ILMON: A UNIX NETWORK MONITORING FACILITY

Lewis Barnett and Michael K. Molloy

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

TR-86-26 November 1986

ILMON: A UNIX Network Monitoring Facility

Lewis Barnett
Michael K. Molloy

ABSTRACT

This report describes ILMON, a network monitoring facility running under the Berkeley UNIX† (4.3BSD) operating system. ILMON is composed of an instrumented version of the device driver for the Interlan NI1010 Ethernet controller [Inte82], a set of programs providing a front-end interface to the instrumented driver and display and analysis tools. ILMON allows the user to observe and analyze the traffic on an Ethernet filtered on criteria which he defines. The features of ILMON will be compared to those of the Excelan Nutcracker [Exce85] and of the network performance tools available on SUN3 workstations. The efficency of ILMON, in terms of packet loss versus throughput, will be discussed for various monitoring functions. The current implementation of ILMON runs on a DEC VAX‡ 11/750 equipped with the previously mentioned Interlan controller.

1. INTRODUCTION

ILMON is a Local Area Network monitoring facility developed in conjunction with the Local Area Network Testbed (LANT) project [Moll83][Barn85] in the Department of Computer Sciences at the University of Texas at Austin. ILMON performs promiscuous reception of network traffic and presents the user with reports on the observed traffic in various formats. Packet reception may be conditioned on many quantities, including hardware source and destination addresses, packet length, packet type, and the error status of the packet. Programs also exist for consolidating data from several monitoring sessions and for formatting the data for graphical representation.

ILMON runs on a Digital Equipment Corporation VAX 11/750, which is connected to the main UT Campus Ethernet and an experimental network with Interlan NI1010

† UNIX is a trademark of Bell Laboratories.
‡ DEC and VAX are trademarks of the Digital Equipment Corporation.

Unibus Ethernet Controllers. It uses the promiscuous reception and receive-on-error modes provided by the NI1010 to monitor network traffic. The 4.3BSD UNIX device driver for the NI1010 was augmented to provide various monitoring modes which the user configures by constructing a *filter* for the monitoring session. A filter identifies which of the available monitoring functions are to be active during a session and under what conditions a packet should be included in the collected information.

ILMON was originally conceived as the passive monitor for the Local Area Network Testbed. In that capacity it is used to collect information on the throughput and stability characteristics of a network running various experimental protocols. Other machines on the experimental network are designated as load generators and produce an artificial workload on the network during experiments. The monitoring software is, however, flexible enough to be of use in monitoring the performance of production networks for purposes of traffic and utilization analysis, and also has possible applications for network fault isolation.

The remainer of the report discusses at greater length the design, implementation, use, and possible extension of ILMON. Section two discusses the design of the system, including the method of configuring monitoring sessions using filters, the types of reports produced, and the practical limitations of the system. Section three discusses the implementation of the device driver extensions and the user interface for ILMON. Section four gives some preliminary results on the efficiency of ILMON and presents some utilization figures for the UT Campus Ethernet collected with ILMON. Section 5 discusses possible extensions to the system.

2. DESIGN

The design of ILMON centers around the notion of using filters to specify the monitoring activities which will occur during a monitoring session. The user specifies a filter with a command interpreter which is loosely based on the UNIX ifconfig utility. The filter itself is a data structure containing flags indicating whether each of the possible monitoring activities is enabled for a session. There are also fields in the structure for various comparison quantities and storage for the data collected during a session. Appendix A contains a listing of the C structure definition used for filters. Most of the capabilities of the Excelan Nutcracker are incorporated into this framework; the principal omissions are analogues to the Nutcracker capability to generate valid and erroneous packets on the network. All traffic generation in the LANT environment is done by the designated load generation machines.

Table 1: Recognized Packet Types

Type	Code	Description
ETHERTYPE_PUP	0x200	PUP protocol
ETHERTYPE_PUPAT	0x201	PUP address transfer
ETHERTYPE_EXP	0x400	LANT experiment packet
ETHERTYPE_IP	0x800	IP protocol
ETHERTYPE_ARP	0x806	Addr. resolution protocol
ETHERTYPE_X75	0x801	X.75 Internet
ETHERTYPE_NBS	0x802	NBS Internet
ETHERTYPE_ECMA	0x803	ECMA Internet
ETHERTYPE_CHAOS	0x804	Chaosnet
ETHERTYPE_X25	0x805	X25 Level 3
ETHERTYPE_NSCOM	0x807	XNS Compatibility
ETHERTYPE_SYMP	0x81C	Symbolics Private
ETHERTYPE_DRCS	0x6002	DEUNA Remote Console Server
ETHERTYPE_DNET	0x6003	DECNET
ETHERTYPE_CRVLN	0x8003	Cronus VLN
ETHERTYPE_CRDIR	0x8004	Cronus Direct
ETHERTYPE_NEST	0x8006	Nestar
ETHERTYPE_EXCL	0x8010	Excelan
ETHERTYPE_RARP	0x8035	Reverse ARP
ETHERTYPE_BR	0x9002	Bridge Status

2.1. The filter mechanism

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

2.2. Filter predicates

ILMON allows packets to be filtered by any combination of several criteria. In the current implementation, to be included packets must satisfy the conjunction of all the criteria specified in the filter used for a session. The remainder of this section discusses each of the available criteria for packet inclusion in some detail.

2.2.1. Error status

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

2.2.2. Hardware addresses

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

2.2.3. Packet length

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

2.2.4. Packet type

Packets may be filtered according to the value in their header's type field. The possible types are shown in Table 1. These types are taken from the most recent "Assigned Numbers." [Reyn85] Type information is often useful in characterizing the applications which contribute to network load.

2.3. Collected quantities

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

1) the number of errors which occurred during the session
2) the number of collision fragments observed during the session
3) the number of packets lost during the session.

These quantities are taken from the NI1010's onboard statistics registers. The total reported for number of lost packets by the interface is actually a count of the number of times one *or more* packets were lost before the successful reception of a packet, so estimates of actual packet loss using this value are not completely accurate. In particular, any loss rate above 50% will be reported as 50% -- every packet received will have the "lost" flag set. The remaining types of data collection are described in the following sections. The packet loss rate for some of the data collection modes is discussed in section 4.

2.3.1. Packet logging

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

2.3.2. Address histograms

Histograms of the number of packets and bytes originating from or intended for each hardware address on the network may be collected. Since maintaining the data structures for such a histograms incurs a large amount of processing overhead, many packets may be lost while collecting data in this mode. Though not currently implemented, the capability of collecting the same type of histograms on IP addresses is planned.

2.3.3. Packet length histogram

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

2.3.4. Packet type histogram

A histogram on the types of the packets observed may be collected. The recognized types are shown in Table 1.

2.4. Comparison to other monitors

Two other network monitoring tools are available at the University of Texas at Austin, the Excelan Nutcracker and the Traffic monitor utility from SUN. This section discusses the similarities and differences among these tools and ILMON.

2.4.1. The Excelan Nutcracker

The Nutcracker is an 8086-based workstation with an enhanced network controller and a 20 megabyte hard disk drive. It is composed of several logical subsystems, each of which is responsible for some aspect of the network monitoring task, such as filtering the input stream, or storing the desired portion of a packet to the disk. These subsystems are hierarchically arranged, and each has an "object" associated with it which specifies the behavior of a subsystem during an experiment. The user must build the objects for each subsystem interactively prior to running the experiment, in much the same way that filters are specified in ILMON. The specification, however, is at a lower level of abstraction for the Nutcracker.

The filtering performed by the Nutcracker is of a slightly different nature than that done by ILMON. The *filter subsystem* consists of twelve "receive channels," each of which can be monitored independently. Each channel has a filter object associated with it. Four of the channels handle error packets of various kinds. The filter objects for the remaining eight receive channels can be specified by the user. The filter object consists of an offset and a string of up to 128 "octet-unions," each of which can be either a constant from 0 to FF (hex), a range of values, or "don't care" which matches all octets. The filter subsystem performs an octet by octet comparison of the pattern with each packet, beginning with the octet specified by the offset. Since the Nutcracker considers everything between the preamble and the CRC checksum to be data, this mechanism allows matching to be done on both header quantities and the data field of packets. This mechanism is more powerful than the filter mechanism of ILMON because it allows packets to be filtered on the contents of the data field. However, it is more difficult to specify experiments using the Nutcracker's method since the user must keep track of where the fields of interest are located within a packet and what data translation must be done in order to successfully match the desired quantities.

The data collected is very similar on both systems. Both systems allow packet logging or tracing. ILMON logs only the Ethernet and IP headers, while the Nutcracker allows an arbitrary "slice" of the packet to be logged, up to and including the entire packet. The Nutcracker also allows the examination of runt packets and collision fragments, which are automatically filtered by the NI1010 even in receive-on-error mode. The available statistics are almost identical. In this area, ILMON has one distinct advantage: extensibility. If the user is not satisfied with the statistics provided, the data files are available to him, and he can extend the analysis package to provide whatever statistics he is interested in.

2.4.2. Monitoring tools for the SUN workstation

The software release from SUN Microsystems, Inc. accompanying their SUN3 workstations includes two network monitoring tools. The program *traffic* provides a graphical, real-time display of network behavior. The program *etherfind* allows the user to view the headers of packets whose contents have satisfied a logical expression.

Traffic

Traffic allows the specification of one or more filters which packets must satisfy before they are included in the display. Each filter checks one of the following conditions:

- source/destination of the packet, specified either by host or by network
- IP protocol family to which the packet belongs, e.g. TCP, UDP, ICMP, etc.

- length of packet within specified range.

Traffic allows more than one filter to be active at the same time. If multiple filters are defined, packets must satisfy the conjunction of all filters. The address checking is done on the IP address of the packet, not the hardware address. As previously stated, IP address checking is not currently implemented in ILMON, nor is filtering on the IP type of packets.

Traffic displays information about the behavior of the network in a number of windows. Each window contains one of the following displays:

- packet size distribution
- usage of each IP protocol type
- traffic generated by each source (top eight displayed)
- traffic intended for each destination (top eight displayed)
- overall network utilization

Each display is a time varying histogram of either packets per second or percentage of total packets. The update interval is under user control for each window; the interval may be varied from 0.1 second to 10 seconds in 0.1 second increments. Larger intervals are not supported. The network utilization is displayed as a strip chart. If filters are defined, they apply to all windows. Other than dumping a snapshot of the screen, there is no mechanism for storing the information presented.

Etherfind

Etherfind takes an expression on the various fields of a packet header as a command line argument and displays the header information of all packets which satisfy the expression. It provides no summary information about packets observed during an execution, but does allow the collected headers to be dumped to a file by redirecting the output. Such a file could be processed to collect summary information.

The expression used to filter packets is built from a set of "primaries" and logical connectives, using the UNIX system of flag names followed by values to distinguish the various primaries. The allowed primaries are:

-dst *destination*	true if the destination of the packet matches *destination*, which may be specified as either a hostname or an IP address.
-src *source*	similar to -dst.

-between *host1 host2*	true if the packet went from one of the specified hosts to the other.
-dstnet *destination*	true if the network part of the destination field matches *destination*, which may be specified as an address or a network name.
-srcnet *source*	similar to -dstnet.
-dstport *port*	true if the packet is of IP type UDP or TCP and has destination port value of *port*.
-srcport *port*	similar to -dstport.
-less *length*	true if the packet's length is less than or equal to *length*.
-greater *length*	true if the packet's length is greater than or equal to *length*.
-proto *protocol*	true if the packet is an IP packet of the type *protocol*.
-byte *byte op value*	true if byte number *byte* of the packet is in relation *op* to *value*. Legal operators are +, <, >, & and |.
-broadcast	true if the packet's destination address is the broadcast address.
-arp	true if the packet's type is ARP.
-rarp	true if the packet's type is RARP.
-ip	true if the packet's type is IP.

Primaries may be grouped using parentheses. Juxtaposed primaries are connected by a logical *and*. Primaries separated by '-o' are connected by a logical *or*. While this method of specifying expressions is fairly comprehensive, it is difficult and confusing for expressions of any complexity. For instance, checking for a particular hardware address would require six '-byte' primaries.

Etherfind parses and displays the header of each packet that satisfies the expression. IP source and destination addresses are automatically converted to hostnames. The header can optionally be dumped in hexadecimal.

2.5. Limitations

Along with the shortcomings noted in relation to other network monitoring systems, there are several practical limitations to ILMON's capabilities. ILMON is not able to log collision fragments due to the fact that the Interlan controller filters them out automatically. This feature of the controller is not programmable in the way that the filtering of

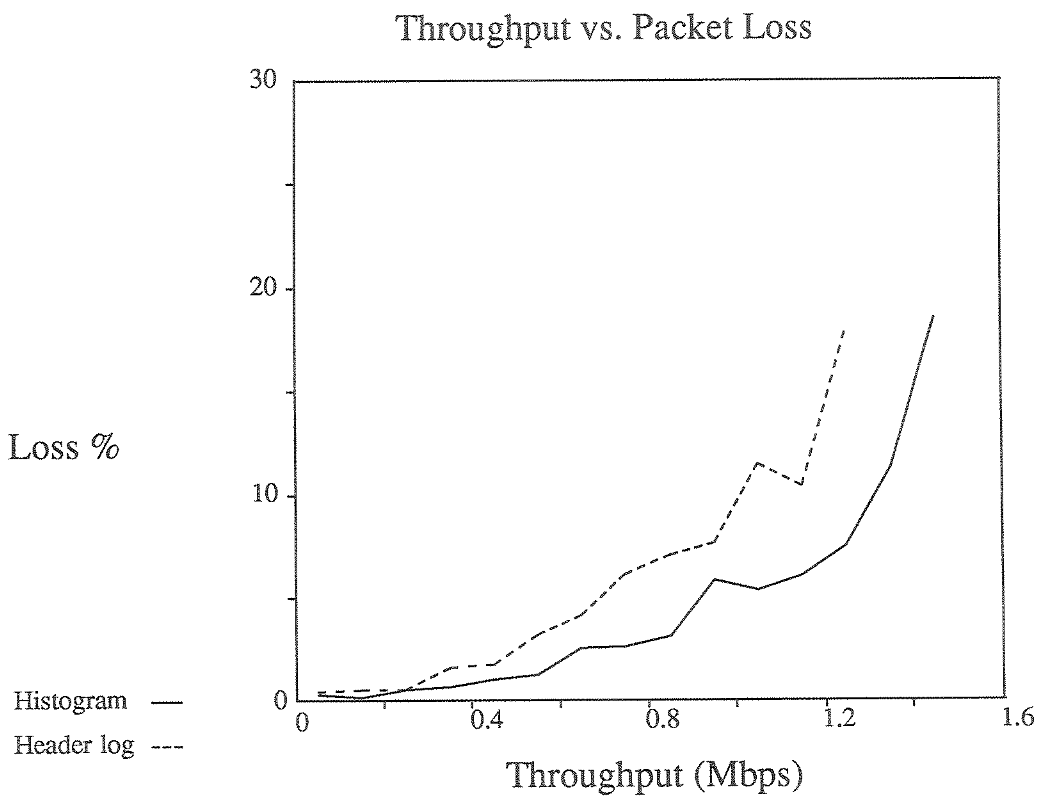

Figure 1. Efficency of packet length histogram collection and packet header logging. Averaged from observations of the UT Campus network.

error packets is. Being able to examine the fragments themselves is a desirable capability in fault diagnosis tasks. In addition to this hardware limitation, the efficiency of the ILMON software is less than perfect. The instrumentation of the controller device driver introduced extra code into the time-critical receive interrupt routine. This means than when monitoring is in progress, some packets are not processed before they are overwritten in the controller's buffers. A quantification of this packet loss appears in section 4.

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

3. IMPLEMENTATION

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

3.1. Driver instrumentation

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

3.2. User interface

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

3.2.1. ILMON

ILMON allows the user to specify, run, and save the results of monitoring sessions. The filter specification process is menu-driven, presenting the user with a list of the possible monitor functions and predicate conditions. Additionally, the user may define filter

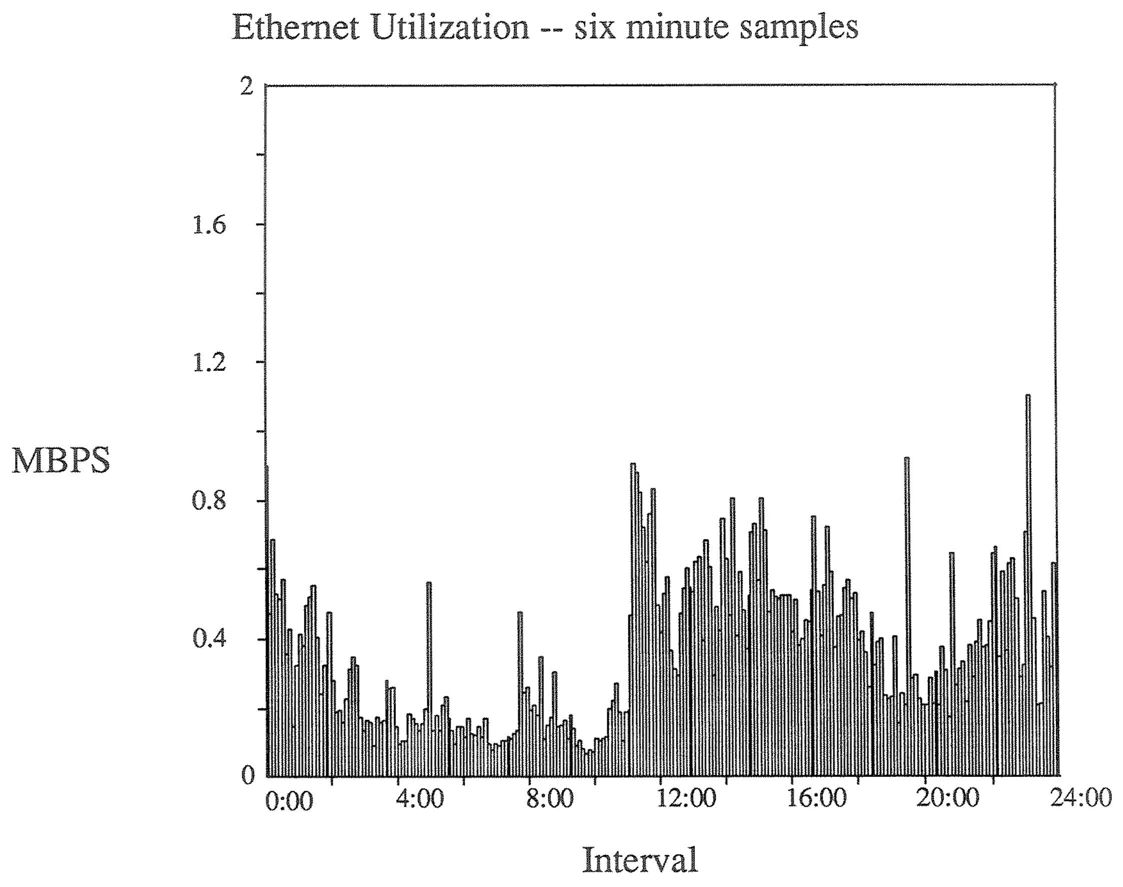

Figure 2. 24 Hour Ethernet utilization, six minute samples.

templates, which, once saved, can be reloaded and used in monitoring sessions at later times. These templates are also used in the non-interactive program *timedmon*. Several other utility functions are available in ILMON, such as retrieving the Interlan's onboard statistics, reading the interface flags from the ifnet structure, reading and setting the driver state flags from the ilsoftc structure, and reading the interface's control registers. Many of these functions are left over from an earlier program used to debug the added ioctl code in the device driver.

Table 2: Xerox vs. ILMON

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

3.2.2. Timedmon

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

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

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

3.2.3. Data examination and processing

Data analysis functions are provided by the Experiment Analysis Package (EAP), a part of the Local Area Network Testbed software. EAP provides some frequently used plots as well as the ability to specify general plots on any field of the filter's data. The standard plots available are: packet length histogram, a raw plot of network utilization data, and an averaged plot of network utilization data. Definition of the axes and labeling is under user control. As work on the testbed proceeds, further plots and data treatments will be added. The general plotting facility allows the user to choose quantities derived from filter data such as throughput, utilization, packet size, packet loss, etc. for the axes of a plot. This facility is easily extensible. EAP works on single filters, or can consolidate the data from many filters for plotting. All of the figures for this report (except for figure 4) were generated using EAP.

There are two other programs which are useful for viewing and analyzing filters. *Prfilter* simply reads in a filter and prints out the contents in human-readable form. The output of *prfilter* is suitable for piping through *lpr*, the UNIX print spooler. *Logsum* produces a summary of filters containing logged packet headers. The information produced includes the timestamps of the first and last packets logged, the number of packets logged, the throughput, the average inter-arrival interval for the logged packets, and a histogram of the inter-arrival time distribution. Figure 4 was produced by *logsum*.

4. PERFORMANCE

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

4.1. Packet loss for various monitoring functions

Figure 1 shows the percentage of packets lost during the collection of packet length histograms and packet header logging. Packet length histogram collection is the minimal monitoring task performed by ILMON, requiring only that one array element be incremented, along with the overhead incurred by any of the monitor functions. This overhead consist of several flag comparisons and at most two Ethernet address comparisons to determine whether a packet should be discarded or passed on to the higher level protocols. On the other hand, header logging requires that the packet header be copied into a log structure, and may require additional copying depending on the packet type. The data plotted in figure 1 was averaged from one minute samples, with bins of 100,000 bits and, where possible, with 20 samples per bin. No users were logged in to the monitor computer while most of the samples were collected, though the computer was operating in multi-user mode. Packet loss for histogram collection remains under 10% for throughput levels up to 1.2 Mbps. At higher throughput, more serious degradation occurs. However, since throughput levels greater than 1.2 Mbps for one minute intervals

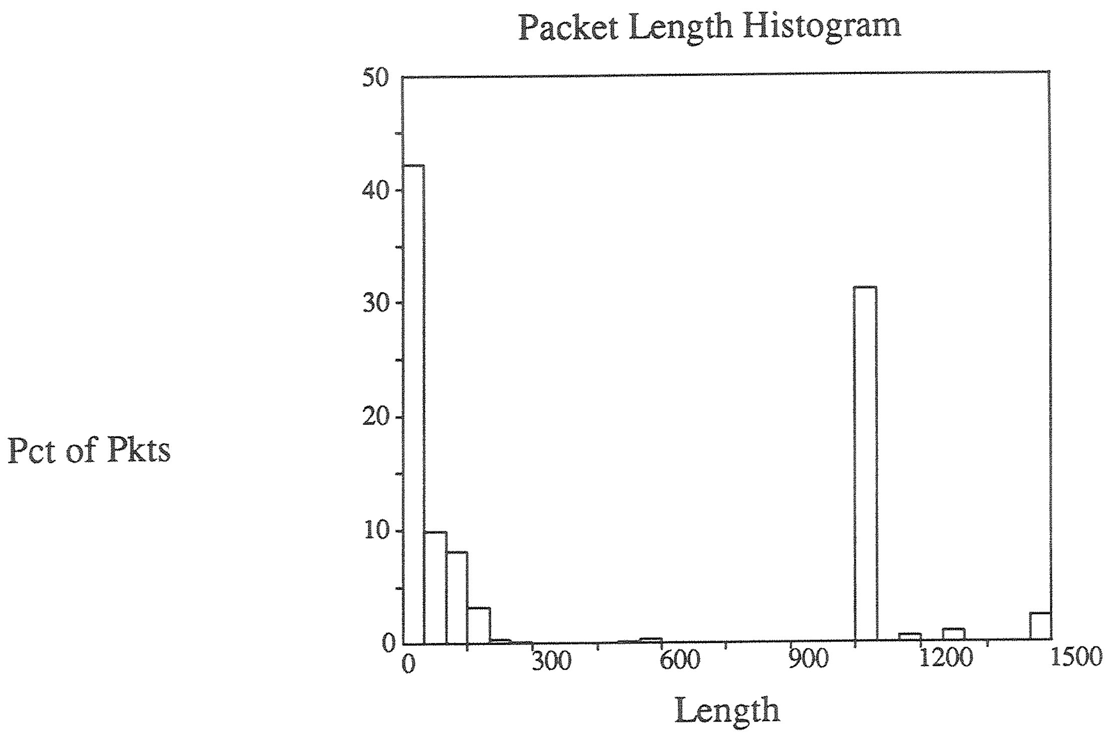

Figure 3: Packet length vs. percent of total packets.

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

4.2. Utilization of the UT Campus Ethernet

ILMON has been used to collect utilization data on the University of Texas campus Ethernet. The network connects some 80 nodes, including mainframes, minicomputers, workstations, fileservers, laser printers and terminal concentrators. Many other machines reside on subnets and contribute to the network's traffic. Several higher level protocols, such as the DOD/ARPA protocol suite, DECnet, and CHAOSNET, are used.

The network was monitored for a 24 hour period, with samples summed over six minute intervals. During this period, 8.7 million packets were transmitted, containing 3.8

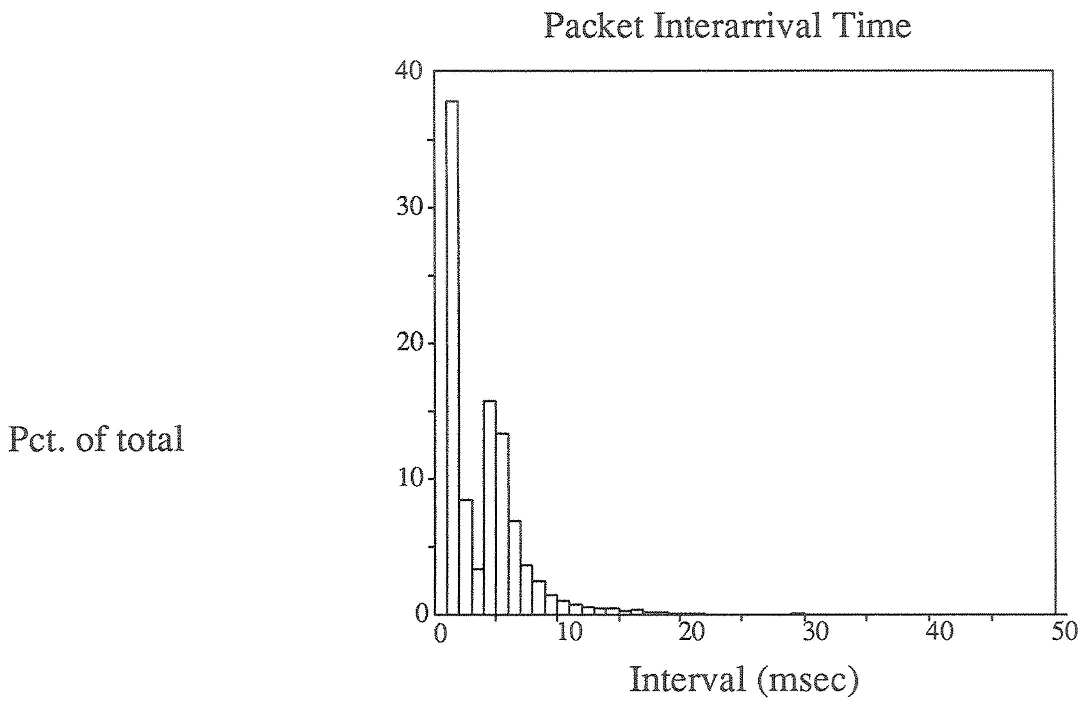

Figure 4. Inter-arrival time distribution. Taken from one thirty second packet logging sample.

billion bytes of information, not including the Ethernet headers, preambles and frame check sequences. The average throughput was 0.358 Mbps, for a utilization of 3.58%. The peak utilization was 11.03%, and the minimum utilization was 0.7%. The results of this observation are shown in figure 2. Figure 3 is a histogram of observed packet lengths. This figure shows the classic bimodal distribution, with a large percentage of small packets carrying terminal traffic, and a smaller percentage of very large packet associated with file transfers. The second group is somewhat exaggerated in the UT environment by the presence of a cluster of diskless SUN workstations and their file server on the backbone network. The SUN Network Disk (ND) protocol has a block size of 1072 bytes. Most of the traffic with packet sizes larger than 1072 are generated by various applications using the User Datagram Protocol. [Post80] Figure 4 is a histogram of the observed interval between packet receptions for the monitor node.

A comparison to a similar set of observations reported in [Shoc80] is shown in table 2. The Shoch data was collected on the Xerox PARC experimental Ethernet, which ran at a rate of 3 Mbps, while the UT Campus Ethernet is a 10 Mbps network. Though this is a less than perfect comparison, the average utilization, peak utilization, and minimum utilization figures are of interest. These quantities indicate that though the two networks were of roughly similar size at the time of the measurements, the UT network was significantly busier. Though the small average packet size for the Xerox experiments is partly an artifact of the PUP protocol in use there, the difference also indicates that subsequent protocol designs have used the medium more efficently. The observed differences between the two networks also reflect the growing dependence of a broad spectrum of computer tasks on Local Area Network communication.

5. EXTENSIONS

Though ILMON is a useful and powerful tool in its present form, several possible extensions have been identified. These extensions fall into three categories: more powerful address checking, more flexible filter predicate specification, and further development of the data reduction and presentation facilities.

5.1. Address checking

As previously noted, address checking will eventually be expanded to include checking IP addresses as well as hardware addresses. Currently, the hardware address checking works only on complete Ethernet addresses; that is, all six bytes of the address must be specified. Ethernet addresses are assigned by Xerox such that all interfaces manufactured by a vendor have addresses which fall within some range. In most cases, the first two or three bytes of the interface address will identify the vendor. (For example, addresses of Interlan controllers begin with 02.07...) Extending the address checking to work on subsets of the hardware address would allow the user to take advantage of such knowledge to check on machines using the same type of network interface hardware. In the case of IP addressing, the benefits are even greater for this type of address checking; checking only a portion of the IP address (as *traffic* does) would allow the user to isolate traffic from specific subnets connected to a backbone network.

5.2. Filter predicate specification

The flexibility of ILMON's filtering method would be greatly improved by allowing the user to specify the grouping and logical connectives for multiple filter conditions. In the current application of ILMON, the use of the conjunction of all filter conditions is adequate; however, this extension would greatly enhance ILMON's value as a general network monitoring tool. In addition, the type checking will be extended to break down

IP packets by the protocol they belong to, e.g. TCP, UDP, etc.

5.3. Data reduction and presentation

The current data reduction and presentation facilities consist of a program which collects the data from multiple filters and produces reports of averages and data files for graphs. The graphing facility allows any of several reduced quantities (throughput, packet rate, loss rate, etc.) to appear on either axis and produces a data file for a plotting package that generates plots in *pic* [Kern82] format. This is very convenient for including graphs in documents. Currently, the reduction program performs only simple smoothing on data for plotting. The addition of various types of curve fitting and extrapolation is planned.

6. SUMMARY

This paper has presented a description of ILMON, a tool for configuring and running network performance monitoring sessions, and several utility programs connected with the use of ILMON. ILMON allows a user to collect information about network traffic which satisfies a filter predicate on the form and contents of the packet. Several programs for the display and analysis of data thus collected are provided. Comparisons to the Excelan Nutcracker and the Sun3 *traffic* and *etherfind* utilities were given. The full implementation of ILMON performs well for network loads up to approximately 1.2 Mbps. At higher loads, packet loss due to monitoring overhead reaches unacceptable levels. For most normal network monitoring activity, this performance is sufficient. However, for use in the Local Area Network Testbed environment, where protocol behavior at extreme loads is of interest, further optimization will be necessary. This optimization will consist of paring down the code in the Interlan device driver to provide only those functions necessary to performance studies. Features such as address checking, packet type histograms, etc. will be eliminated.

7. REFERENCES

Barn85	B. Lewis Barnett and Michael K. Molloy, "Local Area Network Testbed Design," Technical Report TR85-25, Department of Computer Sciences, University of Texas at Austin, May 1985.
Exce85	*Nutcracker User Manual,* Excelan, Inc., San Jose, California, 1985.
Inte82	*NI1010 UNIBUS Ethernet Communications Controller User Manual,* Interlan, Inc., Chelmsford, Massachusetts, 1982.
Kern82	B. W. Kernighan, *PIC - A Graphics Language for Typesetting,* revised edition, March 1982. (online UNIX documentation)
Moll83	Michael K. Molloy, "Experimental Evaluation of New CSMA Protocols," *Proceedings of the National Communications Forum,* October 24 - 26, 1983, pp. 350 - 354.
Post80	J. Postel, "User Datagram Protocol," RFC 768, Information Sciences Institute, August 1980.
Reyn85	J. Reynolds and J. Postel, "Assigned Numbers," RFC 943, Information Sciences Institute, April 1985.

APPENDIX A: FILTER STRUCTURE DEFINITION

```
/*
 *  Structure for selective packet monitor feature in interface driver.
 */
#define HOSTMAX      32  /* # of selectively monitorable hosts */
#define NUMTYPES     19  /* Number of Ethernet packet types */
#define TREEMAX      2048/* Array for binary tree of hgram ptrs */
#define HGMAX        140 /* Maximum size of histogram */
#define LOGMAX       5000/* Maximum size of packet log */
#define QUALMASK 0x7fc00 /* Any qualifiers set? */

#ifndef NS
#define ETHERTYPE_NS 0x0600
#endif

/* Types not used by UNIX but present on our net.  (From CC documents) */
#define ETHERTYPE_CHAOS  0x804   /* Chaosnet */
#define ETHERTYPE_EXCL   0x8010  /* Excelan */
#define ETHERTYPE_RARP   0x8035  /* Reverse ARP */

struct  filter {
    int fil_flags;       /* Action flags & qualifiers - n.b. */
    struct  timeval fil_tsin;    /* Time monitor session started */
    struct  timeval fil_tsout;   /* Time monitor session ended */
    int fil_ensacnt;     /* Size of Enet source addr cklist */
    char fil_ensa[HOSTMAX][EASIZ];    /* Addresses to check for */
    int fil_endacnt;     /* Size of Enet dest addr cklist */
    char fil_enda[HOSTMAX][EASIZ];    /* Addresses to check for */
    int fil_ipsacnt;     /* Size of IP source addr cklist */
    struct  in_addr fil_ipsa[HOSTMAX];   /* IP addresses to check */
    int fil_ipdacnt;     /* Size of IP dest addr cklist */
    struct  in_addr fil_ipda[HOSTMAX];   /* IP addresses to check */
    int fil_type;        /* Packet type to monitor */
    int fil_lengt;       /* Lower bound of lengths to monitor */
    int fil_lenlt;       /* Upper bound of lengths to monitor */
    int fil_leneq;       /* Monitor only pkts of this length */
    int fil_errcnt;      /* Total errors from is_stats */
    int fil_fragcnt;     /* Total frags from is_stats */
    int fil_dropcnt;     /* Packets dropped during session */
    int fil_framecnt;        /* Total frames during session */
    int fil_lenhg[ETHERMTU+1];   /* Packet length histogram */
    struct  exp_hgram fil_typhg[NUMTYPES+1];/* Packet type histogram */
    int fil_ensatr[TREEMAX+1];   /* Binary tree pointers for ensahg */
    int fil_ensatop;     /* Next free element of ensahg */
    struct  exp_hgram fil_ensahg[HGMAX+1];    /* Hgram on Enet src addrs */
    int fil_endatr[TREEMAX+1];   /* Binary tree pointers for ensahg */
    int fil_endatop;     /* Next free element of ensahg */
    struct  exp_hgram fil_endahg[HGMAX+1];    /* Hgram on Enet dst addrs */
    int fil_ipsatr[TREEMAX+1];   /* Binary tree pointers for ensahg */
    int fil_ipsatop;     /* Next free element of ensahg */
    struct  exp_hgram fil_ipsahg[HGMAX+1];    /* Hgram on IP src addrs */
    int fil_ipdatr[TREEMAX+1];   /* Binary tree pointers for ensahg */
    int fil_ipdatop;     /* Next free element of ensahg */
    struct  exp_hgram fil_ipdahg[HGMAX+1];    /* Hgram on IP dst addrs */
    int fil_logtop;      /* Next free element in pkt log */
};

struct  shortlog {
    struct  timeval     timestamp;
    struct  il_rheader  header;
    union {
        struct  ip      iphdr;
    } in_hdr;
};

/*
 *  fil_flags bits.
 */

/* Action flags -- determines the type of monitoring to be done */
#define FLT_PKTLOG   0x00001 /* Retain log of packets filtered */
#define FLT_ENSAHG   0x00002 /* Histogram on Ethernet source addresses */
#define FLT_ENDAHG   0x00004 /* Histogram on Ethernet dest addresses */
#define FLT_IPSAHG   0x00008 /* Histogram on IP source addresses */
#define FLT_IPDAHG   0x00010 /* Histogram on IP dest addresses */
#define FLT_PLENHG   0x00020 /* Packet length histogram */
#define FLT_PTYPHG   0x00040 /* Packet type histogram */

/* Qualifiers -- determines what filtering will be done on monitored pkts */
#define FLT_ERRS 0x00100 /* Count error packets */
#define FLT_VALID    0x00200 /* Count valid packets */
#define FLT_ENSA 0x00400 /* Count pkts w/ addr in fil_ensa */
#define FLT_ENDA 0x00800 /* Count pkts w/ addr in fil_enda */
#define FLT_IPSA 0x01000 /* Count pkts w/ addr in fil_ipsa */
#define FLT_IPDA 0x02000 /* Count pkts w/ addr in fil_ipda */
#define FLT_LNGT 0x04000 /* Count pkts w/ length > fil_lengt */
#define FLT_LNLT 0x08000 /* Count pkts w/ length < fil_lenlt */
#define FLT_LNRN 0x10000 /* Lengths > fil_lengt && < fil_lenlt */
#define FLT_LNEQ 0x20000 /* Count pkts w/ length = fil_leneq */
#define FLT_PTYP 0x40000 /* Count packets of a specific type */
```