LOCAL AREA NETWORK TESTBED DESIGN

B. Lewis Barnett and Michael K. Molloy

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-85-25 October 1985

Local Area Network Testbed Design

B. Lewis Barnett
Michael K. Molloy

1. INTRODUCTION

The current explosion of local area networks for personal computers, professional workstations, and mainframe computers, has been fueled by the need for sharing the cost of specialized peripherals, large databases, and high speed correspondence among a large number of users . There are already several commercially available network offerings to choose from. However, the industry has so far standardized only three of them. There will probably be many more standards as more and more networks are developed. (An example is the manufacturing area protocol (MAP) forced through by General Motors, which is a token bus, but different than ARCNET.) It is clear that the original intention of the IEEE standards committee, to have a single local area network standard, or even a limited number of local area network standards, will not be realized.

In this project, a testbed has been developed to study different local area networks. The research is fundamentally experimental, rather than theoretical, so that incorrect assumptions, implementation problems, and unusual responses to realistic load patterns can be discovered and solved. The current research on various local area networks indicates that the current designs have various problems. Many of these problems have not been observed in the field, because the use of these networks is still in its infancy. However, as operating systems change to adapt to a networking environment and installations increase in size, the problems will become more evident. By studying current protocols and proposed protocols in an experimental environment, a second generation of protocols can be developed and thoroughly understood before the current installations observe any of the undesirable behavior seen in other research projects.

The initial performance studies for this project have been restricted to the subset of protocols supportable on the hardware available, a contention bus. These protocols for multiple access environments either minimize or resolve collisions rather than rerandomize messages as Ethernet does. The study of these protocols will form the foundation for the second generation of protocols for contention networks which will succeed the current technology, Ethernet. When and if additional hardware is obtained, this project will be expanded to include other topologies, such as rings. The design of this system is intended to be flexible enough to handle different types of local area networks.

1.1. Problem Description

In order to communicate between a large number of low activity, bursty devices, a low cost, flexible network is required. Because broadcast networks have complete logical connectivity and are easily expandable, they are an attractive architecture for such a network. The broadcast nature of such a network creates problems in the coordination of the individual users when they attempt to access the shared communication medium. Many alternative procedures exist to implement the access control required. They range from no control (Aloha) to complete predetermination (TDMA).

Under light load conditions, complete predetermination is wasteful of bandwidth and enforces unnecessarily long latency delays. In addition, frequent changes in the topology require frequent reconfigurations and the systems are highly vulnerable. In order to maintain stability and low delay in a lightly loaded network (utility less than 1/2) and to allow a flexible topology, access schemes such as Carrier Sense Multiple Access (CSMA) are needed.

The basic idea of CSMA protocols assumes the user has the ability to sense the status of the transmission medium and determine whether it is active (CS) or has two or more active users (collision detection CD) if it is active. By restricting users from attempting to transmit when the medium is active, some collisions are avoided. However, since users are physically removed from one another, collisions are still possible when users attempt to transmit at times which are separated by less than a propagation delay.

The main extensions to random access protocols are based upon different approaches on what to do after a busy period or when a collision does occur. These new protocols have algorithms to either resolve or minimize collisions depending on whether or not the medium is slotted. The basic Ethernet protocol stopped short of truly addressing this problem and instead delayed the retransmission of the messages involved in the collision for an exponentially distributed amount of time. This heuristic approach demonstrates some undesirable behavior. The delay characteristics for messages sent on the network are bimodal. Most of the time, little delay is incurred, but if the message is involved in a collision, the probability of the delay being long is high. This type of behavior leads to excessive variance in the delay (Alme79). Furthermore, messages following the colliding messages will often arrive at the destination before the messages involved in the collision. This creates an additional load on the host to host protocols to maintain the correct ordering of messages. Finally, in the case of a mixture of message sizes, small interactive messages and large file transfer messages, the delay characteristics are worse for the interactive traffic (Toba79). This is in direct contradiction to the desired behavior of priority for interactive traffic.

1.2. Previous Research

The development of local area network (LAN) technology has accelerated since the introduction of Xerox's Ethernet (Metc76), a multiple access broadcast system. Some of the basic strategies for LANS have already been standardized by the IEEE 802 committee. The standards include the 802.3 standard for CSMA/CD busses (IEEE82a), the the 802.4 standard for token busses (IEEE82b), and the 802.5 standard for token rings (IEEE82c). The existence of multiple standards for LANS is characteristic of the ongoing debate about which network protocol/architecture is best. Research continues on protocols in this area, in spite of the existence of the standards, because technology is a dynamic entity. The existence of local area networks will change the use of computer systems, which, in turn, will change the performance requirements, offered loads, and design of those networks.

One of the major problems involved in this debate is the lack of a mechanism to compare the various technologies. Not only is the actual performance of protocols unknown, but the typical loads to which a LAN will be subjected to, are also unknown. It is clear that there are definite trade-offs between token-rings, token busses and CSMA/CD busses (Bux81). What is not clear is where the trade-offs really are, and whether or not there is a way around them.

Two separate paths of research activity in the area of CSMA have evolved in the last few years. First, several variants of Ethernet which use the basic (CSMA) protocol with an exponential or linear backoff have been proposed (Fiel81, Arth84) or implemented (Witt80, Bela80, Toko77). Second, new algorithms for the resolution (Moll84, Moll83a, Mass80, Gall78, Cape79, Tows82) or minimization (Moll85, Toba79, Kies81) of collisions have been proposed and studied analytically. These alternative directions have left a gap in the experimental research on local networks. Only two projects, the Xerox experiments (Shoc80, Gons85) and the NBS experiments (Amer82, Amer85, Stok83, Toen83) have actually studied a loaded CSMA/CD system. They did not try any new protocols. This research will determine implementation issues and measure the relative performance of an actual local network running each of the protocols.

Several techniques have been suggested to deal with the problem of collisions. The first are intended for unslotted systems and minimize the synchronization effect of the Carrier Sense part of the the CSMA protocol. By deciding with some probability p to attempt transmission at the end of a current busy period, the messages that arrive during that period will have less of a chance of colliding as p decreases. Of course, as p decreases they also have less of a chance to be transmitted. Such algorithms are termed p-persistent CSMA protocols. They still assume a backoff technique to randomize the retransmission of messages which either collide or fail to attempt transmission.

Another approach has been to 'feedback' information from the arrival process to the transmission process. This technique is known as Virtual Time CSMA (Moll85). A virtual clock runs only when the medium is idle and at a rate proportional to the fraction of time the channel is busy. A transmission of a message is attempted when the virtual time reaches the value of the the real arrival time of the message. This technique minimizes the probability of a collision occurring at the end of a busy period and minimizes the number of retransmission randomizations required.

In the case of slotted transmission mediums, collisions will occur only when simultaneous rather than overlapping transmissions are attempted. In such environments it is possible to resolve collisions by using distributed tree algorithms (Cape79). Several variants of this algorithm, which have a higher average throughput and potential deadlocks in error-prone environments, have been developed and studied analytically (Cape79, Gall78, Mass80). These algorithms implement a distributed algorithm which uses knowledge of the history of the transmission medium and the slot synchronization to resolve any collision. By allowing the users involved in a collision to partition themselves by sampling probabilistically, the collision is eventually resolved and the messages originally involved are transmitted. The original Capetanakis algorithm used a binary search on the addresses of the nodes. It was extended to use a binary coin flipped by the stations to partition an arbitrary number of users. The Gallager algorithm used a binary search on the time line where the arrival time of a message was its associated value. These algorithms have the distinct advantage of finite average delays and minimum variance.

In a new approach, Enet II (Moll84), the maximum round trip delay in the 802.3 standard is used to create the feedback necessary for any of the collision resolution protocols described above. The major problem with unslotted systems has been the inability of stations to determine the number of successive idle slots in a resolution interval. By detecting an idle channel for one round trip delay, the stations can force a collision to effectively define the end of that idle step. This delay and then forced collision creates an automatic timeout so the Massey extension to the Capetanakis protocol will not deadlock.

Actual implementations of local networks operating as broadcast mediums (Metc76, Bela80, Stok80, Biba79) are less rich in their diversity. To date only variations on the backoff mechanism in 1-persistent CSMA with collision detection (Ethernet) have been implemented. One difficulty faced by most researchers has been their restriction of considering only high speed networks. This requirement forced them to build network interfaces which incorporate much of the protocol in hardware. Therefore, a single protocol must be selected early in the design phase. Currently, a single company (3Com) has built interfaces which implement only the CSMA/CD part of the protocol in hardware. The algorithm for collisions is implemented in software which requires a large percentage of the CPU during transmission.

The Xerox experiment (Shoc80) is of some interest because it did load an ethernet. However, the experiment was made from the viewpoint of the network and not the user stations. Throughputs of 95% were claimed without any notion of how long a station had to wait, on the average, before being successful. In the NBS experiment (Amer82) the behavior of specific hardware was being evaluated for the benefit of the user community. No attempt to evaluate alternative protocols was made.

1.3. Specific Research Targets

Several problems have already been identified for research on new protocols using an instrumented local area network. The problems have not been previously addressed by the research community since they are basically implementation issues.

To use any of the collision resolution protocols, some implementation mechanism to replace slotting is necessary. Although a centralized mechanism can be used to produce a short message fragment to synchronize the stations, it is vulnerable to failure. The concept developed in the new resolution techniques discussed in the work on collision resolution on the CSMA/CD bus (Moll84) provides a better mechanism for implementing the various resolution protocols.

In the use of Gallager's algorithm for collision resolution, the Poisson arrival process is used as a model. Therefore, the probability of two messages arriving during the same instant is zero. However, in reality, a system cannot measure time with infinite accuracy. Therefore, there is a potential deadlock in a real system using this protocol. The detection of the deadlock is relatively trivial. Whenever the new enabling interval reaches zero a deadlock has occurred. Resolving the deadlock is not as clear cut. Many options exist; exponential backoff, or Capetanakis hybrid are two possibilities. These options will be evaluated in this research.

The virtual time CSMA protocol (Moll85) requires an adaptive measurement of the current load to determine the virtual clock rate. The algorithm to make such a determination has not been proposed or studied. The underlying problem of clock synchronization in a distributed environment still persists. Although Molle has determined that if there is a constant shift in the clock times the algorithm still functions, what is the consequence of slowly drifting clocks? In addition, the virtual time CSMA protocol, like the p-persistent CSMA protocol, does not resolve collisions. By using a collision resolution algorithm, the infrequent collisions which can occur could be resolved.

An important point to be made about all protocols that maintain the FCFS property (such as VTCSMA) is that the performance of the system as a whole will be better in terms of buffer holding times, and overhead in higher levels of the network protocols. By guaranteeing FCFS, additional overhead in host to host protocols to handle out of sequence messages can be avoided.

1.4. Equipment Configuration

The hardware which has been acquired will allow changes to the basic ethernet protocol in software to implement the various new protocols. In order to avoid problems with biasing the results because of symmetrical load patterns, four independent load simulators will be used. If a single source transmits to a single destination, no collisions can occur, and a 100% throughput is possible.

Similarly, because of the process activity and interface hardware architectures which require significant time to change modes from receiving to transmitting, a symmetric A to B and B to A load will tend to minimize collisions. The only way to simulate a more random load (large user population) is by having at least three processors actively communicating with each other. The Xerox experiments indicated that a relatively small number (8) of machines obtained the same throughput (i.e. behaved the same under their metrics) as a large number of machines (120).

The experimental local area network testbed consists of five VAX 11/750 machines (Moll83b). Each machine has a minimal configuration of 2M bytes of main memory, 131M bytes of disk storage and a console. The gateway machine has a removable disk drive for system generation and backup. A single 3Com or DEUNA ethernet board is installed in each machine with an additional Interlan board for the gateway. The gateway machine allows the experimental network to run in isolation from the basic ethernet in the computer science department. The gateway machine allows easy access to the experimental network while providing isolation from the production ethernet environment.

The inherent speed of current hardware (10Mbps) requires a relatively large processor to generate loads in this range. The current benchmarks from BBN indicate that for large packets a single VAX with an Interlan (or 3Com) Ethernet interface can only generate 2.2 Mbps. With four processors, the network can be saturated. (With collisions throughput is less than 10 Mbps.) The measurements made at the University of Delaware (Minn83), indicate that throughputs of 875,000 Bps (7Mbps) are possible on a dedicated Unibus for small packets (250 bytes). Special techniques, such as retransmitting controller buffers without reloading, are being used to increase the load capacity of these machines.

2. DESIGN GOALS

The Local Area Network Testbed provides an environment in which experiments to discover the performance behavior of various local area network communications protocols can be conducted. Experiments should be repeatable and easy to configure. The system should allow the addition of new protocols as they are proposed. It should provide extensive monitoring capabilities, including passive monitoring of production networks and source queuing delay monitoring on testbed hosts. These points are discussed in more detail in the following sections.

2.1. Repeatability

The primary goal in the design of the Local Area Network Testbed is to create an environment where performance experiments can be run in a repeatable and reliable fashion. Many of the performance measurements on local area networks have been done on production LANs operating under real workloads (Shoc80). Much has been learned about typical network traffic in this way, but because of the variable nature of day to day workloads, these measurements were not repeatable. Even when artificial workloads were generated, the networks were not dedicated to the experiments.

2.2. Alternative Protocols

The testbed should be designed in a way that facilitates the investigation of alternative communications protocols. Many such protocols have been suggested and analyzed mathematically or simulated, but very few have actually been implemented and tested on actual networks. Repeatability is especially important when investigating the performance of alternative protocols. The design of the testbed should not be dependent on the medium or on specific interface hardware. Satisfying this condition allows the testbed software to perform equally well on bus networks and token rings.

2.3. High Level Description of Experiments

It should be possible to configure experiments quickly and easily. The process of creating an experiment should be as simple as possible, allowing the experiment to be described at a relatively high level of abstraction. The user should not need to think about the details of writing processes to generate loads or how to implement a different protocol.

2.4. Flexibility of Experiment Descriptions

The level of abstraction at which experiments are specified should also allow a high degree of flexibility in the kinds of experiments that can be created. Ideally, it should be possible to design an experiment that models any observed network behavior closely. Specifically, we would like to be able to simulate network nodes performing a wide range of functions, such as file servers, work stations, mainframes, terminals, and printers.

2.5. Observing Local Queueing Delays

There should be monitoring software on each host in the experimental network capable of measuring the queueing delays incurred by each packet due to traffic conditions and the operation of the protocol. Delay is an important characteristic of a protocol which has been estimated and simulated, but has seldom been actually measured.

2.6. Passive Monitoring Capability

It should be possible to monitor the traffic on the network passively. It is necessary to dedicate a host to this activity in order to prevent monitoring and load generating activities from interfering with each other. Such quantities as total throughput and frequency of source/destination pairs should be recorded. This capability can also be used to monitor the performance of production networks for purposes of tuning and diagnosis.

2.7. Physical Topology

It should also be possible to alter physically the network topology. Some protocols exhibit undesirable characteristics only as a result of a particular arrangement of nodes on the bus. Maintaining the ability to alter the topology of the network will allow the effects of varying propagation delays between nodes to be studied.

3. CURRENT DESIGN

In order to meet the goals discussed in section 2, three general capabilities are needed. First, there must be a way to describe the load behavior we wish to induce on the network. Second, a means of using the experiment description to produce the specified behavior on the network and to collect the data produced is needed. Third, we need tools with which to analyze the collected data.

To meet the first requirement, a program called the Experiment Configuration Package (ECP) has been written. The program allows the user to specify an experiment in terms of load generating processes residing on the various nodes of the network and the probability distributions which govern the length of and delay between packets. By specifying the behavior of the load generators in terms of distributions, we insure repeatability. (Ferr78) The ECP will be discussed in further detail in section 3.1.

Generating an experiment from a description is accomplished by a set of programs collectively referred to as the Execution Package (EXP). The EXP translates the description output by ECP into parameters for load generating processes, and handles transferring these parameter sets to the proper machines. It also controls the execution of the experiment and manages the data collection. The EXP is discussed in section 3.2.

To assist in reducing and analyzing the data collected by the EXP, an Experiment Analysis Package (EAP) is planned. Since we are interested chiefly in the delay behavior of the protocols investigated, the EAP will provide delay versus throughput and delay versus offered load plots. The capability to compare the delay characteristics of protocols under various load conditions as well as the characteristics of various protocols under identical conditions will be provided. It will also provide formatting for various statistics collected during experiments. The EAP is discussed further in section 3.3.

3.1. Experiment Configuration Package

Logically, an experiment runs on a set of nodes, each of which falls into one of two categories. There is one monitor node, and the remainder of the participating nodes are load generating nodes. The monitor node initially performs some control and synchronization tasks, and then acts as a purely passive monitor while the load generation is in progress. The monitor logs information of the packets produced on the network during an experiment, such as source and destination addresses and reception time. The load generating nodes take active part in producing traffic on the network by generating packets according to an experiment description prepared in advance. The load generating nodes also maintain local logs of packets produced and the queueing delay suffered by each packet. The Experiment Configuration package allows the user to specify the load produced by the load generation nodes during an experiment.

3.1.1. Data Structures

In order to achieve a high degree of flexibility in configuring experiments while keeping the configuration process simple, the following scheme for representing experiment descriptions was decided upon. Experiment descriptions conceptually have three parts: a packet size and/or packet interarrival distribution list, a topology, and a protocol selection. These three parts determine the behavior of the load generators and the network during an experiment; altering any of the three effectively creates a different experiment.

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

3.1.1.1. Distribution List

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

3.1.1.2. Experiment Topology

The distribution list is referenced by the topology description of an experiment. This description contains a record for each process on each node in the network. For each process, an index into the distribution list is provided for packet length generation as well as for interpacket delay generation for that process. Also included in the topology record for a process is a list of receivers for the packets produced by the process along with routing probabilities for each receiver. This scheme allows a wide range of packet generation and packet routing behaviors to be specified in a uniform way.

3.1.1.3. Protocol Selection

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

3.1.2. Files

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

3.1.2.1. Header File

The header file contains general information concerning the experiment, the distribution list, and the kernel selection. The general information includes the name of the experiment, a comment on its purpose, and several general house-keeping items such as creation and last modification dates, flags indicating the completeness of the description, and information concerning the storage of the description. There is also a count of the number of processes specified for each node, which is used to read the topology entries in the configuration file.

The header file structure is as follows:

BYTES	CONTENTS
0-2	Section complete flags
3-12	Experiment name (corresponds to file name)
13	Protocol code
14-20	Date of last modification (MMDDYY)
21-27	Date created
28	Unused
29-30	# of Processes on node 1
31-32	# of Processes on node 2
33-34	# of Processes on node 3
35-36	# of Processes on node 4
37-40	# of distributions defined
41-55	Load pattern name
56-70	Unused
71-140	Comment

DISTRIBUTION RECORDS

Exponential Distribution

1-3	Sequence #
4	Distribution code (= 1)
5-13	Value for theta (theta = mean interarrival time)
14-20	Unused

Poisson Distribution

1-3	Sequence #
4	Distribution code (= 2)
5-13	Value for lambda (arrival rate)
14-20	Unused

Geometric Distribution

1-3	Sequence #
4	Distribution code (= 3)
5-13	Value for p (probability of success)
14-20	Unused

Binomial Distribution

1-3	Sequence #
4	Distribution code (= 4)
5-13	Value for n (number of trials)
14-22	Value for p (probability of success for any trial)
23-40	Unused

General Discrete Distribution

1-3	Sequence #
4	Distribution code (= 5)
5-7	# of coordinate pairs entered
8-10	Point sequence #
11-15	x value
16-20	y value

.

.

Piecewise Continuous Distribution

1-3	Sequence #
4	Distribution code (= 6)
5-7	# of coordinate pairs entered
8-10	Point sequence #
11-15	x value
16-20	y value

.

.

.

3.1.2.2. Configuration File

The configuration file contains the topology entries for all processes to be created by the experiment. These records consist of a process number, which will be unique on the node where the process is created, the indices into the distribution list for interpacket delay and packet length, indices into a table of seeds for the random number generation associated with the two distributions, and the list of destinations specified as node/process addresses. There is also a routing probability associated with each entry in this list of destinations.

Each node has a number of process records stored in the configuration file. The number of records for each node is stored in the header file. The configuration file structure is as follows:

BYTES	CONTENTS
1-2	Process number
3-5	Interpacket distribution code (determined from list in header file)
6-8	Seed index for time generation
9-11	Packet length distribution code (determined from list in header file)

12-14	Seed index for packet length generation
15-16	Number of nodes receiving messages from this node

Each process on a node sends messages to a number of other processes in the network. The identity of these receivers is recorded in the Receiver sub-records.

1-2	Receiver #
3	Node receiver resides on
4-5	Process number of receiver
6-8	Probability that a packet goes to this receiver
9-10	Unused

3.1.2.3. Storage Structure

Since many experiments will be run using many different descriptions, it will be important to maintain a reasonable, useful storage structure for experiment descriptions and experiment log files. To accomplish this, the testbed makes use of the Unix hierarchical directory structure to store each description and the data gathered using it.

The software for configuring experiments (to be discussed in section 3.2.3) resides in a directory named LANT. From this directory, there are three subdirectories, each corresponding to a separate classification of experiments. These directories are called experiment, configuration, and protocol. At the lowest level, there exists a directory for each experiment description which contains the header and configuration files for the experiment, any dumps that have been done of the description, and log files for any experiments run using the description. These directories are multiply linked, in some cases through intermediate levels, to the experiment, configuration, and protocol directories. (fig. 1)

The experiment directory is used in two ways. First, the ECP uses the experiment directory path to the individual experiment subdirectories for creating and editing the experiment descriptions. Second, the experiment control software uses this path to access experiment descriptions and to store the log file generated when a description is used to run an experiment.

The configuration directory contains several subdirectories, each corresponding to a different load pattern.

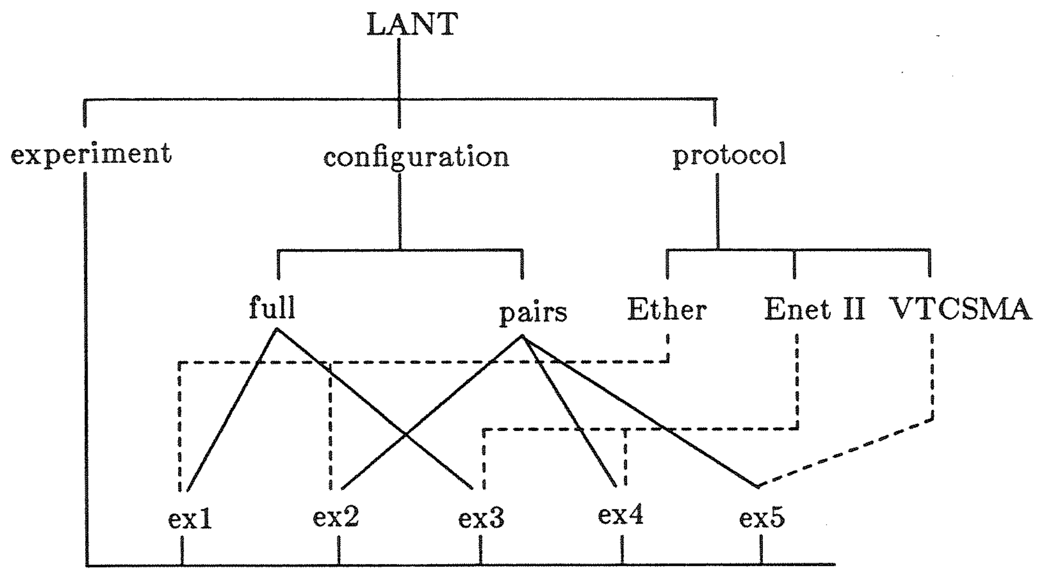

Figure 1 -- Data storage directory structure

DEFINITION: An experiment description is said to belong to some load pattern, say *l*, if it shares the same distribution list and topology as the other experiments in *l*.

Different members of a load pattern may have different protocol selections. The subdirectory for any given load pattern contains links to the individual directories of each of the experiment descriptions that belong to that load pattern. The purpose of this arrangement is to provide the data analysis programs with easy access to experiments that stand in this relationship to one another. This will facilitate comparisons of the performance of different protocols on workloads generated from the same description.

The use of the protocol directory is similar to that of the configuration directory. It contains a subdirectory for each of the currently available protocol selections. Each of these subdirectories is linked to the individual experiment directories in which that protocol is the protocol selection. Again, this allows the analysis programs to access easily all data from experiments which ran under the same protocol, allowing investigation of the effect that different workloads have on the performance of each protocol.

The consequences with respect to this storage structure of editing an experiment description are discussed in detail in the appendix.

3.1.3. Software

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

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

3.2. Execution Package

A set of programs has been written to take an experiment description and run an experiment which consists of a set of processes producing packets according to the description. An experiment is implemented in terms of a process model. The processes fall into two classifications: processes involved in generating traffic during the experiment and processes involved in creating and managing the progress of an experiment. In the following sections, these processes will be described in high-level, functional terms. Figure 2 shows the distribution of these processes among the machines of the testbed.

3.2.1. Data Structures

The data structures used by the EXP relate to the translation of an experiment description into input parameters for the various processes that will be used to generate packets during the experiment. There is one supervisor record which contains the information from the header file. Using this structure, the supervisor creates records for each of the local control processes containing the

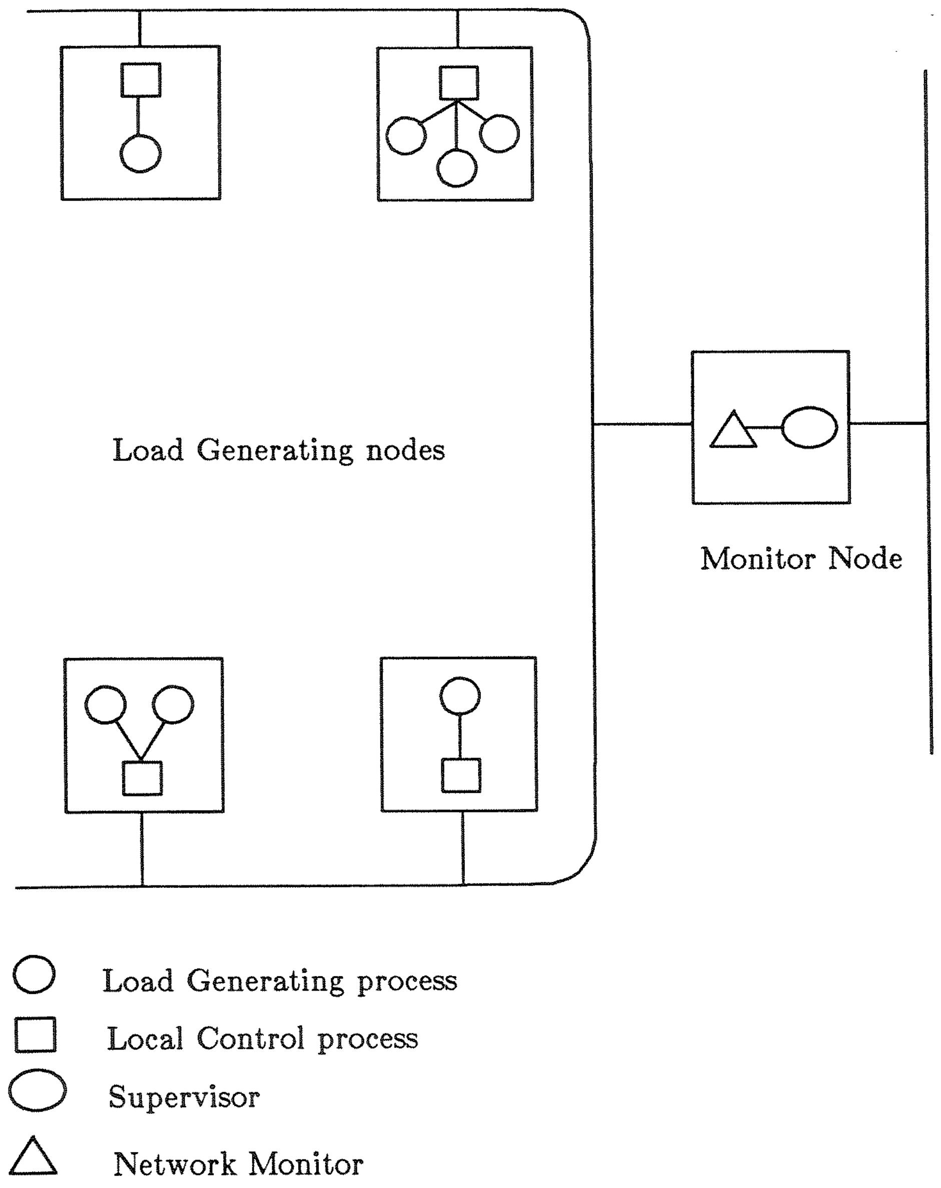

Figure 2 -- Process distribution in the experiment software

distribution list, and process records for each load generator the local controller is to create. These process records are distributed to the load generating processes upon their creation by the local controller.

3.2.2. Files

Information can be logged during an experiment in two ways. In complete logging, each message received is stored in its entirety. In summary logging, only the message header and some statistics about the message are stored. Summary logging provides sufficient information for most purposes; artificially generated packets have nothing meaningful in their data portions. The complete logging option may be useful in situations where real traffic is monitored.

Logging information is collected in two files, a header file and a message file. In summary logging, the message file contains only the packet header, while in complete logging, it also contains the body of the message. Each header file record contains the following fields:

1) timestamp - monitor node system time when message was received
2) record type - current record types are message, error, and collision
3) addresses - source and destination addresses for the packet
4) message length - number of data bytes contained in message
5) offset - pointer to packet in message file.

These fields contain all the information necessary for analysis of network traffic.

The message file contains a record corresponding to each record in the header file. The first portion of these records is the packet header, containing the following fields:

1) addresses - source and destination addresses of the packet
2) sequence number - a number unique to the source node
3) attempt number - the number of collisions suffered by this packet before this transmission
4) creation time - source node system time when packet was created.

If the logger is running in complete logging mode, this is followed by the body of the message.

3.2.3. Software

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

3.2.3.1. Load Generator

The experiment description specifies how many load generators (or sources) will be created on each load generating node, as well as the distributions they will follow to generate packets. Each load generator remains quiescent until it receives an activation signal. The load generator then generates two random numbers, according to distributions passed to it for packet length and inter-packet delay. If the experiment description contained more than one destination for the packets produced by the given source process, a third random number is generated according to a uniform distribution. This number is used as a routing probability to determine the destination of the packet being constructed. Using the generated length, a packet is built, and after the generated delay has elapsed, the packet is submitted for transmission.

3.2.3.2. Monitor

The monitor process actually controls the duration of the experiment. The experiment duration is one of initial parameters of the monitor. The monitor creates a log of all network activity while the experiment is in progress. It is possible to specify how detailed the log of a particular experiment will be. If the traffic is light, each packet can be logged in its entirety, along with its time of arrival. Otherwise, only the timestamp and packet header are retained. The contents of the log file will be discussed in greater detail below. When the monitor has recorded network activity for the specified length of time, it terminates.

3.2.3.3. Supervisor

The supervisor handles overall control of the experiment. The experimenter first chooses an experiment description and specifies a duration for the experiment. No other user intervention is necessary. The supervisor makes contact with a local control process on each of the load generating nodes. These processes can be thought of as local agents of the supervisor. For each of these local control processes, the monitor extracts from the description the pertinent information (number of load generators for that node, length and delay distributions for each load generator, etc.) and transmits it to the controller. When this activity is completed, the supervisor waits for a signal from each of the controllers. This signal indicates that the controller has received the descriptions of the load generating processes successfully and is ready to create them. When a signal has been received from each of the controllers, the supervisor sends a 'go' signal to each. At this point, the monitor process is created, and the supervisor waits for it to terminate. Upon monitor termination, the collection process (discussed below) is created. When it terminates, the experiment is completed, and the supervisor terminates.

3.2.3.4. Local Control Processes

The local controller performs duties specified by the supervisor on a load generating node. After being initiated, the controller waits to be contacted by the supervisor. Upon being contacted, the controller receives the descriptions of the load generation processes it is to create and manage. When these descriptions have been successfully received, the controller sends a signal to the supervisor indicating that it is ready to proceed. At some later time, the controller receives a 'go' signal. The controller then creates the specified number of load generation processes, then sends an activation message to each one in turn. After waiting for the specified duration of the experiment, (plus an added constant factor; the monitor actually determines when the 'experiment proper' ends) the controller terminates each of the load generators. The controller then waits to be contacted by a collection process. Upon being contacted, the controller transmits the local logs compiled during the experiment. When this transmission is completed, the controller receives an 'end' message and terminates.

3.2.3.5. Collector

The collector polls each of the local controllers. Polling is used since the network is still running an experimental protocol and may not be completely reliable. Polling eliminates the possibility of collisions. The collector receives the local logs from each control process and writes the information to storage, then sends an 'end' message to each, indicating that the experiment has successfully concluded. The collector then terminates.

3.2.3.6. Control Flow

The actions of these processes can be grouped into three phases through which an experiment run passes. These phases are initialization, load generation and data acquisition, and cleanup. The actions of the various processes during each phase are represented in table 1.

3.2.3.6.1. Initialization

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

3.2.3.6.2. Load Generation and Data Acquisition

The second phase consists of the actual generation of packets at the load generating nodes and the logging of the network activity on the monitor node. After sending the 'go' message, the supervisor creates the monitor process to handle the logging while the load generating processes on the load generating nodes are activated by their respective controllers to begin producing traffic. This phase lasts until the monitor process and the load generating processes shut down.

3.2.3.6.3. Clean Up

The last phase of an experiment run is initiated by the termination of the monitor process. This reactivates the supervisor, which then creates the

Phase	Process				
	Supervisor	Monitor	Controller	Load Gen.	Collector
Initialize	Select experiment				
	Link expt. kernel and reboot load generators				
	Send expt. description		Rcv expt. description		
	Send synch. signal		Rcv synch. signal		
			Create load generators	Start	
Generate	Send 'go'		Rcv 'go'		
	Create monitor	Start	Send 'start' to ld. gen.	Rcv 'start'	
	Promiscuous mode set				
		log packets	log delay	gen. pkts.	
	Kill monitor	Terminate	Kill ld. gen.	Terminate	
Cleanup	Create collector				Start
			Rcv query		Query controllers
			Send logs		Receive logs
			Terminate		Terminate
	Terminate				

Table 1 -- Process actions during various experiment phases

collector. The collector receives the local logs from each local controller. These logs include statistics on the number and distribution of collisions observed and on the queueing delays suffered by the packets. When this transmission is completed, the local controllers terminate. When the collector has completed its task, it terminates, allowing the supervisor to also terminate. The termination of the supervisor marks the end of an experiment run.

3.3. Experiment Analysis Package

The desired types of analyses on collected data correspond to the structure of storage linkages discussed earlier. First, corresponding to the experiment directory, we shall wish to do data reduction on single experiments. Second, corresponding to the configuration directory, we shall compare the performance of different protocols on the same traffic pattern. Third, corresponding to the protocol directory, we shall compare the performance of a protocol under different traffic conditions. The analysis package should be simple to use and should also be easily extensible in order to incorporate new analysis types as the need arises.

3.3.1. Data Structures

The data structures for the EAP handle the log file information collected by the EXP. The monitor node log contains data on the global outcome of an experiment. The particular data items which can be extracted from the monitor node log are:

1) Actual throughput observed during the experiment.
2) Packet lengths.
3) Reception delays at monitor node.
4) Source and destination addresses for all packets.

The logs maintained on each load generating node contain information about local behavior during an experiment, including:

1) Offered load.
2) Queueing delay before transmission.
3) Backoff distribution (the number of packets involved in 1, 2, 3,... collisions before successful transmission).

3.3.2. Files

The EAP will produce as output a device independent description of the appearance of the graph, histogram, or report requested. This file can be used in two ways. On a workstation such as the Sun, the file could be used as input for an interactive preview system. The file could also be processed for output on a variety of output devices such as plotters or laser printers.

3.3.3. Software

Flexibility is desirable in the analysis package, but we are more interested in some types of analysis than others. To simplify the design, those types of analyses will be emphasized, but the design, through modularity, will allow extension to additional types of analyses. The purpose of the analysis package is to reduce the data collected during experiments and to graphically compare different aspects of the reduced data.

The analysis package will operate on data at three different levels. The first level is examining data collected during a single run of an experiment. On this level, reports will consist of histograms and average values of various collected quantities and plotting of various throughput and delay curves. The second level is examining data across several runs of an experiment. Reports will consist of overall histograms, average values and throughput and delay curves. The third level is comparison of data across experiments. At this level, it will be possible to perform the comparisons among various protocols and traffic conditions discussed in section 3.1.2.3.

In the preliminary design, the analysis package works in terms of reports. The user will design a report by specifying how many series of data points will be displayed, the treatment of each series, and where the data for those series is to be found. The level at which the report is specified will determine the scope of the user's choice of data to be included in the report.

4. SUMMARY

We have presented the motivations for and design of a Local Area Network Testbed. A configuration package for creating experiment descriptions was described. Experiment descriptions consist of sets of parameters for load generating processes that define the frequency and size of packets the process will produce. Software for running experiments based on these descriptions was described, as well as a package for analyzing the data collected. This is a working document intended only to document the design of the components of the Local Area Network Testbed. Descriptions of the implementation of the design and the results of experiments run using the testbed will appear in forthcoming reports.

5. Bibliography

Alme79 Almes, G.; Lazowska, E. "The Behavior of Ethernet-Like Computer Communications Networks," *Proceedings of the 7th Symposium on Operating Systems Principles,* ACM# 534790, Asilomar CA, Dec 1979, pp. 66-81.

Amer82 Amer, P.D. "A Measurement Center for the NBS Local Area Computer Network," *IEEE Transactions on Computers,* Vol. C-31, No. 8, Aug. 1982, pp. 723-729.

Amer85 Amer, P.D.; Kumar, R.N. "Local Area Broadcast Network Measurement Part I -- Measurement Center Design and Implementation," University of Delaware Department of Computer and Information Sciences, Technical Report No. 85-3, April 1985.

Arth84 Arthurs, E.; Stuck, B.W. "A Modified Retry Policy for ETHERNET Version 1.0 Data Link Layer," *IEEE Transactions on Communications,* Vol. COM-32, No. 8, Aug 1984, pp. 977-979.

Bela80 Belanger, P.; Hankins, C.; Navindra, J. "Performance Measurements of a Local Microcomputer Network," *Proceedings of the IFIP WG 6.4 Local Area Networks, Zurich Workshop,* August 1980, to be published by North-Holland Press.

Biba79 Biba, K.; Yeh, J. "Ford Net: A Front-end Approach to Local Computer Networks," *Proceedings of Local Area Computer Network Symposium,* May 1979, pp 199-215.

Brog82 Brogonovo, F.; Fratta, L. "A Collision Resolution Algorithm for Random Access Channels with Echo," *Proceedings of INFOCOM 82,* April 1982, pp. 68-75.

Bux81 Bux, W. "Local Area Subnetworks: a Performance Comparison," *IEEE Transactions on Communications,* Vol. COM-29, No. 10, Oct. 1981, pp. 1465-1473.

Cape79 Capetanakis, "Generalized TDMA: The Multi-Access Tree Protocol," *IEEE Transactions on Communications,* Vol. COM-27, Oct. 1979, pp. 1476-1484.

Ferr78 Ferrari, D. *Computer Systems Performance Evaluation.* Prentice-Hall, Inc., 1978, pp. 66-67.

Ferr84	Ferrari, D. "On the Foundations of Artificial Workload Design," *Proceedings of SIGMETRICS 84, Aug 21-24, 1984, pp. 8-14.*
Fiel81	Fields, J.A.; Wong, J.W. "An Analysis of a Carrier Sense Multiple Access System with Collision Detection," University of Waterloo, Technical Report CCNG E-Report E-95, May 1981.
Gall78	Gallager, R. "Conflict Resolution in Random Access Broadcast Networks," *Proceedings of the AFOSR Workshop in Communication Theory and Applications,* Sept 1978, pp 74-76.
Gons85	Gonsalves, Timothy A. "Performance Characteristics of 2 Ethernets: an Experimental Study," *Proceedings of the 1985 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems,* August 1985, pp. 78 - 86.
IEEE82a	"IEEE Project 802, Local Area Network Standards" Draft IEEE Standard 802.3 CSMA/CD Access Method, Draft D, 1982
IEEE82b	"IEEE Project 802, Local Area Network Standards." Draft IEEE Standard 802.4 Token Passing Bus Access Method, Draft D, 1982.
IEEE82c	"IEEE Project 802, Local Area Network Standards." Draft IEEE Standard 802.5 Token Passing Ring Access Method, Draft D, 1982.
Kies81	Kiesel, W.M.; Kuehn, P.J. "CSMA-CD-DR: A New Multi-access Protocol for Distributed Systems, *Proceedings of NTC 81,* Dec. 1981, pp A2.4.1-A2.4.6.
Mass80	Massey, J. "Collision Resolution Algorithms and Random-Access Communication," UCLA Technical Report UCLA-ENG-8016 1980.
Metc76	Metcalfe, R.; Boggs, D. "Ethernet: Distributed Packet Switching for Local Networks," *CACM,* Vol 19, No 7, July 1976, pp 395-404.
Minn83	Minnich, N.M.; Cotton, C.J. "An Evaluation of Two Unibus Ethernet Controllers," *Proceedings of the 8th Conference on Local Computer Networks,* Oct. 17-19, 1983, pp. 29-36.
Moll83a	Molle, M. "Asynchronous Multiple Access Tree Algorithms," *Proceedings of SIGCOMM 83 Symposium on Communications Architectures and Protocols,* ACM #533830, March, 1983, pp 214-218.
Moll85	Molle, M.; Kleinrock, L. "Virtual Time CSMA: Why Two Clocks are Better than One," *IEEE Transactions on Communications,* Vol. COM-33, No. 9, (September, 1985) pp. 919 - 933.

Moll83b Molloy, M. "Experimental Evaluation of New CSMA Protocols," *Proceedings of the National Communications Forum,* Oct. 24-26, 1983, pp. 350-354

Moll84 Molloy, M. "Collision Resolution on the CSMA/CD Bus," *Proceedings of the 9th Conference on Local Computer Networks,* Oct 8-10, 1984, pp. 44-47

Shoc80 Shoch, J. F.; Hupp, J. A. "Measured performance of an Ethernet local network," *CACM,* vol. 23, Dec 1980, pp. 711-721

Stok80 Stokesberry, P.; Rosenthal, R. "The Design and Engineering of a Performance Center for a Local Network," *Proceedings of the Computer Networking Symposium,* 1980 pp 110-115

Stok83 Stokesberry, D.P. "A Characterization of Traffic on NBSNET," *Proceedings of Workshop on Performance and Evaluation of Local Area Networks,* March 1983, pp. 43 - 82.

Toba79 Tobagi, F.; Hunt, J.A. "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," *Proceedings of the Local Area Network Symposium,* May 1979

Toba80 Tobagi, F. "Multiaccess Protocols in Packet Communication Systems," *IEEE Transactions on Communications,* Vol. COM-28, No. 4, April 1980, pp 468-488

Toen83 Toense, R.E. "Performance Analysis of NBSNET," *Journal of Telecommunication Networks,* Vol. 2, No. 2, Summer 1983, pp. 177 - 186.

Toko77 Tokoro, Mario;Tamara, Kiichiro, "Acknowledging Ethernet," *Proceedings of COMPCON 77,* Sept. 1977, pp. 320-325.

Tows82 Towsley, D.;Venkatesh, G. "Window Random Access Protocols for Local Computer Networks," *IEEE Transactions on Computers,* Vol C-31, No. 8, Aug 1982, pp. 715-722.

Witt80 Wittie, L.D.; Van Tilborg, A.M. "MICROS: A Distributed Operating System for MICRONET, A Reconfigurable Network Computer," *IEEE Transactions on Computers,* Vol. C-29, No. 12, Dec 1980 pp 1133-1144.

6. Appendix: Usage and Operation

This appendix contains a discussion of the actual usage of the various programs mentioned in sections 3.1 and 3.2. These sections should be considered full fledged user manuals.

6.1. Front End

The Experimental Configuration Package (or ECP) is a front end for a set of Local Area Network performance evaluation tools. It is used to enter specifications for artificially generated network communications loads. The use of the specifications generated using the ECP is described in section 6.2 of this report.

6.1.1. The Structure of an Experiment Definition

An experiment definition has three parts: a list of distributions, a topology, or connectivity matrix, and a protocol selection. The goal of the definition is to produce complete descriptions of a set of packet producing processes. You will enter a list of distributions which will be used to generate interpacket times and packet lengths. These distributions are in turn used in the definition of the experiment's topology. For each machine in the network, you will enter the number of processes producing packets, and the characteristics of these processes. The distribution list and topology definition together constitute a load pattern. This load pattern can be used in several experiments with different communications protocols as determined by the protocol selection.

6.1.2. Distribution List

To enter a list of distributions, choose entry 1 at the main menu. (figure A1) The first prompt is for a name for the experiment to work on. If this question has been asked previously in the current session, a choice of continuing to use that experiment will be offered. If you would like to see a list of experiments that already exist, type '?' in response to this question. Experiment names may be a maximum of 10 characters long. If the experiment you named already exists, it will be checked for completeness. If the topology definition has not yet been entered, you will automatically find yourself in the Topology definition section. If both the distribution list and the topology definition are complete for the specified experiment, you will be notified and given the choice of selecting a different name, or re-using the distribution list to create a new experiment with a

```
               Experimental Configuration Package

    Your choices are:
      1) Define topology and load characteristics for an experiment
      2) Select protocol
      3) Edit an experiment definition
      4) Dump a configuration file in printable form
      5) Exit

    For defining a new experiment, steps 1 and 2 should be done in order.

    Choice # ? _
```

Figure A1 -- Main Menu

different topology.

If the name you enter is not currently in use, you will be prompted for another name to identify the load pattern for the experiment. This name will be used to set up groupings of experiments with the same load pattern but different protocol selections. You will then be prompted for a comment to identify the purpose of the experiment. After this is completed, the screen shown in figure A2 will appear. Choose one of the listed distribution types by typing the number that appears next to the distribution name. Depending on the type of distribution you choose, you will be prompted for the parameters needed. If you choose either of the user entered distributions, (distributions 5 and 6) you will be prompted for a list of x and y coordinates. This will continue until you type 'n' in response to the question 'Enter another point?' Likewise, you will continue to be prompted for distributions until you type 'n' in response to the question 'Enter another distribution?'

6.1.3. Topology Definition

To enter a topology definition, one may choose to continue into the topology definition section immediately from the load distribution section, or topology information can be entered at a later time by choosing '1' at the main menu,

```
                LOAD CHARACTERISTICS

   Time intervals between instances of packet generation and packet
lengths will be specified by distributions from the following list.
Special distributions are entered by the user.  Your choices here
form a list which will be referenced in the topology description.

Standard distributions
   1) Exponential
   2) Poisson
   3) Geometric
   4) Binomial

Special distributions
   5) General Discrete
   6) Piecewise Continuous

Choice # ? _
```

Figure A2 -- Distribution List entry screen

then entering the name of an experiment for which the distribution list has already been entered. If the second route is taken, you will go through the same procedure as discussed in section 6.1.2. for entering the experiment name. The screen for entering the topology definition is shown in figure A3. Each repetition of the questions in this screen represents one process. You will be prompted for distributions from the list discussed in section 6.1.2. for interpacket time and packet length for the packets generated by this process. You may look at a list of the distributions that you entered by typing an 'h' in response to either of these questions. You will then be prompted for the number of processes with which this process will communicate. You will then cycle through the questions for a receiver that many times. If you wish to examine the topology you are entering at any point visually, type 'v' in response to any of the questions about the receivers. The process number requested in this subsection does not

```
                TOPOLOGY DEFINITION

Node 1 How many processes will there be for this node?

Process 1
 (Type 'h' to see a recap of the distributions entered)
 Distribution for inter-packet time?
 Distribution for packet length?
 Number of processes receiving packets from this process:

   Receiver 1
   Process address for receiver:  node #
                                  process #
   Probability that a packet goes to this process:
```

Figure A3 -- Topology Definition entry screen

correspond to the sender process on the node specified; 'sources' and 'sinks' are separate entities. If a process has more than one receiver, the probabilities of the receivers should add up to one -- currently the program does not check to be sure this is true.

6.1.4. Protocol Selection

The protocol select section determines what communication protocol will be used for the experiment. When '2' is chosen at the main menu, you are once again asked for an experiment name. The codes and descriptions of the available protocols are displayed, (see figure A4) and you will be prompted for your choice. Enter the code corresponding to the protocol you wish to use for your experiment. and asked to choose the one you wish to edit.

```
                Select Procotol

   Name          Description

1) 802           Ethernet

2) VTCSMA       Virtual Time CSMA


Which protocol would you like to use for this experiment?
```

Figure A4 -- Protocol select screen

6.1.5. Edit Functions

An experiment definition consists of three parts, and these three parts are edited independently. However, editing any section has an overall effect on the experiment. Editing either the distribution list or the topology of a definition or both creates a new load pattern. It is possible to deliberately copy in place, (overwrite) but this should be done only to correct mistakes or before an experiment has actually been run using the definition. If both the distribution list and the topology of a definition are to be edited, the first section edited should be copied to create a new experiment and thus a new load pattern, while the second should be overwritten, to keep a third experiment from being created. After you have entered the name of the experiment you wish to edit, it will be checked to see which of its sections are complete and thus are eligible for editing. You will be presented with a list of these sections,

6.1.5.1. Editing the Distribution List

After you have chosen to edit the load distributions, a menu will appear offering several ways to edit the list. If you already know which member of the list you want to edit, choose '1', Edit a specific entry. You will be prompted for the number of the distribution you want to edit. This is the number that appears beside the distribution in a dump of the experiment, which will be discussed in section 6.1.7. The specified entry will be printed, along with a list of possible values for the distribution code field. The question:

[E]dit entry, [D]elete entry, [Q]uit?

will appear, and you will be prompted for input. Type the letter in brackets from the choice you want. Typing 'e' causes the cursor to move beside each field in the distribution record in turn. If you wish to change the value in that field, type the new value beside the old one. If you want to leave the old value unchanged, simply type a carriage return. Typing 'd' causes the displayed distribution to be removed from the list. Typing 'q' returns you to the edit menu without making any changes. If the distribution chosen was one of the special distributions, the question will also contain the option '[S]can points.' Typing 's' lets you see the x and y coordinates of one point at a time, and change them if you wish.

The second choice at the edit menu allows you to scan through the list of distributions one at a time. You will be asked for an entry at which to begin scanning. Enter an integer corresponding to the element of the list you wish to see first. The requested entry will be displayed, and the following question will appear:

[N]ext entry, [E]dit this entry, [Q]uit scanning?

Typing 'n' will display the next entry in the list. Typing 'e' will cause the editing process discussed above to be invoked on the current entry. Typing 'q' exits back to the edit menu.

At the distribution edit menu you may also choose to add a new distribution. If you choose this option, the data entry process for load distributions (as discussed in section 6.1.2.) will be invoked. The choices you enter will be appended to the end of the distribution list. The new entry will not immediately be available for editing. If changes need to be made to a distribution entered in this way, you must choose 4 (exit) at the edit distribution menu, then begin the edit process again by choosing 3 (edit) at the main menu. This will merge the changes entered in the last edit session with the existing distributions and make them available for editing subsequently.

6.1.5.2. Editing a Topology Description

A topology record consists of two distributions chosen from the list entered in the load distributions section and a list of processes which receive packets from the process being defined. The same editing options are available for topology descriptions as for distribution lists. In addition, there is also an option which allows you to examine the topology you have built visually .

In order to edit a particular entry, (choice 1 in the edit topology menu) you need to know what node the process is defined on, and its process number on that node. At present, valid node numbers are 1 through 4. When these have been correctly entered, the process number, the two distributions, and the number of receiver records are displayed. The following question will be displayed:

[E]dit process,[D]elete proc,[S]can receivers,[A]dd receiver,[Q]uit?

Typing 'e' causes the cursor to move to each of the editable values in turn. To leave the value as it appears, type a carriage return. To change the value, type the new value beside the old value. The number of receivers is not an editable value; it is changed automatically if receiver records are deleted or added. Typing 'd' removes the process and its list of receivers from the experiment. Typing 's' shows you each of the process's receivers in turn. This works the same way as scanning points, described in section 6.1.5.1. After the receiver is displayed, the question

[N]ext receiver,[E]dit this entry,[R]eturn to process level?

will appear at the bottom of the screen. The first two options work the same way their counterparts in point scanning do. Typing 'r' removes the receiver record from the screen and positions the cursor next to the question for the process. Typing 'a' at that question prompts you for the three questions of the receiver subrecord, and adds the resulting record to the current process's list of receivers. Typing 'q' exits to the edit topology menu.

Choosing option 2, scan process records, at the edit topology menu prompts you for a node and process number. If you just want to scan all of the processes on a node, type the node number, then a carriage return for the process number. The specified process is displayed, and the question

[N]ext process,[E]dit this process,[S]can receivers,[Q]uit?

appears. The actions associated with these options correspond to the ones discussed above.

Choosing the option to add a new process at the edit topology menu prompts you for the number of the node you want the new process to reside on. You are then taken through the data entry screen discussed in section 6.1.3 for one process. As before, this process entry will not be immediately available for editing. Exiting to the main menu, then choosing edit will remedy this.

The view topology option allows you to visually display the topology of the current experiment. A screen in built with columns of integers for each node representing the sender and receiver processes which reside there. To see which receivers a sender communicates with, use the 'h','j','k',and 'l' keys as in vi to move the cursor to the sending process you would like to examine. ('h' moves left, 'j' moves down, 'k' moves up, and 'l' moves right) Type a carriage return. The receivers of the process on which the cursor rests will appear in inverse video on the lower half of the screen. (If the terminal you are working on doesn't have inverse video capability, this effect is less than impressive.) When you are finished examining the result, type 'r' and a carriage return to repeat the process with another sender, or 'q' and a carriage return to quit and go back to the edit topology menu.

6.1.5.3. Editing Protocol Select

The protocol selection of an experiment is comprised of a single integer. Editing the protocol selection is, therefore, very simple. The current selection is displayed, along with a list of available protocols. This is the same list that is displayed in the protocol select process, discussed in section 6.1.4. You will be prompted for a choice. If you change you mind about editing the selection, simply type a carriage return and the selection will remain as is. To change the selection, choose a different protocol from the list and type its number. As in the other two edit sections, you will be presented with the possibility of overwriting the current experiment, or copying to create a new experiment. If you choose to copy, you will be prompted for a name for the new experiment. By typing '?' in response, you will be shown a list of existing experiments. When you have typed in the new name, the program will create a copy with the new protocol selection, and link it to the appropriate directories.

6.1.6. Caveats

In the load distributions and topology sections, the functions allow all distributions to be removed, or all receivers to be removed from a process. Both situations are undesirable. The first would render an experiment unusable. Great care should be taken if the distribution list is edited after the topology for an experiment has been entered to insure that the new distributions have the desired meaning in the context of the choices made in the topology description. The second situation will not be fatal, but the proper way to achieve it is to delete the process, not all of its receivers.

The overwrite option should be used only to correct mistakes or make changes in an experiment definition only if the experiment has not yet been run. Overwriting an experiment that has already been run may cause inconsistencies in the analysis phase of the experiment.

If, in creating a new experiment by editing an old one, it is necessary to edit both the load and topology sections, change the load section first and copy it to a new experiment, then change the topology section and overwrite it. Doing this in the opposite order is also permissible.

6.1.7. The Dump Utility

If you choose 4 (dump an experiment description) at the main menu, you will be prompted for an experiment name as before. When the name of an existing experiment has been entered, that experiment will be examined and a list of completed sections will be displayed. If you want a dump of the entire experiment description, type 'y' in response to the question. If you only want to see one or two sections, type a carriage return in response, then, as the cursor is moved to each section in turn, type 'y' by the ones you want to include in the dump, and carriage return by the ones you do not want to see. The dumpfile will be created with a pathname of the form

experiment/<exp name>/<exp name>.dmp

which can be viewed either on the screen using cat or vi, or sent to a printer.

6.1.8. The Protocol Selection Table

When the ECP is first invoked, the file containing the list of available protocols does not exist. To build this table, type 'A' at the main menu choice. You will be prompted for an extension, which must be no longer than 8 characters.

This will correspond to the name of the subdirectory in the protocol directory to which experiments with this protocol selection will be linked. You will then be asked for a description of this protocol. It must be no longer than 70 characters. This will serve to explain the purpose of the protocol
with the extension entered before. These will appear in the protocol select and protocol select edit sections. Typing 'A' permits one additional protocol to be entered, so to initially build this file, procedure must be done repeatedly.

6.2. Experiment Control Software

The usage of the experiment control software is extremely simple; running an experiment is completely automated. A shell script called 'start.sh' in the directory LANT is executed. The shell prompts for experiment name, duration, and a unique suffix for the log files which the experiment will generate. From this point, the activities described in section 3.2.3.6 are carried out without human intervention. The software has built in abort handling capabilities in the event one of the load generating processes fails during an experiment run.

The actual implementation of experiments will change over the lifetime of the Local Area Network Testbed project. The final implementation of the experiment control software will be discussed in a subsequent report. However, the interface between the front end and the experiment software will remain constant. The activity of this interface consists of breaking down an experiment description into the subdescriptions relevant to each of the load generating nodes and the transmission of these subdescriptions to the appropriate local control processes.

The nature of this interface is determined by the data structures used for the translation and transmission of the experiment description to its various destinations. The supervisor maintains a header containing information pertaining to the experiment as a whole. This header consists of the following information:

1) name of the experiment
2) protocol selection code (an index into the protocol table)
3) last modification date for experiment description
4) creation date for experiment description
5) number of load generating processes on each load generating node
6) number of entries in the distribution list

7) name of configuration subdirectory to which the description belongs
8) comment
9) experiment duration

This header is also stored directly in the monitor node log file generated by the experiment.

The information required by each local controller is stored and transmitted in a data structure with the following fields:

1) experiment name
2) number of load generating processes for node
3) number of entries in distribution list
4) experiment duration
5) distribution list
6) process description list

The distribution list consists of records with the following structure:

1) distribution index
2) distribution code (poisson, geometric, etc.)
3) parameters
4) point list (only for user specified distributions)

Each member of the process description list contains:

1) process identification number
2) time distribution index
4) time distribution seed index
4) packet length index
5) length distribution seed index
6) list of receivers.

Each element of the receiver list consists of a destination address and a routing probability. The seed indices are pointers into a table of seeds with desirable characteristics to be used in the generation of packet lengths and delays.

Any implementation will work from these data structures. This maintains a certain modularity in the experiment control system, and facilitates the development of successive implementations. The interface is, in a sense, machine independent; any machine dependent activity occurs in the actual implementation. This approach makes it feasible to transport the experiment control software to different machines, provided they run under the UNIX operating system.