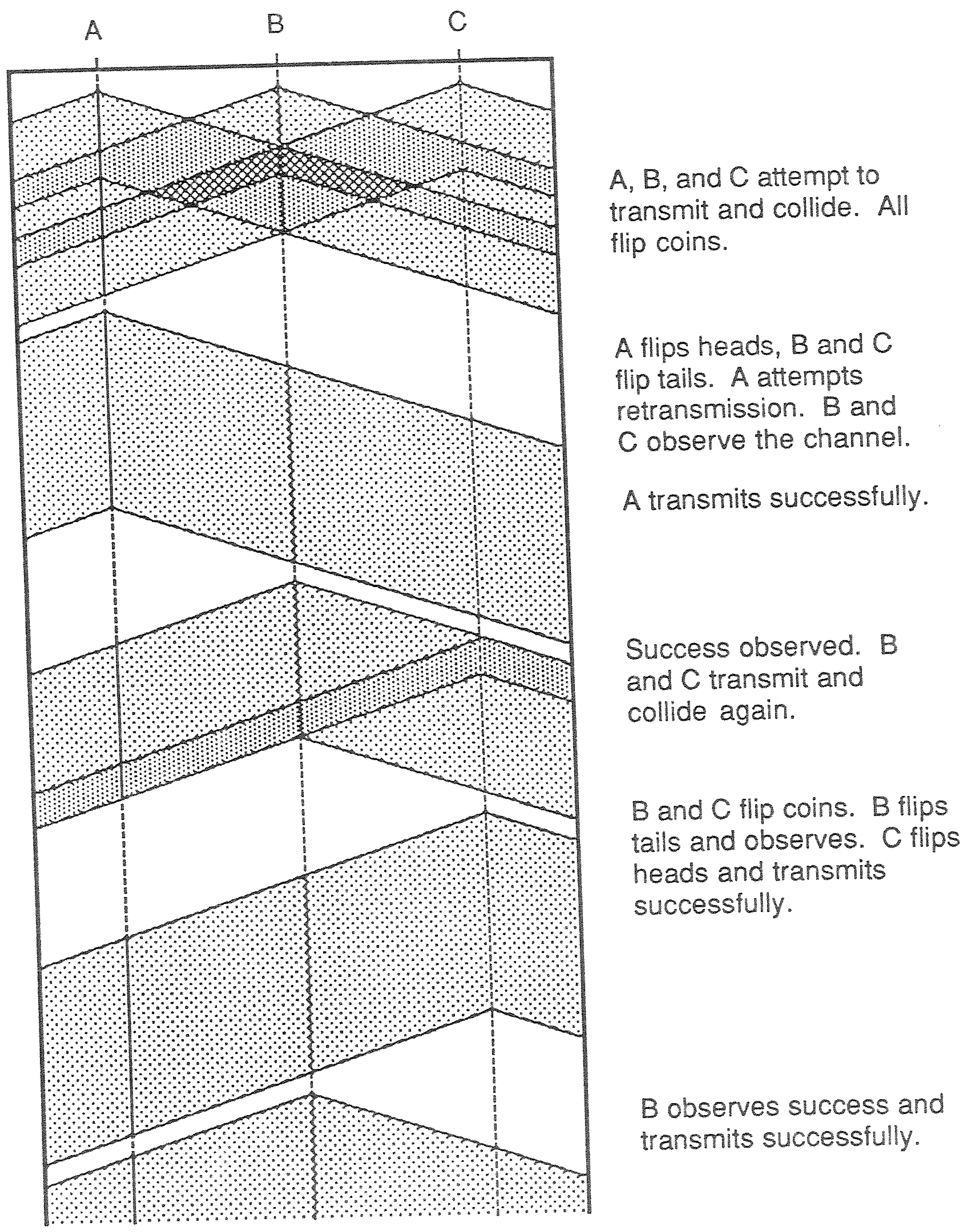

A
B
C
A, B, and C attempt to transmit and collide. All flip coins.
A flips heads, B and C flip tails. A attempts retransmission. B and C observe the channel.
A transmits successfully.
Success observed. B and C transmit and collide again.
B and C flip coins. B flips tails and observes. C flips heads and transmits successfully.
B observes success and transmits successfully.

**Figure 3.5:** Enet II example.

was set while reception was in progress. (The modifications mentioned in Section 3.7.4 would have also made this possible, but the modifications were not performed until the Enet II measurements were completed.)

### 3.8.3. Virtual Time CSMA/CD Implementation

The central idea behind the VTCSMA/CD protocol is the elimination of the synchronizing effect that busy periods have on transmission attempt in 1-persistent CSMA/CD protocols like Ethernet. In such protocols, all stations which experience a new packet arrival during a busy period will attempt to transmit when the busy period ends. If there is more than one new arrival during a busy period, a collision is guaranteed for these protocols. The actions of the VTCSMA/CD protocol provide a mechanism for spreading such arrivals out in time with respect to the end of the busy period. Each station maintains two clocks, a real time clock and a virtual clock. The virtual clock runs only when the channel is idle. When virtual time and real time coincide, the virtual clock runs at the same rate as the real time clock. At all other times, the virtual clock runs at some rate $\eta$ that is faster than real time. When a packet arrives for transmission, the station marks the packet with the current real time. When the virtual clock reaches the packet's timestamp, the packet is transmitted. Note that since the virtual clock is running, the channel must be idle, so VTCSMA/CD follows the CSMA discipline for channel access. However, the actual transmission time of the packet depends both on the end of the most recent busy period and on its timestamp, avoiding many of the collisions experienced by Ethernet. Figure 3.6 shows the operation of the algorithm in C-like pseudocode. The protocol is modeled as two functions, one transmission function and one virtual clock manager. The code assumes that the actually advancement of the virtual clock is taken care of in the same interrupt service routine which advances the real clock by looking at the variables *vc_rate* and *vc_run*.

```
/* Global variables -- initialized at boot time */
unsigned long vc_rate;     /* Virtual clock rate when behind real time */
boolean vc_run;            /* True when virtual clock is running, false otherwise */
unsigned long rt, vt;      /* Real time and virtual time clocks */

/* Protocol constants */
PENDING                          /* Timestamp < vt */
COLLISION                        /* Collision occurred */
SUCCESS                          /* Packet transmitted successfully */

vc_manager()
{
  /* Initially virtual clock runs at same rate as real clock */
  vc_rate = 1; vc_run = TRUE;

  do {
        /* While carrier sense low, only check to see if vt has caught up to rt */
        while ( !carrier )
          if ( vt == rt && vc_rate == η ) vc_rate = 1;

        /* Carrier has gone high.  Stop clock, increase rate. */
        vc_run = FALSE; vc_rate = η;
        while ( carrier )
          ;
  } forever;
}

vc_transmit ( packet )
char      *packet;
{
  unsigned long  arrival_time;     /* Packet's timestamp */
  unsigned       status;           /* State of protocol */

  arrival_time = rt;               /* Value of real time clock */
  status = PENDING;

  while ( status != SUCCESS ) {
    /* Wait for virtual clock to reach timestamp */
    while ( vt < arrival_time )
          ;

    status = transmit ( packet );    /* Attempting to transmit */
    if ( status == COLLISION )
          arrival_time = backoff();        /* Schedule retransmission */
  }
}
```

**Figure 3.6:** The VTCSMA/CD algorithm.

Implementing the Virtual Time CSMA/CD protocol presented severe difficulties on the LANT hardware. One of the requirements of the protocol is that the virtual clock stop whenever the channel is busy. The first attempt at implementing this behavior using only the promiscuous receive capability of the interface was unsatisfactory. The only indication that a promiscuous receive has occurred is the "receive done" interrupt. There is no indication when a promiscuous receive *starts*. It was therefore necessary to approximate the virtual clock behavior by determining the length of the received packet and subtracting the transmission time for a packet of that length from the virtual clock. This led to inconsistencies in the implementation, since events which occurred during the reception which would have been delayed by a correctly stopped virtual clock could not reasonably be aborted.

These difficulties led to modification of the interfaces to facilitate the operation of the virtual clock. This modification consisted of bringing the carrier sense signal out to an unused bit in the Receive Control Register (RCR). While this improved the situation, it still required that the algorithm continuously keep promiscuous receive requests pending, increasing the interrupt rate and thus decreasing the efficiency of the implementation. Since there was nothing meaningful to be gained by examining received packets, for the other two protocols, reception was turned off during experiments to reduce interference with the load generation process.

The experimental packet transmission process operates as follows. The arrival time for a packet is represented by a global variable called *vtalarm*. When the virtual time clock equals or exceeds the value of *vtalarm*, the packet transmission process is initiated. The buffer offset for the specified packet length is loaded into the appropriate buffer, the virtual time clock is halted, and the transmit bit in the Transmit Control Register (XCR) is set. The virtual time clock can safely be halted since either immediate transmission or deferral both indicate a busy channel. While the interface handles the transmission, the experimental transmission process checks the jam bit of the XCR to see whether the packet has suffered a collision. If a collision occurs, a backoff scheme similar to that used in Ethernet is used to back off the retransmission. The backoff algorithm initially generates longer delays than the Ethernet scheme since it is constrained to work in real time clock units of 200 microseconds and must still allow distinction between two stations on the first backoff. The collision routine generates a delay, which it combines with the current virtual time to form a collision alarm. The routine then waits until the virtual time clock is equal to or exceeds this alarm value. While waiting, the carrier sense bit in the RCR is checked and the virtual clock halted when it is high. The next arrival is generated as in the other two implementations, with the exception that the new arrival time is formed by adding a delay to the current virtual time.

It should be noted that the necessity for continuously servicing receive interrupts during the operation of the protocol adds significantly to the overhead of the protocol. The Enet II implementation suffers from this problem to some extent, but the need to perform promiscuous receptions in Enet II is limited to certain points in the resolution process, while VTCSMA must monitor the channel at all times in order to maintain the virtual time clock. In the Ethernet implementation, receive interrupts are disabled during artificial load generation.

## 3.9. Ethernet Simulator

Some aspects of protocol behavior revealed in the LANT experiments were not directly related to parameters under the experimenter's control. To aid understanding in these areas, a simulator was written for Ethernet. The simulator is of the discrete event variety, and simulates a finite population of hosts. Since any number of hosts could be simulated, the offered load can be calculated in a manner similar to the process of calculating the offered load for the LANT. The simulator allows specific arrangements of stations to be simulated, rather than assuming that the stations are uniformly distributed along the cable. Cable length, collision detect time, jam time and transmission overhead are available to the user as input parameters. The simulator also allows the specification of discrete distributions for interarrival time and packet length in a way analogous to the ECP specification of distributions. Special attention is paid to the modeling of collisions, taking into account collision detection times and locations of all stations involved in each collision. Finally, versions of the simulator with both the standard and the truncated version of the backoff algorithm were written. Use of this program gave added insight into the effects of the truncated backoff algorithm, as indicated in Figure 5.2.

## 3.10. Summary

This chapter has described the hardware and software that is collectively referred to as the LANT. This testbed was used to gather and interpret the data presented in Chapters 4 and 5. It differs from previously implemented testbeds in several ways. The LANT was built completely from off-the-shelf, general-purpose hardware; no special purpose hardware was required (though the network interface boards were modified to provide some otherwise unavailable information to the device drivers). The method used for configuring experiments proved very flexible, allowing the specification of a wide range of traffic behaviors for experiments. Finally, the testbed was intended as a platform for the implementation and investigation of a number of protocols, not just for one specific network system.

# Chapter 4

# Measurement Methodology

This chapter discusses the techniques used to measure the performance of the protocols investigated in the LANT project and the describes the design of the artificial load generation experiments. The first two sections discuss modifications to the system software which were necessary to measure the quantities of interest for the experiments. The third section discusses the types of loads used in the experiments and the motivation for the particular loads chosen. In this section, a traffic analysis is presented of the University of Texas campus Ethernet which demonstrates the changes in the patterns of network usage since the pioneering studies of Shoch and Hupp [SH80].

## 4.1. Modifications to UNIX clock management

In the default configuration for the VAX 11/750 version of 4.3BSD Unix, hardware clock interrupts are generated every 10 milliseconds. The Unix interval timer (getitimer/setitimer) and timestamping function (gettimeofday) are built on top of this service. It is possible to adjust the interrupt rate by changing the HZ parameter in the param.c file and recompiling the kernel. However, since the LANT load generation software operates at a lower level than the system calls mentioned, simply increasing the value of HZ was not the desired solution, and in fact would have been counterproductive, since the normal tasks associated with the system clock interrupt would have been executed more often.

A second level of clock interrupt service was implemented on top of the normal clock interrupt service in the *hardclock*() routine in /sys/sys/kern_clock.c. A second compiler flag, RHZ was defined, representing the number of " real time" clock interrupts per hardware clock interrupt service. Clock interrupts occurred HZ*RHZ times per second, but the normal clock interrupt service code was run only every RHZ times this happened, providing the same system interrupt service rate as the default system. A "real time clock" and "real interval timer" were implemented using the extra clock interrupt service requests.

The real time clock was implemented as a globally accessible unsigned long integer variable called *rtc* which was initialized to zero in the system's boot time initialization routine, *init_main*(). This variable was then incremented each time *hardclock*() was executed, providing a timestamp accessible to the network device driver code. Another globally accessible variable, *ric* (real interval clock), was defined. This variable was decremented with each *hardclock*() execution. When it reached 0, a real time process, *rtproc*(), which resided in the network interface device driver, was called. *Rtproc*() set a flag which indicated to the artificial load generation process that it was time to send another packet. The load generator set *ric* to the delay until the next packet after each transmission.

RHZ provided an adjustable interrupt rate. As noted, the Enet II protocol should be able to respond to events with 50 microsecond resolution. A kernel was prepared with RHZ set to provide 50 microsecond timestamping and interval clock settings. While this kernel ran, it apparently spent all of its time incrementing *rtc* and decrementing *ric*; it did not even respond to console input. The real time clock code requires only one increment and one comparison in a typical interrupt service episode, so this was actually a problem of reaching the limit imposed by the length of the normal hardware clock service code. Eventually, it was determined that 200 microsecond resolution was the best compromise between the requirements of the algorithm and reasonable operation of the system. This compromise effectively causes the Enet II protocol to operate as though the network length is four times the maximum length specified for Ethernet.

## 4.2. Network device driver instrumentation

The primary measurement task for the artificial load generation software was measuring the queueing delay for each packet. This was achieved by keeping a descriptor for the packet which contained the packet length and start and stop times for the transmission. The load generation process (*send_xpkts*()) copied *rtc* into the descriptor after loading the offset word of the interface buffer and before setting the transmit bit. The ending value of *rtc* is copied after the transmit done bit is set in the XCR and the interface is reset. The packet length and transmission times are then inserted into histograms, along with the number of collisions suffered by the packet. The interpacket interval is also recorded in a histogram.

## 4.3. Experiment design and analysis

One of the goals of the LANT project was to measure protocols (in particular Ethernet) under realistic traffic loads. The typical assumption in analytic studies, artificial workload measurements and simulations is that packet lengths are either fixed or are distributed exponentially around some mean or uniformly over some range. Measurement of actual traffic such as [SH80] indicated that the loads most prevalent on existing Ethernets did not conform to any well behaved distribution, but were strongly bimodal in nature. Since these studies were several years old at the inception of the LANT project (ca. 1984), the first task undertaken was the construction of the ILMON network monitoring package described in Chapter 3 for use in traffic characterization studies of the University of Texas campus Ethernet. The UT network was then and remains one of the busiest networks in the world, and provided an interesting perspective on the growth of network utilization in the years since the PARC studies were done. Section 4.3.1 reports the results of the traffic characterization study, which were used to guide the design of the mixed packet loads discussed in Chapter 5.

### 4.3.1. Production Network Traffic

ILMON was used to collect utilization data on the University of Texas campus Ethernet. At the time of these measurements, the network connected some 80 nodes, including mainframes, minicomputers, workstations, fileservers, laser printers and terminal concentrators. Many other machines reside on subnets and contribute to the network's traffic. Several higher level protocols, such as the DOD/ARPA protocol suite, DECnet, and CHAOSNET, were in use.

During June of 1986, the network was monitored for a 24 hour period, with samples summed over six minute intervals. The University was at that time involved in the first Summer Session, so traffic was probably somewhat lower than would have been the case during a regular academic session. During this period, 8.7 million packets were transmitted, containing 3.8 billion bytes of information, not including the Ethernet headers, preambles and frame check sequences. The average throughput was 0.358 Mbps, for a utilization of 3.58%. The peak utilization was 11.03%, and the minimum utilization was 0.7%. The results of this observation are shown in Figure 4.1. Figure 4.2 is a histogram of observed packet lengths. This figure shows the classic bimodal distribution, with a large percentage of small packets carrying terminal traffic, and a smaller percentage of very large packet associated with file transfers. The second group was somewhat exaggerated in the UT environment by the presence of a cluster of diskless SUN workstations and their file server on the backbone network. The SUN Network Disk (ND) protocol has a block size of 1072 bytes. Most of the traffic with packet sizes larger than 1072 are generated by various applications using the User Datagram Protocol. [Post80] Figure 4.3 is a histogram of the observed interval between packet receptions for the monitor node.

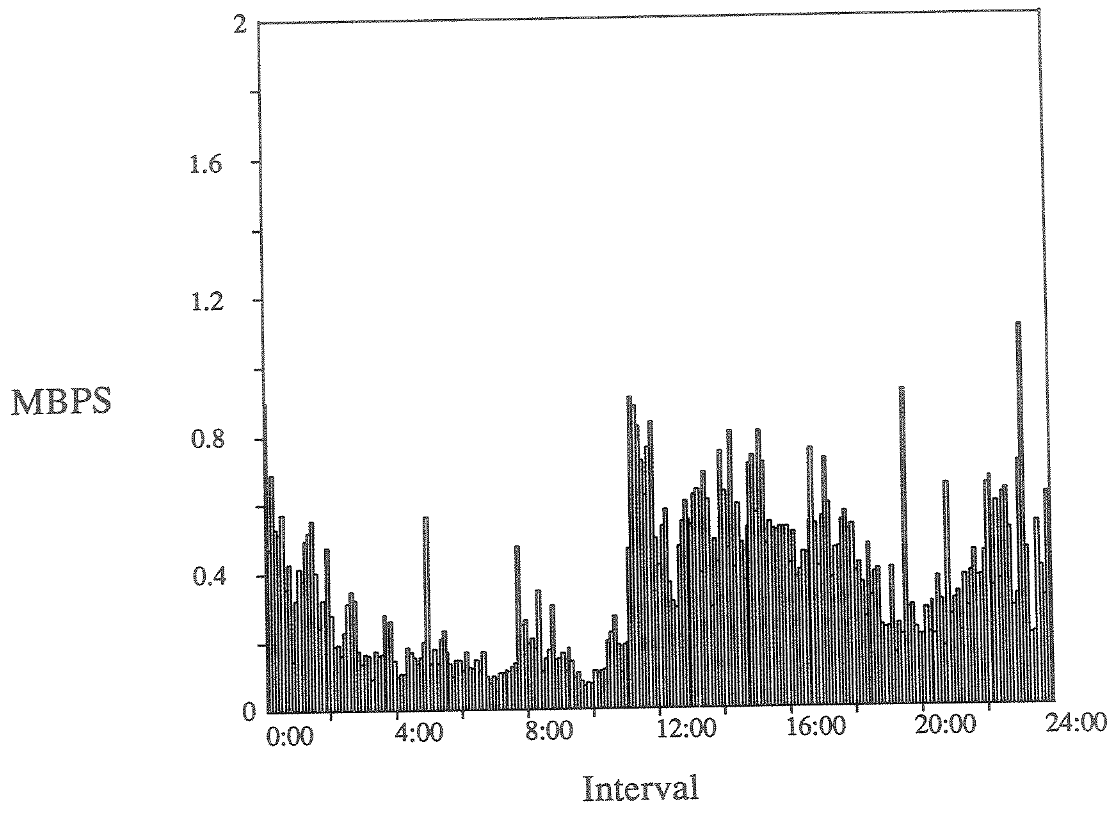


**Figure 4.1.** 24 Hour Ethernet utilization, six minute samples.

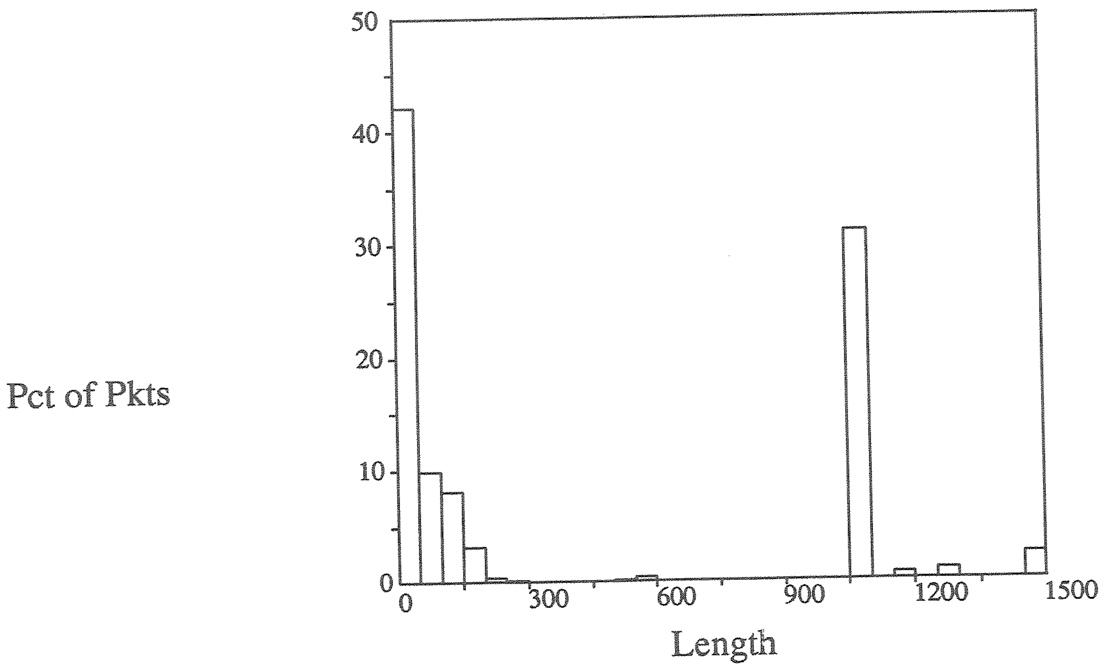


**Figure 4.2:** Packet length vs. percent of total packets.

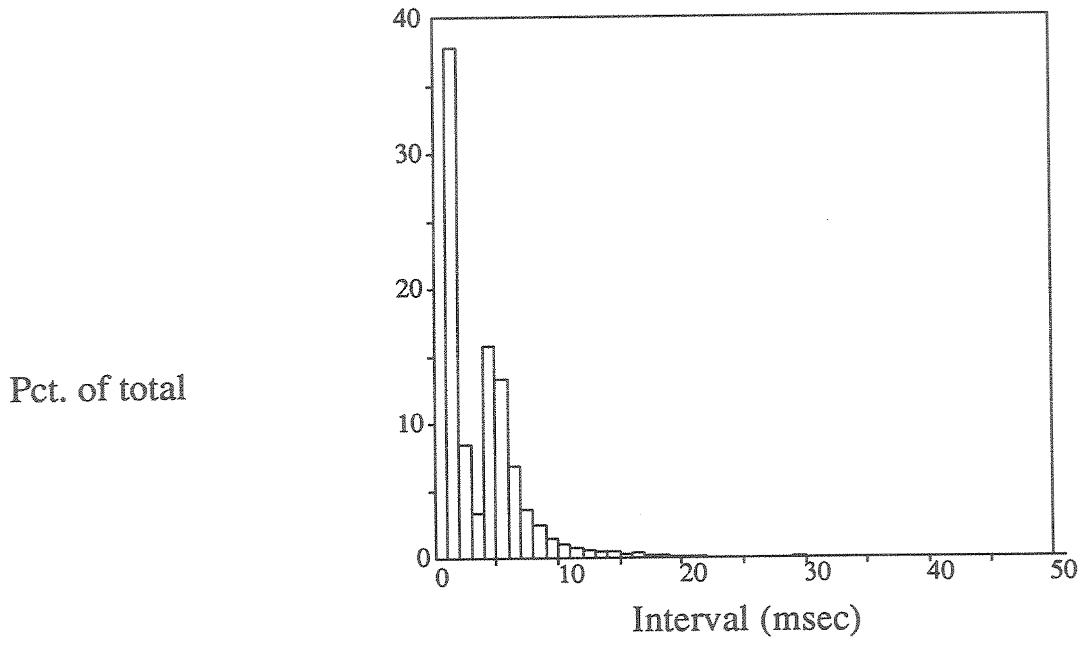


**Figure 4.3.** Interarrival time distribution. Taken from one thirty second packet logging sample.

A comparison to a similar set of observations reported in [SH80] is shown in table 4.1. The Shoch data was collected on the Xerox PARC experimental Ethernet, which ran at a rate of 3 Mbps, while the UT Campus Ethernet is a 10 Mbps network. Though this is a less than perfect comparison, the average utilization, peak utilization, and minimum utilization figures are of interest. These quantities indicate that though the two networks were of roughly similar size at the time of the measurements, the UT network was significantly busier. Though the small average packet size for the Xerox experiments is partly an artifact of the PUP protocol in use there, the difference also indicates that subsequent protocol designs have used the medium more efficiently. The observed differences between the two networks also reflect the growing dependence of a broad spectrum of computer tasks on Local Area Network communication.

**Table 4.1: Xerox vs. ILMON**

| Quantity | Shoch | ILMON |
|---|---:|---:|
| Packets/day (mil.) | 2.2 | 8.7 |
| Bytes/day (mil.) | 300 | 3,800 |
| Avg. Utilization | 0.8% | 3.6% |
| Peak Util. (6 min period) | 7.9% | 11% |
| Min. Util. (6 min period) | 0.2% | 0.7% |
| Mean packet size | 122 | 439 |
| Avg. inter-packet time (ms) | 39.5 | 7 |

Several load patterns which make use of the bimodal distribution observed in this study were defined and used in artifical load generation experiments. The exact composition of these loads is discussed in Chapter 5. For the sake of efficiency, the artificial loads were abstracted to produced packets of two different sizes, one small and one large. The artificial load generation software was constructed to generate this type of generation fairly quickly, while adhering more closely to the observed distribution would have significantly increased the packet generation overhead. Several different mixtures of two packet sizes were used to investigate the effect of increasing the percentage of small packets on the behavior of the protocols. Both fixed and exponentially distributed interpacket delays were generated.

### 4.3.2. Fixed Packet Length Loads

Many analytic studies deal with fixed packet lengths, so several load patterns were defined which generated only a single length. Generating this type of load was more efficient than generating a mixture of lengths, which resulted in higher offered loads for these patterns. As before, both fixed and exponentially distributed interpacket delays were generated. The packet sizes chosen as well as the interpacket interval generation method match those used in the artificial traffic generation study of Gonsalves [Gons85].

### 4.3.3. Statistical Considerations

Previous experimenters have determined that for Ethernet, relatively short experiment durations are sufficient. Gonsalves' experiment runs were 60 seconds in duration. Boggs, et. al. used experiment runs of 20 seconds, collecting data only during the middle ten seconds of each run. Since the LANT experiments were concerned with software implementations of experimental protocols on a small network, experiment duration was chosen conservatively. Each experiment lasted for ten minutes. Each host sent between 75,000 and 150,000 packets, depending on the offered load level. Each experiment was repeated three times, and the average quantities from the runs are reported in Chapter 5.

# Chapter 5

# Protocol Performance Measurements

This chapter describes the use of the LANT to measure the performance of various protocols under artificially generated network traffic loads. Three protocols, the Ethernet protocol, the Enet II protocol, and the Virtual Time CSMA/CD protocol, are investigated. As discussed in Chapter 3, all of these protocols were implemented for the 3Com 3C300 Unibus Ethernet interface, and in the case of the latter two protocols, the performance of the protocol was affected by limitations related to the interface and the operating system software.

The expressed purpose of the Ethernet local network is to provide low latency access for interactive users while still achieving reasonable performance for bulk transfers [MB76]. Measurement studies and analytic results suggest that Ethernet throughput should match the offered load until the point at which the offered load exceeds the capacity of the channel. At that point, the throughput should level off and remain stable until the load is much larger than the capacity of the channel. Leveling off of the throughput curve is accompanied by a sharp increase in delay, as stations are forced to wait for retransmission of their packets.

Enet II was designed to satisfy two goals. The first was to provide a random access alternative to Ethernet which experienced lower and less variable delays at high load than Ethernet. The second was to demonstrate a method for implementing collision resolution on an unslotted network. The algorithm requires an initial gating delay for packets making their initial transmission attempt, and thus experiences a larger average delay under light loads, but a simulation study presented in [Moll85] suggests that average delay should be lower as loads approach the capacity of the channel. This is due the action of the collision resolution algorithm, which prevents packets from waiting through long random delays when transmission is attempted on a heavily loaded network.

VTCSMA/CD is an attempt to reduce the possibility of collisions in a broadcast environment. If stations transmit successfully on their first attempt more often, then their average delay should be lower and their throughput higher. This situation also reduces the impact of the retransmission policy of the protocol, since it will be used less often. Analytic results presented in [MK85] predict that VTCSMA/CD should achieve higher peak throughput than Ethernet as well as a much lower average delay and collision rate under heavy loads.

The measurements presented in this chapter agree with these predictions in many particulars. Ethernet and Enet II show little loss of efficiency until the offered load approaches the capacity of the network. The low-load delay for Enet II is higher than that of Ethernet and VTCSMA/CD by approximately the amount of the gating delay imposed by the protocol. The delay variance of Enet II and VTCSMA/CD show great improvement over the Ethernet delay variance. The average delay advantage among the three protocols typically belongs to VTCSMA/CD, followed by Ethernet, though under some load conditions this relationship is reversed. Where the measurements differ from the expected behavior of the protocols, explanations are offered.

The measurements fall into two broad groups, measurements conducted using a fixed packet size, and measurements using a mixture of packet sizes. Fixed length experiments were run with packets of 1500 bytes and 1024 bytes. 1500 byte packets represent the most efficient utilization of network bandwidth and allowed very heavy traffic loads to be generated. The 1024 byte packets match packets generated by many common network applications such as the Sun Network Disk protocol and also allowed the generation of heavy overloads. For mixed packet traffic, three different packet mixtures were used, each with a bimodal distribution of packet sizes. Each distribution had packets of 1500 bytes and 275 bytes, with 25% small packets for mix 1, 55% small packets for mix 2, and 65% small packets for mix 3. Most networks do not display such a simple distribution of packet sizes, but packet size distribution on real networks does tend to be strongly bimodal. The two size distribution was chosen for efficiency reasons. For a larger number of packet sizes, the load generation process required a binary search for packet length generation, while choosing between two sizes requires only a single comparison. A two-size distribution still allows reasonable traffic to be generated in a five host testbed. Within each of the groups, experiments were run using both fixed interpacket intervals and exponentially distributed interarrival times. The fixed intervals allowed the greatest offered load to be generated, while exponentially distributed arrivals more closely match the arrival process of real networks. As previously stated, the actual distribution of interarrival times is not strictly exponential in nature (see Figure 4.3), but the exponential distribution most closely models the observed arrival process.

Throughout this discussion, the following abbreviations will be used. The offered load will be referred to as $G$. The offered load for a given load pattern is calculated by multiplying the average throughput of a single generator running the pattern in the absence of contention by the number of generators participating in the experiment. $S$ is the observed throughput for the network. It is calculated by summing the average throughput from each generator. $D$ refers to the average per-packet queueing delay. This delay is measured from the time the packet is first submitted for transmission until it is successfully transmitted. This figure includes the transmission time of the packet. $V$ refers to the variance of $D$. Unless otherwise stated, all experiments were run using all five of the testbed computers as load generators.

Error statistics were not available for all of the experiments. The measurements for the Enet II protocol driven by traffic mixture 2 had a confidence interval of 200 microseconds with a confidence level of 95% for the average delay figures. This error is within the timekeeping resolution.

In comparing the implementations of the three protocols, it is instructive to consider the maximum load generated by a single station. This gives us an indication of the overhead associated with the initial transmission of a packet. In all cases, Ethernet achieves the highest single station throughput. This should not be surprising, since the interface was designed for Ethernet, and (in the absence of copying) there is almost no overhead involved in the initial transmission of a message. Since there is no contention for a single transmitting station, there is no delay for deferral. In every case Enet II generated the second highest single station throughput. The initial $3r$ (600 microsecond) gating delay decreased the throughput performance for the algorithm, but once again, it was never necessary to defer while other stations resolved collisions in the single station case. The LANT implementation of VTCSMA/CD displayed the lowest single station throughput under various load patterns. This can be accounted for by the fact that in this implementation VTCSMA/CD required manipulation of the variables associated with the virtual clock for every packet transmitted, and these manipulations were at times forced to wait for clock interrupt processing to complete. This is an artifact of the software implementation of the protocol; in a hardware implementation the effect would be eliminated. It was also the case that both the Enet II implementation and the VTCSMA/CD implementation used promiscuous receives to monitor the channel. Enet II monitored the channel only at certain points in the resolution process, while it was necessary for the VTCSMA/CD implementation to monitor the channel at all times to operate the virtual time clock. This monitoring was implemented with busy waits; attempts to use interrupt service routines for this purpose proved unsatisfactory from a performance standpoint. Busy waiting was a practical solution for the purposes of the testbed, since the computer was expected to perform no other computational activities during experiments.

Table 5.1 shows the throughput and delays for each protocol, arranged according to the load pattern executed. Column a shows the protocol. Column b shows the type of the interpacket interval distribution. Column c shows the packet length distribution. If a number is given, it represents the fixed number of bytes per packet. These three columns constitute the load pattern. Column d shows the peak throughput achieved by a single host for the load pattern. Column e shows the average queueing delay experienced by the sending host for the load pattern. In this table, the queueing delay column does not include the transmission time for the packets, only the overhead associated with the transmission. Column f shows the average interpacket interval for the load pattern. The interpacket interval includes the time to calculate and record the delay for the packet just transmitted and the time to generate the description of the next packet to send. As expected, the load pattern

**Table 5.1: Offered Load Packet Generation Overhead, by Protocol**

| (a) Protocol | (b) Interval Dist. | (c) Length (Bytes) | (d) Thruput (Mbps) | (e) Queueing Delay (usec) | (f) Interpkt int. (usec) |
|---|---|---|---|---|---|
| Ethernet | Fixed | 1500 | 6.379 | 116 | 587 |
| | | 1024 | 5.537 | 109 | 576 |
| | | Mix 1 | 4.532 | 180 | 999 |
| | | Mix 2 | 3.701 | 183 | 975 |
| | | Mix 3 | 3.271 | 183 | 1011 |
| | Exponential | 1500 | 3.251 | 61 | 2477 |
| | | 1024 | 2.499 | 67 | 2441 |
| | | Mix 1 | 2.541 | 127 | 2718 |
| | | Mix 2 | 1.913 | 55 | 2797 |
| | | Mix 3 | 1.664 | 126 | 2764 |
| Enet II | Fixed | 1500 | 4.672 | 789 | 603 |
| | | 1024 | 3.737 | 803 | 600 |
| | | Mix 1 | 3.497 | 788 | 1021 |
| | | Mix 2 | - | - | - |
| | | Mix 3 | 2.423 | 787 | 1020 |
| | Exponential | 1500 | - | - | - |
| | | 1024 | 2.102 | 907 | 2224 |
| | | Mix 1 | 2.133 | 894 | 2681 |
| | | Mix 2 | 1.626 | 886 | 2586 |
| | | Mix 3 | 1.418 | 900 | 2587 |
| VTCSMA | Fixed | 1500 | 5.450 | 155 | 868 |
| | | 1024 | 5.152 | 196 | 985 |
| | | Mix 1 | 3.430 | 183 | 1681 |
| | | Mix 2 | 2.680 | 182 | 1664 |
| | | Mix 3 | 2.379 | 182 | 1668 |
| | Exponential | 1500 | 2.943 | 126 | 2791 |
| | | 1024 | 2.173 | 175 | 2831 |
| | | Mix 1 | 2.159 | 77 | 3445 |
| | | Mix 2 | 1.620 | 152 | 3336 |
| | | Mix 3 | 1.413 | 160 | 3341 |

(packet length and interpacket interval distribution) has little effect on the queueing delay. However, going from a fixed length packet to one of the bimodal packet mixtures adds between 300 and 600 microseconds to the interpacket interval, while going from fixed to exponentially distributed interpacket intervals adds between 1600 and 2000 microseconds to the interpacket delay.

Table 5.2 shows the same data organized by the packet length distribution. The columns are the same as those in Table 5.1, except that column a is now the length distribution and column c is the protocol. This table points out the difference between the protocol implementations. The Ethernet and Enet II implementations show very little difference in terms of the interpacket interval to record delay and generate new packets. The average difference between to two implementations is less than the resolution of the timestamp mechanism. However, the interpacket interval for the VTCSMA/CD implementation is consistently 500 to 700 microseconds slower. This is an indication of the additional overhead associated with virtual time clock maintenance mentioned in Section 3.7.4.

Table 5.2: Offered Load Packet Generation overhead, by Load Pattern

| (a) Length (bytes) | (b) Interval Dist. | (c) Protocol | (d) Thruput (Mbps) | (e) Queueing Delay (usec) | (f) Interpkt int. (usec) |
|---|---|---|---|---|---|
| 1500 | Fixed | Ethernet | 6.379 | 116 | 587 |
| | | Enet II | 4.672 | 789 | 603 |
| | | VTCSMA | 5.450 | 155 | 868 |
| | Exponential | Ethernet | 3.251 | 61 | 2477 |
| | | Enet II | - | - | - |
| | | VTCSMA | 2.943 | 126 | 2791 |
| 1024 | Fixed | Ethernet | 5.537 | 109 | 576 |
| | | Enet II | 3.737 | 803 | 600 |
| | | VTCSMA | 5.152 | 196 | 985 |
| | Exponential | Ethernet | 2.499 | 67 | 2441 |
| | | Enet II | 2.102 | 907 | 2224 |
| | | VTCSMA | 2.173 | 175 | 2831 |
| Mix 1 | Fixed | Ethernet | 4.532 | 180 | 999 |
| | | Enet II | 3.497 | 788 | 1021 |
| | | VTCSMA | 3.430 | 183 | 1681 |
| | Exponential | Ethernet | 2.541 | 127 | 2718 |
| | | Enet II | 2.133 | 894 | 2681 |
| | | VTCSMA | 2.159 | 77 | 3445 |
| Mix 2 | Fixed | Ethernet | 3.701 | 183 | 975 |
| | | Enet II | - | - | - |
| | | VTCSMA | 2.680 | 182 | 1664 |
| | Exponential | Ethernet | 1.913 | 55 | 2797 |
| | | Enet II | 1.626 | 886 | 2586 |
| | | VTCSMA | 1.620 | 152 | 3336 |
| Mix 3 | Fixed | Ethernet | 3.271 | 183 | 1011 |
| | | Enet II | 2.423 | 787 | 1020 |
| | | VTCSMA | 2.379 | 182 | 1668 |
| | Exponential | Ethernet | 1.664 | 126 | 2764 |
| | | Enet II | 1.418 | 900 | 2587 |
| | | VTCSMA | 1.413 | 160 | 3341 |

For each experiment set, plots will be presented of throughput versus offered load, delay versus offered load, delay versus throughput, delay variance versus offered load, and collisions per second versus offered load. Where appropriate, further analyses may be presented.

## 5.1. 1500 Byte Packets

The differences between the protocols examined were in their handling of contention among the host population. In a testbed with only five hosts, contention was not encountered with light traffic, but only when heavy loads were generated. With such a small host population these conditions were best achieved using large packet sizes. This load pattern was also of interest since it presents the situation where each host uses the channel with the greatest efficiency. The load pattern composed of all hosts generating 1500 byte packets with fixed interpacket intervals is first considered. For these experiments, each generator delayed for a fixed period after a successful transmission before sending the next packet. Experiments were also run where hosts generated an exponentially distributed delay after each successful transmission.

### 5.1.1. Fixed Interpacket Intervals

Figure 5.1 shows the throughput versus offered load of the three protocols. The units for both axes are megabits/second (Mbps). As expected, at low loads, the throughput characteristics of Ethernet and Enet II are very similar. VTCSMA/CD actually suffered fewer collisions than either of the other protocols, but overhead associated with each packet limited the throughput achieved by the protocol. Both Ethernet and Enet II exhibit a peak throughput of 9.8 Megabits/second. However, Enet II achieves this peak at an offered load of 10.64 Mbps, while for Ethernet the peak appears at 11.08 Mbps offered load. This difference is attributable to the lower delay variance for Enet II. The peak for VTCSMA/CD is 8.9 Mbps at offered load of 26 Mbps. The throughput for VTCSMA/CD increases much more gradually than that of the other two protocols. As shown in Table 5.2, the implementation suffers greater interpacket processing delay than the other two protocols. The throughput for Ethernet and Enet II matches the offered load very closely until offered load

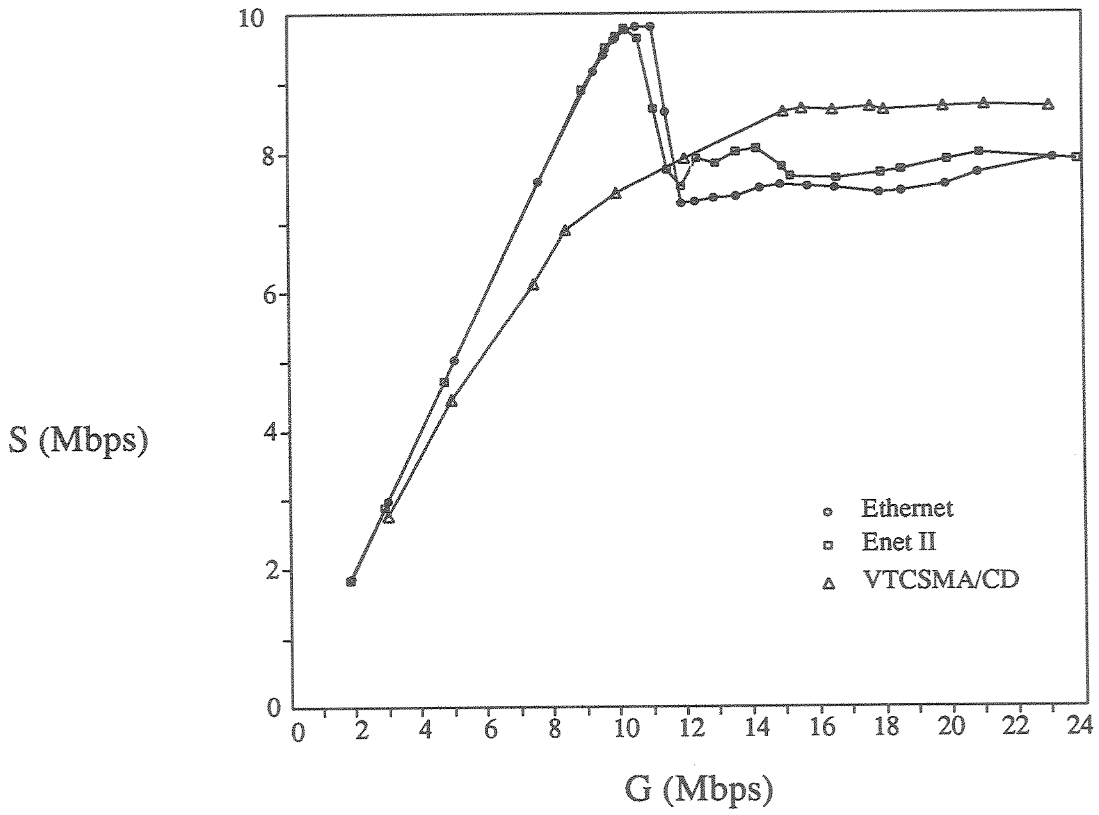


**Figure 5.1:** Throughput (S) versus offered load (G) for 1500 byte packets with fixed interpacket intervals.

increases past 9 megabits per second. At that point, contention has begun to increase and the additional delay involved in rescheduling packets which suffer collisions affects the throughput of the protocols.

Ethernet and Enet II experience a drastic drop in throughput as queueing delays from contention begin to dominate the transmission time. For Ethernet this occurs between 11.08 Mbps and 11.50 Mbps offered load. Enet II experienced this decrease at lower offered load level, between 10.64 Mbps and 11.12 Mbps. Throughput stabilizes again at approximately 7.5 Mbps for Ethernet and 7.8 Mbps for Enet II.

It must be noted that the drop in throughput for Ethernet is in part attributable to the truncated backoff scheme implemented in the 4.3BSD 3Com device driver, as discussed in Section 3.7.2. However, simulations indicate that this factor alone cannot attribute for a drop of the size observed here. The simulation studies indicate that the combination of the truncated backoff scheme and the transmission overhead noted in Table 5.1 for the Ethernet implementation comes closer to duplicating the observed behavior. Figure 5.2 illustrates this point, with curves plotted for Ethernet with regular backoff, Ethernet with regular back-off and transmission overhead included, Ethernet with truncated backoff, and Ethernet with both truncated backoff and transmission overhead included. (Recall that the queueing delay presented in the table did not include packet transmission time.) As load increases and more transmission attempts become necessary before packets are successfully transmitted, the effect is magnified, since the overhead is incurred on every attempt.

For Enet II the distribution of number of collisions suffered per packet indicates that as the throughput declines, the percentage of packets which suffer two or more collisions before transmission increases dramatically, from 4% at the peak throughput to 52% at the point where throughput begins to stabilize again. When packets suffer only one collision before transmission, it means that they were never in the deferred state. In cases where two or more collisions were suffered it is possible (and likely in cases where many more than two collisions occurred before transmission) that the packet was deferred one or more times, thus increasing the delay significantly. Also note in Table 5.1 that Enet II suffers several times the transmission overhead of Ethernet, which also contributes to the throughput drop. VTCSMA/CD does not suffer this drop in throughput, though its peak throughput is not as large as that of the other two protocols. This indicates the success of the collision avoidance strategy of the protocol, since the data indicates that in this implementation the backoff algorithm was not as effective as that employed by Ethernet and Enet II.

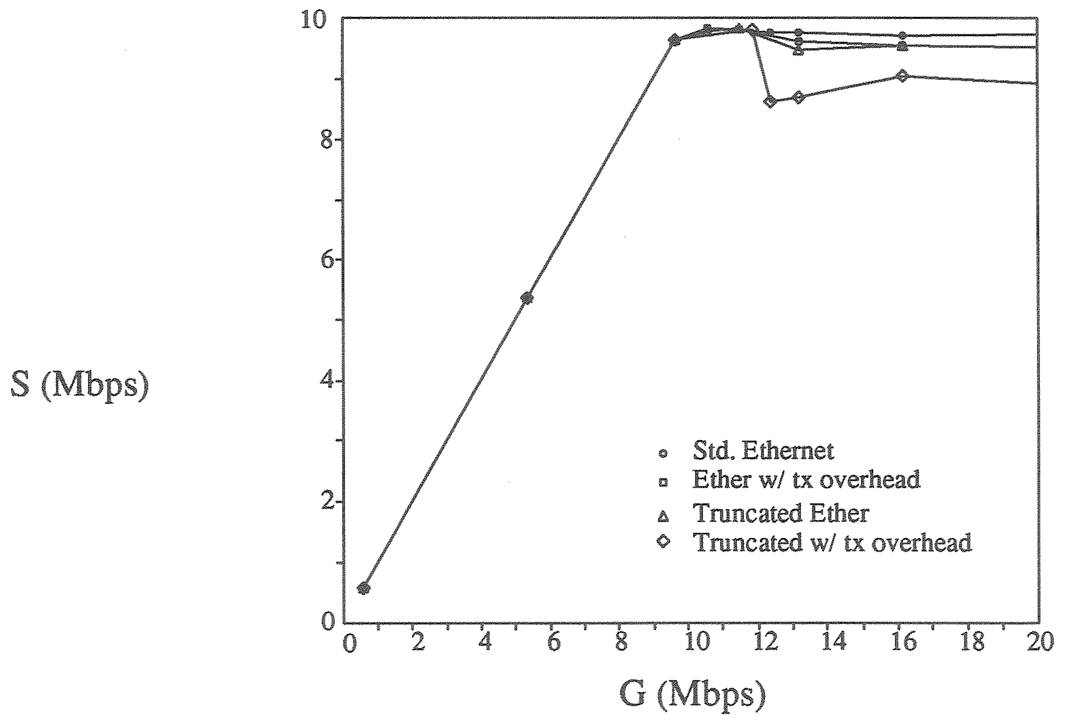


**Figure 5.2:** Simulated Ethernet Throughput (S) versus offered load (G) for 1500 byte packets with fixed interpacket intervals. Effect of various factors on drop in Ethernet throughput.

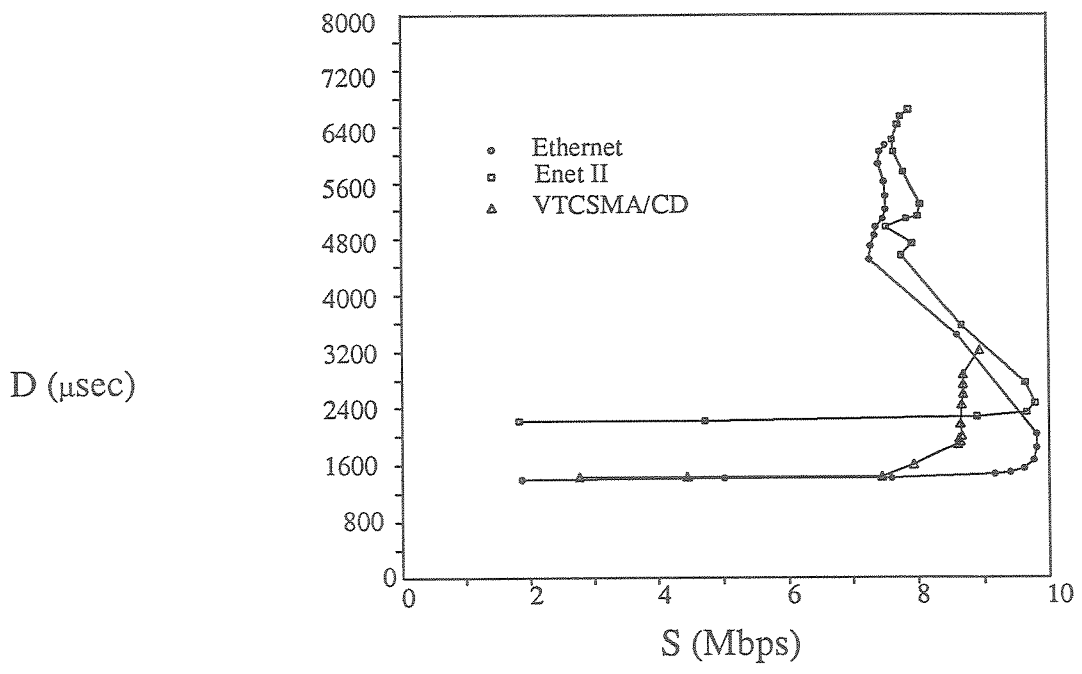


**Figure 5.3:** Queueing delay (D) versus throughput (S) for 1500 byte packets with fixed interpacket intervals.

Figure 5.3 shows the packet queueing delay versus the throughput. The x axis units are Mbps and the y axis units are microseconds. As expected, Ethernet and VTCSMA/CD outperform Enet II at low traffic loads. This is a byproduct of the initial gating delay employed by the Enet II protocol; even in the absence of contention, each packet experiences (for this implementation) a 600 microsecond delay before transmission. Ethernet and Enet II experience little increase in delay until the offered load exceeds 100% of capacity, while the VTCSMA/CD curve begins to increase at about 80% of capacity. It was expected that Enet II would show better average delay than Ethernet in the overloaded portion of the curve. However, rather than improving upon the delay of Ethernet, Enet II merely decreases the distance between the two curves. This indicates that the effect of the initial gating delay is mitigated as the collision resolution process comes into play; however, the average delay advantage predicted for Enet II was not observed. As throughput begins to decline after nearing the capacity of the channel, the distance between the curves is approximately 300 microseconds. This difference can be explained by the interval timer features discussed in Chapter 3. It should be noted that using the larger value of *r* increases the initial gating delay by 450 microseconds, so it is reasonable to expect that an exact implementation of the protocol would demonstrate the expected improvement under overload conditions. If this extra delay is subtracted out, the Enet II protocol does achieve lower average delay over a portion of the curve. The sharp increase in delay corresponds to the sharp decrease in throughput shown in Figure 5.1. The VTCSMA/CD curve crosses the other two curves in the region of heavy overload. This agrees with the higher throughput of VTCSMA/CD in this region of the curve shown in Figure 5.1.

Figures 5.4 and 5.5 show the delay variance behavior of the protocols. The x axis is the offered load in Mbps and the y axis is the delay variance in microseconds. The curve has been split into two graphs due the the large time range covered. Figure 5.4 shows the curve for the region of normal operation. The curves for Ethernet and Enet II are very similar over most of this range, matching exactly in the region between 9 and 10 Megabits/second offered load. The variance of VTCSMA/CD is initially worse than either of the other two protocols, but the curves cross at around 10 Megabits/second offered load. This low load variance behavior can be attributed to the fact that traffic from other stations increases the variance for the VTCSMA/CD implementation due to the promiscuous receives needed for virtual clock manipulation. The Enet II variance is also somewhat affected by the timer resolution. When a collision occurs, the resulting delays are up to four times longer than they need be. This accounts for the relation of the Enet II and Ethernet figures around the knee of the curve.

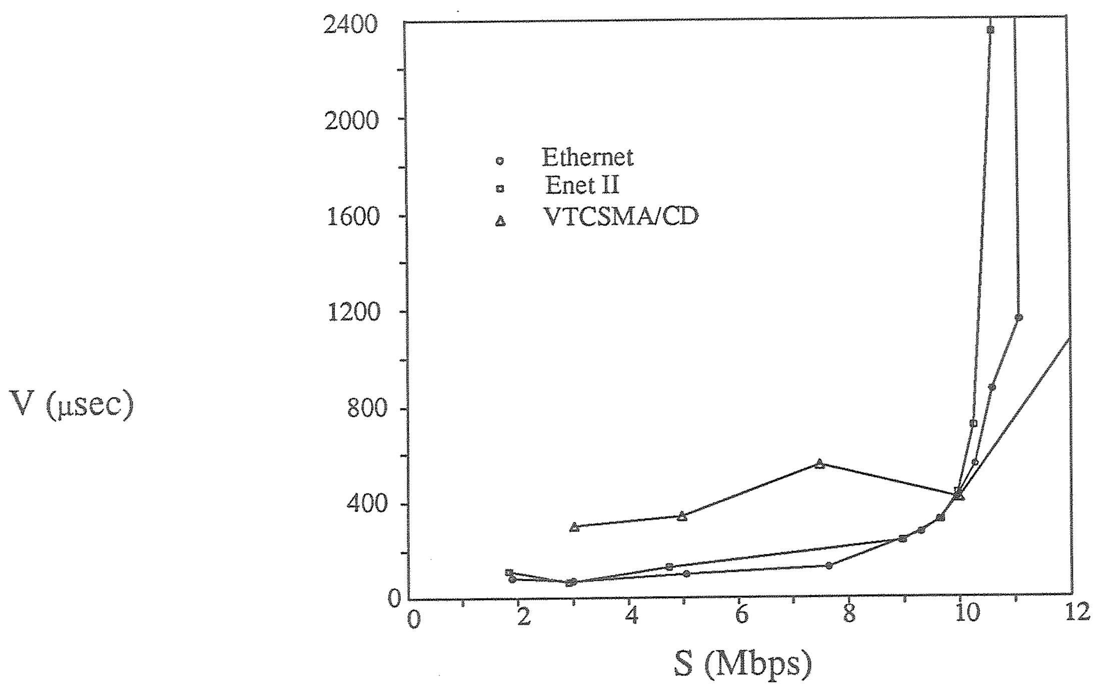


**Figure 5.4:** Queueing delay variance (V) versus offered load (G) for 1500 byte packets with fixed interpacket intervals. Normal conditions.

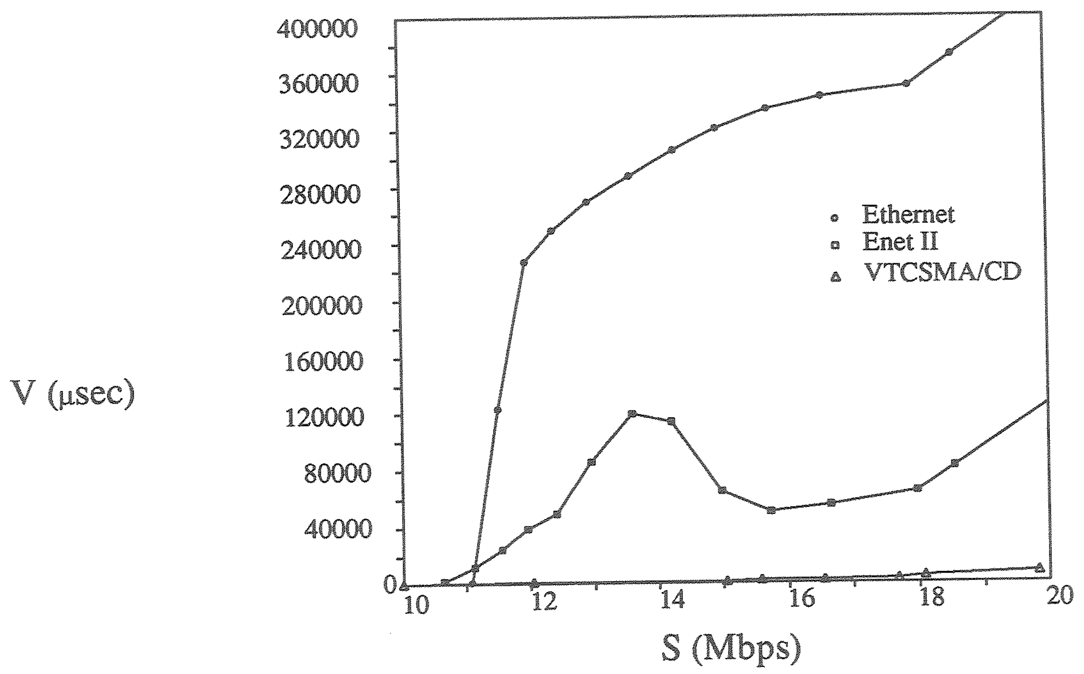


**Figure 5.5:** Queueing delay variance (V) versus offered load (G) for 1500 byte packets with fixed interpacket intervals. Overload conditions.

Figure 5.5 shows the curve for overload situations. Here Enet II is shown to achieve much lower variance than Ethernet, with VTCSMA/CD showing the lowest variance. This would indicate that both Enet II and VTCSMA/CD are more stable under heavily overloaded conditions than Ethernet. Though Enet II suffers greater variance than VTCSMA/CD, over this range of offered loads, the throughput of the two protocols is similar.

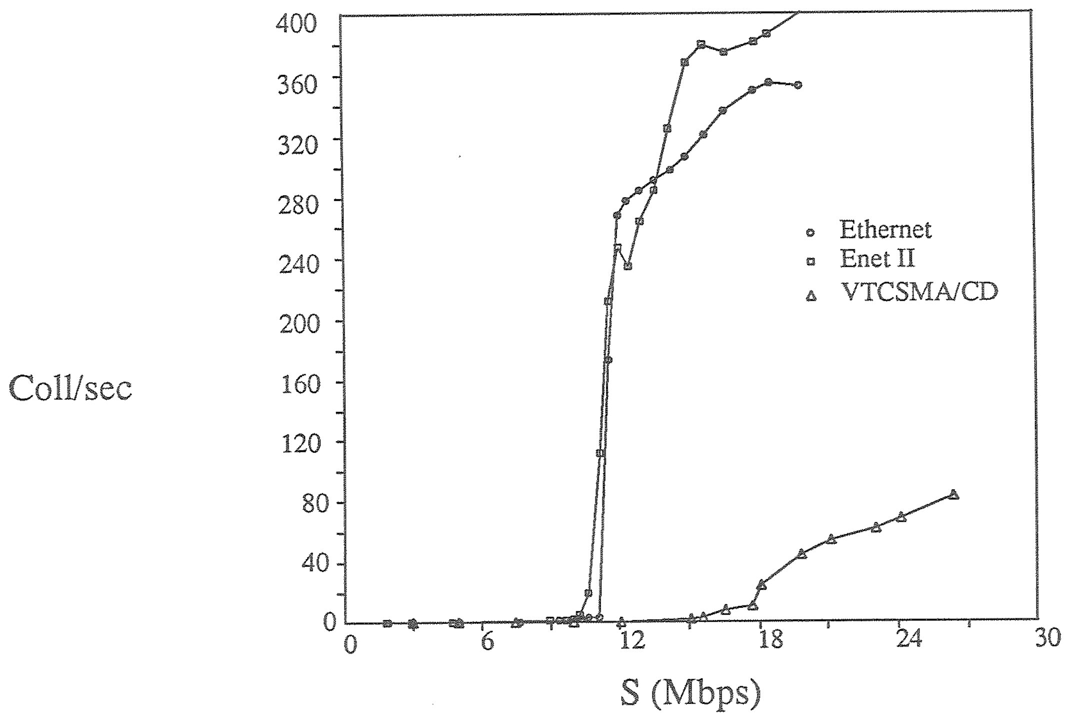


**Figure 5.6:** Collisions per second (C/s) versus offered load (G) for 1500 byte packets with fixed interpacket intervals.

Figure 5.6 shows collisions per second versus offered load for the three protocols. As indicated, VTCSMA/CD experiences far fewer collisions than the other two protocols. Ethernet and Enet II curves are very similar, showing that both protocols experience a sharp increase in collisions just after offered load exceeds the capacity of the channel. For Ethernet, the knee of the curve is at 11.5 megabits/second offered load, and for Enet II the knee occurs at 11.1 megabits/second. It should be noted that even though the two protocols generate virtually the same number of collisions under heavy loads, the variance in this region

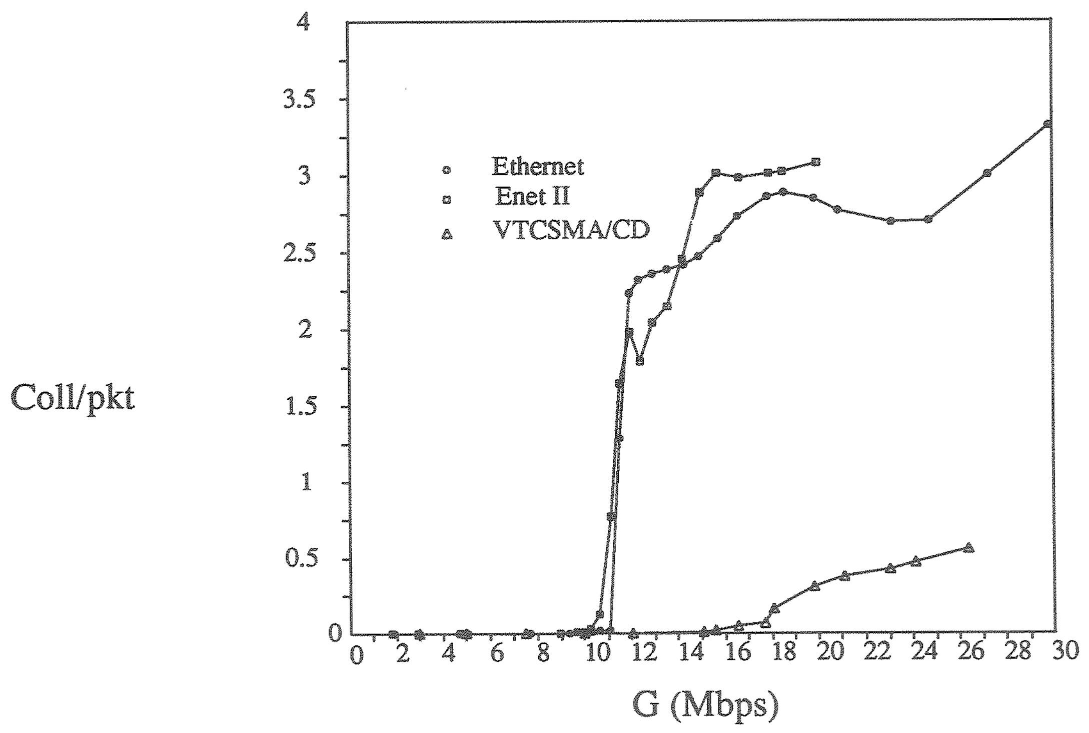


**Figure 5.7:** Collisions per packet (Coll/pkt) versus offered load (G) for 1500 byte packets with fixed interpacket intervals.

is lower for Enet II, indicating that the delay associated with each collision is more regular for Enet II.

The actions taken by protocols such as Enet II amount to a kind of binary search on the set of ready hosts in the network. If unique addresses are used to perform the subdivision at each step, a station in a network of $n$ continuously queued stations can expect to suffer $\lceil \log n \rceil$ collisions for each successfully transmitted packet. In networks where probabilistic methods are used, such as the coin flips employed in Enet II, this figure should be the average number of collisions suffered by a station in a network of $n$ continuously queued transmitters for each success. In the LANT, this number is $\lceil \log 5 \rceil = 3$. Figure 5.7 indicates that as the offered load increases, the average number of collisions suffered by each packet increases to about 3 (at an offered load of approximately 15.5 megabits per second) and then levels off, as expected. The average collisions for Ethernet, however, continue to increase. This figure also demonstrates the effectiveness of the collision avoidance properties of VTCSMA/CD.

### 5.1.2. Exponential Interpacket Intervals

The situation where a LAN host generates packets with clockwork regularity must be considered rare, though probably not nonexistent. Such a situation could conceivably occur if the interface device driver input queue were consistently full. However, evidence from the Xerox experiments and the study described in Chapter 4 indicates that the interpacket time is variable. Since many studies have assumed exponentially distributed arrival times and since it is possible to generate exponentially distributed random numbers fairly efficiently, results were collected from experiments run with exponentially distributed interpacket delays. Results for the Enet II protocol were not available for this load pattern.

Figure 5.8 shows the throughput versus offered load for 1500 byte packets with exponentially distributed interpacket intervals. Over the normal range of operation the throughput of Ethernet is better than the that of the VTCSMA/CD protocol. The peak throughput for Ethernet has dropped from the 9.8 megabits/second level achieved with fixed interpacket intervals to 7.84 megabits per second. Throughput for VTCSMA/CD increases more gradually, but the peak is at 8.46 megabits per second. It should be noted that the lower throughput is due more to the additional overhead involved in generating the exponentially distributed arrivals rather than any property of the arrival process itself. Ethernet still experiences the drop in throughput as contention becomes severe, which is attributed to the truncated backoff scheme noted in Chapter 3. At offered load of 10 megabits/second, the number of collisions suffered by Ethernet begins to approach the number of packets sent, while for VTCSMA/CD only 8% of the packets suffered collisions.

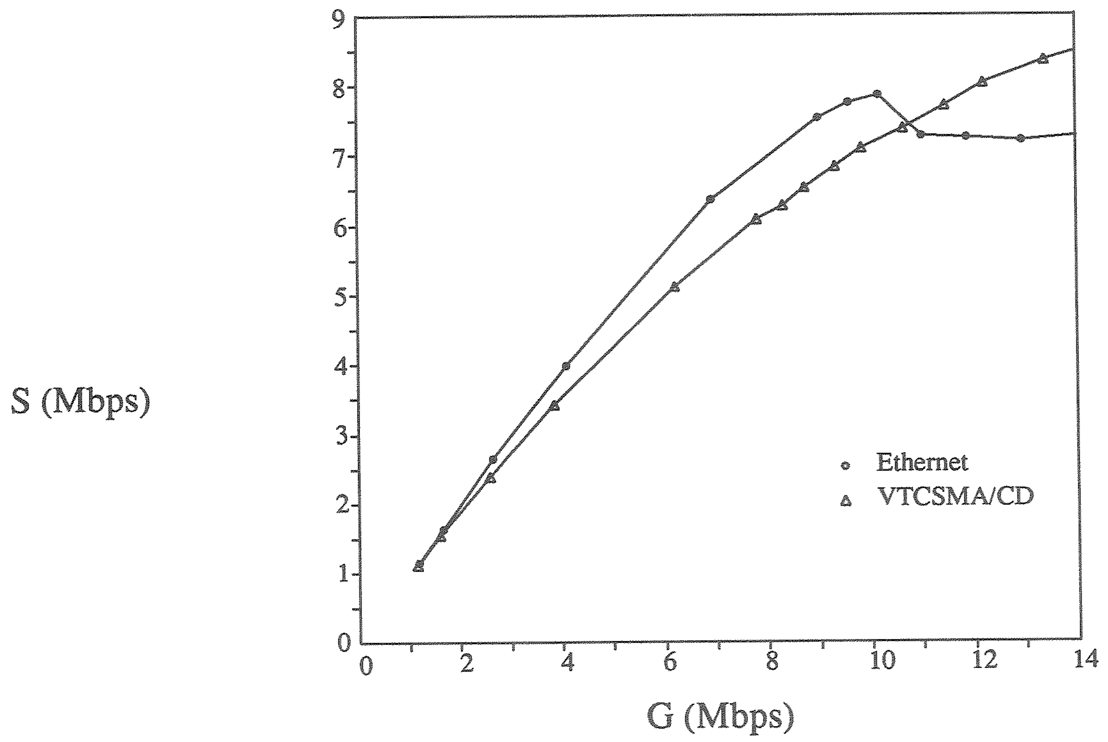


**Figure 5.8:** Throughput (S) versus offered load (G) for 1500 byte packets with exponentially distributed interpacket intervals.

Though its throughput increases more slowly, VTCSMA/CD exceeds the throughput of Ethernet under heavy overloads.

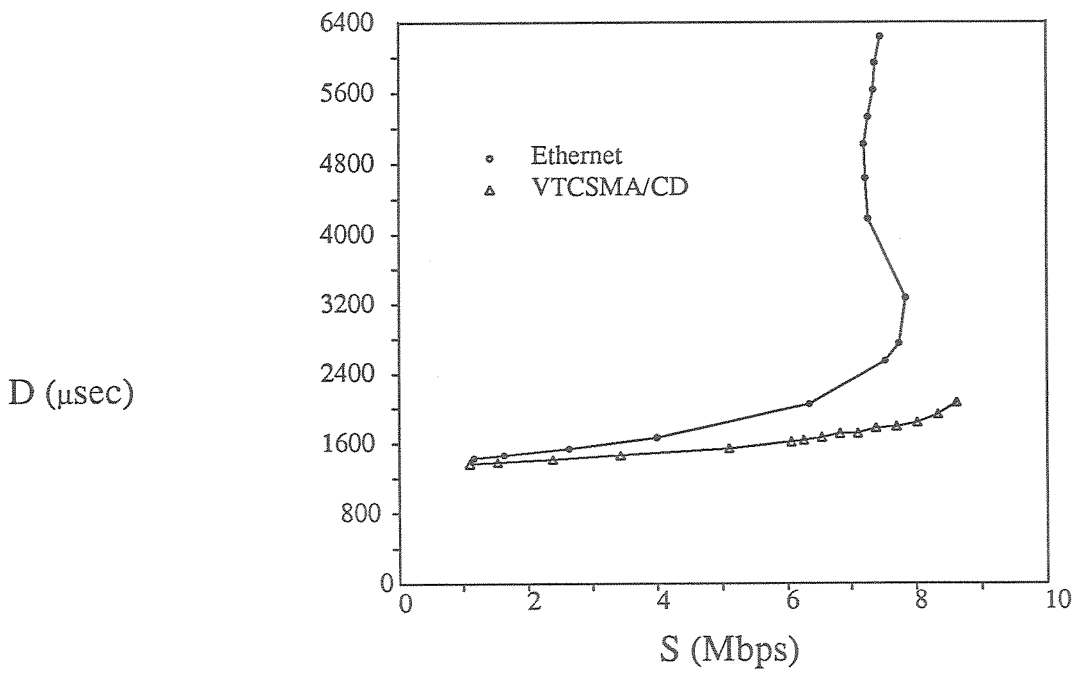


**Figure 5.9:** Queueing delay (D) versus throughput (S) for 1500 byte packets with exponentially distributed interpacket intervals.

Figure 5.9 shows the packet queueing delay versus the throughput for 1500 byte packets with exponentially distributed interpacket intervals. Under these circumstances VTCSMA/CD suffers lower average delay over the entire range of throughputs. The VTCSMA/CD curve very closely matches the curve for the protocol under a fixed interval load for the range of throughputs generated under the exponential load.

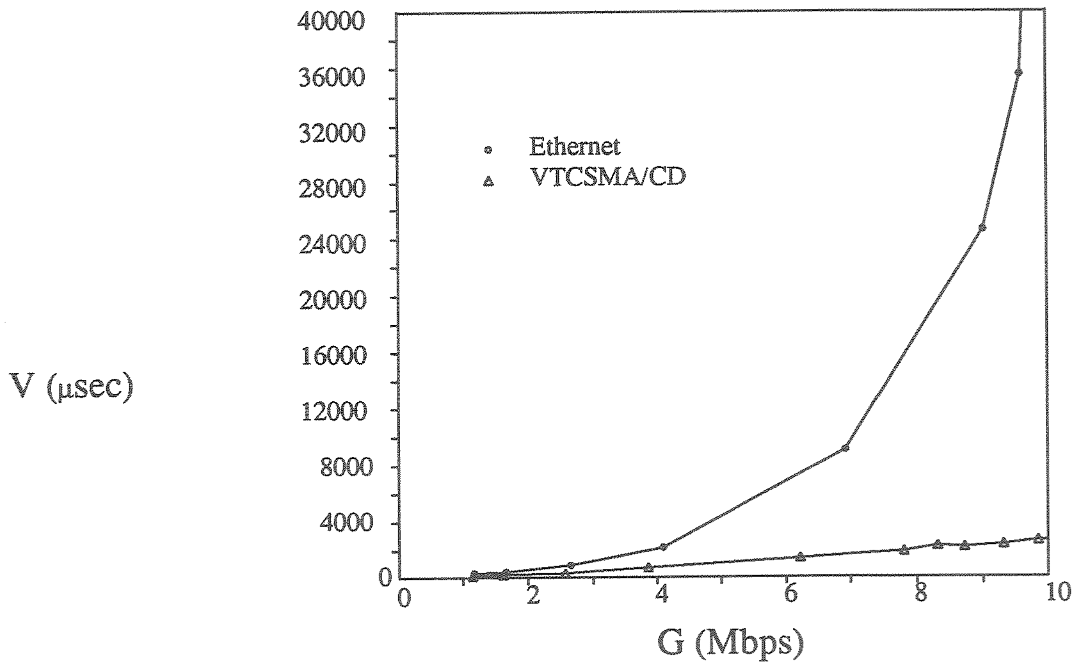


**Figure 5.10:** Delay variance (V) versus offered load (G) for 1500 byte packets with exponentially distributed interpacket intervals. Offered loads in normal operating range.

Figures 5.10 and 5.11 show the variance of the queueing delay versus the offered load. Figure 5.10 shows the variance for offered loads under 10 megabits per second. The variance for Ethernet begins to increase at around 4 megabits per second, while the variance for VTCSMA/CD increases only gradually. Figure 5.11 shows that this trend continues as the offered load increases past the capacity of the channel.

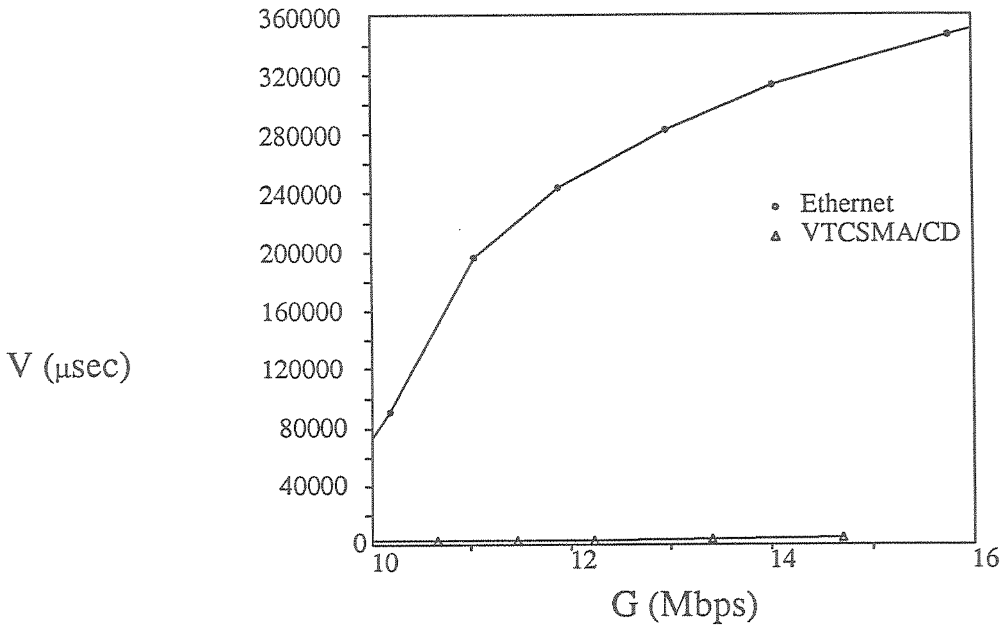


**Figure 5.11:** Delay variance (V) versus offered load (G) for 1500 byte packets with exponentially distributed interpacket intervals. Offered loads extend to overload range.

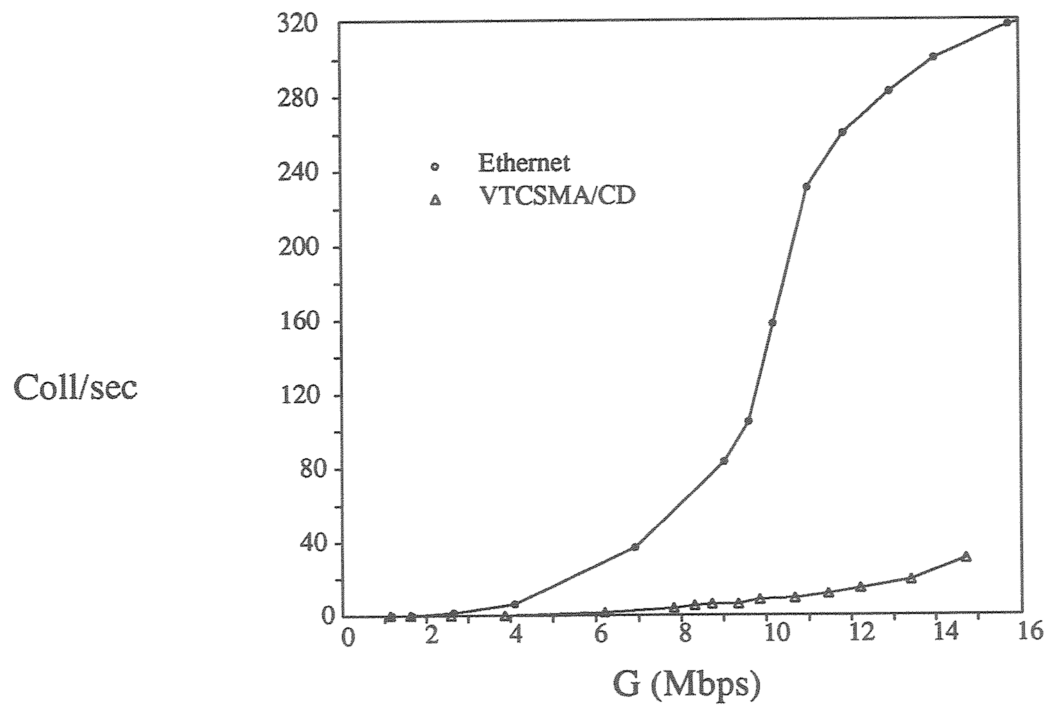


**Figure 5.12:** Collisions per second (C/s) versus offered load (G) for 1500 byte packets with exponentially distributed interpacket intervals.

Figure 5.12 shows the average collisions per second versus offered load for loads generated with 1500 byte packets and exponentially distributed interpacket intervals. The morphology of the curves is very similar to the variance curve, with the Ethernet curve diverging from the VTCSMA/CD curve at offered load of about 4.0 megabits per second. The VTCSMA/CD curve increases only slightly over the entire range of offered loads. The effect of substituting the exponential arrival process for the fixed process had a marked effect on the collision rate for the Ethernet protocol. For the fixed arrival experiment, the increase in collisions was negligible until the load approached the channel capacity, at which time it increased very rapidly. Compared with Figure 5.6, Figure 5.12 shows that contention for Ethernet began much earlier with exponentially distributed arrivals.

The introduction of exponentially distributed interpacket times affected the performance of the VTCSMA/CD implementation discussed here only slightly. This behavior is due to the fact that the VTCSMA/CD protocol imposes its own structure on the actual transmission attempts with the virtual clock, which could be far more strongly affected by the transmission time of long packets than by the rate at which the software generated the arrivals. Additionally, the promiscuous receives necessary for maintaining the virtual clock provided variations in the interpacket delays even in the case where the load generation software was instructed to provide fixed interpacket times.

## 5.2. 1024 Byte Packets

While 1500 byte packets allowed us to examine one endpoint of network behavior, measurements discussed in Chapter 3 indicate that 1500 byte packets do not usually make up a significant portion of the load. The University of Texas network carries a much larger percentage of 1024 byte packets, due to the presence of a number of diskless Sun workstations running the Sun Network Disk Protocol. (The actual length of these packets is 1072 bytes, consisting of a 1024 byte disk block and some header bytes. Since physical buffering and copying played no role in these measurements, 1024 byte packets were chosen for this load pattern for compatibility with existing studies.) The next two sections discuss the behavior of the network under a load composed of 1024 byte packets.

### 5.2.1. Fixed Interpacket Intervals

Figure 5.13 shows the throughput versus offered load curves for the three protocols transmitting fixed 1024 byte packets with fixed interpacket intervals. This graph closely resembles Figure 5.1 in the relative behaviors of the protocols and the shapes of the curves. Ethernet achieves a peak throughput of 9.67 megabits per second at an offered load of 10.79 megabits per second. The same throughput drop is observed, with throughput for Ethernet

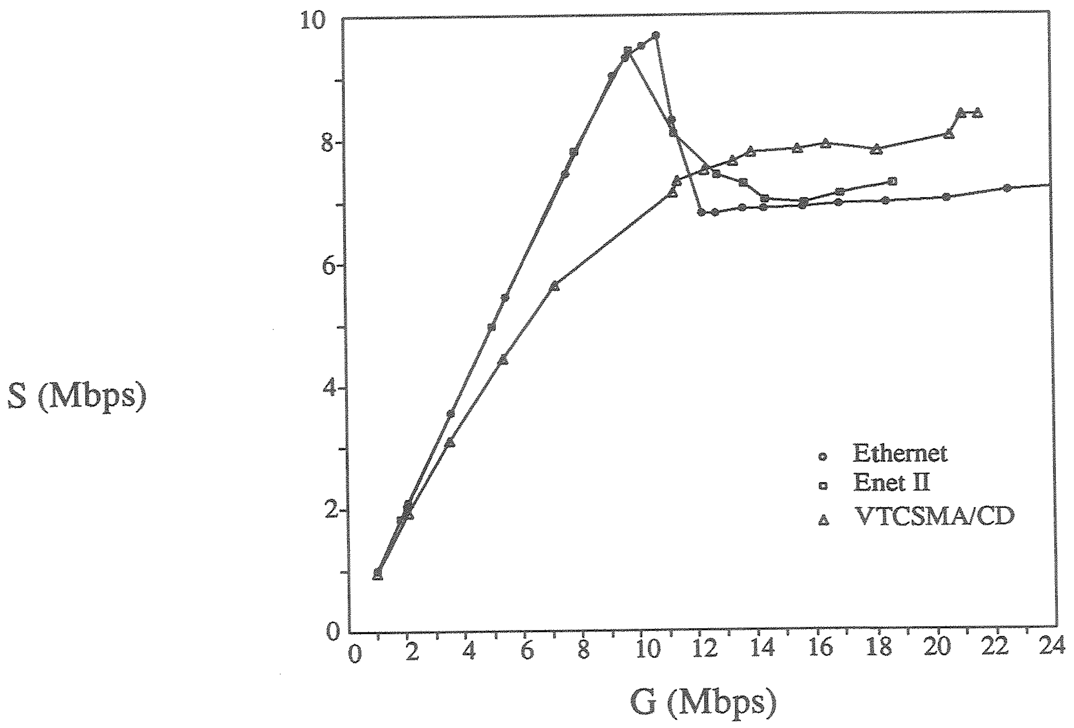


**Figure 5.13:** Throughput (S) versus offered load (G) for 1024 byte packets with fixed interpacket intervals.

stabilizing again at approximately 7 megabits per second. The peak throughput for Enet II is 9.43 megabits per second, occurring at an offered load of 9.77 megabits per second. Enet II also suffers the previously discussed drop in throughput, with throughput restabilizing at a slightly higher level than that of Ethernet. VTCSMA/CD once again experiences a slower increase in throughput, but achieves a higher stable throughput under heavy offered loads. The peak throughput is 8.38 megabits per second at an offered load of 21 megabits per second. For Ethernet, throughput matches the offered load very closely until the load reaches about 9 megabits per second. For Enet II, the dropoff occurs at slightly lower load, around 8 megabits per second. These figures are lower than those observed for the 1500 byte packet case, indicating that contention is increased for the smaller packet size.

Figure 5.14 shows the queueing delay versus throughput for 1024 byte packets with fixed interpacket intervals. Again, the form of the curves is similar to those in Figure 5.3. Under low loads, there is approximately an 800 microsecond difference between the Enet II curve and the curves for Ethernet and VTCSMA/CD, due in part to the gating delay of the Enet II protocol. The gap persists after loads have reached the saturation point of the channel. Enet II gains the advantage only in the extremely heavily loaded experiments. The delay for VTCSMA/CD begins to increase at slightly lower offered load than for the 1500 byte experiments. This is accounted for by the fact that smaller packet sizes mean more packets are sent for a given load level thus increasing the number of collisions encountered.

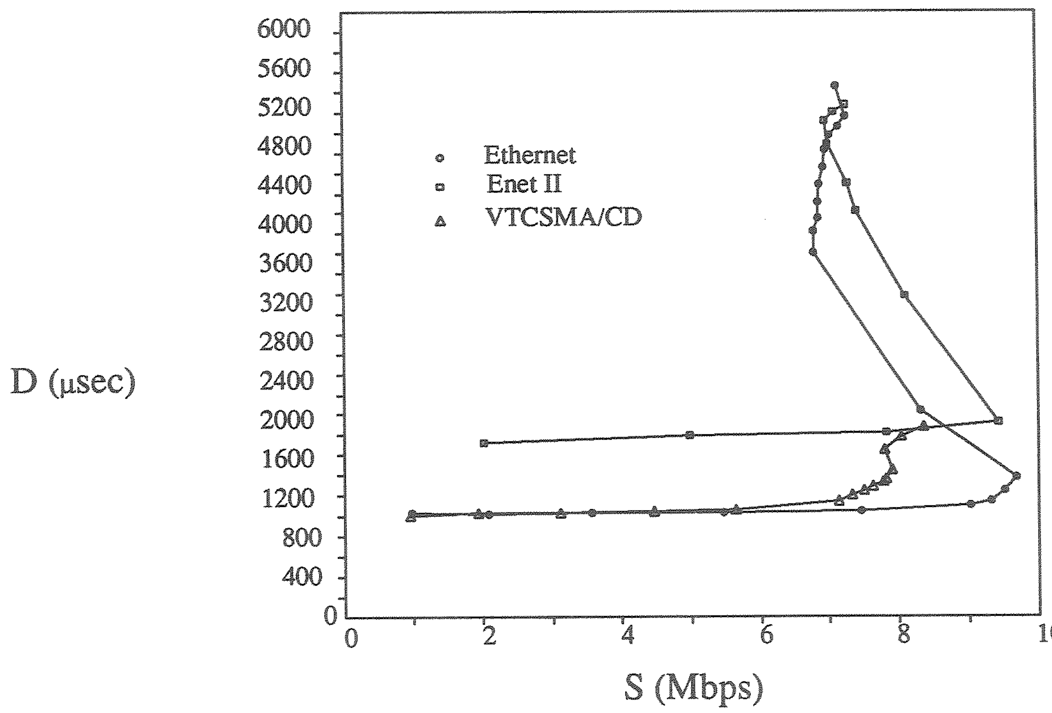


**Figure 5.14:** Queueing delay (D) versus throughput (S) for 1024 byte packets with fixed interpacket intervals.

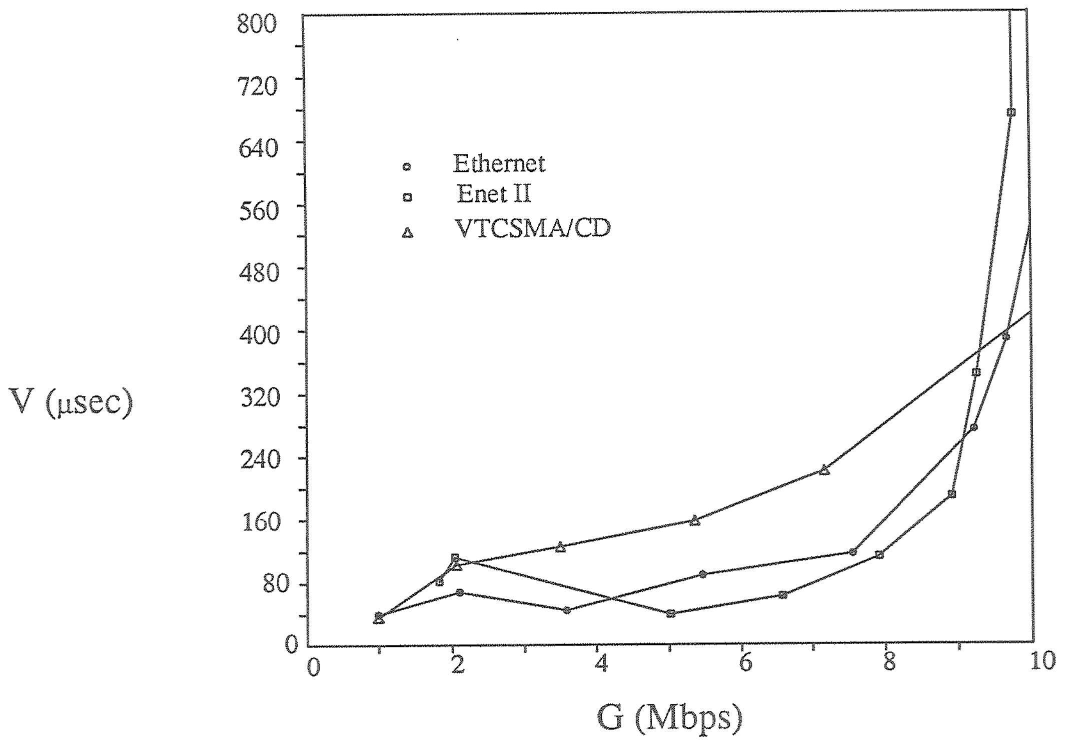


**Figure 5.15:** Delay variance (V) versus offered load (G) for 1024 byte packets with fixed interpacket intervals. Offered loads in normal operating range.

Figures 5.15 and 5.16 show the variance of queueing delay versus offered load for the three protocols over the normal range of operation and over the entire range of generated loads, respectively. As in Figure 5.4, the curves for Ethernet and Enet II are very similar in this range, with Enet II holding a slight advantage over a large portion of the range. VTCSMA/CD also shows slightly higher variance in this region of the curve, as it did in the 1500 byte packet experiments. Figure 5.16 shows characteristics very similar to Figure 5.5, with VTCSMA/CD showing very low variance in the overloaded portion of the curve. Enet

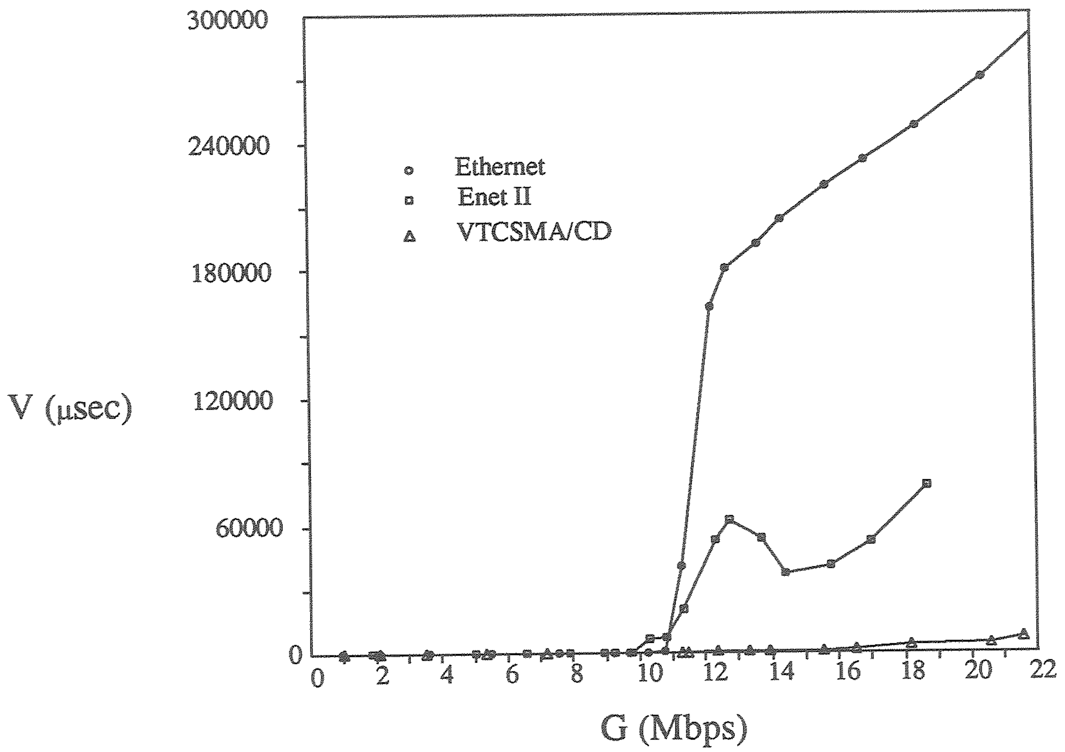


**Figure 5.16:** Delay variance (V) versus offered load (G) for 1024 byte packets with fixed interpacket intervals. Offered loads extend into overload range.

II significantly improves on the behavior of Ethernet, as was the case with the 1500 byte packet experiments.

Figure 5.17 shows the collisions per second versus the offered load for the three protocols when 1024 byte packets are transmitted at fixed intervals. The curves are similar to those shown in Figure 5.6 for 1500 byte packets, but with an expected increase in the maximum values. Ethernet suffers 352 collisions per second at offered load of 20 megabits per second when transmitting 1500 byte packets, and and about 420 collisions per second when transmitting 1024 byte packets. Enet II suffers 400 collisions per second with 1500 byte

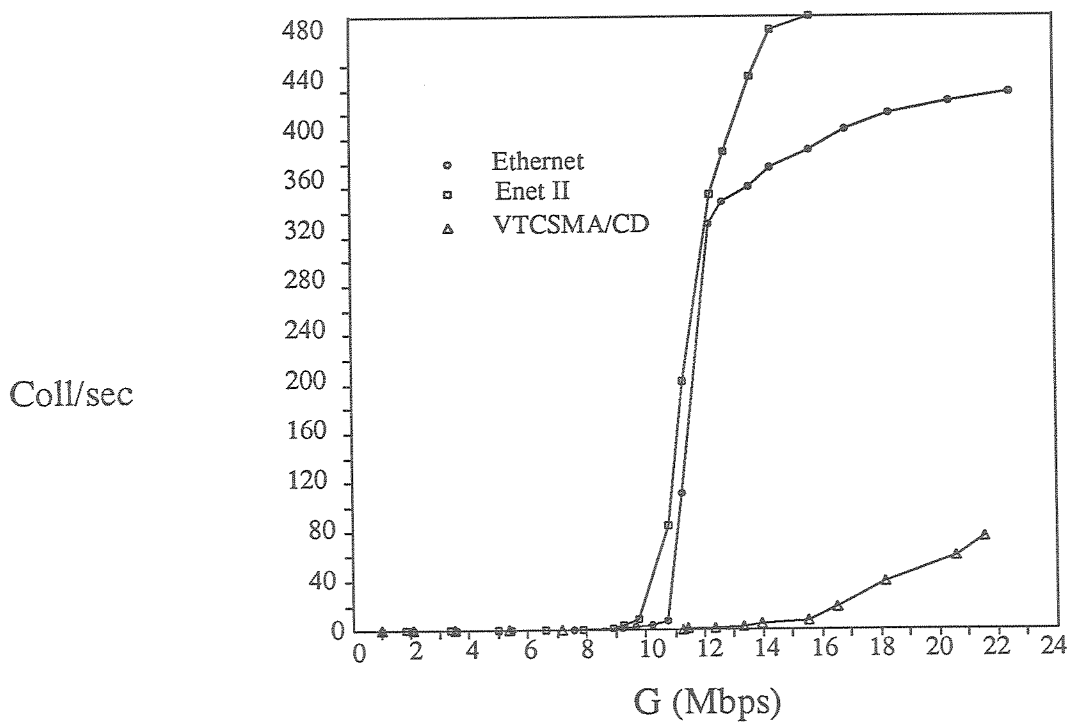


**Figure 5.17:** Collisions per second (Coll/sec) versus offered load (G) for 1024 byte packets with fixed interpacket intervals.

packets and 531 with 1024 byte packets. The collision behavior of VTCSMA/CD remains virtually unchanged. As Figure 5.16 indicates, the collisions in Enet II do not cause the same increase in variance that collisions in Ethernet do, since the collisions produced by Enet II are temporally "closer together" as a result of the resolution algorithm.

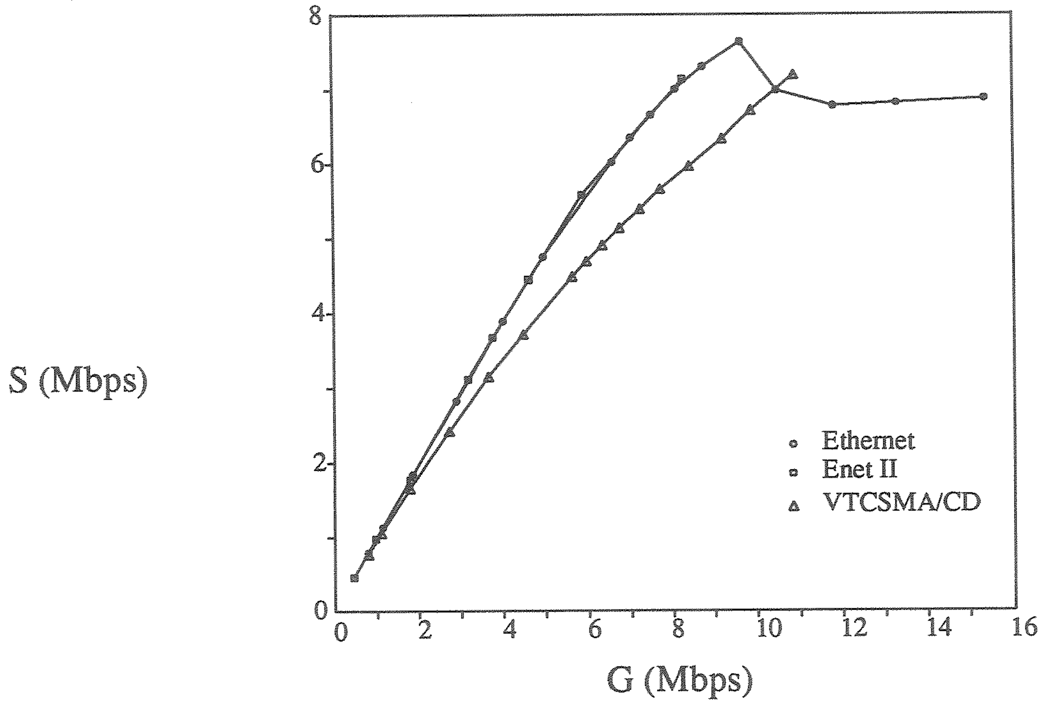


**Figure 5.18:** Throughput (S) versus offered load (G) for 1024 byte packets with exponentially distributed interpacket intervals.

### 5.2.2. Exponential Interpacket Intervals

Figure 5.18 shows throughput versus offered load for 1024 byte packets with exponentially distributed interpacket intervals. The main difference between the curves for 1500 bytes and 1024 bytes with exponential interpacket intervals is the lower peak throughput values. For Ethernet, throughput peaks at 7.62 megabits per second at an offered load of 9.63 megabits per second. Enet II peaks at 7.13 megabits per second at offered load of 8.27 megabits per second. VTCSMA/CD peaks at 7.18 megabits per second at offered load 10.90 megabits per second. The peaks for Enet II and VTCSMA/CD occurred at the highest load the generating software was capable of producing. As with previous load patterns, the Enet II and Ethernet curves were nearly identical over the normal operating range.

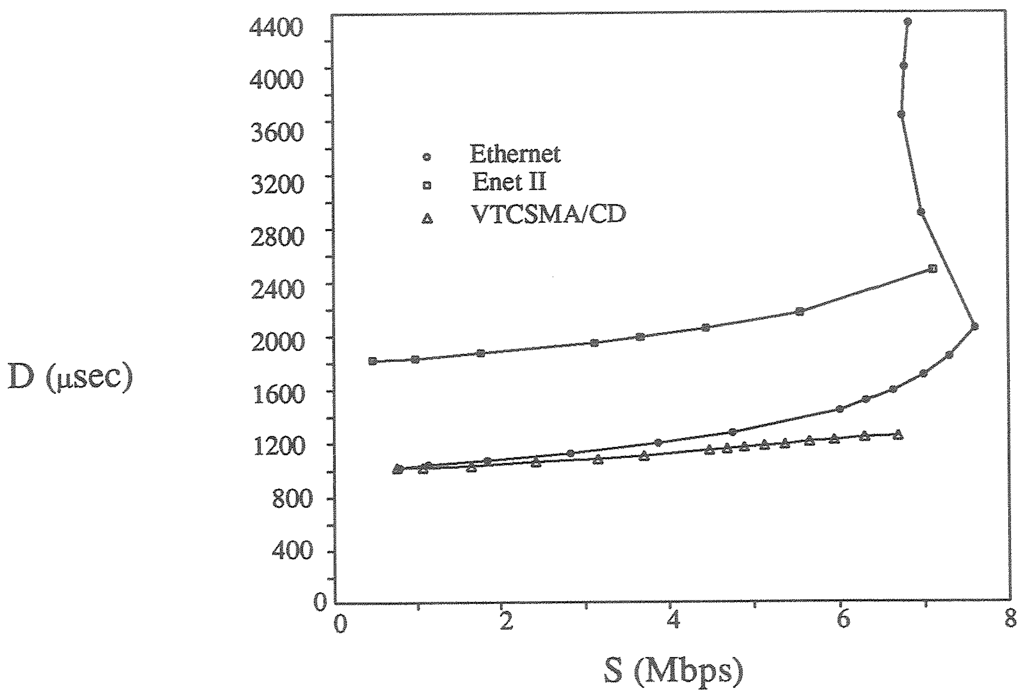


**Figure 5.19:** Packet queueing delay (D) versus throughput (S) for 1024 byte packets with exponentially distributed interpacket intervals.

Figure 5.19 shows packet delay versus throughput for 1024 byte packets with exponentially distributed interpacket times. Though it was not possible to generate overloads for the Enet II and VTCSMA/CD protocols, the relationships between the curves are similar to those for fixed interpacket intervals. The gating delay for Enet II is still evident at low loads. The distance between the two curves is approximately 800 microseconds for loads up to 6 megabits per second. VTCSMA/CD experiences less delay over the entire range of throughputs, with an increased advantage as Ethernet and Enet II begin to suffer contention at higher load levels.

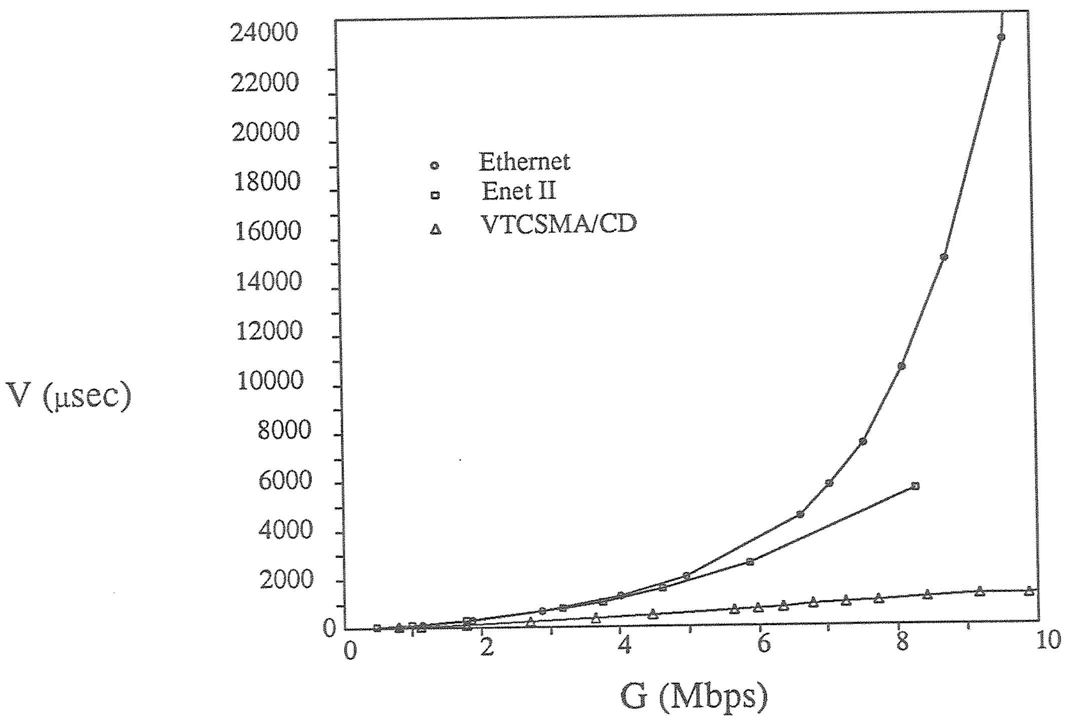


**Figure 5.20:** Packet delay variance (V) versus offered load (G) for 1024 byte packets with exponentially distributed interpacket intervals.

Figure 5.20 shows the variance of packet delay versus the offered load for 1024 byte packets with exponentially distributed interpacket times. The variance behavior of the three protocols is clearly indicated; the three protocols display low delay variance at low loads, but as contention begins and variance increases for Ethernet and Enet II, variance for VTCSMA/CD remains low. Though contention increases the variance for Enet II, the increase affects the variance of the packet delay less strongly than in the case of Ethernet. This behavior is evident at lower offered load for the exponentially distributed arrivals.

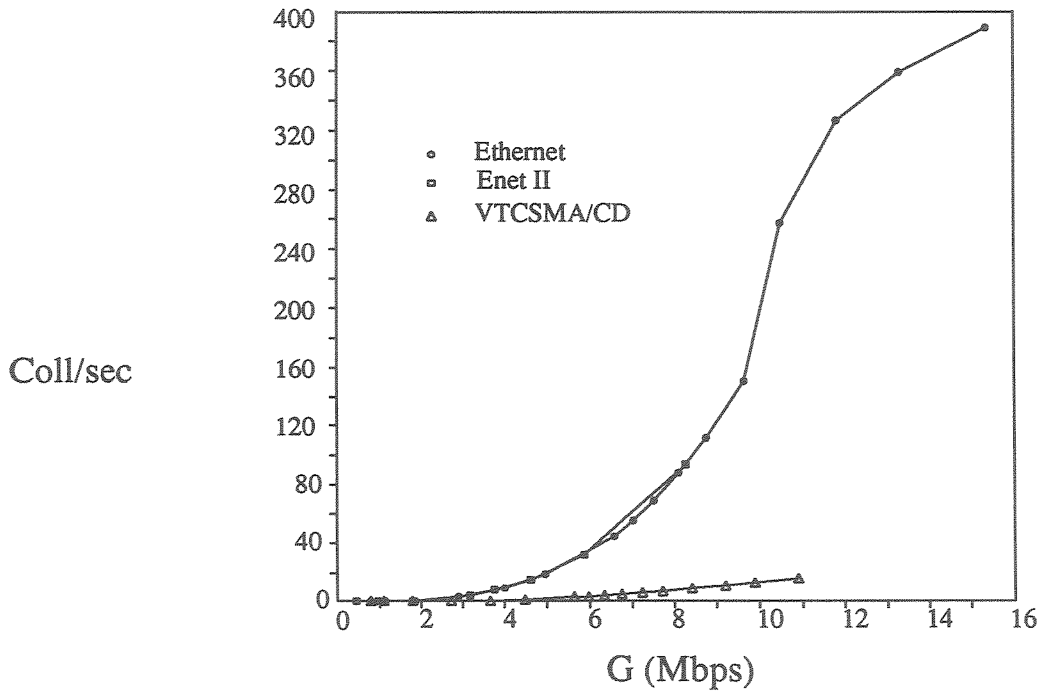


**Figure 5.21:** Collision rate (Coll/sec) versus offered load (G) for 1024 byte packets with exponentially distributed interpacket intervals.

Figure 5.21 shows the collision rate versus the offered load for 1024 byte packets with exponentially distributed interpacket times. The relative behaviors of the three protocols is the same as in the fixed interval experiments; however, as noted with respect to Figure 5.20, the increase in contention for Ethernet and Enet II happens at lower offered load and less suddenly than was the case in the fixed interval experiments. Compared to Figure 5.18, the collision rate begins its increase at lower offered load, as was the case for the 1500 byte packet experiments. Figure 5.21 confirms that this effect is present for the Enet II protocol as well as the Ethernet protocol. The collision avoidance mechanism of VTCSMA/CD is unaffected by the switch to exponentially distributed arrivals, as previously discussed.

## 5.3. Packet Mixture 1

The packet length mixture experiments demonstrate the effect of increasing the number of small (relatively speaking) packets in the offered traffic. In order to push the protocols as close to their capacity as possible, the small packets were 275 bytes in length. Packet mixture 1 consists of 75% 1500 byte packets and 25% 275 byte packets, for an average packet length of 1194 bytes. The study described in Chapter 4 indicated that the small packets on the UT network were just over 72 bytes on the average, but it was difficult to push the testbed into the overloaded range using mixtures with small packets of that size. The size used allowed overloads to be generated for two of the three packet mixtures and also clearly demonstrated the effects of a mixture of packet sizes on the behavior of the protocols. As noted in Table 5.1, generating this mixture of packet lengths causes an extra 300 to 600 microseconds of delay in the setup for each packet. This, in addition to the higher per byte overhead of sending smaller packets results in lower maximum offered loads for the protocols. The peak offered load for Ethernet drops from 33 megabits per second to 22 megabits per second; for Enet II the drop is from 23 megabits per second to 17 megabits per second; and for VTCSMA/CD the maximum offered load drops from 23 megabits per second to 19 megabits per second. The inclusion of smaller packets also increases the number of packet transmissions attempted per unit time, since less time is spent in actual successful transmissions, thus increasing the possibility for collisions.

### 5.3.1. Fixed Interpacket Intervals

Figure 5.22 shows throughput versus offered load for the three protocols when the traffic is composed of 75% 1500 byte packets and 25% 275 byte packets. The form of the curves shows a resemblance to the corresponding curves in Figure 5.1, but the achieved peak throughput is lower, and the features of the curve are less pronounced. The throughput drop observed for previous load patterns is still evident but is not nearly so dramatic. The peak throughput for Ethernet is 7.36 megabits per second at offered load 7.87. The peak for Enet II is 7.43 megabits per second at an offered load of 10.06. VTCSMA/CD again has a

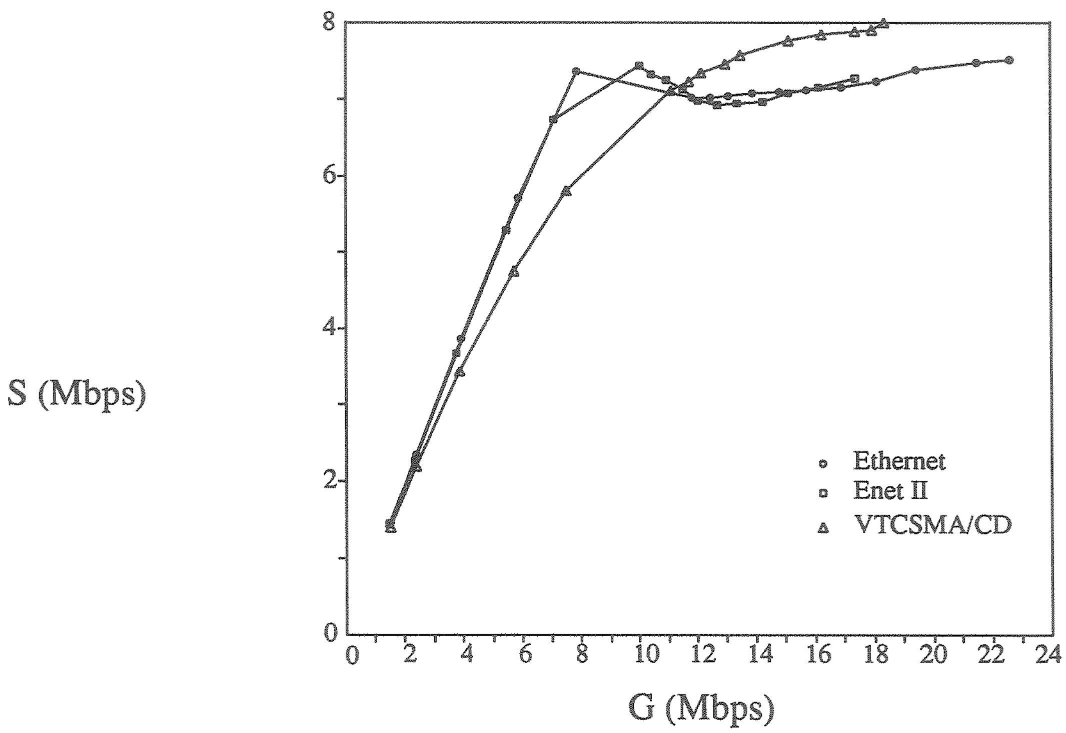


**Figure 5.22:** Throughput (S) versus offered load (G) for packet mixture 1 with fixed interpacket intervals.

more gradual increase in throughput, but achieves a stable throughput of about 8.2 megabits per second at around 14 megabits per second offered load. Contention begins to affect the throughput for Ethernet between 5 and 6 megabits per second offered load; up to that load level, throughput matches offered load very closely for Ethernet. For Enet II the effect occurs at around the same load level, though the correspondence is not quite as close at lower levels due to the gating delay for the protocol.

Figure 5.23 shows packet queueing delay versus throughput for packet mixture 1. The curves are very similar to those shown in figures 5.3 and 5.14, the principal difference being

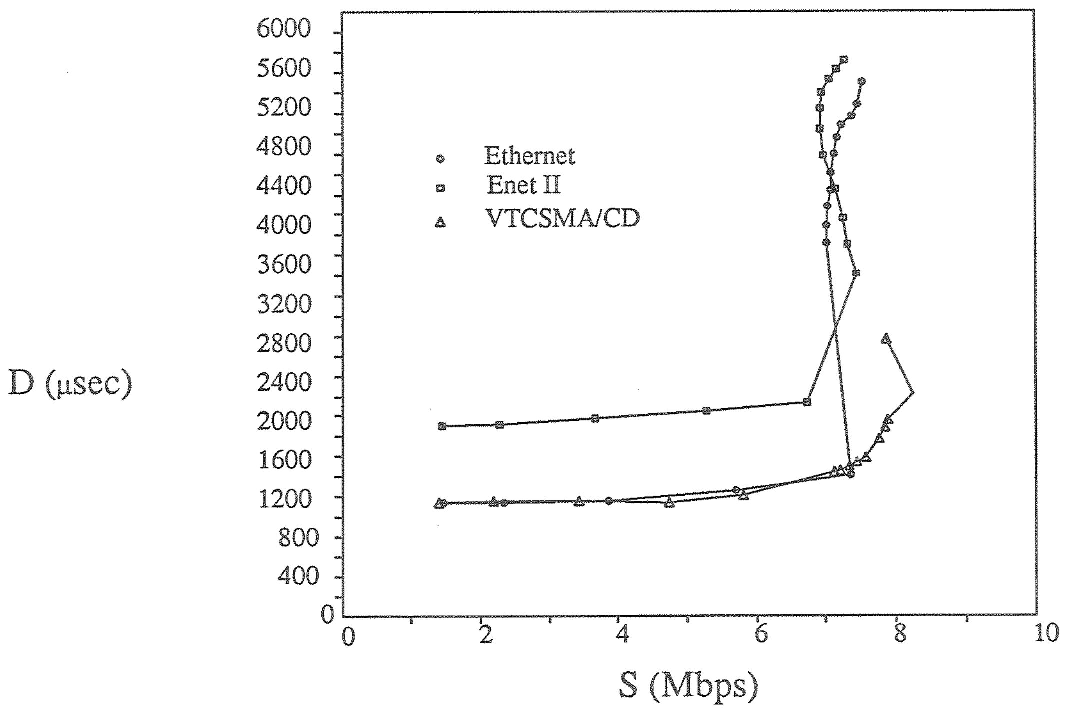


**Figure 5.23:** Packet queueing delay (D) versus throughput (S) for packet mixture 1 with fixed interpacket intervals.

that the protocols did not achieve the throughput level with the packet mixture that was observed with fixed sized packets. The bistable behavior, the gating delay, and the superiority of Enet II over Ethernet under overloaded condition are still evident.

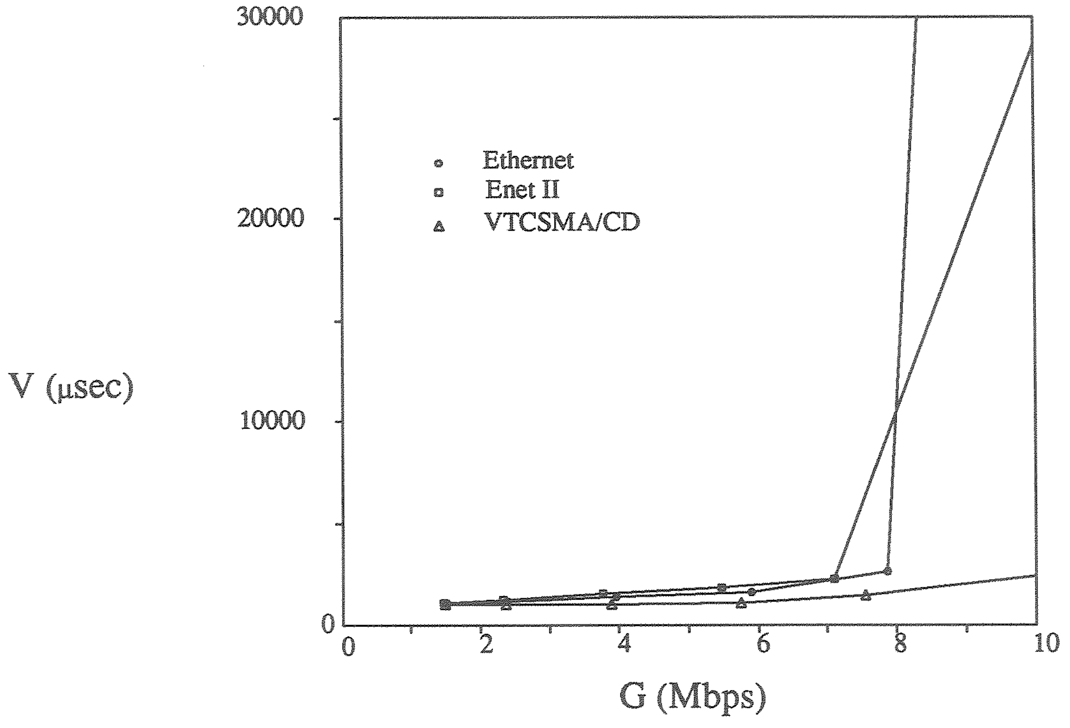


**Figure 5.24:** Packet queueing delay variance (V) versus offered load (G) for packet mixture 1 with fixed interpacket intervals. Normal operating range.

Figures 5.24 and 5.25 show the variance of packet delay versus the offered load for packet mixture 1 with fixed interarrival times. For this mixture of packets, there is very little difference between the protocols for loads under 7 megabits per second. The behavior for loads over 7 megabits per second is consistent with that observed for other loads. The Enet II curve crosses the Ethernet curve at a lower offered load than in the experiments with fixed packet sizes. Figure 5.25 indicates that the improved variance behavior of Enet II is evident at lower offered loads than in the fixed packet size experiments. The introduction of smaller packets has increased the possibility of collisions, thus increasing the variance for Ethernet at lower offered load. As before, VTCSMA/CD demonstrates very low

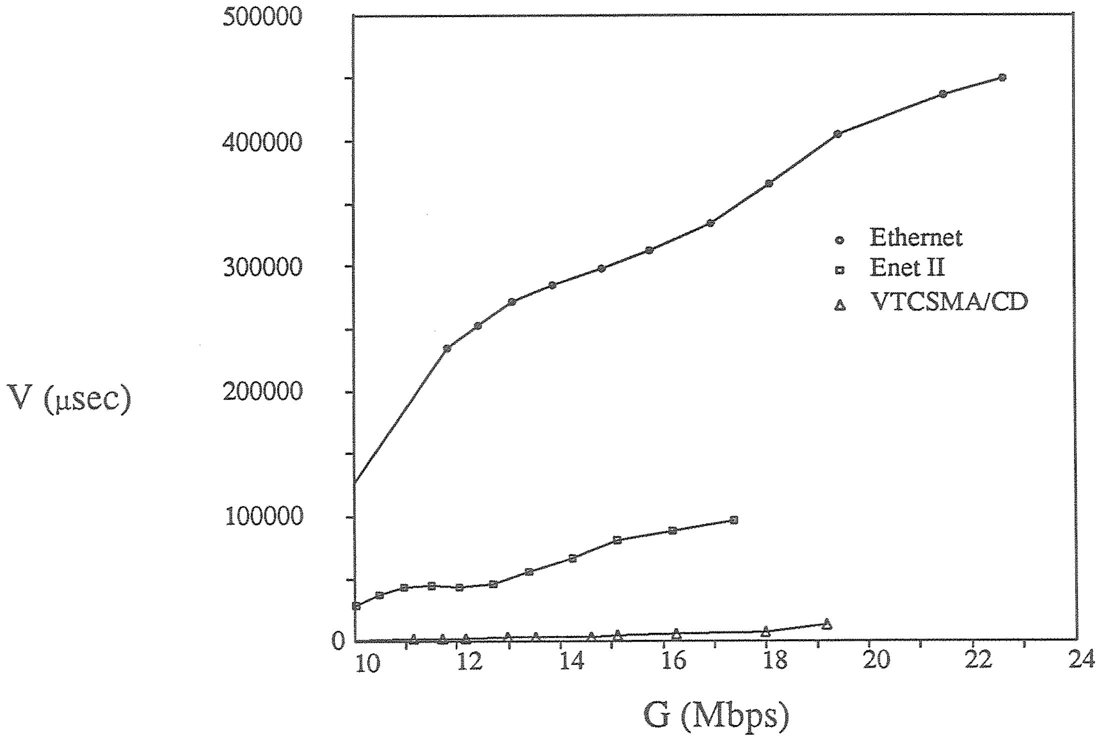


**Figure 5.25:** Packet queueing delay variance (V) versus offered load (G) for packet mixture 1 with fixed interpacket intervals. Overload range.

variance due to the low collision count.

Figure 5.26 shows the collision rate versus the offered load for packet mixture 1 with fixed interpacket intervals. This figure also indicates that contention increases at lower offered load with the addition of smaller packets for Ethernet and Enet II. The behavior of VTCSMA/CD is essentially unchanged.

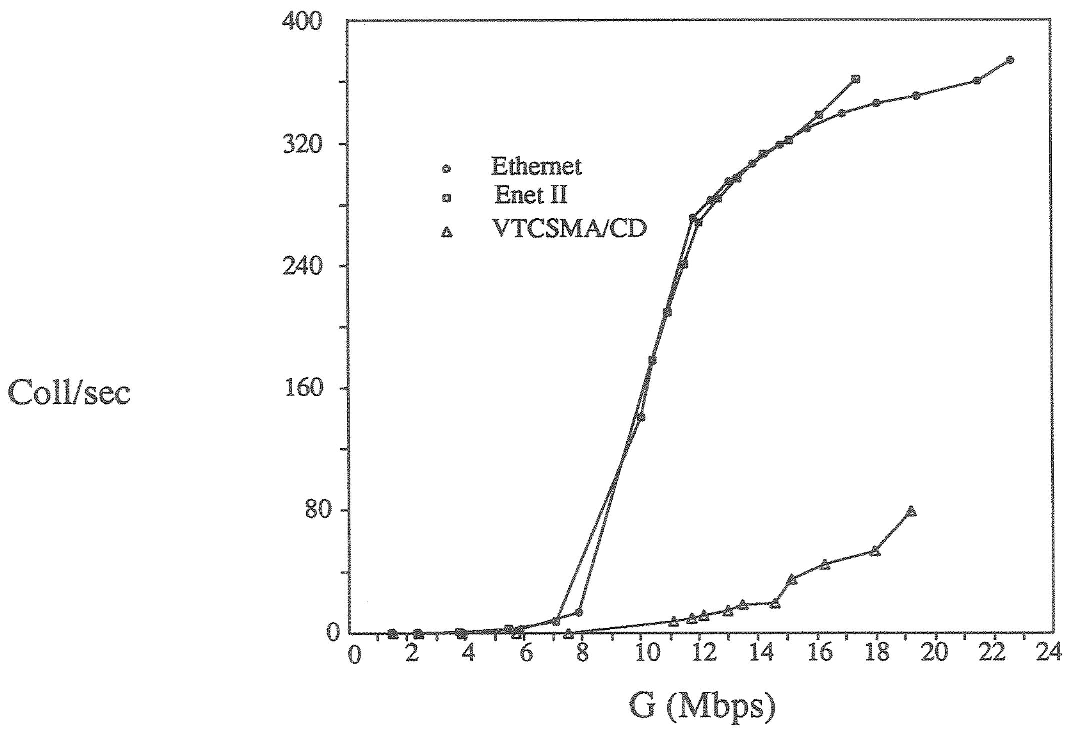


**Figure 5.26:** Collision rate (Coll/sec) versus offered load (G) for packet mixture 1 with fixed interpacket intervals.

## 5.3.2. Exponential Interpacket Intervals

Figure 5.27 shows the throughput versus offered load for packet mixture 1 with exponentially distributed interpacket intervals. The basic features of the graph are similar to those observed for the 1024 byte packets with exponential intervals. The peak throughput for Ethernet is 7.53 megabits per second, which is higher than the observed peak for fixed intervals. However, the peak for fixed intervals occurs at 7.87 megabits per second while the peak for exponential intervals occurs at 9.53 megabits per second. Since there are no other data points for the fixed interval case before 11.84 megabits per second

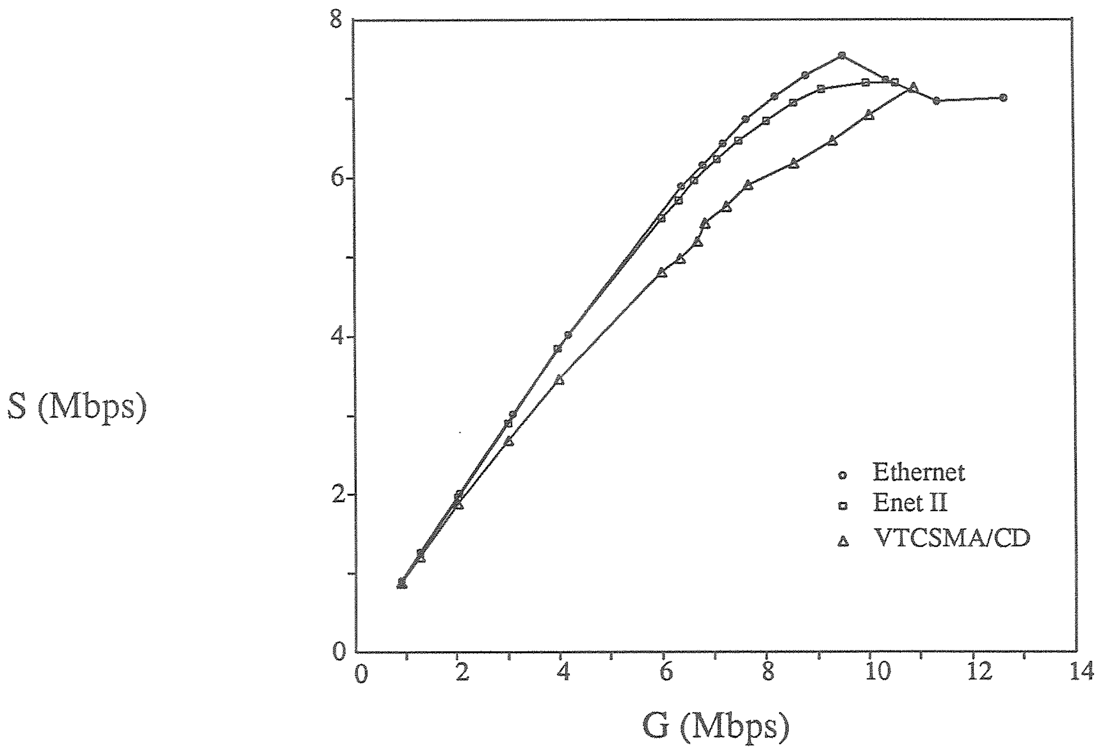


**Figure 5.27:** Throughput (S) versus offered load (G) for packet mixture 1 with exponentially distributed interpacket intervals.

offered load, it is more instructive to consider the throughput achieved at like offered load levels. The observed throughput for exponential intervals at offered load of 7.67 megabits per second is 6.72 megabits per second. It is also notable that the Enet II curve shows throughput degradation at lower offered load than in the previously discussed experiments. This effect occurs because the timer resolution for Enet II means larger delays as contention begins relative to the delay behavior of Ethernet in light contention. This conclusion is borne out by the variance behavior of the two protocols for this portion of the curve. (This effect is more readily seen in figures 5.34 and 5.36.) The VTCSMA/CD curve is similar to those observed for exponential intervals and fixed packet lengths. Also, very little difference is observed in the performance of VTCSMA/CD with fixed and exponential inter-packet intervals for this load pattern, as previously discussed.

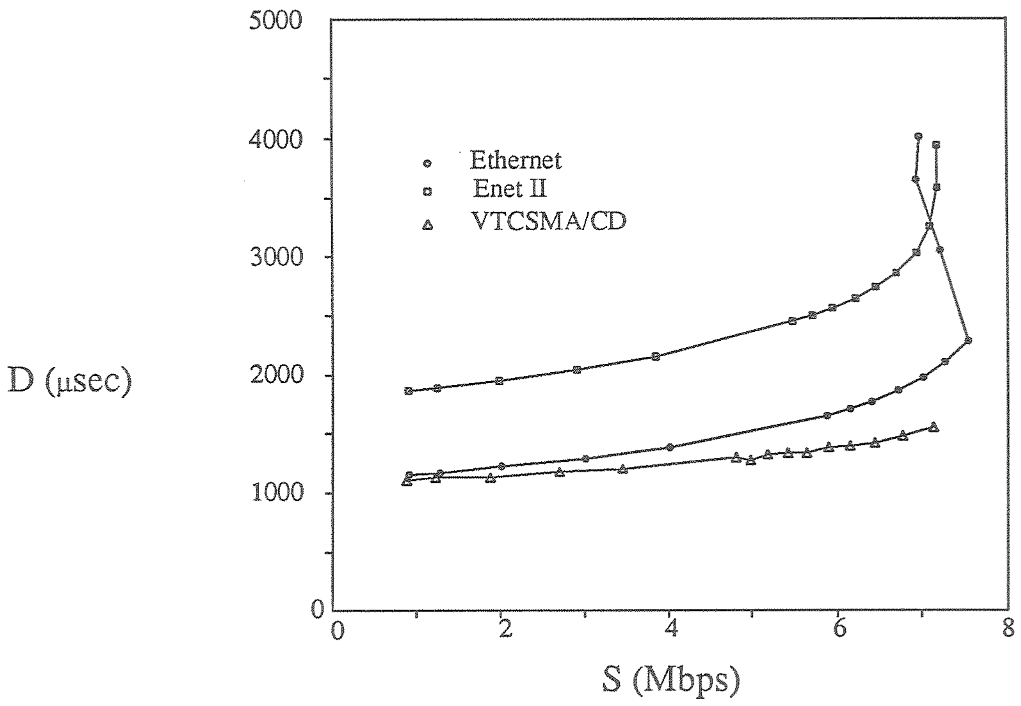


**Figure 5.28:** Packet queueing delay (D) versus throughput (S) for packet mixture 1 with exponentially distributed interpacket intervals.

Figure 5.28 shows the packet queueing delay (D) versus the throughput (S) for packet mixture 1 with exponentially distributed interpacket intervals. The shapes of the curves are very similar to those seen in figure 5.19, with the difference that the introduction of small packet has caused the delay to increase at lower offered load for all three protocols. The gating delay for Enet II is still in evidence, with the average distance between the Enet II curve and the other two curves being approximately 800 microseconds in the normal operating range of the curve.

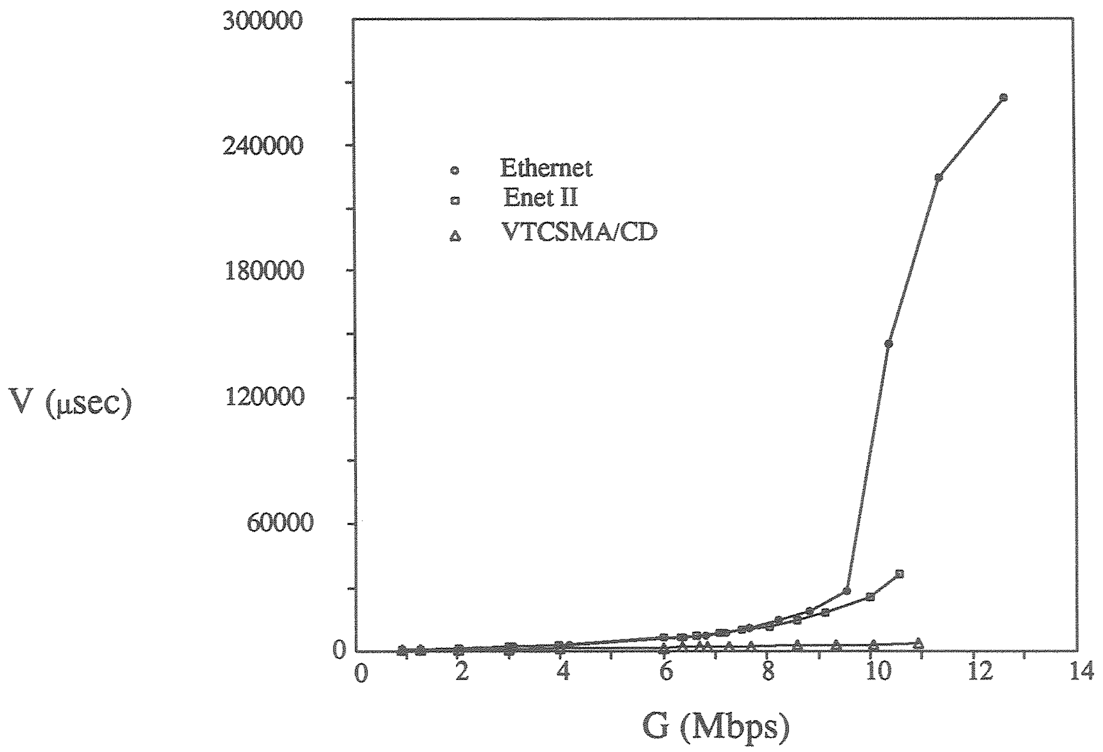


**Figure 5.29:** Packet delay variance (V) versus offered load (G) for packet mixture 1 with exponentially distributed interpacket intervals.

Figure 5.29 shows the delay variance versus the offered load for packet mixture 1 with exponentially distributed interpacket times. The exponentially distributed interpacket delays have caused a more gradual increase in variance for Ethernet and Enet II, though Ethernet still experiences a very sharp increase in delay variance at approximately 9.5 megabits per second offered load. As for the other load patterns, VTCSMA/CD suffers only a slight increase in delay variance as the load increases.

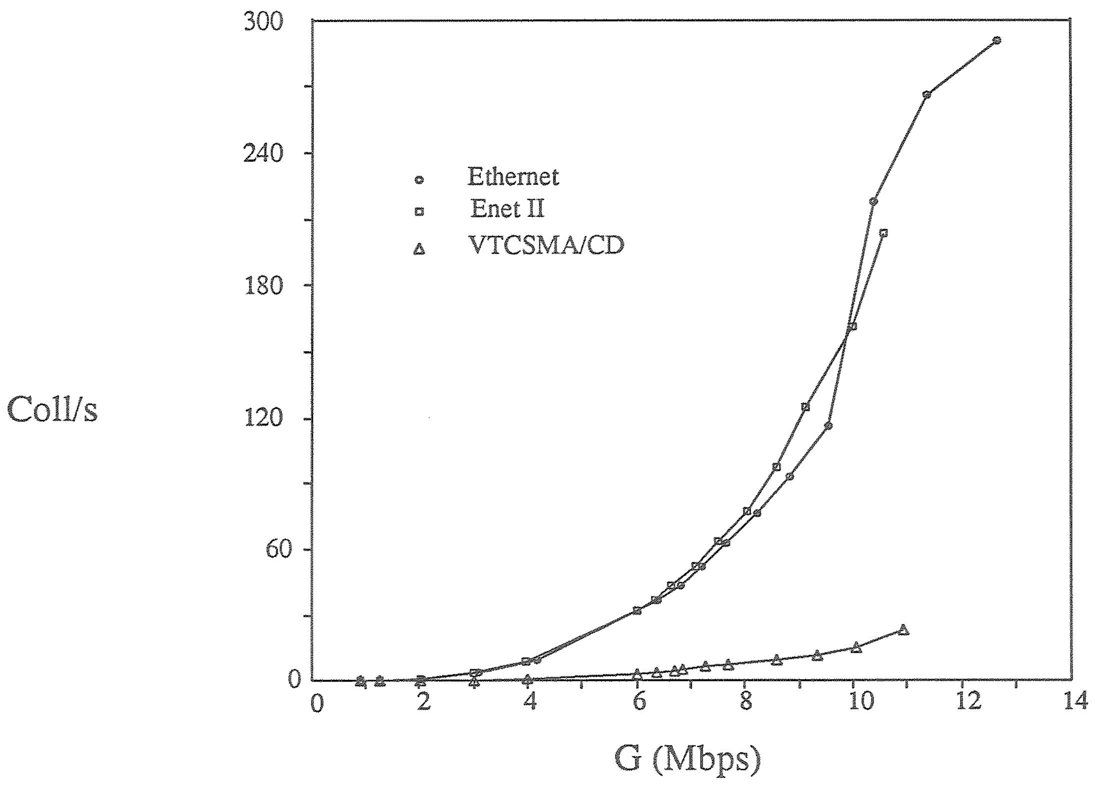


**Figure 5.30:** Collision rate (Coll/sec) versus offered load (G) for packet mixture 1 with exponentially distributed interpacket intervals.

Figure 5.30 shows the collision rate for packet mixture 1 with exponentially distributed interpacket times. The relation of the curves is the same as for the previous loads. This figure also indicates that the effect noted for fixed length packets, i.e. that contention begins for Ethernet and Enet II at lower loads with exponential arrivals, is still evident when a mixture of packet lengths are present.

## 5.4. Packet Mixture 2

Packet mixture 2 consists of 45% 1500 byte packets and 55% 275 byte packets, for an average length of 826 bytes. This mixture most closely resembles the actual traffic discussed in Chapter 4. This mixture is expected to demonstrate an intensification of the effects shown in Section 5.3.

### 5.4.1. Fixed Interpacket Intervals

Figure 5.31 shows the throughput versus offered load for packet mixture 2 with fixed interpacket intervals. Results were not available for the Enet II protocol for this load pattern. Under this offered load, Ethernet throughput peaks at 7 megabits per second at an offered load level of 9.5 megabits per second. Throughput is relatively stable as the offered load increases into the overload region. VTCSMA/CD displays a pattern similar to that observed in other experiments, with throughput increasing more slowly. VTCSMA/CD peaks at 6.8 megabits per second at the maximum generated offered load of 14.5 megabits per second. For Ethernet, contention begins to affect the throughput between 4 and 5 megabits per second offered load. As expected, contention begins at a lower offered load for this mixture than for mixture 3.

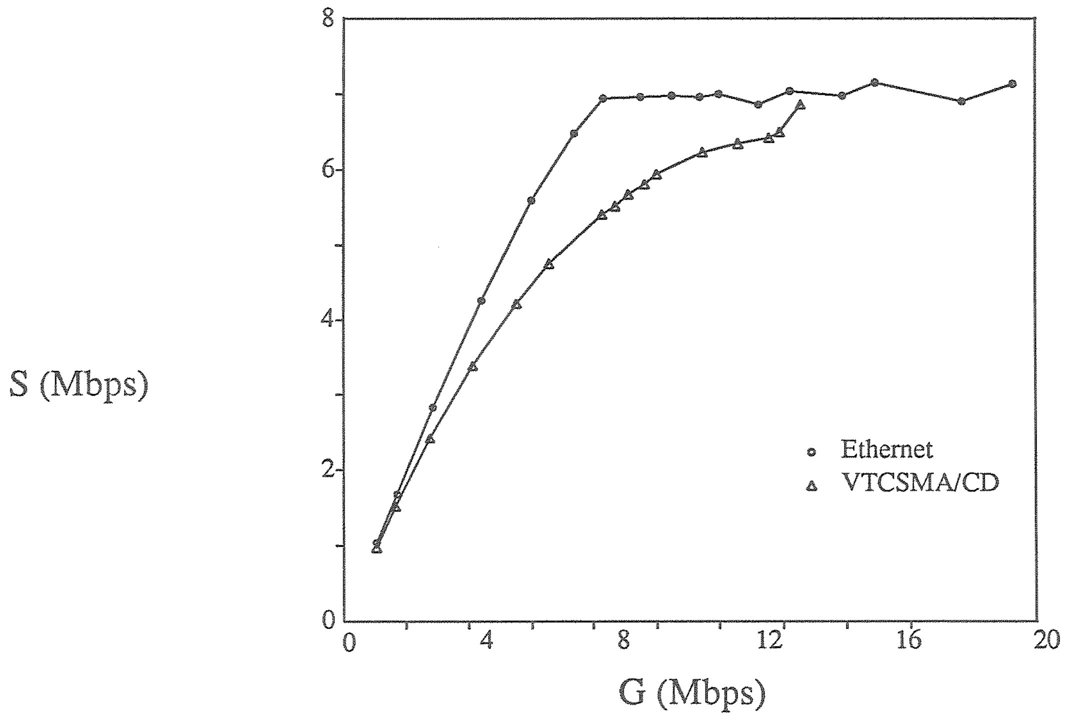


**Figure 5.31:** Throughput (S) versus offered load (G) for packet mixture 2 with fixed interpacket intervals.

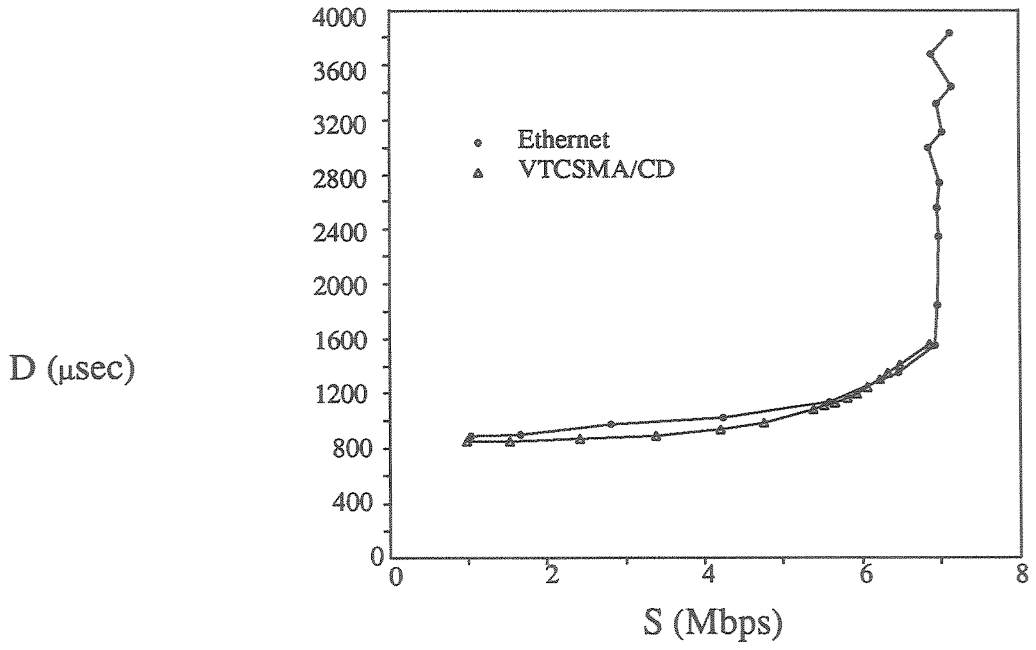


**Figure 5.32:** Packet queueing delay (D) versus throughput (S) for packet mixture 2 with fixed interpacket intervals.

Figure 5.32 shows the packet queueing delay versus the throughput for the two protocols. As for previous load patterns, VTCSMA/CD has a lower average delay over most of the range of throughput. However, unlike the other load patterns, when contention causes the delay to begin increasing for Ethernet, the delay begins increasing for VTCSMA/CD as well, and follows the Ethernet curve very closely over the range of generated throughputs. This indicates that increasing the percentage of small packets in the mix also increases contention in VTCSMA/CD.

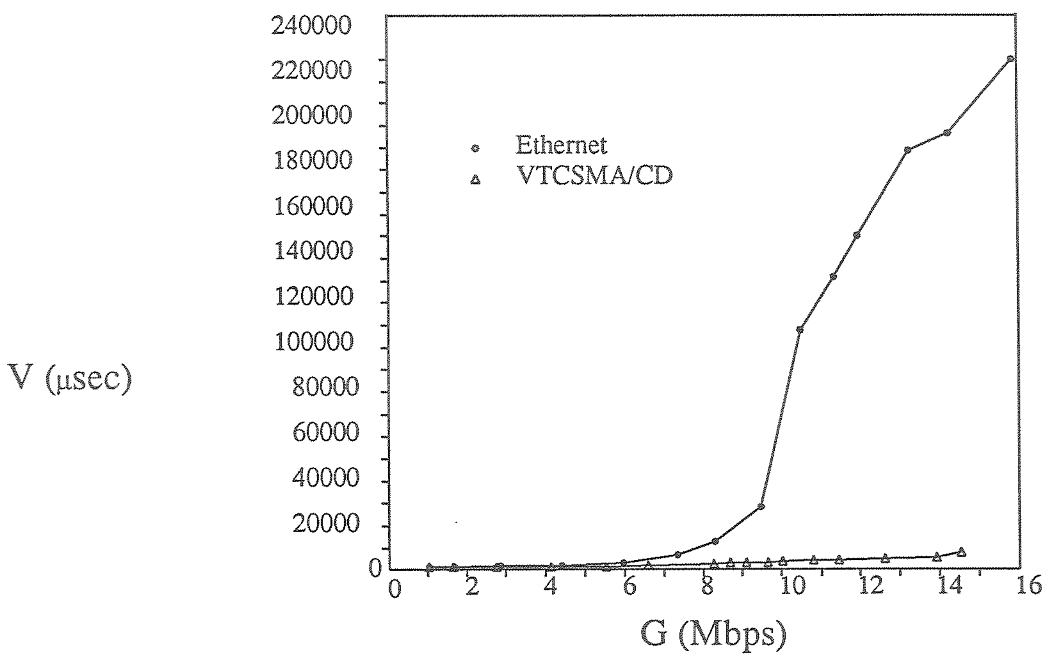


**Figure 5.33:** Delay variance (V) versus offered load (G) for packet mixture 2 with fixed interpacket intervals.

Figure 5.33 shows the delay variance (V) versus the offered load (G) for this load pattern. As before, VTCSMA/CD shows only a slight increase in delay variance relative to Ethernet. However, the increase in variance under heavy load for VTCSMA/CD is more pronounced for mix 2 than for mix 1. At the maximum generated load (14.5 megabits per second), the variance is nearly twice that observed at the same load level for mix 1. This is not surprising, since the number of small packets in the mixture has more than doubled, thus increasing the chances for collision even for VTCSMA/CD. For Ethernet, the increase in variance begins at a lower offered load and increases more steadily until the channel becomes saturated, at which point the same rapid increase in variance is observed.

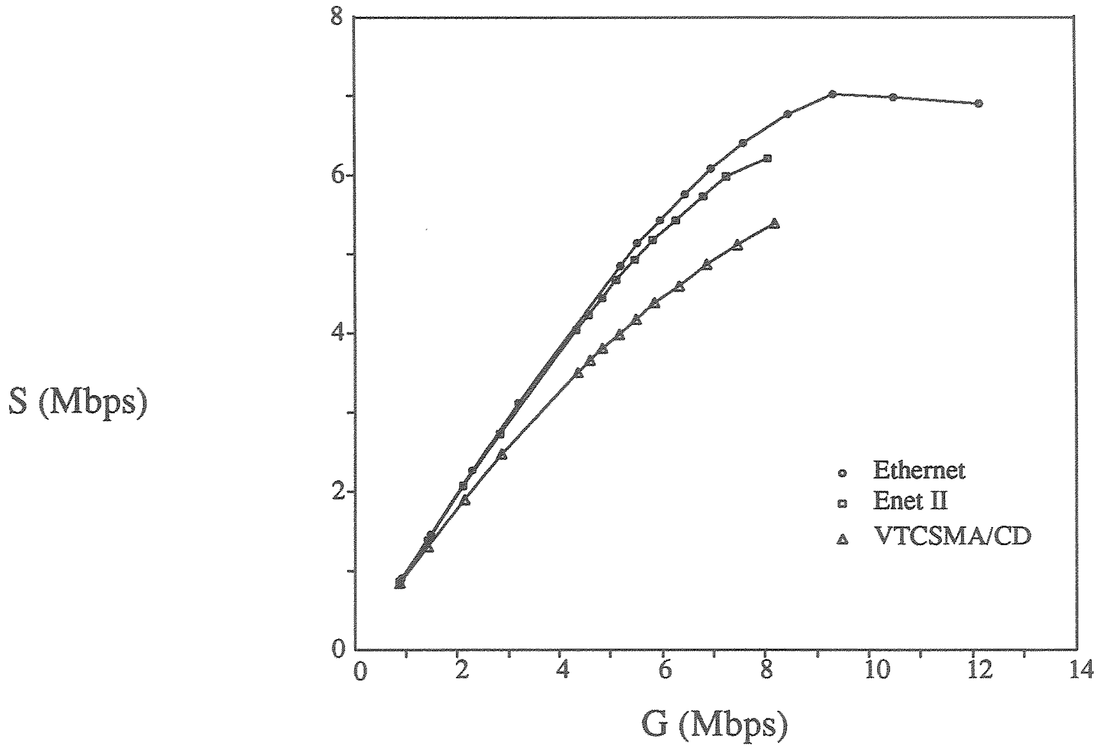


**Figure 5.34:** Throughput (S) versus offered load (G) for packet mixture 2 with exponentially distributed interpacket intervals.

### 5.4.2. Exponential Interpacket Intervals

Figure 5.34 shows the throughput (S) versus offered load (G) for packet mix 2 with exponentially distributed interpacket intervals. The relationship between the curves is as before, with the Enet II throughput deviating from the offered load at a lower level than is the case for Ethernet. Ethernet achieves a peak of 7.0 megabits per second throughput at offered load of 9.34 megabits per second. The Ethernet throughput declines slightly for higher offered loads, but it was not possible to generate high enough loads to discover whether this is due to the abbreviated backoff of the implementation or is merely the sort of fluctuation observed in figure 5.31. The Ethernet throughput matches the offered load closely up to load levels of about 3.5 megabits per second, while the throughput for Enet II begins deviating from the offered load at levels around 3 megabits per second. Though it was not possible to generate overloads for Enet II or VTCSMA/CD, it appears that Enet II would not have matched the peak throughput of Ethernet for this load pattern.

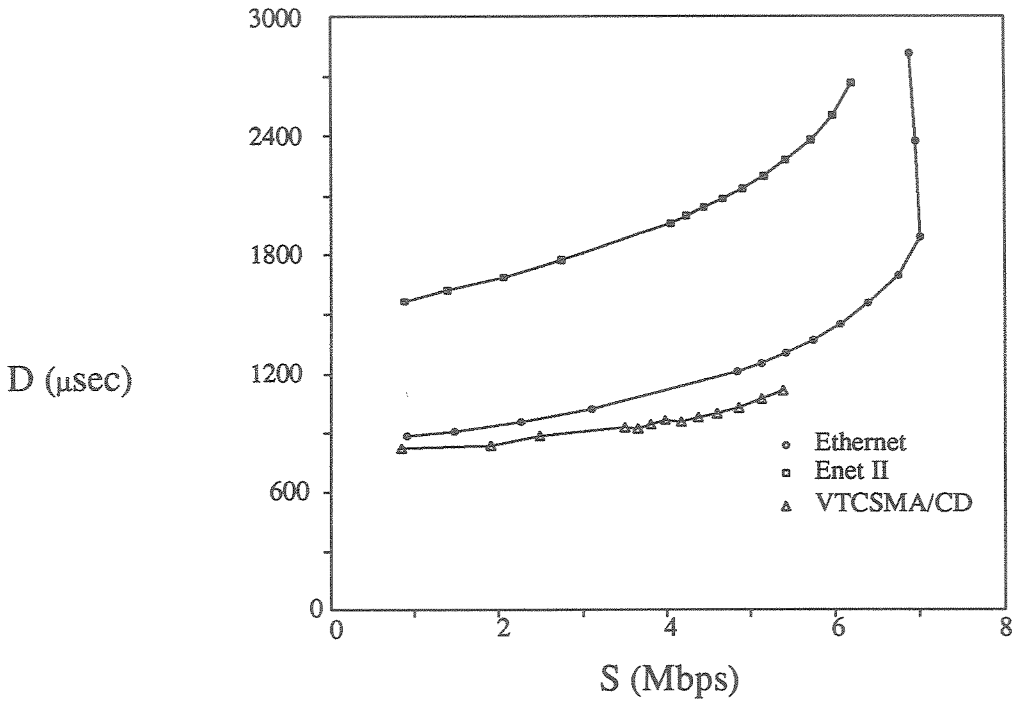


**Figure 5.35:** Packet queueing delay (D) versus throughput (S) for packet mixture 2 with exponentially distributed interpacket intervals.

Figure 5.35 shows the packet queueing delay versus throughput for packet mixture 2 with exponentially distributed interpacket intervals. The relationship between the curves has not changed; however, since overload traffic could not be generated for Enet II or VTCSMA/CD it is impossible to say what the behavior would have been in the region where the curves cross. The average distance between the Enet II curve and the Ethernet curve is slightly more than 700 microseconds, corresponding once more to the gating delay for Enet II. Again, VTCSMA/CD has a lower average delay over the range of generated throughputs.

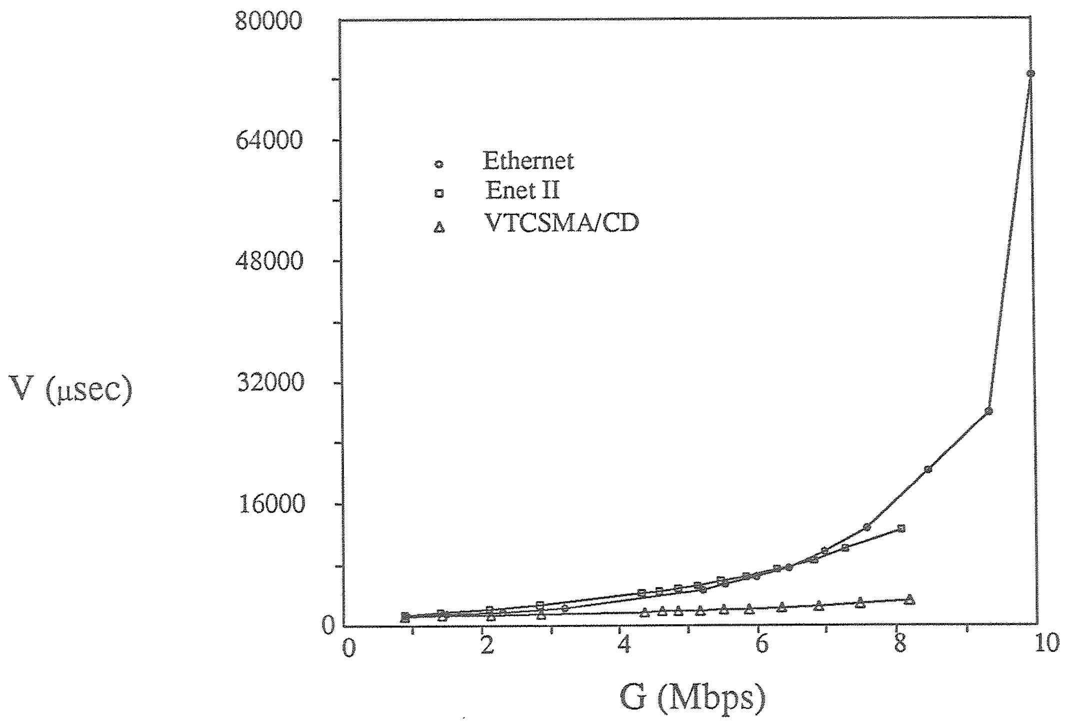


**Figure 5.36:** Delay variance (V) versus offered load (G) for packet mixture 2 with exponentially distributed interpacket intervals.

Figure 5.36 shows the delay variance versus the offered load for packet mixture 2 with exponentially distributed interpacket delays. The variance for VTCSMA/CD is lowest over the range of generated loads. Enet II suffers the highest variance until approximately 6.5 megabits per second offered load. For loads greater than 6.5 megabits per second, the delay variance for Ethernet increases very rapidly.

## 5.5. Packet Mixture 3

Packet mixture 3 consists of 35% 1500 byte packets and 65% 275 byte packets. Average packet size was 704 bytes.

### 5.5.1. Fixed Interpacket Intervals

Figure 5.37 shows the throughput versus offered load for packet mixture 3 with fixed interpacket times. The relationship among the curves is similar to that observed for the other two mixtures, except that for this mixture it appears that Enet II would not achieve the same maximum throughput level as Ethernet. Ethernet peaks at 6.9 megabits per second at a load of 8.6 megabits per second. The decrease in throughput after saturation is reached is evident, but less pronounced than for other load patterns. The peak for Enet II is 6.5 megabits per second at an offered load of 12.1 megabits per second. VTCSMA/CD peaks at 6.2 megabits per second at an offered load of 13 megabits per second. Continuing the trend observed for the other load patterns, throughput diverges from offered load for Ethernet between 3.5 and 4.5 megabits per second offered load and for Enet II between 2.5 and 3.5 megabits per second offered load.

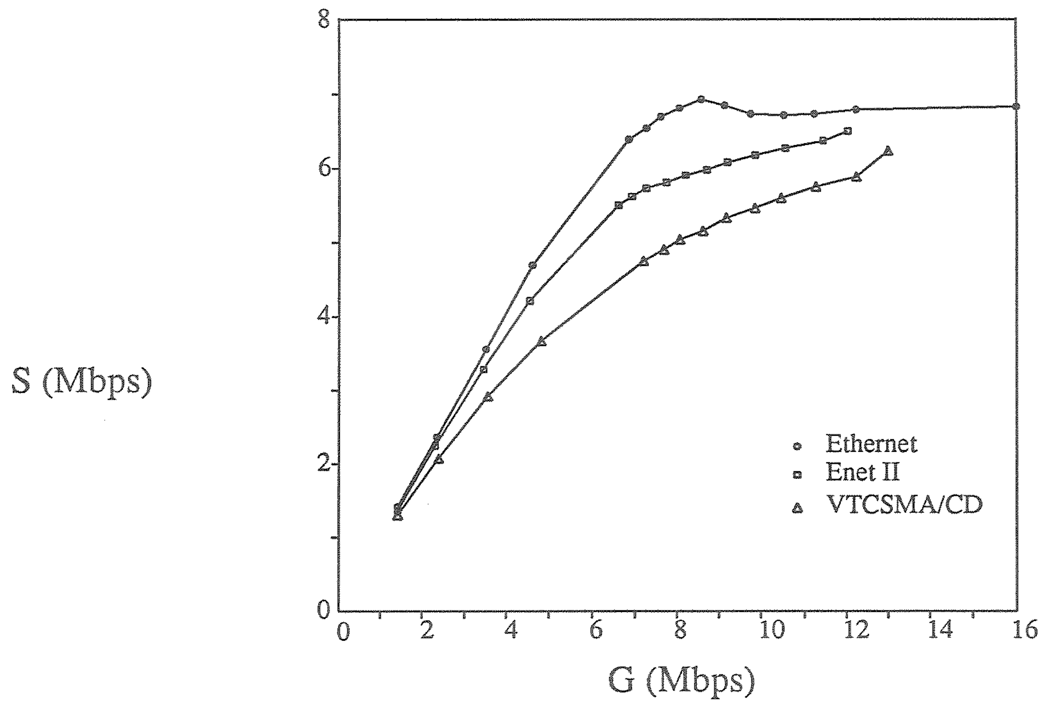


**Figure 5.37:** Throughput (S) versus offered load (G) for packet mixture 3 with fixed interpacket intervals.

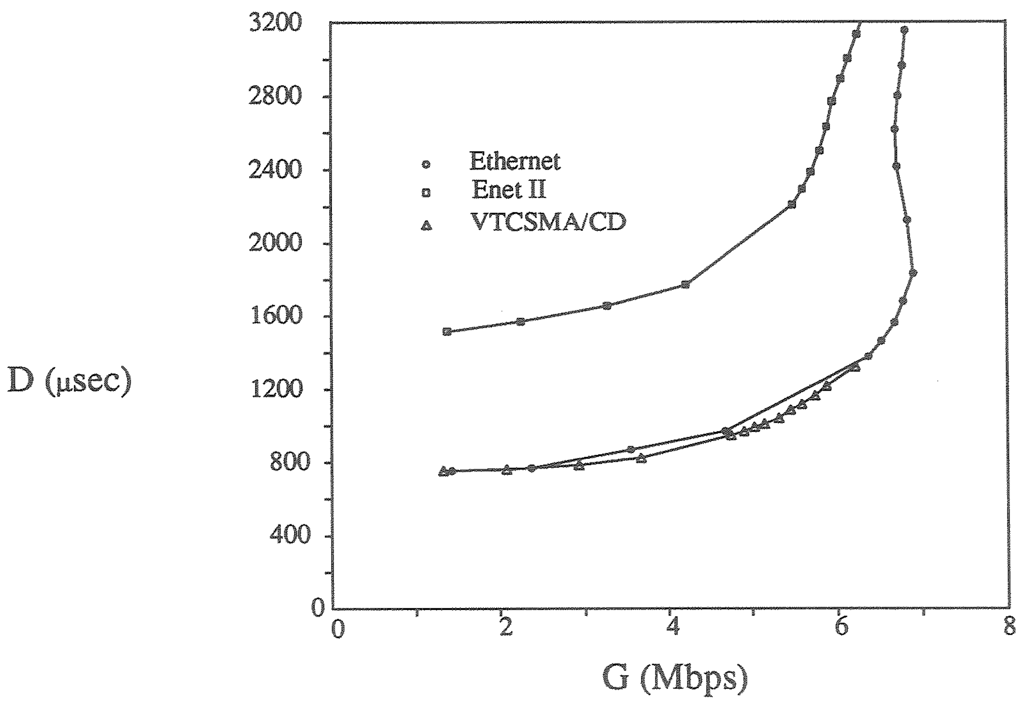


**Figure 5.38:** Packet queueing delay (D) versus throughput (S) for packet mixture 3 with fixed interpacket intervals.

Figure 5.38 shows packet delay versus throughput for packet mixture 2 with fixed interpacket intervals. The distance between the Ethernet and Enet II curves is approximately 750 microseconds, commensurate with the Enet II gating delay. The most notable feature of this graph is the VTCSMA/CD delay curve, which matches the Ethernet curve very closely over the entire range of generated throughputs. This indicates that the delay for VTCSMA/CD is also sensitive to the percentage of small packets present in the traffic mix, though perhaps less so than the other two protocols.

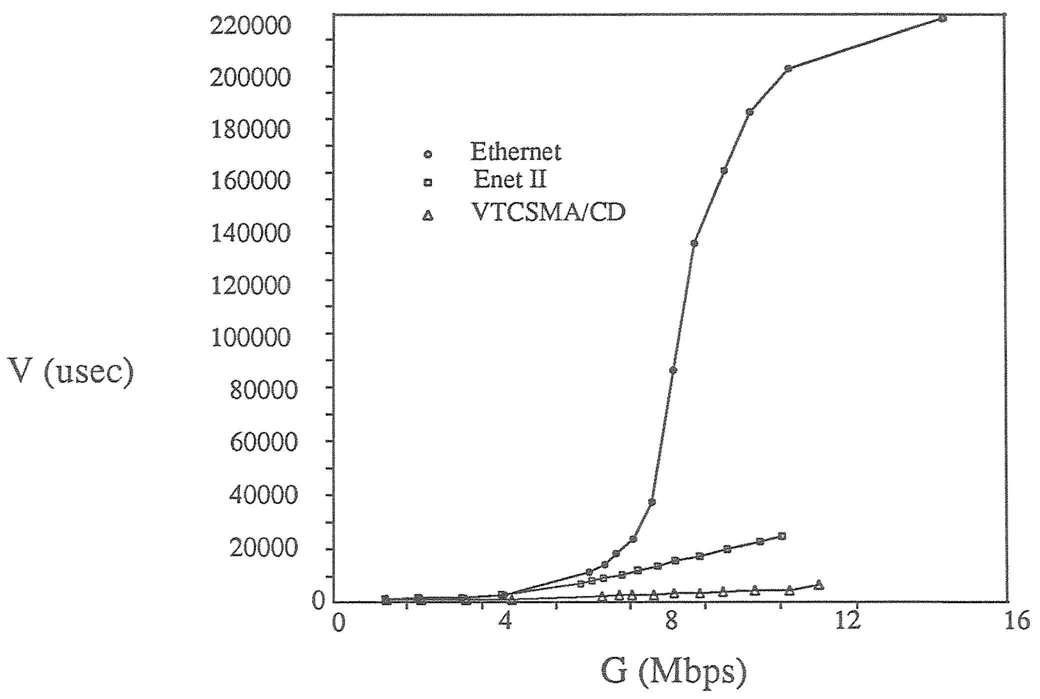


**Figure 5.39:** Delay variance (V) versus offered load (G) for packet mixture 3 with fixed interpacket intervals.

Figure 5.39 shows the delay variance versus the offered load for packet mixture 3 with fixed interpacket delays. The curves are very similar to those shown for the other mixtures, with the rapid increase in variance for Ethernet occurring at a lower offered load.

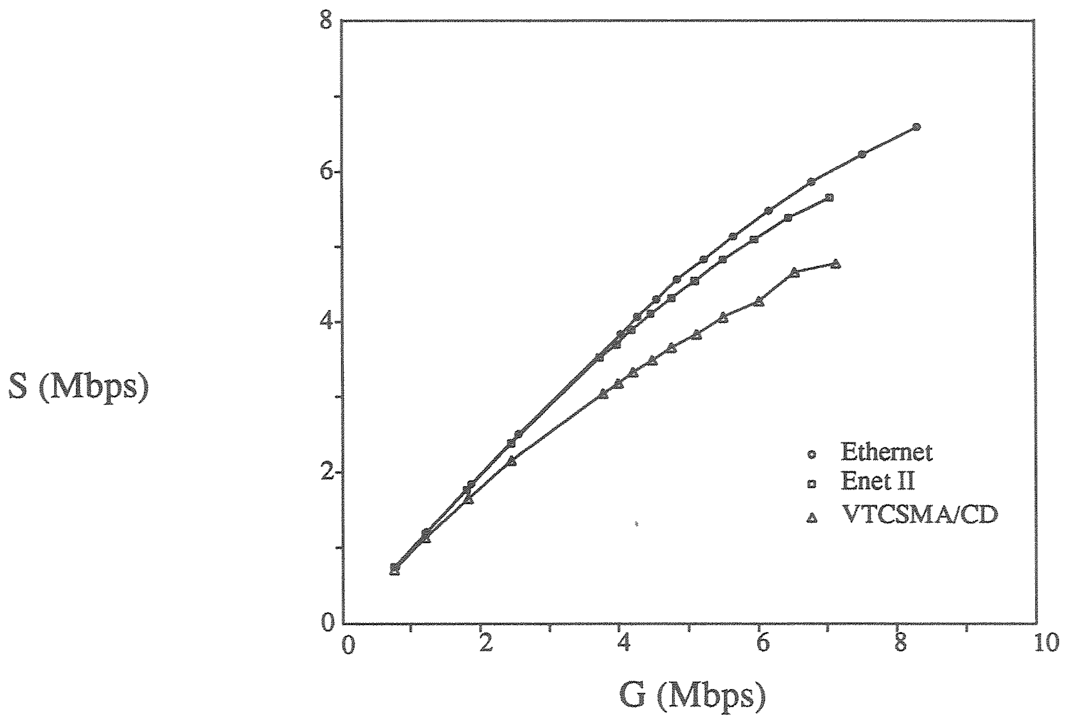


**Figure 5.40:** Throughput (S) versus offered load (G) for packet mixture 3 with exponentially distributed interpacket intervals.

## 5.5.2. Exponential Interpacket Intervals

Figure 5.40 shows the throughput versus offered load for packet mixture 3 with exponentially distributed interpacket times. For this load pattern none of the protocols allowed the generation of overloads, so the peak throughput for each protocol occurs at the highest generated load. For Ethernet, the peak is 6.6 megabits per second at 8.3 megabits per second offered load . For Enet II, the peak is 5.65 megabits per second at 7.1 megabits per second offered load. For VTCSMA/CD the peak is 4.8 megabits per second at 7.1 megabits per second offered load. The relationship among the curves is the same as seen in

Figure 5.34 and 5.27 for the range of loads generated. Significant differences between throughput and offered load begin at loads of about 2.5 megabits per second for both Ethernet and Enet II.

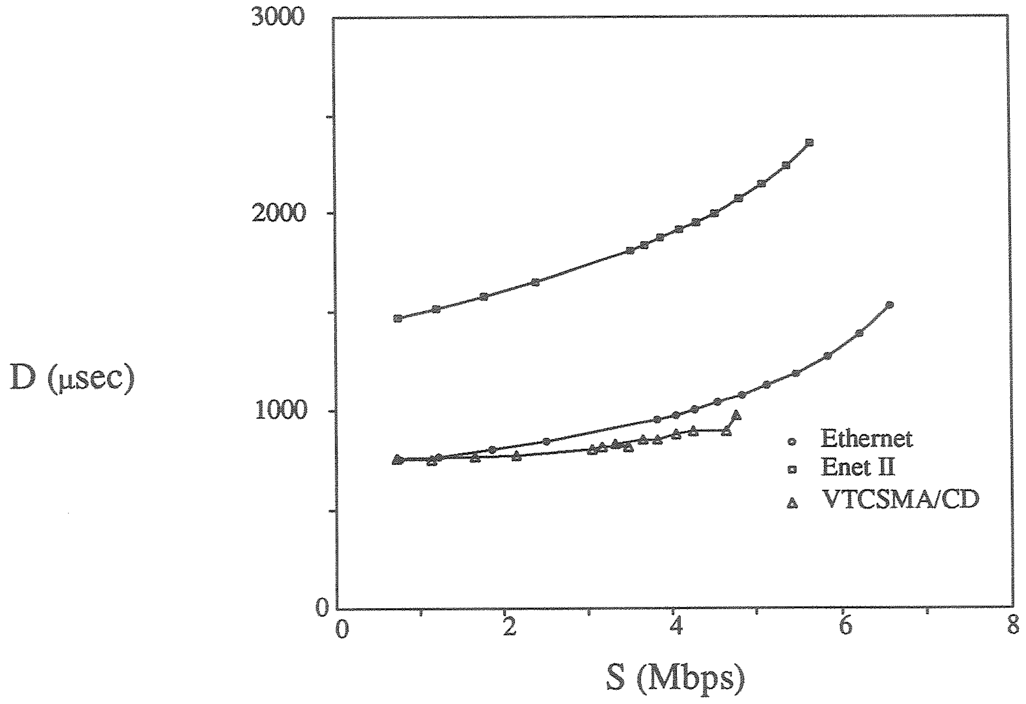


**Figure 5.41:** Packet delay (D) vs. throughput (S) for packet mixture 3 with exponentially distributed interpacket intervals.

The delay versus throughput behavior for this load pattern is shown in figure 5.41. Again, the VTCSMA/CD protocol displays the best average delay, though the behavior is very similar to that observed for Ethernet under this traffic load. The difference between the Ethernet and Enet II curves is about 750 microseconds under low loads, but the delay increases more rapidly for Enet II under heavier loads. This differs from previous load patterns; as the percentage of small packets increases, Enet II sends fewer data bits per resolved collision.

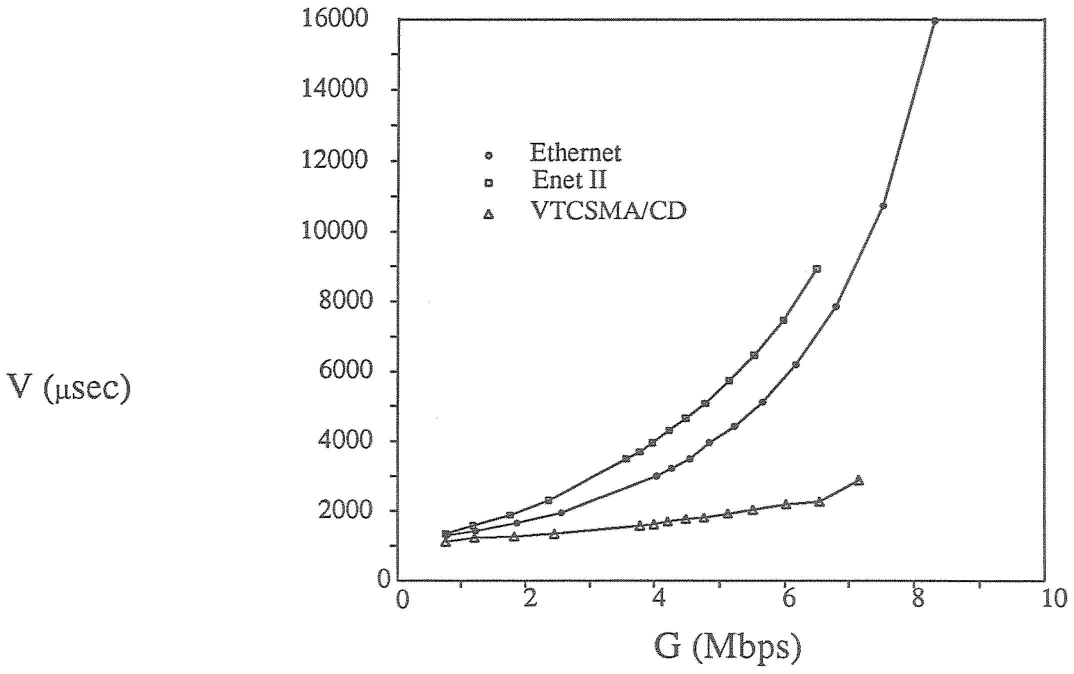


**Figure 5.42:** Delay variance (V) versus offered load (G) for packet mixture 3 with exponentially distributed interpacket intervals.

Variance versus offered load for this load pattern is shown in Figure 5.42. Though the rate of increase of the two protocols is similar, the variance of delay is somewhat higher for Enet II than for Ethernet under this traffic load. For other load patterns, Enet II showed its greatest improvement over Ethernet after channel saturation. Since it was not possible to saturate the channel for this load pattern, it is not possible to determine whether this trend would be repeated. The variance for VTCSMA/CD increases only slightly over the range of generated loads.

## 5.6. Comparison to Other Results

As mentioned in Chapter 2, there are a few other measurement studies of the Ethernet protocol in the literature [SH80][Gons85][BMK88]. Though some of these studies consider a different network, or different topologies, some conclusions concerning the relationship of the studies can be made. As the current work is the only example of implementation and measurement of the Enet II and VTCSMA/CD protocols to date, no such comparisons are possible. However, the agreement of the measurements with analytic models of these two protocols will be discussed.

### 5.6.1. Shoch and Hupp

The original Ethernet measurements carried out by Shoch and Hupp [SH80] were performed on the experimental 3 Megabit/second version of Ethernet. The maximum packet size available on that network was 512 bytes. As noted in Chapter 2, the aims of the study were principally to determine the overall behavior of the network under normal and heavy loads. The artificially generated loads were produced by host populations ranging from 5 hosts to 64 hosts. Obviously, there are significant differences between the conditions of the Xerox study and the present study. Yet there are some parallels. The Xerox study reports peak utilization of 97% with a generating population of five continuously queued hosts sending 512 byte packets. In our experiments, sustained throughput with continuously queued hosts sending 1500 and 1024 byte packets also remained above 97%. Attempts to load the network with smaller sized packets were not successful. The experiments performed at Xerox by varying the offered load also observed sustained utilization rates of approximately 96% with 512 byte packets. This agrees well with the current results with 1500 and 1024 byte packets. Shoch and Hupp reported that little degradation was observed as the offered load was increased to 150% of capacity. While our results indicate a problem in stability at loads greater than 100% of capacity, this is clearly related to the truncated backoff algorithm employed in the device driver.

### 5.6.2. Gonsalves

In 1985, results of experiments similar to those of Shoch and Hupp were presented by Gonsalves [Gons85]. The principal differences in the studies were that Gonsalves included queueing delay as one of his metrics and investigated both the experimental 3-Mbps Ethernet and the production 10-Mbps Ethernet. Offered load, delay, and interpacket time were calculated in the same manner for both these experiments and the LANT experiments. Comparison between the two is somewhat strained, however, by the fact that the host population for Gonsalves' experiments was from 30 to 38 hosts.

The pertinent set of Gonsalves' experiments were run with a fixed packet size of 1500 bytes and uniformly distributed delays. The peak measured throughput for this load was between 80% and 85%. This compares to a peak of 98% and stable utilization of 75% for fixed intervals and peak of 78% utilization and stable level of about 72% for exponentially distributed intervals observed in the analogous LANT experiments. Due to the heavy penalty in overhead incurred to generate the exponential intervals in the LANT experiments, the fixed interval experiments are probably the better comparison. There was also one set of runs performed with the LANT for 500 byte packets, which achieved a peak utilization of 84%, compared to 72% for Gonsalves' measurements for 512 byte packets. (This set of LANT experiments was not discussed because the figures were not available for the other two protocols investigated.) These figures appear in Table 5.3. It is obvious from the delay curves presented in [Gons85] that with the larger host population, contention begins at much lower offered load, and this accounts for the lower peak figure. The delay curves for the Gonsalves experiments show a more gradual increase in delay, but also display the very sharp increase in delay after offered load exceeds channel capacity. Gonsalves' experiments were conducted on a longer wire than the LANT experiments, which would also reduce the peak throughput.

### 5.6.3. Gonsalves and Tobagi

In [GT88], Gonsalves and Tobagi present a simulation study of Ethernet which differs from previous simulation results presented in [AL79] and elsewhere in that it examines networks where stations are not uniformly distributed along the wire. These results indicate

Table 5.3: Comparison of Ethernet Measurement and Modeling Results on Maximum Throughput

| | | Measured | | | Modeled | |
|---|---|---|---|---|---|---|
| Length (bytes) | Slot size (usec) | LANT | Gonsalves | BMK | Tobagi & Hunt | Lam |
| 1500 | 0.43 | 98.19 | | | 98.81 | 99.77 |
| | 4.00 | | | 99 | 97.74 | 97.90 |
| | 11.75 | | 86 | | 95.50 | 94.07 |
| | 15.00 | | 85 | | 94.60 | 92.55 |
| | 23.20 | | | | 92.37 | 88.93 |
| 1024 | 0.43 | 96.79 | | | 98.26 | 99.66 |
| | 4.00 | | | 99 | 96.72 | 96.95 |
| 512 | 0.43 | 84.11 | | | 96.58 | 99.33 |
| | 4.00 | | | 98 | 93.64 | 94.09 |
| | 11.75 | | 72 | | 87.86 | 84.41 |
| | 15.00 | | 72 | | 85.64 | 80.92 |
| | 23.20 | | | | 80.54 | 73.28 |
| 275 | 0.43 | 71.30 | | | 93.82 | 98.76 |

that in networks arranged as the LANT was (see Figure 3.1), with a cluster of stations at one end and a single station at the other end, the single station achieves lower throughput and experiences higher delay than the stations in the cluster. In the LANT experiments, though the effect was not as marked as in [GT88] (where very small packets, a longer network and a larger host population were used), Eyebeam, the isolated station, consistently achieved slightly lower throughput and experienced higher delay than the clustered stations over the entire range of generated load for every load pattern investigated.

### 5.6.4. Boggs, Mogul and Kent

Results of experiments run with various packet sizes and host populations are presented in in [BMK88]. For the most part, only the peak figures for each combination of packet size and host count are reported. The relevant numbers are included in Table 5.3. The numbers shown are for experiments run with 5 hosts, matching the LANT host population. It is interesting to note that the peak throughput observations from this study consistently exceed the Tobagi and Hunt predictions. One contributing factor is that Boggs, et. al. include the preamble, checksum, and interpacket gap in their throughput calculations. The packet generation software also queued packets while waiting for transmissions to complete, allowing true "continuously queued" loads to be generated, while the LANT generation software incurred some overhead after transmission was complete before generating the next packet. This study also varies from the others in that the stations are attached to the network using multiport repeaters rather than with simple taps. It is not clear whether this helps account for the difference between data gathered with similar parameters in the different studies. Experiments with bimodal packet distributions achieved much higher throughput than similar packet mixtures in the LANT study. The observation concerning continuously queued loads is applicable in this case, as well. One interesting observation put forth in this study is that fairness as characterized by the standard deviation of the bit rate of individual hosts increases as the number of hosts increases. This casts some doubt on the appropriateness of evaluating the fairness of Ethernet in a testbed the size of the LANT.

### 5.6.5. Tobagi and Hunt

In [TH80], Tobagi and Hunt extend the CSMA analysis originally presented in [KT75] to networks with collision detection. The main intent of this study was to characterize the improvement in throughput achieved by using collision detection. The analysis assumes a slotted time axis, with the slot size equal to the end-to-end propagation delay and an infinite host population with an aggregate arrival process which is Poisson in nature. Finite population results are also presented for Non-persistent CSMA/CD with fixed and variable packet sizes. The infinite population results for 1-persistent CSMA/CD are considered. Table 5.3 shows a comparison of Tobagi and Hunt's results for maximum throughput with measurements from both the LANT experiments and Gonsalves's experiments. The LANT network spanned a distance of approximately 100 meters, which translates to a propagation delay of 430 nanoseconds given nominal Ethernet specification compliant hardware and wiring. Gonsalves presents results for a 750 meter network including one repeater (11.75 microseconds) and a 1500 meter network including 2 repeaters (15 microseconds). Also included are Tobagi and Hunt's figures for a fully configured Ethernet with propagation delay of 23.2 microseconds. As Gonsalves notes, the analytic model tends to overestimate the maximum throughput, particularly as the packet size decreases, though the agreement with LANT measurements for 1500 and 1024 byte packets is fairly close. This indicates that the Tobagi-Hunt model either does not place sufficient emphasis on the effect of contention and the resulting retransmission on throughput, or simply does not model the retransmission policy used by Ethernet very closely. Note that the peak throughput reported for the LANT measurements was generated at the highest offered load before the truncated backoff algorithm began to affect the performance of the protocol.

Coyle and Liu [CL83] investigated CSMA/CD performance in networks with a finite number of hosts, but limited their investigation to the non-persistent version of the protocol.

### 5.6.6. Lam

In [Lam80], Lam presented closed form solutions for the size of the "distributed queue" (i.e. the number of ready users in the system) and the average message delay for CSMA/CD networks. As in [TH80], the channel is assumed to be time slotted with the slot size equal to the round-trip propagation delay for the network. The variant of CSMA/CD modeled is 1-persistent, which corresponds to the Ethernet protocol. The model assumes an adaptive retransmission algorithm for packets which suffer collisions. The numerical results presented by Lam and the calculations presented here assume that the retransmission algorithm results in a probability of success is a slot during a contention episode to be $1/e$. As in other studies, Lam presents curves for various values of the parameter $\alpha$, which is the ratio of one-way propagation time to packet transmission time. Lam notes that CSMA/CD performance improves as $\alpha$ decreases in size.

Table 5.3 also contains the maximum throughput figures from Lam's model. Relative to the measurements, the model consistently overestimates the maximum throughput except for the measurements of Boggs, et. al. (The relation of these measurements to the other studies is discussed in Section 5.6.4.) For very small values of $\alpha$, the overestimation is slightly worse for Lam's model than for the Tobagi and Hunt model, but as $\alpha$ increases, Lam's figure for maximum throughput is closer to the measured values. This is readily seen by

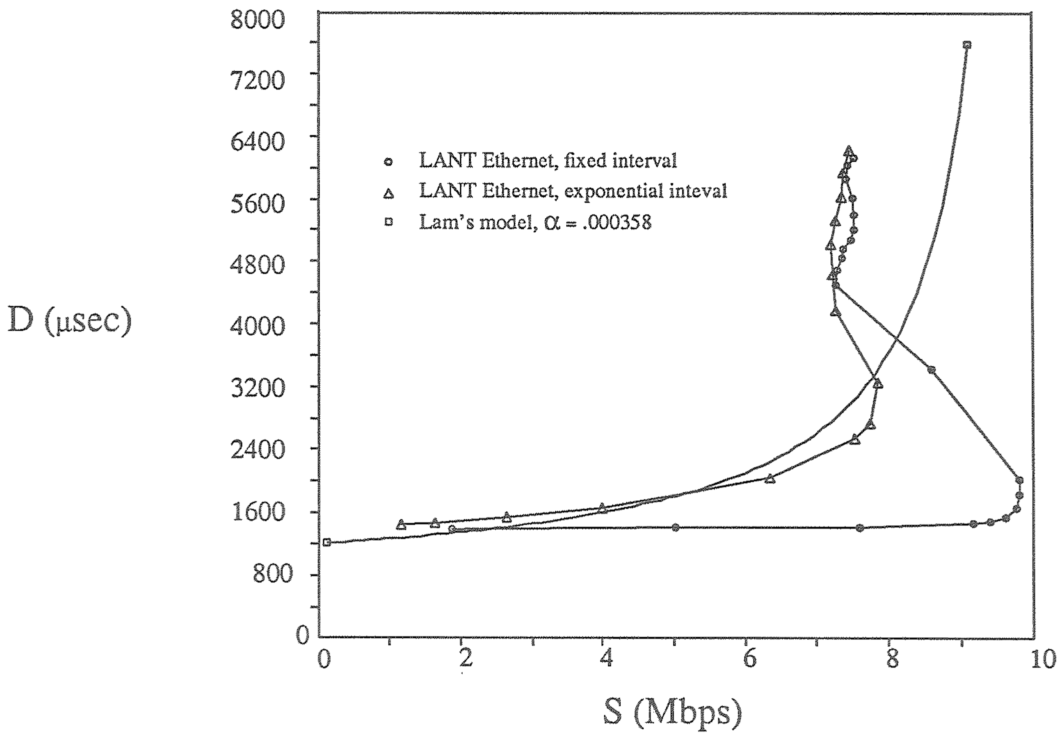


**Figure 5.43:** Packet queueing delay (D) versus throughput (S) for 1500 byte packets. Analytic curve calculated with $\alpha = 0.000358$, corresponding to LANT cable configuration (100m segment) and 1500 byte packets (1.2 millisecond transmission time).

comparing the figures for the two models for 1500 byte packets as the slot size increases.

The mean delay figures for Lam's model with input parameters matching the LANT setup are also shown in Figures 5.43 and 5.44. Figure 5.43 shows the delay versus the throughput for 1500 byte packets. The LANT Ethernet measurement curves for both fixed and exponentially distributed interpacket times are shown. The analytic curve was calculated with propagation time of 430 nanoseconds, making the value of $\alpha$ 0.000358. As expected, the agreement is not particularly good with the fixed interpacket time curve, but for low loads, the analytic curve and the exponentially distributed interpacket curve show fairly good agreement. In this case, the infinite population assumption led to slightly higher delays around the knee of the curve. The agreement lessens as the truncated backoff algorithm and the small host population combine to reduce the throughput of the LANT implementation of Ethernet. Figure 5.44 plots the same quantities for 1024 byte packets, with $\alpha = 0.000525$.

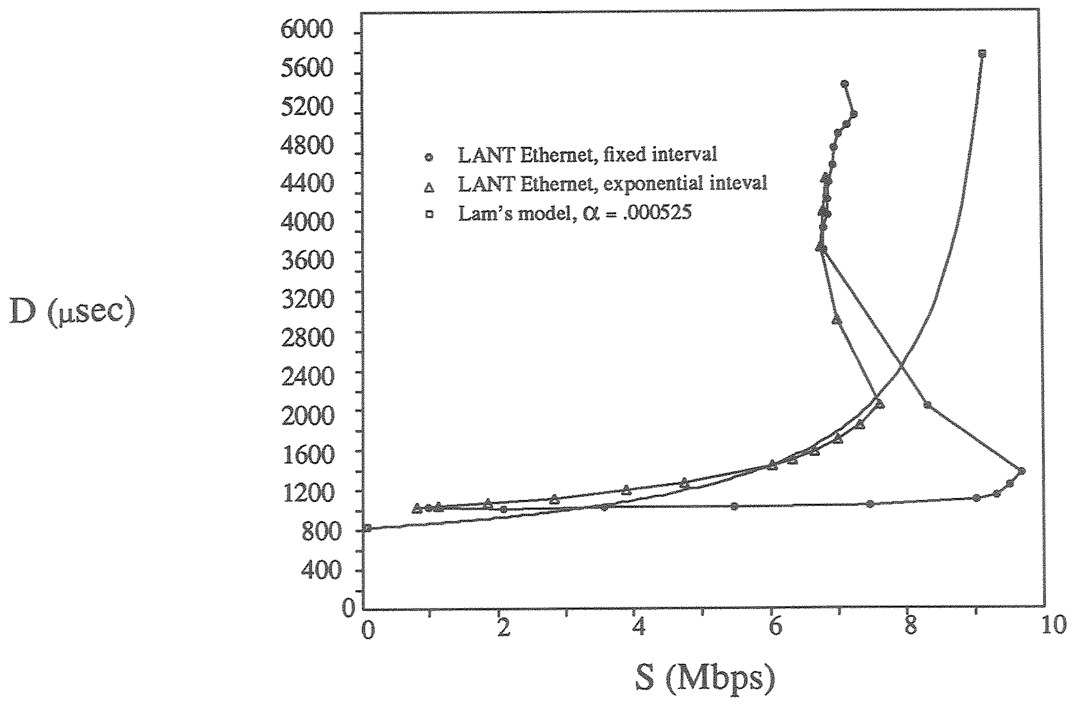


**Figure 5.44:** Packet queueing delay (D) versus throughput (S) for 1024 byte packets. Analytic curve calculated with $\alpha = 0.000525$, corresponding to LANT cable configuration (100m segment) and 1024 byte packets (819.2 microsecond transmission time).

The increase in $\alpha$ leads to closer agreement between the analytic curve and the measurement curve with exponentially distributed interpacket times. As in Figure 5.43, this agreement ends at the throughput drop for the LANT Ethernet implementation. This agrees with the observation made for the maximum throughput calculations that the model is more accurate for larger values of $\alpha$.

### 5.6.7. Enet II Results

Simulation results in [Moll85] indicated that Enet II should experience higher delay than Ethernet for low loads due to the gating delay for new packets and should have lower average delay once contention begins to dominate the transmission process. The simulator that generated these results assumed an infinite population of users and distributed arrival locations uniformly along the wire. The LANT measurements confirm the general thrust of these results, though the LANT clock resolution and the infinite population model used in the simulator prevent direct comparison of the average delay figures. In addition, the LANT measurements confirm the prediction that the variance of delay for Enet II is significantly less than that of Ethernet under heavy loads. An analytic model of Enet II was presented in [LW87], but the results given there are not directly comparable to the LANT measurements.

### 5.6.8. Molle's VTCSMA/CD Analysis

Molle presents analyses of synchronous and asynchronous versions of VTCSMA/CD in [MK85]. Figure 5.45 shows measurements for the LANT experiment run with 1500 byte packets and fixed interarrival times and the curve for the asynchronous model with similar parameters. The parameters used in the model reflect a 100 meter cable with stations sending 1500 byte packets. The model assumes packet transmission time to be unity, leading to the value 0.0003583 for $a$, the propagation delay as a percentage of packet transmission time. The virtual clock rate is the rate implemented in the LANT version of VTCSMA/CD. The parameter $c$ represents the collision clear time, which is given the standard value of the Ethernet jam signal. The figure shows that the model predicts higher throughput over most of the range of loads, though the shape of the two curves is very similar. While exact

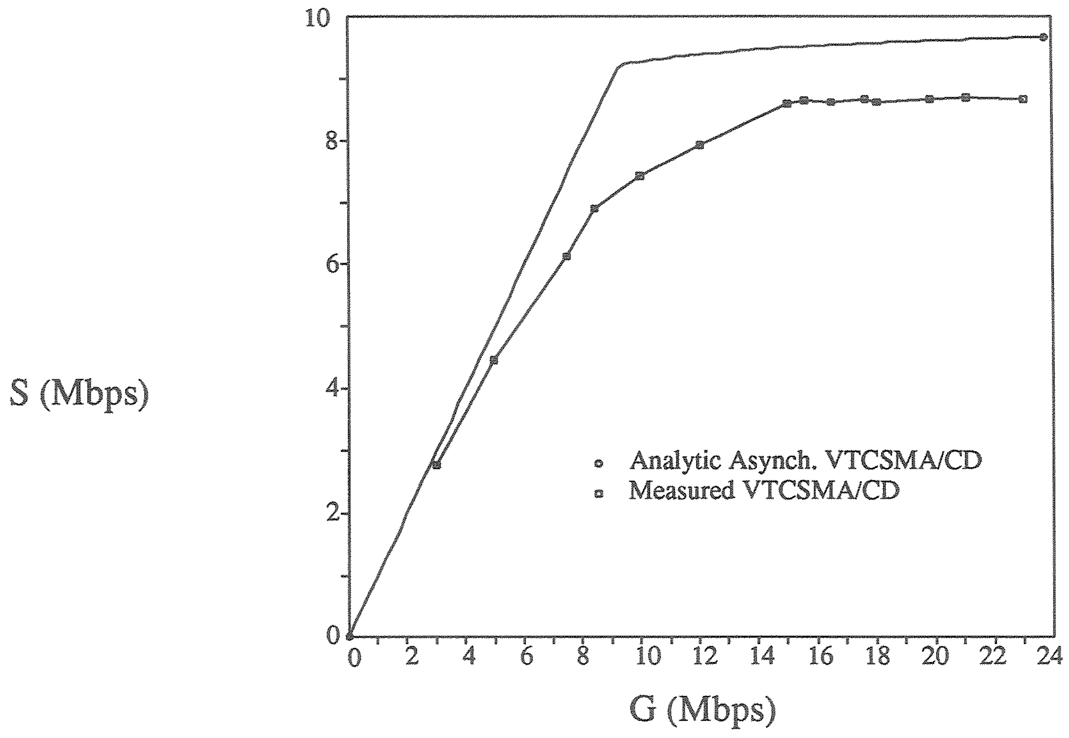


**Figure 5.45:** 1500 byte fixed interval measurements for VTCSMA/CD compared with Molle's asynchronous model. Model parameters: $a = 0.0003583$, $c = 0.004$, and $\eta = 13$.

agreement was not expected, the magnitude of the difference between the measured and predicted behaviors of the protocol indicate that, as previously suggested, the VTCSMA/CD implementation did suffer significantly from the overhead incurred by software manipulation of the virtual clock. An infinite population model of VTCSMA/CD is also given in [ML85], where the stability of the protocol was examined.

# Chapter 6

# On Using Slotted Collision Resolution Protocols on Unslotted Media

Many of the local network protocols developed recently are of a slotted nature. They rely on some external clock or other synchronizing signal to organize the transmissions from each station on the network. Most of these protocols operate using a slot size equal to the transmission time of a packet, and in performance analyses, the throughput of these protocols is expressed in terms of the average fraction of a packet transmitted per slot. The best currently known protocol achieves over .48 packets per slot, while the upper bound is .5 packets per slot. [PTW85] In the light of the confirmed throughput efficiency of Ethernet, which achieves 98% utilization of network bandwidth under some circumstances, this figure seems low. While Ethernet suffers in other respects when compared to the kind of Collision Resolution Protocols (CRP) considered here, it effectively makes use of the asynchronous nature of the CSMA/CD environment by reacting in a timely way to network events. Ideally, features of these two types of protocols could be combined to construct protocols with better delay characteristics than the Ethernet style random backoff protocols and better throughput than the slotted CRPs. Enet II is an example of a protocol which attempts to combine the characteristics of both types of protocols.

The obvious starting point for an investigation of such hybrid protocols is an attempt to adapt the existing slotted CRPs for use in the asynchronous CSMA/CD environment. One area where throughput improvements are possible can be immediately identified: the size of slots. In slotted CRPs, all slots are equal to the transmission time of a packet. If the packet size is large in comparison to the end-to-end propagation time for the network, then a great deal of time is wasted discovering conflicts and idle slots. By using the collision detection mechanism, collisions can be detected and aborted in a fraction of the normal transmission time. Likewise, by exploiting known characteristics of the network such as the maximum end-to-end propagation time, idle slots can be detected in less time than it would take to transmit an entire packet. The general strategy is to alter the protocol to react directly to events observed on the broadcast bus rather than reacting to some external synchronizing signal. Transmissions and collisions are two such events. Handling successful transmissions is straightforward. It will be shown that collisions require some care in their handling, but present no insurmountable problems. However, the most difficult problem lies in determining the end of an idle step in the absence of perfectly synchronized clocks. In order to achieve this, the technique presented in the Enet II algorithm of causing an intentional collision whenever an idle step should have ended is used. Clock synchronization is thus not an issue, unless some clock has a drift rate greater that the amount of a propagation delay during an idle step. In the protocol investigated here, an idle slot lasts one round trip delay.

The price for exploiting the asynchronous nature of the broadcast bus is that the new protocols are more difficult to validate due to the absence of synchronization, so special attention must be paid to the correctness of the adapted protocols. This issue is investigated by specifying an asynchronous version of the Gallager/Tsybakov First Come First Served (FCFS) protocol [Gall78] and proving a bounded delay property for the adapted protocol. This proof is presented in Section 6.4. An adaptation of the protocol which takes full advantage of the collision detection mechanism does not satisfy the bounded delay property. The protocols will be specified in the model of Jain and Lam [JL87][JL88][JL89]. An investigation of the performance impact of using the conservative implementation versus the aggressive version with a deadlock detection component, adapting an analysis of

Towsley to the two versions of the protocol, is presented in Section 6.5.1.

## 6.1. A Model for Reasoning about Contention Protocols

As presented, the basic model from [JL87][JL88][JL89] for a bidirectional bus is used for proving properties of a collision free protocol and thus did not model the collision detection mechanism. The model has been augmented here to handle collision detection. Modified axioms for the bus are presented, reflecting this addition to the model. The primary change required to model collision detection is a distinguished state for the segment of the bus to which a transmitting station is attached.

Jain's model uses a process to model the behavior of each station and a separate process to model the behavior of the bus or channel. Stations are specified by a set of local variables, a set of shared variables, and a program. The channel specification consists of a description of the state of the channel and a set of rules governing how the channel state changes with time.

### 6.1.1. The Station Process

The station process consists of a set of local variables (including time variables used to implement timing constraints), a set of shared variables, and a program written in a subset of Pascal to be described below. Variables may be shared with either a user process or the channel process, but not with another station program. The language is implemented with several temporal constructs which are described below. The explanation of the programming language is drawn largely from [JL89].

The following notation is used in the discussion of the variables used by the station process. *Exclusive-write* variables are shared variables which are written by a single process. *Mutual-write* variables are shared variables which may be written by more than one process.

| | |
|---|---|
| $V$ | set of exclusive write variables |
| $W$ | set of mutual-write variables |

| | |
|---|---|
| $X$ | set of shared variables which are exclusive-write variables of other processes. |
| $v_1, v_2, \cdots, v_m$ | subset of V |
| $w_1, w_2, \cdots, w_n$ | subset of W |
| $e_1, e_2, \cdots, e_{m+n}$ | expressions |
| $e(\tau)$ | value of expression $e$ at time $\tau$ |
| $P$ | predicate over variables in $V$ |
| $Q$ | predicate over variables in $W \cup X$ |

The proofs will also make use of control assertions, indicating where control lies in the station program at any given time. Following the notation of [OL82], the statements of a program are labeled (angle brackets are used here), and the following three assertions are used to express the control state of the program. $S$ refers to both a statement and its label, $p$ is a process, and $t$ is the time.

| | |
|---|---|
| at$(p, S, t)$: | true iff the control for process $p$'s program is just before statement $S$ at time $t$. |
| in$(p, S, t)$: | true iff the control for process $p$'s program is at the beginning of $S$ or inside $S$ at time $t$. |
| after$(p, S, t)$: | true iff the control for process $p$'s program is immediately after statement $S$ at time $t$. |

The syntax of the programming language constructs and the axioms and inference rules for each statement are shown below. $\langle S \rangle$ indicates the statement label for the statement, $p$ is a process, and $t$ is the time.

1. $\langle S \rangle\ S_1; S_2$

   after$(p, S_1, t) \Rightarrow$ at$(p, S_2, t)$

2. $\langle S \rangle$ if Q then $S_1$ else $S_2$

   at$(p, S, t) \wedge \mathrm{Q}(p, t) \Rightarrow$ at$(p, S_1, t)$

   at$(p, S, t) \wedge \neg \mathrm{Q}(p, t) \Rightarrow$ at$(p, S_2, t)$

   after$(p, S_1, t) \vee$ after$(p, S_2, t) \Rightarrow$ after$(p, S, t)$

3. $\langle S \rangle$ **while** Q **do** $S'$

$$\textbf{at}(p, S, t) \wedge Q(p, t) \Rightarrow \textbf{at}(p, S', t)$$

$$\textbf{at}(p, S, t) \wedge \neg Q(p, t) \Rightarrow \textbf{after}(p, S, t)$$

$$\textbf{after}(p, S', t) \wedge Q(p, t) \Rightarrow \textbf{at}(p, S', t)$$

$$\textbf{after}(p, S', t) \wedge \neg Q(p, t) \Rightarrow \textbf{after}(p, S, t)$$

4. $\langle S \rangle$ **set** $v_1, \cdots, v_m, w_1, \cdots, w_n := e_1, \cdots, e_{m+n}$

In the following, $P^{v_1, \cdots, v_m}_{e_1(\tau), \cdots, e_m(\tau)}$ refers to the predicate P with all free occurrences of $v_i$ replaced with $e_i(\tau)$.

$$\{ (\tau = \tau_{begin}) \wedge P^{v_1, \cdots, v_m}_{e_1(\tau), \cdots, e_m(\tau)} \}$$

$$\textbf{set}\, v_1, \ldots, v_m, w_1, \cdots, w_n := e_1, \cdots, e_{m+n}$$

$$\{ (\tau = \tau_{end} = \tau_{begin} + 1) \wedge P(\tau) \}$$

The **set** command assigns $e_1$ to $v_1$, $e_2$ to $v_2$ ... $e_{m+n}$ to $w_n$ in one atomic operation. The **set** command takes one clock tick.

5. **wait-seq**

**Wait-seq** causes the process to halt until a sequence of conditions have been satisfied. Let $Q_1, Q_2, \ldots Q_n$ be boolean conditions, and $T_1, T_2, \ldots T_n$ be time durations. In the following explanation,

$$\text{pattern} \equiv Q_1 \textbf{ for } T_1; Q_2 \textbf{ for } T_2; \ldots Q_n \textbf{ for } T_n;$$

$$R = T_1 + T_2 + \cdots + T_n$$

The execution of the **wait-seq** statement causes the process to halt until a match for the pattern is observed, i.e. condition $C_1$ is observed to be true for $T_1$, $C_2$ is observed to be true for $T_2$, and so forth. Formally,

$$\text{match}(p, \text{pattern}, t) \equiv [\forall t_n\!: t - T_n < t_n \leq t\!: Q(p, t_n)]$$

$$\wedge\ [\forall t_{n-1}\!: t - (T_n + T_{n-1}) < t_{n-1} \leq t - T_n\!: Q_{n-1}(p, t_{n-1})]$$

$$\vdots$$

$$\wedge\ [\forall t_1:\ t-(T_n+\cdots+T_1)<t_1\leq t-(T_n+\cdots+T_2):\ Q_1(p,t_1)]$$

The following rule for **wait-seq** states that the during the execution of the statement, the exclusive write variables of process $p$ do not change. Termination occurs at time $\tau_{end}$ if the match for the pattern started at some time after $\tau_{begin}$ and completed at $\tau_{end}$.

$\{\ \tau = \tau_{begin}\ \wedge\ P(\tau)\ \}$

**wait–seq** (*pattern*)

$\{\ \tau = \tau_{end} \geq \tau_{begin} + R\ \ \wedge\ [\forall t: \tau_{begin} \leq t \leq \tau_{end}:\ P(t)]$

$\wedge\ match\,(pattern\,,\tau_{end})$

$\wedge\ [\forall t':\ \tau_{begin} + R \leq t' < \tau_{end}:\ \neg match\,(pattern\,,t')]\ \}$

Special Cases

5. a) **delay(T)** $\equiv$ **wait-seq**(true for T)

This statement causes the process to halt and remain idle for time T. The execution time for the statement is T.

$\{\ \tau = \tau_{begin}\ \wedge\ P(\tau)\ \}$

**delay**(T)

$\{\ (\tau = \tau_{end} = \tau_{begin} + T)\ \wedge\ [\forall t:\ \tau_{begin} \leq t \leq \tau_{end}:\ P(t)]\ \}$

5. b) **wait**(Q) $\equiv$ **wait-seq**(Q **for** 1)

Wait(Q) causes the process to halt until the condition Q is observed to be true. The time taken by the command depends on when C becomes true.

$\{\ \tau = \tau_{begin}\ \wedge\ P(\tau)\ \}$

**wait**(Q)

$\{\ (\tau = \tau_{end} > \tau_{begin})\ \wedge\ [\forall t:\ \tau_{begin} \leq t \leq \tau_{end}:\ P(t)]\ \wedge\ Q(\tau)\ \}$

6. **wait-par**

**Wait-par** allows the program to execute **wait-seq** statements concurrently. The statement terminates as soon as any one of the **wait-seq** statements completes its execution, or 'fires.' Completion of one **wait-seq** results in the termination of the other statements. The resulting state is the same as if only the statement that fired was executed. After a statement fires, a label is set to indicate the identity of the statement. Statements within the **wait-par** are listed in order of decreasing priority, so that should two statements fire simultaneously, the label indicates the statement with the highest priority. Nesting of **wait-par** statements is allowed. (A more general **wait-par** statement is defined in [JL89]. The special case presented here is sufficient for our purposes.)

The following definitions are used:

$$pattern_i \equiv Q_{i,1} \textbf{ for } T_{i,1};\ldots; Q_{i,n(i)} \textbf{ for } T_{i,n(i)}$$

where $n(i)$ is the number of terms in $pattern_i$, and

$$R_i = T_{i,1} + T_{i,2} + \cdots + T_{i,n(i)}.$$

The axiom for the **wait-par** statement is the following:

$$\{(\tau = \tau_{begin}) \wedge P(\tau)\}$$

```
wait-par
        wait−seq ($pattern_1$); label := $l_1$
        || wait−seq ($pattern_2$); label := $l_2$
          ⋮
        || wait−seq ($pattern_m$); label := $l_m$
end-wait-par
```

$$\begin{aligned}
&\{(\tau = \tau_{end}) \wedge [\forall t\colon \tau_{begin} \le t \le \tau_{end}\colon P(t)] \\
&\quad \wedge label = l_1 \Rightarrow (\tau = \tau_{end} \ge \tau_{begin} + R_1 \wedge match(pattern_1, \tau_{end})) \\
&\quad\quad \vdots \\
&\quad \wedge label = l_m \Rightarrow (\tau = \tau_{end} \ge \tau_{begin} + R_m \wedge match(pattern_m, \tau_{end})) \\
&\quad \wedge [\forall t_1\colon \tau_{begin} + R_1 \le t_1 < \tau_{end}\colon \neg match(pattern_1, t_1)] \\
&\quad\quad \vdots \\
&\quad \wedge [\forall t_m\colon \tau_{begin} + R_m \le t_m < \tau_{end}\colon \neg match(pattern_m, t_m)]\ \}
\end{aligned}$$

### 6.1.2. The Channel Process

The channel is considered to be composed of discrete segments, with the length of each segment being the minimum allowed distance between stations. Segments are numbered left to right from 0 to $D-1$, where $D$ is the one-way propagation delay for the network in segments. A station connected to segment $p$ is referred to as station $p$. The state of the channel consists of the state of each segment and is represented by the boolean array *busy*. The elements of *busy* that correspond to segments of the network to which stations are attached are shared with station programs. In order to model collision handling, a second state vector is added, *transmitting*, which indicates that the station attached to the segment is currently transmitting a message onto the bus. The appropriate elements of *transmitting* are also shared with the station programs, and **transmitting**($p$) is true if station $p$ has its *transmitting* flag set and false otherwise. Further, a station is no longer considered to be busy due to its own transmission. This technique allows a station to detect a collision as it would in an actual network, by comparing its output signal with the signal on the bus. A segment to which a transmitting station is connected is in state **transmitting**, but is not busy unless a signal from another transmitting station has reached its location. So a collision is detected by station $p$ when **transmitting**($p$) and **busy**($p$) are true at the same time.

Time in the system is modeled by a global clock combined with time variables. The **set** statement is assumed to take one time unit. All other programming constructs are assumed to take no time. Jain also defines the following history variables for use in the proof system.

| | |
|---|---|
| $\mathbf{busy}_{LR}(p, \tau)$ | true, if there is a signal at location $p$ at time $\tau$ propagating from left to right,<br>false, otherwise |
| $\mathbf{busy}_{RL}(p, \tau)$ | true, if there is a signal at location $p$ at time $\tau$ propagating from right to left,<br>false, otherwise |

The following history variable is added.

| | |
|---|---|
| **transmitting** $(p, \tau)$ | true, if the station connected to the bus at segment p is transmitting at time $\tau$,<br>false, otherwise |

Since a segment of the network is assumed to be equal in length to the minimum spacing for stations, stations are allowed to be connected to adjacent segments.

The addition of the **transmitting** state for bus segments requires the adjustment of the channel axioms presented by Jain. Let $N$ be the set of stations, $S$ be the set of segments, and $T$ be the set of segments to which stations are attached. In the sequel, absence of quantification indicates universal quantification.

A 6.1: $[\forall p\colon p \in \{S - T\}\colon \neg\, \mathbf{transmitting}(p, \tau)]$

A 6.2: $[\forall p\colon p \in S \wedge p \neq 0\colon$

$$\mathbf{busy_{LR}}(p, \tau) = \mathbf{busy_{LR}}(p - 1, \tau - 1)$$

$$\vee\ (p - 1 \in T \wedge \mathbf{transmitting}(p - 1, \tau - 1))]$$

A 6.3: $\neg\, \mathbf{busy_{LR}}(0, \tau)$

A 6.4: $[\forall p\colon p \in S \wedge p \neq D - 1\colon$

$$\mathbf{busy_{RL}}(p, \tau) = \mathbf{busy_{RL}}(p + 1, \tau - 1)$$

$$\vee\ (p + 1 \in T \wedge \mathbf{transmitting}(p + 1, \tau - 1))]$$

A 6.5: $\neg\, \mathbf{busy_{RL}}(D - 1, \tau)$

A 6.6: $[\forall p\colon p \in S\colon \mathbf{busy}(p, \tau) = \mathbf{busy_{LR}}(p, \tau) \vee \mathbf{busy_{RL}}(p, \tau)]$

A 6.7: $min\_size > 2D + jam + 1$

Axiom A6.1 states that only segments to which a station is attached may be in the **transmitting** state. Axioms A6.2 and A6.4 state the channel process transitions for propagation of signals from left to right and right to left, respectively. Axioms A6.3 and A6.5 state that the leftmost and rightmost segments may never be busy from left to right and right to left, respectively. (An end segment may be in the **transmitting** state if it is in the set $T$ or may be busy from the opposite direction.) Axiom A6.6 is shorthand to indicate that a segment is busy when the direction of the signal is not important. The segment to which a transmitting station is attached is not considered to be busy as a result of its own transmission. Axiom A6.7 states that the minimum packet size, and thus the duration of the shortest successful transmission is greater than a a round trip propagation plus the duration of the collision consensus enforcement jam plus one tick.

The predicate **collision**$(p)$ indicates that station $p$ is currently detecting a collision.

C 6.1: $\textbf{collision}(\mathrm{p}) \equiv \textbf{transmitting}(\mathrm{p}) \wedge \textbf{busy}(\mathrm{p})$

The following history variables are useful.

D 6.1: $\textbf{collision}(p, \tau) \equiv \textbf{transmitting}(p, \tau) \wedge \textbf{busy}(p, \tau)$

D 6.2: $\textbf{tx\_begin}(p, \tau) \equiv \neg\, \textbf{transmitting}(p, \tau - 1) \wedge \textbf{transmitting}(p, \tau)$

D 6.3: $\textbf{tx\_end}(p, \tau) \equiv \textbf{transmitting}(p, \tau - 1) \wedge \neg\, \textbf{transmitting}(p, \tau)$

D 6.4: $\textbf{coll\_begin}(p, \tau) \equiv \neg\, \textbf{collision}(p, \tau - 1) \wedge \textbf{collision}(p, \tau)$

The distance between two stations $i$ and $j$ is given by $\Delta_{ij}$.

D 6.5: $\Delta_{ij} \equiv |i - j|$

D6.1 states that station $p$ is involved in a collision when it is transmitting and finds that the segment to which it is attached is also busy from left to right or right to left. This indicates that the signal from some other station has propagated to the location of $p$ on the bus and overlapped with the signal generated by $p$. The remainder of the definitions simplify references to the detection of the beginning and end of transmissions, the beginning collisions and the propagation of signals between stations. Note that since $D$ is the length of the network, $\Delta_{ij} < D$.

### 6.1.3. Station Transmission Behavior

Stations participating in the protocol should adhere to the CSMA/CD paradigm of deferring to transmissions in progress and aborting transmissions when a collision is detected. This behavior could be incorporated into the protocol specification, but since many protocols display this behavior, it is preferable to think of it as "infrastructure," and hide the details of its implementation. In that spirit, the enhanced packet transmission procedure, **transmit** is defined. The procedure call notation and semantics of [Grie81] are used. Note that the variable *transmitting* is a global variable which is shared between the station process and the channel process. *Jam*, *packet_sent*, and *coll_detected* are constants for the station program.

```
proc transmit (value t; result label);
{ transmitting(τ) = false ∧ τ = τ_begin }
    begin
        ⟨T1⟩ if (busy) then
            ⟨T2⟩ wait(not busy);
        { transmitting(τ) = false ∧ not busy(τ) ∧ τ = τ_1)) }
        ⟨T3⟩ set transmitting := true;
        { tx_begin(τ) ∧ not busy(τ − 1) ∧ τ = τ_1 + 1
            ∧ X ≡ [∃ t_1: t_1 < τ: not busy (t_1)
                ∧ [∀t′: t_1 < t′ ≤ τ: transmitting (t′)]]}
        ⟨T4⟩ wait_par
            if (not collision) then wait(collision); label := coll_detected;
            || if (T > 1) then delay(T - 1); label:= packet_sent;
        end_wait_par;
        { X ∧ [∀t′: τ_1 < t′ < τ: not collision (t′)]
            ∧ ((label(τ) = packet_sent ∧ τ = τ_1 + T ∧ not collision(τ))
                ∨ (label(τ) = coll_detected
                ∧ [∃ d: 0 ≤ d < T: τ = τ_1 + d])) }
        ⟨T5⟩ if (label = coll_detected) then
            ⟨T6⟩ delay(jam);
        { X ∧ (label(τ) = packet_sent ∧ τ = τ_1 + T)
            ∨ (label(τ) = coll_detected
            ∧ [∃ d: 0 ≤ d ≤ T: τ = τ_1 + d + jam]) }
        ⟨T7⟩ set transmitting := false;
    end
{ tx_end(τ) ∧ (label(τ) = packet_sent ∧ τ = τ_1 + T + 1)
    ∨ (label(τ) = coll_detected ∧ [∃ d: 0 ≤ d ≤ T: τ = τ_1 + d + jam + 1]) }
```

This procedure implements deference, by waiting to transmit until the channel is not busy (T1), collision detection and the associated abort (T4), and the jam, or collision consensus enforcement policy (T5). It manipulates the shared variable *transmitting*, which alerts the channel process that a transmission is occurring. It returns the status of the transmission by setting the variable *label*. The formal semantics of the transmit procedure are as follows. $P'$ is the station's set of exclusive write variables except for *transmitting*.

$\{ \tau = \tau_{begin} \wedge P'(\tau) \wedge \neg\, \textbf{transmitting} \}$

$\textbf{transmit}(T, \mathit{label});$

$\{ \tau = \tau_{end} \wedge$

$((\mathrm{label}(\tau) = \mathrm{packet\_sent} \wedge \tau_{end} = \tau_{begin} + T' + T + 2) \vee$

$(\mathrm{label}(\tau) = \mathrm{coll\_detected} \wedge \tau_{end} = \tau_{begin} + T' + d + jam + 2)) \wedge$

$[\forall t, \tau_{begin} + T' \le t \le \tau_{end}\text{: transmitting}(t)] \wedge$

$[\forall t'\text{: } \tau_{begin} \le t' \le \tau_{end}\text{: } P'(t')] \}$

$T'$ is 0 if the channel is not busy at $\tau_{begin}$; otherwise, $T'$ is the time between $\tau_{begin}$ and the time at which the channel is no longer busy at the location of the station attempting to transmit. $d$ is the difference between the time at which transmission actually starts and the time at which the collision is detected by the station attempting to transmit. The additive constant 2 arises from the set primitives used to alter the *transmitting* flag. If all stations in the network use the stated transmit procedure, then $1 \le d \le 2D$, since no station will begin transmitting after the signal from the attempting station propagates to its location, and the attempting station will not begin transmission if there is a signal on the channel at its location. (See Theorem 6.3.) All stations are assumed to enforce some maximum allowed packet length, which will be referred to as *max_size*, so $1 \le T' \le max\_size$, and $min\_size \le T \le max\_size$. The timing characteristics of the transmit procedure follow immediately from these facts and the semantics of the other language constructs.

Given the specification of transmit, the following are invariants.

I1: $\textbf{tx\_begin}(p, t) \Rightarrow \neg\, \textbf{busy}(p, t-1)$

I2: $\mathbf{transmitting}(p, t) \Rightarrow [\exists t_{begin}: t_{begin} \le t: \mathbf{tx\_begin}(p, t_{begin})$

$\wedge [\forall t': t_{begin} \le t' \le t: \mathbf{transmitting}(p, t')]]$

I3: $\mathbf{tx\_end}(p, t) \wedge \mathbf{label}(p, t) = \text{packet\_sent}$

$\Rightarrow [\exists T: min\_size \le T \le max\_size:$

$[\forall t': t - T - 1 < t' < t: \mathbf{transmitting}(p, t') \wedge \neg \mathbf{collision}(p, t')]]$

I4: $\mathbf{tx\_end}(p, t) \wedge \mathbf{label}(p, t) = \text{coll\_detected}$

$\Rightarrow \mathbf{coll\_begin}(p, t - jam - 1)$

$\wedge [\exists t_{begin}: t_{begin} < t: \mathbf{tx\_begin}(p, t_{begin})$

$\wedge [\forall t': t_{begin} \le t' < t: \mathbf{transmitting}(p, t')]$

$\wedge [\forall t'': t_{begin} \le t'' < t - jam - 1: \neg \mathbf{collision}(p, t')]]$

I5: $\mathbf{tx\_begin}(p, t)$

$\Rightarrow [\exists T, \exists t_1: min\_size \le T \le max\_size \wedge t \le t_1 < t + T:$

$[\forall t': t \le t' \le t_1: \mathbf{transmitting}(p, t')]$

$\wedge ((t_1 = t + T - 1) \vee \mathbf{coll\_begin}(p, t_1))]$

I6: $\mathbf{coll\_begin}(p, t)$

$\Rightarrow \text{label}(p, t) = \text{coll\_detected} \wedge \mathbf{tx\_end}(p, t + jam + 1)$

$\wedge [\forall t': t \le t' \le t + jam + 1: \mathbf{transmitting}(p, t')]$

I7: $\mathbf{collision}(p, t)$

$\Rightarrow [\exists t_{collbeg}: t - jam \le t_{collbeg} \le t:$

$\mathbf{coll\_begin}(p, t_{collbeg}) \wedge \mathbf{tx\_end}(p, t_{collbeg} + jam + 1)]$

The first four invariants indicate what can be deduced about the previous behavior of the transmit procedure from the current state. I1 states that for a station to begin transmission at time $t$, it must have sensed the wire idle at time $t - 1$. I2 states that if a station is transmitting at time $t$, then the transmission began at some time $t_1$ no later than $t$ and has continued since that time. I3 states that if a transmission is successful, then the station was transmitting continuously over some interval and no collision occurred during that interval. I4 states that if a transmission results in a collision, then there was some continuous period of time during which the station was transmitting with no collision followed by a continuous period of time during which the station was transmitting and a collision was occurring. The last two invariants show what can be deduced about the subsequent behavior from the current state. I5 indicates that if a station began transmitting, it will continue to do so until it completes transmission or suffers a collision. I6 shows the time at which the value of the label changes and the time at which the transmission ends given that a collision occurs. I7 gives the range of times a collision could have begun given that it is still going on at $t$.

## 6.2. Basic Theorems Concerning Collisions

In this section, theorems are presented concerning the detection, duration, and propagation of collisions on a bidirectional broadcast bus. In the sequel, $p \in N$, $p' \in N$, $q \in N$, $r \in N$, and $p_i \in N$ for $i = 1, 2, \ldots, |N|$. Any unquantified variables are assumed to be universally quantified. When two quantifiers appear separated by a comma, ordering indicates nesting of the scope of the quantifiers. It is also assumed that the transmissions are result of executions of the **transmit** procedure defined above. Each step in the proofs is accompanied by a list of theorems, lemmas, invariants or previous steps which justify it. The list is enclosed in square brackets and set at the right margin on the same line as the conclusion to which it refers. Absence of such a list means that the step involves only algebraic manipulation or follows from the construction of the algorithm to which the step refers. Labels appear in parentheses alone on the line preceding the initial antecedent of the series of implications or equalities to which they refer. The following theorem from [JL87] concerning the basic behavior of the channel process is stated without proof.

*Theorem J1:* The following relations hold for the channel process.

a) $\textbf{busy}(p, \tau) \Rightarrow [\exists p' : p' \neq p : \textbf{transmitting}(p', \tau - \Delta_{pp'})]$

b) $\textbf{transmitting}(p, \tau) \Rightarrow [\forall p' : p' \neq p : \textbf{busy}(p', \tau + \Delta_{pp'})]$

*Theorem 6.1:*

$$\neg\,\mathbf{busy}(p, t_p - 1) \wedge \mathbf{busy}(p, t_p)$$
$$\Rightarrow [\exists q : q \neq p : \mathbf{tx\_begin}(q, t_p - \Delta_{pq})]$$

*Proof:*

(1)

$\neg\,\mathbf{busy}(p, t_p - 1)$ [LHS]

$\Rightarrow [\forall r : r \neq p : \neg\,\mathbf{transmitting}(r, t_p - 1 - \Delta_{pr})]$ [ThJ1]

(2)

$\mathbf{busy}(p, t_p)$ [LHS]

$\Rightarrow [\exists q : q \neq p : \mathbf{transmitting}(q, t_p - \Delta_{pq})]$ [ThJ1]

(3)

(3) $[\forall r, \exists q : r \neq p \wedge q \neq p :$ [(1),(2)]

$\neg\,\mathbf{transmitting}(r, t_p - 1 - \Delta_{pr}) \wedge \mathbf{transmitting}(q, t_p - \Delta_{pq})]$

$\Rightarrow [\exists q : q \neq p : \mathbf{tx\_begin}(q, t_p - \Delta_{pq})]$ [D6.2]

□

Theorem 6.2 states that if a collision is detected by station $p$ at time $\tau$, then $p$ must have begun to transmit at a time earlier than or equal to $\tau$ and there must be another station $q$ which began transmitting at time $\tau - \Delta_{pq}$ whose transmission has now propagated to $p$.

*Theorem 6.2:*

$$\mathbf{coll\_begin}(p, \tau) \Rightarrow [\exists q : q \neq p : \mathbf{tx\_begin}(q, \tau - \Delta_{pq})]$$

*Proof*:

$$\textbf{coll\_begin}(p,\tau) \equiv \neg\,\textbf{collision}(p,\tau-1) \wedge \textbf{collision}(p,\tau) \qquad [D6.4]$$

$$\equiv (\neg\,\textbf{transmitting}(p,\tau-1) \vee \neg\,\textbf{busy}(p,\tau-1)) \qquad [D6.1]$$

$$\wedge\ \textbf{transmitting}(p,\tau) \wedge \textbf{busy}(p,\tau)$$

$$\equiv (\neg\,\textbf{transmitting}(p,\tau-1) \wedge \textbf{transmitting}(p,\tau) \wedge \textbf{busy}(p,\tau))$$

$$\vee\ (\neg\,\textbf{busy}(p,\tau-1) \wedge \textbf{transmitting}(p,\tau) \wedge \textbf{busy}(p,\tau))$$

$$\equiv (\textbf{tx\_begin}(p,\tau) \wedge \textbf{busy}(p,\tau)) \qquad [D6.2]$$

$$\vee\ (\neg\,\textbf{busy}(p,\tau-1) \wedge \textbf{transmitting}(p,\tau) \wedge \textbf{busy}(p,\tau))$$

$$\Rightarrow \neg\,\textbf{busy}(p,\tau-1) \wedge \textbf{busy}(p,\tau) \qquad [I1]$$

$$\Rightarrow [\exists\, q:\ q \neq p:\ \textbf{tx\_begin}(q,\tau-\Delta_{pq})] \qquad [Th6.1]$$

□

Theorem 6.3 shows the range of times that a station could have begun transmission given that it was involved in a collision. The result is in terms of the location of a station $q$ which which station $p$ collides.

*Theorem 6.3*:

$$\textbf{coll\_begin}(p,\tau) \Rightarrow$$

$$[\exists\, q:\ q \neq p:\ [\exists\, t_{begin}:\ \tau-2\Delta_{pq} \le t_{begin} \le \tau:\ \textbf{tx\_begin}(p,t_{begin})]]$$

*Proof:*

(1)

$$\textbf{coll\_begin}(p,\tau) \qquad [LHS]$$

$$\Rightarrow [\exists\, q:\ q \neq p:\ \textbf{tx\_begin}(q,\tau-\Delta_{pq})] \qquad [Th6.2]$$

$$\Rightarrow [\exists\, q:\ q \neq p:\ \neg\,\textbf{busy}(q,\tau-\Delta_{pq}-1)] \qquad [I1]$$

$\Rightarrow [\exists q\colon q \neq p\colon [\forall q'\colon q' \neq q\colon$ [ThJ1]

$\neg\,\mathbf{transmitting}(q', \tau - \Delta_{pq} - 1 - \Delta_{qq'})]]$

$\Rightarrow [\exists q\colon q \neq p\colon \neg\,\mathbf{transmitting}(p, \tau - 2\Delta_{pq} - 1)]$

(2)

$\mathbf{coll\_begin}(p, \tau)$ [LHS]

$\Rightarrow \mathbf{transmitting}(p, \tau)$ [D6.4,D6.1]

$\Rightarrow [\exists t_{begin}\colon t_{begin} \leq \tau\colon \mathbf{tx\_begin}(p, t_{begin})$ [I2]

$\wedge [\forall t'\colon t_{begin} \leq t' \leq \tau\colon \mathbf{transmitting}(p, t')]]$

(3)

$\mathbf{coll\_begin}(p, \tau)$ [LHS]

$\Rightarrow [\exists q\colon q \neq p\colon \neg\,\mathbf{transmitting}(p, \tau - 2\Delta_{pq} - 1)]$ [(1),(2)]

$\wedge [\exists t_{begin}\colon t_{begin} \leq \tau\colon \mathbf{tx\_begin}(p, t_{begin})$ [I2]

$\wedge [\forall t'\colon t_{begin} \leq t' \leq \tau\colon \mathbf{transmitting}(p, t')]]$

$\Rightarrow [\exists q, \exists t_{begin}\colon q \neq p \wedge t_{begin} \leq \tau\colon$

$\neg\,\mathbf{transmitting}(p, \tau - 2\Delta_{pq} - 1) \wedge \mathbf{tx\_begin}(p, t_{begin})$

$\wedge [\forall t'\colon t_{begin} \leq t' \leq \tau\colon \mathbf{transmitting}(p, t')]]$

$\Rightarrow [\exists q, \exists t_{begin}\colon q \neq p \wedge t_{begin} \leq \tau\colon$

$\mathbf{tx\_begin}(p, t_{begin}) \wedge \tau - 2\Delta_{pq} - 1 < t_{begin}]$

$\Rightarrow [\exists q\colon q \neq p\colon [\exists t_{begin}\colon \tau - 2\Delta_{pq} \leq t_{begin} \leq \tau\colon \mathbf{tx\_begin}(p, t_{begin})]]$

□

The following corollary to Theorem 6.3 gives the absolute earliest and latest times at which a station could have begun transmission given that it is involved in a collision. It follows from Theorem 6.3 by the fact that the maximum propagation time between stations is

$D$ ticks.

*Corollary T6.3.1:*

$$\textbf{coll\_begin}(p, \tau) \Rightarrow [\exists\, t_{begin}:\ \tau - 2D \le t_{begin} \le \tau:\ \textbf{tx\_begin}(p, t_{begin})]$$

This theorem and corollary give the range of times at which it is possible for a station to have begun a transmission given that the station suffers a collision in the attempt. This range is generally referred to as the *vulnerable period* of a transmission.

Corollary T6.3.2 extends the result of Theorem 6.3 to specify the the interval over which a station transmits given that it suffers a collision. Corollary T6.3.2 follows from Theorem 6.3 by invariant I5.

*Corollary T6.3.2:*

$$\begin{aligned}&\textbf{coll\_begin}(p, t_{coll})\\ &\qquad \Rightarrow [\exists\, t_{begin}:\ t_{coll} - 2D \le t_{begin} \le t_{coll}:\ \textbf{tx\_begin}(p, t_{begin})\\ &\qquad\quad \wedge [\forall t':\ t_{begin} \le t' \le t_{coll}:\ \textbf{transmitting}(p, t')]],\end{aligned}$$

where $D$ is the propagation delay (number of segments) for the network.

The question of the duration of a collision among stations using the **transmit** primitive is now considered.

*Theorem 6.4:*

$$\begin{aligned}&[\forall p,\ \forall q,\ \forall \tau_p,\ \forall \tau_q:\ p \ne q\ \wedge\ |\tau_p - \tau_q| \le \Delta_{pq}:\\ &\quad \textbf{tx\_begin}(p, \tau_p)\ \wedge\ \textbf{tx\_begin}(q, \tau_q)]\\ &\qquad \Rightarrow [\exists\, tbusy_p,\ \exists\, tbusy_q:\ tbusy_p \le \tau_q + \Delta_{pq}\ \wedge\ tbusy_q \le \tau_p + \Delta_{pq}:\\ &\qquad\quad \textbf{coll\_begin}(p, tbusy_p)\ \wedge\ \textbf{coll\_begin}(q, tbusy_q)]]\end{aligned}$$

*Proof:*

(1)

$[\forall p, \forall q, \forall \tau_p, \forall \tau_q: p \neq q \wedge |\tau_p - \tau_q| \leq \Delta_{pq}:$ [LHS]

$\mathbf{tx\_begin}(p, \tau_p) \wedge \mathbf{tx\_begin}(q, \tau_q)]$

$\Rightarrow [\forall p, \forall q, \forall \tau_p, \forall \tau_q: p \neq q \wedge |\tau_p - \tau_q| \leq \Delta_{pq}:$ [I1,ThJ1]

$\neg\, \mathbf{busy}(p, \tau_p - 1) \wedge \mathbf{busy}(p, \tau_q + \Delta_{pq})]$

$\Rightarrow [\forall p, \forall q, \forall \tau_p, \forall \tau_q, \exists\, tbusy_p:$

$q \neq p \wedge |\tau_p - \tau_q| \leq \Delta_{pq} \wedge \tau_p \leq tbusy_p \leq \tau_q + \Delta_{pq}:$

$\neg\, \mathbf{busy}(p, tbusy_p - 1) \wedge \mathbf{busy}(p, tbusy_p)]$

(2)

$\mathbf{tx\_begin}(p, \tau_p)$ [LHS]

$\Rightarrow [\exists T, \exists t_1: min\_size \leq T \leq max\_size \wedge \tau_p \leq t_1 \leq \tau_p + T:$ [I5]

$[\forall t': \tau_p \leq t' \leq t_1: \mathbf{transmitting}(p, t')]$

$\wedge ((t_1 = \tau_p + T - 1) \vee \mathbf{coll\_begin}(p, t_1))]$

(3)

$[\forall p, \forall q, \forall \tau_p, \forall \tau_q: p \neq q \wedge |\tau_p - \tau_q| \leq \Delta_{pq}:$ [LHS]

$\mathbf{tx\_begin}(p, \tau_p) \wedge \mathbf{tx\_begin}(q, \tau_q)]$

$\Rightarrow [\forall p, \forall q, \forall \tau_p, \forall \tau_q, \exists\, tbusy_p:$ [(1),(2)]

$p \neq q \wedge |\tau_p - \tau_q| \leq \Delta_{pq} \wedge \tau_p \leq tbusy_p \leq \tau_q + \Delta_{pq}:$

$\mathbf{busy}(p, tbusy_p)$

$\wedge [\exists T, \exists t_1: min\_size \leq T \leq max\_size \wedge \tau_p \leq t_1 \leq \tau_p + T:$

$$[\forall t': \tau_p \le t' \le t_1: \textbf{transmitting}(p, t')]$$

$$\wedge ((t_1 = \tau_p + T - 1) \vee \textbf{coll\_begin}(p, t_1))]$$

$$\wedge\ tbusy_p < \tau_p + T - 1] \qquad [A6.7]$$

$$\Rightarrow [\forall q, \exists\ tbusy_p: q \ne p \wedge \tau_p \le tbusy_p \le \tau_q + \Delta_{pq}:$$

$$\textbf{coll\_begin}(p, tbusy_p)] \qquad [D6.4]$$

A symmetric argument applies to $q$.

□

Theorem 6.4 states that if two stations begin transmission at times which are separated by less than the propagation time between the stations, a collision results. In the theorem, $tbusy_p \le \tau_q + \Delta_{pq}$ rather than $tbusy_p = \tau_q + \Delta_{pq}$ to indicate that there may be other stations involved in the collision whose signal reached $p$ before the signal from $q$. In the absence of other transmitting stations, the relation would be equality.

In Theorem 6.5, the minimum and maximum duration of a collision is determined.

*Theorem 6.5:*

$$\textbf{coll\_begin}(p, t_{coll})$$

$$\Rightarrow [\exists\ t_{begin}, \exists\ t_{end}: t_{begin} \le t_{coll} \wedge t_{begin} + jam + 1 \le t_{end} \le t_{begin} + 2D + jam + 1:$$

$$\textbf{tx\_begin}(p, t_{begin}) \wedge \textbf{tx\_end}(p, t_{end})]$$

*Proof:*

(1)

$$\textbf{coll\_begin}(p, t_{coll}) \qquad [LHS]$$

$$\Rightarrow [\exists\ t_{begin}: t_{coll} - 2D \le t_{begin} \le t_{coll}: \textbf{tx\_begin}(p, t_{begin})] \qquad [Th6.3]$$

(2)

$$\textbf{coll\_begin}(p, t_{coll}) \qquad \text{[LHS]}$$

$$\Rightarrow [\exists\, t_{end}\colon t_{end} = t_{coll} + jam + 1\colon \textbf{tx\_end}(p, t_{end})] \qquad \text{[I6]}$$

(3)

$$\textbf{coll\_begin}(p, t_{coll}) \qquad \text{[LHS]}$$

$$\Rightarrow [\exists\, t_{begin}, \exists\, t_{end}\colon t_{coll} - 2D \le t_{begin} \le t_{coll} \wedge t_{end} = t_{coll} + jam + 1\colon \qquad \text{[(1),(2)]}$$

$$\textbf{tx\_begin}(p, t_{begin}) \wedge \textbf{tx\_end}(p, t_{end})]$$

$$\Rightarrow [\exists\, t_{begin}, \exists\, t_{end}\colon t_{begin} + jam + 1 \le t_{end} \le t_{begin} + 2D + jam + 1\colon$$

$$\textbf{tx\_begin}(p, t_{begin}) \wedge \textbf{tx\_end}(p, t_{end})]$$

□

Figure 6.1 illustrates the maximum collision duration for a collision between two stations. This result, in combination with Axiom A6.7 allows observing stations to distinguish between successful transmissions and collision fragments based on the length of the observed packet. The introduction of other stations between $a$ and $b$ will reduce the distance between any station and the nearest colliding station, causing the collision to be detected earlier and thus the end of transmission will occur sooner.

The following lemmas describe the behavior of a station given that a busy period occurs at the station. Lemma 6.1 states that given a range over which a station is busy, either the station was transmitting when it became busy or it is not transmitting over the entire busy range plus one tick. Lemma 6.2 states that if station $p$ is transmitting over some range of times, then it was also not busy over a related range of times. The idle period results from either the deferral of other stations due to the signal from $p$, or from other stations aborting their transmissions due the the signal from $p$.

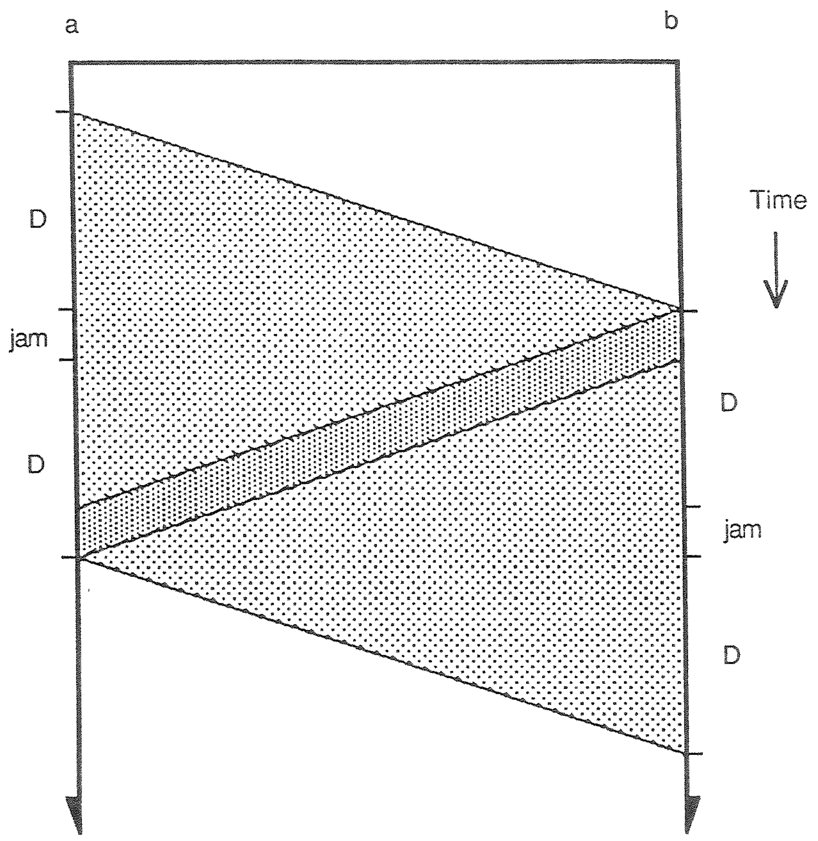


**Figure 6.1:** Maximum collision duration.

*Lemma 6.1:*

$$[\forall t' : t_1 \le t' \le t_2 : \textbf{busy}(p, t')]$$

$$\Rightarrow (\textbf{transmitting}(p, t_1) \vee [\forall t' : t_1 \le t' \le t_2 + 1 : \neg\, \textbf{transmitting}(p, t')])$$

*Proof:*

$$[\forall t' : t_1 \le t' \le t_2 : \textbf{busy}(p, t')]$$

$$\Rightarrow [\forall t' : t_1 \le t' \le t_2 : \neg\, \textbf{tx\_begin}(p, t' + 1)] \qquad [I1]$$

$\Rightarrow [\forall t' : t_1 < t' \leq t_2 + 1:$ [D6.2]

$\neg$ **transmitting** $(p, t') \vee$ **transmitting** $(p, t' - 1)]$

$\Rightarrow$ (**transmitting** $(p, t_1) \vee [\forall t' : t_1 \leq t' \leq t_2 + 1: \neg$ **transmitting** $(p, t')])$

□

Corollary L6.1.1 states that if a station is busy over a range of times and is not transmitting when that range begins then the station is not transmitting over the entire busy range. This follows from the definition of **tx_begin** (D6.2).

*Corollary L6.1.1:*

$[\forall t' : t_1 \leq t' \leq t_2:$ **busy** $(p, t') \wedge \neg$ **transmitting** $(p, t_1)]$

$\Rightarrow [\forall t' : t_1 \leq t' \leq t_2 + 1: \neg$ **transmitting** $(p, t')]$

Corollary L6.1.2 states that if a station is busy over a range of times then either a collision is occurring at the earlier endpoint of the range or the station is not transmitting over the entire range plus 1 tick. This follows from the definition of **collision** (D6.1).

*Corollary L6.1.2:*

$[\forall t' : t_1 \leq t' \leq t_2:$ **busy** $(p, t')]$

$\Rightarrow$ (**collision** $(p, t_1) \vee [\forall t' : t_1 \leq t' \leq t_2 + 1: \neg$ **transmitting** $(p, t')])$

*Lemma 6.2:*

$[\forall t' : t_a \leq t' \leq t_b:$ **transmitting** $(p, t')] \wedge (t_b \geq t_a + jam)$

$\Rightarrow [\forall t' : t_a + jam + 2\Delta_{p,min} \leq t' \leq t_b + 1 + 2\Delta_{p,max}: \neg$ **busy** $(p, t')]$

*Proof:*

$[\forall t':\ t_a \le t' \le t_b:\ \textbf{transmitting}(p, t')] \wedge (t_b \ge t_a + jam)$

(1)...

$\Rightarrow [\forall q:\ q \ne p:\ [\forall t':\ t_a + \Delta_{pq} \le t' \le t_b + \Delta_{pq}:\ \textbf{busy}(q, t')]]$ [ThJ1]

$\wedge\ (t_b \ge t_a + jam)$

$\Rightarrow [\forall q:\ q \ne p:\ \textbf{collision}(q, t_a + \Delta_{pq})$ [CL6.1.2]

$\vee\ [\forall t':\ t_a + \Delta_{pq} \le t' \le t_b + \Delta_{pq} + 1:\ \neg\,\textbf{busy}(q, t')]]$

$\wedge\ (t_b \ge t_a + jam)$

(2)...

$\Rightarrow [\forall q:\ q \ne p:\ [\exists\, t_{endq}:\ t_a + \Delta_{pq} < t_{endq} \le t_a + \Delta_{pq} + jam + 1:$ [I6,I7]

$\neg\,\textbf{transmitting}(q, t_{endq})]$

$\vee\ [\forall t':\ t_a + \Delta_{pq} \le t' \le t_b + \Delta_{pq} + 1:\ \neg\,\textbf{transmitting}(q, t')]]$

$\wedge\ (t_b \ge t_a + jam)$

$\Rightarrow [\forall q:\ q \ne p:$ [(1),(2),CL6.1.1]

$[\exists\, t_{endq}:\ t_a + \Delta_{pq} < t_{endq} \le t_a + \Delta_{pq} + jam + 1:$

$[\forall t':\ t_{endq} \le t' \le t_b + \Delta_{pq} + 1:\ \neg\,\textbf{transmitting}(q, t')]]$

$\vee\ [\forall t':\ t_a + \Delta_{pq} \le t' \le t_b + \Delta_{pq} + 1:\ \neg\,\textbf{transmitting}(q, t')]]$

$\Rightarrow [\forall q:\ q \ne p:[\forall t':\ t_a + \Delta_{pq} + jam + 1 \le t' \le t_b + \Delta_{pq} + 1:$

$\neg\,\textbf{transmitting}(q, t')]]$

$\Rightarrow [\forall q:\ q \ne p:[\forall t':\ t_a + 2\Delta_{pq} + jam + 1 \le t' \le t_b + 2\Delta_{pq} + 1:$ [ThJ1]

$\neg\,\textbf{busy}(p, t')]]$

$$\Rightarrow [\forall t': t_a + 2\Delta_{p,min} + jam + 1 \le t' \le t_b + 2\Delta_{p,max} + 1: \neg \mathbf{busy}(p, t')]$$

□

## 6.3. An Asynchronous Specification of the FCFS Protocol

The FCFS protocol will now be specified using the constructs of [JL87] and the Pascal programming language. The control flow notation of [JL89] is adopted. The FCFS protocol as specified in [TV82] is taken as a starting point, since the performance analysis presented there can be adapted to show the difference between the conservative asynchronous protocol and the aggressive asynchronous protocol, discussed below. A more recent specification for the slotted FCFS protocol which includes some improvements appears in [BG87].

The FCFS protocol is a variant of the Tree protocol [Cape79] using message arrival time as the criterion for subdividing the set of colliding stations rather than addresses or randomly generated bit sequences. Each station keeps track of an enabled window of length $\omega$ which lies $\upsilon$ time units in the past. In keeping with the notation of the previous section, to the current time will be referred to as $\tau$. When a collision occurs, $\omega$ is reduced by half and $\upsilon$ adjusted to reflect the length of time which has passed and the change in $\omega$. If a station finds that the arrival time of its packet is within the new window, the packet is transmitted, otherwise, the station observes the network to discover the outcome of the transmission of stations with packets whose arrival time falls within the new window. This is repeated until two consecutive successes are observed, at which time $\omega$ is reset to a predetermined quantity, $\omega_0$. Because it is important for all stations to search the same window, it is necessary for a station to monitor the network at all times and adjust $\omega$ and $\upsilon$, even when there are not packets waiting for transmission.

The system uses three shared variables, **transmitting**($p$), **busy**($p$) and **packet_to_send**($p$). **Transmitting**($p$) is set by station $p$ when it begins transmission and cleared when the transmission ends. These alterations to the value of **transmitting** occur only in the **transmit** procedure. While **transmitting**($p$) is true, the segment to which $p$ is attached is in state **transmitting**. **Transmitting**($p$) is shared with the channel process, for which it is a read-only variable. **Busy($p$)** is set by the channel when the segment to which station $p$ is attached becomes busy from the left or right. It is shared with station $p$, for which it is a read-only variable. **Packet_to_send($p$)** is shared between the station program and a user process. The user process sets **packet_to_send($p$)** to true when it submits a packet for transmission, and the station program sets it to false when the packet is successfully transmitted. The timestamp arrival_time is assumed to be generated automatically when **packet_to_send($p$)** is set by the user process. **Transmit($p$)**, **busy($p$)** and **packet_to_send($p$)** appear as transmit, busy and packet_to_send in the statement of the station program. The following condition is used in the station program.

$$\textbf{enabled} \equiv \textbf{packet\_to\_send} \wedge \tau - \upsilon - \omega < \text{arrival\_time} < \tau - \upsilon \qquad \text{(C6.2)}$$

C6.2 indicates that the arrival time of the station's packet falls within the currently enabled window. It is assumed that $\omega_0$ is an integer such that $\omega_0 = 2^n$ for some $n$. $\omega_0$ represents the number of segments in the initial enabled window. When the window is split, integer division is used to perform the calculation. Initially, $\tau$, $\upsilon$ and $\omega$ are 0. One iteration of the outer loop will occur before any packets can be enabled.

## Conservative Asynchronous FCFS Station Program

```
begin
    ⟨S1⟩ repeat
        ⟨S2⟩ if enabled then
            begin
                ⟨S3⟩ tx_start := $\tau$;
                ⟨S4⟩ transmit(T, label);
                ⟨S5⟩ tx_stop := $\tau$ - 1;
            end;
        else
            begin
                ⟨S6⟩ wait_start := $\tau$;
                ⟨S7⟩ wait (busy or 2*D);
                ⟨S8⟩ if not busy then
                    begin
                        ⟨S9⟩ $\omega$ := min ($\omega_0$, $\upsilon$ + 2*D);
                        ⟨S10⟩ $\upsilon$ := max (0, $\upsilon$ + 2*D - $\omega_0$);
                    ⟨S11⟩ label := idle;
                end
            else
                begin
                    ⟨S12⟩ tx_start := $\tau$;
                    ⟨S13⟩ wait(not busy);
                    ⟨S14⟩ tx_stop := $\tau$;
                    ⟨S15⟩ if (tx_stop - tx_start) < min_size then
                            ⟨S16⟩ label := other_coll
                    else
                            ⟨S17⟩ label := other_tx
                end;
        end;
⟨S18⟩ if label = other_tx then
        begin
            ⟨S19⟩ $\omega$ := min ($\omega_0$, $\upsilon$ + (tx_stop - wait_start))
            ⟨S20⟩ $\upsilon$ := max (0, $\upsilon$ + (tx_stop - wait_start) - $\omega_0$);
        end
⟨S21⟩ else if label = packet_sent then
        begin
            ⟨S22⟩ $\omega$ := min ($\omega_0$, $\upsilon$ + tx_time);
            ⟨S23⟩ $\upsilon$ := max (0, $\upsilon$ + tx_time + 1 - $\omega_0$);
            ⟨S24⟩ packet_to_send := false
        end
⟨S25⟩ else if label = coll_detected or label = other_coll then
        begin
            ⟨S26⟩ if label = coll_detected then
                    begin
                        ⟨S27⟩ if (tx_stop - tx_start) < (2D + jam + 1) then
                                ⟨S28⟩ delay ((2D + jam + 1)
                                        - (tx_stop - tx_start));
                        ⟨S29⟩ $\upsilon$ := $\upsilon$ + 2*D + jam + 1 + $\omega/2$ ;
                    end
                else
                    begin
                        ⟨S30⟩ if (tx_stop - tx_start) < (2D + jam + 1) then
                                ⟨S31⟩ delay ((2D + jam + 1)
                                        - (tx_stop - tx_start));
                        ⟨S32⟩ $\upsilon$ := $\upsilon$ + (tx_start - wait_start)
                                + 2*D + jam + 1 + $\omega/2$ ;
                    end
            ⟨S33⟩ $\omega$ := $\omega/2$;
```

```
⟨S34⟩ while not (label = packet_sent or label = other_tx) do begin
      ⟨S35⟩ if enabled then
              begin
                    ⟨S36⟩ tx_start := τ;
                    ⟨S37⟩ transmit(T, label);
                    ⟨S38⟩ tx_stop := τ – 1;
              end
      else
              begin
                    ⟨S39⟩ wait_start := τ;
                    ⟨S40⟩ wait (busy or 2*D);
                    ⟨S41⟩ if not busy then
                       begin
                          ⟨S42⟩ υ := υ + 2*D - ω/2;
                          ⟨S43⟩ ω := ω/2;
                          ⟨S44⟩ label := idle
                       end
                    else
                       begin
                          ⟨S45⟩ tx_start := τ;
                          ⟨S46⟩ wait(not busy);
                          ⟨S47⟩ tx_stop := τ;
                          ⟨S48⟩ if (tx_stop - tx_start) < min_size
                                then
                                    ⟨S49⟩ label := other_coll
                             else
                                    ⟨S50⟩ label := other_tx
                       end;
              end;
      ⟨S51⟩ if label = other_tx then
              ⟨S52⟩ υ := υ + (tx_stop - wait_start) - ω;
      ⟨S53⟩ else if label = packet_sent then
              begin
                    ⟨S54⟩ υ := υ + tx_time + 1 - ω;
                    ⟨S55⟩ packet_to_send := false
              end
      ⟨S56⟩ else if label = coll_detected or label = other_coll
         then
              begin
                    ⟨S57⟩ if label = coll_detected then
                            begin
                               ⟨S58⟩ if (tx_stop - tx_start)
                                   < (2D + jam + 1) then
```

```
                                        ⟨S59⟩ delay ((2*D + jam + 1)
                                                - (tx_stop - tx_start));
                                    ⟨S60⟩ $\upsilon := \upsilon + 2*D + jam + 1 + \omega/2$;
                                  end
                              else
                                  begin
                                      ⟨S61⟩ if (tx_stop - tx_start) < (2D +
                                      jam + 1) then
                                          ⟨S62⟩ delay ((2D + jam + 1)
                                                  - (tx_stop - tx_start));
                                      ⟨S63⟩ $\upsilon := \upsilon$ + (tx_start - wait_start)
                                              + 2D + jam + 1 + $\omega/2$;
                                  end
                              ⟨S64⟩ $\omega := \omega/2$
                          end
                  end
              end     (* End while *)
    forever (* End of repeat *)
end
```

## 6.4. Bounded Delay of the Asynchronous FCFS Protocol

The bounded delay of the Asynchronous FCFS protocol as specified in the previous section is now considered. It will be shown that a collision experienced by stations while the algorithm is in its initial state will eventually result in at least one successful transmission followed by a successful transmission or idle step which returns the algorithm to the initial state. In [BG87], the protocol is discussed in terms of searching the left and right halves of the original enabled window. Statements S1 through S24 carry out the "right interval search," while the loop composed of statements S34 through S64 perform the "left interval search." The initial interval is thus considered to be a "right " subinterval. The protocol is constructed so that whenever a subdivision is performed, the previous right interval is "forgotten," to be included in the initial window of the next resolution round. This means that it is possible for the protocol to resolve only a fraction of the original window in any given execution of the resolution process. The resulting stability criterion for the asynchronous version of this protocol is discussed in [TV82]. When a successful transmission results from the left interval search, the current right interval is resolved. Observing a successful transmission or an idle period in the right interval search results in the protocol resetting to its initial state. A *contention epoch* is defined to be the period of time beginning with a station in the initial state ($\omega = \omega_0$) experiencing a collision while attempting to transmit a packet, and ending with a successful transmission or idle in the right interval search, resulting in a return to the initial state. Thus, to demonstrate that the protocol progresses, it must be shown that a contention epoch takes bounded time. It is assumed that the execution of the **set** construct takes one tick, and that comparisons and assignments take no time.

The station program is constructed so that the *transmitting* flag is altered only in the transmit procedure. This means that whenever station program control is not in the calls to the transmit procedure (S4 and S37) it can be inferred that the station is not transmitting.

Note that the possibility for simultaneous packet arrivals exists in the model. Since the protocol resolves collisions by subdividing the colliding stations according to their arrival times, it will be unable to resolve a conflict between two stations with packets which arrived during the same clock tick. Initially it will be assumed that arrival times are distinct; modifications will subsequently be discussed to allow to the protocol to handle simultaneous arrivals.

In the following proofs it is sometimes necessary to distinguish between the value of local parameters of the various stations at different times in the execution of the protocol. When this is the case, arguments indicating the station and the time are appended to the variable name, e.g. $\omega(p, t)$, where $p$ is the station name and $t$ is the time.

The properties of the right interval search are considered first. The actions of the left interval search are identical to those of the right interval search, with the only difference being the interval of arrival times upon which the algorithm operates. Thus, a single iteration of the left and right interval searches have identical properties. The following are invariants for the portion of the station program which implements the right interval search.

SI 1: $\mathbf{at}(p, S2, t)$

$$\Rightarrow (\mathbf{enabled}(p, t) \Rightarrow \mathbf{at}(p, S4, t)) \wedge (\neg\,\mathbf{enabled}(p, t) \Rightarrow \mathbf{at}(p, S7, t))$$

SI 2: $\mathbf{at}(p, S4, t_1)$

$\Rightarrow [\exists\, t_2\colon\ t_1 + jam < t_2\colon\ \mathbf{at}(p, S5, t_2) \wedge \neg\, \mathbf{transmitting}(p, t_2)$

$\wedge\ (\text{label}(p, t_2) = \text{packet\_sent}$

$\Rightarrow \mathbf{at}(p, S2, t_2) \wedge [\exists\, T\colon\ min\_size < T \le max\_size\colon\ t_2 = t_1 + T + 1$

$\wedge\ [\forall t'\colon\ t_2 - T \le t' < t_2\colon\ \mathbf{transmitting}(p, t') \wedge \neg\, \mathbf{busy}(p, t')]])$

$\wedge\ (\text{label}(p, t_2) = \text{coll\_detected}$

$\Rightarrow t_2 \le t_1 + 2D + jam + 1 \wedge \mathbf{coll\_begin}(p, t_2 - jam - 1)$

$\wedge\ \mathbf{at}(p, S5, t_2))$

$\wedge\ [\forall t''\colon\ t_2 - jam - 1 \le t'' < t_2\colon\ \mathbf{transmitting}(p, t'')]]$

SI 3: $\mathbf{at}(p, S5, t) \wedge \text{label}(p, t) = \text{coll\_detected}$

$\Rightarrow \mathbf{coll\_begin}(p, t - jam - 1)$

$\wedge\ [\exists\, t_1\colon\ t_1 \le t - jam - 1\colon\ \mathbf{tx\_start}(p, t_1)$

$\wedge\ [\forall t'\colon\ t_1 \le t' < t\colon\ \mathbf{transmitting}(p, t')]$

$\wedge\ [\exists\, t''\colon\ t'' \le t_1 + 2D + jam + 1\colon\ \mathbf{at}(p, S35, t'')]]$

SI 4: $\mathbf{at}(p, S7, t_1)$

$\Rightarrow [\exists\, t_2\colon\ t_2 > t_1\colon\ \mathbf{after}(p, S7, t_2)$

$\wedge\ ((t_2 = t_1 + 2D \wedge [\forall t'\colon\ t_1 < t' \le t_2\colon\ \neg\, \mathbf{busy}(p, t')])$

$\Rightarrow \mathbf{at}(p, S2, t_1 + 2D))$

$\wedge\ ((t_2 \le t_1 + 2D \wedge \mathbf{busy}(p, t_2) \wedge [\forall t''\colon\ t_1 < t'' < t_2\colon\ \neg\, \mathbf{busy}(p, t'')]$

$\Rightarrow \mathbf{at}(p, S13, t_2))]$

SI 5: $\textbf{at}(p, S13, t_1) \wedge \textbf{after}(p, S13, t_2)$

$$\Rightarrow [\forall t': t_1 < t' < t_2: \textbf{busy}(p, t')] \wedge \neg\,\textbf{busy}(p, t_2)$$

$$\wedge\ (t_2 - t_1 \le \mathit{min\_size} \Rightarrow \text{label}(p, t_2) = \text{other\_coll} \wedge \textbf{at}(p, S30, t_2))$$

$$\wedge\ (t_2 - t_1 > \mathit{min\_size} \Rightarrow \text{label}(p, t_2) = \text{other\_tx} \wedge \textbf{at}(p, S2, t_2))$$

SI 6: $\textbf{at}(p, S30, t_2)$

$$\Rightarrow \text{label}(p, t_2) = \text{other\_coll} \wedge \neg\,\textbf{busy}(p, t_2)$$

$$\wedge\ [\exists t_1: t_1 < t_2: [\forall t': t_1 < t' < t_2: \textbf{busy}(p, t')] \wedge \neg\,\textbf{busy}(p, t_1)$$

$$\wedge\ [\exists t_3: t_3 \le t_1 + 2D + \mathit{jam} + 1: \textbf{at}(p, S35, t_3)]]$$

These invariants enumerate the possible flows of control for the right interval search. A single execution of the right interval search is shown to be bounded by demonstrating that each possible flow of control leads to all stations reaching synchronization at either statement S2 or statement S35 in bounded time. Several new definitions are used in the proof.

D 6.6: $\textbf{t\_detect}(p, t_p) \equiv \neg\,\textbf{busy}(p, t_p - 1) \wedge (\textbf{busy}(p, t_p) \vee \textbf{tx\_begin}(p, t_p))$

D 6.7: $\text{tx\_time}(p) \equiv \text{tx\_stop}(\text{p}) - \text{tx\_start}(\text{p})$.

D 6.8: $\textbf{synch}(p, t_p, q, t_q, S) \equiv \textbf{at}(p, S, t_p) \wedge \textbf{at}(q, S, t_q) \wedge |t_p - t_q| \le \Delta_{pq}$

D 6.9: $\textbf{ss\_busy}(p, t, q) \equiv \textbf{transmitting}(q, t - \Delta_{pq})$

D 6.10: $\textbf{quiescent}(p, t) \equiv \neg\,\textbf{busy}(p, t) \wedge \neg\,\textbf{transmitting}(p, t)$

The history variable **t_detect** is used to indicate the time at which a station either became busy or began transmitting. This is the reference time used to determine whether an observed transmission is a collision or a successful transmission. Tx_time is the duration of an observed or attempted transmission. Note that tx_start corresponds to the time at which **t_detect** becomes true. The history variable **synch** is the synchronization constraint for the station program. It indicates that two stations arrived at the same point in the algorithm at times which are separated by less than the propagation delay between the two stations. The final definition is a *station specific* busy condition, i.e. it indicates that a station is busy as the result of a signal from some specific station. When $\neg$ **ss_busy** is true for all stations, $\neg$ **busy** is true. The history variable **quiescent** indicates that there is no activity at a station at a given time.

Each of the following theorems assumes that initially, all stations are synchronized as per **D6.8** at statement S2 and that all stations are quiescent as per **D6.10** at this synchronization point. The results state that within bounded time, all stations are synchronized at statement S2 again or all stations are synchronized at statement S35 and are quiescent again. From the fact that the initial conditions are reestablished in each instance, it is possible to conclude that all single executions of the right interval search are bounded.

## Right Interval: Idle Interval

*Theorem 6.6:*

$$\begin{aligned}&\mathbf{synch}(p, t_p, q, t_q, S2)\\&\wedge\ \mathbf{quiescent}(p, t_p) \wedge \mathbf{quiescent}(q, t_q)\\&\wedge\ \mathbf{at}(p, S2, t_p+2D) \wedge \mathrm{label}(p, t_p+2D) = \mathrm{idle}\\&\qquad\Rightarrow [\exists\, t_{e,q} :: \mathbf{synch}(p, t_p+2D, q, t_{e,q}, S2) \wedge \mathrm{label}(q, t_{e,q}) = \mathrm{idle}\\&\qquad\quad \wedge\ \mathbf{quiescent}(p, t_p+2D) \wedge \mathbf{quiescent}(q, t_{e,q})]\end{aligned}$$

*Proof:*

(1)

$$\mathbf{synch}(p, t_p, q, t_q, S2) \wedge \mathbf{at}(p, S2, t_p+2D) \wedge \mathrm{label}(p, t_p+2D) = \mathrm{idle} \qquad [\mathrm{LHS}]$$

$$\Rightarrow \mathbf{at}(p, S2, t_p) \wedge \neg\,\mathbf{enabled}(p, t_p)$$

$$\Rightarrow \mathbf{at}(p, S7, t_p) \qquad [\mathrm{SI1}]$$

(2)

$\mathbf{at}(p, S7, t_p) \wedge \mathbf{at}(p, S2, t_p + 2D) \wedge \text{label}(p, t_p + 2D) = \text{idle}$ [(1),LHS]

$\Rightarrow [\forall t' : t_p \le t' \le t_p + 2D : \neg\, \mathbf{busy}(p, t') \wedge \neg\, \mathbf{transmitting}(p, t')]$ [SI4]

$\Rightarrow [\forall t' : t_p \le t' \le t_p + 2D : \neg\, \mathbf{transmitting}(q, t' - \Delta_{pq})]$ [ThJ1,D6.10]

$\wedge\ \mathbf{quiescent}(p, t_p + 2D)$

(3)

$\mathbf{synch}(p, t_p, q, t_q, S2)$ [LHS,(2)]

$\wedge\ [\forall t' : t_p \le t' \le t_p + 2D : \neg\, \mathbf{transmitting}(q, t' - \Delta_{pq})]$

$\Rightarrow \mathbf{at}(q, S2, t_q) \wedge \neg\, \mathbf{enabled}(q, t_q)$

$\Rightarrow \mathbf{at}(q, S7, t_q)$ [SI1]

(4)

$\mathbf{synch}(p, t_p, q, t_q, S7)$ [(1),(3)]

$\Rightarrow [\forall r : r \ne q : \mathbf{synch}(q, t_q, r, t_r, S7)$ [D6.5,D6.8]

$\wedge\ [\forall t' : t_r \le t' \le t_r + 2D : \neg\, \mathbf{transmitting}(r, t')]$

$\wedge\ t_r + 2D + \Delta_{rq} \ge t_q + 2D\,]$

$\Rightarrow [\forall r : r \ne q : [\forall t' : t_q \le t' \le t_q + 2D : \neg\, \mathbf{ss\_busy}(q, t')]]$ [ThJ1,D6.9]

$\Rightarrow [\forall t' : t_q \le t' \le t_q + 2D : \neg\, \mathbf{busy}(q, t') \wedge \neg\, \mathbf{transmitting}(q, t')]$

$\Rightarrow \mathbf{at}(q, S2, t_q + 2D) \wedge \text{label}(q, t_q + 2D) = \text{idle}$ [SI4,D6.10]

$\wedge\ \mathbf{quiescent}(q, t_q + 2D)$

(5)

$\mathbf{at}(p, S2, t_p + 2D) \wedge \text{label}(p, t_p + 2D) = \text{idle}$ [LHS,(2),(4),(5)]

$\wedge$ **quiescent** $(p, t_p + 2D)$

$\wedge$ **at** $(q, S2, t_q + 2D)$ $\wedge$ label $(q, t_q + 2D)$ = idle $\wedge$ **quiescent** $(q, t_q + 2D)$

$\Rightarrow$ [ $\exists\, t_{e,q}$:: **synch** $(p, t_p + 2D, q, t_{e,q}, S2)$ $\wedge$ label $(q, t_{e,q})$ = idle

$\wedge$ **quiescent** $(p, t_p + 2D)$ $\wedge$ **quiescent** $(q, t_{e,q})$]

□

## Right Interval: Successful Transmission, Transmitting Station

*Theorem 6.7:*

**synch** $(p, t_p, q, t_q, S2)$ $\wedge$ **quiescent** $(p, t_p)$ $\wedge$ **quiescent** $(q, t_q)$

$\wedge$ [ $\exists\, T$: *min_size* $< T \leq$ *max_size*: **at** $(p, S2, t_p + T + 1)$

$\wedge$ label $(p, t_p + T + 1)$ = packet_sent ]

$\Rightarrow$ [ $\exists\, T$: *min_size* $< T \leq$ *max_size*:

**synch** $(p, t_p + T + 1, q, t_p + T + 1 + \Delta_{pq}, S2)$

$\wedge$ label $(q, t_p + T + 1 + \Delta_{pq})$ = other_tx

$\wedge$ **quiescent** $(p, t_p + T + 1)$ $\wedge$ **quiescent** $(q, t_p + T + 1 + \Delta_{pq})$]

*Proof:*

(1)

**synch** $(p, t_p, q, t_q, S2)$ [LHS]

$\wedge$ [ $\exists\, T$: *min_size* $< T \leq$ *max_size*: **at** $(p, S2, t_p + T + 1)$

$\wedge$ label $(p, t_p + T + 1)$ = packet_sent ]

$\Rightarrow$ **at** $(p, S2, t_p)$ $\wedge$ **enabled** $(p, t_p)$

$\Rightarrow$ **at** $(p, S4, t_p)$ [SI1]

(2)

$\mathbf{at}(p, S4, t_p) \wedge \neg \mathbf{busy}(p, t_p)$ [(1),LHS,D6.10]

$\wedge \text{ label}(p, t_p + T + 1) = \text{packet\_sent}$

$\Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon \mathbf{at}(p, S2, t_p + T + 1)$ [SI2]

$\wedge [\forall t'\colon t_p + 1 \le t' < t_p + T + 1\colon$

$\mathbf{transmitting}(p, t') \wedge \neg \mathbf{busy}(p, t')]]$

$\Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon [\forall t'\colon t_p + 1 \le t' < t_p + T + 1\colon$ [ThJ1]

$\neg \mathbf{transmitting}(q, t' - \Delta_{pq}) \wedge \mathbf{busy}(q, t' + \Delta_{pq})]]$

(3)

$\mathbf{synch}(p, t_p, q, t_q, S2)$ [LHS,(2)]

$\wedge [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon [\forall t'\colon t_p + 1 \le t' < t_p + T + 1\colon$

$\neg \mathbf{transmitting}(q, t' - \Delta_{pq}) \wedge \mathbf{busy}(q, t' + \Delta_{pq})]]$

$\Rightarrow \mathbf{at}(q, S2, t_q) \wedge \neg \mathbf{enabled}(q, t_q)$

$\wedge [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon [\forall t'\colon t_p + 1 \le t' < t_p + T + 1\colon$

$\neg \mathbf{transmitting}(q, t' - \Delta_{pq}) \wedge \mathbf{busy}(q, t' + \Delta_{pq})]]$

$\Rightarrow \mathbf{at}(q, S7, t_q) \wedge t_q < t_p + 1 + \Delta_{pq} < t_q + 2D$ [SI1,D6.8]

$\wedge [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon [\forall t'\colon t_p + 1 \le t' < t_p + T + 1\colon$

$\neg \mathbf{transmitting}(q, t' + \Delta_{pq}) \wedge \mathbf{busy}(q, t' + \Delta_{pq})]]$

$\Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon$ [SI4]

$\mathbf{at}(q, S13, t_p + 1 + \Delta_{pq}) \wedge \mathbf{after}(q, S13, t_p + T + 1 + \Delta_{pq})$

$\wedge [\forall t'\colon t_p + 1 \le t' < t_p + T + 1 + \Delta_{pq}\colon \neg \mathbf{ss\_busy}(p, t' + \Delta_{pq}, q)]]$

$\Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon \mathbf{at}(q, S2, t_p + T + 1 + \Delta_{pq})$ [SI5]

$\wedge$ label $(q, t_p + T + 1 + \Delta_{pq})$ = other_tx

$\wedge \neg$ **busy** $(q, t_p + T + 1 + \Delta_{pq}) \wedge \neg$ **transmitting** $(q, t_p + T + 1 + \Delta_{pq})$

$\wedge \neg$ **ss_busy** $(p, t_p + T + 1, q)]$

$\Rightarrow [\exists T$: *min_size* $< T \leq$ *max_size*: **at** $(q, S2, t_p + T + 1 + \Delta_{pq})$ [D6.9,D6.10]

$\wedge$ label $(q, t_p + T + 1 + \Delta_{pq})$ = other_tx

$\wedge$ **quiescent** $(q, t_p + T + 1 + \Delta_{pq})$

$\wedge \neg$ **busy** $(p, t_p + T + 1)]$

(4)

$[\exists T$: *min_size* $< T \leq$ *max_size*: [LHS,(3)]

**at** $(p, S2, t_p + T + 1) \wedge$ **at** $(q, S2, t_p + T + 1 + \Delta_{pq})$

$\wedge$ label $(q, t_p + T + 1 + \Delta_{pq})$ = other_tx

$\wedge \neg$ **transmitting** $(p, t_p + T + 1) \wedge \neg$ **busy** $(p, t_p + T + 1)]$

$\wedge$ **quiescent** $(q, t_p + T + 1 + \Delta_{pq})]$

$\Rightarrow [\exists T$: *min_size* $< T \leq$ *max_size*: [D6.10]

**synch** $(p, t_p + T + 1, q, t_p + T + 1 + \Delta_{pq}, S2)$

$\wedge$ label $(q, t_p + T + 1 + \Delta_{pq})$ = other_tx

$\wedge$ **quiescent** $(p, t_p + T + 1) \wedge$ **quiescent** $(q, t_p + T + 1 + \Delta_{pq})]$

□

## Right Interval: Successful Transmission, Observing Station

*Theorem 6.8:*

**synch** $(p, t_p, q, t_q, S2)$

$\wedge$ **quiescent** $(p, t_p) \wedge$ **quiescent** $(q, t_q)$

$\wedge\ [\exists\, t_w\colon\ t_w \le 2D\colon\ [\exists\, T\colon\ \mathit{min\_size} < T \le \mathit{max\_size}\colon$

$\quad \mathbf{at}(p, S2, t_p + t_w + T)\ \wedge\ \mathrm{label}(p, t_p + t_w + T) = \mathrm{other\_tx}\,]]$

$\quad\ \Rightarrow\ [\exists\, t_w\colon\ t_w \le 2D\colon\ [\exists\, T\colon\ \mathit{min\_size} < T \le \mathit{max\_size}\colon\ [\exists\, t_{e,q}\colon\colon$

$\qquad \mathbf{synch}(p, t_p + t_w + T, q, t_{e,q}, S2)$

$\qquad\ \wedge\ (\mathrm{label}(q, t_{e,q}) = \mathrm{other\_tx}\ \vee\ \mathrm{label}(q, t_{e,q}) = \mathrm{packet\_sent})$

$\qquad\ \wedge\ \mathbf{quiescent}(p, t_p + t_w + T)\ \wedge\ \mathbf{quiescent}(q, t_{e,q})]]]$

*Proof:*

(1)

$\mathbf{synch}(p, t_p, q, t_q, S2)$ [LHS]

$\wedge\ [\exists\, t_w\colon\ t_w \le 2D\colon\ [\exists\, T\colon\ \mathit{min\_size} < T \le \mathit{max\_size}\colon\ \mathbf{at}(p, S2, t_p + t_w + T)$

$\quad \wedge\ \mathrm{label}(p, t_p + t_w + T) = \mathrm{other\_tx}\,]]$

$\quad\ \Rightarrow\ \mathbf{at}(p, S2, t_p)\ \wedge\ \neg\,\mathbf{enabled}(p, t_p)$

$\qquad \wedge\ [\exists\, t_w\colon\ t_w \le 2D\colon\ [\exists\, T\colon\ \mathit{min\_size} < T \le \mathit{max\_size}\colon$

$\qquad\quad [\forall t'\colon\ t_p \le t' < t_p + t_w\colon\ \neg\,\mathbf{busy}(p, t')]$

$\qquad\quad\ \wedge\ [\forall t''\colon\ t_p + t_w \le t'' < t_p + t_w + T\colon\ \mathbf{busy}(p, t'')]$

$\qquad\quad\ \wedge\ \neg\,\mathbf{busy}(p, t_p + t_w + T)\ \wedge\ \neg\,\mathbf{transmitting}(p, t_p + t_w + T)]]$

$\quad\ \Rightarrow\ [\exists\, t_w\colon\ t_w \le 2D\colon\ [\exists\, T\colon\ \mathit{min\_size} < T \le \mathit{max\_size}\colon$ [ThJ1,D6.10]

$\qquad \mathbf{quiescent}(p, t_p + t_w + T)$

$\qquad \wedge\ [\exists\, r\colon\ r \ne p\colon\ [\forall t''\colon\ t_p + t_w \le t'' < t_p + t_w + T\colon$

$\qquad\qquad \mathbf{transmitting}(r, t' - \Delta_{pr})]]]]$

(2)

$\mathbf{synch}(p, t_p, q, t_q, S2)$ [LHS,(1)]

$\wedge\ [\exists\ t_w\!: t_w \le 2D : [\exists\ T\!: min\_size < T \le max\_size :$

$[\exists\ r\!: r \ne p : [\forall t''\!: t_p + t_w \le t'' < t_p + t_w + T : \mathbf{transmitting}(r, t' - \Delta_{pr})]]]]$

$\Rightarrow (\neg\, \mathbf{enabled}(q, t_q) \Rightarrow \mathbf{at}(q, S7, t_q))$ [SI1,D6.8]

$\wedge\ (\mathbf{enabled}(q, t_q) \Rightarrow \mathbf{at}(q, S7, t_q))$

$\wedge\ [\exists\ t_w\!: t_w \le 2D : [\exists\ T\!: min\_size < T \le max\_size :$

$[\exists\ r\!: r \ne p : [\forall t''\!: t_p + t_w \le t'' < t_p + t_w + T :$

$\mathbf{transmitting}(r, t' - \Delta_{pr})]]]]$

$\Rightarrow (\ \mathbf{at}(q, S7, t_q)$ [ThJ1]

$\wedge\ [\exists\ t_w\!: t_w \le 2D : [\exists\ T\!: min\_size < T \le max\_size :$

$[\exists\ r\!: r \ne p : [\forall t'\!: t_p + t_w \le t' < t_p + t_w + T :$

$\mathbf{busy}(q, t' - \Delta_{rp} + \Delta_{rq})]$

$\wedge\ \neg\, \mathbf{busy}(q, t_p + t_w + T - \Delta_{rp} + \Delta_{rq})$

$\wedge\ [\forall t''\!: t_q \le t'' \le t_p + t_w + T - \Delta_{rp} + \Delta_{rq}\!:$

$\neg\, \mathbf{transmitting}(q, t''\,)]]]]])$

$\vee\ (\ \mathbf{at}(q, S4, t_q)$

$[\exists\ T\!: min\_size < T \le max\_size :$

$[\forall t'\!: t_q + 1 \le t' < t_q + 1 + T : \mathbf{transmitting}(q, t'\,)]$

$\wedge\ \neg\, \mathbf{transmitting}(q, t_q + 1 + T)])$

$\Rightarrow [\exists\ T\!: min\_size < T \le max\_size :$

$([\exists\ r\!: r \ne p : [\exists\ t_w\!: t_w \le 2D : \mathbf{at}(q, S13, t_p + t_w - \Delta_{rp} + \Delta_{rq})$ [SI4]

$\wedge\ \mathbf{after}(q, S13, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})$

$\wedge \neg$ **busy** $(q, t_p + t_w + T - \Delta_{rp} + \Delta_{rq})$

$\wedge \neg$ **transmitting** $(q, t_p + t_w + T - \Delta_{rp} + \Delta_{rq})])$

$\vee$ ( **after** $(q, S4, t_q + 1 + T)$ $\wedge$ label $(q, t_q + 1 + T)$ = packet_sent ) [SI2]

$\wedge$ $[\forall s\colon s \neq q\colon [\forall t'\colon t_q + 1 + \Delta_{qs} \le t' \le t_q + T + \Delta_{qs} + 1\colon$

**busy** $(s, t')$ $\wedge \neg$ **transmitting** $(s, t')]])]]$ [ThJ1]

$\Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon$

$([\exists r\colon r \neq p\colon [\exists t_w\colon t_w \le 2D\colon$

**at** $(q, S2, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})$ [SI5]

$\wedge$ label $(q, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})$ = other_tx

$\wedge$ **quiescent** $(q, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})]])$ [D6.10]

$\vee$ ( **at** $(q, S2, t_q + 1 + T)$ $\wedge$ label $(q, t_q + 1 + T)$ = packet_sent ) [SI2]

$\wedge$ $[\forall s\colon s \neq q\colon [\forall t'\colon t_q + 1 + \Delta_{qs} \le t' \le t_q + T + \Delta_{qs} + 1\colon$ [ThJ1]

$\neg$ **ss_busy** $(q, t', s)]]])]$

(3)

$[\exists t_w\colon t_w \le 2D\colon [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon$ [LHS,(1),(2)]

**at** $(p, S2, t_p + t_w + T)$ $\wedge$ **quiescent** $(p, t_p + t_w + T)$

$\wedge$ $([\exists r\colon r \neq p\colon$ **at** $(q, S2, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})$

$\wedge$ label $(q, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})$ = other_tx

$\wedge$ **quiescent** $(q, t_p + t_w - \Delta_{rp} + T + \Delta_{rq})])$

$\vee$ ( **at** $(q, S2, t_q + 1 + T)$ $\wedge$ label $(q, t_q + 1 + T)$ = packet_sent )

$\wedge$ $[\forall s\colon s \neq q\colon [\forall t'\colon t_q + 1 + \Delta_{qs} \le t' \le t_q + T + \Delta_{qs} + 1\colon$

$\neg$ **ss_busy** $(q, t', s)]$

$\wedge \neg$ **transmitting** $(q, t_q + T + 1)])]]$

$\Rightarrow [\exists\, t_w\colon\ t_w \leq 2D\colon [\exists\, T\colon\ min\_size < T \leq max\_size\colon [\exists\, t_{e,q}\colon$ [D6.8,D6.10]

**synch** $(p, t_p + t_w + T, q, t_{e,q}, S2)$

$\wedge$ (label $(q, t_{e,q})$ = other_tx $\vee$ label $(q, t_{e,q})$ = packet_sent )

$\wedge$ **quiescent** $(p, t_p + t_w + T)$ $\wedge$ **quiescent** $(q, t_{e,q})]]]$

□

## Right Interval: Collision, Participating Station

The proof of boundedness in the case of collision requires the following result concerning the time at which an observing station once again finds the channel idle given that a collision occurs. The following notation is used: $t_{b,s}$ indicates the time at which a station $s$ is synchronized at statement S2; $t_{d,s}$ indicated the time at which a station $s$ detects a transmission in the case where $s$ is not enabled; and $t_{c,s}$ indicates the time at which a transmitting station $s$ detects a collision.

*Lemma 6.3:*

**synch** $(p, t_{b,p}, r, t_{b,r}, S2)$ $\wedge$ **t_detect** $(r, t_{d,r})$ $\wedge$ **tx_begin** $(p, t_{b,p} + 1)$

$\wedge$ **coll_begin** $(p, t_{c,p})$ $\wedge$ $t_{b,r} \leq t_{d,r}$ $\wedge$ $t_{b,p} + 1 \leq t_{c,p}$

$\Rightarrow \neg$ **busy** $(r, t_{d,r} + 2D + jam + 1)$ $\wedge$ **at** $(r, S35, t_{d,r} + 2D + jam + 1)$

*Proof:*

(1)

**at** $(r, S2, t_{b,r})$ [LHS,D6.8]

$\Rightarrow$ ($\neg$ **enabled** $(r, t_{b,r})$ $\Rightarrow$ **at** $(r, S7, t_{b,r})$) [SI1]

$\wedge$ (**enabled** $(r, t_{b,r})$ $\Rightarrow$ **at** $(r, S4, t_{b,r})$)

(2)

$\textbf{at}(r, S7, t_{b,r}) \wedge \textbf{t\_detect}(r, t_{d,r}) \wedge t_{b,r} < t_{d,r}$ [(1),LHS]

$\Rightarrow \textbf{at}(r, S7, t_{b,r}) \wedge \neg\,\textbf{busy}(r, t_{d,r} - 1) \wedge \textbf{busy}(r, t_{d,r}) \wedge t_{b,r} < t_{d,r}$ [D6.6]

$\Rightarrow \textbf{at}(r, S13, t_{d,r})$ [SI4]

(3)

$\textbf{at}(r, S4, t_{b,r}) \wedge \textbf{t\_detect}(r, t_{b,r}) \wedge t_{b,r} < t_{d,r} \wedge \textbf{tx\_begin}(p, t_{b,p} + 1)$ [(1),LHS]

$\Rightarrow \textbf{tx\_begin}(r, t_{d,r}) \wedge \textbf{in}(r, S4, t_{d,r}) \wedge \textbf{busy}(r, t_{b,p} + 1 + \Delta_{pr})$ [D6.6,ThJ1]

$\Rightarrow \textbf{in}(r, S4, t_{d,r}) \wedge \textbf{coll\_begin}(r, t_{b,p} + 1 + \Delta_{pr})$ [D6.4,D6.8]

$\Rightarrow \textbf{at}(r, S5, t_{b,p} + 1 + \Delta_{pr} + jam + 1)$ [SI2]

$\wedge\ \textbf{label}(r, t_{b,p} + 1 + \Delta_{pr} + jam + 1) = \text{coll\_detected}$

$\Rightarrow \textbf{at}(r, S35, t_{d,r} + 2D + jam + 1)$ [SI5,SI6]

(4)

$\textbf{t\_detect}(r, t_{d,r})$ [LHS]

$\Rightarrow \neg\,\textbf{busy}(r, t_{d,r} - 1) \wedge \textbf{busy}(r, t_{d,r})$ [D6.6]

$\Rightarrow [\forall q:\ q \neq r:\ \neg\,\textbf{transmitting}(q, t_{d,r} - 1 - \Delta_{qr})]$ [ThJ1]

$\Rightarrow [\forall q:\ q \neq r:\ \textbf{tx\_begin}(q, t_{b,q} + 1) \Rightarrow t_{b,q} + 1 \geq t_{d,r} - \Delta_{qr}]$ [D6.2]

(5)

$\textbf{at}(p, S2, t_{b,p}) \wedge \textbf{tx\_begin}(p, t_{b,p} + 1)$ [LHS]

$\Rightarrow \textbf{at}(p, S4, t_{b,p})$ [SI1]

(6)

$\mathbf{at}(p, S4, t_{b,p}) \wedge \mathbf{tx\_begin}(p, t_{b,p}+1) \wedge \mathbf{coll\_begin}(p, t_{c,p})$ [LHS,(5)]

$\Rightarrow \mathbf{at}(p, S5, t_{c,p}+jam+1) \wedge \mathrm{label}(p, t_{c,p}+jam+1) = \mathrm{coll\_detected}$ [SI2]

$\Rightarrow [\forall t': t_{c,p}+jam+1 < t' \le t_{b,p}+1+2D+jam+1:$ [SI3]

$\mathbf{in}(p, S28, t') \wedge \neg\, \mathbf{transmitting}(p, t')]$

$\Rightarrow [\forall t': t_{c,p}+jam+1 < t' \le t_{d,r}-\Delta_{pr}+2D+jam+1:$ [(4)]

$\mathbf{in}(p, S28, t') \wedge \neg\, \mathbf{transmitting}(p, t')]$

(7)

$\mathbf{tx\_begin}(p, t_{b,p}+1) \wedge [\forall u: u \ne p: \mathbf{synch}(p, t_{b,p}, u, t_{b,u}, S2)]$ [LHS]

$\Rightarrow \mathbf{tx\_begin}(p, t_{b,p}+1)$ [SI1]

$\wedge [\forall u: u \ne p: ((\neg\, \mathbf{enabled}(u, t_{b,u}) \Rightarrow \mathbf{at}(u, S7, t_{b,u}))$

$\vee (\mathbf{enabled}(u, t_{b,u}) \Rightarrow \mathbf{at}(u, S4, t_{b,u})))$

$\wedge \mathbf{busy}(u, t_{b,p}+\Delta_{pu})]$ [ThJ1]

$\Rightarrow [\forall u: u \ne p: \mathbf{tx\_begin}(u, t_{b,u}+1)$ [D6.2]

$\Rightarrow t_{b,u}+1 \le t_{b,p}+1+\Delta_{pu} \wedge \mathbf{busy}(u, t_{b,p}+1+\Delta_{pu})]$

$\Rightarrow [\forall u: u \ne p: \mathbf{tx\_begin}(u, t_{b,u})$ [D6.4,SI2]

$\Rightarrow [\exists\, t_{c,u}: t_{c,u} \le t_{b,p}+1+\Delta_{pu}: \mathbf{coll\_begin}(u, t_{c,u})$

$\wedge \mathbf{at}(u, S5, t_{c,u}+jam+1)]]$

$\Rightarrow [\forall u: u \ne p: \mathbf{tx\_begin}(u, t_{b,u})$ [SI3]

$\Rightarrow [\exists\, t_{c,u}: t_{c,u} \le t_{b,p}+1+\Delta_{pu}:$

$[\forall t': t_{c,u}+jam+1 < t' \le t_{b,u}+2D+jam+1:$

$\mathbf{in}(u, S28, t') \wedge \neg\, \mathbf{transmitting}(u, t')]]]$

$\Rightarrow [\forall u:\ u \neq p:\ \textbf{tx\_begin}(u, t_{b,u})$ [(4)]

$\Rightarrow [\forall t':\ t_{b,p} + \Delta_{pu} + jam + 1 < t' \le t_{d,r} - \Delta_{u,r} + 2D + jam + 1:$

$\textbf{in}(u, S28, t') \wedge \neg\, \textbf{transmitting}(u, t')]]]$

(8)

$[\forall u:\ u \neq p:\ \textbf{synch}(p, t_{b,p}, u, t_{b,u}, S2)]$ [LHS]

$\Rightarrow [\forall u:\ u \neq p:\ (\textbf{enabled}(u, t_{b,u}) \vee \neg\, \textbf{enabled}(u, t_{b,u}))]$ [SI1]

(9)

$[\forall u:\ u \neq p:\ \textbf{enabled}(u, t_{b,u})$

$\Rightarrow [\forall t':\ t_{b,p} + 1 + \Delta_{pu} + jam + 1 < t' \le t_{d,r} - \Delta_{ur} + 2D + jam + 1:$ [(7)]

$\textbf{in}(u, S28, t') \wedge \neg\, \textbf{transmitting}(u, t')]]$

(10)

$[\forall u:\ u \neq p:\ \neg\, \textbf{enabled}(u, t_{b,u}) \wedge \textbf{tx\_begin}(p, t_{b,p} + 1)$

$\Rightarrow [\exists\, t_{d,u}:\ t_{d,u} \le t_{b,p} + 1 + \Delta_{pu}:\ \textbf{t\_detect}(u, t_{d,u})]$ [SI4,SI5,SI6]

$\wedge\ [\forall t':\ t_{b,u} \le t' \le t_{d,u} + 2D + jam + 1:$

$\neg\, \textbf{transmitting}(u, t')]]$

$\Rightarrow [\forall t':\ t_{b,u} \le t' \le t_{d,r} - \Delta_{ur} + 2D + jam + 1:$ [(4)]

$\neg\, \textbf{transmitting}(u, t')]$

(11)

$[\forall u:\ u \neq r:\ \neg\, \textbf{transmitting}(u, t_{d,r} - \Delta_{ur} + 2D + jam + 1)]$ [(6),(9),(10)]

$\Rightarrow \neg\, \textbf{busy}(r, t_{d,r} + 2D + jam + 1)$ [ThJ1]

(12)

$$\textbf{at}(r, S13, t_{d,r}) \wedge \neg\, \textbf{busy}(r, t_{d,r} + 2D + jam + 1) \qquad [(2),(11)]$$

$$\Rightarrow \textbf{at}(r, S13, t_{d,r})$$

$$\wedge\ [\exists\, t_{i,r}\colon\ t_{d,r} < t_{i,r} \leq t_{d,r} + 2D + jam + 1\colon\ \textbf{after}(r, S13, t_{i,r})]$$

$$\wedge\ \neg\, \textbf{busy}(r, t_{d,r} + 2D + jam + 1)$$

$$\Rightarrow \textbf{at}(r, S35, t_{d,r} + 2D + jam + 1) \qquad [\text{SI5,SI6}]$$

The result follows from (3), (11), and (12).

□

*Theorem 6.9:*

$$\textbf{synch}(p, t_p, q, t_q, S2)$$

$$\wedge\ \textbf{quiescent}(p, t_p) \wedge \textbf{quiescent}(q, t_q)$$

$$\wedge\ \textbf{at}(p, S35, t_p + 2D + jam + 2) \wedge \text{label}(p, t_p + 2D + jam + 2) = \text{coll\_detected})$$

$$\Rightarrow [\exists\, t_{e,q}\colon\colon\ \textbf{synch}(p, t_p + 2D + jam + 2, q, t_{e,q}, S35)$$

$$\wedge\ (\text{label}(q, t_{e,q}) = \text{other\_coll} \vee \text{label}(q, t_{e,q}) = \text{coll\_detected}$$

$$\wedge\ \textbf{quiescent}(p, t_p + 2D + jam + 2) \wedge \textbf{quiescent}(q, t_{e,q})]$$

*Proof:*

(1)

$$\textbf{at}(p, S2, t_p) \wedge \textbf{quiescent}(p, t_p) \qquad [\text{LHS}]$$

$$\wedge\ \textbf{at}(p, S35, t_p + 2D + jam + 2) \wedge \text{label}(p, t_p + 2D + jam + 2) = \text{coll\_detected}$$

$$\Rightarrow \textbf{at}(p, S2, t_p) \wedge \textbf{enabled}(p, t_p) \wedge \neg\, \textbf{busy}(p, t_p) \qquad [\text{D6.10}]$$

$$\wedge\ [\exists\, t_{c,p}\colon\ t_p + 1 \leq t_{c,p} \leq t_p + 2D + 1\colon\ \neg\, \textbf{busy}(p, t_{c,p} - 1) \wedge \textbf{busy}(p, t_{c,p})]$$

$$\Rightarrow \textbf{at}(p, S4, t_p) \wedge \neg\, \textbf{busy}(p, t_p) \qquad [\text{SI1}]$$

$\wedge\ [\exists\, t_{c,p}\colon\ t_p + 1 \le t_{c,p} \le t_p + 2D + 1\colon\ \neg\,\textbf{busy}(p, t_{c,p} - 1)\ \wedge\ \textbf{busy}(p, t_{c,p})]$

$\Rightarrow\ \textbf{tx\_begin}(p, t_p + 1)$ [D6.2]

$\wedge\ [\exists\, t_{c,p}\colon\ t_p + 1 \le t_{c,p} \le t_p + 2D + 1\colon\ \textbf{coll\_begin}(p, t_{c,p})$ [D6.4

$[ThJ1]\ \wedge\ [\exists\, r\colon\ r \ne p\colon\ \textbf{tx\_begin}(r, t_{c,p} - \Delta_{rp})]]$

(2)

$\textbf{tx\_begin}(p, t_p + 1)\ \wedge\ \textbf{at}(q, S2, t_q)$ [LHS,(1),D6.8]

$\Rightarrow\ \textbf{t\_detect}(q, t_{d,q})\ \wedge\ t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}$ [D6.6]

(3)

$\textbf{synch}(p, t_p, q, t_q, S2)\ \wedge\ \textbf{t\_detect}(q, t_{d,q})\ \wedge\ t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}$ [LHS,(1),(2)]

$\wedge\ \textbf{tx\_begin}(p, t_p + 1)\ \wedge\ [\exists\, t_{c,p}\colon\ t_p + 1 \le t_{c,p} \le t_p + 2D + 1\colon\ \textbf{coll\_begin}(p, t_{c,p})]$

$\Rightarrow\ \neg\,\textbf{busy}(q, t_{d,q} + 2D + jam + 1)\ \wedge\ \textbf{at}(q, S35, t_{d,q} + 2D + jam + 1)$ [L6.3]

$\Rightarrow\ \textbf{at}(q, S35, t_{d,q} + 2D + jam + 1)$

$\wedge\ \neg\,\textbf{busy}(q, t_{d,q} + 2D + jam + 1)$

$\wedge\ \ \wedge\ \neg\,\textbf{transmitting}(q, t_{d,q} + 2D + jam + 1)$

(4)

$\textbf{at}(q, S2, t_q)$ [LHS,D6.8]

$\Rightarrow\ ((\neg\,\textbf{enabled}(q, t_q)\ \Rightarrow\ \textbf{at}(q, S7, t_q))$ [SI1]

$\wedge\ (\textbf{enabled}(q, t_q)\ \Rightarrow\ \textbf{at}(q, S4, t_q)))$

(5)

$$\textbf{quiescent}(q, t_q) \wedge \textbf{at}(q, S4, t_q) \wedge \textbf{tx\_begin}(p, t_p + 1) \qquad \text{[LHS,(1),(4)]}$$

$$\Rightarrow \textbf{in}(q, S4, t_q + 1) \wedge \textbf{tx\_begin}(q, t_q + 1) \wedge \textbf{busy}(q, t_p + 1 + \Delta_{pq}) \qquad \text{[ThJ1]}$$

$$\Rightarrow [\exists\, t_{c,q}\colon t_q + 1 \le t_{c,q} \le t_p + 1 + \Delta_{pq}\colon \qquad \text{[D6.4]}$$

$$\textbf{in}(q, S4, t_{c,q}) \wedge \textbf{coll\_begin}(q, t_{c,q})]$$

$$\Rightarrow [\exists\, t_{c,q}\colon t_q + 1 \le t_{c,q} \le t_p + 1 + \Delta_{pq}\colon \textbf{at}(q, s5, t_{c,q} + jam + 1) \qquad \text{[SI2]}$$

$$\wedge\ \text{label}(q, t_{c,q} + jam + 1) = \text{coll\_detected}]$$

$$\Rightarrow [\exists\, t_{c,q}\colon t_q + 1 \le t_{c,q} \le t_p + 1 + \Delta_{pq}\colon \qquad \text{[SI3]}$$

$$\textbf{at}(q, S35, t_q + 2D + 2D + jam + 2)$$

$$\wedge\ \text{label}(q, t_{c,q} + jam + 1) = \text{coll\_detected}]$$

(6)

$$\textbf{at}(q, S7, t_q) \wedge \textbf{tx\_begin}(p, t_p + 1) \qquad \text{[(1),(4)]}$$

$$\Rightarrow [\exists\, t_{d,q}\colon t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon \qquad \text{[D6.6]}$$

$$\textbf{at}(q, S7, t_q) \wedge \textbf{t\_detect}(q, t_{d,q}) \wedge t_{d,q} \le t_q + 2D]$$

$$\Rightarrow [\exists\, t_{d,q}\colon t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon \textbf{at}(q, S13, t_{d,q})] \qquad \text{[SI4]}$$

(7)

$$[\exists\, t_{d,q}\colon t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon \qquad \text{[(3),(6)]}$$

$$\textbf{at}(q, S13, t_{d,q}) \wedge \neg\, \textbf{busy}(q, t_{d,q} + 2D + jam + 1)]$$

$$\Rightarrow [\exists\, t_{d,q}\colon t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon \textbf{at}(q, S13, t_{d,q})$$

$$\wedge\ [\exists\, t_{i,q}\colon t_{d,q} < t_{i,q} \le t_{d,q} + 2D + jam + 1\colon\ \wedge\ \textbf{after}(q, S13, t_{i,q})]]$$

$$\Rightarrow [\exists\, t_{d,q}\colon t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon \qquad \text{[SI5]}$$

$\text{label}(q, t_{d,q} + 2D + jam + 1) = \text{other\_coll}$

$\wedge\ [\exists\, t_{i,q}\colon\ t_{d,q} < t_{i,q} \le t_{d,q} + 2D + jam + 1\colon\ \textbf{at}(q, S30, t_{i,q})]]$

$\Rightarrow [\exists\, t_{d,q}\colon\ t_q < t_{d,q} \le t_p + 1 + \Delta_{pq}\colon$ [SI6]

$\text{label}(q, t_{d,q} + 2D + jam + 1) = \text{other\_coll}$

$\wedge\ \textbf{at}(q, S35, t_{d,q} + 2D + jam + 1)]]$

The result follows from (3), (5), and (7).

□

## Right Interval: Collision, Observing Station

*Theorem 6.10:*

$\textbf{synch}(p, t_p, q, t_q, S2)$

$\wedge\ \textbf{quiescent}(p, t_p)\ \wedge\ \textbf{quiescent}(q, t_q)$

$\wedge\ [\exists\, t_{e,p}\colon\ t_p < t_{e,p} \le t_p + 4D + jam + 1\colon$

$\textbf{at}(p, S35, t_{e,p})\ \wedge\ \text{label}(p, t_{e,p}) = \text{other\_coll}]$

$\Rightarrow [\exists\, t_{e,p}\colon\ t_p < t_{e,p} \le t_p + 4D + jam + 1\colon$

$[\exists\, t_{e,q}{::}\ \textbf{synch}(p, t_{e,p}, q, t_{e,q}, S35)$

$\wedge\ (\text{label}(q, t_{e,q}) = \text{other\_coll}\ \vee\ \text{label}(q, t_{e,q}) = \text{coll\_detected})$

$\wedge\ \textbf{quiescent}(p, t_{e,p})\ \wedge\ \textbf{quiescent}(q, t_{e,q})]]$

*Proof:*

(1)

$\textbf{at}(p, S2, t_p)\ \wedge\ \textbf{quiescent}(p, t_p)$ [LHS,D6.8]

$\wedge\ [\exists\, t_{e,p}\colon\ t_p < t_{e,p} \le t_p + 4D + jam + 1\colon$

$\textbf{at}(p, S35, t_{e,p})\ \wedge\ \text{label}(p, t_{e,p}) = \text{other\_coll}]$

$\Rightarrow$ $\textbf{at}(p, S7, t_p) \wedge \neg\, \textbf{busy}(p, t_p)$ [SI1,D6.10]

$\wedge\ [\exists\, t_{e,p}:\ t_p < t_{e,p} \le t_p + 4D + jam + 1:\ \text{label}(p, t_{e,p}) = \text{other\_coll}]$

$\Rightarrow$ $[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$ [D6.8]

$[\forall t':\ t_p \le t' < t_{b,p}:\ \neg\, \textbf{busy}(p, t') \wedge \textbf{in}(p, S7, t')]$

$\wedge\ \textbf{busy}(p, t_{b,p})]$

$\Rightarrow$ $[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ \textbf{t\_detect}(p, t_{b,p})$ [SI4,ThJ1,D6.6]

$\wedge\ \textbf{at}(p, S13, t_{b,p}) \wedge [\exists\, s:\ s \ne p:\ \textbf{tx\_begin}(s, t_{b,p} - \Delta_{ps})]]$

(2)

$[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ \textbf{at}(p, S13, t_{b,p})]$ [LHS,(1)]

$\wedge\ [\exists\, t_{e,p}:\ t_p < t_{e,p} \le t_p + 4D + jam + 1:$

$\textbf{at}(p, S35, t_{e,p}) \wedge \text{label}(p, t_{e,p}) = \text{other\_coll}]$

$\Rightarrow$ $[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$

$[\exists\, t_{i,p}:\ t_{b,p} < t_{i,p} \le t_{b,p} + 2D + jam + 1:$

$\neg\, \textbf{busy}(p, t_{i,p}) \wedge \textbf{after}(p, S13, t_{i,p})]$

(3)

$[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$ [(2)]

$[\exists\, t_{i,p}:\ t_{b,p} < t_{i,p} \le t_{b,p} + 2D + jam + 1:\ \neg\, \textbf{busy}(p, t_{i,p})]]$

$\Rightarrow$ $[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$ [ThJ1]

$[\exists\, t_{i,p}:\ t_{b,p} < t_{i,p} \le t_{b,p} + 2D + jam + 1:$

$[\forall u:\ u \ne p:\ \neg\, \textbf{transmitting}(u, t_{i,p} - \Delta_{pu})]]]$

(4)

$[\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ \exists\, s:\ s \ne p:\ \textbf{tx\_begin}(s, t_{b,p} - \Delta_{ps})]$ [(1),(3)]

$\wedge\ [\exists\, t_{i,p}:\ t_{b,p} < t_{i,p} \le t_{b,p} + 2D + jam + 1:$

$[\forall u:\ u \ne p:\ \neg\, \textbf{transmitting}(u, t_{i,p} - \Delta_{pu})]]]$

$\Rightarrow\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ [\exists\, s:\ s \ne p:\ \textbf{tx\_begin}(s, t_{b,p} - \Delta_{ps})$

$[\exists\, t_{i,s}:\ t_{i,s} \le t_{b,p} + 2D + jam + 1 - \Delta_{ps}:$

$[\forall t':\ t_{b,p} - \Delta_{ps} \le t' \le t_{i,s}:\ \textbf{transmitting}(s, t')]$

$\wedge\ \neg\, \textbf{transmitting}(s, t_{i,s})\ \wedge\ t_{i,s} - (t_{b,p} - \Delta_{ps}) \le min\_size\,]]]]$

$\Rightarrow\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ [\exists\, s:\ s \ne p:$ [I3,A6.7]

$[\exists\, t_{c,s}:\ t_{c,s} \le t_{b,p} + 2D - \Delta_{ps}:\ \textbf{coll\_begin}(s, t_{c,s})]]]]$

(5)

$\textbf{at}(q, S2, t_q)$ [LHS,D6.8]

$\Rightarrow\ ((\neg\, \textbf{enabled}(q, t_q)\ \Rightarrow\ \textbf{at}(q, S7, t_q))$ [SI1]

$\wedge\ (\textbf{enabled}(q, t_q)\ \Rightarrow\ \textbf{at}(q, S4, t_q)))$

(6)

$\textbf{at}(q, S4, t_q)\ \wedge\ \neg\, \textbf{busy}(q, t_q)$ [LHS,D6.10,(5)]

$\Rightarrow\ \textbf{tx\_begin}(q, t_q + 1)$ [D6.2]

$\equiv\ \textbf{t\_detect}(q, t_q + 1)$ [D6.6]

(7)

$\textbf{at}(q, S7, t_q)$ [(1),(3),(5)]

$\wedge\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ [\exists\, r:\ r \ne p:\ \textbf{tx\_begin}(r, t_{b,p} - \Delta_{pr})]$

$\Rightarrow [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ [\exists\, r:\ r \ne p:$ [SI4]

$[\exists\, t_{d,q}:\ t_{d,q} \le t_{b,p} - \Delta_{p,r} + \Delta_{rq}:$

$\textbf{t\_detect}(q, t_{b,p} - \Delta_{pr} + \Delta_{rq}) \wedge \textbf{at}(q, S13, t_{b,p} - \Delta_{pr} + \Delta_{rq})]]]$ [D6.6]

(8)

$[\exists\, s:\ s \ne p:\ \textbf{synch}(s, t_{b,s}, q, t_{b,q}, S2)$ [LHS]

$\wedge\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$

$[\exists\, t_{d,q}:\ t_{d,q} = t_q + 1 \vee t_{d,q} \le t_{b,p} - \Delta_{ps} + \Delta_{sq}:\ \textbf{t\_detect}(q, t_{d,q})$ [(6),(7)]

$\wedge\ \textbf{tx\_begin}(s, t_{b,p} - \Delta_{ps}) \wedge \textbf{coll\_begin}(s, t_{c,s}) \wedge t_{b,q} \le t_{d,q} \wedge t_{b,s} \le t_{c,s}]]]$ [(1),(4)]

$\Rightarrow [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ [\exists\, s:\ s \ne p:$ [L6.3]

$[\exists\, t_{d,q}:\ t_{d,q} \le t_{b,p} - \Delta_{ps} + \Delta_{sq}:$

$\neg\,\textbf{busy}(q, t_{d,q} + 2D + jam + 1) \wedge \textbf{at}(q, S35, t_{d,q} + 2D + jam + 1)]]]$

(9)

$[\exists\, s:\ s \ne p:\ \textbf{synch}(s, t_{b,s}, p, t_p, S2)$ [LHS]

$\wedge\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ \textbf{t\_detect}(p, t_{b,p})$ [(1)]

$\wedge\ \textbf{tx\_begin}(s, t_{b,p} - \Delta_{ps}) \wedge \textbf{coll\_begin}(s, t_{c,s}) \wedge t_{b,s} \le t_{c,s}]]]$ [(1),(4)]

$\Rightarrow [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:$ [L6.3]

$\neg\,\textbf{busy}(p, t_{b,p} + 2D + jam + 1) \wedge \textbf{at}(p, S35, t_{b,p} + 2D + jam + 1)]]]$

(10)

$\textbf{at}(q, S4, t_q)$ [(1),(3),(5)]

$\wedge\ [\exists\, t_{b,p}:\ t_p < t_{b,p} \le t_p + 2D:\ \textbf{busy}(p, t_{b,p})$

$[\exists\, t_{i,p}:\ t_{b,p} < t_{i,p} \le t_{b,p} + 2D + jam + 1:$

$[\forall s\colon s \neq p\colon \neg\,\textbf{transmitting}(s, t_{i,p} - \Delta_{ps})]]]$

$\Rightarrow \textbf{tx\_begin}(q, t_q + 1)$

$\quad \wedge [\exists t_{b,p}\colon t_p < t_{b,p} \leq t_p + 2D\colon t_q + 1 \geq t_{b,p} - \Delta_{pq}$

$\quad\quad \wedge [\exists t_{i,p}\colon t_{b,p} < t_{i,p} \leq t_{b,p} + 2D + jam + 1\colon$

$\quad\quad\quad \neg\,\textbf{transmitting}(q, t_{i,p} - \Delta_{pq})]]$

$\Rightarrow \textbf{tx\_begin}(q, t_q + 1)$

$\quad \wedge [\exists t_{b,p}\colon t_p < t_{b,p} \leq t_p + 2D\colon$

$\quad\quad [\exists t_{i,p}\colon t_{b,p} < t_{i,p} \leq t_{b,p} + 2D + jam + 1\colon$

$\quad\quad\quad t_{i,p} - \Delta_{pq} - (t_{b,p} - \Delta_{pq}) \leq 2D + jam + 1$

$\quad\quad\quad [\exists t'\colon t' \leq t_{i,p} - \Delta_{pq}\colon \textbf{at}(q, S5, t') \wedge$

$\quad\quad\quad\quad \text{label}(q, t') = \text{coll\_detected}]]$

$\Rightarrow \textbf{at}(q, S35, t_q + 2D + jam + 2)$ [SI3]

$\quad \wedge \neg\,\textbf{transmitting}(q, t_q + 2D + jam + 2)$

$\quad \wedge \text{label}(q, t_q + 2D + jam + 2) = \text{coll\_detected}$

(11)

$[\exists t_{b,p}\colon t_p < t_{b,p} \leq t_p + 2D\colon [\exists s\colon s \neq p\colon$ [(7),(8)]

$\quad [\exists t_{d,q}\colon t_{d,q} \leq t_{b,p} - \Delta_{ps} + \Delta_{sq}\colon$

$\quad\quad \textbf{at}(q, S13, t_{d,q}) \wedge \neg\,\textbf{busy}(q, t_{d,q} + 2D + jam + 1)$

$\quad\quad\quad \wedge\ \textbf{at}(q, S35, t_{d,q} + 2D + jam + 1)]]]$

$\quad\quad\quad\quad \Rightarrow [\exists t_{b,p}\colon t_p < t_{b,p} \leq t_p + 2D\colon [\exists s\colon s \neq p\colon$

$\quad\quad\quad\quad\quad [\exists t_{d,q}\colon t_{d,q} \leq t_{b,p} - \Delta_{ps} + \Delta_{sq}\colon$

$\quad\quad\quad\quad\quad\quad [\exists t_{i,q}\colon t_{i,q} \leq t_{d,q} + 2D + jam + 1\colon$

$$\mathbf{at}(q, S30, t_{i,q}) \wedge \neg\,\mathbf{busy}(q, t_{d,q} + 2D + jam + 1)$$

$$\wedge\ \text{label}(q, t_{i,q}) = \text{other\_coll}$$

$$\wedge\ \mathbf{at}(q, S35, t_{d,q} + 2D + jam + 1)]]]]$$

$$\Rightarrow [\exists\, t_{b,p}\colon t_p < t_{b,p} \le t_p + 2D\colon [\exists\, s\colon s \ne p\colon$$

$$[\exists\, t_{d,q}\colon t_{d,q} \le t_{b,p} - \Delta_{ps} + \Delta_{sq}\colon$$

$$\mathbf{at}(q, S35, t_{d,q} + 2D + jam + 1)$$

$$\wedge\ \text{label}(q, t_{d,q} + 2D + jam + 1) = \text{other\_coll}$$

$$\wedge\ \neg\,\mathbf{busy}(q, t_{d,q} + 2D + jam + 1)]]]$$

(12)

$$[\exists\, t_{b,p}\colon t_p < t_{b,p} \le t_p + 2D\colon \mathbf{at}(p, S35, t_{b,p} + 2D + jam + 1) \qquad [(9)]$$

$$\wedge\ [\exists\, s\colon s \ne p\colon [\exists\, t_{d,q}\colon t_{d,q} \le t_{b,p} - \Delta_{ps} + \Delta_{sq}\colon$$

$$\mathbf{at}(q, S35, t_{d,q} + 2D + jam + 1) \wedge \neg\,\mathbf{busy}(q, t_{d,q} + 2D + jam + 1) \qquad [(8)]$$

$$\wedge\ (\text{label}(q, t_{d,q} + 2D + jam + 1) = \text{other\_coll} \qquad [(10),(11)]$$

$$\vee\ \text{label}(q, t_{d,q} + 2D + jam + 1) = \text{coll\_detected})$$

$$\wedge\ \neg\,\mathbf{transmitting}(q, t_{d,q} + 2D + jam + 1)]]]$$

$$\Rightarrow [\exists\, t_{e,p}\colon t_p < t_{e,p} \le t_p + 4D + jam + 1\colon \qquad [\text{D6.8,D6.10}]$$

$$[\exists\, t_{e,q}\colon\colon \mathbf{synch}(p, t_{e,p}, q, t_{e,q}, S35)$$

$$\wedge\ (\text{label}(q, t_{e,q}) = \text{other\_coll} \vee \text{label}(q, t_{e,q}) = \text{coll\_detected})$$

$$\wedge\ \mathbf{quiescent}(p, t_{e,p}) \wedge \mathbf{quiescent}(q, t_{e,q})]]$$

□

Theorems 6.6 through 6.10 show that a single iteration of the right interval search takes bounded time. Invariants corresponding to SI1 - SI6 and results corresponding to Theorems 6.6 - 6.10 exists for the left interval search. Because the arguments are very similar, these theorems are stated without proof.

SI 7: $\mathbf{at}(p, S35, t)$

$\Rightarrow (\mathbf{enabled}(p, t) \Rightarrow \mathbf{at}(p, S37, t)) \wedge (\neg\, \mathbf{enabled}(p, t) \Rightarrow \mathbf{at}(p, S40, t))$

SI 8: $\mathbf{at}(p, S37, t_1)$

$\Rightarrow [\exists\, t_2\colon t_1 + jam < t_2\colon \mathbf{at}(p, S38, t_2) \wedge \neg\, \mathbf{transmitting}(p, t_2)$

$\wedge\ (\text{label}(p, t_2) = \text{packet\_sent}$

$\Rightarrow \mathbf{at}(p, S2, t_2) \wedge [\exists\, T\colon min\_size \le T < max\_size\colon t_2 = t_1 + T + 1$

$\wedge\ [\forall t'\colon t_2 - T \le t' < t_2\colon \mathbf{transmitting}(p, t') \wedge \neg\, \mathbf{busy}(p, t')]])$

$\wedge\ (\text{label}(p, t_2) = \text{coll\_detected}$

$\Rightarrow t_2 \le t_1 + 2D + jam + 2 \wedge \mathbf{coll\_begin}(p, t_2 - jam - 1)$

$\wedge\ \mathbf{at}(p, S38, t_2))$

$\wedge\ [\forall t''\colon t_2 - jam - 1 \le t'' < t_2\colon \mathbf{transmitting}(p, t'')]]$

SI 9: $\mathbf{at}(p, S38, t) \wedge \text{label}(p, t) = \text{coll\_detected}$

$\Rightarrow \mathbf{coll\_begin}(p, t - jam - 1)$

$\wedge\ [\exists\, t_1\colon t_1 \le t - jam - 1\colon \mathbf{tx\_start}(p, t_1)$

$\wedge\ [\forall t'\colon t_1 \le t' < t\colon \mathbf{transmitting}(p, t')]$

$\wedge\ [\exists\, t''\colon t'' \le t_1 + 2D + jam + 1\colon \mathbf{at}(p, S35, t'')]]$

SI 10: $\mathbf{at}(p, S40, t_1)$

$\Rightarrow [\exists\, t_2\colon t_2 > t_1\colon \mathbf{after}(p, S40, t_2)$

$\wedge\ ((t_2 = t_1 + 2D \ \wedge\ [\forall t' :\ t_1 < t' \le t_2 :\ \neg\, \mathbf{busy}(p, t')])$

$\Rightarrow\ \mathbf{at}(p, S35, t_1 + 2D))$

$\wedge\ ((t_2 \le t_1 + 2D \ \wedge\ \mathbf{busy}(p, t_2)\ \wedge\ [\forall t'' :\ t_1 < t'' < t_2 :\ \neg\, \mathbf{busy}(p, t'')]$

$\Rightarrow\ \mathbf{at}(p, S46, t_2))]$

**SI 11:** $\mathbf{at}(p, S46, t_1)\ \wedge\ \mathbf{after}(p, S46, t_2)$

$\Rightarrow\ [\forall t' :\ t_1 < t' < t_2 :\ \mathbf{busy}(p, t')]\ \wedge\ \neg\, \mathbf{busy}(p, t_2)$

$\wedge\ (t_2 - t_1 \le \mathit{min\_size}\ \Rightarrow\ \text{label}(p, t_2) = \text{other\_coll}\ \wedge\ \mathbf{at}(p, S61, t_2))$

$\wedge\ (t_2 - t_1 > \mathit{min\_size}\ \Rightarrow\ \text{label}(p, t_2) = \text{other\_tx}\ \wedge\ \mathbf{at}(p, S2, t_2))$

**SI 12:** $\mathbf{at}(p, S61, t_2)$

$\Rightarrow\ \text{label}(p, t_2) = \text{other\_coll}\ \wedge\ \neg\, \mathbf{busy}(p, t_2)$

$\wedge\ [\exists\, t_1 :\ t_1 < t_2 :\ [\forall t' :\ t_1 < t' < t_2 :\ \mathbf{busy}(p, t')]\ \wedge\ \neg\, \mathbf{busy}(p, t_1)$

$\wedge\ [\exists\, t_3 :\ t_3 \le t_1 + 2D + \mathit{jam} + 1 :\ \mathbf{at}(p, S35, t_3)]]$

## Left Interval: Idle Interval

*Theorem 6.11:*

$\mathbf{synch}(p, t_p, q, t_q, S35)$

$\wedge\ \mathbf{quiescent}(p, t_p)\ \wedge\ \mathbf{quiescent}(q, t_q)$

$\wedge\ \mathbf{at}(p, S35, t_p + 2D)\ \wedge\ \text{label}(p, t_p + 2D) = \text{idle}$

$\Rightarrow\ [\exists\, t_{e,q} ::\ \mathbf{synch}(p, t_p + 2D, q, t_{e,q}, S35)\ \wedge\ \text{label}(q, t_{e,q}) = \text{idle}$

$\wedge\ \mathbf{quiescent}(p, t_p + 2D)\ \wedge\ \mathbf{quiescent}(q, t_{e,q})]$

**Left Interval: Successful Transmission, Transmitting Station**

*Theorem 6.12:*

$$\begin{aligned}
&\textbf{synch}(p, t_p, q, t_q, S35) \wedge \textbf{quiescent}(p, t_p) \wedge \textbf{quiescent}(q, t_q) \\
&\quad \wedge [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon \textbf{at}(p, S2, t_p + T + 1) \\
&\qquad \wedge \text{label}(p, t_p + T + 1) = \text{packet\_sent}] \\
&\qquad\quad \Rightarrow [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon \\
&\qquad\qquad \textbf{synch}(p, t_p + T + 1, q, t_p + T + 1 + \Delta_{pq}, S2) \\
&\qquad\qquad \wedge \text{label}(q, t_p + T + 1 + \Delta_{pq}) = \text{other\_tx} \\
&\qquad\qquad \wedge \textbf{quiescent}(p, t_p + T + 1) \wedge \textbf{quiescent}(q, t_p + T + 1 + \Delta_{pq})]
\end{aligned}$$

**Left Interval: Successful Transmission, Observing Station**

*Theorem 6.13:*

$$\begin{aligned}
&\textbf{synch}(p, t_p, q, t_q, S35) \\
&\quad \wedge \textbf{quiescent}(p, t_p) \wedge \textbf{quiescent}(q, t_q) \\
&\quad \wedge [\exists t_w\colon t_w \le 2D\colon [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon \\
&\qquad \textbf{at}(p, S2, t_p + t_w + T) \wedge \text{label}(p, t_p + t_w + T) = \text{other\_tx}]] \\
&\qquad\quad \Rightarrow [\exists t_w\colon t_w \le 2D\colon [\exists T\colon \mathit{min\_size} < T \le \mathit{max\_size}\colon [\exists t_{e,q}:: \\
&\qquad\qquad \textbf{synch}(p, t_p + t_w + T, q, t_{e,q}, S2) \\
&\qquad\qquad \wedge (\text{label}(q, t_{e,q}) = \text{other\_tx} \vee \text{label}(q, t_{e,q}) = \text{packet\_sent}) \\
&\qquad\qquad \wedge \textbf{quiescent}(p, t_p + t_w + T) \wedge \textbf{quiescent}(q, t_{e,q})]]]
\end{aligned}$$

## Left Interval: Collision, Participating Station

*Theorem 6.14:*

$$\mathbf{synch}(p, t_p, q, t_q, S35)$$
$$\land\ \mathbf{quiescent}(p, t_p) \land \mathbf{quiescent}(q, t_q)$$
$$\land\ \mathbf{at}(p, S35, t_p + 2D + jam + 2) \land \text{label}(p, t_p + 2D + jam + 2) = \text{coll\_detected}$$
$$\Rightarrow [\exists\, t_{e,q} :: \mathbf{synch}(p, t_p + 2D + jam + 2, q, t_{e,q}, S35)$$
$$\land\ (\text{label}(q, t_{e,q}) = \text{other\_coll} \lor \text{label}(q, t_{e,q}) = \text{coll\_detected}$$
$$\land\ \mathbf{quiescent}(p, t_p + 2D + jam + 2) \land \mathbf{quiescent}(q, t_{e,q})]$$

## Left Interval: Collision, Observing Station

*Theorem 6.15:*

$$\mathbf{synch}(p, t_p, q, t_q, S35)$$
$$\land\ \mathbf{quiescent}(p, t_p) \land \mathbf{quiescent}(q, t_q)$$
$$\land\ [\exists\, t_{e,p} :\ t_p < t_{e,p} \le t_p + 4D + jam + 1:$$
$$\mathbf{at}(p, S35, t_{e,p}) \land \text{label}(p, t_{e,p}) = \text{other\_coll}]$$
$$\Rightarrow [\exists\, t_{e,p} :\ t_p < t_{e,p} \le t_p + 4D + jam + 1:$$
$$[\exists\, t_{e,q} :: \mathbf{synch}(p, t_{e,p}, q, t_{e,q}, S35)$$
$$\land\ (\text{label}(q, t_{e,q}) = \text{other\_coll} \lor \text{label}(q, t_{e,q}) = \text{coll\_detected})$$
$$\land\ \mathbf{quiescent}(p, t_{e,p}) \land \mathbf{quiescent}(q, t_{e,q})]]$$

## Resolution Termination

Theorems 6.6 through 6.15 establish that the delay for an iteration of the left and right interval searches is bounded. The condition of the system when the loop terminates is now considered. The argument rests on the fact that each iteration of the loop either successfully transmits a packet at some station, in which case the window advances or the window size is halved. Since arrival times are distinct, this reduction in window size will result in a successful transmission within finite time.

Theorem 6.16 states the maximum amount of time between the invocation of the left interval search and the termination of the left interval search loop.

*Theorem 6.16:* (Left Interval Search Loop Termination)

$$\begin{aligned}
&\mathbf{at}(p, S35, t_p) \wedge \mathbf{quiescent}(p, t_p) \\
&\quad \Rightarrow [\exists\, n:\ 1 \le n \le log(\omega(p, t_p)):\ [\exists\, t_{e,p}:\ t_{e,p} \le t_p + n(4D + jam + 2): \\
&\quad\ \mathbf{at}(p, S2, t_{e,p}) \\
&\quad\quad \wedge (\text{label}(p, t_{e,p}) = \text{packet\_sent} \vee \text{label}(p, t_{e,p}) = \text{other\_tx}) \\
&\quad\quad \wedge (t_{e,p} - \upsilon(p, t_{e,p}) - \omega(p, t_{e,p})) \\
&\quad\quad\quad - (t_p - \upsilon(p, t_p) - \omega(p, t_p)) > 0 \\
&\quad\quad \wedge \omega(p, t_{e,p}) \le \omega(p, t_p)]]
\end{aligned}$$

*Proof:*

The proof proceeds by showing that each of the control flows through the left interval search enumerated by station program invariants SI7 through SI12 preserves the desired relationship between the initial and final values of the parameters $\upsilon$ and $\omega$. Termination follows from the assumption that no two arrival times are identical.

(1)

$$\begin{aligned}
&\mathbf{at}(p, S35, t_p) && \text{[LHS]} \\
&\quad \Rightarrow (\mathbf{enabled}(p, t_p) \Rightarrow \mathbf{at}(p, S37, t_p)) && \text{[SI7]} \\
&\quad\quad \wedge (\neg\, \mathbf{enabled}(p, t_p) \Rightarrow \mathbf{at}(p, S40, t_p))
\end{aligned}$$

(2)

$\mathbf{at}(p, S37, t_p)$ [(1)]

$\Rightarrow [\exists\, t_{i,p}:\ t_p + jam < t_{i,p}:\ \mathbf{at}(p, S38, t_{i,p}) \wedge \neg\, \mathbf{transmitting}(p, t_{i,p})$ [SI8]

$\wedge\ (\text{label}(p, t_{i,p}) = \text{packet\_sent}$

$\Rightarrow \mathbf{at}(p, S2, t_{i,p}) \wedge t_{i,p} = t_p + T + 1$

$\wedge\ [\forall t':\ t_{i,p} - T \le t' < t_{i,p}:\ \mathbf{transmitting}(p, t') \wedge \neg\, \mathbf{busy}(p, t')]$

$\wedge\ \upsilon(p, t_{i,p}) = \upsilon(p, t_p) + T + 1 - \omega(p, t_p)$

$\wedge\ \omega(p, t_{i,p}) = \omega(p, t_p))$

$\wedge\ (\text{label}(p, t_{i,p}) = \text{coll\_detected}$

$\Rightarrow t_{i,p} \le t_p + 2D + jam + 1 \wedge \mathbf{coll\_begin}(p, t_{i,p} - jam - 1)$

$\wedge\ \mathbf{at}(p, S38, t_{i,p}))$

$\wedge\ [\forall t'':\ t_{i,p} - jam - 1 \le t'' < t_{i,p}:\ \mathbf{transmitting}(p, t'')]]$

(3)

$\mathbf{at}(p, S38, t_{i,p}) \wedge \text{label}(p, t_{i,p}) = \text{coll\_detected}$ [(2)]

$\Rightarrow \mathbf{coll\_begin}(p, t_{i,p} - jam - 1)$ [SI9]

$\wedge\ [\exists\, t_p:\ t_p \le t_{i,p} - jam - 2:\ \mathbf{tx\_start}(p, t_p + 1)$

$\wedge\ [\forall t':\ t_p + 1 \le t' < t_{i,p}:\ \mathbf{transmitting}(p, t')]$

$\wedge\ [\exists\, t_{e,p}:\ t_{e,p} \le t_p + 2D + jam + 2:\ \mathbf{at}(p, S35, t_{e,p})$

$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + 2D + jam + 1 + \omega(p, t_p)/2$

$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p)/2]]$

(4)

$\textbf{at}(p, S40, t_p)$ [(1)]

$\Rightarrow [\exists\, t_2\colon t_2 > t_p\colon \textbf{after}(p, S40, t_2)$ [SI10]

$\wedge\ ((t_2 = t_p + 2D\ \wedge\ [\forall t'\colon t_p < t' \le t_2\colon \neg\,\textbf{busy}(p, t')])$

$\Rightarrow \textbf{at}(p, S35, t_p + 2D)\ \wedge\ t_{e,p} = t_p + 2D$

$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + 2D - \omega(p, t_p)/2$

$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p)/2)$

$\wedge\ ((t_2 \le t_p + 2D\ \wedge\ \textbf{busy}(p, t_2)\ \wedge\ [\forall t''\colon t_p < t'' < t_2\colon \neg\,\textbf{busy}(p, t'')]$

$\Rightarrow \textbf{at}(p, S46, t_2))]$

(5)

$\textbf{at}(p, S46, t_{b,p})\ \wedge\ \textbf{after}(p, S46, t_{i,p})$ [(4)]

$\Rightarrow [\forall t'\colon t_{b,p} < t' < t_{i,p}\colon \textbf{busy}(p, t')]\ \wedge\ \neg\,\textbf{busy}(p, t_{i,p})$ [SI11]

$\wedge\ (t_{i,p} - t_{b,p} \le min\_size\ \Rightarrow \text{label}(p, t_{i,p}) = \text{other\_coll}\ \wedge\ \textbf{at}(p, S61, t_{i,p}))$

$\wedge\ (t_{i,p} - t_{b,p} > min\_size\ \Rightarrow \text{label}(p, t_{i,p}) = \text{other\_tx}\ \wedge\ \textbf{at}(p, S2, t_{i,p})$

$\wedge\ \upsilon(p, t_{i,p}) = \upsilon(p, t_p) + (t_{i,p} - t_{b,p}) - \omega(p, t_p)$

$\wedge\ \omega(p, t_{i,p}) = \omega(p, t_p))]$

(6)

$\textbf{at}(p, S61, t_{i,p})$ [(5)]

$\Rightarrow \text{label}(p, t_{i,p}) = \text{other\_coll}\ \wedge\ \neg\,\textbf{busy}(p, t_{i,p})$ [SI12]

$\wedge\ [\exists\, t_{b,p}\colon t_{b,p} < t_{i,p}\colon [\forall t'\colon t_{b,p} < t' < t_{i,p}\colon \textbf{busy}(p, t')]\ \wedge\ \neg\,\textbf{busy}(p, t_{b,p})$

$\wedge\ [\exists\, t_{e,p}\colon t_{e,p} \ge t_{b,p} + 2D + jam + 1\colon \textbf{at}(p, S35, t_{e,p})$

$$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + (t_{i,p} - t_{b,p}) - \omega(p, t_p)/2$$

$$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p)/2]]$$

The left interval search ends when a successful transmission takes place as shown in (2) and (5). Any other iteration ends with $\omega$ being halved as indicated in (3), (4) and (6). In the case of an idle step, the window is also advanced and halved as shown in (2). It has been assumed that $\omega_0 = 2^n$ for some $n$. In the worst case, it is possible for the window to be reduced to size 1 without a successful transmission. Since $\omega$ is at most $\omega_0/2$ when the left interval search is invoked, within at most $n - 1$ iterations, $\omega = 1$. By assumption, a successful transmission is guaranteed during this iteration, thus ending the left interval search. Since each iteration in this case takes less than $4D + jam + 2$ time units (Theorem 6.10), the completion of the left interval search will take less than $(n - 1)(4D + jam + 2)$ time units.

$\square$

Note that the left interval search always results in the window being advanced; some portion of the arrival time line is eliminated, and a packet is successfully transmitted.

The remaining issue is the boundedness of the right interval search. Theorem 6.17 shows that a right interval search results in either a successful transmission, or a left interval search followed by a right interval search on a reduced window. The argument is similar to that presented for Theorem 6.16.

*Theorem 6.17:* (Left Interval Search Loop Termination)

$$\mathbf{at}(p, S2, t_p) \wedge \omega(p, t_p) = \omega_0$$

$$\Rightarrow [\exists\, n:\ 1 \le n \le log(\omega(p, t_p)):\ [\exists\, t_{e,p}:\ t_{e,p} \le t_p + n^2(4D + jam + 2):$$

$$\mathbf{at}(p, S2, t_{e,p}) \wedge \omega(p, t_{e,p}) = \omega_0$$

$$\wedge\ (t_{e,p} - \upsilon(p, t_{e,p}) - \omega(p, t_{e,p}))$$

$$- (t_p - \upsilon(p, t_p) - \omega(p, t_p)) > 0]]$$

*Proof:*

The proof proceeds by showing that each of the control flows through the right interval search enumerated by station program invariants SI1 through SI6 results either in a successful transmission or in the invocation of the left interval search which Theorem 6.16 shows to be bounded.

(1)

$\mathbf{at}(p, S2, t_p)$ [LHS]

$\Rightarrow (\mathbf{enabled}(p, t_p) \Rightarrow \mathbf{at}(p, S4, t_p))$ [SI1]

$\vee (\neg\, \mathbf{enabled}(p, t_p) \Rightarrow \mathbf{at}(p, S7, t_p))$

(2)

$\mathbf{at}(p, S4, t_p)$ [(1)]

$\Rightarrow [\exists\, t_{i,p}\colon t_p + jam < t_{i,p}\colon \mathbf{at}(p, S5, t_{i,p}) \wedge \neg\, \mathbf{transmitting}(p, t_{i,p})$ [SI2]

$\wedge\ (\text{label}(p, t_{i,p}) = \text{packet\_sent}$

$\Rightarrow \mathbf{at}(p, S2, t_{i,p}) \wedge t_{i,p} = t_p + T + 1$

$\wedge\ [\forall t'\colon t_{i,p} - T \le t' < t_{i,p}\colon \mathbf{transmitting}(p, t') \wedge \neg\, \mathbf{busy}(p, t')]$

$\wedge\ \upsilon(p, t_{i,p}) = \upsilon(p, t_p) + T + 1 - \omega(p, t_p)$

$\wedge\ \omega(p, t_{i,p}) = \omega(p, t_p))$

$\wedge\ (\text{label}(p, t_{i,p}) = \text{coll\_detected}$

$\Rightarrow t_{i,p} \le t_p + 2D + jam + 1 \wedge \mathbf{coll\_begin}(p, t_{i,p} - jam - 1)$

$\wedge\ \mathbf{at}(p, S5, t_{i,p})$

$\wedge\ [\forall t''\colon t_{i,p} - jam - 1 \le t'' < t_{i,p}\colon \mathbf{transmitting}(p, t'')])]$

(3)

$\mathbf{at}(p, S5, t_{i,p}) \wedge \text{label}(p, t_{i,p}) = \text{coll\_detected}$ [(2)]

$\Rightarrow \mathbf{coll\_begin}(p, t_{i,p} - jam - 1)$ [SI3]

$\wedge\ [\exists\, t_p\colon\ t_p \le t_{i,p} - jam - 2\colon\ \mathbf{tx\_start}(p, t_p + 1)$

$\wedge\ [\forall t'\colon\ t_p + 1 \le t' < t_{i,p}\colon\ \mathbf{transmitting}(p, t')]$

$\wedge\ [\exists\, t_{e,p}\colon\ t_{e,p} \le t_p + 2D + jam + 2\colon\ \mathbf{at}(p, S35, t_{e,p})$

$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + 2D + jam + 1 + \omega(p, t_p)/2$

$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p)/2]]$

(4)

$\mathbf{at}(p, S7, t_p)$ [(1)]

$\Rightarrow\ [\exists\, t_2\colon\ t_2 > t_p\colon\ \mathbf{after}(p, S7, t_2)$ [SI4]

$\wedge\ ((t_2 = t_p + 2D\ \wedge\ [\forall t'\colon\ t_p < t' \le t_2\colon\ \neg\,\mathbf{busy}(p, t')])$

$\Rightarrow\ \mathbf{at}(p, S2, t_p + 2D) \wedge t_{e,p} = t_p + 2D$

$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + 2D - \omega(p, t_p)$

$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p))$

$\wedge\ ((t_2 \le t_p + 2D\ \wedge\ \mathbf{busy}(p, t_2)\ \wedge\ [\forall t''\colon\ t_p < t'' < t_2\colon\ \neg\,\mathbf{busy}(p, t'')]$

$\Rightarrow\ \mathbf{at}(p, S13, t_2))]$

(5)

$\mathbf{at}(p, S13, t_{b,p}) \wedge \mathbf{after}(p, S13, t_{i,p})$ [(4)]

$\Rightarrow\ [\forall t'\colon\ t_{b,p} < t' < t_{i,p}\colon\ \mathbf{busy}(p, t')] \wedge \neg\,\mathbf{busy}(p, t_{i,p})$ [SI5]

$\wedge\ (t_{i,p} - t_{b,p} \le min\_size\ \Rightarrow\ \text{label}(p, t_{i,p}) = \text{other\_coll}\ \wedge\ \mathbf{at}(p, S30, t_{i,p}))$

$$\wedge\ (t_{i,p} - t_{b,p} > min\_size \Rightarrow \text{label}(p, t_{i,p}) = \text{other\_tx} \ \wedge\ \textbf{at}(p, S2, t_{i,p})$$

$$\wedge\ \upsilon(p, t_{i,p}) = \upsilon(p, t_p) + (t_{i,p} - t_{b,p}) - \omega(p, t_p)$$

$$\wedge\ \omega(p, t_{i,p}) = \omega(p, t_p))$$

(6)

$$\textbf{at}(p, S30, t_{i,p}) \qquad [(5)]$$

$$\Rightarrow \text{label}(p, t_{i,p}) = \text{other\_coll} \ \wedge\ \neg\, \textbf{busy}(p, t_{i,p}) \qquad [SI6]$$

$$\wedge\ [\exists\, t_{b,p}\colon\ t_{b,p} < t_{i,p}\colon\ [\forall t'\colon\ t_{b,p} < t' < t_{i,p}\colon\ \textbf{busy}(p, t')] \ \wedge\ \neg\, \textbf{busy}(p, t_{b,p})$$

$$\wedge\ [\exists\, t_{e,p}\colon\ t_{e,p} \geq t_{b,p} + 2D + jam + 1\colon\ \textbf{at}(p, S35, t_{e,p})$$

$$\wedge\ \upsilon(p, t_{e,p}) = \upsilon(p, t_p) + (t_{i,p} - t_{b,p}) - \omega(p, t_p)/2$$

$$\wedge\ \omega(p, t_{e,p}) = \omega(p, t_p)/2\,]]$$

(7)

$$\textbf{at}(p, S35, t_p)$$

$$\Rightarrow [\exists\, n\colon\ 1 \leq n \leq log(\omega(p, t_p))\colon\ [\exists\, t_{e,p}\colon\ t_{e,p} \leq t_p + n(4D + jam + 2)\colon$$

$$\textbf{at}(p, S2, t_{e,p})$$

$$\wedge\ (\text{label}(p, t_{e,p}) = \text{packet\_sent} \ \vee\ \text{label}(p, t_{e,p}) = \text{other\_tx})$$

$$\wedge\ (t_{e,p} - \upsilon(p, t_{e,p}) - \omega(p, t_{e,p}))$$

$$- (t_p - \upsilon(p, t_p) - \omega(p, t_p)) > 0$$

$$\wedge\ \omega(p, t_{e,p}) \leq \omega(p, t_p)]]$$

In the worst case, the sequence of events indicated in (7) will be repeated until $\omega$ is 1, at which point a successful transmission is guaranteed. This takes at most $n$ repetitions, where $\omega_0 = 2^n$. The result then follows from (2), (3), (4), (5), and (6).

□

## 6.5. The Aggressive Asynchronous FCFS Protocol

The conservative asynchronous adaptation of the FCFS protocol has now been shown to have bounded delay in the worst case. An adaptation is now proposed which takes a more aggressive approach to interpreting network events associated with collisions. In the ideal scenario, this protocol achieves better throughput than the conservative protocol, particularly as the size of the network increases. However, it will be demonstrated that the protocol is not free from deadlock under certain circumstances. A deadlock detection and recovery mechanism for the protocol is presented. The recovery protocol is a resolution procedure based on unique addresses which can be generalized to a novel topology-dependent resolution protocol.

The collision handling portions of the conservative protocol (statements S26 and S57) cause the protocol to wait for an interval equal to the worst case duration of a collision ($2D + jam$) after the channel is detected busy before proceeding with the next step in the resolution process. This prevents any ambiguity in interpreting signals in the event that a non-contiguous collision should occur, as illustrated in Figure 6.2. However, many collisions are actually over much sooner, and in these cases, the extra delay is not necessary. The aggressive protocol eliminates the extra delay. The specification of the aggressive protocol is identical to that for the conservative algorithm with the exception that the delays of statements S28, S31, S59 and S62 are eliminated.

### 6.5.1. Performance of Aggressive FCFS

A best-case estimate of the performance gains possible for the aggressive protocol is produced from the analysis of the conservative FCFS protocol by Towsley and Venkatesh. In [TV82], the performance of the unslotted FCFS protocol is discussed in terms of *epochs*, an epoch being the interval starting with the stations conducting a right interval search on a window of the default size, and ending when the protocol successfully transmits a packet while searching the right interval. If a collision occurs, the epoch consists of all algorithms steps required to produce a successful transmission from the right interval search, otherwise the length of the epoch is the time necessary to detect an idle step or to transmit a packet. The following expression is derived for the expected time to complete an epoch in the

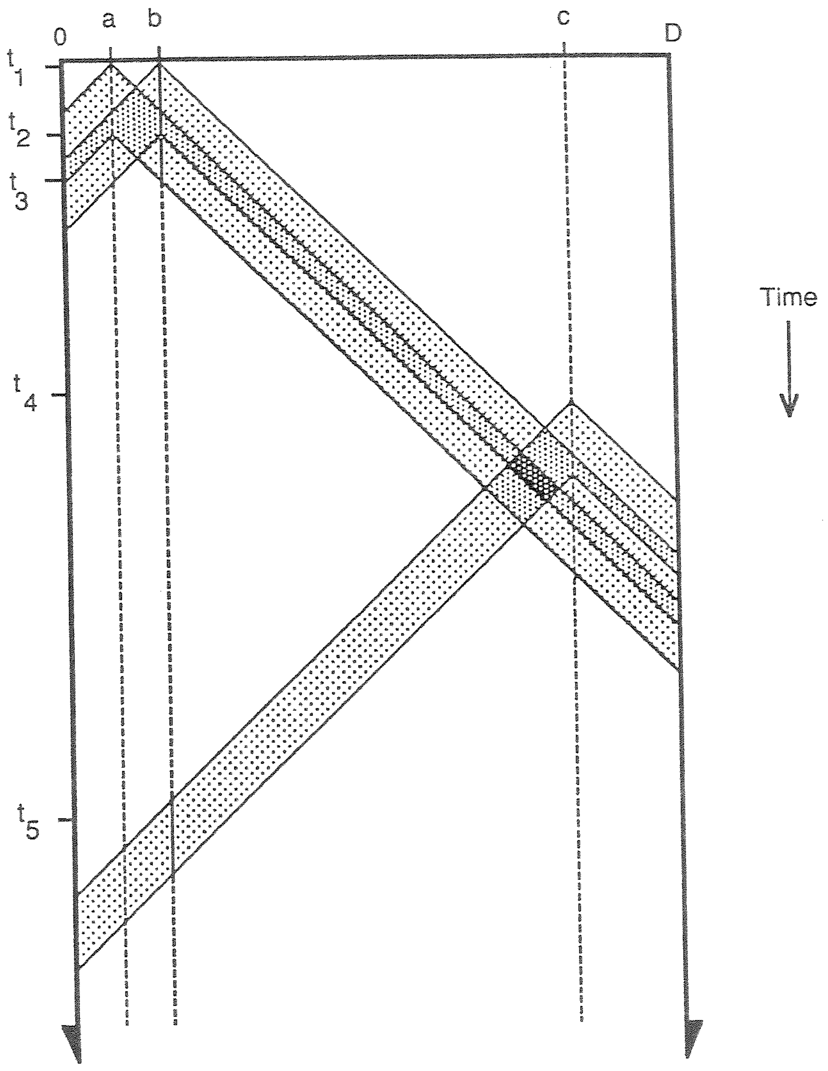


**Figure 6.2:** A non-contiguous collision.

unslotted FCFS protocol, given than $n$ arrivals occurred during the currently enabled interval:

$$E[t|A_n] = \begin{cases} \alpha, & n = 0 \\ 1+\alpha, & n = 1 \end{cases} \qquad \text{(E1)}$$

$$E[t|A_n] = \{t_c(n)[1-(1-p)^n] + n(1-p)^{n-1}pE[t|A_{n-1}]$$
$$+ \sum_{i=1}^{n-1} \binom{n}{i} (1-p)^{n-i}p^i E[t|A_i]\}/[1-(1-p)^n - p^n], \qquad n \geq 2,$$

where the following notation is used:

| | |
|---|---|
| $\alpha$: | The propagation delay for the network, expressed as a fraction of packet transmission time. |
| $A_n$: | The event that there are n messages in the currently enabled window. |
| $P(A_n)$: | The probability that there are n messages in the currently enabled window. Towsley and Venkatesh assume that the arrival process is Poisson, so $P(A_n) = (\lambda\omega)^n e^{-\lambda\omega}/n!$, where $\omega$ is the length of the interval. |
| $t$: | A random variable for the time required for an epoch to complete. |
| $s$: | A random variable for the fraction of a window (enabled interval) resolved during an epoch. This variable is necessary due to the fact that the algorithm "discards " the previous right interval when a left interval is subdivided. |
| $t_c(n)$: | The average duration of a collision among n messages. Towsley and Venkatesh assume that for protocols with collision detection, $t_c(n) = 2\alpha$ for all collisions. |

The quantity $t_c$ is incorporated into the algorithm presented by Towsley and Venkatesh. By altering the value of $t_c(n)$ in equation 1, the effect of a lower collision detection time on the performance of the algorithm can be investigated. In Figure 6.3, the maximum throughput for the protocol for various network lengths has been plotted for several values of $t_c(n)$. Throughput is taken to be the value of the Poisson parameter $\lambda$ at which the expected value of $t$ is equal to the expected value of $s$ times the default window size. An expression similar to (E1) is given in [TV82] for $E[s]$. The propagation delay is again normalized to the duration of a packet transmission, and throughput is expressed as a percentage of the capacity of the channel. The lower curve is the performance of the conservative asynchronous FCFS protocol, i.e. the collision clear time is $2\alpha$. The upper curve is the ideal case where

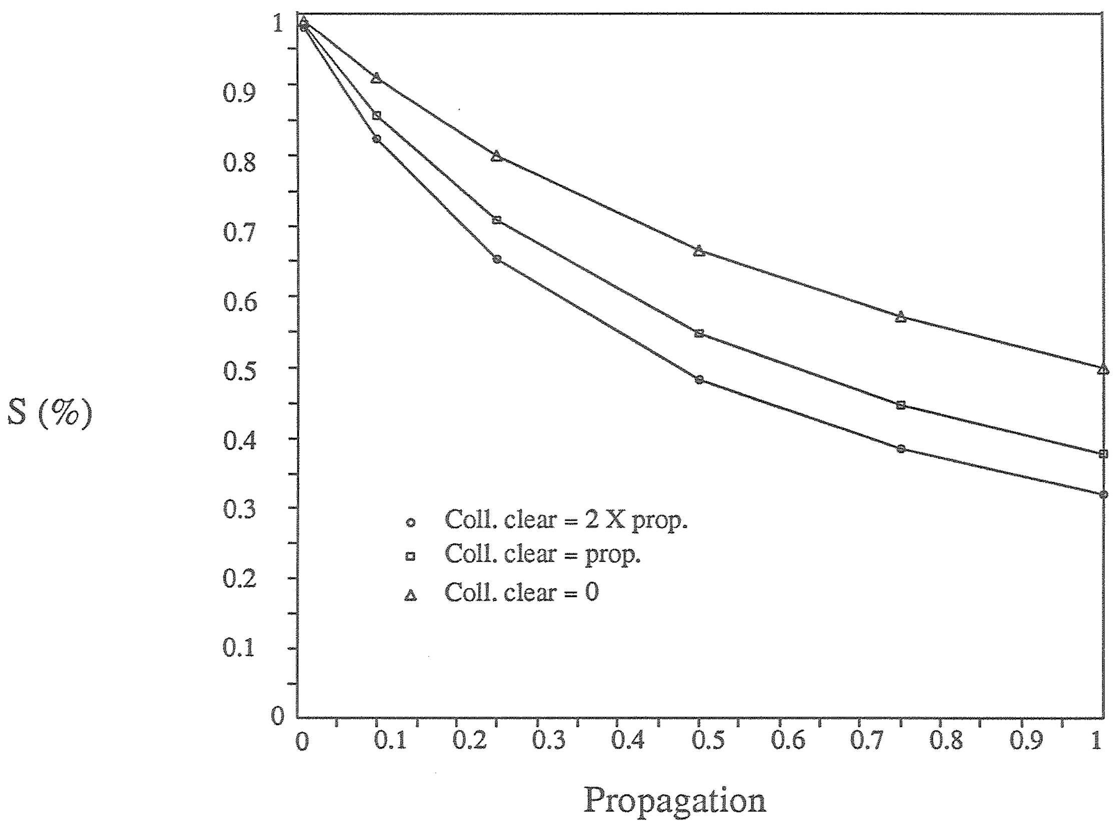


**Figure 6.3:** Performance benefits of aggressive FCFS. Propagation delay is normalized to packet transmission time. S is throughput expressed as a percentage of the capacity.

collisions are instantaneously detected and aborted so that the term in equation (E1) for the time due to collisions is eliminated. The middle curve shows a reasonable average case where collision duration is one propagation delay. Figure 6.3 shows that for very short networks, there is little gain, but as network length increases, performance of the aggressive protocol can be significantly better than that of the conservative protocol. For networks whose length is half the transmission time, the conservative protocol achieves a peak throughput of 48% of capacity, while the ideal aggressive protocol peaks at 66% and the average case at 55%.

### 6.5.2. Potential Deadlock

These figures assume that the protocol operates correctly. However, the algorithm as presented makes no provisions for distinguishing between new collisions and non-contiguous portions of old collisions, and as a result, is prone to deadlocks in a particular situation which will be discussed below. This emphasizes the effect that design decisions can have on the correctness of protocols adapted from slotted to unslotted networks. Modifications to correct this aspect of the algorithm will be presented. Since the actual change to the algorithm is slight, the algorithm description will not be repeated. References to the algorithm refer to the previous description.

The deadlock scenario for the aggressive FCFS protocol is shown in Figures 6.4 and 6.5. Figure 6.4 shows the situation on the channel resulting from the initial transmissions of a set of stations which produces a particular kind of non-contiguous collision. Figure 6.5 uses a time line to show how the actions of the protocol affect the protocol's enabled window. In Figure 6.4, the pairs of stations at opposite ends of the channel have been "synchronized" by a successful transmission by a station near the center of the channel. All four of these station have arrival times that lie in the later half of the current enabled window. For the sake of simplicity it is assumed that all of the enabled stations begin to transmit at the same time. This time is called $t_1$. Figure 6.5(a) shows the arrival times of the four stations and the currently enabled window. The resulting collision is of the non-contiguous type, with two localized subcollisions which are detected and aborted quickly. There is an idle period on the channel at each station between the end of the localized collision and the detection of the signal from the colliding pair at the opposite end of the network. In the conservative algorithm, at time $t_3$ the colliding stations begin to wait for two propagation delays to allow the channel to clear. In the aggressive algorithm, the stations proceed immediately to the subdivision of the interval. Since the arrival times are in the later half of

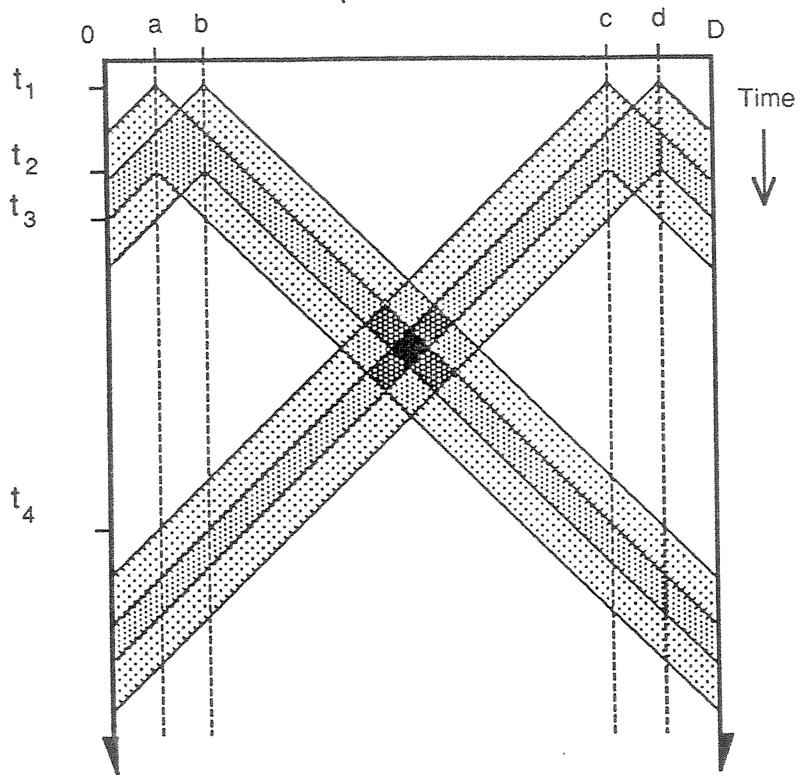


**Figure 6.4:** Channel behavior for the aggressive FCFS deadlock scenario.

the current window, when the subdivision is performed, all four stations will defer and wait for colliding stations in the early half of the window (now the enabled window) to transmit. At this point, the enabled interval is as shown in Figure 6.5(b). During this wait, at time $t_4$ in Figure 6.4, the collisions from the opposite end of the channel will be detected. This causes the stations to assume that a collision occurred among enabled stations in the current window, when, in fact, no such stations exist. Since a collision occurred, the current interval is once again subdivided, with the first half enabled, as shown in Figure 6.5(c). Nothing occurs, so the stations conclude that the phantom colliding stations must be in the latter half of the interval. This interval is once again subdivided... *ad infinitum*, as shown in Figure 6.5(d). This problem is similar to the one uncovered in the Massey improvement to the Capetanakis Tree protocol [Mass80] when coins are flipped immediately after a collision followed by an idle slot.

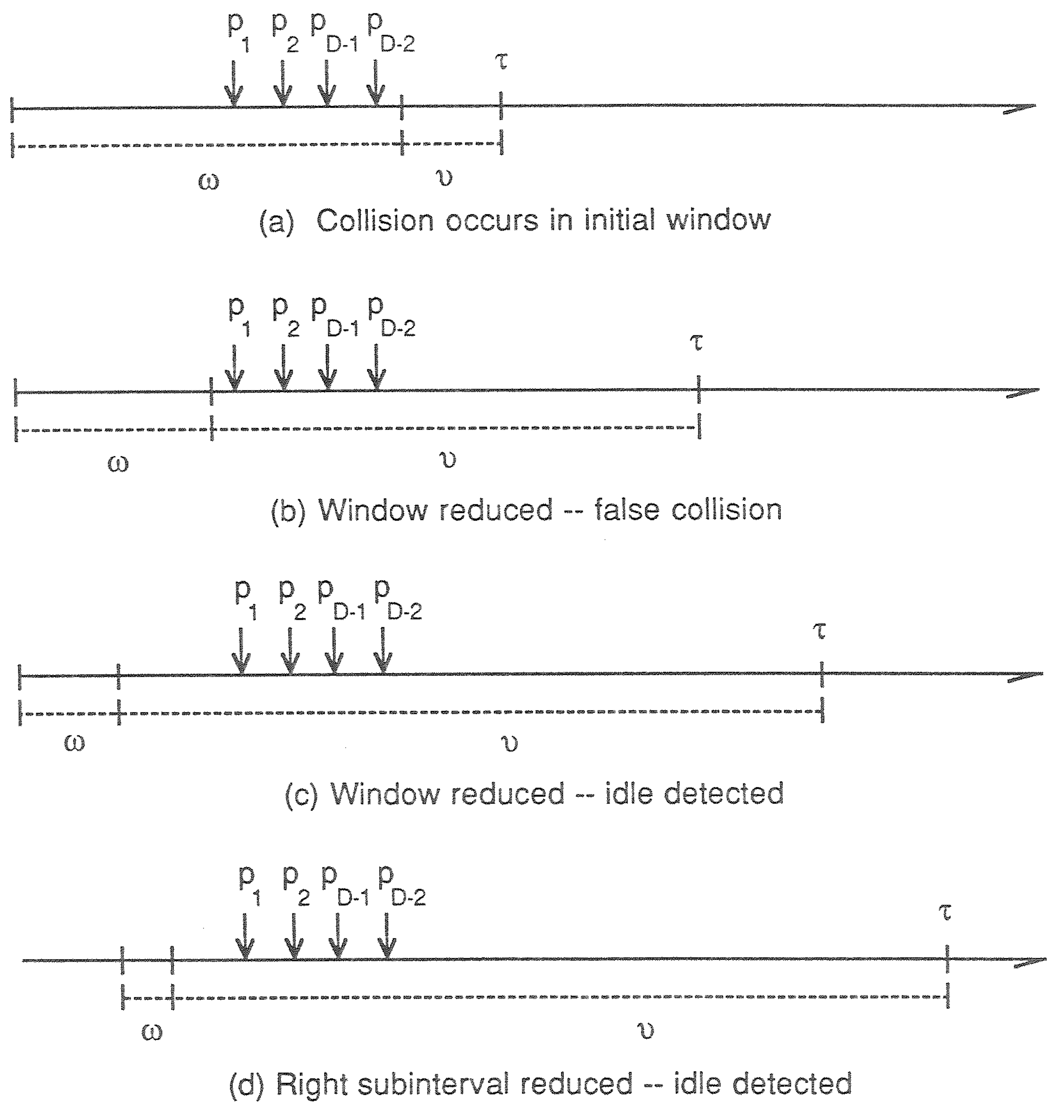

p1 p2 pD-1 pD-2
τ
ω
υ
(a) Collision occurs in initial window
p1 p2 pD-1 pD-2
τ
ω
υ
(b) Window reduced -- false collision
p1 p2 pD-1 pD-2
τ
ω
υ
(c) Window reduced -- idle detected
p1 p2 pD-1 pD-2
τ
ω
υ
(d) Right subinterval reduced -- idle detected

**Figure 6.5:** Interval and arrival time relationship for the aggressive FCFS deadlock scenario.

More formally, consider a network with channel of length $D$. A successful transmission originating from the station attached to the center segment has just completed. Let there be four stations, $p_1$, $p_2$, $p_{D-2}$ and $p_{D-1}$ with packets to send, where the subscript indicates the segment to which the station is attached. This set of stations will be referred to as $X$. If $\tau$ is the current time, $\upsilon$ is the current delay, $\omega$ the current enabled interval, and **arrival_time** is the arrival time for the current packet at a station, let the following condition be true:

(1)

$$[\forall p:\ p \in X:\ \tau - \upsilon - \omega/2 < \textbf{arrival\_time}(p) < \tau - \upsilon].$$

The station is thus enabled and its arrival time is in the latter half of the current window. Since a successful transmission has just completed, all stations are at the beginning of the right interval search.

(2)

$$[\forall p:\ p \in X:\ \textbf{at}(p, S3, \tau)\ \wedge\ \textbf{quiescent}(p, \tau)\ \wedge\ \textbf{enabled}(p, \tau)]$$

$$\Rightarrow [\forall p:\ p \in X:\ \textbf{in}(p, S4, \tau)\ \wedge\ \textbf{tx\_begin}(p, \tau + 1)] \qquad [SI1]$$

$$\Rightarrow [\forall p, \exists \varepsilon:\ p \in X:\ [\forall t':\ \tau \le t' < \tau + \varepsilon:\ \textbf{transmitting}(p, t') \qquad [I5]$$

$$\wedge\ ((\varepsilon = \textbf{tx\_time}(p) - 1)\ \vee\ \textbf{coll\_begin}(p, t + \varepsilon))]]$$

(2) indicates that the four stations of interest have begun transmission and will continue to transmit until their packet is successfully transmitted or a collision occurs. The relationship between the interval and arrival times at this point is as shown in figure 6.5(a).

(3)

$$(\ \textbf{tx\_begin}(p_1, \tau + 1)\ \wedge\ \textbf{tx\_begin}(p_2, \tau + 1)\ \wedge\ \Delta_{p_1 p_2} = 1\ ) \qquad [(1)]$$

$$\Rightarrow (\ \textbf{coll\_begin}(p_1, \tau + 2)\ \wedge\ \textbf{coll\_begin}(p_2, \tau + 2)\ ) \qquad [Th6.4, ThJ1]$$

(4)

$$( \textbf{tx\_begin}(p_{D-1}, \tau+1) \wedge \textbf{tx\_begin}(p_{D-2}, \tau+1) \wedge \Delta_{p_{D-1}p_{D-2}} = 1 ) \qquad [(1)]$$

$$\Rightarrow ( \textbf{coll\_begin}(p_{D-1}, \tau+2) \wedge \textbf{coll\_begin}(p_{D-2}, \tau+2) ) \qquad [\text{Th6.4,ThJ1}]$$

These statements indicate that the pairs of stations have collided among themselves. Since the stations are one segment apart, the collisions are detected one tick after the transmissions begin. (Note that having the outside stations begin transmission one tick later than the inside stations would not have affected the outcome. The two collisions would still occur.)

(5)

$$[\forall p : p \in X : \textbf{in}(p, S4, \tau) \wedge \textbf{tx\_begin}(p, \tau+1) \wedge \textbf{coll\_begin}(p, \tau+2)] \qquad [(2),(3),(4)]$$

$$\Rightarrow [\forall p : p \in X : \textbf{tx\_end}(p, \tau + jam + 3) \qquad [\text{I6}]$$

$$\wedge \text{ label}(p, \tau + jam + 3) = \text{coll\_detected}]$$

$$\Rightarrow [\forall p : p \in X : \textbf{after}(p, S4, \tau + jam + 3)]$$

Upon detecting the collisions, the stations delay for *jam* ticks, then end their transmissions. At this point the transmit procedure terminates. Consider station $p_1$.

(6)

$$\neg \textbf{busy}(p_1, \tau + jam + 3 + \Delta_{p_1p_2}) \qquad [(5)]$$

$$\equiv \neg \textbf{busy}(p_1, \tau + jam + 4). \qquad [\text{D6.5}]$$

It is also clear from (2) that

(7)

$$\textbf{busy}(p_1, \tau + 1 + \Delta_{p_{D-2}p_1}).$$

This indicates that the signal from $p_{D-2}$ has reached $p_1$. As long as $\Delta_{p_{D-2}p_1}$ is larger than $jam + 4$, there will be an idle interval on the channel at segment 1 between the end of the signal from station $p_2$ and the beginning of the signal from station $p_{D-2}$. This condition is assumed to hold.

(8)

$$( \textbf{after}(p_1, S4, \tau + jam + 3) \qquad [(5),(6)]$$

$$\wedge\ \text{label}(p_1, \tau + jam + 3) = \text{coll\_detected}\ \wedge \neg\,\textbf{busy}(p_1, \tau + jam + 4)\ )$$

$$\Rightarrow (\ \textbf{at}(p_1, S35, \tau + jam + 4) \qquad [SI3]$$

$$\wedge\ \upsilon(p_1, \tau + jam + 4) = \upsilon(p_1, \tau) + jam + 4 + \omega(p_1, \tau)/2$$

$$\wedge\ \omega(p_1, \tau + jam + 4) = \omega(p_1, \tau)/2\ )$$

$$\Rightarrow \neg\,\textbf{enabled}(p_1, \tau + jam + 4) \qquad [C6.2,(1)]$$

The station has subdivided the interval in response to the collision, resulting in the arrival time of the station being outside the enabled window. This situation is shown in Figure 6.5(b) One execution of the left interval search has been completed.

(9)

$$(\ \textbf{at}(p_1, S35, \tau + jam + 4)\ \wedge \neg\,\textbf{enabled}(p_1, \tau + jam + 4)\ ) \qquad [(8)]$$

$$\Rightarrow \textbf{in}(p_1, S40, \tau + jam + 4). \qquad [SI7]$$

Since the station is not enabled, it waits in statement S40 for transmissions by stations in the enabled window. Invoking the assumption that $\Delta_{p_D \to p_1} > jam + 3$,

(10)

$$(\ \textbf{in}(p_1, S40, \tau + jam + 4)\ \wedge\ \textbf{busy}(p_1, \tau + 1 + \Delta_{p_D \to p_1})\ ) \qquad [(7),(9)]$$

$$\Rightarrow \textbf{in}(p_1, S46, \tau + 1 + \Delta_{p_D \to p_1}). \qquad [SI10]$$

As noted, the signal from the stations at the opposite end of the channel reach $p_1$ at $\tau + 1 + \Delta_{p_D \to p_1}$. This causes the station to monitor the duration of the detected signal, which appears to station $p_1$ to have originated from a station which is currently enabled. As noted, there are no such stations.

(11)

$$( \textbf{tx\_end}(p_{D-1}, \tau + jam + 2) \wedge \tau + jam + 2 + \Delta_{p_{D-2}p_1} < \tau + jam + 2 + \Delta_{p_{D-1}p_1} ) \qquad [(5)]$$

$$\Rightarrow \neg\, \textbf{busy}(p_1, \tau + jam + 2 + \Delta_{p_{D-1}p_1}) \qquad [\text{ThJ1}]$$

The fact that all stations ceased transmission at $\tau + jam + 2$ as indicated in (5) allows us to determine that $p_1$ is no longer busy when that end of signal has propagated from $p_{D-1}$ to $p_1$.

(12)

$$\textbf{in}(p_1, S46, \tau + 1 + \Delta_{p_{D-1}p_1}) \qquad [(10),(11)]$$

$$\wedge\ \neg\, \textbf{busy}(p_1, \tau + jam + 2 + \Delta_{p_{D-1}p_1}) \wedge jam + 1 < \text{min\_size} \qquad [\text{A6.7}]$$

$$\Rightarrow ( \text{label}(p_1, \tau + jam + 2 + \Delta_{p_{D-1}p_1}) = \text{other\_coll} \qquad [\text{SI11,SI12}]$$

$$\wedge\ \textbf{at}(p_1, S35, \tau + jam + 2 + \Delta_{p_{D-1}p_1})$$

$$\wedge\ \upsilon(p_1, \tau + jam + 2 + \Delta_{p_{D-1}p_1})$$

$$= \upsilon(p_1, \tau + jam + 4) + \Delta_{p_{D-1}p_1} + \omega(p_1, \tau + jam + 4)$$

$$\wedge\ \omega(p_1, \tau + jam + 2 + \Delta_{p_{D-1}p_1}) = \omega(p_1, \tau + jam + 4)/2 )$$

$$\Rightarrow \textbf{in}(p, S40, \tau + jam + 2 + \Delta_{p_{D-1}p_1}). \qquad [\text{SI7}]$$

Axiom A6.7 indicates that the minimum packet transmission time is larger than the maximum collision duration. This allows stations to determine that a short transmission is a collision fragment, as in this case. Upon determining that a collision involving other stations has taken place, the interval is further subdivided, resulting in the situation shown in figure 6.5(c). This completes a second iteration of the left interval search. The same argument can be made for all of the stations in $X$.

$$[\forall p : p \in X : \textbf{tx\_end}(p, \tau + jam + 3) \wedge \textbf{in}(p, S40, \tau + jam + 2 + \Delta_{p_{D-1}p_1})] \quad [(5),(12)]$$

$$\Rightarrow [\forall p : p \in X : \textbf{after}(p, S44, \tau + jam + 2 + \Delta_{p_{D-1}p_1} + 2D)$$

$$\wedge\ \upsilon(p, \tau + jam + 2 + \Delta_{p_{D-1}p_1} + 2D) = \upsilon(p, \tau + jam + 2 + \Delta_{p_{D-1}p_1}) + 2D - \omega$$

$$\wedge\ \omega(p, \tau + jam + 2 + \Delta_{p_{D-1}p_1} + 2D) = \omega(p, \tau + jam + 2 + \Delta_{p_{D-1}p_1})/2]$$

$$\Rightarrow [\forall p: p \in X: \mathbf{at}(p, S35, \tau + jam + 2 + \Delta_{p_{D-1}p_1} + 2D)$$

$$\wedge \neg\, \mathbf{enabled}(p, \tau + jam + 2 + \Delta_{p_{D-1}p_1} + 2D)]$$

All signals from the original collision have completed their propagation, so the enabled stations observing the channel detect an idle step. Since their wait was in response to a perceived collision, the algorithms advance the window by the size of the current interval and immediately subdivide the interval again, since all of the (phantom) colliding stations must have been in the current right subinterval. This completes a third iteration of the left interval search. The situation is shown in 6.5(d). All subsequent left interval searches will have the same result, leading to a fault when the resolution of $\omega$ is reached.

## 6.5.3. Deadlock Detection

Given that the potential for deadlock exists in the aggressive protocol, the possibility of detecting the deadlock and recovering from it is now considered. The demonstrated deadlock occurs because the algorithm erroneously specifies subdivision of an enabled window which contains no arrival times. The suggested deadlock detection mechanism is to specify some threshold for the number of subdivisions performed or the minimum size for the enabled interval.

Most computer systems keep time using integer valued registers. This is in accord with the integer clock of the model used here, so the following scheme for detection of deadlocks is suggested: let the initial value of the interval, $\omega_0$ be a power of two, and let the splitting fraction for the algorithm be 1/2. A deadlock is detected when the value of $\omega$ becomes zero. This will occur whether the deadlock is due to the situation described in Section 6.5.2 or due to multiple stations with identical arrival times. The protocol can thus test the value of $\omega$ whenever a subdivision is performed and respond appropriately when the value becomes zero. This response could range from returning an error message and declaring the network broken, as is the case for most Ethernet interfaces when the collision limit of the backoff algorithm is reached, to attempting to recover from the deadlock in some fashion. Using this scheme, a deadlock is detected after at most $\log(\omega_0) \times 2D$ ticks. This is similar to the approach taken in [Humb86] to producing a Limited Sensing algorithm. Direct application of the methods presented in [Humb86] would require the introduction of another counter to keep track of idle steps in the algorithm.

### 6.5.4. Deadlock Recovery

Having suggested a method for detecting the deadlock produced by the aggressive adaptation of the protocol, the following possible recovery strategies are considered. One possible strategy is to remember the protocol state at the point when the last signal was detected and to attempt to "roll back" to that point and take the correct action. However, this assumes that the only type of deadlock that occurs in the protocol is the one demonstrated here, and this may not be the case. As discussed in the previous section, the fact that most computers use integer time keeping means that in normal operation of the protocol it is possible for two participants to believe that they have packets with identical arrival times, a situation which will also result in a deadlock. The deadlock detection scheme works equally well in both situations, the recovery scheme should also handle both of the deadlock situation equally well.

The FCFS algorithm bases collision resolution on the arrival times of messages. In the case of the deadlocks described, this method of performing the resolution has failed, therefore, some other resolution criterion for recovery from deadlocks must be used. Taking a cue from the work of Capetanakis, a resolution algorithm based on unique addresses as the backup resolution mechanism is chosen. The resulting protocol is essentially a version of the Tree protocol [Cape79] for finite population networks which maintains the consistency of the parameters for the arrival time resolution. Thus, when the deadlocked stations have been allowed to transmit, the FCFS algorithm can resume at the point where it left off. The recovery mechanism must run until all deadlocked stations have transmitted, since a resumption of the initial resolution algorithm with more than one of the deadlocked stations still active would result in another deadlock and further wasted time.

The recovery mechanism is implemented by checking the value $\omega$ after each subdivision and executing the procedure space_resolve() if the value is 0. The following enabling condition is used in the recovery algorithm:

$$\text{space\_enabled} \equiv \text{enabled} \qquad \text{(C6.3)}$$
$$\wedge\ D - \text{top\_offset} - \text{interval} \leq \text{my\_address} < D - \text{top\_offset},$$

where *my_address* is assumed to be a unique address assigned to the station. The resulting protocol prioritizes the stations according to address; stations with lower addresses transmit before stations with higher addresses. Note that in the case of the phantom-collision deadlock, the FCFS property is not maintained by the recovery algorithm, but in the case of identical arrival times, it is. Busy and transmitting are shared variables, as before. The following are constants accessible to all routines: *D, LEFT, RIGHT, idle, packet_sent, other_tx, coll_detected, other_coll, min_size.* The structure of the code is reminiscent of the left interval search in the main statement of the algorithm, since it is always the case that a collision has occurred when the space_resolve() routine is called. However, the routine does resolve the entire address space, as noted above.

```
procedure space_resolve(value tx_time, ω, busy;
            value result transmitting, packet_to_send, υ, label);
  begin
     var interval, top_offset, tx_start, tx_stop, wait_start:INTEGER;

     ⟨R1⟩ interval := D/2;
     ⟨R2⟩ top_offset := interval;
     ⟨R3⟩ push(LEFT);

     ⟨R4⟩ while interval < D do begin
               ⟨R5⟩ if space_enabled then
                    begin
                         ⟨R6⟩ tx_start := τ;
                         ⟨R7⟩ transmit (tx_time, label);
                         ⟨R8⟩ tx_stop := τ - 1;
                    end
              else /* if not space_enabled */
                    begin
                         ⟨R9⟩ wait_start := τ;
                         ⟨R10⟩ wait(busy or 2*D);
                         ⟨R11⟩ if not busy then
                              begin
                                   ⟨R12⟩ υ := υ + 2 * D;
                    ⟨R13⟩ if tos() = LEFT then
                        begin
                            ⟨R14⟩ backup(interval);
                            ⟨R15⟩ push(LEFT);
                            ⟨R16⟩ interval := interval/2;
                        end
                    else
                        ⟨R17⟩ backup(interval);
                    ⟨R18⟩ top_offset := top_offset - interval;
                    ⟨R19⟩ label := idle;
                end
            else  /* if busy */
                begin
                    ⟨R20⟩ tx_start := $\tau$;
                    ⟨R21⟩ wait (not busy);
                    ⟨R22⟩ tx_stop := $\tau$;
                    ⟨R23⟩ if (tx_stop - tx_start) < min_size then
                        ⟨R24⟩ label := other_coll;
                    else
                        ⟨R25⟩ label := other_tx;
                end  /* if not busy */
            end /* if space_enabled */

    ⟨R26⟩ if label = other_tx or label = packet_sent then
        begin
            ⟨R27⟩ if label = other_tx then
                ⟨R28⟩ $\upsilon$ := $\upsilon$ + (tx_stop - tx_start);
            else
                begin
                    ⟨R29⟩ $\upsilon$ := $\upsilon$ + tx_time + 2;
                    ⟨R30⟩ packet_to_send := FALSE;
                end
            ⟨R31⟩ backup(interval);
            ⟨R32⟩ top_offset := top_offset - interval;
        end
    ⟨R33⟩ else if label = coll_detected or label = other_coll then
        begin
            ⟨R34⟩ if label = coll_detected then
                ⟨R35⟩ $\upsilon$ := $\upsilon$ + (tx_stop - tx_start) + 2;
            else
                ⟨R36⟩ $\upsilon$ := $\upsilon$ + (tx_stop - wait_start);
```

```
                                        ⟨R37⟩ push(LEFT);
                                        ⟨R38⟩ interval := interval/2;
                                        ⟨R39⟩ top_offset := top_offset + interval;
                                    **end**
                        **end** /* while */
    **end** /* procedure space_resolve */


**procedure** backup (**value result** interval);
    **begin**
        ⟨B1⟩ **while** tos() = RIGHT **do begin**
                ⟨B2⟩ interval := interval*2;
                ⟨B3⟩ pop();
            **end** /* while */
        ⟨B4⟩ pop();
        ⟨B5⟩ push(RIGHT);
    **end** /* procedure backup */
```

The procedure space_resolve () keeps a stack recording whether the intervals resolved so far (" subtrees" in the nomenclature of Capetanakis; " intervals" will be used here to preserve consistency with arrival time resolution) were left or right intervals, with "left" indicating the interval containing lower addresses and " right" indicating the interval containing higher addresses. The stack is manipulated by the push(), pop(), and tos() calls, which respectively push an element onto the top of the stack, pop the top element from the stack, and return the value of the top element of the stack. The procedure backup() is used to determine how far "up the tree" to return when a success or idle step occurs, i.e. backup() determines the correct interval size for the next step of the algorithm given the contents of the stack.

Space resolution is analogous to arrival time resolution. The highest address, here represented by $D$, is analogous to $\tau$, the current time in the arrival time resolution. The variable *top_offset* is similar to $\upsilon$, indicating how far to the right the current address window starts. The variable *interval* performs the function of $\omega$, indicating the size of the enabled address window.

The addition of an address resolution phase results in an instance of a class of hybrid protocols which can be thought of as time-space resolution protocols, suggested to the author by Simon Lam [Lam88]. In general, such a protocol could use some function of the packet arrival time and the position of the station on the network as the criterion for subdivision. The resulting protocol should have similar analytic properties to the original FCFS protocol, though the FCFS property would be lost. The FCFS protocol is an example of one extreme, the time resolution protocol. An example of the other extreme, the space resolution protocol, is discussed in Section 6.6.

It should also be noted that using the suggested deadlock detection method and the space_resolve() recovery procedure allows new stations to join the network without having complete knowledge of the channel history. Two factors make this possible. First, the deadlock detection method guarantees that no more than $log(\omega_0)$ idle steps will occur before the space_resolve() procedure is invoked. Second, the space_resolve() procedure will produce a successful transmission within at most $log(D)$ steps, where $D$ is the number of segments (or stations) on the network. Thus, there can be at most $log(\omega_0) \times log(D)$ idle steps in a row. A station observing the channel idle for longer than this is guaranteed that the currently enabled interval is a right interval and that the size of the interval is $\omega_0$. While the likelihood of an idle period of this length depends on the load on the network, it introduces the possibility of joining the protocol in progress. Should this prove impossible within a reasonable amount of time, other measures to reset the state of all participating stations would be necessary. A noise burst of duration greater than the maximum packet transmission time (or any other unique globally detectable event) could be used to signal that such a reset is needed.

## 6.6. The Space Division Multiple Access Protocol

As noted, the FCFS protocol is an example of one of the logical extremes of the class of time-space resolution protocols. This section discusses an example of the other extreme of the class, which will be referred to as Space Division Multiple Access (SDMA). The protocol discussed here is a specialization of protocols such as the Capetanakis Tree protocol which use a unique address to perform collision resolution. These protocols do not require any correlation between the address of a station and the location of the station on the network; this simplifies adding new stations to a network. It is simply necessary to find a unique address to assign to the new station. However, it is possible to imagine performance improvements which could result from using information about the relative positions of stations on the network. The protocol presented here exploits one such improvement. (Various unidirectional broadcast protocols such as Expressnet [TBF83] and Hymap [RG85] have made use of upstream/downstream positional information. These protocols are intended for use on folded bus or dual bus topologies.)

The SDMA protocol is similar to the address-based Capetanakis protocol with one important difference; if the address of the station is interpreted as a location (e.g. distance from one end of the cable), then the collision clear time required for the conservative implementation of the protocol in an unslotted medium can be reduced with each address space subdivision. To see why this is so, consider the situation when a collision occurs in such a network. (Assume that the splitting fraction is one half.) Divide the stations into two subsets, one subset containing addresses less than $N/2$, where $N$ is the number of positions on the network, and the other subset containing stations whose address is greater than $N/2$. Since addresses correspond to locations, this corresponds to a physical partitioning of the cable, with all the stations in the enabled subset on the same half of the network. This means that the maximum propagation delay between enabled stations will be half of the full propagation delay of the network. So, should another collision occur, the collision itself may be of shorter duration due to the proximity of the colliding stations, and the subsequent channel clear time will be half what it was for the previous step. Likewise, it will take only half as long to detect an idle step. While this protocol is equally suited to implementation on slotted or unslotted networks, the savings are even more important on unslotted networks where the intentional collision method is used to signal the end of idle steps.

There are several restrictions imposed by this scheme. As presented, SDMA imposes priorities by location; stations with lower addresses will transmit earlier. Under extreme loads, this will result in a round-robin ordering of transmissions. It is possible to alleviate this to some degree by alternating the order in which the subsets are resolved, and so on. Stations are required to continuously monitor the channel at all times as in FCFS, etc. As presented, SDMA also expects a single linear cable, a significant topology limitation. The protocol can be adapted to handle branching topologies by mapping addresses to create a logical linear cable with length equal to the sum of the lengths of the branches. On sparsely populated cables, further gains can be made by keeping track of the unused slots. If an enabled range of addresses consists wholly of unused slots, a step can be skipped.

# Chapter 7

# Conclusions

This thesis argues for the experimental investigation of data link level protocols for local area networks. This case cannot be made too strongly; in the area of broadcast bus networks, one method came to dominate the marketplace before other alternatives were ever implemented and evaluated. By the time this work was undertaken, it was all but impossible to perform the sorts of experiments necessary without going to the effort of custom fabricating hardware. In the interim, the situation has gotten worse rather than better; today, protocols seem to be issued as official standards before the first interface is ever built and tested.

## 7.1. Measurement Results

Measurement experiments were conducted on protocol implementations representing three general strategies for multiple access on broadcast bus networks. Ethernet is an example of a technique referred to here as random backoff. The Enet II protocol is an example of an unslotted Collision Resolution Protocol. The Virtual Time CSMA/CD protocol is an example of a Collision Avoidance Protocol. The experiment design subjected each protocol to artificially generated loads for ten different load patterns with varied packet length and arrival time distributions. The Enet II protocol was also used for normal operation of the testbed network for several months and performed satisfactorily. The importance of these studies is manifold. First, the Enet II protocol and the VTCSMA/CD protocol had not previously been implemented and studied in this way before. Second, no measurement studies of Ethernet which examined the behavior of the protocol under heavy loads with realistic traffic mixtures had been previously undertaken. Third, no direct comparison of implementations of the three approaches to collision handling represented in these three protocols had been done.

### 7.1.1. General Behavior of the Protocols

The measurement experiments produced a consistent picture of the relative performance of the three protocols across a wide range of load patterns and operating conditions. Under the normal operating conditions (offered loads of between 0 and 10 Megabits/second), Ethernet and Enet II produced throughputs closely matching the offered load, with Ethernet showing a slight edge in peak throughputs as the load approached saturation. Hampered by the overhead necessary for manipulating the virtual clock, the VTCSMA/CD protocol was not as effective under normal loads. When the offered load exceeded the capacity of the channel, both Ethernet and Enet II suffered a decrease in throughput followed by a restabilization at a lower throughput level. The stable level for Enet II was typically slightly higher than that for Ethernet. For Ethernet, we believe that the drop in throughput was due to the combination of an error in the backoff algorithm supplied in the 4.3BSD Unix device driver for the 3Com Ethernet interface, transmission overhead in the interface, and the small host population participating in the experiments. For Enet II, we believe that the drop in throughput was also due to a combination of factors, namely, the increase in the percentage of packets suffering two or more collisions before transmission (and thus possibly entering the deferred state) as the offered load increased past the capacity of the channel, the larger unit of delay used in the implementation due to the clock resolution problems discussed in Chapter 4, and the transmission overhead and host population factors mentioned for Ethernet. Under overloaded conditions, VTCSMA/CD produced stable throughput levels significantly higher than the other two protocols except in the situation where a large percentage of small packets were included in the traffic mix.

The delay behavior was also consistent across all of the load patterns investigated. Under normal loads with minimal contention, Ethernet and VTCSMA/CD experienced roughly the same delay, while Enet II incurred the initial gating delay specified in the algorithm. Without the gating delay, however, the delay curve for Enet II would have matched the curve for Ethernet very closely. Both Enet II and Ethernet experienced sharp increases in delay when the offered load reached the saturation point of the channel, while VTCSMA/CD experienced only slight increases under heavy loads. In some cases, mostly where the load was very heavy and some small packets were included in the traffic mixture, Enet II experienced lower average delay under overloaded conditions, but in most cases, the distance between the two curves merely decreased from the contention free portion of the curves. We note that a relationship more closely matching a correct implementation of the Enet II protocol could be gained by uniformly subtracting 450 microseconds from the Enet II delay. (This is the difference between the correct initial gating delay and the delay used in the implementation presented here due to the clock resolution.) In this case, Enet II would consistently experience lower average delays for the overloaded portion of the curve.

The relation of the delay variance curves was also consistent across the various loads. As one would expect, when contention was light, the variance of the delay was low. As the loads approached saturation, the delay variance increased with contention. The increase was slight for VTCSMA/CD, indicating the effectiveness of the collision avoidance strategy it employed. Note, however, that this variance effect is the result of the lower collision rate of the Virtual Time protocol rather than a feature of the backoff algorithm. The delay variance increase was more severe for Enet II, but was most severe for the random backoff protocol, Ethernet. This indicates that the the unslotted collision resolution method used by Enet II is a genuine improvement over the random strategy exemplified by Ethernet. Further evidence for this conclusion is drawn from the fact that the collision rates for Ethernet and Enet II were nearly identical for all load patterns and all offered loads, indicating that the variance improvement for Enet II is in fact a result of improved performance on a packet by packet basis. This result clearly indicates the desirability of unslotted Collision Resolution Protocols for networks where heavy loads are the norm.

### 7.1.2. The Effect of Exponential versus Fixed Arrivals

The extra overhead involved in generating exponentially distributed arrivals makes direct comparison of peak throughputs achieved by the two arrival processes of little value. However, we can make some general observations about the effects on the behavior of the protocols of the two arrival processes. The relative performance of the three protocols was not changed by the use of exponential arrivals. The observed bistability of Ethernet and Enet II was moderated under the exponential arrival process, though it is difficult to say whether this was a result simply of the lower packet generation rates or whether the fixed arrival process generated some more complex interaction or synchronization among the small experimental host population. The most noticeable difference between the behavior of the protocols under the two arrival processes was the collision rate. For fixed interpacket arrivals, the collision rate remained very low until the load reached the capacity of the channel, then increased dramatically. For the exponential arrivals, contention began at a loads well below the channel capacity for the Ethernet and Enet II protocols. VTCSMA/CD was not strongly affected by the change in arrival process. This is because whatever the arrival process is, VTCSMA/CD spreads transmission attempts relative to the end of busy periods using the virtual clock. This effect was possibly heightened in the LANT implementation by the overhead of operating the virtual clock, which could have been a further source of variation even in the fixed arrival case.

### 7.1.3. The Effect of Small Packets

As expected, the introduction of small packets into the packet length distribution had the effect of increasing contention and thus lowering throughput and increasing delay for all three protocols. VTCSMA/CD, which for the most part seemed immune to changes in the load pattern, did experience increased delay and delay variance for the load pattern containing the highest percentage of small packets. Throughput for the protocol was strongly affected, and the average delay became very close to that of Ethernet under heavy loads. Enet II also showed slightly greater deterioration of throughput and average delay than Ethernet as the percentage of small packets in the load pattern were increased. The clock resolution arguably contributed to this behavior.

## 7.2. Adapting Unslotted Protocols to Slotted Networks

The experiments discussed here indicate the desirability of applying collision resolution techniques to the multiple access problem for unslotted broadcast bus communication. The successful implementation of Enet II and VTCSMA/CD on such a network also indicates that it is possible to use these techniques, which previously had been designed strictly for time slotted media, on such unslotted networks. The Enet II protocol provides an important technique, signaling the end of idle periods with intentional collisions, for such implementations. This makes it reasonable to begin investigating the adaptation of slotted protocols for use on unslotted networks. However, it is also necessary to have the ability to provide formal assurance that such an adaptation is correct and maintains the salient features of the slotted version. Until recently, little progress had been made on general methods for proving formal properties of broadcast protocols. The development of Jain's model remedies this situation and makes investigation of the slotted-unslotted adaptation practical.

Chapter 6 presents an extension of Jain's model to handle collision detection more naturally and states the channel axioms reflecting the change. Subsequent results are developed describing the essential behavior of the network in the event that collisions occurred. These results are in turn used to provide a proof of bounded delay for the resolution of one contention epoch of an asynchronous version of Gallager's First Come First Served protocol.

A modification of the protocol specification which involves interpreting the end of a collision more liberally and thus potentially gaining efficiency is proposed. The resulting protocol is referred to as Aggressive FCFS. A demonstration is given that under some circumstances, this protocol can suffer deadlocks. This demonstration makes two important points; first, that formal proofs of correctness are vital to effective conversions of this sort and second, that collision handling behavior is the crucially difficult matter in such conversions. One of the important properties of the asynchronous version of FCFS is that all stations detect the channel clear following a collision within one propagation delay of each other. In the aggressive version of the protocol, this difference can be up to two propagation delays.

A deadlock detection and recovery method for the protocol is suggested in Chapter 6. Detection is performed by limiting the number of interval subdivisions that the protocol performs. This method is a variant of the method used by Humblet in [Humb86] for Limited Sensing protocols. This is a natural method, given that most computer systems keep time with integers; when the enabled arrival time window of the protocol reaches zero, the system concludes that a deadlock has occurred, either due to the deadlock situation we describe or due to the fact that two stations believe they have the same arrival time. The recovery mechanism is a second resolution process which uses unique addresses rather than arrival times as the basis for the resolution. In the case of the aggressive deadlock scenario, the FCFS property is violated. However, since the identical-arrival deadlock can occur even in the conservative version of the protocol, this detection/recovery method is still necessary, and in this case the FCFS property is preserved.

## 7.3. Space Division Multiple Access

The FCFS plus deadlock recovery protocol can be thought of as a member of a class of protocols suggested by Lam [Lam88] that perform collision resolution based on a function of both space and time A protocol is suggested that lies at the endpoint of this continuum of protocols, one which performs resolution strictly on the spatial location of stations. This protocol is referred to as Space Division Multiple Access. The interesting aspect of this protocol is that each subdivision is in fact a physical reduction in the size of the network upon which resolution must occur. This means that the time necessary for an idle step to be detected and the time for the channel to clear after a collision are halved at each subdivision, resulting in improved performance, particularly in situations where the length of the network is a large fraction of the packet transmission time and the load is normally heavy.

## 7.4. Contributions

The successful implementation of the Enet II protocol demonstrates the practicality of adapting collision resolution techniques for use in unslotted broadcast bus networks. The performance evaluation of Enet II demonstrates that the collision resolution method it employs successfully reduces the variance of delay for packets under heavy loads without resorting to centralized control.

The performance measurement results for the VTCSMA/CD protocol demonstrate its viability and the remarkable effectiveness of its collision avoidance technique. The protocol is shown to have better average delay characteristics and throughput under heavy loads than the Enet II and Ethernet protocols.

The problem of using slotted algorithms on unslotted networks is discussed. A successfully adapted protocol is presented along with a formal proof of bounded delay. The critical aspect of the adaptation, the interpretation of collision duration, is identified. A deadlock detection and recovery method, necessary in any implementation of the protocol presented that uses integer timekeeping, is presented. A related Collision Resolution Protocol which takes advantage of knowledge of the location of stations is presented.

## 7.5. Future Work

The success of the LANT demonstrated the feasibility and desirability of a testbed for data link layer protocols. However, there are several limitations imposed by the equipment used in the testbed. The protocols evaluated using the system were chosen partly due to their amenability to implementation with the available hardware and software. The available hardware also limited investigation to one network topology among many interesting choices. The concept of the LANT should be extended to a system which provides a consistent software environment for the investigation of protocols for many network architectures. Such a system would be composed of a large number of workstations with interfaces which could download protocol code for execution by onboard processors, avoiding the overhead incurred by the LANT software implementations. The benefits of such a testbed are considerable.

The field of computer communication in general is moving in several interesting directions. The idea of mixed service networks carrying voice and video information as well as data has gained popularity in recent years. Lessons learned in the LANT studies can profitably be applied to the design of protocols for this type of network. Likewise, new protocols are needed for the ultra-high speed networks envisioned which will operate over much larger distances as speeds an order of magnitude or more faster than the current generation of LANs.

The Space Division Multiple Access protocol is an interesting but recent development. It bears further attention, particularly toward the development of results on its performance.

A version of the model used in Chapter 6 which relaxes the global clock assumption is under development. Adapting the proofs presented in Chapter 6 to this model is desirable.

Finally, the guidelines for adapting slotted algorithms to unslotted networks presented in the introduction to chapter 6 have been tested by the specification and proof of correctness for one slotted CRP. Further conversions should be done to increase confidence in this method of adapting algorithms. Further investigation into the performance penalities implicit in the collision handling aspects of the resulting unslotted algorithms is also in order. In addition, it is possible that rules could be developed concerning maintaining correctness across the transformation of the algorithms from slotted to unslotted operation.

# Appendix A

# ILMON Filter Structure Definition

```
/*
 *      Structure for selective packet monitor feature in interface driver.
 */
#define HOSTMAX         32      /* # of selectively monitorable hosts */
#define NUMTYPES        19      /* Number of Ethernet packet types */
#define TREEMAX         2048    /* Array for binary tree of hgram ptrs */
#define HGMAX           140     /* Maximum size of histogram */
#define LOGMAX          5000    /* Maximum size of packet log */
#define QUALMASK        0x7fc00 /* Any qualifiers set? */

#ifndef NS
#define ETHERTYPE_NS    0x0600
#endif

/* Types not used by UNIX but present on our net.  (From CC documents) */
#define ETHERTYPE_CHAOS         0x804   /* Chaosnet */
#define ETHERTYPE_EXCL 0x8010  /* Excelan */
#define ETHERTYPE_RARP 0x8035  /* Reverse ARP */

struct  filter {
        int     fil_flags;              /* Action flags & qualifiers - n.b. */
        struct  timeval fil_tsin;       /* Time monitor session started */
        struct  timeval fil_tsout;      /* Time monitor session ended */
        int     fil_ensacnt;            /* Size of Enet source addr cklist */
        char fil_ensa[HOSTMAX][EASIZ]; /* Addresses to check for */
        int     fil_endacnt;            /* Size of Enet dest addr cklist */
        char fil_enda[HOSTMAX][EASIZ]; /* Addresses to check for */
        int     fil_ipsacnt;            /* Size of IP source addr cklist */
        struct  in_addr fil_ipsa[HOSTMAX];      /* IP addresses to check */
        int     fil_ipdacnt;            /* Size of IP dest addr cklist */
        struct  in_addr fil_ipda[HOSTMAX];      /* IP addresses to check */
        int     fil_type;               /* Packet type to monitor */
        int     fil_lengt;              /* Lower bound of lengths to monitor */
        int     fil_lenlt;              /* Upper bound of lengths to monitor */
        int     fil_leneq;              /* Monitor only pkts of this length */
        int     fil_errcnt;           /* Total errors from is_stats */
        int     fil_fragcnt;          /* Total frags from is_stats */
        int     fil_dropcnt;          /* Packets dropped during session */
        int     fil_framecnt;         /* Total frames during session */
        int     fil_lenhg[ETHERMTU+1]; /* Packet length histogram */
        struct  exp_hgram fil_typhg[NUMTYPES+1];/* Packet type histogram */
        int     fil_ensatr[TREEMAX+1]; /* Binary tree pointers for ensahg */
        int     fil_ensatop;          /* Next free element of ensahg */
        struct  exp_hgram fil_ensahg[HGMAX+1]; /* Hgram on Enet src addrs */
        int     fil_endatr[TREEMAX+1]; /* Binary tree pointers for ensahg */
        int     fil_endatop;          /* Next free element of ensahg */
        struct  exp_hgram fil_endahg[HGMAX+1]; /* Hgram on Enet dst addrs */
        int     fil_ipsatr[TREEMAX+1]; /* Binary tree pointers for ensahg */
        int     fil_ipsatop;          /* Next free element of ensahg */
        struct  exp_hgram fil_ipsahg[HGMAX+1]; /* Hgram on IP src addrs */
        int     fil_ipdatr[TREEMAX+1]; /* Binary tree pointers for ensahg */
        int     fil_ipdatop;          /* Next free element of ensahg */
        struct  exp_hgram fil_ipdahg[HGMAX+1]; /* Hgram on IP dst addrs */
        int     fil_logtop;           /* Next free element in pkt log */
};

struct  shortlog {
        struct  timeval         timestamp;
        struct  il_rheader      header;
        union {
                struct  ip              iphdr;
        } in_hdr;
};

/*
 *      fil_flags bits.
 */

/* Action flags -- determines the type of monitoring to be done */
#define FLT_PKTLOG      0x00001 /* Retain log of packets filtered */
#define FLT_ENSAHG      0x00002 /* Histogram on Ethernet source addresses */
#define FLT_ENDAHG      0x00004 /* Histogram on Ethernet dest addresses */
#define FLT_IPSAHG      0x00008 /* Histogram on IP source addresses */
#define FLT_IPDAHG      0x00010 /* Histogram on IP dest addresses */
#define FLT_PLENHG      0x00020 /* Packet length histogram */
#define FLT_PTYPHG      0x00040 /* Packet type histogram */

/* Qualifiers -- determines what filtering will be done on monitored pkts */
#define FLT_ERRS        0x00100 /* Count error packets */
#define FLT_VALID       0x00200 /* Count valid packets */
#define FLT_ENSA        0x00400 /* Count pkts w/ addr in fil_ensa */
#define FLT_ENDA        0x00800 /* Count pkts w/ addr in fil_enda */
#define FLT_IPSA        0x01000 /* Count pkts w/ addr in fil_ipsa */
```

```
#define FLT_IPDA       0x02000 /* Count pkts w/ addr in fil_ipda */
#define FLT_LNGT       0x04000 /* Count pkts w/ length > fil_lengt */
#define FLT_LNLT       0x08000 /* Count pkts w/ length < fil_lenlt */
#define FLT_LNRN       0x10000 /* Lengths > fil_lengt && < fil_lenlt */
#define FLT_LNEQ       0x20000 /* Count pkts w/ length = fil_leneq */
#define FLT_PTYP       0x40000 /* Count packets of a specific type */
```

# Appendix B

# Experiment Configuration File Structures

## B.1. Header File Structure

| BYTES | CONTENTS |
|---|---|
| 0-2 | Section complete flags |
| 3-12 | Experiment name (corresponds to file name) |
| 13 | Protocol code |
| 14-20 | Date of last modification (MMDDYY) |
| 21-27 | Date created |
| 28 | Unused |
| 29-30 | # of Processes on node 1 |
| 31-32 | # of Processes on node 2 |
| 33-34 | # of Processes on node 3 |
| 35-36 | # of Processes on node 4 |
| 37-40 | # of distributions defined |
| 41-55 | Load pattern name |
| 56-70 | Unused |
| 71-140 | Comment |

## DISTRIBUTION RECORDS

### Exponential Distribution

| | |
|---|---|
| 1-3 | Sequence # |
| 4 | Distribution code (= 1) |
| 5-13 | Value for theta (theta = mean interarrival time) |
| 14-20 | Unused |

### Poisson Distribution

| | |
|---|---|
| 1-3 | Sequence # |
| 4 | Distribution code (= 2) |
| 5-13 | Value for lambda (arrival rate) |
| 14-20 | Unused |

### Geometric Distribution

| | |
|---|---|
| 1-3 | Sequence # |
| 4 | Distribution code (= 3) |
| 5-13 | Value for p (probability of success) |
| 14-20 | Unused |

### Binomial Distribution

| | |
|---|---|
| 1-3 | Sequence # |
| 4 | Distribution code (= 4) |
| 5-13 | Value for n (number of trials) |
| 14-22 | Value for p (probability of success for any trial) |
| 23-40 | Unused |

### General Discrete Distribution

| | |
|---|---|
| 1-3 | Sequence # |

| | |
|---|---|
| 4 | Distribution code (= 5) |
| 5-7 | # of coordinate pairs entered |
| 8-10 | Point sequence # |
| 11-15 | x value |
| 16-20 | y value |
| . | |
| . | |
| . | |

Piecewise Continuous Distribution

| | |
|---|---|
| 1-3 | Sequence # |
| 4 | Distribution code (= 6) |
| 5-7 | # of coordinate pairs entered |
| 8-10 | Point sequence # |
| 11-15 | x value |
| 16-20 | y value |
| . | |
| . | |
| . | |

## B.2. Configuration File Structure

| BYTES | CONTENTS |
|---|---|
| 1-2 | Process number |
| 3-5 | Interpacket distribution code (determined from list in header file) |

| | |
|---|---|
| 6-8 | Seed index for time generation |
| 9-11 | Packet length distribution code (determined from list in header file) |
| 12-14 | Seed index for packet length generation |

# Appendix C

# ILMON User's Manual

## C.1. Using ILMON

The general sequence for performing network monitoring with ilmon is as follows:

1) Run ilmon and choose the interface (by unit number) with which you wish to do the monitoring with the "chif" command.

2) Load a filter description either manually using the "setfil" command or from a stored template using the "loadtmp" command.

3) Start promiscuous mode monitoring using the "setprom" command. Setting the mode flag to "1" puts the chosen interface into promiscuous mode.

4) Network activity can be periodically checked using the "getfil" command. If this is done before the interface is returned to normal mode, some of the quantities presented (stop time, frame count...) will be inaccurate.

5) End promiscuous mode monitoring using the "setprom" command with mode flag set to "0."

6) Retrieve the filter using the "getfil" command. This will present a summary of the information collected.

7) Save the filter to disk using the "savefil" command. The resulting file can be viewed with the program prfilter and turned into graphs and reports of average quantities with the program eap.

## C.2. Command Summary

| | |
|---|---|
| chif | If the computer running ilmon is a gateway or has more than one Interlan ethernet interface, chif allows you to determine which network will be monitored by switching interfaces. You will be informed of the unit number of the current interface and prompted for the unit number of the interface to change to. The default setting in unit 0. |
| cntr | Retrieve and print out the on-board statistics registers. Unix reads and resets these registers periodically, so what you get is the values over the last 0 to 60 seconds. |
| crcverr | Clear receive error mode. Necessary only when a rcverr has been previously performed. |
| getfil | Retrieve a filter from the device driver and display the collected statistics. Unless promiscuous mode is turned off before getfil is called, some of the statistics will be invalid. |
| hw-addr | This command prompts for a hostname and returns the internet and hardware addresses of the machine. Useful for deciphering source and destination address histograms. |
| loadfil | Read a filter (previously stored with savefil) from disk and display its contents. |
| loadtmp | Read a filter (previously stored with savetmp) from disk. A filter that was stored with savetmp *before* a monitoring session was run using it (i.e. it contains no data, only a description for monitoring sessions.) is called a template. After being read in, the template can be used to run a monitoring session. |
| rcverr | Set receive error mode. Normally the NI1010 automatically filters out error packets, but when crcverr is performed, all packets are returned to the device driver regardless of error status. (setprom |

| | |
|---|---|
| | performs a rcverr automatically, so it is not necessary to do this prior to setprom if you want to see error packets too.) |
| savefil | Write the results of a monitoring session out to disk for later examination. |
| savetmp | Write a filter that has been setup with setfil, but not yet used for a monitoring session, to a disk file. Templates that have been saved in this way can be loaded for use later using loadtmp, or may be used with the program timedmon. |
| setfil | Configure a monitoring session. This command is described in greater detail below. |
| setprom | Set (or clear) promiscuous mode. You will be prompted for a flag. 0 (the default) clears promiscuous mode and returns the interface to normal operation. 1 puts the interface in promiscuous receive mode and begins collecting the information specified in the filter. |
| start | Put the interface on line. (Left over from debugging, but may be useful.) |
| stop | Put the interface off line. (Left over from debugging, as above.) |
| z-cntr | Read and reset the onboard statistics registers. (Doing this may disrupt the interface level statistics reported by netstat.) |

## C.3. Configuring a Monitoring Session

Filters are used to tell the device driver what kind of packets we're interested in and how we want information about them stored. Filters are constructed with the "setfil" command. When you issue the setfil command, you will be presented with menus for "actions" (or what information to collect) and "qualifiers" (or the conditions a packet must satisfy to be included in the collected data. In either case, multiple entries are accepted, with choices separated by spaces.

The action menu allows the following choices.

1) Log packet headers - the ethernet and IP (where appropriate) headers of packets are collected in a buffer for individual examination.
2) Ethernet source address histogram - count of the number of packets originating at each source. Due to overhead of maintaining the data structures for this option, many packets are usually lost.
3) Ethernet destination address histogram - count of the number of packets intended for each destination. Warning from (2) applies.
4) IP Source address histogram - not implemented.
5) IP Destination address histogram - not implemented.
6) Packet length histogram - counts of packets of each length. Length is data area only, does not include ethernet header.
7) Packet type histogram - counts of packets of each type (e.g. IP, ARP, NS, ND, etc.)

1, 2, and 3 tend to lose packets. 6 and 7 are better about this, but still fail to keep up when traffic gets too heavy.

The qualifier menu offers the following choices.

1) Log error packets - if 1 is not chosen, only valid packets are counted.
2) Log valid packets - count valid packets. One or the other (or both) of items 1 and 2 must be chosen. If neither is chosen, no packets will be counted.
3) Ethernet source address filtering - count packets only if they are from one of the ethernet addresses specified. Enet addresses are entered with the bytes separated by spaces.
4) Ethernet destination address filtering - count packets only if they are to one of the ethernet addresses specified. Enet addresses are entered with the bytes separated by spaces.

5) IP source address filtering - not implemented.

6) IP destination address filtering - not implemented.

7) Length > lower bound - count only packets whose data length is greater than some lower bound. You will be prompted for the lower bound.

8) Length < upper bound - count only packets whose data length is less than some upper bound. You will be prompted for the upper bound. Note that it is possible to specify bounds in 7 and 8 that no packet can satisfy. (e.g. packets with length less than 100 bytes AND greater than 500 bytes)

9) Length in range - count packets whose length falls between an upper and lower bound. You will be prompted for the bounds.

10) Length == - count packets whose length matches the specified value.

11) Packet type - count packets with the specified ethernet packet type. You will be presented with a list of types from which to choose.

Again, multiple choices are entered on the same line, separated by spaces. When multiple choices are specified, packets must satisfy the logical conjunction of the conditions to be counted.

# Appendix D

# Experiment Configuration Package User's Manual

## D.1. Introduction

This appendix contains a discussion of the actual usage of the various programs mentioned in Chapter 3, Section 3.

## D.2. Front End

The Experimental Configuration Package (or ECP) is a front end for a set of Local Area Network performance evaluation tools. It is used to enter specifications for artificially generated network communications loads. The specification generated by the ECP are used by the Experiment Execution Package (EXP).

### D.2.1. The Structure of an Experiment Definition

An experiment definition has three parts: a list of distributions, a topology, or connectivity matrix, and a protocol selection. The goal of the definition is to produce complete descriptions of a set of packet producing processes. You will enter a list of distributions which will be used to generate interpacket times and packet lengths. These distributions are in turn used in the definition of the experiment's topology. For each machine in the network, the user will enter the characteristics of the load generating process. The distribution list and topology definition together constitute a load pattern. This load pattern can be used in several experiments with different communications protocols as determined by the protocol selection.

```
            Experimental Configuration Package

      Your choices are:
        1) Define topology and load characteristics for an experiment
        2) Select protocol
        3) Edit an experiment definition
        4) Dump a configuration file in printable form
        5) Exit

      For defining a new experiment, steps 1 and 2 should be done in order.

      Choice # ? _
```

Figure D.1: Main Menu

### D.2.2. Distribution List

To enter a list of distributions, choose entry 1 at the main menu. (Figure D.1) The first prompt is for a name for the experiment to work on. If this question has been asked previously in the current session, a choice of continuing to use that experiment will be offered. If you would like to see a list of experiments that already exist, type '?' in response to this question. Experiment names may be a maximum of 10 characters long. If the experiment you named already exists, it will be checked for completeness. If the topology definition has not yet been entered, you will automatically find yourself in the Topology definition section. If both the distribution list and the topology definition are complete for the specified experiment, you will be notified and given the choice of selecting a different name, or re-using the distribution list to create a new experiment with a different topology.

If the name you enter is not currently in use, you will be prompted for another name to identify the load pattern for the experiment. This name will be used to set up groupings of experiments with the same load pattern but different protocol selections. You will then be prompted for a comment to identify the purpose of the experiment. After this is completed, the screen shown in Figure D.2 will appear. Choose one of the listed distribution types by typing the number that appears next to the distribution name. Depending on the type of distribution you choose, you will be prompted for the parameters needed. If you choose either

```
            LOAD CHARACTERISTICS

   Time intervals between instances of packet generation and packet
lengths will be specified by distributions from the following list.
Special distributions are entered by the user.  Your choices here
form a list which will be referenced in the topology description.

Standard distributions
   1) Exponential
   2) Poisson
   3) Geometric
   4) Binomial

Special distributions
   5) General Discrete
   6) Piecewise Continuous

Choice # ? _
```

Figure D.2: Distribution List entry screen

of the user entered distributions, (distributions 5 and 6) you will be prompted for a list of x and y coordinates. This will continue until you type 'n' in response to the question 'Enter another point?' Likewise, you will continue to be prompted for distributions until you type 'n' in response to the question 'Enter another distribution?'

### D.2.3. Topology Definition

To enter a topology definition, one may choose to continue into the topology definition section immediately from the load distribution section, or topology information can be entered at a later time by choosing '1' at the main menu, then entering the name of an experiment for which the distribution list has already been entered. If the second route is taken, you will go through the same procedure as discussed in section 1.1.2. for entering the experiment name. The screen for entering the topology definition is shown in Figure D.3. Each repetition of the questions in this screen represents one process. You will be prompted for distributions from the list discussed in section 1.1.2. for interpacket time and

```
          TOPOLOGY DEFINITION

Node 1 How many processes will there be for this node?

Process 1
 (Type 'h' to see a recap of the distributions entered)
 Distribution for inter-packet time?
 Distribution for packet length?
 Number of processes receiving packets from this process:

  Receiver 1
  Process address for receiver:  node #
                         process #
  Probability that a packet goes to this process:
```

Figure D.3: Topology Definition entry screen

packet length for the packets generated by this process. You may look at a list of the distributions that you entered by typing an 'h' in response to either of these questions. You will then be prompted for the number of processes with which this process will communicate. You will then cycle through the questions for a receiver that many times. If you wish to examine the topology you are entering at any point visually, type 'v' in response to any of the questions about the receivers. The process number requested in this subsection does not correspond to the sender process on the node specified; 'sources' and 'sinks' are separate entities. If a process has more than one receiver, the probabilities of the receivers should add up to one -- currently the program does not check to be sure this is true.

### D.2.4. Protocol Selection

The protocol select section determines what communication protocol will be used for the experiment. When '2' is chosen at the main menu, you are once again asked for an experiment name. The codes and descriptions of the available protocols are displayed, (see Figure D.4) and you will be prompted for your choice. Enter the code corresponding to the protocol you wish to use for your experiment. and asked to choose the one you wish to edit.

```
                Select Protocol

  Name        Description

1) 802          Ethernet

2) VTCSMA          Virtual Time CSMA

3) Enet II       Asynchronous Collision Resolution

Which protocol would you like to use for this experiment?
```

Figure D.4: Protocol select screen

### D.2.5. Edit Functions

An experiment definition consists of three parts, and these three parts are edited independently. However, editing any section has an overall effect on the experiment. Editing either the distribution list or the topology of a definition or both creates a new load pattern. It is possible to deliberately copy in place, (overwrite) but this should be done only to correct mistakes or before an experiment has actually been run using the definition. If both the distribution list and the topology of a definition are to be edited, the first section edited should be copied to create a new experiment and thus a new load pattern, while the second should be overwritten, to keep a third experiment from being created. After you have entered the name of the experiment you wish to edit, it will be checked to see which of its sections are complete and thus are eligible for editing. You will be presented with a list of these sections,

#### Editing the Distribution List

After you have chosen to edit the load distributions, a menu will appear offering several ways to edit the list. If you already know which member of the list you want to edit, choose '1', Edit a specific entry. You will be prompted for the number of the distribution you want to edit. This is the number that appears beside the distribution in a dump of the experiment, which will be discussed in section 1.1.7. The specified entry will be printed, along with a list of possible values for the distribution code field. The question:

[E]dit entry, [D]elete entry, [Q]uit?

will appear, and you will be prompted for input. Type the letter in brackets from the choice you want. Typing 'e' causes the cursor to move beside each field in the distribution record in turn. If you wish to change the value in that field, type the new value beside the old one. If you want to leave the old value unchanged, simply type a carriage return. Typing 'd' causes the displayed distribution to be removed from the list. Typing 'q' returns you to the edit menu without making any changes. If the distribution chosen was one of the special distributions, the question will also contain the option '[S]can points.' Typing 's' lets you see the x and y coordinates of one point at a time, and change them if you wish.

The second choice at the edit menu allows you to scan through the list of distributions one at a time. You will be asked for an entry at which to begin scanning. Enter an integer corresponding to the element of the list you wish to see first. The requested entry will be displayed, and the following question will appear:

[N]ext entry, [E]dit this entry, [Q]uit scanning?

Typing 'n' will display the next entry in the list. Typing 'e' will cause the editing process discussed above to be invoked on the current entry. Typing 'q' exits back to the edit menu.

At the distribution edit menu you may also choose to add a new distribution. If you choose this option, the data entry process for load distributions (as discussed in section 1.1.2.) will be invoked. The choices you enter will be appended to the end of the distribution list. The new entry will not immediately be available for editing. If changes need to be made to a distribution entered in this way, you must choose 4 (exit) at the edit distribution menu, then begin the edit process again by choosing 3 (edit) at the main menu. This will merge the changes entered in the last edit session with the existing distributions and make them available for editing subsequently.

**Editing a Topology Description**

A topology record consists of two distributions chosen from the list entered in the load distributions section. The same editing options are available for topology descriptions as for distribution lists. In addition, there is also an option which allows you to examine the topology you have built visually .

In order to edit a particular entry, (choice 1 in the edit topology menu) you need to know what node the process is defined on, and its process number on that node. At present, valid node numbers are 1 through 4. When these have been correctly entered, the process number and the two distributions are displayed. The following question will be displayed:

[E]dit process,[D]elete proc,[Q]uit?

Typing 'e' causes the cursor to move to each of the editable values in turn. To leave the value as it appears, type a carriage return. To change the value, type the new value beside the old value. The number of receivers is not an editable value; it is changed automatically if receiver records are deleted or added. Typing 'd' removes the process from the experiment.

Choosing option 2, scan process records, at the edit topology menu prompts you for a node and process number. If you just want to scan all of the processes on a node, type the node number, then a carriage return for the process number. The specified process is displayed, and the question

[N]ext process,[E]dit this process,[Q]uit?

appears. The actions associated with these options correspond to the ones discussed above.

Choosing the option to add a new process at the edit topology menu prompts you for the number of the node you want the new process to reside on. You are then taken through the data entry screen discussed in section 1.1.3 for one process. As before, this process entry will not be immediately available for editing. Exiting to the main menu, then choosing edit will remedy this.

The view topology option allows you to visually display the topology of the current experiment. A screen in built with columns of integers for each node representing the process which resides there. When you a finished viewing the topology, type 'q' and a carriage return to quit and go back to the edit topology menu.

### Editing Protocol Select

The protocol selection of an experiment is comprised of a single integer. Editing the protocol selection is, therefore, very simple. The current selection is displayed, along with a list of available protocols. This is the same list that is displayed in the protocol select process, discussed in section 1.1.4. You will be prompted for a choice. If you change you mind about editing the selection, simply type a carriage return and the selection will remain as is. To change the selection, choose a different protocol from the list and type its number. As in the other two edit sections, you will be presented with the possibility of overwriting the current experiment, or copying to create a new experiment. If you choose to copy, you will be prompted for a name for the new experiment. By typing '?' in response, you will be shown a list of existing experiments. When you have typed in the new name, the program will create a copy with the new protocol selection, and link it to the appropriate directories.

## D.2.6. Caveats

In the load distributions and topology sections, the functions allow all distributions to be removed. This situation is undesirable, as it would render an experiment unusable. Great care should be taken if the distribution list is edited after the topology for an experiment has been entered to insure that the new distributions have the desired meaning in the context of the choices made in the topology description.

The overwrite option should be used only to correct mistakes or make changes in an experiment definition only if the experiment has not yet been run. Overwriting an experiment that has already been run may cause inconsistencies in the analysis phase of the experiment.

If, in creating a new experiment by editing an old one, it is necessary to edit both the load and topology sections, change the load section first and copy it to a new experiment, then change the topology section and overwrite it. Doing this in the opposite order is also permissible.

### D.2.7. The Dump Utility

If you choose 4 (dump an experiment description) at the main menu, you will be prompted for an experiment name as before. When the name of an existing experiment has been entered, that experiment will be examined and a list of completed sections will be displayed. If you want a dump of the entire experiment description, type 'y' in response to the question. If you only want to see one or two sections, type a carriage return in response, then, as the cursor is moved to each section in turn, type 'y' by the ones you want to include in the dump, and carriage return by the ones you do not want to see. The dumpfile will be created with a pathname of the form

```
experiment/<exp name>/<exp name>.dmp
```

which can be viewed either on the screen using cat or vi, or sent to a printer.

### D.2.8. The Protocol Selection Table

When the ECP is first invoked, the file containing the list of available protocols does not exist. To build this table, type 'A' at the main menu choice. You will be prompted for an extension, which must be no longer than 8 characters. This will correspond to the name of the subdirectory in the protocol directory to which experiments with this protocol selection will be linked. You will then be asked for a description of this protocol. It must be no longer than 70 characters. This will serve to explain the purpose of the protocol with the extension entered before. These will appear in the protocol select and protocol select edit sections. Typing 'A' permits one additional protocol to be entered, so to initially build this file, procedure must be done repeatedly.

# Bibliography

[Abra70] Abramson, N., " The ALOHA System -- Another Alternative for Computer Communication," *Proceedings of the Fall Joint Computer Conference,* 1970, pp. 281 - 285.

[Abra87] Abrams, Marc, " Design of a Measurement Instrument for Distributed Systems," IBM Zurich Research Laboratory Research Report RZ 1639, October 30, 1987.

[AL79] Almes, G. T., and Lazowska, E. D., "The Behaviour of Ethernet-like Computer Communication Networks," *Proceedings of the 7th Symposium on Operating Systems Principles,* Asilomar, California, December 1979, pp. 66 - 81.

[Amer82] Amer, P.D., " A Measurement Center for the NBS Local Area Computer Network," *IEEE Transactions on Computers,* Vol C-31, No. 8, Aug. 1982, pp. 723-729.

[AK85] Amer, P.D. and Kumar, R.N., "Local Area Broadcast Network Measurement Part I -- Measurement Center Design and Implementation," University of Delaware Department of Computer and Information Sciences, Technical Report No. 85-3, April 1985.

[AKKPC86]Amer, P.D., Kumar, R.N., Kao, R., Phillips, J.T., and Cassel, L.N, "Local Area Broadcast Network Measurement: Traffic Characterization," University of Delaware Department of Computer and Information Sciences, Technical Report No. 86-12, January 1986.

[AS84] Arthurs, E. and Stuck, B.W., "A Modified Access Policy for ETHERNET Version 1.0 Data Link Layer," *IEEE Transactions on Communications,* Vol COM-32, No. 8, August 1984, pp. 977-979.

[BM87] Barnett, B. Lewis, III and Molloy, Michael K., "ILMON: A Unix Network Monitoring Facility," *Proceedings of the Winter 1987 USENIX Technical Conference,* Washington, D.C., January 21 - 23, 1987, pp. 133-144.

[BG87] Bertsekas, D. and Gallager, R.G., *Data Networks,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987, pp. 229-238.

[BMK88] Boggs, David R., Mogul, Jeffrey C., and Kent, Christopher A., "Measured Capacity of an Ethernet: Myths and Reality," *Proceedings of the SIGCOMM 1988 Symposium on Communications, Architectures, and Protocols,* Stanford, California, August 16 - 19, 1988, pp. 222 - 234.

[Bost88] Bostic, Keith, private communication, November 1, 1988.

[Cape79] Capetanakis, J., "Tree Algorithms for Packet Broadcast Channels," *IEEE Transactions on Information Theory,* Vol. IT-25, September 1979, pp. 505-515.

[CL83] Coyle, Edward J. and Liu, Bede, "Finite Population CSMA/CD Networks," *IEEE Transactions on Communications* Vol. COM-31, No. 11, November 1983, pp. 1247 - 1251.

[DIX82] *The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications (Version 2.0),* Digital Equipment Corporation, Intel Corporation, Xerox Corporation, November 1982.

[Exce85] *Nutcracker User Manual,* Excelan, Inc., San Jose, California, 1985.

[Ferr78] Ferrari, D., *Computer Systems Performance Evaluation,* Prentice-Hall, Inc., 1978, pp. 66-67.

[Ferr84] Ferrari, D., "On the foundations of artificial workload design," *Proceedings of the 1984 SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* Cambridge, MA, August 1984, pp. 8 - 14.

[FW81] Fields, J.A. and Wong, J.W., "An Analysis of a Carrier Sense Multiple Access System with Collision Detection," University of Waterloo, Technical Report CCNG E-Report E-95, May 1981.

[Frat83] Fratta, L., "An improved access protocol for data communication bus networks with control wire," *Proceedings of the ACM SIGCOMM Symposium*, Austin, TX, March 1983.

[Gall78] Gallager, R.G., "Conflict Resolution in Random Access Broadcast Networks, " *Proceedings of the AFOSR Workshop on Communication Theory Applications*, Provincetown, MA, September 17-20, 1978, pp. 74-76.

[Gons85] Gonsalves, Timothy A., " Performance Characteristics of 2 Ethernets: an Experimental Study," *Proceedings of the 1985 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, August 1985, pp. 78 - 86.

[GT88] Gonsalves, Timothy A., and Tobagi, Fouad A., "On the Performance Effects of Station Locations and Access Parameters in Ethernet Networks," *IEEE Transactions on Communications*, Vol. COM-36, No. 4, April 1988, pp. 441 - 449.

[IEEE82] " IEEE Project 802, Local Area Network Standards," Draft IEEE Standard 802.3 CSMA/CD Access Method, Draft D, 1982.

[Inte82] *NI1010 UNIBUS Ethernet Communications Controller User Manual*, Interlan, Inc., Chelmsford, Massachusetts, 1982.

[Jaco88] Jacobson, Van, "4BSD TCP Ethernet Throughput,"
USENET comp.protocols.tcp-ip posting, message I.D.
8810242033.AA29183@helios.ee.lbl.gov, October 24, 1988.

[JL87] Jain, P. and Lam, S.S., "Modeling and Verification of Real-Time Protocols for Broadcast Networks," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 8, August 1987, pp. 924-937.

[JL88] Jain, P. and Lam, S. S., " Specification and Verification of Collision-Free Broadcast Networks," *Proceedings of the SIGCOMM Symposium on Communications Architectures and Protocols*, Stanford, California, August 16 - 19, 1988, pp. 282 - 291.

[JL89] Jain, P. and Lam, S.S., University of Texas Department of Computer Sciences technical report (in preparation).

[Humb86] Humblet, Pierre A., "On the Throughput of Channel Access Algorithms with Limited Sensing," *IEEE Transactions on Communications*, Vol. COM-34, No. 4, April 1986, pp. 345 - 347.

[Kern82] Kernighan, B. W., *PIC - A Graphics Language for Typesetting*, revised edition, March 1982. (online UNIX documentation)

[KK81] Kiesel, W.M. and Kuehn, P.J., "CSMA-CD-DR: A New Multi-access Protocol for Distributed Systems," *Proceedings of National Telecommunications Conference 1981*, Dec. 1981, pp A2.4.1-A2.4.6.

[KT75] Kleinrock, L. and Tobagi, F., "Packet Switching in Radio Channels: Part I -- Carrier Sense Multiple-Access Modes and their Throughput-Delay Characteristics," *IEEE Transactions on on Communications*, Vol. COM-23, No. 12, December 1975, pp. 1400-1416.

[Lam80] Lam, Simon S., " A Carrier Sense Multiple Access Protocol for Local Networks," *Computer Networks*, 4, 1980, pp. 21 - 32.

[Lam88] Lam, Simon, private communication, December 7, 1988.

[Lewi89] Lewis, John M., " Bandwidth Utilization of a Large Local Area Network," *IEEE Communications Magazine*, Vol. 27, No. 2, February 1989, pp. 25 - 30.

[LW87] Liu, Y.-C. and Wise, G. L., "Performance of a CSMA/CD Protocol for Local Area Networks," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-5, No. 6, July 1987, pp. 948 - 955.

[Mass80] Massey, J., "Collision Resolution Algorithms and Random-Access Communication," UCLA Technical Report UCLA-ENG-8016, 1980.

[ML85] Meditch, J. S. and Lea, C. T. A., " Stability and Throughput in Virtual Time CSMA," *Computer Networks and ISDN Systems*, Vol. 10, 1985, pp. 19 - 26.

[MB76] Metcalfe, R. and Boggs, D., "Ethernet: Distributed Packet Switching for Local Networks," *Communications of the ACM*, Vol. 19, No. 7, July 1976, pp. 395-

404.

[MC83] Minnich, N.M. and Cotton, C.J., "An Evaluation of Two Unibus Ethernet Controllers," *Proceedings of the 8th Conference on Local Computer Networks,* Minneapolis, Minnesota, Oct. 17-19, 1983, pp. 29-36.

[Molle83] Molle, M., "Asynchronous Multiple Access Tree Algorithms," *Proceedings of SIGCOMM 83 Symposium on Communications Architectures and Protocols,* ACM #533830, March, 1983, pp 214-218.

[MK85] Molle, Mart L. and Kleinrock, Leonard, " Virtual Time CSMA: Why Two Clocks are Better than One," *IEEE Transactions on Communications,* Vol. COM-33, No. 9, September 1985, pp. 919 - 933.

[MSV87] Molle, Mart L., Sohraby, Khosrow, and Venetsanopolous, Anastasios N., " Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-5, No. 6, July 1987, pp. 956 - 968.

[Moll83] Molloy, Michael K., " Experimental Evaluation of New CSMA Protocols," *Proceedings of the National Communications Forum,* October 24 - 26, 1983, pp. 350 - 354.

[Moll85] Molloy, Michael K., " Collision Resolution in an Unslotted Environment," *Computer Networks,* Vol. 9, No. 3, March 1985, pp. 209-214.

[Moll86] Molloy, Michael K., "Living on the Leading Edge of Ethernet Use," *Proceedings of the 11th Conference on Local Computer Networks,* Minneapolis, Minnesota, October 6 - 8, 1986, pp. 124 - 131.

[Nabi84] Nabielsky, Jose, "Interfacing to the 10Mbps Ethernet: Observations and Conclusions," *Proceedings of the ACM SIGCOMM Symposium on Communications Architectures and Protocols,* June 6-8, 1984, Montreal, Quebec, Canada, pp. 124 - 131.

[Net87] *IEEE Network,* Special issue on LAN Test Tools and Performance Measurements, Vol. 1, No. 3, July 1987.

[OL82] Owicki, S. and Lamport, L., "Proving Liveness Properties of Concurrent Programs," *ACM Transactions on Programming Languages and Systems,* Vol. 4, No. 3, July 1982, pp. 455-495.

[PTW85] Panwar, S.S., Towsley, D. and Wolf, J.K., "On the Throughput of Degenerate Intersection and First-Come First-Served Collision Resolution Algorithms," *IEEE Transactions on Information Theory,* Vol. IT-31, No. 2, March 1985, pp. 274-279

[Post80] Postel, J., "User Datagram Protocol," RFC 768, Information Sciences Institute, August 1980.

[RP85] Reynolds, J. and Postel, J., "Assigned Numbers," RFC 943, Information Sciences Institute, April 1985.

[RG85] Rios, M. and Georganas, N. D., "A Hybrid Multiple-Access Protocol for Data and Voice-Packet Over Local Area Networks," *IEEE Transactions on Computers,* Vol. C-34, No. 1, January 1985, pp. 90 - 94.

[SH80] Shoch, J. F. and Hupp, J. A., "Measured performance of an Ethernet local network," *CACM,* vol. 23, Dec 1980, pp. 711-721.

[TK85] Takagi, Hideaki and Kleinrock, Leonard, "Throughput Analysis for Persistent CSMA Systems," *IEEE Transactions on Communications,* Vol. COM-33, No. 7, July 1985, pp. 627 - 638.

[TH80] Tobagi, F. and Hunt, J.A., "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," *Computer Networks,* Vol. 4, 1980, pp. 245 - 259.

[Toba80] Tobagi, F., "Multiaccess Protocols in Packet Communication Systems," *IEEE Transactions on Communications,* Vol COM-28, No. 4, April 1980, pp 468-488.

[TBF83] Tobagi, F., Borgonovo, F., and Fratta, L., "Expressnet: A High-performance integrated services local area network," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-1, Nov. 1983, pp. 898 - 912.

[TT77] Tokoro, Mario and Tamara, Kiichiro, "Acknowledging Ethernet," *Proceedings of COMPCON 77*, Sept. 1977, pp. 320-325.

[TV82] Towsley, D. and Venkatesh, G., "Window Random Access Protocols for Local Computer Networks," *IEEE Transactions on Computers,* Vol C-31, No. 8, August 1982, pp. 715-722.

## VITA

Benjamin Lewis Barnett III was born in Spartanburg, South Carolina, on August 2, 1959, the son of Annalyne Hall Barnett and Dr. Benjamin Lewis Barnett, Jr. He graduated *Magna Cum Laude* from Furman University in Greenville, South Carolina, in 1981. For one year after graduation, he was employed as a Programmer/Analyst by the Clinical Computing Laboratory, a part of the Department of Internal Medicine of the Medical College of the University of Virginia. He entered the Graduate School of the University of Texas, Department of Computer Sciences, in Fall of 1982. While attending the University of Texas, he has been employed as a consultant by Advanced Computer Engineering of Austin, Texas and as a Software Engineering intern by National Instruments of Austin, Texas. He received the Master of Science in Computer Science degree from the University of Texas Department of Computer Sciences in May of 1988. His studies at the University of Texas were supported in part by National Science Foundation CER Grant #MCS8122039.

Permanent address: 2406 Northfields Road
Charlottesville, Virginia
22901

This dissertation was typed by the author.