ON DETERMINING PERFORMANCE OF A LOCALNET 20 NETWORK

Nancy Schult
University of Virginia

Computer Science Report No. RM-86-02
April 8, 1986

ON DETERMINING PERFORMANCE OF A LOCALNET 20 NETWORK

Nancy Schult

April 1986

NSWC Contract No. N60921-81-K-A370

1. INTRODUCTION

A major task for local area network administrators is determining the network's performance and then accurately predicting it under different circumstances. To do this, careful testing and evaluation of the network first needs to be done under both normal and user-defined conditions [AMER83]. In order to predict performance, an analytic or simulation model can be developed and calibrated with the measurements obtained from a given workload. With the calibrated model, the impact of different parameters can be examined in order to predict network performance under varying conditions.

This study addresses the task of determining and predicting performance of the *LocalNet** 20 network at NSWC. In section two, the topic of network measurements is discussed. Section three describes the implementation of a CSMA/CD analytic model defined by Tobagi and Hunt [TOBA80] (see Appendix A), developed for the purpose of evaluating a *LocalNet* channel. In the fourth section, future direction and efforts in evaluating *LocalNet* are discussed, and finally, different papers of interest and the model program listing are included as appendices.

2. MEASUREMENTS

The topics covered in this section include the considerations that need to be addressed in setting up network measurements, the issues involved in designing measurement environments, and some case studies described in the literature. Also discussed are the measurements that can currently be made on the *LocalNet* network.

2.1. Considerations

In setting up network measurements, the first step is to determine what to measure [AMER82]. The ideal situation is to have a measurement center that records

* LocalNet is a Trademark of Sytek, Inc.

all possible information in order to avoid having to redesign the measurement tools and having to rerun the tests at a later time. Unfortunately, the amount of storage required to handle this information is huge. As a result it becomes necessary to first decide what questions need to be addressed by these measurements. Some potential questions include the following:

- What problems have surfaced in the daily network operations that need to be evaluated?
- How much and what kind of traffic can the network handle? What are the impacts on performance?
- Does the network traffic need to be characterized for later analytic or simulation purposes, and if so, what information is necessary?
- Could the network be reconfigured to perform "better" ?

The questions raised about the network are usually concerned with obtaining either a traffic or a performance characterization of the network over a given period of time. Ultimately both characterizations are goals to be addressed by the measurements. For a traffic characterization, the aim is to describe the workload that has been placed on the network over the measurement period which has resulted in a particular performance measure [AMER82]. If the traffic is generated by artificial means, the workload is already known. The results of characterizing traffic can then be used as inputs to an analytic or simulation model of the network or subnetwork. Also, if it is possible to replicate the traffic at individual nodes (using artificial traffic generators), then with a change in network configuration, the same nodal traffic patterns can be rerun and results compared to the original ones. A discussion of workload or traffic characterization is found in [FERR83].

A performance characterization consists of the performance results obtained from a given load and network configuration. Some metrics include time delays, such as

mean waiting time of packets, and channel utilizations. These metrics can be used to aid in calibrating an analytic or simulation model after feeding it the inputs found by characterizing the traffic.

Once the questions have been decided, the next step is to determine what network information is necessary to satisfy these questions. If a measurement goal is to characterize traffic, it would be advantageous to gather data about the frequencies of different kinds of packets over a period of time and their respective sizes. On the other hand, if a measurement goal is to characterize performance, one might want to keep track of the channel acquisition time of each packet generated.

Finally, once the information required has been defined, the methods for gathering this information need to be determined. Can the necessary information be gathered by a central monitor on the channel or only by the individual nodes? Information that can be obtained from a packet's header can usually be gathered by a central monitor, while other items such as waiting time or channel acquisition time of packets need to be gathered at individual nodes. This consideration is further discussed in the next section.

2.2. Design Issues of Measurement Environments

There are four areas that need to be addressed in the design of a measurement environment: artifact, the location of the measurements in the network, artificial traffic generation, and off-line versus on-line analysis. The following discussion on these areas is similar to that presented by Amer [AMER82] (see appendix B). The topic of measurement environment design has also been discussed in other publications [AMER83] [STOK80] [SHOC80] [GONS85] [FERR83].

Artifact is defined as the amount of interference on the system being measured, due to the monitoring device(s). Ideally, this is a quantity to be minimized. If there is any artifact resulting from measurements, its magnitude should be calculated

in order to determine the impact on measurements taken. The location of the network measurements, discussed next, has an effect on the amount of artifact in the network.

In considering where to *locate the measurement gathering activities*, there are three approaches to evaluate: centralized, decentralized, and hybrid. With the centralized approach, there is a single measurement device or interface that observes the activities on the channel. Information found in packet headers can be obtained, in addition to channel performance measurements such as channel busy periods, channel utilization, and the interarrival times of packets onto the channel. This is the method of measurement gathering implemented on Sytek's *LocalNet* [SYTE83a].

There are several advantages and disadvantages to a centralized approach. One advantage is the absence of artifact on the network. The centralized approach is also less costly, since only a single monitoring device is needed and no additional hardware or wires are required. The main disadvantage is that not all necessary information can be obtained with this approach. The centralized approach is very useful when the network is centrally controlled, but the CSMA/CD network is distributed and thus the information available only at each interface or node is not collected. For example, the amount of time a given node defers transmitting a packet is a value that a centralized monitor cannot determine. Another disadvantage to a centralized approach is that some of the timings obtained may be biased due to the propagation delays incurred between a node and the monitor.

With the decentralized approach, additional memory and real-time clocks usually need to be incorporated at each node or interface in order to gather the nodal measurements. The advantage to decentralized measurements is that most information is now available for evaluation. The tasks that a central measurement center would do have now been distributed to each node or interface, with the

individual node statistics collected as well. However, the decentralized approach is more expensive than the centralized method, due to the additional memory and clocks needed. There are also the potential problems of clock synchronization and the possible overhead or artifact introduced if it is necessary to transfer data to a central location during the measurement period.

The hybrid approach is a compromise between the centralized and decentralized methods. With this approach, all possible data is collected centrally; however, at each user node or interface, minimal modifications are made in order to gather some measurements locally. The advantage of a hybrid method is that accurate and comprehensive measurements can be obtained which are not as costly as a decentralized approach. However, a certain amount of complexity has been added in coordinating the decentralized and centralized analysis [AMER82]. It may be desired to periodically consolidate information at a central location during the measurement period, which may introduce artifact if sent over the network, or additional wiring if not. Variations of the hybrid approach have been implemented by [AMER82], [SHOC80], and [GONS85].

Artificial traffic generators is another area to be considered when designing a measurement center. One reason for including them is for the purpose of "stressing" the network. A network's performance is usually not significantly stressed under "normal" operating conditions [AMER83], [METC76], [SHOC80]. As a result, in order to determine the network's performance at heavy loads, it becomes necessary to generate these loads artificially. A second reason for including artificial traffic generators is for the added capabilities of producing repeatable traffic patterns and knowing exactly the characterization of traffic. These allow for the comparison of a network to either another network or to an analytic or simulation model with identical workloads defined. Artificial traffic generators have been implemented as part of test

programs defined at different nodes [SHOC80] [GONS85], and as a microcomputer system [STOK80].

The fourth area to be addressed is whether to do *off-line or on-line analysis* of measurements. In off-line analysis, measurements are stored and evaluated after the measurement period has completed. On-line measurements analyze and summarize measurements on a display during the measurement period. Off-line analysis is simplier to accomplish but requires more storage, while on-line analysis is more complicated and may introduce artifact.

2.3. Reported Network Measurements

Three examples of measurement studies on CSMA/CD-based networks are discussed in this section. For each study, the method of measurement gathering, the experiments conducted (if known), and its goals are presented. For the actual measurement and performance results obtained, the reader is referred to the respective papers.

Shoch and Hupp Study

This study investigated characterizing both the traffic under normal network operations and the performance for high loads on an Ethernet network. A hybrid approach was used to gather measurements. To aid in characterizing normal traffic, most of the data was collected by a central monitor or interface. Among the data gathered for this purpose included network load variations over different time periods, source-destination traffic patterns, inter-packet arrival times to the channel over a given time period, packet latency, and overhead. A summary of the actual data obtained is found in their paper [SHOC80].

To investigate performance for high loads and for the special case of having continuously queued sources, artificial traffic generators were used to create the

network load. Each generator was actually part of a test program that had been sent to idle machines on the network by a special control program. Statistics on successful packets were collected at each machine and retrieved by the control program upon termination of the measurement period. Measurement tests were performed only at night during low network usage. This method was also used by Gonsalves [GONS85].

Performance was evaluated for both high loads and continuously queued sources in terms of channel utilization, stability, and fairness. Channel utilization or throughput is defined as the fraction of total time that good packets are being transmitted on the network [SHOC80]. The stability of a network is concerned with network's ability not to halt or deadlock due to an increasing offered load. Ideally the dynamic control algorithms are supposed to provide stability as the load increases. A discussion of stability and ways to evaluate it can also be found in the Toense study [TOEN83], which is the next study presented. Finally, fairness deals with the way a channel is shared. In a non-prioritized network, each host should have an equal opportunity to use the channel. Methods on how to measure fairness are described in various papers [SHOC80] [TOEN83] [GONS85].

Toense Study

In his report on the performance study of NBSNET, Toense describes the measurement experiments performed, as well as the evaluation of these measurements [TOEN83] (see Appendix C). To gather the necessary measurements for characterizing performance, a hybrid approach was employed. For the experiments six traffic generators (representing nodes) were used which contained identical packet length and interarrival time distributions. Packet lengths were constant values and varied over the sets of tests.

The measurements gathered were the result of two classes of experiments. In the first class, the packet interarrival times at each node were exponentially distributed, while in the second, continuously queued conditions existed at each node. For the first class of experiments, the interarrival times were varied for a given packet length in order to achieve a wide range of offered loads on the network. In the second class, there was always a packet waiting at each queue to be transmitted; the number of traffic generators active provided the range of offered loads.

Performance of the network was evaluated in terms of stability, channel acquisition delay, throughput, and fairness. For each of these, the impact of different packet sizes was investigated. The data obtained from the measurements was further analyzed by a special high-level language that included, among other things, the abilities of non-linear fitting and data analysis.

Gonsalves Study

Two Ethernet networks with different transmission rates were evaluated in this study, with the goal of characterizing their performances from high to very high offered loads [GONS85] (see Appendix D). To gather the necessary measurements, a hybrid approach was employed that is similar to the method used by Shoch and Hupp [SHOC80]. In order to generate the high loads needed for evaluation, artificial traffic generators were used that were part of test programs sent by a control program to idle machines. All nodes or machines had identical behavior in the measurement tests. Packet lengths were fixed for a given set of tests, and interarrivals were uniformly-distributed. However, both packet lengths and interarrivals varied among sets of tests in order to obtain different network offered loads.

Performance of each network was evaluated in terms of throughput, fairness, and channel utilization. Among the items investigated were the effects of varying

packet length and transmission rates on throughput, mean delay, and the delay distribution. Also studied was the effectiveness of various analytic models in predicting network performance; these included models by Metcalfe and Boggs [METC76], Lam [LAM80], and Tobagi and Hunt [TOBA80].

2.4. LocalNet 20 Measurements

The *LocalNet* has been described as "a collection of multichannel broadband local networks connected by point-to-point links" [ENNI83]. It is discussed in varying levels of detail in a number of papers [ENNI83] [BIBA81] [DINE81] [DINE80] [SYTE83a] [SYTE83b] [SYTE83c] [SYTE83d]. For conducting measurement tests on the *LocalNet* 20, our general goals are to obtain traffic and performance characterizations. Of particular interest with respect to performance is determining if the network becomes unstable, and if so, determining when it occurs. In order to make an evaluation of stability, utilization needs to be investigated over a variety of offered loads, in particular high loads. Some sort of artificial generator is needed to create the desired loads, a capability that unfortunately *LocalNet* currently does not have.

To aid in collecting the measurement data, the *LocalNet* 50/120 Statistical Monitor can be utilized. This monitor is hooked up to the network at some point in order to eavesdrop on a channel [SYTE83a]. A second possible method of data collection, the NCC program REDUCE [ENNI85], is not used here due to a lack of available information. In this section, a brief description of the statistical monitor is first given along with the data that it collects, followed by a discussion of tests that can presently be executed on the *LocalNet* 20. Additional tests that cannot be performed at the present time but would be useful are addressed in section four.

Statistical Monitor

The *LocalNet* 50/120 Statistical Monitor is an example of the centralized approach to measurement gathering. Its purpose is to provide a snapshot of a given channel's activities for a specified time period which can range from one to sixty minutes in length. However, like most centralized monitors, it does not gather any information that is available at the individual nodes or PCUs, such as the number of transmissions by a given node over a time period, the mean channel acquisition delay of packets at a node, and interarrival times of packets. A description of the monitor and its capabilities can be found in its Preliminary User's Manual [SYTE83a].

From the documentation it appears that only one type of report is generated, with the majority of its data consisting of various counter results. The counters reflect values obtained during the current time period, the peak time period, and the cumulative time since the counters were last initialized using the "reset" command. Some of the existing counts include the number of packets on and off the channel, number of packets aborted (estimated), number of bad packets detected, the total number of characters and the number of data characters, and the number of sessions on and off the channel. Also presented in the report are the current, peak, and cumulative channel utilizations, five packet type totals for the current time period, and a packet size histogram that reflects packets transmitted since the last "reset" command. The results are displayed in either a short or verbose form.

There are a few items that need to be considered about the data obtained by the monitor. First of all, there is no overflow detection of counter values. The network administrator needs be aware of this when determining the time length of a measurement period. Another item to be considered is that there are only five packet types monitored, when there are nine packet types defined in the literature [ENNI83]. Not included in this group is the data packet type. A potential assumption

to make about data packets is that any packet containing user characters is a data packet, and thus the number of data packets can be obtained by summing the counts in the packet size histogram. However, the histogram evaluates data packets over the cumulative period, not just for the current period. One possible way to get the number of data packets for the current period with the above assumption is to initialize all counters by executing the "reset" command before each report period; another way is to not discard the previous report in order to compare its results to the current report and determine the impact of the given time period.

Current Measurements

Measurements obtained using the Statistical Monitor can be applied to characterizing "normal" traffic behavior on a given channel. The purpose of our study is to obtain an idea about typical network behavior on different channels of the *LocalNet* 20. Not investigated though are the actual source-destination values of each packet on a given channel. To examine a *LocalNet* channel, measurement experiments can be divided into two sets, with one set addressing the channel load over the course of a day and the other set looking at the channel during the same time period over a few days. These sets of experiments can be repeated for each channel of the network.

In the first set of experiments, the focus is on the nonhomogeneous traffic behavior of a given channel over the period of a day. For these experiments, the channel should be observed at least over the course of a working day, if not for the whole 24 hours. The total time period needs to be broken up into a series of measurement intervals. At the end of each interval, a report is generated that reflects the traffic and its impact on the network for that interval. At the beginning of the total time period, all counters are initialized; whether they need to be reinitialized before each measurement period in order to prevent counter overflow is

a decision for the network administrator to make. With the sum of the measurement reports generated, trends in behavior can be determined over the total time period.

The measurement interval for this set of experiments needs to be small enough to detect general trends, but not too small to detect every fluctuation. This interval should also not be so large that certain trends go undetected or cause counters to overflow. A recommended interval is six minutes, the same used by Shoch and Hupp [SHOC80].

For the second set of tests, the network is observed at a fixed time each day over a number of days. The goal of these tests is to first observe in more detail the network fluctuations and then to see if the day-to-day behavior is indeed similar. The resulting traffic patterns should be more homogeneous in nature than those observed in the first set of experiments. The total evaluation period and measurement interval defined are both shorter than those used in the first set. Recommended intervals are a six minute total evaluation period, with a measurement interval of one minute (the smallest interval possible with the Statistical Monitor). It is not critical as to the time of day these experiments occur, only that they happen at the same time each day. However, it would probably be advantageous to investigate one of the busier periods. Right before the evaluation period begins, the counters need to be initialized using the "reset" command. After each measurement interval, a report should be generated and saved for later evaluation. Determining whether it is necessary to reinitialize the counters in order to avoid overflow is a decision for the network administrator to make.

3. CHANNEL EVALUATION TOOL

An analytic model implementation for evaluating performance aspects of a *LocalNet* channel is discussed. The model implemented is one of two presented by Tobagi and Hunt for analyzing the performance of CSMA/CD protocols [TOBA80]. In this section, an overview of both Tobagi and Hunt models is first given, followed by a discussion of the model implementation.

3.1. Tobagi and Hunt Models

In their paper Tobagi and Hunt describe two stochastic analytic models [TOBA80]. The first model is used to evaluate channel capacity, while the second is used for throughput-delay analysis. Both models are described in this section, although only one, the more difficult second model, has been implemented. For both models the time axis is assumed slotted, with one slot time equal to the end-to-end propagation delay of the channel. All devices are synchronized and can only begin transmission at the start of a slot. Both models evaluate fixed-sized and variable-sized packets. A more in-depth discussion of these models can be found in the Tobagi and Hunt paper in Appendix A.

An infinite population model is used to evaluate channel capacity. It is assumed that all devices on the channel together form a single independent Poisson source and that the average retransmission delay is arbitrarily large. Every packet generated from the single source becomes part of the group of packets competing for the channel [CHLA85]. Using this model non-persistent and 1-persistent CSMA/CD protocols are evaluated. For both of these protocols the time axis is observed to be an alternating sequence of busy and idle periods. A cycle is composed of a single busy period followed by an idle period. Since an infinite population of users is assumed, all cycles are statistically identical. Using the assumption that the input source is Poisson, expressions for the length of successful and unsuccessful busy

periods are determined, as well as for the average idle period, average transmission period, throughput, and channel capacity. Gonsalves uses this model to determine channel capacity for his evaluations [GONS85].

The second Tobagi and Hunt model is used for delay analysis of CSMA/CD. Only the non-persistent CSMA/CD protocol is investigated using a "linear feedback model" [TOBA80]. A finite population of M devices is assumed with each device in either a thinking or backlogged state. In the thinking state a device generates a packet in a given slot with probability σ and transmits it, if the channel is sensed idle, in the same slot. In the backlogged state the packet already generated by a given device has either had a channel collision or been blocked due to a busy channel. The probability that a backlogged user senses the channel in the current slot is denoted by ν; this is equivalent to saying that the rescheduling delay of backlogged packets is geometrically distributed with mean $1 / \nu$ slots. Note that each device has the same probabilities defined.

In the delay analysis it is assumed that M, σ, and ν are invariant. The random variable N^t is defined as the number of backlogged devices found at the beginning of slot t. An embedded Markov chain is next defined at the first slot of each idle period. For this chain, the stationary probability matrix $\overline{P}$ between consecutive embedded points needs to be determined, which is the product of several single-slot matrices.

Once the matrix $\overline{P}$ is obtained, the next step is to determine the stationary probability distribution of N^t, $\Pi = [\pi_0, \pi_1, ..., \pi_M]$, at the embedded points. This $1 \times (M+1)$ matrix can be obtained by solving the system of equations $\Pi = \Pi \overline{P}$, with the constraint that $\sum_{i=0}^{M} \pi_i = 1$. Upon obtaining Π regenerative process properties are used to compute average stationary channel throughput (S), average channel backlog (N), average packet delay (D), and average waiting time or channel

acquisition time of a packet (W). Toense looks at this model in investigating channel acquisition delay for NBSNET [TOEN83].

3.2. Model Implementation

The "linear feedback model" presented by Tobagi and Hunt [TOBA80] has been implemented in software for *LocalNet* analysis. A listing of the Pascal program *model* is found in Appendix E. In this section, the general assumptions and program features, parameters, results, and usage are discussed.

General Assumptions and Program Features

The *model* program was developed for the purpose of evaluating a *LocalNet* channel. Using variable-sized packets, the impact of other network channels on the given channel is reflected in the probabilities of encountering different packet types. There are currently four packet types defined, although this quantity can easily be changed. These packets types include on-channel data, on-channel control, off-channel data, and off-channel control.

Additional assumptions include the use of a non-persistent protocol and equal priority of packets. A non-persistent protocol is used because of a comment by Edholm that "basic operational analysis done by Tobagi/Hunt for non-persistent CSMA/CD baseband can be extended with conservative interpretation to LocalNet 20" [EDHO83]. Since no information about the CSMA/CD persistency on *LocalNet* has been obtained, the assumption for non-persistency was made.

The primary goal in the design of the *model* program was to make it easy to understand upon inspection, in order to make future program modifications easier to accomplish. An effort was made to use identifiers that were either identical to those used in the Tobagi and Hunt discussion (where applicable) or descriptive. Also, the program was divided into six logical sections: global declarations, support routines,

initialization routines, input routines, calculation routines, and output routines. The calculations described by Tobagi and Hunt are performed by the calculation routines. The global declaration section lists all global constants, types, and variables. The support routines section includes all general-purpose routines that serve as an aid to the calculations. These include routines for handling different matrix operations, error handling, and computations of mathematical functions not implemented in Pascal. The initialization section assigns defaults to all input variables, while the input routines give the user the option to change them. Finally, the output routines print out the results obtained from the calculations.

Some comments need to be made on how the stationary probability distribution Π is determined in the *model* program. In this implementation, the system of equations $\Pi = \Pi \bar{P}$, with the constraint that $\sum_{i=0}^{M} \pi_i = 1$, is solved by using a Gauss elimination method with backward substitution and scaled partial pivoting [CONT80]. In the preliminary testing a problem with underflow has been detected as the number of devices (M) increases. Storage also becomes an issue as M increases. Tobagi and Kleinrock [TOBA77] address these problems in an appendix of their paper that deals with stability of CSMA. Their method of solving the system of equations is one that could be implemented at a later date.

Parameters and Results

A list of program input parameters, needed for the analytic model implementation is given along with the results obtained from the program. For each input parameter, a definition is given and a default value specified, where applicable. The variable defaults can be changed when running the program, a feature that is discussed in more detail in the next section dealing with program usage.

Constants:

BPS - The bandwidth of the channel, in bits per second. The default value is 128,000.

MAXDEVICES - The maximum number of devices attached to the channel. The default value is 30.

NUMLENGTHS - The number of different packet types (lengths) evaluated by the model. The default is 4.

INTERACTIVE - A flag indicating whether input mode is interactive or not. It is set to true, since this is the only mode currently implemented.

ROWSIZE - The number of real elements printed out in a row of output.

Variables:

M - The number of devices attached to the channel. Its default value is 20.

tau (τ) - End-to-end propagation delay, in seconds. Its default value is 0.0003, the delay for "a system with a longest cable leg of 30 km" [ENNI83]. This delay is equivalent to one slot time.

gamma (γ) - Time that all devices stop transmitting, given that a collision occurs, in slots. It is computed using τ, ξ, and ζ. The variable "igamma" is its corresponding integer value.

nu (ν) - Probability that a backlogged device senses the channel in the current slot. This is set to 0.01.

sigma (σ) - Probability that a device in the thinking state generates a new packet and, if the channel is sensed idle, attempts to transmit it. Its default value is 0.02.

xi (ξ) - Time for a device to detect interference, once the interference has reached it. It is assumed that this time is 1 bit time $\equiv 1 / BPS$.

zeta (ζ) - Period used for collision consensus reenforcement. Since no information

was found on this with respect to *LocalNet*, its default value has been set to 0.

DEBUG - Flag used to print out intermediate values computed by the program. These values are sent to the file "outdebug" ; its initial value is false.

TRACE - Flag used for determining the routines executed by the program. Its output is sent to the file "outdebug" ; its initial value is false.

length - Array containing the length of the different packet types, in bits. Its default values are 916, 108, 956, and 148, representing an on-channel data packet, on-channel control packet, off-channel data packet, and off-channel control packet, respectively.

problength - Array containing the probabilities associated with the different packet types. Its default values are 0.3, 0.5, 0.05, and 0.15.

T - Array containing the transmission time of packets, in slots. This is computed from length, tau, and BPS.

Tmean - Mean transmission time of packets, in slots. Its associated integer value is "iTmean."

A description of other global variables used in the *model* program can be obtained from the program listing, found in Appendix E.

Intermediate results, as mentioned earlier, can be obtained by setting the DEBUG flag to a true value. If DEBUG is true, it is also generally useful to set the TRACE flag to true as well. The results of the program are listed below.

- Average stationary channel throughput
- Average packet delay (3 values)

1) Normalized to Tmean, in slots
2) In slots
3) In seconds

- Mean waiting time or channel acquisition time (2 values)

1) In slots
2) In seconds

- Average channel backlog

Usage

The questions of how to actually run the *model* program and how to use it for evaluation purposes are addressed here. When the *model* program is executed, the user is first presented with a list of parameters and default values and is given the opportunity to change them. If any change is desired, the program then lists each default value with the user given the option to modify it. After all defaults have been reviewed, the user is given the chance to reevaluate them again if so desired. If no modifications are necessary, the model calculations begin. A list of results are displayed on the screen upon the successful termination of the calculations. Some sample runs of program execution are found in Appendix F.

The *model* program is a measurement tool that can be used to determine some basic channel characteristics. Only the low-level channel access mechanisms involving CSMA/CD can be evaluated; not included are the effects of the higher level protocols. The most desirable situation for using this tool is after measurements have been obtained using artificial traffic generators, with both individual node and overall channel measurements gathered. At this point the traffic is pretty much characterized and the model can be effectively calibrated with the measurements. A certain amount of calibration can also be made from observed measurements obtained without traffic generators. However, it is a more difficult task due to a probable lack

of information, and some accuracy is lost in the process. Once the model has been calibrated by channel measurements, its parameters can be varied in order to determine their impact on model performance; from this study, the associated channel performance can be inferred.

Upon determining ν, γ, and τ for a given *LocalNet* channel, the effect of changing σ and thus the load on the channel can be investigated. A similar evaluation was done by Tobagi and Hunt [TOBA80], who also performed several other evaluations using the "linear feedback model". Gonsalves makes the observation that Tobagi and Hunt "predicted that instability sets in when the probability of a packet transmission attempt" in two slot times approaches one [GONS85]. This result could be used for predicting instability on the *LocalNet* channel. Upon obtaining results from various σ values, plots could be generated to depict the effects on delay and throughput.

4. FUTURE EFFORTS

A few ideas are discussed in this section with respect to future direction and efforts in *LocalNet* evaluation. Concerning our efforts, some investigation needs to be done on the detected underflow problem mentioned earlier, as well as completing the testing of the *model* program. Other tasks that could be considered and pursued are listed below, along with some additional comments.

(1) Determine what questions need to be addressed in doing network and channel measurements. Once this has been decided, determine what information needs to be gathered and how to collect it. These tasks were addressed earlier, in the measurement section of this paper.

(2) Implement the first model developed by Tobagi and Hunt, which investigates channel capacity. The most difficult part of this task is determining how to maximize throughput S in order to approach the channel capacity.

(3) Develop artificial traffic generators for different nodes of a channel. Probably one of the easier and less costly ways to accomplish this task would be something similar to that done by Gonsalves [GONS85] and by Shoch and Hupp [SHOC80]: develop a control program that passes generating programs to detected idle nodes. These programs are then set to start at the same time and run for a fixed period of time. With these generators the ability to set packet size(s) and packet generation times would be included. A known load can thus be applied to the channel and stability can be determined. There also needs to be added to each node some capacity to collect measurements. Possible data to be collected includes channel acquisition time of packets and number of collisions involved at that node per packet.

(4) Investigate the effects of higher level protocols on performance. Needed are analytic models of these higher levels which are implemented in the PCUs.

Our program would then be one of several components to be evaluated in order to determine values such as user response time.

(5) Carry out timing studies of several items in the network. These include the following:

- timing of the bridge, i.e., the amount of time to send a packet through the bridge, once all locations are known.
- timing of the link, i.e., the amount of time to send a packet through the link, once all locations are known.
- timing of a PCU, i.e., the amount of time to send some information through the layers at a PCU before being transmitted.
- slot time of a channel, i.e., the time to send a bit between the two farthest points on the channel.

(6) Investigate the possibility of having dynamic control in the network (or channel). It may be of some interest to restrict network traffic, if the load reaches some predetermined threshold value or condition. There are a couple of items to be considered in such an investigation: how to determine when the threshold or user saturation point is reached and what to do once it has been detected. A study by Crigler attempted to determine the user-saturation point of a single channel [CRIG85]. As was discovered by this study, it is not an easy task. The measurement investigated was packets per second, which if measured by a central monitor probably represents the number of *valid* packets on the channel. As the load on the channel increases to very high loads, the number of aborted packets will also increase, a quantity not considered in the packets per second measurement.

It is also important to consider what can be done once a determination of

threshold is made. One obvious action to take is to have some sort of message or signal sent to the PCUs to "slow down", when the threshold is detected.

(7) Obtain from Sytek any information they may have on *LocalNet* 20 measurements and evaluations they have made. A writeup for the viewgraphs presented by Edholm [EDH083] would be very useful.

REFERENCES

[AMER82] P. Amer, A Measurement Center for the NBS Local Area Computer Network, *IEEE Trans. on Computers*, **C-31**(8), Aug. 1982, pp. 723-729.

[AMER83] P. Amer, R. Rosenthal and R. Toense, Measuring a Local Network's Performance, *Data Communications*, Apr. 1983, pp. 173-182.

[BIBA81] K. Biba, LocalNet: A Digital Communications Network for Broadband Coaxial Cable, *COMPCON*, 1981, pp. 59-63.

[CHLA85] I. Chlamtac and M. Eisinger, Performance of Integrated Services (Voice/Data) CSMA/CD Networks, *Performance Evaluation Review*, **13**(2), Aug. 1985, pp. 87-93.

[CONT80] S. Conte and C. deBoor, *Elementary Numerical Analysis, An Algorithmic Approach*, McGraw-Hill, New York, NY , third edition 1980.

[CRIG85] Crigler, Detection of Single Channel Saturation on the Local Area Network, (NSWC, memorandum), June 21, 1985.

[DINE80] M. Dineson and J. Picazo, Broadband Technology Magnifies Local Networking Capability, *Data Communications*, Feb. 1980, pp. 61-79.

[DINE81] M. Dineson, Broadband Local Networks Enhance Communication Design, *EDN*, Mar. 4, 1981, pp. 77-86.

[EDHO83] P. Edholm, LocalNet 20, Modeling Analysis and Performance, (Sytek, Viewgraphs from NSWC presentation), June 1, 1983.

[ENNI83] G. Ennis and P. Filice, Overview of a Broad-Band Local Area Network Protocol Architecture, *IEEE Journal on Selected Areas in Communications*, **SAC-1**(5), Nov. 1983, pp. 832-841.

[ENNI85] G. Ennis, LocalNet/20 Balancing, (Sytek, Viewgraphs from NSWC presentation), June 1985.

[FERR83] D. Ferrari, G. Serazzi and A. Zeigner, *Measurement and Tuning of Computer Systems*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[GONS85] T. Gonsalves, Performance Characteristics of 2 Ethernets: an Experimental Study, *Performance Evaluation Review*, **13**(2), Aug. 1985, pp. 78-86.

[LAM80] S. Lam, A Carrier Sense Multiple Access Protocol for Local Networks, *Computer Networks*, **4**(1), Feb. 1980, pp. 21-32.

[METC76] R. Metcalfe and D. Boggs, Ethernet: Distributed Packet Switching for Local Computer Networks, *Comm. ACM*, **19**(7), July 1976, pp. 395-404.

[SHOC80] J. Shoch and J. Hupp, Measured Performance of an Ethernet Local Network, *Comm. ACM*, **23**(12), Dec. 1980, pp. 711-721.

[STOK80] D. Stokesberry and R. Rosenthal, The Design and Engineering of a Performance Measurement Center for a Local Area Network, *Proceedings, Computer Networking Symposium*, Washington, D. C. , Dec. 1980, pp. 110-115.

[SYTE83a] Sytek, *LocalNet 50/120 Statistical Monitor: Preliminary User's Guide*, Sytek, Inc., Mar. 31, 1983.

[SYTE83b] Sytek, LocalNet 50/201, Bridge With Link Option, User's Manual, Document Number U1000.1/1-01B, Sytek, Inc., 1983.

[SYTE83c] Sytek, LocalNet 20, Reference Manual and Installation Guide, Software Version v2.2.5, Document Number R2000.1/1-03A, Sytek, Inc., 1983.

[SYTE83d] Sytek, LocalNet 20/220, Packet Communication Unit, Preliminary User's Manual, Document Number U1050.1/1-01A, Sytek, Inc., 1983.

[TOBA77] F. Tobagi and L. Kleinrock, Packet Switching in Radio Channels: Part IV - Stability Considerations and Dynamic Control in Carrier Sense Multiple Access, *IEEE Transactions on Communications*, **COM-25**(10), Oct. 1977, pp. 1103-1119.

[TOBA80] F. Tobagi and V. Hunt, Performance Analysis of Carrier Sense Multiple Access with Collison Detection, *Computer Networks*, **4**(5), Oct. / Nov. 1980, pp. 245-259.

[TOEN83] R. Toense, Performance Analysis of NBSNET, *Journal of Telecommunication Networks*, 2(2), Spring 1983, pp. 177-186.

APPENDIX A: Tobagi/Hunt paper

Performance Analysis of Carrier Sense Multiple Access with Collision Detection*

Fouad A. Tobagi and V. Bruce Hunt **
Computer Systems Laboratory, Stanford University, Stanford, California, USA

Packet broadcasting in computer communication is attractive in that it combines the advantages of both packet-switching and broadcast communication. All stations share a common channel which is multi-accessed in some random fashion. Among the various random access schemes known, carrier sense multiple access (CSMA) has been shown to be highly efficient for environments with relatively short propagation delay. Packet broadcasting (and in particular CSMA) has been successfully applied to coaxial cables thus providing an efficient means for communication in local environments. In addition, in such environments the possibility of detecting collisions on the coaxial cable enhances the performance of CSMA by aborting conflicting transmissions, thus giving rise to the carrier sense multiple access schemes with collision detection (CSMA-CD). In this paper we extend an analysis of CSMA to accommodate collision detection. The analysis provides the throughput-delay performance of CSMA-CD and its dependence on such key system parameters as the average retransmission delay and the collision recovery time.

* This research was supported by the Advanced Research Projects Agency, Department of Defense, Contract No. MDA903-79-C-0201, Order No. AO3717, monitored by the office of Naval Research.

** V. Bruce Hunt is now with the Telecommunications Sciences Center SRI-International, 333 Ravenswood Avenue, Menlo Park, California 94025, USA.

Computer Networks 4 (1980) 245–259

1. Introduction

There are numerous reasons why advances in local area communication networks have significantly increased in the past few years. The recent interest in the application of the (now available) inexpensive processing power to office and industrial automation, the necessity for the sharing of expensive scarce resources, the need for local collection and dissemination of information, and the rising interest in distri-

Dr. Tobagi is Assistant Professor of Electrical Engineering, School of Engineering, Stanford University. His current research interests include computer communication networks, packet switching in ground radio and satellite networks, modeling and performance evaluation of computer communications systems.

From 1971 to 1974, he was with the University of California, Los Angeles, where he participated in the ARPA Network Project as a postgraduate research engineer and did research in packet radio communication. During the summer of 1972, be was with the Communications Systems Evaluation and Synthesis Group, IBM, J. Watson Research Center, Yorktown Heights, NY. From December 1974 to June 1978, he was a research staff project manager with the ARPA project at the Computer Science Department, UCLA, and engaged in the modeling, analysis and measurements of packet radio systems. In 1978, he joined the faculty at Stanford.

Dr. Tobagi received the engineering degree from Ecole Centrale des Arts et Manufactures, Paris, France, in 1970 and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1971 and 1974, respectively.

V. Bruce Hunt is employed by SRI International and is a graduate student at Stanford University. Previously, he was with Zilog Corporation where he designed, implemented and analyzed ARIEL, a microprocessor-based local network based on the principles employed by Ethernet (packet broadcasting CSMA-CD). His current research interests include distributed operating system design and local network analysis and design.

From 1972 to 1978 he was with the Stanford Center for Information Processing working on the multiprocessor system at the Stanford Linear Accelerator Center and the Stanford time sharing system. He was also the group leader for WILBUR of the Stanford time sharing system.

Mr. Hunt received a bachelor of science degree in mathematics from Harvey Mudd College. He was a California State Scholar from 1966 to 1970 and a Mayer Scholar in 1966. He is a member of A.C.M. and IEEE.

buted architectures for data processing are but a few examples.

Just as in any field, the development of local area computer communication systems is subject to a number of constraints. Simplicity, flexibility and reliability usually portray these constraints. The environments in question are generally characterized by a large and often variable number of devices requiring interconnection. Such environments call for networks with simple topologies and simple inexpensive connection interfaces which provide great flexibility to accommodate the variability in the environment, and which achieve the desired level of reliability.

Several architectures have been proposed which include MITRE'S Mitrix, Bell Telephone Laboratory's Spider, and U.C. Irvine's Distributed Computing System (DCS) [1–4]. Spider and DCS use a ring topology, while Mitrix uses two one-way busses implemented by CATV technology. As for system control, Mitrix and Spider use a central mini-computer for switching and bandwidth allocation, while DCS uses distributed control.

Another network architecture, based on the packet broadcasting technology and exemplified by Ethernet [5] appears to be a very effective solution in satisfying the above mentioned constraints. Packet broadcasting is attractive in that it combines the advantages of both packet switching and broadcast communication. Packet switching offers the efficient sharing of communication resources by many contending users with unpredictable demands; broadcast communication, whenever possible, eliminates complex topological design problems. Given that computer communication traffic is bursty in nature, it has been well established that it is more efficient to provide the available communication bandwidth as a single high-speed channel to be shared by the many contending users, thus attaining the benefits of the strong law of large numbers. This clearly results in a multiaccess environment that calls for schemes to control access to the channel, referred to as random access schemes. The earliest and simplest such scheme is the so-called pure-ALOHA, first used in the ALOHA-System [6]; unfortunately, pure-ALOHA provides a maximum channel utilization which does not exceed 18%. Another such scheme, carrier sense multiple access (CSMA), has been shown to be highly efficient in environments with propagation delays which are short compared to the packet transmission time [7]. In essence, CSMA reduces the level of interference caused by overlapping packets in the random multiaccess channel by allowing devices to sense carrier due to other users' transmissions, and inhibit transmission when the channel is in use. Packets which are inhibited or suffer a collision are rescheduled for transmission at a later time according to some rescheduling policy.

Ethernet is a local communication network which uses CSMA on a tapped coaxial cable to which all the communicating devices are connected. The device connection interface is a passive cable tap so that failure of an interface does not prevent communication among the remaining devices. The use of a single coaxial cable naturally achieves broadcast communication. Moreover, given the physical characteristics of data transmission on coaxial cables, in addition to sensing carrier, it is possible for Ethernet transceivers to detect interference among several transmissions (including their own) and abort transmission of their colliding packets. This produces a variation of CSMA which we refer to as carrier sense multiple access with collision detection (CSMA-CD). It is networks of the Ethernet type that we address in this paper.

CSMA in fully connected environments has been previously analyzed and its performance derived [7–10]. We extend here the analysis of CSMA to accommodate collision detection. This analysis provides the throughput-delay performance of CSMA-CD and its dependence on such key system parameters as the average rescheduling delay and collision recovery time. We furthermore characterize the improvement gained by CSMA-CD over CSMA for fixed and variable size packets.

The CSMA-CD schemes are described in section 2, followed by the analysis in section 3. Numerical results are discussed in section 4.

2. The CSMA-CD schemes

Carrier sense schemes require that each device with a packet ready for transmission senses the channel prior to transmission. A number of protocols exist which pertain to the action taken by the terminal after observing the state of the channel. In particular, a terminal never transmits when it senses that the channel is busy. Tobagi and Kleinrock described two such protocols in the context of ground radio channels [7,11]. They are the non-persistent CSMA and the p-persistent CSMA protocols. These protocols are extended here to environments in which the collision detection capability is available.

In the non-persistent CSMA-CD scheme, a terminal with a packet ready for transmission senses the channel and proceeds as follows.

1. If the channel is sensed idle, the terminal initiates transmission of the packet.
2. If the channel is sensed busy, then the terminal schedules the retransmission of the packet to some later time and repeats the algorithm.
3. If a collision is detected during transmission, the transmission is aborted and the packet is scheduled for retransmission at some later time. The terminal then repeats the algorithm.

In the 1-persistent CSMA-CD protocol (a special case of p-persistent CSMA), a terminal which finds the channel busy persists on transmitting as soon as the channel becomes free. Thus a ready terminal senses the channel and proceeds as in nonpersistent CSMA-CD, except that, when the channel is sensed busy, it monitors the channel until it is sensed idle and then with probability one initiates transmission of the packet.

The p-persistent protocol is an enhancement of the 1-persistent protocol by allowing ready terminals to randomize the start of transmission following the instant at which the channel goes idle. Thus a ready terminal senses the channel and proceeds as in the above schemes except that when the channel is sensed busy, the terminal persists until the channel is idle, and

(i) with probability p it initiates transmission of the packet
(ii) with probability $1 - p$ it delays transmission by τ seconds (the end-to-end propagation delay); if, at this new point in time, the channel is sensed idle, then the terminal repeats this process [steps (i) and (ii)], otherwise, it schedules retransmission of the packet to some later time.

Note that the p-persistent and non-persistent protocols become identical if the rescheduling delays are chosen for both protocols as an integer number of τ delay units geometrically distributed, with parameter p (the parameter in the p-persistent protocol). This follows because of the memoryless property of the geometric distribution. In this paper we analyze only the non-persistent and 1-persistent protocols.

In all CSMA-CD protocols, given that a transmission is initiated on an *empty* channel, it is clear that it takes at most one end-to-end propagation delay, τ, for the packet transmission to reach all devices, as depicted in fig. 1; beyond this time the channel is guaranteed to be sensed busy for as long as data transmission is in process [1]. A collision can occur only if another transmission is initiated before the current one is sensed, and it will then take, at most, one additional end-to-end delay before interference reaches all devices. (See fig. 1.) Let ξ denote the time it takes a device to detect interference once the latter has reached it. ξ depends on the implementation and can be as small as 1 bit transmission time, as is the case with Ethernet [5]. Furthermore, Ethernet has a collision consensus reenforcement mechanism by which a device, experiencing interference, jams the channel to ensure that all other interfering devices detect the collision. We denote by ζ the period used for collision consensus reenforcement. Given that a collision occurs, the time until all devices stop transmission, γ, is thus given by [2]

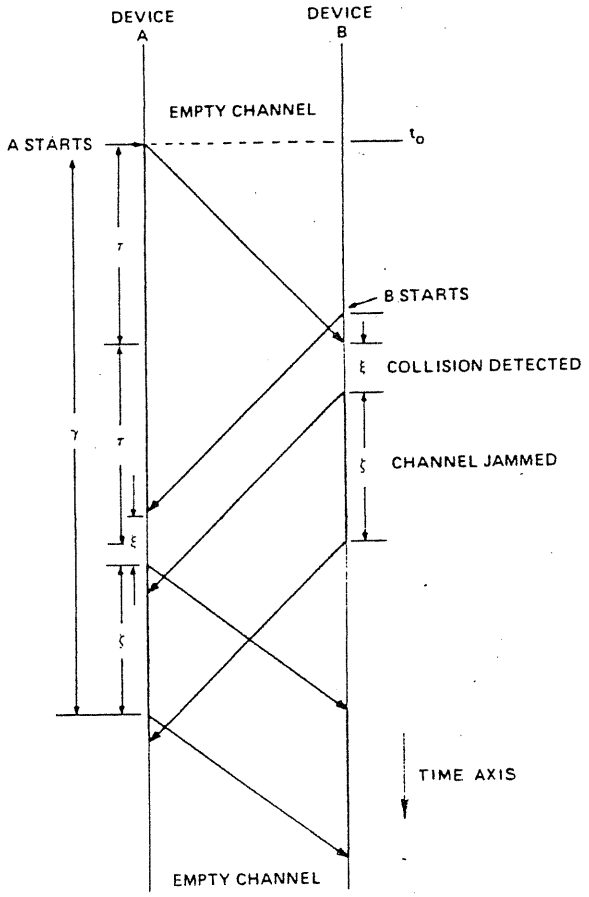


Fig. 1. Collision Detection and Recovery Time in CSMA-CD.

[1] We assume that the sensing operation is instantaneous on this (high-bandwidth) channel.

[2] This assumes that all interfering devices undertake the collision consensus reenforcement.

$\gamma = 2\tau + \xi + \zeta$.

The time until the channel is again sensed idle by all devices is clearly $\gamma + \tau$.

3. Analysis

We assume that the time axis is slotted where the slot size is the end-to-end propagation delay. For simplicity in analysis, we consider all devices to be synchronized and forced to start packet transmission only at the beginning of a slot. When a device becomes ready in some slot, it senses the channel during the slot and then operates according to the CSMA-CD protocols described above.

3.1. Analysis of CSMA-CD with fixed size packets

3.1.1. Channel capacity

As in previous analysis of random access schemes, channel capacity is obtained by considering an infinite population model which assumes that all devices collectively form an independent Poisson source, and that the average retransmission delay is arbitrarily large [7,10].

Consider first the non-persistent CSMA-CD protocol. We observe on the time axis an alternate sequence of transmission periods (successful or unsuccessful) and idle periods. A transmission period followed by an idle period is called a cycle (see fig. 2). With the infinite population assumption, all cycles are statistically identical. Let g denote the rate of devices becoming ready during a slot. Let T denote the transmission time (in slots) of a packet. A successful transmission period is of length $T + 1$ slots. In case of a collision, the length of a transmission period is $\gamma + 1$ slots. Given that the source is Poisson, the probability that a transmission is successful is $P_s = ge^{-g}/(1 - e^{-g})$; the average idle period is $\bar{I} = e^{-g}/(1 - e^{-g})$; the average transmission period is $\overline{TP} = P_s T + (1 - P_s)\gamma + 1$; and the throughput is given by

$$S = \frac{P_s T}{\overline{TP} + \bar{I}}$$

$$= \frac{Tg\, e^{-g}}{Tge^{-g} + (1 - e^{-g} - ge^{-g})\gamma + 1} . \qquad (1)$$

The channel capacity is obtained by maximizing S with repect to g.

Consider now the 1-persistent CSMA-CD protocol. We observe on the time axis an alternate sequence of busy and idle periods, whereby a busy period is any collection of juxtaposed transmission periods surrounded by idle periods. A busy period followed by an idle period constitutes a cycle (see fig. 3). Again, all cycles are statistically identical. Let $\bar{B}$ denote the average duration of a busy period, $\bar{I}$ the average duration of an idle period, and $\bar{U}$ the average time during a cycle that the channel is carrying successful transmissions. The throughput is given by $S = \bar{U}/(\bar{B} + \bar{I})$. In this infinite population model, the success or failure of a transmission period in the busy period is only dependent on the preceding transmission period (and thus its length), except for the first transmission period of the busy period, which depends on arrivals in the preceding slot. Accordingly, given that a transmission period in the busy period is of length X ($X = T + 1$ or $\gamma + 1$), the length of the remainder of the busy period is a function of X, and we let $B(X)$ denote its average. In the same manner we define $U(X)$. Let $q_i(X)$ be the probability that there are i arrivals in X slots. Under the Poisson assumption,

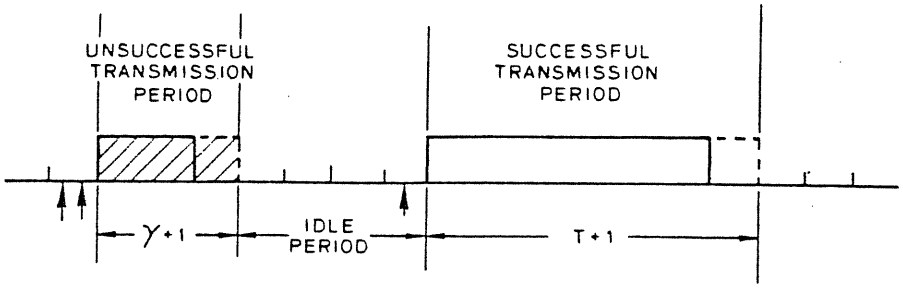


Fig. 2. Transmission and Idle Periods in Slotted Nonpersistent CSMA-CD. (Vertical arrows represent users becoming ready to transmit).

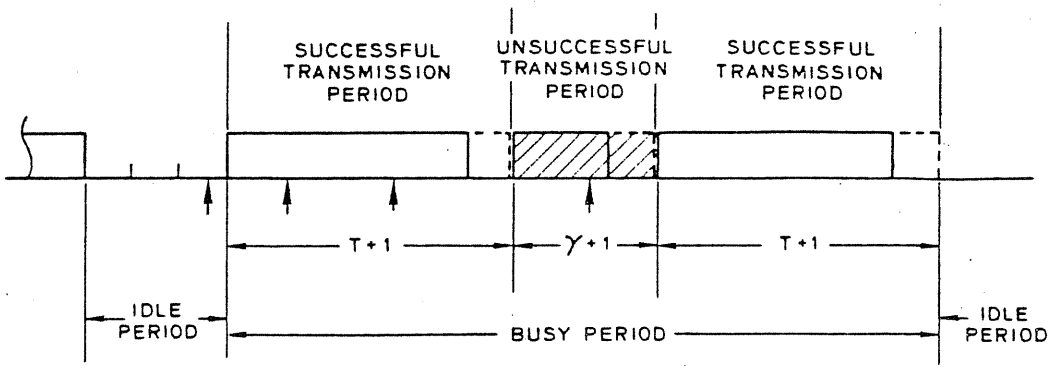


Fig. 3. Busy and Idle Periods in Slotted 1-persistent CSMA-CD. (Vertical arrows represent users becoming ready to transmit).

$q_i(X) = (gX)^i e^{-gX}/i!$. $B(X)$ is given by

$$B(X) = \frac{q_1(X)}{1 - q_0(X)} [T + 1 + (1 - q_0(T+1)) B(T+1)] + \left[1 - \frac{q_1(X)}{1 - q_0(X)}\right] \times [\gamma + 1 + (1 - q_0(\gamma+1)) B(\gamma+1)] . \quad (2)$$

Writing eq. (2) with $X = T + 1$ and $X = \gamma + 1$, we obtain two equations in the two unknowns, $B(T + 1)$ and $B(\gamma + 1)$. The average busy period $\bar{B}$ is then given by $B(1)$, expressed in terms of $B(T + 1)$ and $B(\gamma + 1)$.

Similarly, $U(X)$ is given by

$$U(X) = \frac{q_1(X)}{1 - q_0(X)} [T + (1 - q_0(T+1)) U(T+1)] + \left[1 - \frac{q_1(X)}{1 - q_0(X)}\right][(1 - q_0(\gamma+1)) U(\gamma+1)] . \quad (3)$$

By taking $X = T + 1$ and $X = \gamma + 1$, we obtain two equations in the two unknowns $U(T + 1)$ and $U(\gamma + 1)$. As above, $\bar{U} = U(1)$ given in terms of $U(T + 1)$ and $U(\gamma + 1)$. $\bar{I}$ is simply equal to $1/(1 - e^{-g})$. Note that when $\gamma = T$, the expression for the throughput of 1-persistent CSMA-CD reduces to that of 1-persistent CSMA as given in [7].

3.1.2. Delay Analysis

We consider here the non-persistent protocol. To analyze packet dealy, we adopt the same *"linear feedback model"* used for the analysis of CSMA in [9,10]. The model consists of a finite population of M devices in which each device can be in one of two states: backlogged or thinking. In the thinking state, a device generates and transmits (provided that the channel is sensed idle) a new packet in a slot with probability σ. A device is said to be backlogged if its packet either had a channel collision or was blocked because of a busy channel. A backlogged device remains in that state until it completes successful transmission of the packet, at which time it switches to the thinking state. The rescheduling delay of a backlogged packet is assumed to be geometrically distributed with a mean of $1/\nu$ slots; this in effect is identical to considering that each backlogged user senses the channel in the current slot with a probability ν.

In this study, we assume M, σ and ν to be time invariant. We consider τ (the slot size) to be the unit of time. We again denote by S the average stationary channel throughput defined as the fraction of channel time occupied by valid transmissions. We denote by C the channel capacity defined as the maximum achievable channel throughput. We finally denote by D the average packet delay defined as the time lapse from when the packet is first generated until it is successfully received by the destination device.

Let N^t be a random variable representing the number of backlogged devices at the beginning of slot t. We follow the approach used in [9], and consider the embedded Markov chain identified by the first slot of each idle period (see fig. 4). We then use properties resulting from the theory of regenerative processes to derive the stationary channel performance under CSMA-CD, as outlined in [9,10].

We seek the transition probability matrix P between consecutive embedded points. P is the product of several single-slot transition matrices which we now define. N^t is invariant over the entire idle period except over slot $t_e + I - 1$. We denote by R the transition matrix for slot $t_e + I - 1$ and Q for all

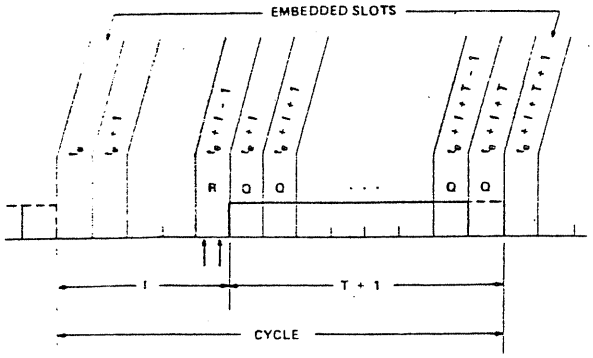


Fig. 4. The Embedded Slots in Nonpersistent CSMA Schemes.

remaining slots of the busy period. Since the length of the busy period depends on the number of devices which become ready in slot $t_e + I - 1$, we write R as $R = S + F$, where the (i, k)th elements of S and F are defined as

$$s_{ik} = \Pr\{N^{t_e+I} = k \text{ and transmission is successful} \mid N^{t_e+I-1} = i\}, \tag{4}$$

$$f_{ik} = \Pr\{N^{t_e+I} = k \text{ and transmission is unsuccessful} \mid N^{t_e+I-1} = i\}. \tag{5}$$

For any slot t in the busy period, Q simply reflects the addition to the backlog from the $M - N^t$ thinking devices. If the transmission is successful, the transmission period has length $T + 1$; if it is unsuccessful, its length is $\gamma + 1$. The transition matrix P is therefore expressed as

$$P = SQ^{T+1}J + FQ^{\gamma+1}, \tag{6}$$

where S, F, and Q are given by

$$s_{ik} = \begin{cases} 0 & \text{for } k < i \\ \dfrac{(1-\sigma)^{M-i}[i\nu(1-\nu)^{i-1}]}{1-(1-\nu)^i(1-\sigma)^{M-i}} & \text{for } k = i \\ \dfrac{(M-i)\,\sigma(1-\sigma)^{M-i-1}[1-\nu]^i}{1-(1-\nu)^i(1-\sigma)^{M-i}} & \text{for } k = i+1 \\ 0 & \text{for } k > i+1 \end{cases} \tag{7}$$

$$f_{ik} = \begin{cases} 0 & \text{for } k < i \\ \dfrac{(1-\sigma)^{M-i}[1-(1-\nu)^i - i\nu(1-\nu)^{i-1}]}{1-(1-\nu)^i(1-\sigma)^{M-i}} & \text{for } k = i \\ \dfrac{(M-i)\,\sigma(1-\sigma)^{M-i-1}[1-(1-\nu)^i]}{1-(1-\nu)^i(1-\sigma)^{M-i}} & \text{for } k = i+1 \\ \dfrac{\binom{M-i}{k-i}(1-\sigma)^{M-k}\sigma^{k-i}}{1-(1-\nu)^i(1-\sigma)^{M-i}} & \text{for } k > i+1 \end{cases} \tag{8}$$

$$q_{ik} = \begin{cases} 0 & \text{for } k < i \\ \binom{M-i}{k-i}(1-\sigma)^{M-k}\sigma^{k-i} & \text{for } k \geqslant i \end{cases} \tag{9}$$

and where J represents the fact that a successful transmission decreases the backlog by 1, its (i, k)th elements being defined as

$$j_{ik} = \begin{cases} 1 & \text{for } k = i-1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

It is clear that with $\gamma = T$, the above expression for P then corresponds to CSMA without CD. Let $\Pi = [\pi_0, \pi_1, \ldots, \pi_M]$ denote the stationary probability distribution of N^t at the embedded points. Π is obtained by the recursive solution of $\Pi = \Pi P$.

Since N^{t_e} is a regenerative process, the average stationary channel throughput is computed as the ratio of time the channel is carrying successful transmission during a cycle (an idle period followed by a busy period) averaged over all cycles, to the average cycle length [9,10]. Therefore we have

$$S = \frac{\sum_{i=0}^{M} \pi_i P_s(i)\, T}{\sum_{i=0}^{M} \pi_i \left\{ \dfrac{1}{1-\delta_i} + 1 + P_s(i)\, T + [1 - P_s(i)]\,\gamma \right\}}, \tag{11}$$

$P_s(i)$ is the probability of a successful transmission during a cycle with $N^t e = i$, and is given by

$$P_s(i) = ((M-i)\,\sigma(1-\sigma)^{M-i-1}(1-\nu)^i + i\nu(1-\nu)^{i-1}(1-\sigma)^{M-i})/(1-(1-\nu)^i(1-\sigma)^{M-i}) \quad (12)$$

$(1-\delta_i)^{-1}$, where $\delta_i = (1-\nu)^i(1-\sigma)^{M-i}$, is the average idle period given $N^t e = i$.

Similarly, the average channel backlog is computed as the ratio of the expected sum of backlogs over all slots in a cycle (averaged over all cycles), to the average cycle length [9,10]. Therefore we have

$$\bar{N} = \frac{\sum_{i=0}^{M} \pi_i \left[\frac{i}{1-\delta_i} + A(i)\right]}{\sum_{i=0}^{M} \pi_i \left\{\frac{1}{1-\delta_i} + 1 + P_s(i)\,T + [1-P_s(i)]\,\gamma\right\}}, \quad (13)$$

where $A(i)$ is the expected sum of backlogs over all slots in the busy period with $N^t e = i$, and is given by [3]

$$A(i) = \sum_{l=0}^{T}\sum_{j=i}^{M} j[SQ^l]_{ij} + \sum_{l=0}^{\gamma}\sum_{j=i}^{M} j[FQ^l]_{ij} = \sum_{j=i}^{M} j\left[S\sum_{l=0}^{T} Q^l + F\sum_{l=0}^{\gamma} Q^l\right]_{ij} \quad (14)$$

By Little's result [12], the average packet delay (normalized to T] is simply expressed as

$$D = \bar{N}/S \quad (15)$$

3.2. Analysis of CSMA-CD with variable size packets

T is now a discrete random variable. Let

$$G_T(z) \triangleq \sum_{t=1}^{\infty} z^t \Pr\{T=t\} \quad (16)$$

be the generating function of the distribution of T. In case of collison, regardless of the number of colliding packets and their lengths, the length of the busy period is $\gamma + 1$. In case of success, the length of the busy period is now random and has the same distribution as $T + 1$. The reason this is true, despite the fact that the length of a packet remains constant during its entire lifetime, is simply explained by the fact that the successful or unsuccessful outcome of a transmission period is solely dependent on the number of devices becoming ready at the beginning of that transmission period, and is independent of the lengths of the contending packets. Since the length of the busy period in case of a collision is constant (equal to $\gamma + 1$), the evolution of the channel over time is statistically identical to that in which the length of a packet is drawn from the packet length distribution only when its transmission is successful.

However, in the case of CSMA *without* collision detection, the collision period is a function of the lengths of the contending packets so the evolution of the channel over time is not statistically identical to that in which the length of a packet is drawn from the length distribution upon success. We include in appendix A an approximate analysis of CSMA in which the packet length at each transmission is independently redrawn from the packet length distribution.

The performance of nonpersistent CSMA-CD can thus be obtained from the previous analysis with the following simple modifications. The matrix P is now rewritten as

$$P = SG_T(Q)\,QJ + FQ^{\gamma+1} \quad (17)$$

and T is replaced by

$$\bar{T} \triangleq \sum_t t \Pr\{T=t\} \quad (18)$$

the average packet size, in all of equations (1), (11), (13) and (14).

In this case, the average packet delay given by Eq. (15) is normalized with respect to $\bar{T}$. For the same reason stated above, the average channel acquisition time (i.e., the time from when a packet is generated until it starts its successful transmission), denoted by W (in slots), is given by

$$W = D\bar{T} - \bar{T} \quad (19)$$

Accordingly, the delay incurred by packets of length t is expressed as

$$D_t = W + t \text{ (in slots)}. \quad (20)$$

It is interesting to note that for any throughput S, the difference in the delay incurred by packets of two different sizes is just the difference in transmission time of these packets. Smaller packets incur a smaller delay. The throughput contributed by packets of

[3] For an arbitrary matrix $\underline{B}$, we adopt the notation $[\underline{B}]_{ij}$ to represent the (i,j)th element of $\underline{B}$.

size t, denoted by S_t is expressed as

$$S_t = \frac{t\Pr\{T = t\}\ S}{\overline{T}}. \tag{21}$$

4. Numerical Results and Discussion

4.1. Fixed Packet Size

The behavior of CSMA-CD for fixed γ is, as expected, similar to that of CSMA [9], namely its throughput-delay performance is sensitive to ν, and therefore to the average retransmission delay. Figures 5 and 6 display the throughput-delay curves for nonpersistent CSMA and CSMA-CD respectively, with $M = 50$, $T = 100$, $\gamma = 2$, and various values of ν. For a fixed value of ν, the channel exhibits a maximum achievable throughput which depends on that value, hereafter referred to as the ν-capacity. We observe that, for a given ν, CSMA-CD always achieves, again as expected, lower delay for a given throughput and a higher ν-capacity [4]. The optimum throughput-delay performance is obtained by taking the lower envelope of all fixed-ν curves. Overall, CSMA-CD provides an improvement both in terms of channel capacity and throughput-delay characteristics.

We discuss now the sensitivity of this improvement to the collision detect time, γ, and the packet length T. Just as with CSMA, the larger T is, the better is the CSMA-CD performance for fixed γ. In fig. 7, we plot the channel capacity for the nonpersistent CSMA-CD versus γ for various packet lengths. The capacity at $\gamma = T$ is that of CSMA. The relative improvement in channel capacity obtained by CSMA-CD becomes more important as T decreases, that is, as the performance of CSMA degrades. We note for example that at best (i.e., when $\gamma = 2$) this relative improvement is about 16% (0.62 to 0.76) for $T = 10$ and about 11% (0.86 to 0.96) for $T = 100$. Clearly, for larger T ($T \geqslant 100$), nonpersistent CSMA provides relatively high channel capacity, and thus leaves little margin for improvement. With the 1-persistent protocol, however, the improvement can be more substantial. Channel capacity increases from about 0.53 for

[4] Note that for all values of ν used in plotting figs. 5 and 6, the ν-capacity with CSMA-CD approached the channel capacity (maximized over ν); there are values of ν (higher than $\nu = 0.15$) for which the ν-capacity is much lower than the CSMA-CD channel capacity similarly to what is seen in fig. 5 for CSMA and $\nu = 0.10$.

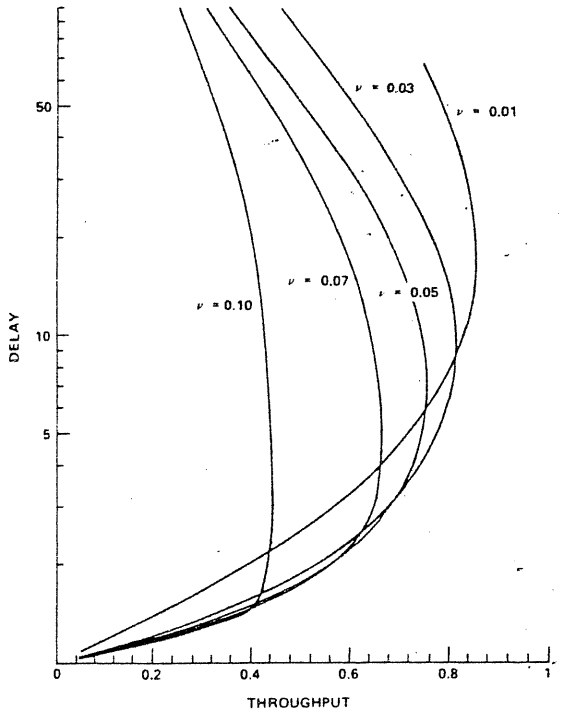


Fig. 5. The Throughput-Delay Tradeoff in CSMA at Fixed ν.

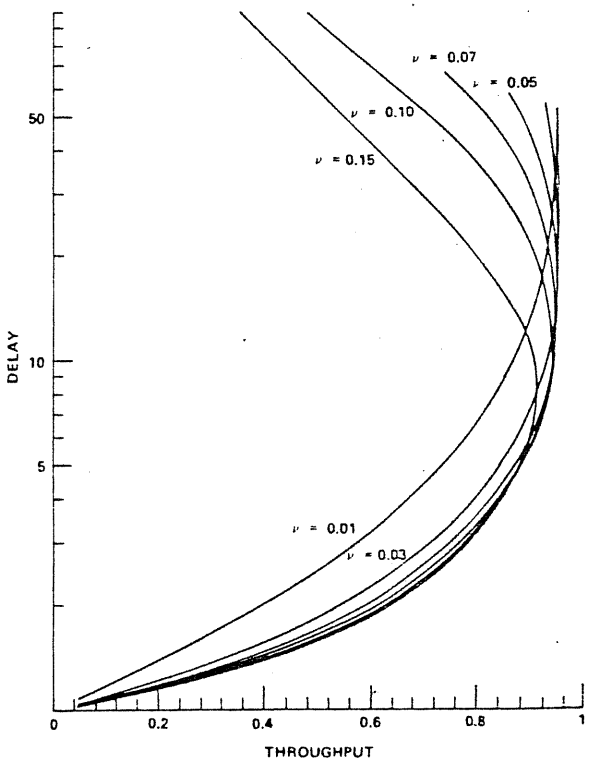


Fig. 6. The Throughput-Delay Tradeoff in CSMA-CD at Fixed ν.

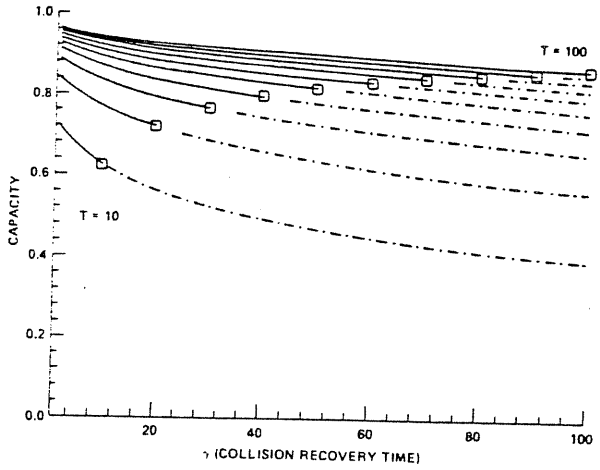


Fig. 7. Channel Capacity Versus γ in CSMA-CD.

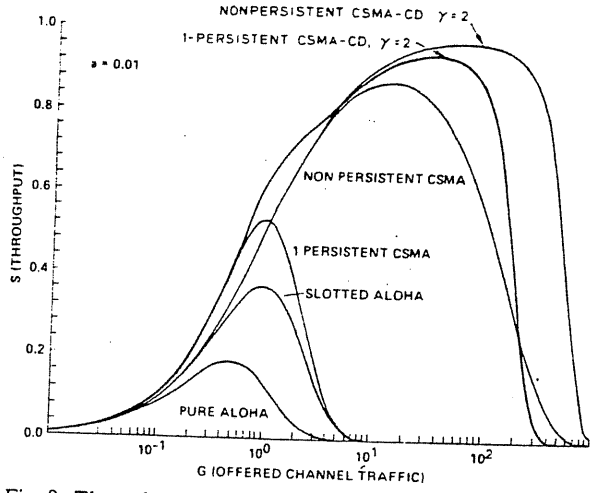


Fig. 9. Throughput Versus Channel Traffic (Infinite Population Model).

1-persistent CSMA to about 0.93 for 1-persistent CSMA-CD with $\gamma = 2$.

The effect of CD on the minimum delay (optimized with respect to ν) for a fixed channel throughput is seen in fig. 8, where we plot this minimum delay versus γ for the nonpersistent case with $M = 50$ and $T = 100$. We note that the higher the throughput is the better is the improvement. At low throughput (e.g., $S = 0.20$), the delay is insensitive to γ. With moderately high throughputs (e.g., $S = 0.68$), the delay with CSMA-CD (at $\gamma = 2$) is 70% that of CSMA.

As the throughput approaches the CSMA channel capacity (e.g. $S = 0.84$) the ratio in delay can be as low as 1/3 in favor of collision detection ($\gamma = 2$). Of course, for even higher throughputs, CSMA-CD achieves a finite delay as long as γ is sufficiently small.

The (S, G) relationship for CSMA-CD is displayed in fig. 9 along with the curves corresponding to the ALOHA and CSMA schemes. This figure exhibits again the improvement in channel capacity gained by CSMA-CD over all other schemes. For random access

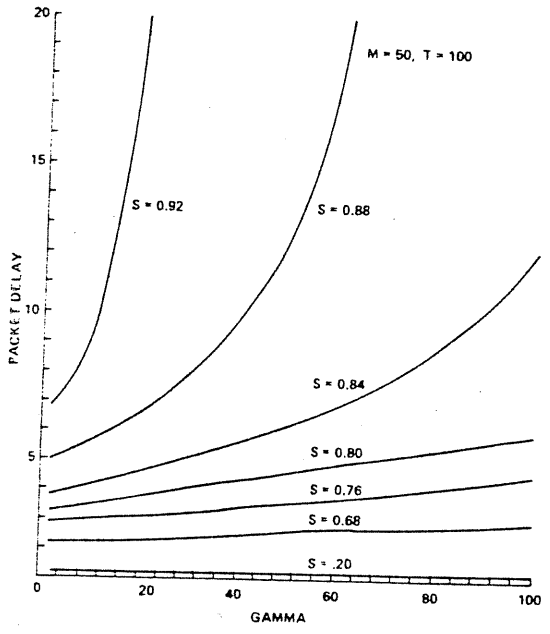


Fig. 8. Packer Delay in CSMA-CD at Fixed Throughput Versus γ.

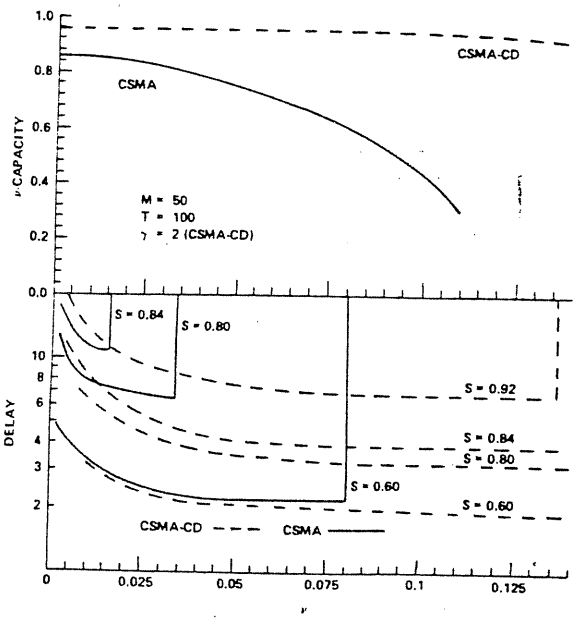


Fig. 10. Channel Capacity and Packet Delay at Fixed Throughput Versus ν for CSMA and CSMA-CD.

schemes in general, the fact that the throughput drops to zero as the offered channel traffic increases indefinitely is indicative of unstable behavior [9,14]. With CSMA-CD the ability to maintain a throughput relatively high and near capacity over a very large range of the offered channel traffic (see fig. 9) suggests that CSMA-CD may not be as unstable as the other schemes. That is, in the absence of dynamic control, CSMA-CD is capable of sustaining proper behavior when the channel load exceeds that for which the system has been tuned (i.e., optimized with respect to ν). (Note that, with respect to this stability argument, the nonpersistent CSMA-CD proves to be superior to 1-persistent CSMA-CD, in that it offers high throughput over a larger range of the offered traffic.) We further illustrate this important feature by plotting in fig. 10, as a function of ν, the ν-capacity and the packet delay at various channel throughputs for both nonpersistent CSMA and CSMA-CD ($\gamma = 2$) with $T = 100$ and $M = 50$. As ν approaches zero, the delay at fixed throughput gets arbitrarily large (due to large retransmission delays), while as ν approaches 1, the ν-capacity approaches zero (due to higher level of interference among backlogged devices). Thus there is a limited range for ν which is of practical interest. As we see in fig. 10, for $T = 100$ this range is about (0.005, 0.3). The ν-capacity curve for CSMA-CD is flat over a large portion of this range; with CSMA, the ν-capacity drops steadily as ν increases. and exhibits insensitivity only for smaller values of ν falling outside our range. Consider now CSMA. Given a channel throughput S, packet delay decreases as we increase ν (starting from relatively small values) and remains relatively constant, until, due to the decrease in ν-capacity, we reach a value of ν for which the ν-capacity approaches S, and thus the delay increases very sharply; this "practical" range of ν gets narrower as S increases, indicating that for high throughput, the system requires fine tuning. Let $S = 0.60$ be, for example, the (moderate) stationary channel throughput we expect the system to support. The channel is properly tuned (i.e., minimum delay is achieved) for ν in the range (0.04, 0.08). Consider now that the offered load on the channel is time-varying and suppose that the desired throughput exceeds 0.60 reaching values close to channel capacity (e.g., $S = 0.84$). This actually happens for increasing values of σ (i.e., when devices generate packets at a faster rate). If the desired load remains at such a high value for a relatively long period of time the channel saturates (i.e., the throughput drops to a low value, nearly all devices become backlogged and packet delay increases indefinitely). We can certainly support variations in offered load covering the entire range of achievable throughputs ($S \leqslant 0.84$) by setting ν at a value in the (now narrow) range corresponding to $S = 0.84$. This is achieved at the expense of increased average delay for $S = 0.60$ of 36% (from 2.2 to 3) unless, of course, dynamic control is exercised [9].

With CSMA-CD, on the contrary, there is a relatively wide range of ν for which the delay at fixed throughput is near optimum for all throughput levels up to 0.92.

Numerical results obtained for different values of the system parameters, namely $M = 50$ and 250, and $T = 10$ and 100 have shown that basically as T decreases or as M increases or both, then CSMA-CD starts exhibiting a behavior similar to that of CSMA, while always achieving improved performance.

In summary, the kind of improvement over slotted ALOHA we saw in [9] for CSMA due to carrier sensing, is now seen in CSMA-CD over CSMA.

4.2. Variable Packet Size

It is clear from the above discussion that as the packet size decreases the improvement obtained with collision detection is more important. We inquire here about the performance of the channel with collision detection when packets are of variable length. Instead

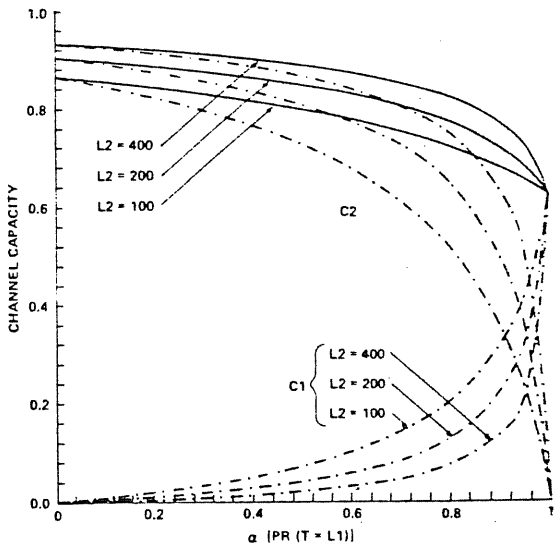


Fig. 11. CSMA Channel Capacity Versus α for Dual Packet Size.

of examining the general message length distribution case, we present here numerical results for the simpler dual packet size case; that is, traffic consists of a mixture of short and long packets. This simple distribution represents accurately many real situations, among them the important instance of the mixture of short packets resulting from interactive traffic and long packets resulting from file transfers. In fact, measurements performed on Xerox's Ethernet have clearly exhibited such a distribution [15]. Moreover, results obtained here are expected to be representative of the performance of a channel in more general packet length distributions.

We let L_1 (L_2) denote the transmission time of short (long) packets. We let α denote the fraction of packets generated which are short. Figure 11 displays the non-persistent CSMA channel capacity versus α for the case of short packets equal to 10 (slots) and three cases of long packets (100, 200, 400). The capacity of the channel decreases as α increases. With larger values of L_2 (e.g., $L_2 = 400$), this decrease is fairly slow until α is about 0.80; beyond 0.80 the capacity rapidly declines to reach the (lower) capacity of $T = L_1$. This shows that a relatively small fraction of long packets in the traffic mix can result in a channel utilization close to that obtained with only long packets. However it is important to note that, as the fraction of long packets increases, the fraction of channel capacity due to long packets, denoted by C_2, increases extremely rapidly to the detriment of that due to short packets, denoted by C_1, which decreases dramatically. This is seen in fig. 11 where we also plot C_1 and C_2 versus α. Recall that, by Equation (21) which also holds for CSMA under the independence assumption, C_1 and C_2 are given by

$$C_1 = \frac{\alpha L_1}{\alpha L_1 + (1-\alpha) L_2} C, \tag{22}$$

$$C_2 = \frac{(1-\alpha) L_2}{\alpha L_1 + (1-\alpha) L_2} C, \tag{23}$$

where C is the channel capacity.

In fig. 12 we plot the capacity versus α for CSMA-CD ($L_1 = 10, L_2 = 100$) at various values of γ, along with the corresponding CSMA capacity curve. The insensitivity of CSMA-CD capacity to variations of α over a large range of α is more apparent than with CSMA. However, the relative importance of C_1 and C_2 remains the same as in CSMA since the ratio of C_1 and C_2 is independent of the capacity.

In fig. 13 we plot the packet delay (averaged over

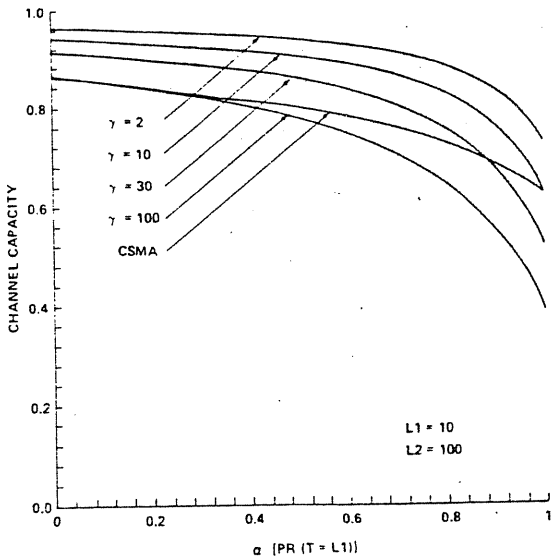


Fig. 12. CSMA-CD Channel Capacity Versus α for Dual Packet Size.

all packets and normalized to L_1) versus throughput for various values of α for both CSMA and CSMA-CD. Packet delay includes the (successful) transmission time of the packet; thus clearly as the fraction of long packets increases, so does the average packet delay. Figure 13 exhibits the clear tradeoff between average packet delay and attainable channel capacity as the mix α varies. The improvement due to collision detection is also apparent for all values of α.

Most commonly, short packets belong to interactive users who require small delay, while long packets result from file transfer which, when intro-

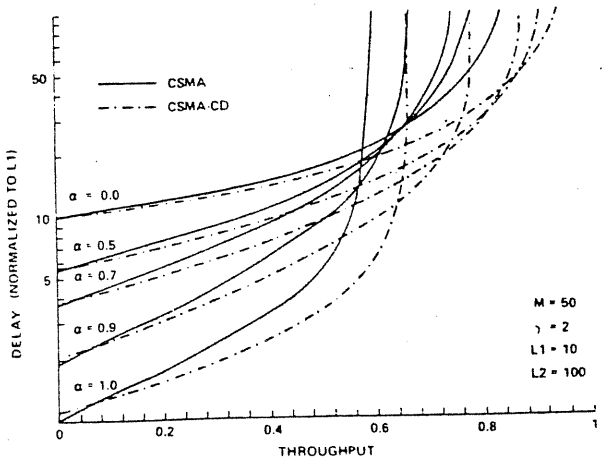


Fig. 13. Average Packet Delay Versus Channel Throughput for Various Values of α (Dual Packet Size).

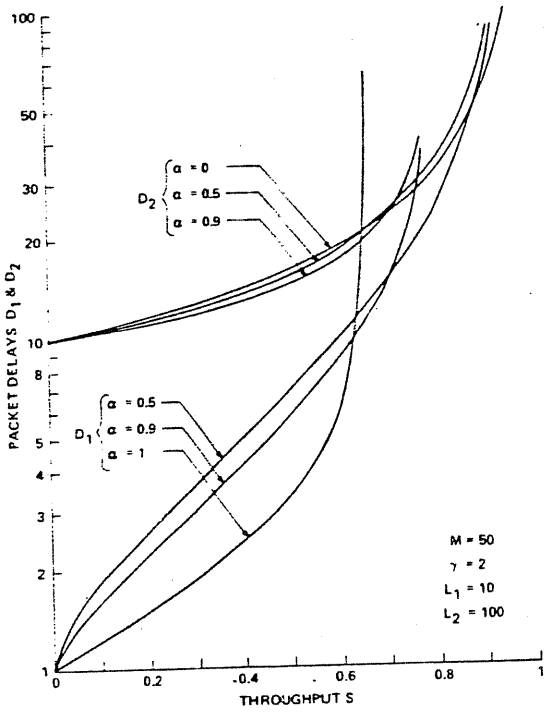


Fig. 14. Packet Delays for Short and Long Packets Versus Channel Throughput for Fixed α.

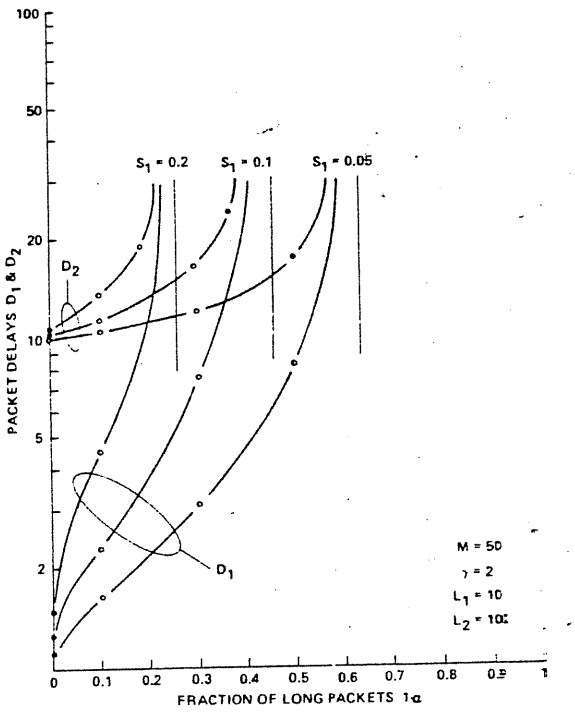


Fig. 16. D_1 and D_2 Versus $1 - \alpha$ for Constant S_1.

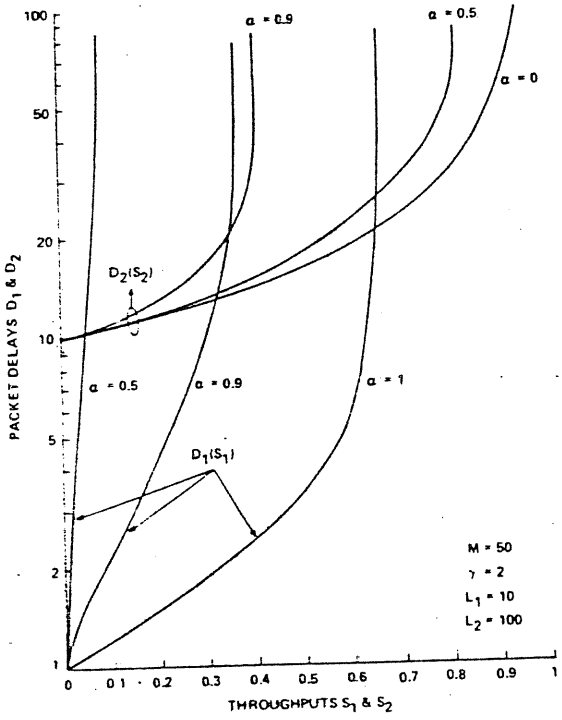


Fig. 15. Throughput-Delay Characteristics for Short and Long Packets.

duced, allow to recover an important fraction of the excess capacity. We inquire now as to the behavior and relative importance of the system performance measures with respect to each of the two packet sizes. Let S_1 (S_2) and D_1 (D_2) denote the throughput and

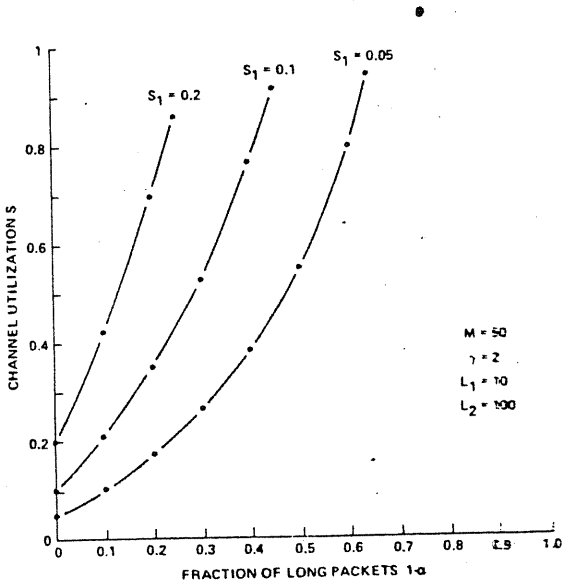


Fig. 17. Channel Throughput S Versus $1 - \alpha$ for Constant S_1.

packet delay for short (long) packets, respectively. In fig. 14 we plot for CSMA-CD, D_1 and D_2 versus S ($S_1 + S_2$) for various values of α and $M = 50; L_1 = 10; L_2 = 100; \gamma = 2$. As pointed out in the previous section the difference between D_2 and D_1 for a given value of α is always $L_2 - L_1$. For a given global achievable channel utilization S, D_1 and D_2 increase as the fraction of long packets increases in the mix; indeed the presence of long packets increases the waiting time W (the time to acquire the channel). In fig. 15, we plot D_1 and D_2 versus S_1 and S_2 respectively for various values of α, illustrating the degradation in throughput-delay tradeoff for short packets as the fraction of long packets $1 - \alpha$ increases. The throughput-delay tradeoff for long packets, however, improves.

Consider now a channel required to support interactive traffic at some level S_1. Certainly S_1 has to be lower than the channel capacity at $\alpha = 1$. Assume that S_1 is at some low level (e.g., 0.05 to 0.2). The introduction of long (file transfer) packets in view of achieving a higher channel utilization has the negative effect of significantly increasing the delay for the interactive traffic. This is illustrated in fig. 16 where we plot D_1 versus $1 - \alpha$ for fixed values of S_1. Clearly the channel utilization increases with $1 - \alpha$ as shown in fig. 17 where we plot S versus $1 - \alpha$ for fixed S_1. Thus in summary as the traffic mix includes more and more long packets, the overall channel capacity is improved in favor of long packets and to the detriment of the throughput-delay performance of short packets, indicating the need for priority schemes to maintain good performance for interactive traffic.

5. Conclusion

We extended the models used in the analysis of CSMA to cover the cases of collision detection and variable size packets. It was shown that the throughput-delay characteristics of CSMA-CD are better than the already highly efficient CSMA scheme. We characterized the improvement in terms of the achievable channel capacity and of the packet delay at a given channel utilization as a function of the collision detection time. Furthermore we established the fact that in uncontrolled channels (i.e., with a fixed average retransmission delay) CSMA-CD is more stable than CSMA, in that with CSMA-CD both channel capacity and packet delay are less sensitive to variations in the average retransmission delay.

We then studied the performance of these schemes in presence of variable size packets. Numerical results have been obtained for the interesting case of dual packet size. It was shown that a small fraction of long packets is sufficient to recover a channel capacity close to the (higher) capacity achieved with only long packets. However the improvement experienced by the introduction of long packets is in favor to the latter and to the detriment of the throughput-delay performance of short packets, establishing the necessity to design and implement priority schemes.

Acknowledgments

The authors would like to acknowledge the Stanford Linear Accelerator Center for providing the computer time used in this study.

Appendix

A. Variable Packet Size CSMA Without Collision Detection

All previous analyses of CSMA have dealt with fixed packet size [7–10]. In order to compare the performance of CSMA-CD to that of CSMA in presence of variable size packets, we undertake here the analysis of the latter. An important factor contributes to the complexity of an exact analysis. Contrary to CSMA-CD, the length of a busy period here is a function of the number of contending devices and their packet lengths. Accordingly, the backlog at an embedded point is a function of not only the backlog at the previous embedded point but also on the length of packets in the backlog. Conversely, the packet length distribution for those packets in the backlog is correlated with the number of such packets. For the sake of tractability we consider an approximate analysis based on removing this correlation by continually redrawing the lengths of packets independently from the packet length distribution [5].

Let $N^te = i$ be the state of the system at some embedded point t_e; let k denote the number of backlogged devices at the start of the corresponding transmission period (that is, $k - i$ new devices have joined the backlog in the last slot of the idle period). Let B be the random variable representing the number of devices simultaneously transmitting. If the transmission period is successful, then $B = 1$ with probability one. Given $N^te = i$ and given that the transmission period is unsuccessful, the distribution of B is given by

[5] This assumption was made by Ferguson in the analysis of pure-ALOHA which exhibits a similar correlation; the validity of the assumption in the context of pure-ALOHA was verified by simulation [13].

$P_B(b|i;\text{failure};k-i) \triangleq$

$\Pr\{B = b \,|\, N^{t_e} = i \text{ and transmission unsuccessful and } k-i \text{ new arrivals}\}$

$$= \begin{cases} \binom{i}{b-k+i} \nu^{b-(k-i)}(1-\nu)^{k-b} & k-i \geq 2;\ k-i \leq b \leq k \\ \dfrac{\binom{i}{b-1}\nu^{b-1}(1-\nu)^{i-b+1}}{1-(1-\nu)^i} & k-i=1;\ 2 \leq b \leq k \\ \dfrac{\binom{i}{b}\nu^b(1-\nu)^{i-b}}{1-i\nu(1-\nu)^{i-1}-(1-\nu)^i} & k-i=0;\ 2 \leq b \leq k \\ 0 & \text{otherwise} \end{cases} \tag{A1}$$

The length of the busy period, denoted by T_{max}, is equal to the maximum length among all B packets. Given $B = b$, the distribution of T_{max} is given by

$$P_{T_{max}}(t|b) \triangleq \Pr\{T_{max} \leq t | B = b\} = [\Pr\{T \leq t\}]^b . \tag{A2}$$

It is thus clear from the above discussion that the length of the busy period is a function of the state of the system in slot $t_e(N^{t_e} = N^{(t_e+I-1)} = i)$ and in slot $t_e + I(N^{t_e+I} = k)$. Given the two latter conditions, and given that $T_{max} = t$, the state of the system at the next embedded point is j with probability $[Q^{t+1}]_{kj}$. Therefore, removing all conditions, the (i, j)th element of the transition matrix P is now given by

$$P_{ij} = [SG_T(Q)\,QJ]_{ij} + \sum_{k=i}^{M}\left\{\sum_{b=k-i}^{k}\left[\sum_{t=1}^{\infty} f_{ik}(Q^{t+1})_{kj} P_{T_{max}}(t|b)\right] \times P_B(b|i;\text{failure};k-i)\right\} . \tag{A3}$$

Similar considerations lead to the following expressions for the stationary channel throughput and backlog:

$$S = \frac{\sum_{i=0}^{M} \pi_i P_s(i)\,\bar{T}}{\sum_{i=0}^{M} \pi_i \left[\frac{1}{1-\delta_i} + 1 + P_s(i)\,\bar{T} + \bar{T}_{max}(i)\right]} , \tag{A4}$$

$$\bar{N} = \frac{\sum_{i=0}^{M} \pi_i \left[\frac{i}{1-\delta_i} + A(i)\right]}{\sum_{i=0}^{M} \pi_i \left[\frac{1}{1-\delta_i} + 1 + P_s(i)\,\bar{T} + \bar{T}_{max}(i)\right]} , \tag{A5}$$

where

$$\bar{T} = \sum_{t=1}^{\infty} t \Pr\{T = t\} \tag{A6}$$

$$\bar{T}_{max}(i) = \sum_{k=i}^{M}\left[\sum_{b=k-i}^{k}\left(\sum_{t=1}^{\infty} t f_{ik} P_{T_{max}}(t|b)\right) \times P_B(b|i;\text{failure};k-i)\right] \tag{A7}$$

$$A(i) = \sum_{j=i}^{M} j \left(\underline{S}\sum_{l=0}^{\bar{T}} \underline{Q}^l\right)_{ij} + \sum_{j=i}^{M} j \left\{\sum_{k=i}^{M}\left\{\sum_{b=k-i}^{k}\left[\sum_{t=1}^{\infty} f_{ik}\left(\sum_{l=0}^{t} Q^l\right)_{kj} P_{T_{max}}(t|b)\right] \times P_B(b|i;\text{failure};k-i)\right\}\right\} \tag{A8}$$

Under the independence assumption made in analyzing CSMA with variable size packets, the delay obtained by the ratio $\bar{N}/S$ is normalized with respect to $\bar{T}$. Moreover under this assumption equations (19), (20), and (21) hold here too.

B. Derivation of Channel Capacity Using the Infinite Population Model

Here all cycles are statistically identical. The average time during a cycle that the channel is carrying a valid transmission is simply $\bar{U} = \bar{T}ge^{-g}/(1-e^{-g})$. The average idle period is, as before, $\bar{I} = e^{-g}/(1-e^{-g})$. The distribution of B is given by $\Pr[B = b] = g^b e^{-g}/b!(1-e^{-g})$. Given $B = b$, the average transmission period is

$$\overline{TP} = \sum_{t=1}^{\infty} [1 - (\Pr\{T \leq t\})^b] . \tag{A9}$$

Therefore the throughput is given by

$$S = \frac{\bar{T}g}{1 + \sum_{b=1}^{\infty}\sum_{t=1}^{\infty} (g^b/b!)[1-(\Pr\{T \leq t\})^b]} . \tag{A10}$$

References

[1] D.G. Willard, Mitrix: A Sophisticated Digital Cable Communications System, Proceedings of the National Telecommunications Conference, November 1973.

[2] A.G. Fraser, A Virtual Channel Network, Datamation, February 1975.

[3] D.J. Farber, et al., The Distributed Computing System, Proceedings of the 7th Annual IEEE Computer Society International Conference, February 1973.

[4] D.J. Farber, A Ring Network, Datamation, February 1975.

[5] R.M. Metcalfe and D.R. Boggs, Ethernet: "Distributed Packet Switching for Local Computer Networks," Communications of the ACM, vol. 19, no. 7, pp. 395–403, 1976.

[6] N. Abramson, The ALOHA system – Another alternative for computer communications, in 1970 Fall Joint Comput. Conf. AFIPS Conf. Proc., vol. 37, Montvale, NJ: AFIPS Press, 1970, pp. 281–285.

[7] L. Kleinrock and F.A. Tobagi, Packet switching in radio channels: Part I – Carrier sense multiple-access modes and their throughput-delay characteristics, IEEE Trans. Commun., vol. COM-23, pp. 1400–1416, Dec. 1975.

[8] F.A. Tobagi and L. Kleinrock, "Packet Switching in Radio Channels: Part II – The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution," IEEE Trans. Commun., vol. COM-23, pp. 1417–1433, December 1975.

[9] F. Tobagi and L. Kleinrock, Packet switching in radio channels: Part IV – Stability considerations and dynamic control in carrier sense multiple access, IEEE Trans. Commun., vol. COM-25, pp. 1103–1120, October 1977.

[10] F. Tobagi, et al., Modeling and Measurement Techniques in Packet Communication Networks, IEEE Proceedings, pp. 1423–1447, November 1978.

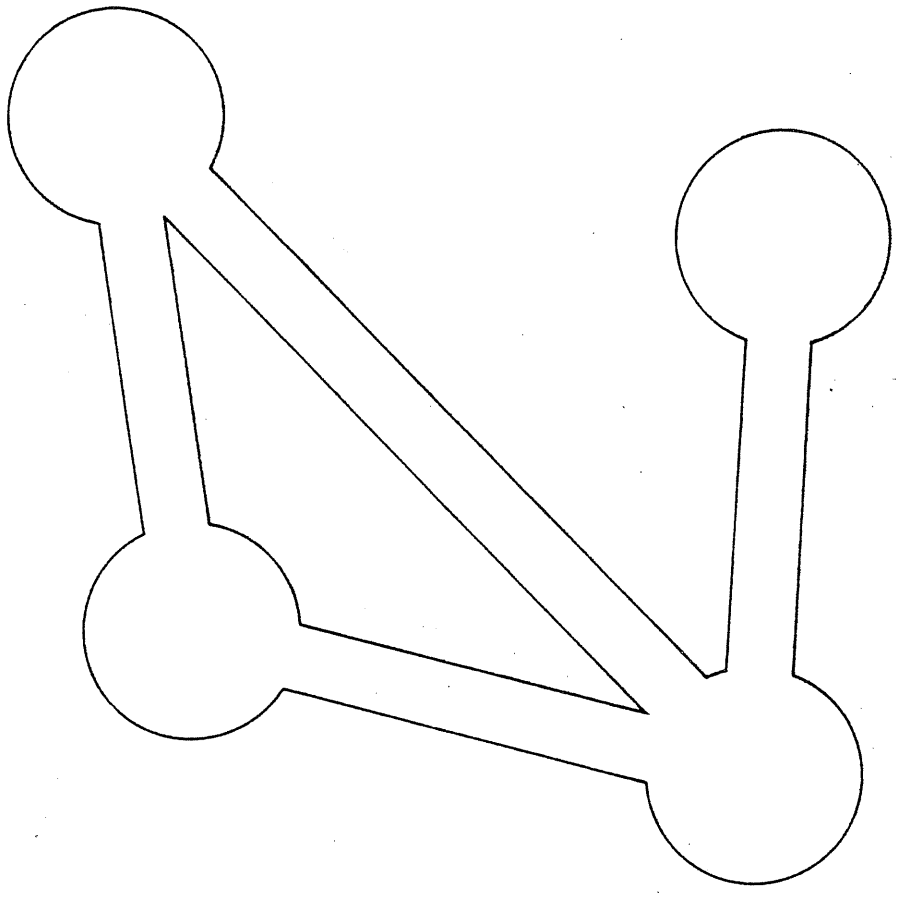
[11] F. Tobagi, Random access techniques for data transmission over packet switched radio networks. Ph.D. dissertation, Comput. Sci. Dep., School of Eng. and Appl. Sci., Univ. of California, Los Angeles, rep. UCLA-ENG 7499, December 1974.

[12] J. Little, "A Proof of the Queueing Formula L = λW," Operation Res., vol. 9, pp. 383–387, March-April 1961.

[13] M.J. Ferguson, An approximate Analysis of Delay for Fixed and Variable Length Packets in an Unslotted ALOHA Channel, IEEE Trans. Commun., pp. 644–654, July 1977.

[14] L. Kleinrock and S.S. Lam, Packet Switching in a Multiaccess Broadcast Channel: Performance Evaluation, IEEE Trans. Communications, pp. 410–423, April 1975.

[15] J.F. Shoch and J.A. Hupp, Performance of an Ethernet Local Network – A Preliminary Report, Proceedings of the Local Area Communication Network Symposium, Boston, May 1979.



APPENDIX B: Amer paper

A Measurement Center for the NBS Local Area Computer Network

PAUL D. AMER

Abstract—This paper describes a measurement center for the NBSNET, a distributed, broadcast local area computer network (LAN) at the National Bureau of Standards. A LAN measurement center allows careful testing and evaluation of a network under normal and varying user-defined conditions. The measurement center consists of three components: an artificial traffic generator, a monitoring system, and data analysis software. The traffic generator emulates varied loads on the network, allowing for controlled experimentation and functional testing. The monitoring system captures measurement information about both artificial and normal network traffic. Analysis software summarizes this information into ten measurement reports following each monitoring period. Implementation issues and problems are discussed.

Index Terms—Broadcast network, CSMA/CD, local network, measurement, multiple access channel, performance evaluation.

I. Introduction

A LOCAL area computer network (LAN) is broadly defined in [7] as a set of "computers" whose communication takes place over limited distances between 10 and 10 000 m. Besides micros, minis, and large-scale systems, the set of computers includes terminals, line printers, and other devices requiring and/or providing transmission of data. The communication is accomplished via a variety of media including twisted pairs, coaxial cable, radio broadcast, and fiber optics. Many business applications are ideally suited for LAN technology. For instance, LAN's are recognized as a cost-effective approach to office automation, handling applications such as electronic mail, word processing, and information retrieval.

A recent study performed for the National Bureau of Standards (NBS) predicts LAN's to show the largest growth of all forms of computer-communications over the next decade. Estimations are that by 1985 over 1400 LAN's will be installed by the U.S. Government alone to support intrasite networking communications. The total present-value dollars to be spent on this communications support during the 1981–1985 time period, plus the budgeted dollars in 1985 for future support, will be between $75 million and $117 million [11].

In implementing a LAN, questions arise regarding network functionality and performance. For operational networks, performance can be investigated by designing a local area network measurement center (LAN-MC). A LAN-MC facilitates careful testing and evaluation of a network under both normal and controlled conditions. Such testing is essential for effective use of networks and for their improved future design.

This paper describes a LAN-MC implemented by NBS for the NBSNET. NBSNET consists of two distributed, broadcast LAN's at the Gaithersburg, MD and Boulder, CO facilities connected with a 9600 Bd link. A major component of this MC is a set of ten measurement reports which summarize NBSNET traffic during a monitored period. Although most applicable to broadcast networks such as NBSNET, many of these reports are independent of the underlying network topology and would be useful in designing measurement centers for other LAN architectures. Before proceeding with a description of NBSNET and the reports, we describe the general components of a MC.

II. A LAN Measurement Center

A LAN measurement center has three components: a monitoring system, data analysis software, and an artificial traffic generator. The monitoring system gathers measurement information, such as the size of a message being transmitted, and prepares the information for statistical analysis. Much is known about the characteristics, advantages, and disadvantages of computer system monitors [13]. Based on this knowledge, the best monitor for a LAN is a hybrid monitor employing software to relate network state transitions with their stimuli, and hardware to minimize any overhead network traffic which results from the monitoring activity. Microscopic analysis measures activities in milliseconds and microseconds, thereby characterizing channel activities where communication is on the order of 1 Mbit/s.

The software analysis component of a measurement center summarizes the information captured by the monitoring system either at the conclusion of a measurement period for off-line analysis or during measurement for on-line dynamic analysis. These summaries are statistical overviews of the LAN traffic in the form of performance reports. They provide information such as network delays, traffic distributions, and types of traffic transmitted. This information both supports functional testing of the network and its components and documents LAN performance under varying protocols and other tunable system parameters.

Finally, an artificial traffic generator places varied traffic

Manuscript received March 4, 1981; revised July 27, 1981 and March 4, 1982. This work was supported in part by a grant from the University of Delaware Research Foundation.

The author is with the Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 20234 and the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19711.

0018-9340/82/0800-0723$00.75

loads on a network, thereby allowing for controlled experimentation. For example, a generator can emulate increasing amounts of traffic to observe a network's performance limits under increasing stress. Stress problems therefore can be confronted early in a network's implementation before production activities make modifications costly.

III. National Bureau of Standards NBSNET

A. Overview

NBSNET is a local area broadcast network which has been operational since October 1979. Users in 20 buildings are connected with the most distant pair separated by approximately 1.5 km. NBSNET employs a carrier sense multiple access with collision detection (CSMA-CD) protocol similar, but not identical to the Ethernet [10].

Logically, NBSNET consists of a single 1 Mbit/s coaxial cable (channel) with multiple ports. Each port consists of a microprocessor-based interface node called Terminal Interface Equipment (TIE). The TIE, described in detail in [1], is programmed to adapt each user device to the network. The TIE has three main components: the user board, the network board, and the TAP.

User boards handle communication between user devices and the network board. In turn, the network board controls the communication to and from the coaxial cable. Output from the network board is coupled to the coaxial distribution cable by the TAP. Each network board supports one to eight user boards. It continuously polls the user boards until one requesting transmission is found. Discussion of the information control flow between users by way of the TIE's is available in [2].

B. Communication Protocol

The following is a discussion of the CSMA-CD protocol as implemented on NBSNET. Key words are italicized. Their definitions are required for understanding the 10 performance reports described in Section IV.

NBSNET employs a *1-persistent protocol* [9]. When a user board has a *packet* to transmit, the board *contends* for the channel. Two cases are possible: 1) if no carrier is sensed, the user *accesses* the channel and begins transmitting, or 2) if the channel is busy with another user's transmission, the user *defers* or waits until the channel becomes idle, at which time it automatically transmits.

In both cases a potential exists for two or more users to overlap their transmissions thereby resulting in a *collision*. A collision can occur in case 1) because the signal propagation delay between users allows for multiple users to sense the channel idle at approximately the same time. A collision can occur in case 2) if two or more users simultaneously defer to another user's transmission. With a 1-persistent protocol, these deferring users are assured of colliding. (Other persistent protocols avoid this problem [9].)

With the TAP hardware, a user listens to its own transmission and detects if it is colliding. Whenever a collision is detected, the user aborts transmission, jams the channel with a special signal to ensure that all other users also detect the collision, and reschedules its transmission according to a randomized backoff distribution. Backoff is randomized to prevent repeated collisions.

If a user's transmission propagates beyond all other users without colliding, the user *acquires* the channel. Case 2) guarantees that no collision can occur for the duration of that transmission. When a user acquires the channel and successfully sends an entire packet, that packet is referred to as a *transmission*.

NBSNET employs a simple positive acknowledgment scheme with a window size of one. A destination, upon receiving a packet, generates an acknowledgment packet, or *piggybacks* an acknowledgment onto another outgoing packet. At that time the received packet has been communicated and is also considered to be a *communication*. (Note that transmission and communication have special meaning in this paper.) For various reasons, such as electrical noise, faulty hardware, or full buffers at the destination, a transmission may not be received and therefore not be a communication. Both transmission and communication imply successful placement of a packet onto the channel, but only the latter implies successful receipt at the destination.

When an acknowledgment is not received from the destination within a predetermined time-out period, the source will retransmit the packet. Hence, each transmission (packet) is either an *original* or a *duplicate* transmission (packet). For most packet types, NBSNET allows up to 7 duplicate transmissions before assuming a malfunction or failure at the destination and breaking the connection. The maximum number of duplicates permitted and the time-outs between duplicate transmissions are tunable protocol parameters.

If a packet is communicated and its acknowledgment is lost, the source will transmit the packet again. All additional transmissions which take place after a packet has been communicated are *redundant* transmissions.

C. Design Issues of Measurement

Local area network behavior is described in terms of discrete events. Measurement tools therefore should be capable of detecting these events and the times when they occur. Svoboda [16] and Ferrari [6] describe the structure, strengths, and weaknesses of various measurement tools. In general, there are four design issues of importance: artifact, location of measurement, traffic generation, and on-line versus off-line analysis. Each is discussed in turn.

1) Artifact: Artifact is the interference on a target system caused by the introduction of a monitoring device [13]. Typically, a portion of a system's resources is allocated to measuring how itself and other resources are performing. For example, if CPU utilization is monitored at 60 percent, perhaps 55 percent is for processing the workload and 5 percent is for controlling the monitor.

Besides minimizing artifact, it is important to know the magnitude of the artifact introduced. When known, this interference can be removed from the final measurements, thus providing an unbiased indication of performance. It may be better to increase artifact to gain additional knowledge of its magnitude.

2) *Location of Measurement:* There are three approaches to measuring the activities of a local area network: centralized, decentralized, and hybrid measurement. These approaches are described below.

a) Centralized Measurement: A broadcast network lends itself naturally to a centralized measurement approach [Fig. 1(a)]. Acting in a "promiscuous" mode [10], a modified interface tapping onto the channel can monitor all packets on the network. Therefore, all packet header information as well as additional timing and count information (e.g., interarrival time since last packet, packet size) is available to a central monitor. This approach was employed by Shoch and Hupp in their performance study of an ETHERNET local network [14].

Centralized measurement introduces no artifact and tends to be less costly than other approaches. However, some desirable information cannot be monitored centrally. The timing of when a packet arrives at an interface (from a user device) and the amount of time a user board defers in transmitting a packet are only available at that user board. Similarly, although a centralized monitor can detect a collision, it cannot determine which and how many user boards were involved.

With central measurement, some timings, such as the arrival of a packet onto the network, are biased. An arrival recorded by a central monitor represents the moment when the packet reaches the monitor's TAP, not when the packet enters the channel. To eliminate bias, a central monitor could account for its physical distance from each interface and the propagation delay for signals to cover this distance.

b) Decentralized Measurement: In a decentralized measurement approach [Fig. 1(b)], additional memory and real time clocks are incorporated at each network interface. Packets arriving at or departing from an interface can be appropriately time stamped. Information such as collision induced delays and collision counts, which are not available centrally, can be recorded at each interface.

Periodically, each node must transmit its information to a central location for consolidation. In a pure decentralized measurement approach, the central collector does no monitoring; only data collection and reduction. The periodic transmission of measurement information from each node may be as frequent as with every packet (by including measurement information in the transmitted packet), after every r time units (time driven), after every n events (event driven), or upon request from the central collector.

With a decentralized approach all information about the network traffic is available. This includes precise times for transmission and receipt of packets and the source/destination addresses of all colliding packets. Each interface captures the demands placed on the network as they occur and not after a variable delay as with a central measurement device.

Although more accurate, decentralized measurement requires overhead communication for the periodic transmission of data to the central collector. If the transmissions are sent over dedicated lines, extra costs are involved. If sent over the main channel, these transmissions introduce artifact. Artifact can be reduced by less frequent transmission of measurements and by local data reduction (e.g., histograms). However, these alternatives require additional storage and intelligence, respectively, either of which may be prohibitive.

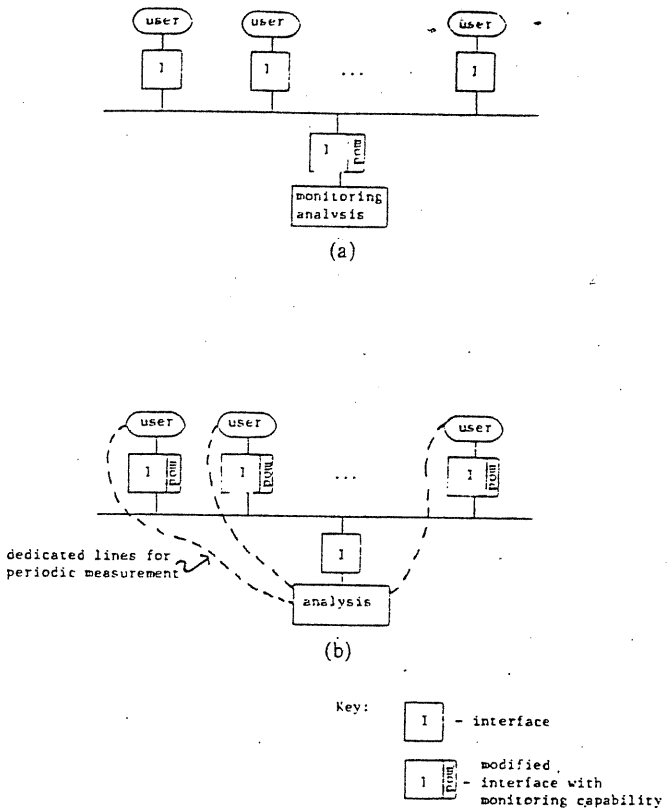


Fig. 1. Location of measurement in a broadcast network. (a) Centralized. (b) Decentralized.

Depending on experimental objectives, decentralized measurement may require synchronization of interface timers. This problem is a classical one and is discussed in [5]. Finally, because decentralized measurement requires modifications at every interface, implementation and maintenance tend to be more costly than with centralized measurement.

c) Hybrid Measurement: Because of the advantages and disadvantages of centralized and decentralized measurement, a hybrid approach was chosen for monitoring the NBSNET (Fig. 2). As much information as possible is collected centrally. Minimal modifications were made to all user interfaces to allow local measurement collection and reduction. These measurements are transferred over the channel to a central site at the termination of each logical connection. Additional modifications were made to those user boards which act as artificial traffic generators. These boards collect timing and collision information (only about artificial traffic) and periodically transmit them to a central site over special inexpensive lines so as not to interfere with the main channel.

A hybrid approach allows accurate and comprehensive measurement. One disadvantage is the complexity of coordinating the analysis of decentralized and centralized measurement. Careful planning minimizes this problem. Independent of the location of measurement, a LAN-MC can be designed for complete or partial measurement. For complete measurement packets must not arrive faster than the measurement system can process them. If packets arrive too frequently, only a sample of the traffic can be collected. This

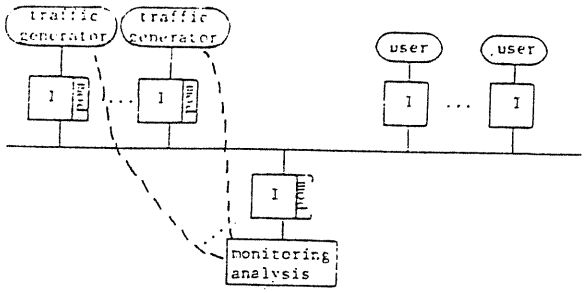


Fig. 2. Hybrid measurement of NBSNET.

significantly affects the algorithms which analyze the data. Timing considerations for the NBSNET measurement system indicate complete measurement is possible [15].

3) Artificial Traffic Generation: Analysis of LAN performance requires the ability to generate known artificial traffic loads on a system. Traffic generators are beneficial for two reasons. Normal traffic on a new LAN is typically quite low and rarely stresses its capacity. With traffic generators, however, it is possible to emulate high load conditions. Therefore, network testing and debugging can be accomplished before the network is burdened with production activities when corrections are expensive. Second, traffic generators can produce repeatable traffic patterns. Repeatability facilitates comparison investigations, such as the effect of different communication protocols on network performance.

Traffic generation for NBSNET is accomplished by connecting eight modified user boards to the channel. These generators are programmed to: 1) generate packets with a constant, uniform, or Poisson size distribution, 2) generate packets with constant, uniform, or exponential interarrival times, 3) direct packets to any specified destination, 4) communicate with the monitoring system to synchronize traffic generation and data collection, and 5) permit on-line experimenter control.

4) On-Line Versus Off-Line Analysis: Measurement records generated by a monitoring system can be placed in mass storage for off-line future analysis or summarized on the run for on-line monitor display. Current analysis of NBSNET is off-line. When packets are transmitted at a rapid rate, there is limited time to assimilate the data and to simultaneously maintain meaningful statistics on network activity. Except for simple counts, on-line analysis (i.e., performance summaries every 1-2 s) for LAN's is very difficult. This reduces, but does not eliminate, the potential for a complex network feedback system which dynamically adjusts tunable network parameters in response to changing performance and/or workload conditions. NBSNET provides off-line analysis with a delay on the order of 5-10 min following a measurement period.

IV. Measurement Reports

A major problem in designing a local area network measurement center (LAN-MC) is deciding what measurements to collect within a monitored period and what statistics to report. One approach is to measure everything. This implies deriving a database from which the original traffic can be reconstructed completely. Total measurement avoids redesign costs which occur if useful information is omitted from the initial implementation. For most LAN's, however, a total measurement approach requires a prohibitive amount of storage.

A more realistic approach initially determines what managerial and research questions are to be answered by measurement, and what performance reports need be generated. Then one determines the specific network information necessary to satisfy these questions and reports. Finally, the user decides upon an implementation for capturing the information. This is dependent on the network being monitored and the design issue of location of measurement.

Ten performance reports have been implemented for describing NBSNET traffic. The reports were derived in part from the above approach and in part from past experience in measuring long haul networks [3], [8]. They consist of statistics summarizing network activity during any given monitoring period. Each report is classified as either traffic characterization or performance analysis type. Traffic characterization reports indicate the workload placed on the network. This information is a primary source for functional testing of the network. Performance characterization reports indicate the time delays, utilizations, etc., which result from a given load and network configuration. They describe the dependent variables which are observed rather than controlled, and are used for tuning and performance comparisons. A comment section follows each report description where implementation issues are discussed.

A. Host Communication Matrix

The Host Communication Matrix indicates the traffic flow between connected user boards. Tabulated for each source-destination (s-d) pair are the (number, proportion) of (packets, data packets, data bytes) transmitted from s to d. Tabulated for each source (destination) are the (number, proportion) of (packets, data packets, data bytes) transmitted from s (to d). Reported as a summary of the total traffic are the number of (packets, data packets, data bytes) transmitted, the mean number of data bytes per data packet, and the proportion of packets which are data packets.

Comment: Source, destination, and packet type information is part of each packet's header. The number of bytes in a packet is counted by the MC hardware as the packet is tapped off the channel (type: traffic characterization).

B. Building (Group) Communication Matrix

The Building Communication Matrix indicates the traffic flow between buildings or any other user defined groupings of user boards. For example, a group may consist of all user boards connected to the same host. This table condenses the Host Communication Matrix by summing together several groups of rows (columns) into single rows (columns). All tabulated information is identical to that provided in the Host Communication Matrix.

Comment: The NBSNET MC has access to an address-building table which maps user addresses to buildings (type: traffic characterization).

C. Packet Type Histogram

The Packet Type Histogram indicates the distribution of each type of packet transmitted. In measuring NBSNET, packet types are limited only to those defined by the lowest level protocols. Tabulated for each type are (number, proportion) of packets. As summary information (and as a check against other reports), the total number of packets is reported.

Comment: There are currently 12 packet types on the NBSNET. The measurement center counts each packet type by observing the control field of every transmission (type: traffic characterization).

D. Data Packet Size Histogram

The Data Packet Size Histogram records the number and proportion of data packets of particular length classes. Tabulated for each length class *i-j* (say 8–15 bytes) are the (number, proportion) of data packets with (between *i* and *j*, up to *j*) data bytes, inclusive. Values between *i* and *j* estimate the packet size probability distribution, while values up to *j* estimate the cumulative distribution. Reported as summary information are the total number of data packets and data bytes and mean number of data bytes per data packet.

Comment: Data packets are distinguished from other packet types in the packet header control field (type: traffic characterization).

E. Throughput-Utilization Distribution

The Throughput-Utilization Distribution indicates the flow of bytes on the network. Both total bytes transmitted and information bytes communicated are measured. Information bytes are defined to be only those data bytes contained in communications. Overhead such as header bytes and bytes in unacknowledged data packets are not counted since they communicate no information at the packet level. Reported for each user address *i* are: 1) the user channel throughput (bytes per second in all transmissions from address *i*. This count includes synchronization and checksum bytes, but not bytes that are involved in collisions). 2) The user channel utilization (user channel throughput divided by channel capacity). 3) The user information throughput (information bytes communicated per second from address *i*). 4) The user information utilization (user information throughput divided by channel capacity).

Reported as summary information are the total (channel throughput, channel utilization, information throughput, information utilization), and the number of seconds in the measurement period.

Comment: Information statistics describe the beneficial usage of the channel. Information throughput is analogous to the transfer rate of information bits (TRIB), or the number of bits accepted by the receiver divided by time [4]. Total channel utilization excludes the periods when the channel has a signal which is interfered with by noise, faulty hardware, collisions, etc. In computing information throughput, the MC identifies unacknowledged packets by checking packet sequence numbers (type: performance analysis).

F. Packet Interarrival Time Histogram

The Packet Interarrival Time Histogram indicates the number of packet interarrival times which fall into particular time classes. An interarrival time is the time between consecutive carrier (network busy) signals. Reported for each time class *i-j* (say 11–15 ms) are the (number, proportion) of interarrival times (between *i* and *j*, up to *j*) time units, inclusive.

Comment: The MC has a separate timer with 10 μs resolution. This timer is recorded and reset to 0 whenever a carrier signal is monitored. For NBSNET many interarrivals will be less than 1 ms since acknowledgment packets tend to immediately follow receipt of data packets. Other spikes in this histogram are likely to represent service turnaround times for certain processes [14]. For NBSNET, special watchdog packets are transmitted approximately every 2 min to verify inactive connections. Therefore, the longest interarrival time is bounded as long as a connection on the network exists.

Since interarrival times are monitored centrally, they are biased by the propagation delays between interfaces and the MC. These delays are estimated as less than 10 μs. Also, the number of interarrival times will be underestimated since only one interarrival time is collected when multiple arrivals result in a collision. As a result, some times that are tabulated are overstated (type: performance analysis).

G. Channel Acquisition Delay Histogram

The Channel Acquisition Delay Histogram records the times spent by user boards contending for and acquiring the channel. A channel acquisition delay begins when a user board becomes ready to transmit a packet and ends when its first bit is transmitted onto the channel. Included is all of the time spent deferring due to a busy channel and the time recovering and backing off from one or more collisions. Indicated for various time classes *i-j* (say 50–100 μs) are the (number, proportion) of packets whose acquisition delay was (between *i* and *j*, up to *j*) time units, inclusive. Also reported are the total number of packets transmitted, and the (mean, standard deviation) channel acquisition delay time.

Comment: This table reflects the efficiency of a CSMA-CD protocol and the process in which the network board polls each of the user boards. The NBSNET MC collects channel acquisition delays only for packets originating at traffic generators. Modification of all user interfaces for collecting these times was prohibitive. When a packet's transmit flag is set ready for transmission, a timer is reset to 0. When the packet is transmitted (successfully), the timer value is stored locally and then appended to the next packet transmitted (type: performance analysis).

H. Communication Delay Histogram

The Communication Delay Histogram indicates the delays that user boards incur in communicating packets to their destinations. A communication delay begins when an original packet becomes ready for transmission and ends when that packet is received (i.e., communicated) by the destination (which may be several transmissions later). One communi-

cation delay exists for each packet communicated. This delay may include several channel acquisition delays.

For completeness, a packet which is never acknowledged and therefore transmitted the maximum number of allowable times, causes an entry in this histogram. This permits analysis of the maximum number of transmissions parameter in terms of time delay. Reported for each time class *i-j* (say 0.5–1 s) are the (number, proportion) of packets which are communicated or which surpass the maximum number of allowable transmissions (herein referred as table entries) whose communication delay time is (between *i* and *j*, up to *j*) time units, inclusive. Also reported are the (total number, mean communication delay, standard deviation communication delay) of table entries, packets which are communicated, and packets which surpass the maximum number of allowable transmissions).

Comment: By definition, a communication delay excludes the time to generate and communicate an acknowledgment packet back to the original sender. Since the destination begins processing a received packet before (or while) it handles acknowledgment, this time does not reflect the delay in communicating a packet.

As with channel acquisition delays, the NBSNET implementation only captures communication delays for those packets originating at traffic generators. When a buffer's transmit flag is set ready for an original packet, a timer is reset to 0. Each time the last bit of a buffer is transmitted onto the channel, the timer value is saved. When a transmission is acknowledged, the saved value is used to update a local histogram. The histogram is transmitted to a central location after the monitoring period ends. Since No-op packets are not acknowledged, their communication delays are not recorded.

With this decentralized approach, the communication timer value will not include the time to propagate a signal to the destination, nor will it include the destination's software latency to recognize it.

Some factors which influence the communication delay are: the acknowledgement time-out periods before packet retransmission, the maximum number of allowable duplicate transmissions before a connection is broken automatically, and the hardware reliability.

A communication delay differs from the "one hop" delay time defined for measurement of a Packet Radio network (PRnet) [17]. One hop delay is the time interval between when a packet is ready for transmission and its corresponding acknowledgment packet is received. Hence, a one hop delay time includes the time for the destination to communicate its acknowledgment back to the source, while a communication delay time does not (type: performance analysis).

I. Collision Count Histogram

The Collision Count Histogram tabulates the number of collisions a packet of any type encounters before being transmitted. Reported for each collision count value *i* are the (number, proportion) of packets which incurred (*i*, *i* or fewer) collisions before being transmitted. Reported as a summary are the total number of (packets transmitted, collisions) and the (mean, standard deviation) number of collisions per transmission.

Comment: Like the Packet Interarrival Time Histogram, this histogram indicates the efficiency of a CSMA-CD protocol in allowing interfaces to acquire the channel. With NBSNET's 1-persistent protocol, the number of channel contentions for a packet should be one more than the number of collisions incurred in getting the packet transmitted.

Collision count information is available only for artificially generated packets. A collision count is maintained in the header of each packet to be transmitted. Each time a collision is detected, the traffic generator increments the collision count field. When a transmission occurs, the count is sent out as part of the packet and is then monitored by the MC (type: performance analysis).

J. Transmission Count Histogram

The Transmission Count Histogram indicates the number of times a packet is transmitted (original transmission plus duplicate transmissions) before it is communicated to its destination. Redundant transmissions as defined in Section III-B are not included in the histogram since they occur after communication has taken place. Reported for each transmission count *i* are the (number, proportion) of packets whose communication required (*i*, *i* or fewer) transmissions. Reported as a summary are the total number of (packets communicated, packets transmitted, redundant transmissions, piggybacked acknowledgments, No-op packets) and the (mean, standard deviation) number of transmissions required for a communication.

Comment: By observing packet sequence numbers, the MC recognizes the first through last times a particular packet is transmitted and which transmission is the communication. When consecutive transmissions from a particular source-destination pair have identical sequence numbers, then all but the first are duplicate transmissions. The transmission just prior to an acknowledgment is considered the communication. Under ideal conditions, the number of transmissions required per communication is 1 and the number of redundant transmissions is 0.

By counting the number of redundant transmissions, this report indicates what Tobagi *et al.* [17] call the "echo acknowledgment efficiency." With any positive acknowledgment protocol, an interesting metric is how often packets are retransmitted because the acknowledgment was lost even though the packet was communicated (type: performance analysis).

V. Conclusions

Two areas of investigation are planned following verification of the NBSNET-MC. The first area will investigate existing NBSNET low traffic loads for the purpose of uncovering possible errors or inefficiencies. Questions to be answered include the following.

- Is traffic evenly distributed among the network users or are there source-destination pairs with unusually heavy traffic?
- What is the percentage of each type of packet? Are some packet types of unusually high frequency indicating an error or an inefficient protocol?

- What is the distribution of data packet sizes? Are variable size data packets worth the additional overhead or would fixed size packets suffice?
- What are the channel acquisition and communication delay distributions? Are these times excessive?
- Are collisions a factor in getting packets transmitted, indicating possible faulty hardware or protocols?
- What is the information utilization and throughput? How do the information statistics compare with the channel statistics?

The second area will investigate the effects of increasing traffic load and varying packet sizes on network performance. Questions to be answered include the following.

- What is the effect of traffic load on utilization, throughput, and time delays? When, if ever, does traffic load start to degrade system performance?
- Defining a stable network as one whose utilization is a nondecreasing function of traffic load, what is the tradeoff between stability, throughput, and delay?
- What is the maximum capacity of the channel under normal operating conditions? How many active users are necessary to reach this maximum?
- Do larger packets increase or decrease throughput and delay?
- How does constant packet size affect utilization and delay?

Answering these and other questions will help support or dispute the many analytic and simulation results on LAN performance. With an existing network and existing measurement system, it is feasible to determine the validity of the assumptions and simplifications of these past studies.

ACKNOWLEDGMENT

The author gratefully appreciates the active roles that R. Rosenthal, D. Stokesberry, R. Toense, M. L. Fahey, and D. Rorrer played in the design and implementation of the NBSNET Measurement Center.

REFERENCES

[1] R. J. Carpenter, J. Sokol, Jr., and R. Rosenthal, "A microprocessor-based local network node," in *Proc. IEEE COMPCON*, Fall 1978, pp. 104-109.
[2] R. J. Carpenter and J. Sokol, Jr., "Serving users with a local area network," *Comput. Networks*, Oct. 1980.
[3] G. Cole, "Performance measurements on the ARPA computer network," *IEEE Trans. Commun.*, vol. COM-20, pp. 630-636, June 1972.
[4] I. W. Cotton, "Criteria for the performance evaluation of data communications services for computer networks," Nat. Bureau of Standards, Washington, DC, NBS Tech. Note 882, Sept. 1975.
[5] C. Ellingson and R. Kulpinski, "Dissemination of system time," *IEEE Trans. Commun.*, vol. COM-21, May 1973.
[6] D. Ferrari, *Computer Systems Performance Evaluation*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
[7] IFIP Working Group 6.4 on Local Computer Networks, *Statement of Aims and Scope*, June 1979.
[8] L. Kleinrock and W. Naylor, "On measured behavior of the ARPA network," in *Proc. Nat. Comput. Conf.*, AFIPS, vol. 43, 1974, pp. 767-780.
[9] L. Kleinrock and F. A. Tobagi, "Packet switching in radio channels: Part 1—Carrier sense multiple-access modes and their throughput-delay characteristics," *IEEE Trans. Commun.*, vol. COM-23, pp. 1400-1416, Dec. 1975.
[10] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. Ass. Comput. Mach.*, vol. 19, pp. 395-404, July 1976.
[11] Network Analysis Corp. "Impact assessment of standards for LAN: A final report prepared for U.S. Dept. of Commerce," FR.293.01R2, Sept. 1980.
[12] NBSNET User's Information, Nat. Bureau of Standards, Int. Document, Feb. 1980.
[13] G. Nutt, "Tutorial: Computer systems monitors," *Computer*, vol. 8, pp. 51-61, Nov. 1975.
[14] J. F. Shoch and J. A. Hupp, "Measured performance of an Ethernet local network," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 711-720, Dec. 1980.
[15] D. Stokesberry and R. Rosenthal, "The design and engineering of a performance measurement center for a local area network," in *Proc. Comput. Networking Symp.*, Washington, DC, Dec. 1980, pp. 110-115.
[16] L. Svobodova, "Computer system measurability," *Computer*, pp. 9-17, June 1976.
[17] F. Tobagi, S. E. Lieberson, and L. Kleinrock, "On measurement facilities in packet radio systems," in *Proc. Nat. Comput. Conf.*, AFIPS, vol. 45, 1976, pp. 589-596.

Paul D. Amer received the B.S. degree in mathematics from the State University of New York, Albany in 1974, and the M.S. and Ph.D. degrees in computer and information science from The Ohio State University, Columbus, in 1976 and 1979, respectively.

Currently, he is an Assistant Professor on the Department of Computer and Information Sciences at the University of Delaware, Newark. He is also a Faculty Research Assistant at the National Bureau of Standards. At the State University of New York, Albany, he was employed for three years in the Computing Center, with primary responsibility for interacting with faculty and graduate students who used the Center's statistical packages, such as BMD and SPSS. At Ohio State University, his primary research was in the area of computer system performance evaluation. He investigated problems in workload characterization and computer system/service selection employing empirical techniques, simulation, and statistical theory. He was also a Teaching Associate for which he received the 1979 $1000 Graduate Teaching Award. As a Research Assistant at the National Bureau of Standards, he placed his computer selection research within the context of government procurement of interactive computer services. He is also the author of two NBS Special Publications, one of which received the U.S. Department of Commerce Certificate of Recognition. Over the last two years his research has changed emphasis within the area of computer system performance evaluation from system selection to modeling, measurement, and analysis of local area computer networks. This change in direction was motivated by increased importance placed in local network technology by NBS and the predicted boom in the usage of local networks for office automation. His current research activities involve investigation of design issues for measuring any local area network architecture, specifying topology-independent local network performance metrics, and specifying a measurement and control center for a local broadcast network which employs a CSMA/CD communication protocol.

Dr. Amer is a member of the Association for Computing Machinery, SCS, SIGCOMM, SIGMETRICS, and SIGSIM.

APPENDIX C: Toense paper

Performance Analysis of NBSNET

Robert E. Toense
National Bureau of Standards

ABSTRACT The performance of NBSNET, a broadcast, packet switched, carrier sense multiple access with collision detection (CSMA/CD) local area network, is analyzed in terms of utilization, stability, delay, and fairness. Traffic generators transmit packets of known arrival rate and packet length distributions on an isolated network segment. The packets are recorded and time-stamped for analysis. Analysis of these empirical laboratory data shows that (1) utilization of the network under heavy and overloaded conditions approaches the theoretical limit and is predictable, (2) the network remains stable under the conditions observed, (3) the mean delay introduced by the network is predictable as a hyperbolic function of the observed channel utilization, and (4) the network is fair with uniformly distributed individual node utilizations.
Key words: computer network; CSMA/CD; delay; fairness; local area network; NBSNET; performance analysis; stability; utilization.

1. Introduction

NBSNET is a local area network that was designed at the National Bureau of Standards' Institute for Computer Sciences and Technology (ICST) and has been operational since October 1979 [1]. NBSNET is a 1 Mbit/sec, baseband, CSMA/CD network using a 1-persistent protocol and a fixed binary exponential backoff table.

Other studies of CSMA/CD networks have predicted system performance [2,3]. The following analysis verifies that the implementation of the NBSNET access method fulfills the expectations predicted by these studies. Empirical data are used to verify the expected performance of the network under various loading conditions, and the performance is characterized in terms of utilization, stability, delay, and fairness.

Initial performance data indicated that the maximum channel utilization was less than predicted for CSMA/CD networks, but close to that predicted for CSMA networks. A close examination of the signals on the cable revealed that some transmissions did not terminate when a collision occurred. The problem was traced to a design flaw that prevented detection of some collisions in the collision detection circuitry. The flaw was corrected and this study reflects the performance of the network with updated hardware.

The performance data for NBSNET was collected in the ICST Local Area Networking Laboratory using the measurement center described by Stokesberry and Rosenthal [4] and the measurement reports described by Amer [5]. The measurement center and a laboratory "stub" were disconnected from the network by removing power from an active repeater. This permitted running controlled experiments without affecting other network users. Data from the measurement center were further analyzed using "DATAPLOT," an interactive, high-level language for graphics, non-linear fitting, data analysis and mathematics, developed at the National Bureau of Standards by Filliben [6].

One node in the laboratory is modified to behave as a network monitor, capturing only header information for each packet transmitted over the network. The monitor also inserts a timing record into the packet header stream at one second intervals to time stamp

the data. The headers are then transmitted to a data acquisition system for later processing.

Six other nodes are modified to serve as traffic generators. They contain a hardware clock and enhanced firmware permitting automatic submission of packets to the network and providing additional statistics on channel acquisition delay and collisions per transmission.

NBSNET components were used in the design of experiments to analyze network performance. These components include a 1 Mbit coaxial cable, a transceiver, 30 meters of cable connecting the transceiver to a network board and one to eight microprocessor based interfaces called *user boards*. Figure 1 shows these components. While the NBSNET architecture allows up to 8 user boards per network board, only 1 was used. For all experiments, up to 6 traffic generators were active at any one time and they were programmed to send datagram packets.

The performance data were collected from two classes of experiments: (1) traffic generators with exponentially distributed packet interarrival times and (2) continuously queued traffic generators. The experiments using exponentially distributed packet interarrival times provided data for performance measurements with moderate, heavy, and overload conditions. Six traffic generators were active for these experiments, and each contained identical packet length and interarrival time distributions. The packet lengths used were constant packet lengths of 50, 100, 150, and 200 bytes. Each traffic generator transmitted these packet lengths according to exponentially distributed interarrival times with means of 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 18, and 22 milliseconds. These conditions permitted measuring the performance of NBSNET with offered loads of 10% to 275% of channel capacity.

Using the same packet lengths described above, one to six continuously queued traffic generators provide maximum individual offered loads with total offered loads of 59% to 511% of channel capacity. By changing the number of active nodes, these experiments provide data for analysis of overload conditions.

2. Channel Utilization

Shoch and Hupp [2] demonstrated that the channel utilization for the 2.94 Mbit Xerox Ethernet, a similar CSMA/CD system, equalled the offered load for offered loads less

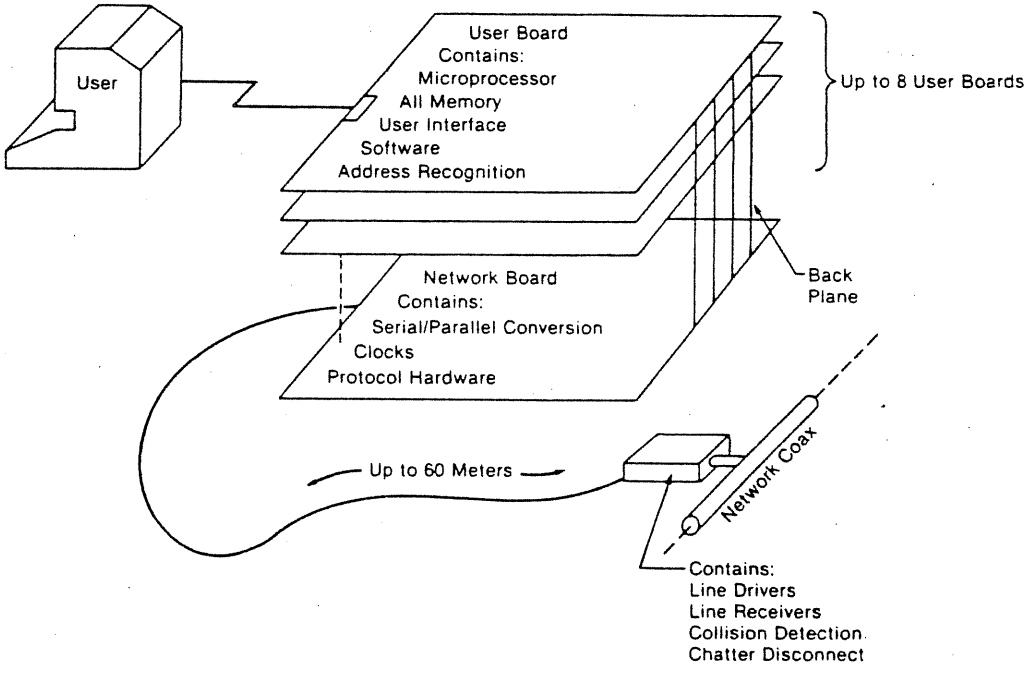


Figure 1. The Hardware.

than 90% of channel capacity with maximum size packets and ten active nodes. The utilization approached 98% for higher offered loads.

The data in Figure 2 represent the utilizations measured with six active nodes and constant packet lengths for NBSNET. The offered load was varied by changing the arrival rate. These results are similar to those reported by Shoch and Hupp. These data show that short packets produced channel utilizations within 1% of the offered load until approximately a 60% offered load. The utilization was within 3% of the offered load until 75% offered load. Long packets produced utilizations within 1% of the offered load until about 70% capacity. Higher offered loads produced utilizations still less than the offered load. The maximum utilization approached 90% to 95% capacity, depending on packet length.

For simple estimation purposes, the data fit an exponential function of the form

$$\% \text{ Utilization} = A \times (1 - \exp(B \times \% \text{ Offered Load})).$$

This predicted a maximum utilization (A) of 93% with 50 byte packets increasing to 95% with 200 byte packets given an infinite offered load and six active nodes. The period of the function (B) is .00084 for 50 byte packets and increases to .0013 for 200 byte packets. This indicates that short packets diverge from the ideal linear curve faster than long packets.

Shoch and Hupp also demonstrated the behavior of the 2.94 Mbit Ethernet with continuously queued sources. Their data showed that decreasing the packet size or increasing the number of active nodes produced lower maximum utilizations. As the packet transmission time approached the contention interval, the utilization approached 1/e, the maximum utilization of a slotted Aloha network [7].

A single, continuously queued NBSNET traffic generator cannot offer 100% of the channel capacity of the network. A 300 microsecond latency in attempting to transmit the next packet determines the maximum offered load possible from an individual node. A calibration point for the offered load from an individual, continuously queued generator at each packet length is the measured utilization for a single generator because there is no chance for contention and the packet is transmitted immediately.

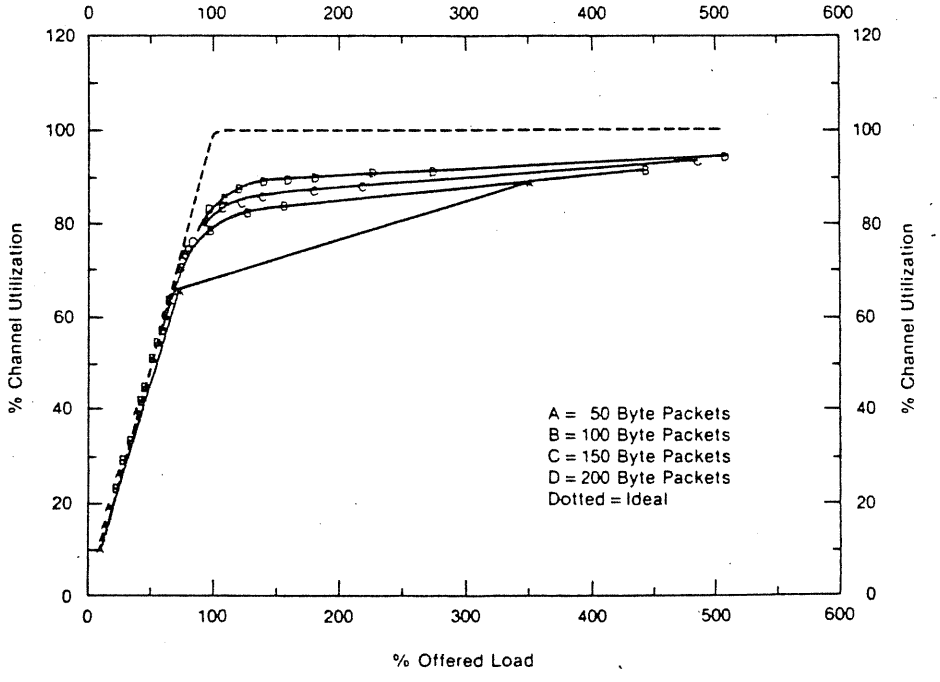


Figure 2. Channel Utilization with Six Active Nodes.

Figure 3 shows the measured utilizations with one to six continuously queued traffic generators for NBSNET. An increase in the number of active nodes with a constant packet length and continuously queued sources produces a decrease in the utilization. Also, a decrease in the packet length produces decreased utilization. The worst case measured was six nodes with 50 byte packets. It produces a utilization which had dropped to 89%.

3. Stability

An important criterion for any local area network is that it remain stable. This is, for any offered load, utilization is a function of offered load. Further, an impulse in the offered load produces a utilization that returns to its original equilibrium value [8]. If the utilization is a function of the offered load and has a positive slope, the utilization is guaranteed to be stable. If the slope is negative, a potential for instability exists, and the impulse response must be examined. To test stability on NBSNET, we examined the effect of changes in arrival rate, packet length, and number of active nodes on utilization.

3.1 Changing Arrival Rate. The data in Figure 2 show the effect of changing arrival rates and represent the utilizations measured with six active nodes and constant packet lengths. The offered load was varied by changing the arrival rate. The utilization is an increasing function of the arrival rate for a given packet length.

3.2 Changing Packet Length. The data in Figure 2 also show the effect of changing packet lengths. The utilization curves for larger packets are strictly greater than the curves for smaller packets. Therefore, an increase in the packet length produces an increase in the utilization.

Since an increase in the arrival rate or an increase in the packet length results in an increase in the channel utilization, changes in these components of the offered load will not cause the network to become unstable for the number of nodes observed.

3.3 Changing Number of Active Nodes. Figure 3 shows the effect on utilization of changing the number of active nodes. It shows that an increase in the number of

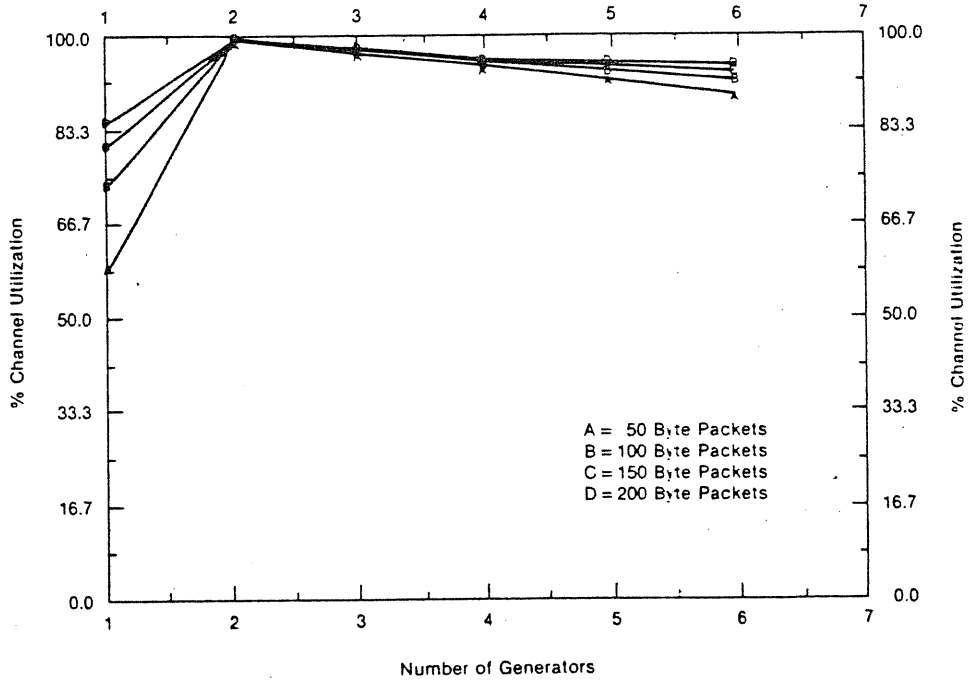


Figure 3. Channel Utilization with Continuously Queued Generators.

active nodes with a constant packet length and continuously queued sources produces a decrease in the utilization. This indicates a potential for instability with respect to the number of nodes.

We observed that for the cases examined, an increase in the number of nodes causes the utilization to reach a new equilibrium point at a reduced value. Removing the additional offered load caused the utilization to increase, returning to its previous value. This indicates stable operation.

The utilization lost is due to the increased overhead required to resolve collisions between more nodes. Given more nodes to collide, and a binary exponential backoff, the expected number of collisions needed to resolve the collision increases. As long as the overhead remains bounded, the network will reach new equilibrium point at a reduced utilization and remain stable.

4. Channel Acquisition Delay

Tobagi and Hunt predicted the mean channel acquisition delay for a CSMA/CD network as a function of throughput and backlog [3]. Their results show that a packet experiences infinite delay as the throughput approaches the channel capacity when the system has no backlog of packets waiting to be sent. If a backlog is present, the delay goes to infinity at lower throughput and is no longer a function of the throughput as it can take two values for a given throughput.

Similar measurements of channel acquisition delay were made on NBSNET. In these experiments, the channel utilization is equivalent to the throughput as defined by Tobagi and Hunt. The channel acquisition delay is measured from the time a packet arrives for service until the first bit is transmitted on the cable on the final attempt to transmit the packet. This time includes time for deferrals and collision attempts. This corresponds to a packet arriving with no backlog in the study by Tobagi and Hunt. NBSNET follows the predicted behavior, and a function is found to describe the delay.

Figure 4 shows the delay as a function of the measured utilization for six active nodes with Poisson arrival rates. For the range of utilizations observed, the delay is close to zero at low utilizations and is over 8 milliseconds at 90% utilization and increasing rapidly.

A hyperbolic function of the form

$$\text{Delay} = C/(\text{Utilization} - A) + B$$

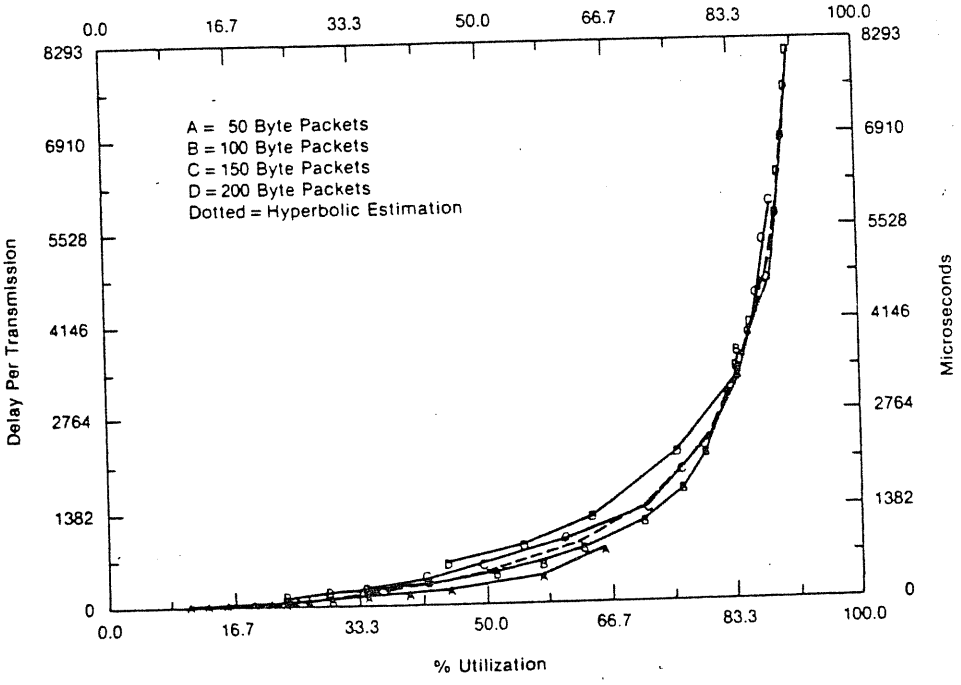


Figure 4. Channel Acquisition Delay.

where A, B, and C are constants with values of 98.3, −750, and −60,000, respectively, is useful for estimating the acquisition delay, over the entire range of experimental data. The error function was less than 5% of the predicted delay for all utilizations observed.

The utilization asymptote, "A," is 98.3. This implies that utilization levels approaching 98% are possible with six active nodes. However, the channel acquisition delay at such utilizations will be very large.

In all cases measured, the standard deviation of the delay was between 2.1 and 2.8 times the mean. The longest delay measured under any condition was 250 milliseconds. This delay involved a 200 byte packet with overload conditions.

One component of the acquisition delay is the time spent resolving collisions. This time is measured by counting the number of collisions in a transmission and summing the backoff times and latency for number of collisions experienced. For NBSNET, the latency in starting a backoff time is 150 microseconds and the mean backoff delay in microseconds for a given collision is

$$\text{backoff time} = 2 \hat{} n + (2 \hat{} (n-1) - 1) / 2$$

where n is the number of collisions for that packet.

Figure 5 illustrates the behavior of the mean number of collisions per packet transmitted for a given utilization and packet length with six active nodes. Most packets are transmitted without a collision for utilizations as high as 60%. The standard deviation for this measurement varied from 1.5 to 2.5 times the mean, with the lower standard deviations corresponding to high utilizations.

For estimation purposes, the mean collisions per transmission fit an exponential function of the form

$$\text{Collisions per Transmission} = A \times \exp(B \times (\%\ \text{Utilization})) + C.$$

The values for A, B, and *C* were found to be 0.170, 0.031, and −0.281, respectively, for six active nodes. This function tends to underestimate collisions at utilizations less than 20%. Here, there are few collisions and the number of collisions is an insignificant factor in the total delay. It predicts the expected number of collisions per transmission within 20% for those utilizations measured greater than 20%.

Statistics on the number of collisions that occur, given that at least one collision occurs,

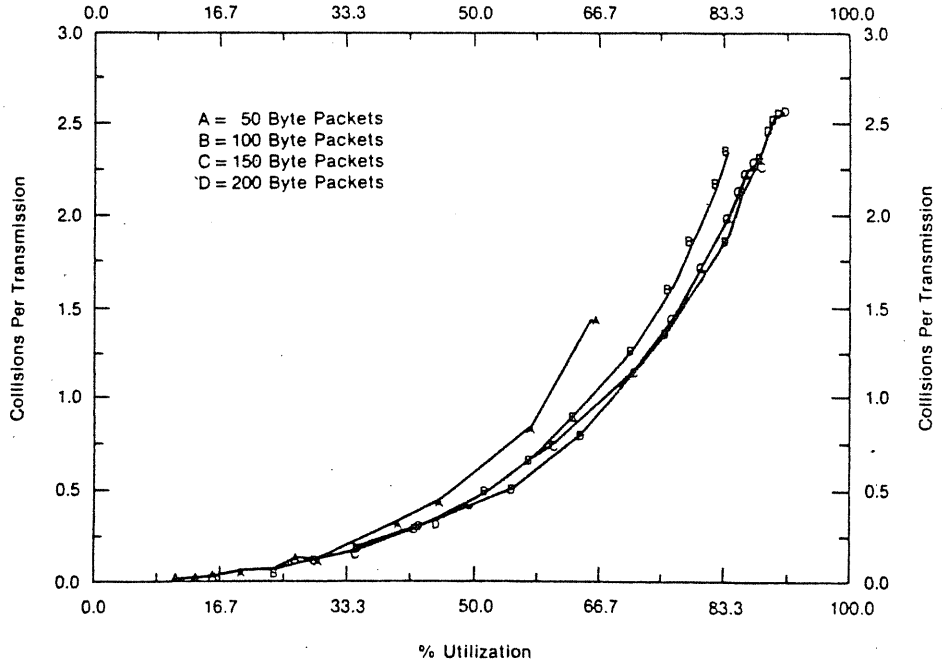


Figure 5. Mean Collisions per Transmission.

are useful to determine the effectiveness of the backoff algorithm and to indicate the delay expected for an individual collided packet. Figure 6 illustrates the behavior of this parameter with six active nodes and several packet lengths as a function of utilization. The mean number of collisions per collided transmission varies only from slightly over 2 to slightly over 3 and is almost linear. This is a small variation compared to the collisions per transmission. This implies that nearly the same number of attempts are required to resolve any collision, regardless of the utilization. This is because the number of slots available to the backoff algorithm and the number of nodes participating in a collision are the primary factors in determining the probability of resolving a collision.

Combining the latency per transmission attempt and the values for the backoff delays, the mean delay for all packets due to collisions is no more than 475 microseconds per transmission with six nodes active. At utilizations under 70% of capacity, the mean delay due to collisions is less than 100 microseconds. This time is less than 10% of the expected total delay. Given that a packet has collided, the mean delay is between 325 microseconds at low utilizations and 500 microseconds at high utilizations.

Individual packets may experience many more collisions than the mean. Collisions that require many transmission attempts may result in delays due to collisions on the order of 100 ms on an individual transmission. These are experienced less than .1% of the time with utilizations greater than 90%. Even under these conditions, many packets are transmitted on the first attempt. If a collision does occur, it is most likely to be resolved on the next attempt.

5. Fairness

NBSNET is designed to be a nonprioritized network. This means that if the network is fair, each node should have an equal chance of gaining access to the channel. Therefore, the portion of the total channel utilization received by each node should be proportional to its fraction of the total offered load to the network.

A useful tool to measure the fairness as a function of offered load is a normalized individual utilization (NIU). This is defined as the ratio of the utilization achieved by a single

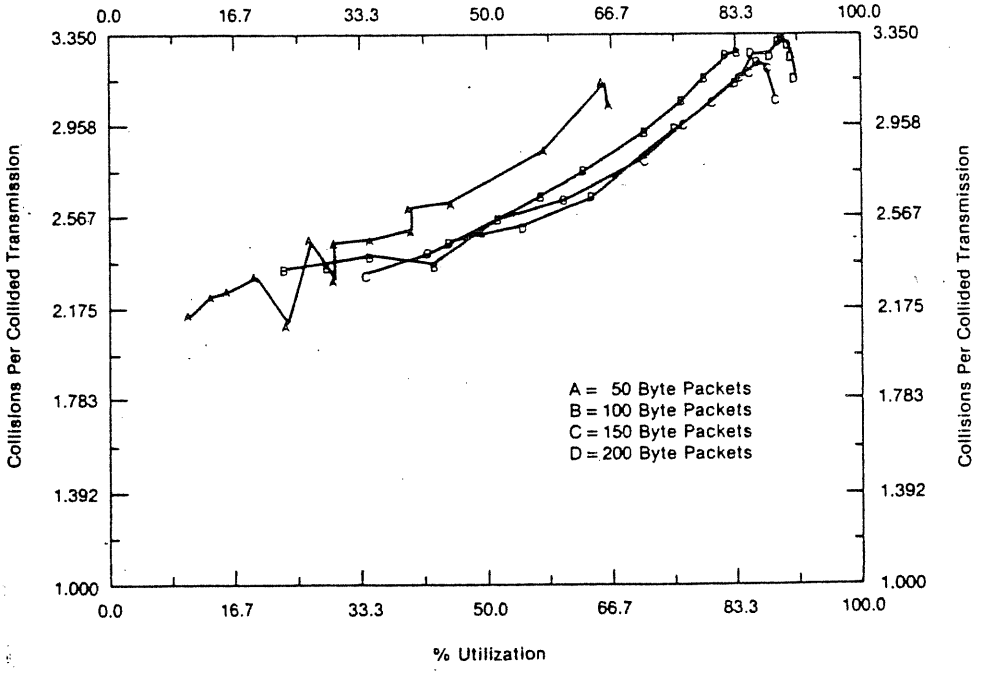


Figure 6. Collided Collisions.

node to the expected utilization for that node, assuming fair behavior of the network. A NIU of one indicates that the node received exactly the expected utilization. A value greater than one indicates favoritism for that node, and a value of less than one indicates that the node was discriminated against. The standard deviation of the normalized individual utilizations for all active nodes at a given offered load is used as an index of fairness. A low index of fairness indicates fair operation.

For these experiments, each node attempted to offer identical individual loads with identical packet lengths and arrival rates. Because of the symmetry of this setup, the utilization each node received should be exactly the total utilization divided by the number of active nodes. Figure 7 illustrates the NIUs in the high load case with six active nodes and packet lengths of 50, 100, 150, and 200 bytes. The corresponding graph of the index of fairness is shown in Figure 8.

The index of fairness indicates very fair behavior, according to the definition of fairness for offered loads less than 90%. For these loads, the index of fairness is less than 0.05. Higher offered loads cause the fairness of the network to deteriorate rapidly.

Repeated experiments using the same offered load show that the same nodes are always favored for a given offered load as the network becomes less fair. Also, the fairness degrades as the collisions per transmission increase. Since NBSNET uses a fixed binary exponential backoff table for determining backoff time, certain nodes may be favored, given an expected number of collisions per transmission. A random backoff scheme may improve fairness under these conditions for a constant offered load.

6. Conclusions

The foregoing results demonstrate the following features of NBSNET:

1. The channel utilization approaches theoretical limits for all offered loads measured and is predictable as an exponential function of the offered load.
2. The mean delay introduced by the network is predictable as a hyperbolic function of the observed channel utilization.
3. The network is stable for changes in packet length and arrival rate for the number of nodes observed. However, it shows a potential for instability with respect to changes in the number of active nodes.
4. The utilization for individual nodes is uniformly distributed and therefore follows

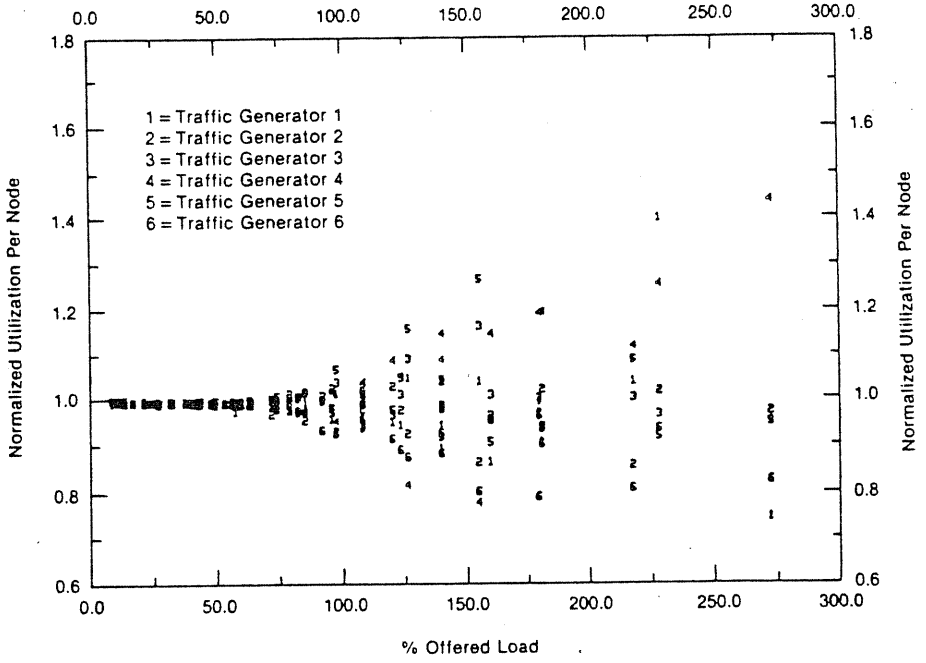


Figure 7. Individual Normalized Utilizations.

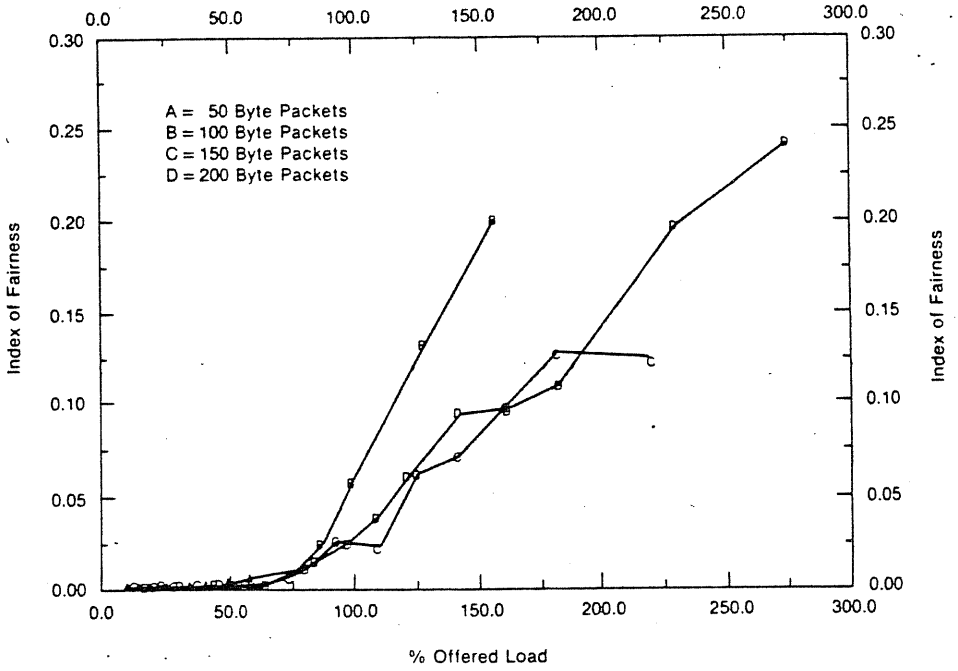


Figure 8. Index of Fairness for Six Active Nodes.

the definition of fairness. The fixed backoff table for resolving collisions does not affect fair channel allocation.

In conclusion, NBSNET fulfills the expectations predicted by theoretical studies. Our approach of describing NBSNET performance in terms of utilization, stability, delay, and fairness has proved useful. We expect that performance characterizations for other CSMA/CD networks can use the same approach.

Acknowledgments

The author appreciates the technical assistance given by P. Amer, R. Carpenter, and D. Stokesberry. Special thanks go to M. L. Fahey for her help in data collection.

References

1. Carpenter, R. J. and Sokol, J. Jr. "Serving Users with a Local Area Network," *Computer Networks*, October 1980.

2. Shoch, J. F. and Hupp, J. A. "Performance of an Ethernet Local Network: A Preliminary Report," *Proceedings of the Local Area Communications Network Symposium*, May 1979, pp. 113-123.

3. Tobagi, F. A. and Hunt, V. B. "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," *Proceedings of the Local Area Communications Network Symposium*, May 1979, pp. 217-245.

4. Stokesberry, D. and Rosenthal, R. "The Design and Egineering of a Performance Measurement Center for a Local Area Network," *Proceedings of the Computer Networking Symposium*, December 1980, pp. 110-115.

5. Amer, P. "A Measurement Center for the NBS Local Area Computer Network," *IEEE Transactions on Computers*, Vol. C-31, No. 8, August 1982.

6. Filliben, J. J. "DATAPLOT—An Interactive High-Level Language for Graphics, Non-Linear Fitting, Data Analysis, and Mathematics," *Computer Graphics*, Vol. 15, No. 3, August 1981, pp. 199-213.

7. Abramson, N. "The Throughput of Packet Broadcasting Channels," *IEEE Transactions on Communications*, Vol. COM-25, No. 1, January 1977, pp. 117-128.

8. Kleinrock, L. and Lam, S. S. "Packet Switching in a Multiaccess Broadcast Channel: Performance Evaluation," *IEEE Transactions on Communications*, April 1975, pp. 410-423.

About the Author

Robert E. Toense is an electronics engineer at the National Bureau of Standards, Institute for Computer Sciences and Technology. There he conducts research in local area networking technology for which he has received the U.S. Departyment of Commerce Certificate of Recognition. His work has involved participation in the design and implementation of a network measurement facility including monitoring and traffic generation facilities. Most recently his work has involved performance studies of local area networks. During the years 1973 to 1980 he worked for the Naval Surface Weapons Center, White Oak, Maryland. There he

worked on the design and implementation of acoustic detection and control systems and automatic test equipment. Mr. Toense received his B.S. in electrical engineering from the University of Maryland in 1976. He is a member of the IEEE, Eta Kappa Nu, and Tau Beta Pi.

APPENDIX D: Gonsalves paper

Performance Characteristics of 2 Ethernets: an Experimental Study

Timothy A Gonsalves

Computer Systems Laboratory
Department of Electrical Engineering
Stanford University, Stanford, CA-94305

Abstract

Local computer networks are increasing in popularity for the interconnection of computers for a variety of applications. One such network that has been implemented on a large scale is the Ethernet. This paper describes an experimental performance evaluation of a 3 and a 10 Mb/s Ethernet. The effects of varying packet length and transmission speed on throughput, mean delay and delay distribution are quantified. The protocols are seen to be fair and stable. These measurements span the range from the region of high performance of the CSMA/CD protocol to the upper limits of its utility where performance is degraded. The measurements are compared to the predictions of existing analytical models. The correlation is found to range from good to poor, with more sophisticated models yielding better results than a simple one.

1. Introduction

In the past few years, local computer networks for the interconnection of computers and shared resources within a small area such as a building or a campus have rapidly increased in popularity. These networks support applications such as file transfers and electronic mail between autonomous computers, the shared use of large computers and expensive peripherals, and distributed computation. Typically, these networks have bandwidths in the range 0.1 - 10 Mb/s and span distances of 0.1 - 10 km. Local networks are also being interconnected to form multi-hop *internets* which allow communication over larger areas.

Many architectures have been proposed for local networks, and several have been implemented. The Ethernet [9] was one of the earliest to be implemented. Use of 3 Mb/s *experimental Ethernets* at several interconnected locations by a large community of users over several years and an experimental study of the performance of one under varying conditions have proved its merit for a variety of data transfer applications [11]. This has led to the introduction of a 10 Mb/s Ethernet as a commercial product [5].

Support and experimental facilities for this work were provided by the Xerox Palo Alto Research Centers. Additional support was provided by an IBM Graduate Fellowship, and by the Defense Advanced Research Projects Agency under Contract MDA 903-84-K-0249, monitored by the Office of Naval Research.

This paper describes the results of experimental measurements of the performance of a 3 Mb/s Experimental Ethernet and a 10 Mb/s Ethernet[1]. We present throughput, mean delay and delay distributions for the networks under artificially generated traffic conditions, making comparisons to the predictions of existing analytical models [8, 9, 15]. These results show the capabilities and limitations of the networks as well as the effects of increasing the channel speed. These experiments significantly increase the available base of data for the calibration and validation of analytical and simulation models of CSMA/CD networks.

In the next section we describe the principles of the Ethernet architecture and relevant details of the implementations used in this work, and we review prior work in this area. Section 3 describes the details of the experimental procedures. Results are presented and discussed in section 4. Section 5 is a summary of the paper.

2. Background and Previous Work

2.1. The Ethernet Architecture

The Ethernet [9] is an example of a packet-switched local network using a single shared bus or channel for communication among several hosts. The nature of the channel is such that only one packet can be transmitted at a time and every packet traverses the entire channel. This requires that all the hosts observe some protocol to gain control of the channel in an orderly and fair manner.

The Ethernet uses an access protocol called *CSMA/CD*. *CSMA*, or *carrier sense multiple-access*, is one of the distributed schemes proposed to efficiently utilize a broadcast channel [7]. In the CSMA scheme, a host desiring to transmit a packet waits until the channel is idle, i.e., there is no carrier, and then starts transmission. If no other hosts decide to transmit during the time taken for the first bit to propagate to the ends of the channel, the packet is successfully transmitted (assuming that there are no errors due to noise). However, due to the finite propagation delay, some other hosts may also sense the channel to be idle and decide to transmit. Thus, several packets may collide. To ensure reliable transmission, acknowledgements must be generated by higher level protocols. Unacknowledged packets must be retransmitted after some time.

In order to improve performance, the Ethernet protocol incorporates *collision detection* into the basic CSMA scheme, hence the abbreviation CSMA/CD. To detect collisions, a transmitting host monitors the channel while transmitting and aborts transmission if there is a collision. It then *jams* the channel for a short period to ensure that all other transmitting hosts abort transmission also. Each host then schedules its packet for retransmission after a random interval chosen according to some *retransmission or back-off algorithm*. The randomness is essential to avoid repeated collisions between the same set of hosts. The retransmission algorithm can

[1] Hereafter we use the term *Ethernet* to refer to both networks

affect such characteristics as the stability of the network under high loads and the fairness to contending hosts.

2.2. A 3 Mb/s Ethernet Implementation

We summarize the relevant details of the 3 Mb/s Ethernet local network used in our experiments. This network has been described in detail earlier [4, 9]. The network used in our experiments has a channel about 550 metres long with baseband transmission at 2.94 Mb/s. The propagation delay in the interface circuitry is estimated to be about 0.25 μs [3]. Thus, the end-to-end propagation delay, τ_p is thus about 3 μs. The retransmission algorithm implemented in the Ethernet hosts (for the most part, Alto minicomputers [13]) is an approximation to the binary exponential back-off algorithm. In the binary exponential back-off algorithm, the mean retransmission interval is doubled with each successive collision of the same packet. Thus, there can be an arbitrarily large delay before a packet is transmitted, even if the network is not heavily loaded. To avoid this problem, the Ethernet hosts use a truncated binary exponential back-off algorithm. Each host maintains an estimate of the number of hosts attempting to gain control of the network in its load register. This is initialised to zero when a packet is first scheduled for transmission. On each successive collision the estimated load is doubled by shifting the load register 1 bit left and setting the low-order bit to 1. This estimated load is used to determine the retransmission interval as follows. A random number, χ, is generated by ANDing the contents of the load register with the low-order 8 bits of the processor clock. Thus, χ is approximately uniformly distributed between 0 and $2n$-1, where n is the number of successive collisions, and has a maximum value of 255, which is the maximum number of hosts on the network. The retransmission interval is then chosen to be χ time units. The time unit should be no less than the round-trip end-to-end propagation delay. It is chosen to be 38.08 μs for reasons of convenience. After 16 successive collisions, the attempt to transmit the packet is abandoned and a status of load overflow is returned for the benefit of higher level software.

Thus, the truncated back-off algorithm differs from the binary exponential back-off algorithm in two respects. Firstly, the retransmission interval is limited to 9.7 ms (255 X 38.08 μs). Secondly, the host makes at most 16 attempts to transmit a packet.

2.3. A 10 Mb/s Ethernet Implementation

The 10-Mb/s Ethernet used is similar to the network described in the previous section with a few exceptions. The channel consists of three 500 metre segments connected in series by two repeaters. The end-to-end propagation delay, including delay in the electronics, is estimated to be about 30 μs (see pg. 52 in [5]). The truncated binary exponential back-off algorithm uses a time unit of 51.2 μs. The load estimate is doubled after each of the first 10 successive collisions. Thus the random number, χ, has a maximum value of 1023, yielding a maximum retransmission interval of 53.4 ms. A maximum of 16 transmission attempts are made for each packet.

2.4. Previous Work

The literature contains many theoretical and simulation studies of CSMA/CD networks with various traffic patterns [1, 8, 10, 14, 15]. However, there are few reported performance measurements of such networks. The first measurements were reported by Shoch [11] on the throughput of a 3 Mb/s Ethernet under normal and artificially generated data traffic. A concise treatment of part of that study appeared subsequently [12]. Gonsalves [6] reported the measured delay-throughput characteristics of the same network under artificially generated voice traffic.[2] Amer [2] describes measurement tools implemented on a 1 Mb/s CSMA/CD network. Toense [16] reported measurements on that network with up to 6 traffic generating stations.

Our experiments differ from those reported by Shoch and Hupp [12] in two respects. Firstly, we measure packet delays in addition to throughput and the other quantities measured in the earlier experiments. Delay is an important performance metric particularly for interactive applications. Secondly, we present results for a 10 Mb/s network as well as for the 3 Mb/s network reported upon in the earlier study.

3. Experimental Environment

In this section we describe the techniques used to measure the performance of the Ethernet under varying load conditions. First, we describe the experimental setup and procedures and then we characterize the traffic patterns and performance measures used.

3.1. Experimental Setup and Procedures

To set up and run an experiment we use a special control program, running on a host on the network, to find idle hosts on the network and to load a test program into each [12]. The controller is then used to set parameters describing the traffic pattern to be generated by the test programs. Next, the test programs are started simultaneously. They generate traffic and record statistics for the duration of the run. At the end of the run, the statistics are collected from the participating hosts by the control program.[3] The duration of each run is typically 60 seconds. Our tests show that there is no significant variation in statistics for run times from 10 to 600 seconds.

Measurements on the 3 Mb/s Ethernet [11] and our informal observations of traffic on the 10 Mb/s Ethernet indicate that at night the normal load rarely exceeds a small fraction of 1% of the network capacity. Thus, it is possible to conduct controlled experiments with specific traffic patterns and loads on the networks during the late night hours.

3.2. Traffic Patterns and Performance Metrics

Each of the N hosts is assumed to have one packet buffer. After completion of transmission of a packet, the host waits for a random period, T_{idle}, before the next packet is queued for transmission in the buffer. The mean idle period and packet length, P, and their distributions are set for each host for the duration of each run. Note that this corresponds to a central server queueing model, with the network being represented by the central server. The *offered load* of host i, G_i, is defined to be the fraction of C, the network bandwidth, that the host uses if it is the only active transmitter on the network. Thus:

$$G_i = T_p / (T_p + T_{idle}),$$

where $T_p = P/C$ is the average transmission time of a packet. The *total offered load*, G is given by $\Sigma_{(i=1,N)} G_i$. In our experiments we use a homogeneous population of hosts. Packet lengths are fixed and packet generation times are uniformly distributed random variables.

We ignore packets lost due to collisions which the transmitter cannot detect ([11], p. 72) and due to noise since these errors have been shown to be very infrequent [12]. That is, we assume that if a packet is successfully transmitted it is also successfully received. Thus, all the participating hosts in an experiment are transmitters of packets. In computing throughput, η, we assume that the entire packet, except for a 6-byte header and checksum[4], is useful data. Thus, our results represent upper bounds on performance since many actual applications will include additional protocol information in each packet. Packet delay, D, is defined to be the time from when the packet is generated, that is, the time at which it is queued for network

[2] The Ethernet used in these two studies is the 3 Mb/s Ethernet on which this work was performed.

[3] On the 10 Mb/s Ethernet, owing to differences in software, loading of the test program was done manually. The rest of the control was automated as described.

[4] We assume 4 bytes of overhead in the case of the 10 Mb/s Ethernet

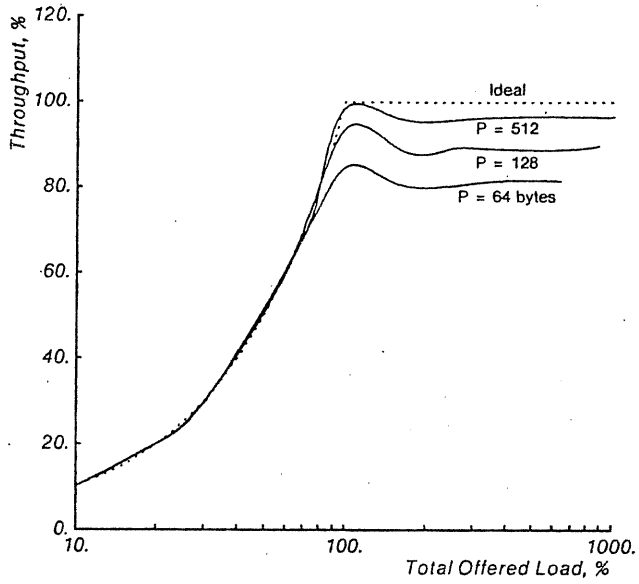


Fig. 4.1 3 Mb/s Ethernet: Throughput vs. Total Offered Load

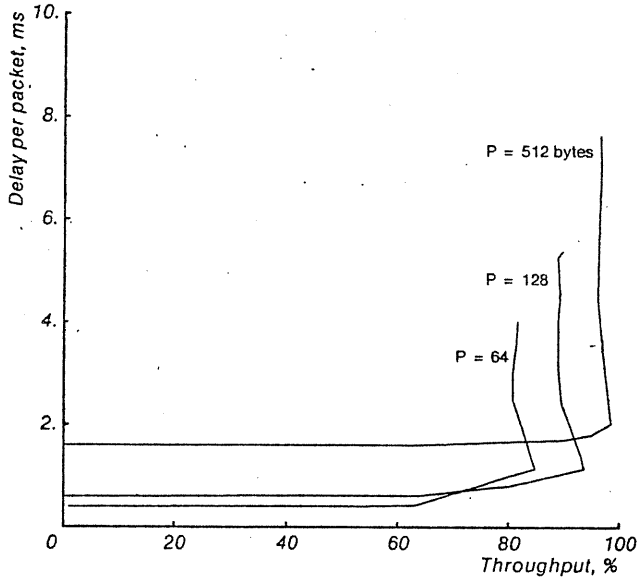


Fig. 4.2 3 Mb/s Ethernet: Delay vs. Throughput

access, to the time at which transmission is successfully completed. Packets whose transmission is abandoned due to too many collisions are not included in the mean delay computations.

4. Results

In this section we present the results of our experiments. First, we discuss the performance of the 3- and 10-Mb/s Ethernets separately. Next we make some comparisons between the two set of experiments. Finally, we compare our measurements to the predictions of some existing analytical models. In all experiments, fixed length packets were used. The inter-arrival times of packets at each host were uniformly distributed random variables.

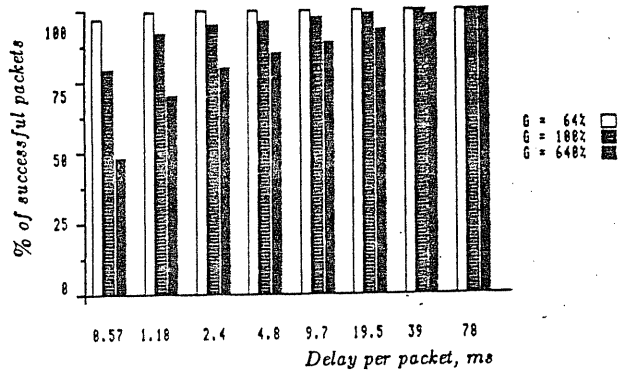


Fig. 4.3 3 Mb/s Ethernet: Delay Distributions, $P = 64$ bytes

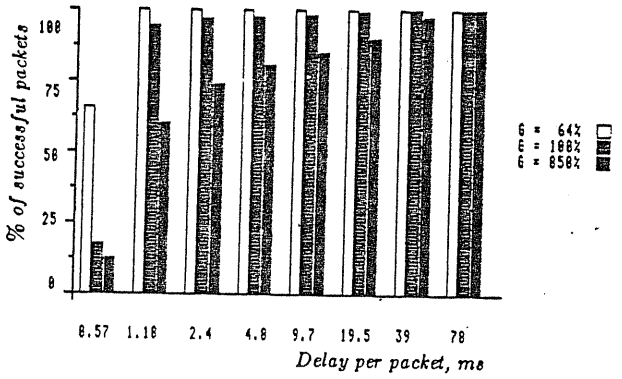


Fig. 4.4 3 Mb/s Ethernet: Delay Distributions, $P = 128$ bytes

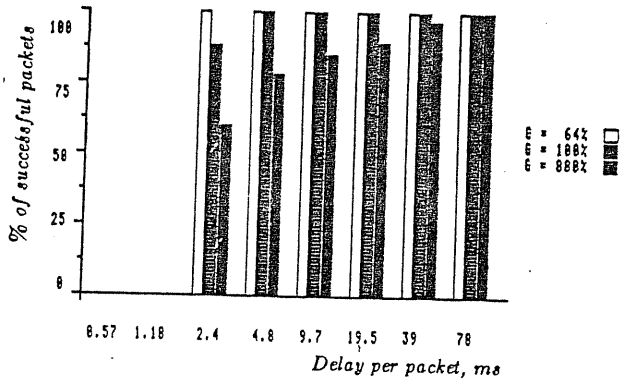


Fig. 4.5 3 Mb/s Ethernet: Delay Distributions, $P = 512$ bytes

.1. 3 Mb/s Experimental Ethernet

Figs. 4.1 - 4.7 display various aspects of the performance of the 3 Mb/s Ethernet.

Utilization: Fig. 4.1 shows the variation of total throughput, η, with total offered load, G, for $P = 64$, 128 and 512 bytes. For G less than 80-90% virtually no collisions occur and η is equal to G. Thereafter, packets begin to experience collisions and η levels off to some value directly related to P after reaching a peak, η_{max}. For short packets, $P = 64$, this maximum is about 80%, for longer packets, $P = 512$, it is above 95%. The network remains stable even under conditions of heavy overload owing to the load-regulation of the back-off algorithm. (These curves are similar to the ones obtained by Shoch and Hupp [12]).

P bytes	G %	Successful Packets / Total Packets %
64	64	100.0
	100	99.8
	640	96.1
128	64	100.0
	100	99.7
	850	88.3
512	64	100.0
	100	99.9
	880	58.2

Table 4.1 3 Mb/s Ethernet: Successfully transmitted packets as a fraction of total packets

Delay: Fig. 4.2 shows the delay-throughput performance for the same set of packet lengths. In each case, for η less than η_{max}, the delay is approximately equal to the packet transmission time, i.e., there is almost no queueing delay for access to the network. As the throughput approaches the maximum, the delay rises rapidly to several times the packet transmission time owing to collisions and the associated back-offs.

Figs. 4.3 - 4.5 show the histograms of the cumulative delay distributions for low, medium and high offered loads for $P = 64$, 128, 512 bytes. The delay bins are logarithmic. The labels on the X-axis indicate the upper limit of each bin. The left-most bin includes packets with delay $\leq$ 0.57 ms, the next bin, packets with delay $\leq$ 1.18 ms, and so on. The ordinate is the number of packets expressed as a percentage of all successfully transmitted packets. Table 4.1 gives the fraction of the total packets generated that were successfully transmitted.

We see that for $G = 64\%$, the delay of all packets is approximately the minimum, i.e., there is little queueing for network access. Even with $G = 100\%$ most packets suffer delays of less than 5 ms. Under heavy load conditions, however, only about 75% of the packets have delays of less than 5 ms, with the remainder suffering delays of up to 80 ms.

Fairness: To investigate the fairness of the protocol to contending hosts, we examine variations in performance metrics measured by individual hosts with increase in G. In Fig. 4.6 we plot the normalized mean of the individual throughputs vs. G for $P = 64$ and 512 bytes. The vertical bars indicate the normalized standard deviation, i.e., the coefficient of variation. Also shown are the maximum and minimum individual throughputs. For low G, there is little variation in individual throughput. Under overload, the variation increases but remains less than $\pm 10\%$.

Fig. 4.7 contains similar plots for the mean delay per packet measured by each host. The variations with G are seen to be similar to those in Fig. 4.6. Thus the protocol is seen to be fair to all contenders. (This has been noted in other experiments([6, 12])).

The metrics show slightly higher variation for $P = 512$ than for $P = 64$ bytes. During the successful transmission of a 512 byte packet a larger number of hosts are likely to queue for access to the network. Thus, at the end of the transmission all these hosts will attempt to transmit and will collide. The time for resolution of the collision is dependent on the number of colliding hosts and hence may be expected to be longer for $P = 512$ than for $P = 64$.

Discussion: The 3 Mb/s Ethernet is found to achieve high throughput for the range of packet sizes considered. The protocol is fair and stable under overload.

For $G < 100\%$, i.e. most practical situations, the delay is within a small multiple of the packet transmission time, T_p. Under such conditions, the network could support real-time traffic with delay constraints satisfactorily. However, at heavy load, the stability of the protocol is achieved at the expense of large delays, two orders of magnitude larger than T_p for a fraction of the packets. The majority of the packets still suffer relatively minimal delays.

Our results do not contradict the analytical predictions [15] that CSMA/CD becomes unstable at sufficiently high offered loads. The analysis predicts that instability sets in when the probability of a packet transmission attempt in $2 \times \tau_p$ reaches approximately unity. Given the maximum retransmission back-off of 9.7 ms and τ_p of 3 μs, this occurs when there are more than 1000 hosts continuously attempting to transmit, much larger than the number of hosts in our experiments. For a longer network, this number would be proportionately lower.

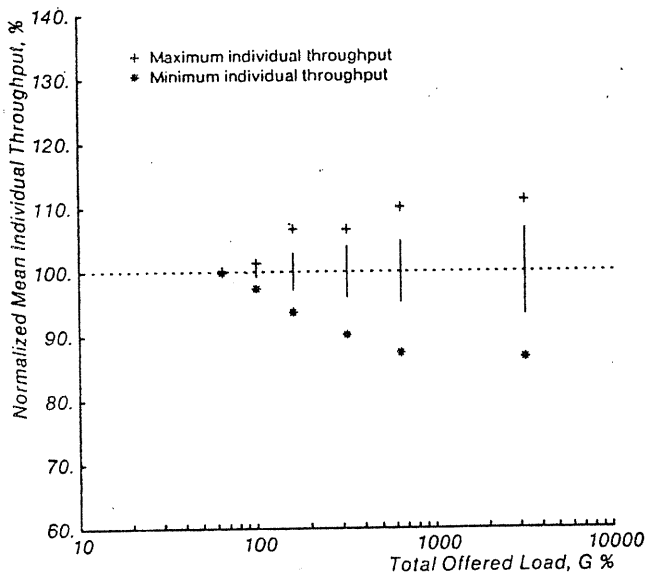


Fig. 4.6a 3 Mb/s Ethernet: Variation in η per Host
$P = 64$ bytes

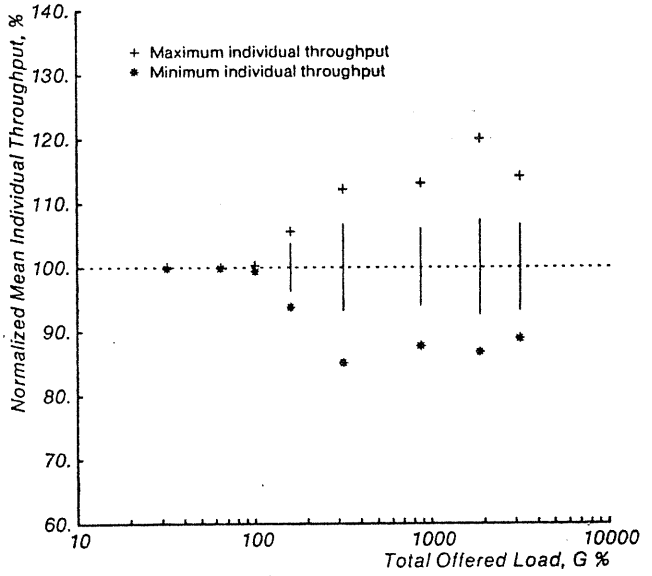


Fig. 4.6b 3 Mb/s Ethernet: Variation in η per Host
$P = 512$ bytes

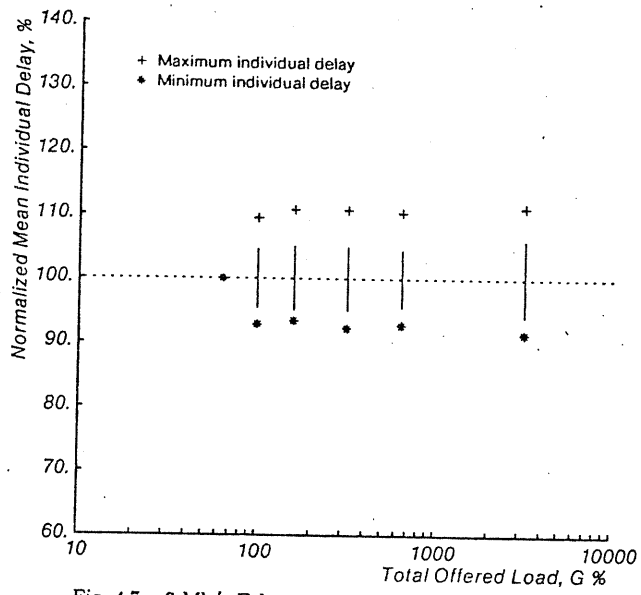


Fig. 4.7a 3 Mb/s Ethernet: Variation in Delay per Host
$P = 64$ bytes

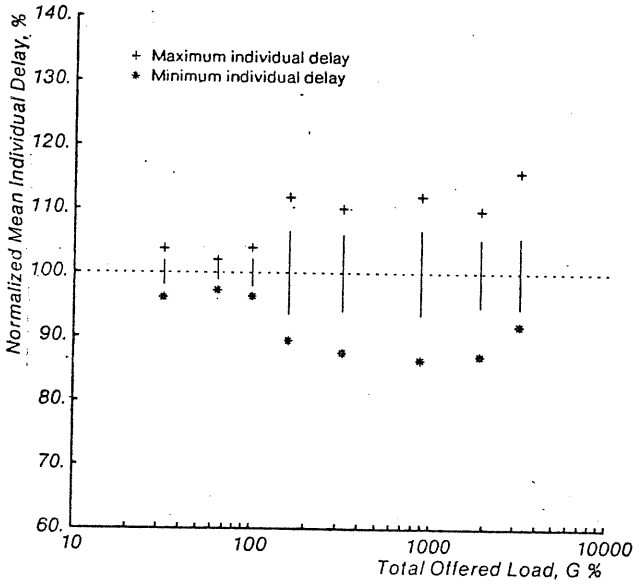


Fig. 4.7b 3 Mb/s Ethernet: Variation in Delay per Host
$P = 512$ bytes

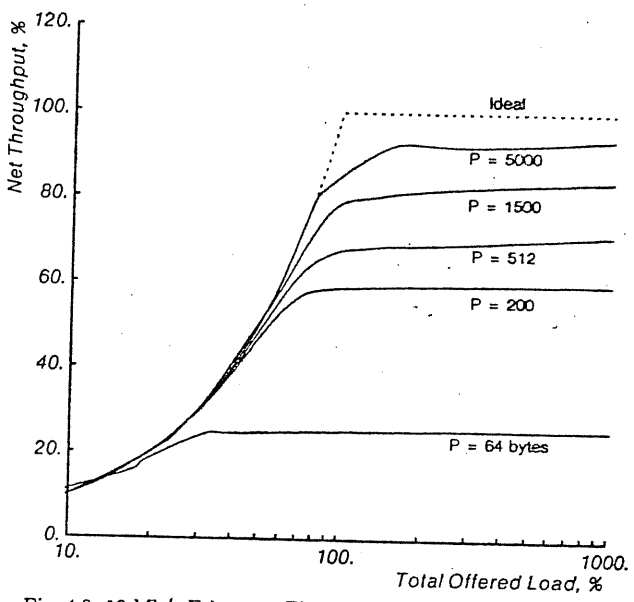


Fig. 4.8 10 Mb/s Ethernet: Throughput vs. Total Offered Load

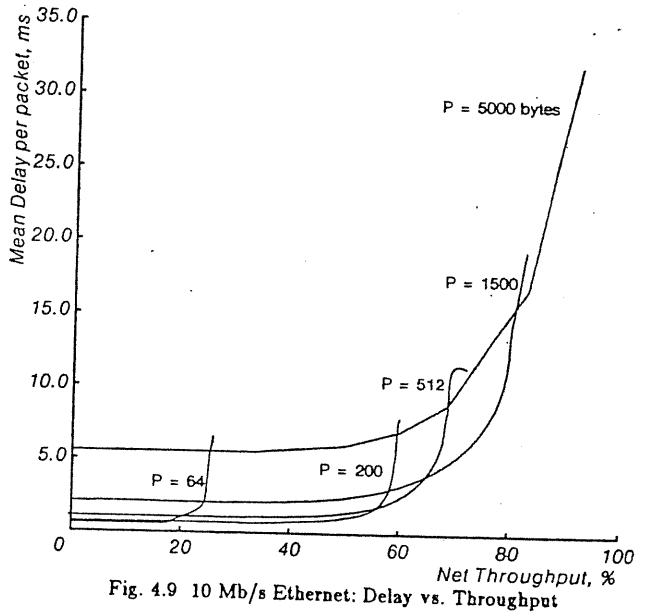


Fig. 4.9 10 Mb/s Ethernet: Delay vs. Throughput

4.2. 10 Mb/s Ethernet

Figs. 4.8 - 4.14 display various aspects of the performance of the 10 Mb/s Ethernet. The results shown were obtained using 30 - 38 transmitting hosts[5]. In all cases, fixed length packets were used and the inter-packet times were uniformly distributed random variables.

Utilization: Fig. 4.8 shows the throughput as a function of total offered load, G, for P ranging from 64 to 5000 bytes. The shape of the curves is similar to the corresponding curves for the 3 Mb/s Ethernet. However, maximum throughput varies from 25% for $P =$ 64 bytes (the minimum allowed by the Ethernet specifications [5]), to 80% for $P = 1500$ bytes (the maximum allowed), to 94% for very long packets of 5000 bytes. We note that for each curve, for G below the knee point, the throughput is approximately equal to G. Even under conditions of heavy overload, the network remains stable.

Delay: Fig. 4.9 shows the delay-throughput performance for the same set of packet lengths. Again, the curves have similar shapes to the curves for the 3 Mb/s network. For G below the knee points, the delay is minimal, whilst above the knee points, it rises sharply. The knees in this case are less pronounced, especially for larger P.

Figs. 4.10 - 4.12 show the histograms of the cumulative delay distributions for low, medium and high offered loads for $P = 64$, 512 and 1500 bytes. Table 4.2 gives the fraction of the total packets generated that were successfully transmitted. For low loads, delay is minimal for $P = 512$ and 1500 bytes. For $P = 64$, though, even at G = 19% delays range up to $10 \times T_p$. At high loads, for all packet lengths, the majority of packets suffer moderately increased delays,

[5] The number of hosts for a given set of experiments, i.e., for a single packet length, was constant

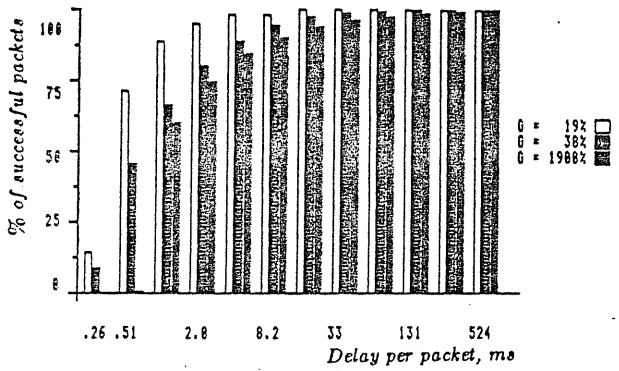


Fig. 4.10 10 Mb/s Ethernet: Delay Distributions, $P = 64$ bytes

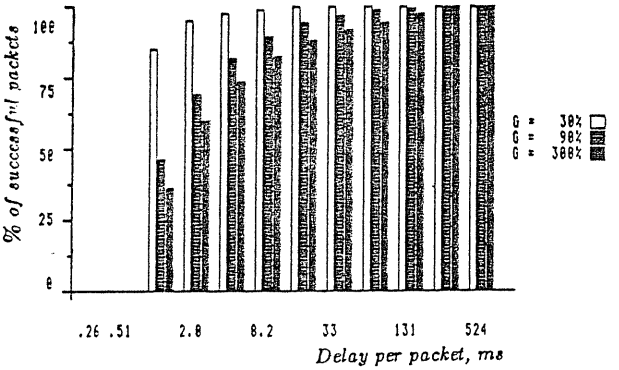


Fig. 4.11 10 Mb/s Ethernet: Delay Distributions, $P = 512$ bytes

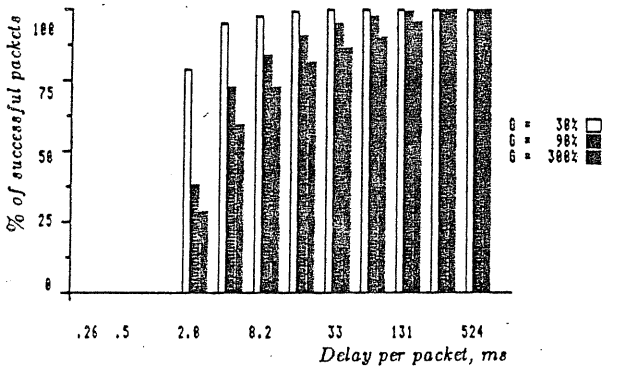


Fig. 4.12 10 Mb/s Ethernet: Delay Distributions, $P = 1500$ bytes

while a fraction suffer very high delays, up to 0.5s. for $P = 512$ and 1500, about 75% or all packets suffer delays $\leq 10 \times T_p$. For $P = 64$, 75% of packets suffer delays $\leq 15 \times T_p$.

Fairness: We examine variations in performance metrics measured by individual hosts with increase in G. In Fig. 4.13 and 4.14 we plot the normalized means of the individual throughputs and delays respectively vs. G for $P = 64$ and 512 bytes. The vertical bars indicate the normalized standard deviation, i.e., the coefficient of variation. Also shown are the maximum and minimum individual throughputs.

The metrics show higher variation than in the 3 Mb/s case, in the range $\pm 35\%$. This may be attributed to the larger retransmission

P bytes	G %	Successful Packets / Total Packets %
64	19	100.0
	38	100.0
	1900	99.9
512	30	100.0
	90	99.9
	300	98.7
1500	30	100.0
	90	99.7
	300	93.3

Table 4.2 10 Mb/s Ethernet: Successfully transmitted packets as a fraction of total packets

periods in the 10 Mb/s back-off algorithm (ref. Sections 2.2 and 2.3). Also, the dependence on G is less marked. Contrary to the 3 Mb/s case, the variation is slightly lower here for the larger packets size.

Discussion: With short packets, e.g. 64 bytes, T_p becomes comparable to τ_p and hence the maximum throughput is low. With long packets, we see the high throughputs achieved with the 3 Mb/s net. Thus, packet lengths of the order of 64 bytes on a 10 Mb/s net approach the limit of utility of the Ethernet protocol. This does not imply that such short packets should not be used. A study of traffic on a typical local computer network shows that approximately 20% of the total traffic is composed of short packets, whilst the remainder of 80% consists of long packets [12]. The 10 Mb/s Ethernet studied could support a high throughput with such a traffic mix although it cannot do so with only short packets.

Packet delay is minimal for low to moderate loads. However, for short packets the delay increases even at fairly moderate G. At higher G, a fraction of packets suffer delays ranging from the minimum up to 0.5 s.

For the loads used the network is fair and stable. Instability predicted by analytic studies will occur with about 1500 hosts continuously attempting to transmit.

4.3. Comparison of the 3 and 10 Mb/s Ethernets

In this section we examine the effect of the difference in bandwidth between the two Ethernets on performance. The differences in some important parameters, such as network length, in the two cases should be borne in mind.

Utilization: Fig. 4.15 shows the throughput, η, as a function of total offered load, G, for several packet lengths for the 2 networks. For $P = 64$ bytes, the throughputs of the two networks are almost equal. For longer packets, the 10 Mb/s network exhibits substantially higher throughput. The throughput increases less than linearly with increase in bandwidth. This is shown in Table 4.3 in which the ratio of the absolute throughput at 10 Mb/s to that at 3 Mb/s is given for several values of P.

P, bytes	Ratio of throughputs η_{10}/η_3
64	1.05
512	2.45
1500	2.90

Table 4.3 Increase in η with increase in C from 3 to 10 Mb/s

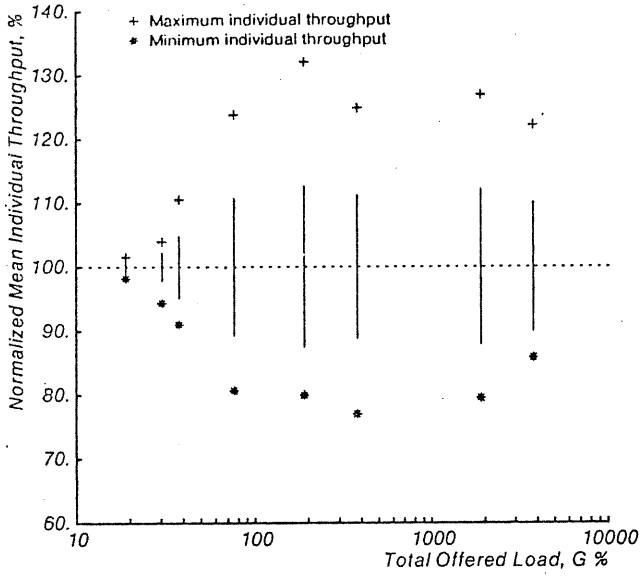


Fig. 4.13a 10 Mb/s Ethernet: Variation in η per Host
$P = 64$ bytes

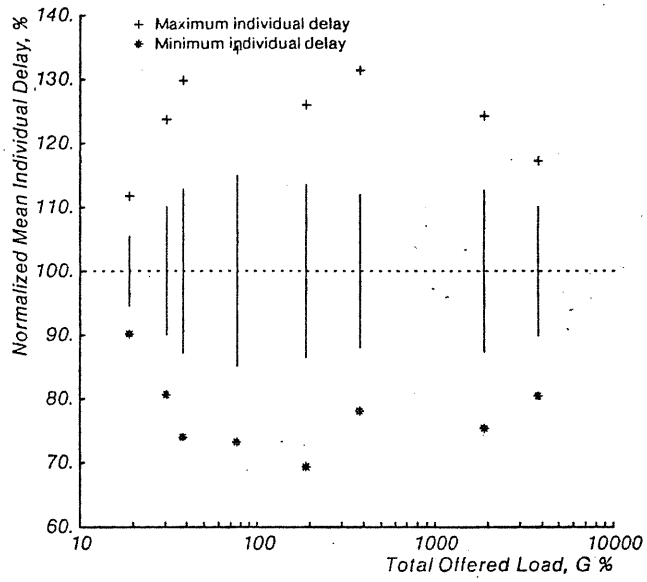


Fig. 4.14a 10 Mb/s Ethernet: Variation in Delay per Host
$P = 64$ bytes

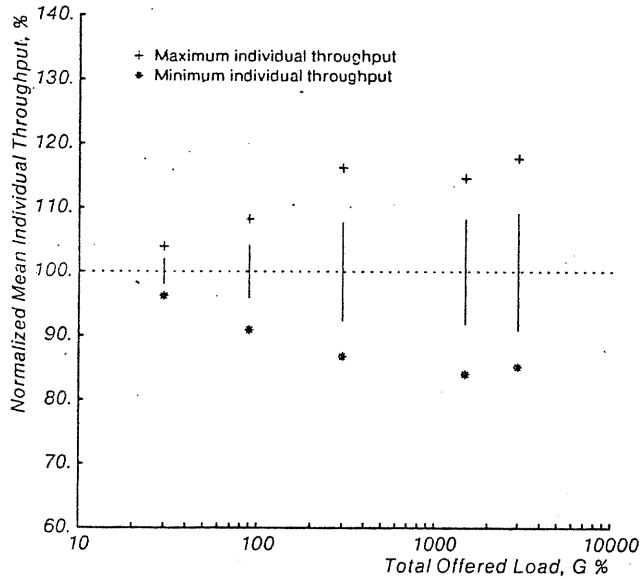


Fig. 4.13b 10 Mb/s Ethernet: Variation in η per Host
$P = 512$ bytes

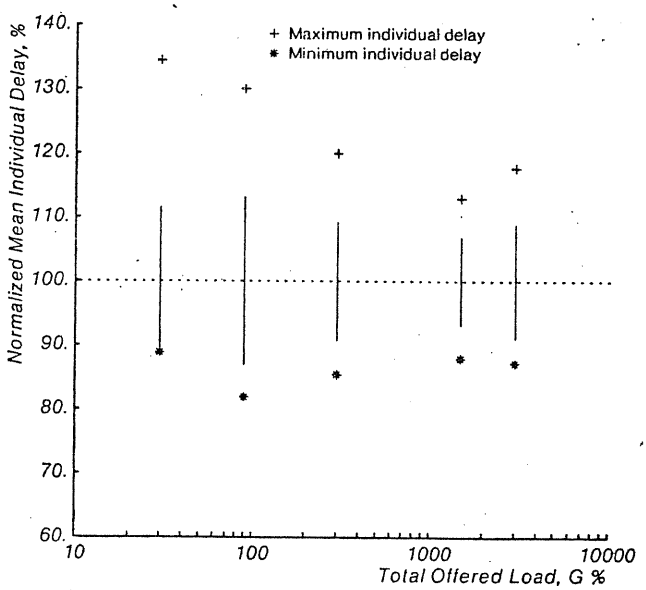


Fig. 4.14b 10 Mb/s Ethernet: Variation in Delay per Host
$P = 512$ bytes

Delay: Fig. 4.16 shows mean packet delay as a function of total offered load, in Mb/s, for the two networks and several values of P. The shapes of all the curves are similar: there is a region of minimal delay at low G, then there is a rapid increase in delay to some saturation value at high G. For low G, the delay is lower in the 10 Mb/s network than in the 3 Mb/s one for a given P. However, in the region of overload, the 3 Mb/s network exhibits lower delay. This is due to the more severe back-off algorithm of the 10 Mb/s network (ref. Sections 2.2 and 2.3).

4.4. Comparison with Analytical Predictions

We consider three analytical models: a simple model for computing maximum throughput [9], and two more sophisticated models for computing maximum throughput and delay-throughput characteristics [8, 15]. All the models assume that the channel is slotted, with slot-time related to the end-to-end propagation delay. Performance is expressed in terms of a parameter, a, equal to the ratio of the slot-time to the transmission time of a packet, i.e.,

$$a = \tau_p \,/\, (P \,/\, C)$$

Further details of the analyses may be found in the papers cited.

First we consider the maximum throughput. From [9] we compute η with the number of hosts, N, equal to 32 to correspond to our measurements (η does not vary much with N). We use the infinite population analysis of [15] to obtain η_{max} from the formula for η as a function of G. Table 4.4 shows measured and computed values of maximum throughput for various values of P for the 3 Mb/s Ethernet. τ_p is estimated to be 3 μs (see Section 2.2). Table 4.5 and 4.6 show corresponding sets of values for the 10 Mb/s Ethernet. The measured

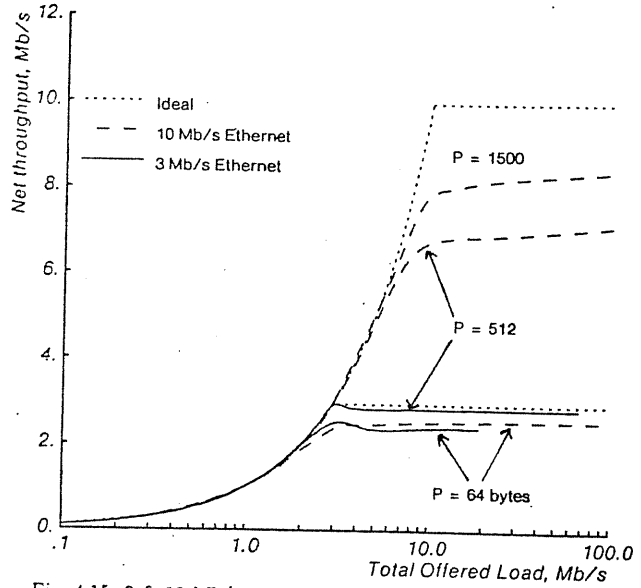


Fig. 4.15 3 & 10 Mb/s Ethernets: Throughput vs. Offered Load

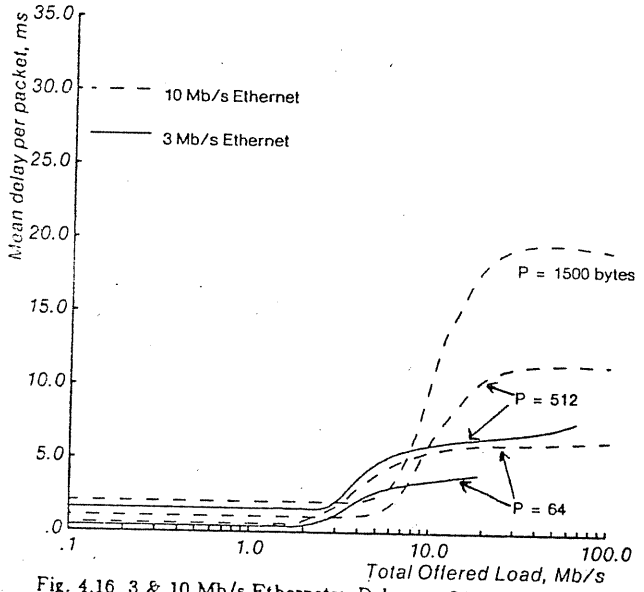


Fig. 4.16 3 & 10 Mb/s Ethernets: Delay vs. Offered Load

P bytes	a	Measured	Metcalfe & Boggs	Tobagi & Hunt
64	0.016	82	87	83
128	0.008	89	93	91
512	0.002	97	98	98

Table 4.4 3 Mb/s Ethernet: Maximum Throughput, %
$\tau_p = 3\ \mu s$ (550 m)

values in Table 4.5 were obtained on a configuration consisting of 750 m of cable with 1 repeater. The measured values in Table 4.6 were obtained on the configuration described in Section 2.3. τ_p is estimated at 11.75 and 15 μs respectively.[6]

P bytes	a	Measured	Metcalfe & Boggs	Tobagi & Hunt
64	0.22	26	55	51
200	0.072	62	79	72
512	0.028	72	91	87
1500	0.0098	86	97	96
5000	0.0029	95	99	99

Table 4.5 10 Mb/s Ethernet: Maximum Throughput, %
$\tau_p = 11.75\ \mu s$ (750 m + 1 repeater)

P bytes	a	Measured	Metcalfe & Boggs	Tobagi & Hunt
64	0.28	26	49	46
200	0.092	60	75	68
512	0.036	72	88	83
1500	0.012	85	96	94
5000	0.0037	94	99	98
10000	0.0019	97	99	99

Table 4.6 10 Mb/s Ethernet: Maximum Throughput, %
$\tau_p = 15\ \mu s$ (1500 m + 2 repeaters)

It is seen that both the models overestimate η_{max}, the formula of Metcalfe & Boggs being less accurate than that of Tobagi & Hunt. The error is larger at smaller values of P, i.e., at larger values of a. Some factors affecting the accuracy of the analytical predictions are the accuracy of the estimation of τ_p and the various assumptions made in the analyses. In particular, the assumption that transmission attempts are made only at the beginning of slots would lead to higher predicted throughputs. On the other hand, the assumption that every pair of hosts is separated by τ_p would lower the predicted throughput. A third differences between the experimental and analytical models include the retransmission algorithm: Metcalfe & Boggs assume an optimum policy, Tobagi & Hunt assume an infinite retransmission delay; and the packet arrival process: the analyses assume Poisson arrivals, the measurements were made with uniformly distributed arrivals.

In Fig. 4.17, measured and predicted ([8], Fig. 2) delay-throughput characteristics are plotted for $a = 0.1$ and 0.01. The measured curves for the 10 Mb/s Ethernet with $a = 0.092$ and 0.012 correspond to $P = 200$ and 1500 bytes respectively. For the 3 Mb/s Ethernet curve, $P = 128$ bytes.[7] Considering $a = 0.1$, at high η the predicted and measured curves correspond closely. At low η, however, the measured delay is much higher than predicted. This is due in part to the software overhead in the measurements - delays of a fraction of a millisecond could not be measured. Looking at $a = 0.01$, we note again the larger measured delays at low η. At high η, in the case of the 3 Mb/s Ethernet, the prediction varies from being optimistic to approximate correspondence. In the case of the 10 Mb/s Ethernet, however, the analysis consistently underestimates delay. Possible reasons for these discrepancies have been discussed above.

In summary, the simplistic formula of Metcalfe & Boggs overestimates the maximum throughput. The error is greatest for large a. This, though, is precisely the region in which accurate estimation of performance is most important because network

[6] a is computed using the total packet length, including overhead. Throughputs shown are net, excluding overhead.

[7] From the numerical results in [8] we note that the percentage change in predicted values for small Δa is less than $\Delta a/a$. Hence, we can make comparisons between measured curves with $a = 0.012$ and 0.008 and a predicted curve with $a = 0.01$, and between a measured curve with $a = 0.092$ and a predicted curve with $a = 0.1$.

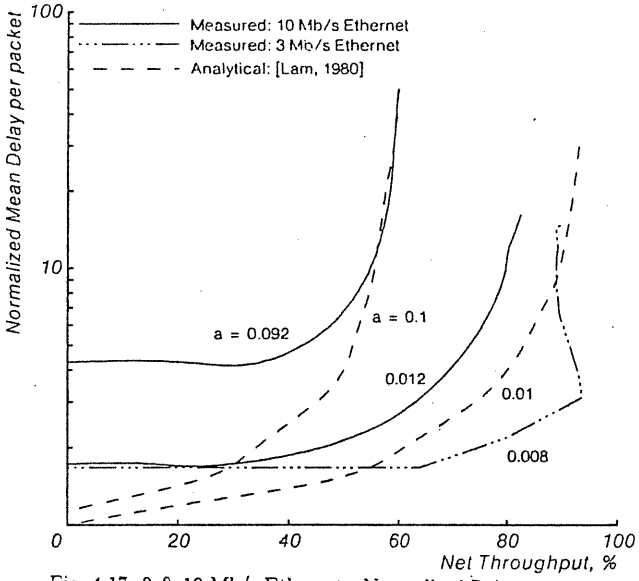


Fig. 4.17 3 & 10 Mb/s Ethernets: Normalized Delay vs. η
Comparison of Measurement and Analysis

performance in this region is poor. The more sophisticated analysis of Tobagi & Hunt yields more accurate though still optimistic predictions.[8] Lam's delay-throughput predictions are seen to vary from approximately correct to optimistic. The accuracy is not consistent for a given network or a.

5. Conclusions

We have presented results of performance measurements on 3 and 10 Mb/s Ethernets with various packet lengths and offered loads ranging from a small fraction of network bandwidth to heavy overload. These experiments span the range from the region of high performance of CSMA/CD networks to the limits at which performance begins to degrade seriously. The former occurs when the packet transmission time is large compared to the round-trip propagation delay. The latter occurs when the two times are comparable in magnitude.

Packet delay is seen to be minimal over normal operating ranges of offered load. However, at saturation, the delay for some packets increases by orders of magnitude although the increase in average delay in much lower. Saturation occurs at lower values of offered load when the packet transmission time is comparable to the round-trip propagation delay. The protocol is fair to all contenders, with the 10 Mb/s Ethernet exhibiting somewhat higher variation in performance measured by individual hosts than the 3 Mb/s Ethernet.

Comparison of our measurments with the predictions of several analytical models indicate that the predictions tend to be optimistic, the more complex analyses yielding more accurate predictions than a simple model. The accuracy of the predictions is affected by the estimation of the propagation delay across the network.

Thus, the Ethernet protocol is seen to be very suitable for local interconnection when the packet length can be made sufficiently large and occasionally highly variable delays can be tolerated. This matches the nature and requirements of most computer communication traffic. However, for real-time communications, such as digital voice telephony, the utility of the Ethernet is more restricted.

[8] Tobagi & Hunt also analyse several aspects of CSMA/CD performance that we do not look at here.

6. Acknowledgements

Thanks are due to several people for their contributions to this work. John Shoch made available the measurement tools developed by himself and Jon Hupp for the 3 Mb/s Ethernet. Alan Freier and Hal Murray provided assistance in the development of measurement tools for the 10 Mb/s Ethernet. Forest Baskett, Yogen Dalal and Larry Garlick provided advice at various stages of the experiments. Fouad Tobagi's technical and editorial comments were helpful.

References

1. G. T. Almes, and E. D. Lazowska. The Behaviour of Ethernet-like Computer Communication Networks. Proc. of 7th Symp. on Operating Sys. Prins., Asilomar, California, Dec., 1979, pp. 66-81.

2. P. D. Amer. "A Measurement Center for the NBS Local Area Computer Network." *IEEE Transactions on Computers TC-31*, 8 (Aug. 1982), 723-729.

3. D. R. Boggs. Private communication, Xerox PARC, 1982.

4. R. C. Crane, and E. A. Taft. Practical Considerations in Ethernet Local Network Design. 13th Hawaii Intl. Conf. on System Sciences, Honolulu, Jan., 1980, pp. 166-174.

5. *The Ethernet, A Local Area Network: Data Link Layer and Physical layer Specifications.* Version 1 edition, DEC, Intel & Xerox Corps., 1980.

6. T. A. Gonsalves. Packet-Voice Communication on an Ethernet Local Network: an Experimental Study. ACM SIGCOMM Symposium on Communications Architectures and Protocols, Austin, Texas, Mar., 1983, pp. 178-185.

7. L. Kleinrock, and F. A. Tobagi. "Packet Switching in Radio Channels: Part I – Carrier Sense Multiple-access Modes and their Throughput-delay Characteristics." *IEEE Transactions on Communications COM-23*, 12 (Dec. 1975), 1400-1416.

8. S. S. Lam. "A Carrier Sense Multiple Access Protocol for Local Networks." *Computer Networks 4*, 1 (Feb. 1980), 21-32.

9. R. M. Metcalfe, and D. R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." *Comm. ACM 19*, 7 (Jul. 1976), 395-404.

10. G. J. Nutt, and D. L. Bayer. "Performance of CSMA/CD Networks Under Combined Voice and Data Loads." *IEEE Transactions on Communications COM-30*, 1 (Jan. 1982), 6-11.

11. J. F. Shoch. *Design and Performance of Local Computer Networks.* Ph.D. Th., Dept. of Computer Science, Stanford Univ., Stanford, CA-94305, Aug. 1979.

12. J. F. Shoch, and J. Hupp. "Measured Performance of an Ethernet Local network." *Comm. ACM 23*, 12 (Dec. 1980), 711-721.

13. C. P. Thacker, E. M. McCreight, B. W. Lampson, and D. R. Boggs. Alto: A Personal Computer. In *Computer Structures: Principles and Examples*, D. P. Siewiorek, C. G. Bell, and A. Newell, Eds., McGraw-Hill Book Co., 1982. pp. 549-572.

14. F. A. Togabi, and N. Gonzalez-Cawley. On CSMA-CD Local Networks and Voice Communication. *INFOCOM '82*, Las Vegas, Nevada, Mar./Apr., 1982, pp. .

15. F. A. Tobagi, and V. B. Hunt. "Performance Analysis of Carrier Sense Multiple Access with Collision Detection." *Computer Networks 4*, 5 (Oct./Nov. 1980), 245-259.

16. R. E. Toense. "Performance Analysis of NBSNET." *Journal of Telecommunication Networks 2*, 2 (Spring 1983), 177-186.

APPENDIX E: Program listing

```
(*-------------------------------------------------------------------
  |                                                                 |
  |                          G L O B A L S . I                      |
  |                                                                 |
  -------------------------------------------------------------------*)


const
      BPS    = 128000;          (* Transmission rate                 *)
      MAXDEVICES = 30;          (* Maximum devices on channel        *)
      NUMLENGTHS =  4;          (* Number of different message
                                     lengths.                       *)
      INTERACTIVE = true;       (* Flag determining whether inputs
                                     are interactive                *)
      ROWSIZE = 5;              (* Row size of printing matrix       *)

type
      matrix    = array [ 0..MAXDEVICES, 0..MAXDEVICES ] of real;
      list      = array [ 0..MAXDEVICES ] of real;

      messlist   = array [ 1..NUMLENGTHS ] of real;
      lengthlist = array [ 1..NUMLENGTHS ] of integer;

      statuslist = (err1, err2, err3, err4, err5);
      statusset  = set of statuslist;

var
      p,          (* Transition prob.matrix between consecutive
                        embedded points (first slot of each idle
                        period )                                    *)

      q,          (* Transition prob.matrix for all busy period slots*)

      f,          (* Transition prob.matrix, given transmission
                        unsuccessful for slot Te+I-1                *)

      s,          (* Transition prob.matrix, given transmission
                        successful for slot Te+I-1                  *)

      j           (* Prob.matrix that a successful transmission
                        decreases backlog by 1                      *)

         : matrix;


      pi,         (* Stationary prob distribution of Nt at embedded
                        points                                      *)

      ps,         (* Probability of a successful transmission during a
                        cycle, given the number of backlogged devices *)

      delta,      (* Avg. idle period, given Nte = i               *)

      a           (* Expected sum of backlogs over all slots in the
                        busy period, with Nte = i                   *)
```

```
    : list;

T,          (* Transmission times in slots of a packet          *)

problength
            (* Probabilities of encountering different message
               lengths                                          *)

    : messlist;

length      (* Different message lengths                        *)

    : lengthlist;

xi,         (* Time it takes a device to detect interference,
               once the latter has reached it                   *)

zeta,       (* Period used for collision consensus reenforcement *)

sigma,      (* Prob that a device generates and transmits a new
               packet in a slot, given that the channel is
               sensed idle.                                     *)

nu,         (* Prob that each backlogged user senses the channel
               in the current slot                              *)

delay,      (* Average packet delay, in seconds                 *)

delaytmean, (* delay above, normalized to Tmean                 *)

delayslot,  (* delay above, in slot                             *)

gamma,      (* Given a collision occurs, the time until all
               devices stop transmission                        *)

nmean,      (* Average channel backlog                          *)

thruput,    (* Average stationary channel thruput               *)

W,          (* Average channel acquisition time/mean waiting
               time until successful transmission               *)

Wslot,      (* W above, in slots                                *)

Tmean,      (* Mean transmission time, in slots, of a packet /
               average packet size                              *)

tau,        (* End-to-end propagation delay = 1 slot time       *)

value       (* Error value used in determinestatus to aid user *)

    : real;
```

```
M,        (* Number of devices, must be < MAXDEVICES          *)
iImean,   (* IMEAN rounded to an integer                      *)
igamma    (* gamma rounded to an integer                      *)
    : integer;

outdebug  (* output file for debug purposes                   *)
    : text;

TRACE,                    (* Flag for tracing execution      *)
DEBUG,                    (* Flag for debugging program      *)
errorflag      (* flag indicating if error occurs in procedure
                  calculations.                                *)
    : boolean;

status    (* error status set                                 *)
    : statusset;
```

```
(*-------------------------------------------------------------
 :                                                            :
 :                  S U P P O R T R O U T I N E S . I         :
 :                                                            :
 -------------------------------------------------------------*)


(*******************************************************)
function power (base : real; superscript : integer) : real;
(*******************************************************)
(*                                                     *)
(*  Purpose:  This routine computes the base raised to a   *)
(*            given power (integer).                   *)
(*                                                     *)
(*  Called by:                                         *)
(*            Omatrix, Fmatrix, Smatrix, computePS,    *)
(*            computeDELTA                             *)
(*                                                     *)
(*  Routines called:  ---                              *)
(*                                                     *)
(*  Global variables: ---                              *)
(*                                                     *)
(*******************************************************)

   var i : integer;
       plusscript : integer;
       temp : real;

   begin
   temp := 1.0;

   if superscript >= 0 then

       for i := 1 to superscript do
           temp := temp * base

   else

       begin
       plusscript := abs(superscript);
       for i := 1 to plusscript do
           temp := temp * (1/base)
       end;

   power := temp;
   end;


(*******************************************************)
function factorial (n : integer) : integer;
(*******************************************************)
(*                                                     *)
(*  Purpose:  This routine computes the factorial of a     *)
(*            given positive integer.                  *)
(*                                                     *)
(*  Called by:                                         *)
```

```
(*              Qmatrix, Fmatrix                                    *)
(*                                                                  *)
(*  Routines called:  ---                                           *)
(*                                                                  *)
(*  Global variables: ---                                           *)
(*                                                                  *)
(******************************************************************)


   var i,                  (* counter                               *)
       tempfact            (* temporary variable used in computing
                              factorial                             *)
           : integer;

   begin
   tempfact := 1;

   if n >= 0 then

       for i := n downto 2 do
          tempfact := tempfact * i

   else
       if INTERACTIVE then
          writeln('In factorial: n < 0');


   factorial := tempfact
   end;


(******************************************************************)
procedure matrixzeros ( var result : matrix );
(******************************************************************)
(*                                                                  *)
(*  Purpose:  This routine initializes a given matrix to           *)
(*            contain all zero elements                             *)
(*                                                                  *)
(*  Called by:                                                      *)
(*            computeGT                                             *)
(*                                                                  *)
(*  Routines called:  ---                                           *)
(*                                                                  *)
(*  Global variables: ---                                           *)
(*                                                                  *)
(******************************************************************)

   var row, col        (* counters for manipulating matrix          *)
           : integer;

   begin

   for row := 0 to M do
      for col := 0 to M do
         result[row, col] := 0.0;
```

```
end;

(*******************************************************************)
procedure matrixones ( var result : matrix );
(*******************************************************************)
(*                                                                 *)
(*  Purpose:  This routine initializes a given matrix to           *)
(*            contain all ones                                     *)
(*                                                                 *)
(*  Called by:                                                     *)
(*            matrixpower                                          *)
(*                                                                 *)
(*  Routines called:  ---                                          *)
(*                                                                 *)
(*  Global variables: ---                                          *)
(*                                                                 *)
(*******************************************************************)


   var row, col       (* counters for manipulating matrix    *)
          : integer;

   begin

   for row := 0 to M do
      for col := 0 to M do
         result[row, col] := 1.0;

   end;

(*******************************************************************)
procedure matrixequate ( first : matrix; var result : matrix );
(*******************************************************************)
(*                                                                 *)
(*  Purpose:  This routine equates the second matrix to the        *)
(*            first.                                               *)
(*                                                                 *)
(*  Called by:                                                     *)
(*            matrixpower                                          *)
(*                                                                 *)
(*  Routines called:  ---                                          *)
(*                                                                 *)
(*  Global variables: ---                                          *)
(*                                                                 *)
(*******************************************************************)


   var row, col       (* counters for manipulating matrix    *)
          : integer;

   begin

   for row := 0 to M do
```

```
      for col := 0 to M do
         result[row, col] := first[row,col];

   end;


(*********************************************************)
procedure matrixadd (first, second:matrix; var result:matrix);
(*********************************************************)
(*                                                       *)
(*  Purpose:  This routine adds two matrices together and *)
(*            stores the result in the third parameter   *)
(*                                                       *)
(*  Called by:                                           *)
(*            Pmatrix, computeGT, computeA               *)
(*                                                       *)
(*  Routines called:  ---                                *)
(*                                                       *)
(*  Global variables: ---                                *)
(*                                                       *)
(*********************************************************)


   var row, col       (* counters for manipulating matrix   *)
          : integer;

   begin

   for row := 0 to M do
      for col := 0 to M do
         result[row, col] := first[row,col] + second[row,col];

   end;


(*********************************************************)
procedure matrixXconstant (number : real;
                           base:matrix;
                           var result : matrix);
(*********************************************************)
(*                                                       *)
(*  Purpose:  This routine multiplies a matrix by a constant*)
(*            and stores the result in the third parameter *)
(*                                                       *)
(*  Called by:                                           *)
(*            computeGT                                  *)
(*                                                       *)
(*  Routines called:  ---                                *)
(*                                                       *)
(*  Global variables: ---                                *)
(*                                                       *)
(*********************************************************)


   var row, col       (* counters for manipulating matrix   *)
          : integer;
```

```
   begin

   for row := 0 to M do
      for col := 0 to M do
         result[row, col] := number * base[row,col];

   end;


(*************************************************************)
procedure matrixmultiply (first, second : matrix;
                          var result : matrix );
(*************************************************************)
(*                                                           *)
(*  Purpose:  This routine multiplies two matrices together  *)
(*            and stores the result in the third             *)
(*                                                           *)
(*  Called by:                                               *)
(*            Pmatrix, computeA                              *)
(*                                                           *)
(*  Routines called:  ---                                    *)
(*                                                           *)
(*  Global variables: ---                                    *)
(*                                                           *)
(*************************************************************)


   var row, col, i      (* counters for manipulating matrix  *)
         : integer;
       sum : real;

   begin

   for row := 0 to M do
      for col := 0 to M do
         begin

         sum := 0.0;

         for i := 0 to M do
             sum := sum + first[row,i]*second[i,col];

         result[row,col] := sum;

         end;

   end;


(*************************************************************)
procedure matrixpower ( base:matrix; superscript:integer;
                        var result : matrix );
(*************************************************************)
(*                                                           *)
(*  Purpose:  This routine computes a matrix raised to a     *)
```

```
(*             given power, with the result stored in the     *)
(*             third parameter.                               *)
(*                                                            *)
(*  Called by:                                                *)
(*             computeGT, Pmatrix, computeA                   *)
(*                                                            *)
(*  Routines called:                                          *)
(*             matrixequate, matrixones                       *)
(*                                                            *)
(*  Global variables: ---                                     *)
(*                                                            *)
(**********************************************************)

   var row,col,i,l : integer;  (* counters for manipulating
                                  matrix                     *)
       sum : real;
       temp : matrix;

   begin

   if superscript = 0 then
       matrixones (result)
   else if superscript = 1 then
       matrixequate (base, result)

   else if superscript >= 2 then
       begin

       matrixequate(base,temp);

       for i := 2 to superscript do
          begin
          for row := 0 to M do
              for col := 0 to M do
                 begin
                 sum := 0.0;
                 for l := 0 to M do
                     sum := sum + temp[row,l] * base[l,col];
                 result[row,col] := sum;
                 end;
          matrixequate(result,temp);
          end;
       end

   else
       if INTERACTIVE then
          writeln('In matrixpower: Superscript must be >= 0');

   end;

(**********************************************************)
procedure printmatrix ( row , col : integer; var out : matrix );
(**********************************************************)
(*                                                        *)
(*  Purpose:  This routine prints outs a matrix that is   *)
```

```
(*            row X col.                                                *)
(*                                                                      *)
(*  Called by: Qmatrix, Jmatrix, Fmatrix, Smatrix,                      *)
(*             computeGT, Pmatrix, gausselimination                     *)
(*                                                                      *)
(*  Routines called:  ---                                               *)
(*                                                                      *)
(*  Global variables: ---                                               *)
(*                                                                      *)
(************************************************************************)

   var  i, k, l : integer;
        index, printrow, lastrow : integer;

   begin
   printrow := col  div ROWSIZE;
   lastrow  := col  mod ROWSIZE;

   for i := 0 to (row-1) do
       begin
       writeln(outdebug);
       writeln(outdebug,'...row ',i:3);
       index := 0;
       for k := 1 to printrow do
           begin
           for l := 1 to ROWSIZE do
               begin
               write(outdebug,'      ',out[i,index]);
               index := index + 1
               end;
           writeln(outdebug);
           end;

       for k := 1 to lastrow do
           begin
           write(outdebug,'      ',out[i,index]);
           index := index + 1
           end;

       writeln( outdebug );

       end;

    writeln( outdebug );
    flush (outdebug);

   end;


(************************************************************************)
procedure initstatus;
(************************************************************************)
(*                                                                      *)
(*  Purpose:  This routine initializes the error status set             *)
(*            used in procedure calculations.                           *)
(*                                                                      *)
```

```
(*  Called by: initializations                                          *)
(*                                                                      *)
(*  Routines called:  ---                                               *)
(*                                                                      *)
(*  Global variables: status                                            *)
(*                                                                      *)
(**********************************************************************)

    begin
    status := [];
    end;


(**********************************************************************)
procedure addstatus(err : statuslist);
(**********************************************************************)
(*                                                                      *)
(*  Purpose:  This routine adds the error "err" to the error*)
(*            status set.                                               *)
(*                                                                      *)
(*  Called by: gausselimination, testpi                                 *)
(*                                                                      *)
(*  Routines called:  ---                                               *)
(*                                                                      *)
(*  Global variables: status                                            *)
(*                                                                      *)
(**********************************************************************)

    begin
    status := status + [ err ];
    end;


(**********************************************************************)
procedure determinestatus;
(**********************************************************************)
(*                                                                      *)
(*  Purpose:  This routine determines which errors were                 *)
(*            encountered in procedure calculations and                 *)
(*            prints out an appropriate message for each.               *)
(*                                                                      *)
(*  Called by: calculations                                             *)
(*                                                                      *)
(*  Routines called:  ---                                               *)
(*                                                                      *)
(*  Global variables: status                                            *)
(*                                                                      *)
(**********************************************************************)

    begin

    if err1 in status then
       begin

       if INTERACTIVE then
          begin
```

```
      writeln('***ERROR: in procedure GAUSSELIMINATION***');
      writeln('     Cannot find element <> 0 in row; the ');
      writeln('     P matrix is not invertible.');
      end;

   if TRACE or DEBUG then
      begin
      writeln(outdebug,'***ERROR: in procedure GAUSSELIMINATION***');
      writeln(outdebug,'     Cannot find element <> 0 in row; the ');
      writeln(outdebug,'     P matrix is not invertible.');
      end;

   end;

if err2 in status then
   begin

   if INTERACTIVE then
      begin
      writeln('***ERROR: in procedure GAUSSELIMINATION***');
      writeln('     P[M,M] = 0; the P matrix is not invertible.');
      end;

   if TRACE or DEBUG then
      begin
      writeln(outdebug,'***ERROR: in procedure GAUSSELIMINATION***');
      writeln(outdebug,'     P[M,M] = 0; the P matrix is not invertible.');
      end;

   end;

if err3 in status then
   begin

   if INTERACTIVE then
      begin
      writeln('***ERROR: in procedure TESTPI***');
      writeln('     The sum of PIs <> 1.0');
      end;

   if TRACE or DEBUG then
      begin
      writeln(outdebug,'***ERROR: in procedure TESTPI***');
      writeln(outdebug,'     The sum of PIs <> 1.0');
      end;

   end;

if err4 in status then
   begin

   if INTERACTIVE then
      begin
      writeln('***ERROR: in procedure TESTPI***');
      writeln('     Last row of matrix w X pi <> 1, but to ', value);
      end;
```

```
    if TRACE or DEBUG then
       begin
       writeln(outdebug,'***ERROR: in procedure TESTPI***');
       writeln(outdebug,'     Last row of matrix w X pi <> 1, but to ', value
       end;

    end;

 if err5 in status then
    begin

    if INTERACTIVE then
       begin
       writeln('***ERROR: in procedure TESTPI***');
       writeln('     Colmax value too small');
       end;

    if TRACE or DEBUG then
       begin
       writeln(outdebug,'***ERROR: in procedure TESTPI***');
       writeln('     Colmax value too small');
       end;

    end;

 end;
```

```
(*-------------------------------------------------------------
 |                                                            |
 |                        I N P U T S . I                     |
 |                                                            |
 -------------------------------------------------------------*)


(*********************************************************)
procedure getinputs;
(*********************************************************)
(*                                                       *)
(*  Purpose: This routine prints out the default input values *)
(*           and gives the user the chance to change them.  *)
(*                                                       *)
(*  Called by:                                           *)
(*           model                                       *)
(*                                                       *)
(*  Routines called:                                     *)
(*           printinputs, changedefaults                 *)
(*                                                       *)
(*  Global variables:                                    *)
(*           tau, M, xi, zeta, sigma, nu, igamma, gamma, *)
(*           problength, length, T, Tmean, iTmean, TRACE, *)
(*           DEBUG                                       *)
(*                                                       *)
(*********************************************************)

  type printtype = (initial,default);

  var ch : char;
      i  : integer;


   (*-----------------------------------------------------*)
   procedure printinputs( kind:printtype);
   (*-----------------------------------------------------*)
   (*                                                     *)
   (*  Purpose:  This routine prints out the input values *)
   (*            standard output.                         *)
   (*                                                     *)
   (*  Called by:                                         *)
   (*            getinputs                                *)
   (*                                                     *)
   (*  Routines called:                                   *)
   (*            -----                                    *)
   (*                                                     *)
   (*  Global variables:                                  *)
   (*            tau, M, xi, zeta, sigma, nu, igamma, gamma, *)
   (*            problength, length, T, Tmean, iTmean, TRACE, *)
   (*            DEBUG                                    *)
   (*                                                     *)
   (*-----------------------------------------------------*)

      var i : integer;

      begin
```

```
   writeln;
   writeln;

   case kind of
      default : writeln('DEFAULTS:');
      initial : writeln('INITIALIZATIONS:');
    end;

   writeln;

   writeln('...Message Info:');
   writeln;
   writeln('   Message #   Length      Prob.       Slot Time(T)');
   writeln('   ---------   ------    ----------    ----------------');
   for i := 1 to NUMLENGTHS do
      writeln('    ',i:5,length[i]:12,'     ',problength[i]:10,'    ',
                                 T[i]:16);


   writeln;
   writeln('...Other Info:');
   writeln;
   writeln('    ','M=',M:2,'                          TAU=',tau:6);
   writeln('    ','GAMMA=',gamma:8,'                IGAMMA=',igamma:2);
   writeln('    ','NU=',nu:8,'                    SIGMA=',sigma:8);
   writeln('    ','TMEAN=',Tmean:10,'             ITMEAN=',iTmean:2);
   writeln('    ','XI=',xi:10,'                  TRACE=',TRACE);
   writeln('    ','ZETA=',zeta:10,'               DEBUG=',DEBUG);
   writeln;
   writeln;

   end;

(*-------------------------------------------------------------*)
procedure changedefaults;
(*-------------------------------------------------------------*)
(*                                                             *)
(*  Purpose: This routine lets the user change the default     *)
(*           values. Each default is printed and the user      *)
(*           is given the option of changing it.  After        *)
(*           all defaults have been evaluated, the user        *)
(*           is given the option to redo them again.           *)
(*                                                             *)
(*  Called by:                                                 *)
(*           getinputs                                         *)
(*                                                             *)
(*  Routines called:                                           *)
(*           ask, prompt                                       *)
(*                                                             *)
(*  Global variables:                                          *)
(*           tau, M, xi, zeta, sigma, nu, igamma, gamma,       *)
(*           problength, length, T, Tmean, iTmean, TRACE,      *)
(*           DEBUG                                             *)
(*                                                             *)
(*-------------------------------------------------------------*)

   const
```

```
   epsilon = 0.00001;

var
   valid, done : boolean;
   i    : integer;
   sum, temp : real;

(*.....................................................*)
function ask : boolean;
(*.....................................................*)
(*                                                     *)
(*  Purpose: This routine prompts the user,to determine *)
(*           if they want to change a given default    *)
(*           value.  The user must respond with a Y, y, *)
(*           N, or n to leave this routine.  The       *)
(*           function is then assigned the appropriate  *)
(*           true or false value.                      *)
(*                                                     *)
(*  Called by:                                         *)
(*           changedefaults                            *)
(*                                                     *)
(*  Routines called:  ----                             *)
(*                                                     *)
(*  Global variables: ----                             *)
(*                                                     *)
(*.....................................................*)

   var ch : char;

   begin

   write('...Do you want to change this? (Y/N):  ');
   readln(ch);

   if ch in ['Y','y','N','n'] then
      case ch of
        'Y','y':  ask := true;
        'N','n':  ask := false
        end
   else
      begin
      repeat
        write('Try again...Do you want to change this? (Y/N):  ');
        readln(ch)
      until ch in ['Y','y','N','n'];

      case ch of
        'Y','y':  ask := true;
        'N','n':  ask := false
        end;
      end;

   end;


(*.....................................................*)
```

```
procedure prompt;
(*.....................................................*)
(*                                                     *)
(*  Purpose:  This routine prints out a prompt for the *)
(*            user to enter the new input value.       *)
(*                                                     *)
(*  Called by:                                         *)
(*            changedefaults                           *)
(*                                                     *)
(*  Routines called:  -------                          *)
(*                                                     *)
(*  Global variables:  ------                          *)
(*                                                     *)
(*.....................................................*)

  begin
  write('>>>Enter value:  ');
  end;


(*.....................................................*)

begin  (* changedefaults *)

repeat
   writeln;
                              (* Change tau?           *)

   writeln('Slot time (tau) =',tau:8);
   if ask then
      begin
      prompt;
      readln(tau);
      end;

                              (* Change M?             *)
   writeln;
   writeln('Number of devices (M) =',M:3);
   if ask then
      repeat
         prompt;
         readln(M);
         if M > MAXDEVICES then
            begin
            valid := false;
            writeln('***Error: MAXDEVICES =',MAXDEVICES)
            end
         else
            valid := true
      until valid;

                              (* Change xi?            *)
   writeln;
   writeln('Bit delay for device to recognize interference, ',
           'once it has ');
   writeln('reached device ( #bits/BPS = xi ) =',xi*BPS:5:1);
```

```
if ask then
   begin
   prompt;
   readln(temp);
   xi := temp/BPS;
   end;

                              (* Change zeta?                *)
writeln;
writeln('Period used for collision reinforcement, ',
        'in seconds ');
writeln('(zeta) =',zeta:8);
if ask then
   begin
   prompt;
   readln(zeta);
   end;

                              (* Change sigma?               *)
writeln;
writeln('Prob. that device generates and xmits new ',
        'packet in slot, ');
writeln('given that channel is idle (sigma) = ',sigma:8);
if ask then
   begin
   prompt;
   readln(sigma);
   end;

                              (* Change nu?                  *)
writeln;
writeln('Prob.that backlogged user senses channel in ',
        'current slot ');
writeln('(nu) =',nu:8);
if ask then
   begin
   prompt;
   readln(nu);
   end;

                              (* Recompute gamma & igamma *)

gamma := (2*tau + xi + zeta)/tau;

igamma := round(gamma);

                              (* Change packet probs?        *)
repeat
   sum := 0.0;
   for i := 1 to NUMLENGTHS do
      begin
      writeln;
      writeln('Prob. of packet #',i:1,' =',problength[i]:8);
      if ask then
         begin
         prompt;
```

```
            readln(problength[i]);
            end;
         sum := sum + problength[i];
         end;

   if abs(1-sum) > epsilon then
      begin
      writeln('***Error: sum of probs <> 1.0...Try again');
      valid := false;
      end
   else
      valid := true;

until valid;

                              (* Change packet lengths?    *)
for i := 1 to NUMLENGTHS do
   begin
   writeln;
   writeln('Packet length #',i:1,' =',length[i]:6,' bits');
   if ask then
      begin
      prompt;
      readln(length[i]);
      end;
   end;
                           (* Recompute T[i],Tmean,iTmean *)
Tmean := 0.0;
for i := 1 to NUMLENGTHS do
   begin
   T[i] := ( length[i]/BPS ) / tau;
   Tmean := Tmean + problength[i]*T[i];
   end;

iTmean := round(Tmean);

if iTmean=0 then
   iTmean := 1;

                              (* Change boolean flags?     *)
writeln;
writeln('DEBUG flag = ',DEBUG);
if ask then
   DEBUG := not DEBUG;

writeln;
writeln('TRACE flag = ',TRACE);
if ask then
   TRACE := not TRACE;


                              (* Done??                    *)
writeln;
repeat
   writeln;
   write('Are you satisfied with changes? (Y/N):  ');
```

```
      readln(ch)
   until ch in ['Y','y','N','n'];

   if ch in ['Y','y'] then
      done := true
   else done := false;

until done;

end;

(*-------------------------------------------------------*)

begin (* getinputs *)

if INTERACTIVE then

   begin

   printinputs (default);

   repeat
      writeln;
      write('Do you want to change any defaults? (Y/N): ');
      readln(ch)

   until ch in ['Y','y','N','n'];

   if ch in ['Y','y'] then
      begin
      changedefaults;
      printinputs (initial);
      end;

   end;

if DEBUG or TRACE then
   begin
   writeln(outdebug);
   writeln(outdebug,'INITIALIZATIONS:');
   writeln(outdebug);
   writeln(outdebug,'...Message Info:');
   writeln(outdebug);
   writeln(outdebug,'   Message #   Length      Prob.       Slot Time(T)');
   writeln(outdebug,'   ---------   ------   ----------   ----------------');
   for i := 1 to NUMLENGTHS do
       writeln(outdebug,'   ',i:5,length[i]:12,'    ',problength[i]:10,'   ',
                        T[i]:16);

   writeln(outdebug);
   writeln(outdebug,'...Other Info:');
   writeln(outdebug);
   writeln(outdebug,'   ','M=',M:2,'                    TAU=',tau:6);
   writeln(outdebug,'   ','GAMMA=',gamma:8,'          IGAMMA=',igamma:2);
   writeln(outdebug,'   ','NU=',nu:8,'              SIGMA=',sigma:8);
```

```
    writeln(outdebug,'    ','TMEAN=',Tmean:10,'          ITMEAN=',iTmean:2);
    writeln(outdebug,'    ','XI=',xi:10,'              TRACE=',TRACE);
    writeln(outdebug,'    ','ZETA=',zeta:10,'            DEBUG=',DEBUG);
    writeln(outdebug);
    end;

end;
```

```
(*-------------------------------------------------------------
 |                                                            |
 |                    C A L C R O U T I N E S . I             |
 |                                                            |
 -------------------------------------------------------------*)


(*************************************************************)
procedure Qmatrix;
(*************************************************************)
(*                                                           *)
(*  Purpose: This routine computes the Q matrix, a           *)
(*           transition probability matrix for all busy      *)
(*           period slots.  This matrix is used in           *)
(*           computing the P matrix.                         *)
(*                                                           *)
(*  Called by:                                               *)
(*           calculations                                    *)
(*                                                           *)
(*  Routines called:                                         *)
(*           factorial,  power                               *)
(*                                                           *)
(*  Global variables:                                        *)
(*           q, M, sigma, TRACE, DEBUG                       *)
(*                                                           *)
(*************************************************************)


   var i , k : integer;                  (* Counters         *)

   begin

   if TRACE then
      writeln(outdebug,'>>>In Qmatrix');

   for i := 0 to M do
      for k := 0 to M do

         if i = 0 then

            q[i,k] := 0.0

         else if k < i then

            q[i,k] := 0.0

         else

            q[i,k] := ( factorial(M-i)/(factorial(k-i)*factorial(M-k)) )
                           * power(1-sigma,M-k) * power(sigma,k-i);

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'Q matrix:');
      printmatrix(M+1, M+1, q );
```

```
      end;

   end;


(*****************************************************)
procedure Jmatrix;
(*****************************************************)
(*                                                   *)
(*  Purpose:  This routine calculates the J matrix, which*)
(*            keeps track of decreasing the backlog by   *)
(*            one when a successful transmission occurs  *)
(*                                                   *)
(*  Called by:  calculations                         *)
(*                                                   *)
(*  Routines called: ---                             *)
(*                                                   *)
(*  Global variables:                                *)
(*            j, M, TRACE, DEBUG                     *)
(*                                                   *)
(*****************************************************)

   var i, k : integer;                (* Counters      *)

   begin

   if TRACE then
      writeln(outdebug,'>>>In Jmatrix');

   for i := 0 to M do
      for k := 0 to M do

         if (k = i-1) then
            j[i,k] := 1.0
         else
            j[i,k] := 0.0;

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'J matrix:');
      printmatrix(M+1, M+1, j );
      end;

   end;


(*****************************************************)
procedure Fmatrix;
(*****************************************************)
(*                                                   *)
(*  Purpose:  This routine calculates the F matrix, which*)
(*            is the transition probability matrix given *)
(*            that the transmission is unsuccessful for  *)
(*            Te+I-1.                                *)
(*                                                   *)
```

```
(*  Called by:  calculations                                        *)
(*                                                                  *)
(*  Routines called: power, factorial                               *)
(*                                                                  *)
(*  Global variables:                                               *)
(*            f, M, sigma, nu, TRACE, DEBUG                         *)
(*                                                                  *)
(*******************************************************************)


   var i, k : integer;                  (* Counters                *)

   begin

   if TRACE then
      writeln(outdebug,'>>>In Fmatrix');

   for i := 0 to M do
      for k := 0 to M do

         if (i=0) and (k=0) then

             f[i,k] := 0.0

         else if k < i then

             f[i,k] := 0.0

         else if k = i then

             f[i,k] := power(1-sigma,M-i)
                       * ( 1 - power(1-nu,i) - i*nu*power(1-nu,i-1) )
                       / ( 1 - power(1-nu,i)*power(1-sigma,M-i) )

         else if k = i+1 then

             f[i,k] := (M-i) * sigma * power(1-sigma,M-i-1)
                       * ( 1 - power(1-nu,i) )
                       / ( 1 - power(1-nu,i)*power(1-sigma,M-i) )

         else if k > i+1 then

             f[i,k] := ( factorial(M-i) / (factorial(k-i)*factorial(M-k)) )
                       * power(1-sigma,M-k) * power(sigma,k-i)
                       / ( 1 - power(1-nu,i)*power(1-sigma,M-i) ) ;

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'F matrix:');
      printmatrix(M+1, M+1, f );
      end;

   end;
```

```
(*****************************************************)
procedure Smatrix;
(*****************************************************)
(*                                                   *)
(*  Purpose:  This routine calculates the S matrix, which*)
(*            is the transition probability matrix given *)
(*            that the transmission is successful for    *)
(*            Te+I-1.                                    *)
(*                                                   *)
(*  Called by:  calculations                         *)
(*                                                   *)
(*  Routines called: power                           *)
(*                                                   *)
(*  Global variables:                                *)
(*            s, M, sigma, nu, TRACE, DEBUG          *)
(*                                                   *)
(*****************************************************)

   var i, k : integer;          (* counters          *)

   begin

   if TRACE then
      writeln(outdebug,'>>>In Smatrix');

   for i := 0 to M do
      for k := 0 to M do

         if (i = 0) and (k = 0) then

            s[i,k] := 0.0

         else if (k < i) or (k > i+1) then

            s[i,k] := 0.0

         else if k = i then

            s[i,k] := ( power(1-sigma,M-i) * i * nu * power(1-nu,i-1) )
                      / ( 1 - power(1-nu,i) * power(1-sigma,M-i) )

         else if k = i+1 then

            s[i,k] := ( (M-i) * sigma * power(1-sigma,M-i-1) * power(1-nu,i) )
                      / ( 1 - power(1-nu,i) * power(1-sigma,M-i) );

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'S matrix:');
      printmatrix( M+1, M+1, s );
      end;

   end;
```

```
(*************************************************)
procedure Pmatrix;
(*************************************************)
(*                                               *)
(*  Purpose: This routine computes the P matrix, the  *)
(*           transition probability matrix between    *)
(*           consecutive embedded points (first slot of *)
(*           each idle period.                        *)
(*                                                    *)
(*  Called by:                                        *)
(*           calculations                             *)
(*                                                    *)
(*  Routines called:                                  *)
(*           computeGT, matrixmultiply, matrixpower,  *)
(*           matrixadd                                *)
(*                                                    *)
(*  Global variables:                                 *)
(*           q, f, j, p, M, TRACE, DEBUG, igamma      *)
(*                                                    *)
(*************************************************)

   var GT : matrix;        (* matrix that includes  message
                              lengths                  *)
       result1,result2,    (* used to aid in calculating P *)
       term1,term2         (* repr. terms 1 and 2 in P eqn *)
          : matrix;

   (*-----------------------------------------------*)
   procedure computeGT;
   (*-----------------------------------------------*)
   (*                                               *)
   (*  Purpose: This routine computes the matrix that   *)
   (*           results from having the 0 matrix be the *)
   (*           input parameter to the generating       *)
   (*           function of the distribution of T.      *)
   (*                                                   *)
   (*  Called by:  Pmatrix                              *)
   (*                                                   *)
   (*  Routines called:  matrixpower, matrixXconstant   *)
   (*                    matrixadd, matrixzeros         *)
   (*                                                   *)
   (*  Global Variables: q, t, problength, TRACE, DEBUG, *)
   (*                    GT                             *)
   (*                                                   *)
   (*---------------------------------------------------*)

     var i,                  (*  counters               *)
         iti                 (*  integer t[i] value     *)
            : integer;

     begin

     if TRACE then
        writeln(outdebug,'>>>>>>In computeGT');

     matrixzeros(GT);
```

```
    for i := 1 to NUMLENGTHS do
        begin
        iti := round(T[i]);
        if iti = 0 then
           iti := 1;
        matrixpower (q, iti, result1);
        matrixXconstant (problength[i], result1, result1 );
        matrixadd (GT, result1, GT);
        end;

    if DEBUG then
        begin
        writeln(outdebug);
        writeln(outdebug,'GT matrix:');
        printmatrix( M+1, M+1, GT );
        end;


    end;

(*-----------------------------------------------*)

begin

if TRACE then
   writeln(outdebug,'>>>In Pmatrix');

computeGT;

if TRACE then
   writeln(outdebug,'>>>In Pmatrix');

matrixpower (q, igamma+1, result1);

matrixmultiply (f, result1, term2);
matrixmultiply (s, GT, result1);
matrixmultiply (result1, q, result2);
matrixmultiply (result2, j, term1);

matrixadd( term1, term2, p);

if DEBUG then
   begin
   writeln(outdebug);
   writeln(outdebug,'P matrix:');
   printmatrix( M+1, M+1, p );
   end;

end;

(*************************************************)
procedure computePS;
(*************************************************)
(*                                               *)
```

```
(*  Purpose: This routine calculates the ps array, which *)
(*           contains the probs. of a successful trans-  *)
(*           mission during a cycle, given the number of *)
(*           backlogged devices.                         *)
(*                                                       *)
(*  Called by:                                           *)
(*           calculations                                *)
(*                                                       *)
(*  Routines called:                                     *)
(*           power                                       *)
(*                                                       *)
(*  Global variables:                                    *)
(*           ps, M, sigma, nu, TRACE, DEBUG              *)
(*                                                       *)
(*******************************************************)


   var i : integer;
       numerator, denominator : real;

   begin

   if TRACE then
      writeln(outdebug,'>>>In computePS');

   for i := 0 to M do
      begin
      numerator := (M-i) * sigma * power(1-sigma,M-i-1)
                   * power ( 1-nu, i)
                   + i * nu * power (1-nu,i-1)
                        * power(1-sigma,M-i);
      denominator := 1 - power(1-nu,i) * power(1-sigma,M-i);
      ps[i] := numerator/denominator;
      end;

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'PS array:');
      for i := 0 to M do
         writeln(outdebug, ps[i]);
      writeln(outdebug);
      end;

   end;


(*******************************************************)
procedure computeDELTA;
(******* *****************************************)
(*                                                     *)
(*  Purpose: This routine computes the delta array, which*)
(*           is the average idle period given that Nte=i *)
(*                                                     *)
(*  Called by:                                         *)
(*           calculations                              *)
```

```
(*                                                                  *)
(*  Routines called:                                                *)
(*            power                                                 *)
(*                                                                  *)
(*  Global variables:                                               *)
(*            delta, M, nu, sigma, TRACE, DEBUG                     *)
(*                                                                  *)
(*******************************************************************)


   var i : integer;

   begin

   if TRACE then
      writeln(outdebug,'>>>In computeDELTA');

   for i := 0 to M do
       delta[i] := power(1-nu,i) * power(1-sigma,M-i);

   if DEBUG then
       begin
       writeln(outdebug);
       writeln(outdebug,'DELTA array:');
       for i := 0 to M do
          writeln(outdebug, delta[i]);
       writeln(outdebug);
       end;

   end;



(*******************************************************************)
procedure computeA;
(*******************************************************************)
(*                                                                  *)
(*  Purpose: This routine computes the A array, which               *)
(*           is the average idle period given that Nte=i            *)
(*                                                                  *)
(*  Called by:                                                      *)
(*           calculations                                           *)
(*                                                                  *)
(*  Routines called:                                                *)
(*           power                                                  *)
(*                                                                  *)
(*  Global variables:                                               *)
(*           A, M, iTmean, igamma, TRACE, DEBUG                     *)
(*                                                                  *)
(*******************************************************************)


   var i, l : integer;      (* counters                             *)
       sum1, sum2 : matrix;
       result1, result2 : matrix;
       finalsum   : real;
```

```
begin

if TRACE then
   writeln(outdebug,'>>>In computeA');

matrixzeros(sum1);
matrixzeros(sum2);

for l := 0 to iTmean do
   begin
   matrixpower ( q, l, result1);
   matrixadd ( sum1, result1, sum1);
   end;

for l := 0 to igamma do
   begin
   matrixpower ( q, l, result1);
   matrixadd ( sum2, result1, sum2);
   end;

matrixmultiply ( s, sum1, result1);
matrixmultiply ( f, sum2, result2);
matrixadd (result1, result2, sum1);

for i := 0 to M do
   begin
   finalsum := 0.0;
   for l := i to M do
       finalsum := finalsum + l*sum1[i,l];

   a[i] := finalsum;
   end;

if DEBUG then
    begin
    writeln(outdebug);
    writeln(outdebug,'A array:');
    for i := 0 to M do
       writeln(outdebug, a[i]);
    writeln(outdebug);
    end;

end;


(*************************************************************)
procedure computePI ( p : matrix; var errorflag : boolean);
(*************************************************************)
(*                                                           *)
(*  Purpose: This routine computes the pi array, which       *)
(*           represents the stationary prob. distribution*)
(*           of Nt at embedded points.                       *)
(*                                                           *)
(*  Called by:                                               *)
(*           calculations                                    *)
```

```
(*                                                                  *)
(*  Routines called:                                                *)
(*           initw, gausselimination, backsub, testpi               *)
(*                                                                  *)
(*  Global variables:                                               *)
(*           pi, M, p, TRACE, DEBUG, value                          *)
(*                                                                  *)
(********************************************************************)


  var
      w : matrix;


  (*------------------------------------------------------------*)
  procedure initw;
  (*------------------------------------------------------------*)
  (*                                                            *)
  (*  Purpose: This routine initializes the w matrix            *)
  (*           that will be used to calculate pi                *)
  (*                                                            *)
  (*  Called by:                                                *)
  (*           computepi                                        *)
  (*                                                            *)
  (*  Routines called:  ---                                     *)
  (*                                                            *)
  (*  Global variables:                                         *)
  (*           p, M, w, TRACE, DEBUG                            *)
  (*                                                            *)
  (*------------------------------------------------------------*)


     var i,j : integer;

     begin

     if TRACE then
        writeln(outdebug,'>>>>>>In initw');

     for i := 0 to M do
        for j := 0 to M do
           w[i,j] := p[j,i];

     if DEBUG then
        begin
        writeln(outdebug);
        writeln(outdebug,'W matrix in initw-#1');
        printmatrix(M+1,M+1,w);
        end;

     for i := 0 to M do
        w[i,i] := w[i,i] - 1.0;

     if DEBUG then
        begin
```

```
      writeln(outdebug);
      writeln(outdebug,'W matrix in initw-#2');
      printmatrix(M+1,M+1,w);
      end;

   for i := 0 to M do
      for j := 0 to M-1 do
         w[i,j] := w[i,j] - w[i,M];

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'W matrix in initw-#3');
      printmatrix(M+1,M+1,w);
      end;

   for i := 0 to M-1 do
      w[i,M] := -w[i,M];

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'W matrix after initw');
      printmatrix(M+1,M+1,w);
      end;

   end;

(*-------------------------------------------------------*)
procedure gausselimination (var errorflag : boolean);
(*-------------------------------------------------------*)
(*                                                       *)
(*  Purpose: This routine performs gaussian elimin-      *)
(*           ation on the w matrix.                      *)
(*                                                       *)
(*  Called by:                                           *)
(*           computePI                                   *)
(*                                                       *)
(*  Routines called:  ---                                *)
(*                                                       *)
(*  Global variables:                                    *)
(*           M, w, TRACE, DEBUG                          *)
(*                                                       *)
(*-------------------------------------------------------*)



   const
     epsilon = 0.000001;

   var
     i,j,k,istar,iflag : integer;
     awikod : real;
     rowmax,colmax,temp : real;
     ratio : real;
     d : list;
```

```
begin
iflag := 1;

for i := 0 to M-1 do
   begin
   rowmax := 0.0;

   for j := 0 to M-1 do
      if ( abs( w[i,j] ) > rowmax) then
         rowmax := abs( w[i,j] );

   if rowmax < epsilon then
      begin
      addstatus(err1);
      iflag := 0;
      rowmax := 0.0;
      end;

   d[i] := rowmax;

   end;

if (M-1) > 0 then
   begin

   for k := 0 to M-2 do
      begin
      colmax := abs( w[k,k] ) / d[k];
      istar := k;

      for i := k+1 to M-1 do
         begin
         awikod := abs( w[i,k] )/d[i];
         if awikod > colmax then
            begin
            colmax := awikod;
            istar := i;
            end
         end;

      if colmax < epsilon then
         begin
         iflag := 0;
         addstatus(err5);
         end
      else
         begin
         if (istar > k) then
            begin
            iflag := -iflag;

            temp := d[istar];
            d[k] := temp;

            for j := 0 to M do
```

```
                    begin
                    temp := w[istar,j];
                    w[istar,j] := w[k,j];
                    w[k,j] := temp;
                    end;
                end;

            for i := k+1 to M-1 do
                begin
                ratio := w[i,k]/w[k,k];
                w[i,k] := 0.0;
                for j := k+1 to M do
                    w[i,j] := w[i,j] - ratio*w[k,j];
                end;

            end; (* of else *)

        end; (* of do *)

    end; (* of if *)

if abs(w[M-1,M-1]) < epsilon then
    begin
    iflag := 0;
    addstatus(err2);
    end;

if iflag = 0 then
    errorflag := true;

if DEBUG then
    begin

    writeln(outdebug);
    writeln(outdebug,'W matrix in gausselimination:');
    printmatrix( M+1, M+1, w );

    writeln(outdebug);

    end;


end;



(*-------------------------------------------------*)
procedure backsub;
(*-------------------------------------------------*)
(*                                                 *)
(*  Purpose: This routine performs backward substit-  *)
(*           ution, now that an upper triangular    *)
(*           matrix has been found.                 *)
(*                                                 *)
(*  Called by:                                     *)
(*           computePI                             *)
```

```
(*                                                              *)
(*  Routines called: ---                                        *)
(*                                                              *)
(*  Global variables:                                           *)
(*            p, pi, w, M, TRACE, DEBUG                         *)
(*                                                              *)
(*--------------------------------------------------------------*)


    var i, k : integer;
        sum1 : real;
        b : list;

    begin

    if TRACE then
       writeln(outdebug,'>>>>>>In backsub');

    for i := 0 to M-1 do
       for k := 0 to M-1 do
             p[i,k] := w[i,k];

    for i := 0 to M-1 do
       b[i] := w[i,M];

    for k := M-1 downto 0 do
       begin
       sum1 := 0.0;
       for i := k+1 to M-1 do
          sum1 := sum1 + p[k,i] * pi[i];

       pi[k] := ( b[k] - sum1 ) / p[k,k];
       end;

    sum1 := 0.0;
    for i := 0 to M-1 do
       sum1 := sum1 + pi[i];
    pi[M] := 1.0 - sum1;

    if DEBUG then
       begin
       writeln(outdebug);
       writeln(outdebug,'PI array:');
       for i := 0 to M do
           writeln(outdebug, pi[i]);
       writeln(outdebug);
       end;


    end;

(*--------------------------------------------------------------*)
procedure testpi ( var errorflag : boolean );
(*--------------------------------------------------------------*)
(*                                                              *)
(*  Purpose: This routine tests the pi array to                 *)
```

```
(*           determine if the elements sum to one.     *)
(*           If so, errorflag is set to false; else    *)
(*           its set to true.                          *)
(*                                                     *)
(*  Called by:                                         *)
(*           computePI                                 *)
(*                                                     *)
(*  Routines called:  ---                              *)
(*                                                     *)
(*  Global variables:                                  *)
(*           pi, M, value, TRACE, DEBUG                *)
(*                                                     *)
(*-----------------------------------------------------*)


   const
       epsilon = 0.1;
   var
       i : integer;
       sum1 : real;

   begin

   if TRACE then
      writeln(outdebug,'>>>>>>In testpi');

   if (pi[M] < 0.0) then
      begin
      errorflag := true;
      addstatus(err3);
      end;

   sum1 := 0.0;
   for i := 0 to M-1 do
      sum1 := sum1 + w[M,i]*pi[i];
   sum1 := sum1 + w[M,M];

   if abs(sum1) > epsilon then
      begin
      errorflag := true;
      addstatus(err4);
      value := sum1;
      end;

   end;

(*-----------------------------------------------------*)

begin (* computePI *)

if TRACE then
   writeln(outdebug,'>>>In computePI');

initw;

if TRACE then
```

```
      writeln(outdebug,'>>>In computePI');

   gausselimination(errorflag);

   if TRACE then
      writeln(outdebug,'>>>In computePI');

   if not errorflag then
      begin
      backsub;

       if TRACE then
          writeln(outdebug,'>>>In computePI');

      testpi(errorflag);

      if TRACE then
          writeln(outdebug,'>>>In computePI');

      end;

   end;

(*************************************************)
procedure computeTHRUPUT;
(*************************************************)
(*                                               *)
(*  Purpose: This routine computes the average station-  *)
(*           ary channel thruput.                *)
(*                                               *)
(*  Called by:                                   *)
(*           calculations                        *)
(*                                               *)
(*  Routines called:  ---                        *)
(*                                               *)
(*  Global variables:                            *)
(*           pi, ps, iTmean, igamma, thruput, TRACE,     *)
(*           DEBUG                               *)
(*                                               *)
(*************************************************)


   var sumnum, sumdenom, term : real;
       i : integer;

   begin

   if TRACE then
      writeln(outdebug,'>>>In computeTHRUPUT');

   sumnum := 0.0;
   sumdenom := 0.0;

   for i := 0 to M do
      begin
      sumnum := sumnum + pi[i]*ps[i]*iTmean;
```

```
      term := 1 / ( 1 - delta[i] );
      sumdenom := sumdenom + pi[i]*( term + 1 + ps[i]*iTmean
                                     + (1 - ps[i])*igamma )
      end;

   thruput := sumnum / sumdenom;

   if DEBUG then
      begin
      writeln(outdebug);
      writeln(outdebug,'Thruput: ', thruput);
      writeln(outdebug);
      end;

   end;


(*************************************************)
procedure computeNmean;
(*************************************************)
(*                                               *)
(*  Purpose: This routine computes the average channel   *)
(*           backlog.                            *)
(*                                               *)
(*  Called by:                                   *)
(*           calculations                        *)
(*                                               *)
(*  Routines called: ---                         *)
(*                                               *)
(*  Global variables:                            *)
(*           delta, a, pi, ps, igamma, iTmean, TRACE,    *)
(*           DEBUG                               *)
(*                                               *)
(*************************************************)


   var i : integer;
       sum1, sum2 : real;
       term : real;

   begin

   if TRACE then
      writeln(outdebug,'>>>In computeNmean');

   sum1 := 0.0;
   sum2 := 0.0;

   for i := 0 to M do
      begin
      term := 1 / ( 1 - delta[i] );
      sum1 := sum1 + pi[i]*(i*term + a[i]);
      sum2 := sum2 + pi[i]*(term + 1 + ps[i]*iTmean
                  + (1-ps[i])*igamma);
      end;
```

```
    nmean := sum1 / sum2;

    if DEBUG then
       begin
       writeln(outdebug);
       writeln(outdebug,'Nmean: ', nmean);
       writeln(outdebug);
       end;


    end;


(*****************************************************)
procedure computeDandW;
(*****************************************************)
(*                                                   *)
(*  Purpose: This routine computes the average packet *)
(*           delay (normalized to Tmean) and mean waiting*)
(*           time until successful transmission.     *)
(*                                                   *)
(*  Called by:                                       *)
(*           calculations                            *)
(*                                                   *)
(*  Routines called:  ---                            *)
(*                                                   *)
(*  Global variables:                                *)
(*           nmean, thruput, delaytmean, delayslot,  *)
(*           delay, W, Wslot, iTmean, TRACE, DEBUG   *)
(*                                                   *)
(*****************************************************)


    begin

    if TRACE then
       writeln(outdebug,'>>>In computeDandW');

    delaytmean := nmean/thruput;
                                (* avg packet delay, normal-
                                   ized to Tmean (in slots) *)
    delayslot := delaytmean*iTmean;
                                (* avg packet delay, in
                                   slots                    *)

    delay := delayslot*tau;     (* avg packet delay in sec *)

    Wslot := delayslot - iTmean; (* mean waiting time, in
                                    slots                   *)

    W := Wslot * tau;           (* mean waiting time, in
                                    seconds                 *)

    if DEBUG then
       begin
       writeln(outdebug);
```

```
      writeln(outdebug,'DELAY, in seconds: ',delay);
      writeln(outdebug);
      writeln(outdebug,'W, in seconds: ',W);
      writeln(outdebug);
      end;

   end;

(*****************************************************)
procedure calculations;
(*****************************************************)
(*                                                   *)
(*  Purpose: This routine performs the necessary     *)
(*           calculations in determining certain perform-*)
(*           ance parameters concerning the network.  *)
(*                                                   *)
(*  Called by:                                       *)
(*           model                                   *)
(*                                                   *)
(*  Routines called:                                 *)
(*           Qmatrix, Jmatrix, Fmatrix, Smatrix, Pmatrix *)
(*           computePS, computeDELTA, computeA, computePI*)
(*           computeTHRUPUT, computeNmean, computeDandW  *)
(*                                                   *)
(*  Global variables:                                *)
(*           TRACE, p, errorflag                     *)
(*                                                   *)
(*****************************************************)


   begin

   if TRACE then
      writeln(outdebug,'>>>In calculations');

   Qmatrix;
   Jmatrix;
   Fmatrix;
   Smatrix;
   Pmatrix;
   computePS;
   computeDELTA;
   computeA;
   computePI(p,errorflag);

   if not errorflag
      then begin
           computeTHRUPUT;
           computeNmean;
           computeDandW;
           end
      else determinestatus;

   end;
```

APPENDIX F: Sample runs

(Example 1)

```
<{ uvacs }> model

DEFAULTS:

...Message Info:

   Message #   Length      Prob.        Slot Time(T)
   ---------   ------   ----------   ----------------
       1          916   3.000e-01    2.385416667e+01
       2          108   5.000e-01    2.812500000e+00
       3          956   5.000e-02    2.489583333e+01
       4          148   1.500e-01    3.854166667e+00

...Other Info:

   M=20                        TAU=3.0e-04
   GAMMA= 2.0e+00              IGAMMA= 2
   NU= 1.0e-02                 SIGMA= 2.0e-02
   TMEAN= 1.039e+01            ITMEAN=10
   XI= 7.812e-06               TRACE=false
   ZETA= 0.000e+00             DEBUG=false


Do you want to change any defaults? (Y/N): n


RESULTS:

...Average stationary channel throughput:   5.98731840529769e-01

...Average Packet Delay:

      Normalized to Tmean (slots):   3.54869352539207e+01
      In slots:   3.54869352539207e+02
      In seconds:   1.06460805761762e-01

...Mean Waiting Time:

      In slots:   3.44869352539207e+02
      In seconds:   1.03460805761762e-01

...Average Channel Backlog:   2.12471580593407e+01


<{ uvacs }>
```

```
<{ uvacs }> model

DEFAULTS:                                          (Example 2)

...Message Info:

   Message #   Length      Prob.        Slot Time(T)
   ---------   ------   ----------   ----------------
       1          916   3.000e-01    2.385416667e+01
       2          108   5.000e-01    2.812500000e+00
       3          956   5.000e-02    2.489583333e+01
       4          148   1.500e-01    3.854166667e+00

...Other Info:

   M=20                        TAU=3.0e-04
   GAMMA= 2.0e+00              IGAMMA= 2
   NU= 1.0e-02                 SIGMA= 2.0e-02
   TMEAN= 1.039e+01            ITMEAN=10
   XI= 7.812e-06               TRACE=false
   ZETA= 0.000e+00             DEBUG=false


Do you want to change any defaults? (Y/N): y

Slot time (tau) = 3.0e-04
...Do you want to change this? (Y/N):  n

Number of devices (M) = 20
...Do you want to change this? (Y/N):  y
>>>Enter value:  10

Bit delay for device to recognize interference, once it has
reached device ( #bits/BPS = xi ) =  1.0
...Do you want to change this? (Y/N):  1
Try again...Do you want to change this? (Y/N):  n

Period used for collision reinforcement, in seconds
(zeta) = 0.0e+00
...Do you want to change this? (Y/N):  n

Prob.  that device generates and xmits new packet in slot,
given that channel is idle (sigma) =  2.0e-02
...Do you want to change this? (Y/N):  n

Prob. that backlogged user senses channel in current slot
(nu) = 1.0e-02
...Do you want to change this? (Y/N):  n

Prob.  of packet #1 = 3.0e-01
...Do you want to change this? (Y/N):  n

Prob.  of packet #2 = 5.0e-01
...Do you want to change this? (Y/N):  n

Prob.  of packet #3 = 5.0e-02
...Do you want to change this? (Y/N):  n

Prob.  of packet #4 = 1.5e-01
...Do you want to change this? (Y/N):  n
```

```
Packet length #1 =   916 bits
...Do you want to change this? (Y/N):  n

Packet length #2 =   108 bits
...Do you want to change this? (Y/N):  n

Packet length #3 =   956 bits
...Do you want to change this? (Y/N):  n

Packet length #4 =   148 bits
...Do you want to change this? (Y/N):  n

DEBUG flag = false
...Do you want to change this? (Y/N):  n

TRACE flag = false
...Do you want to change this? (Y/N):  n

Are you satisfied with changes? (Y/N):  y

INITIALIZATIONS:

...Message Info:

  Message #   Length      Prob.          Slot Time(T)
  ---------   ------   ----------   ----------------
      1          916   3.000e-01    2.385416667e+01
      2          108   5.000e-01    2.812500000e+00
      3          956   5.000e-02    2.489583333e+01
      4          148   1.500e-01    3.854166667e+00

...Other Info:

  M=10                      TAU=3.0e-04
  GAMMA= 2.0e+00            IGAMMA= 2
  NU= 1.0e-02               SIGMA= 2.0e-02
  TMEAN= 1.039e+01          ITMEAN=10
  XI= 7.812e-06             TRACE=false
  ZETA= 0.000e+00           DEBUG=false

RESULTS:

...Average stationary channel throughput:   5.026031509982 20e-01

...Average Packet Delay:

      Normalized to Tmean (slots):   1.754308103979 09e+01
      In slots:   1.754308103979 09e+02
      In seconds:   5.262924311937 27e-02

...Mean Waiting Time:

      In slots:   1.654308103979 09e+02
      In seconds:   4.962924311937 27e-02

...Average Channel Backlog:   8.817207808816 05e+00
```