Mississippi State University
College of Engineering

Department of
Electrical and Computer Engineering
P.O. Drawer EE
Mississippi State, MS 39762

Telephone: (601) 325-3912
TELEX: 785045
FAX: (601) 325-2298

MARSHALL GRANT
IN-20-CR
115076
P.44

August 3, 1992

Walter Robinson
EB32
NASA
MSFC, AL 35812

Dear Mr. Settle:

With regards to NASA Grant NAG8–866, please find a six month progress report enclosed. If you have any questions, please feel free to contact either of us.

Sincerely,

Robert Moorhead | Wayne Smith

Dr. Robert J. Moorhead II
Principal Investigator

Dr. Wayne Smith
Co–Principal Investigator

cc: NASA Scientific & Technical Information Facility
P.O. Box 8757
Baltimore/Washington International Airport
Maryland 21240

(NASA-CR-190619) MONTHLY PROGRESS REPORT (Mississippi State Univ.) 44 p

N92-34139

Unclas

G3/20 0115076

ABSTRACT

This report is the mid–year report intended for the design concepts for the communication network for the Advanced Solid Rocket Motor (ASRM) facility being built at Yellow Creek near Iuka, MS.

The overall network is to include heterogeneous computers, to use various protocols, and to have different bandwidths. Performance consideration must be given to the potential network applications in the network environment. The performance evalution of X window applications has been given the major emphasis in this report. A simulation study using Bones will be included later.

This mid–year report has three parts: Part I is an investigation of X window traffic using TCP/IP over Ethernet networks; part II is a survey study of performance concepts of X window applications with Macintosh computers; and the last part is a tutorial on DECnet protocols. The results of this report should be useful in the design and operation of the ASRM communication network.

I. X WINDOW TRAFFIC

Introduction

About X Window system

The X window system, or just X for short, is a combination of several software components working together to provide a high performance graphical interface to users. In the X environment, users do their jobs on network–based and bit–mapped windows that are controlled and managed by a terminal server. X supports multiple windows so that it provides users with a multitask application environment. Users may open a window for each interactive applications. The X components are many things. From the user's point of view, they include a window server, a window manager, a display manager (or none), and a collection of application programs (called clients). To the X programmer, they also contain a communication protocol and a structured library that is the lowest level routines with access to the X protocol. Programmers use this library to build user interfaces.

The X window system is well described by a *client–server* model with a communication protocol between the client and the server. The *server* is a software program residing at the terminal. It takes user inputs (from the keyboard, mouse, etc.) and sends them to relevant programs (*clients*). Then it receives the client display requests and does the actual drawing on the screen. The *client* is the application program the user runs. It may run locally if the terminal is a workstation, or run remotely at a mainframe with results being displayed at simple terminals. The X protocol controls the various client–server interactions supported by X. It is defined as a set of primitives: *request, event, reply,* and *error.* The request is an order sent from a client to the server as a response to a user input/action to request the server to make some changes to the graphics display, e.g. creating/closing a window. The client may also send a request to the server for some information from the server. In this case, no drawing action will be taken. The event, sent from the server to a client, is used to indicate a device action has been initiated by the user. The difference from conventional client–server relationships is that the server gets inputs from users then produces relevant events to inform clients. A reply is used by the server to send some information a client requested. A client may request and retain the server information when being created so that the amount of interaction traffic is reduced. The server sends error messages to the client to report errors.

History, Standards, and Application Development

X was originally developed at the Massachusetts Institute of Technology with the help of various manufacturers in 1984. The first commercial implementation of X was introduced by DEC, running on the VAXstation under the Ultrix operating system. The latest version of X is the Version 11 (X11). It is a joint effort by major computer vendors. The first release of X11 (named X11R1) became available in 1987. Then came the MIT X Consortium, a large and influential collection of software and hardware companies, to keep the X open to all. The X Consortium conducts the current development of the X window system. The latest version is X11R5, released in Dec. 1991. However, X11R4 is still the most widely used. The X window system has been widely accepted as a *de facto* standard for distributed window management.

The X11 standard consists of a C programming library Xlib and an inter–client communication protocol (often called the X11 protocol). The Xlib is a large collection of C language routines which provide a lot of basic windowing functionality. It is the lowest programming interface to the underlying X protocol. Since X was designed to be "policy free", it does not specify a standard user interface. It is up to the X programmer to write his/her favorite user interface. X allows any style of user interface by providing a very flexible set of routines in Xlib. The X11 protocol is the core of the X window standard. All the communications between the client and server are through this protocol. To implement a X system is to implement the protocol. It is a bidirectional asynchronous stream–oriented protocol. It may operate on top of any network layer as long as the underlying network layer provides a sequenced and unduplicated byte delivery service. If the client and server are on the same machine (workstation style), the protocol is typically supported by an interprocess communication mechanism. Otherwise, a network connection is assumed. With asynchronous communication, there is no explicit acknowledgement by the client/server. The client/server process does not have to wait for the consequences of previous packets, but assumes reliable receipt and processing. This approach speeds up the system operation and improves the network performance. Another improvement to the network efficiency of the X protocol is its ability to group a number of requests into a single packet before sending them over the lower network layer. This can greatly reduce the network overhead.

The X11 standard is the heart of the X window system. It was designed to be terminal device independent and network transparent. X developers are encouraged to develop window systems most suited to users interests. They do not have to have the knowledge of the underlying network that is to support the X protocol. The display and the applications can be running on different machines over a network. This distribution ability allows X users to access almost all network resources and to take advantage of different platform capabilities via the network.

The only device dependent part in X system is the server. Running at the terminal (or workstation), the window server takes display requests issued by clients through the X protocol and generates the graphic output on the screen. Platform vendors are free to optimize their server software for their hardware.

Although Xlib is a basic interface for X programmers to build up application programs, it does not provide programming efficiency. To create a simple display object will involve a number of Xlib routines. This can be analog to programming with assembly language. To allow easy application creation, X vendors put extensive effort into the development of toolkits, e.g., MIT 's X Toolkit Intrinsic (also known as Xt Intrinsic) and OSF's Motif. Motif is even built upon the Xt Intrinsic. These toolkits are the high–level languages of the X window system. A general X window system structure is shown in Figure 1.

Another thing that needs to be pointed out is that X is intended to be operating system independent. It only requires a reliable data communication service between the server and clients. However, some network protocols reside in the operating system kernel, e.g., TCP/IP in the BSD UNIX operating system and the DECnet protocol with DEC's VAX/VMS machines. Certain factors related to the operating system may need be taken into account for X implementation. But as long as there

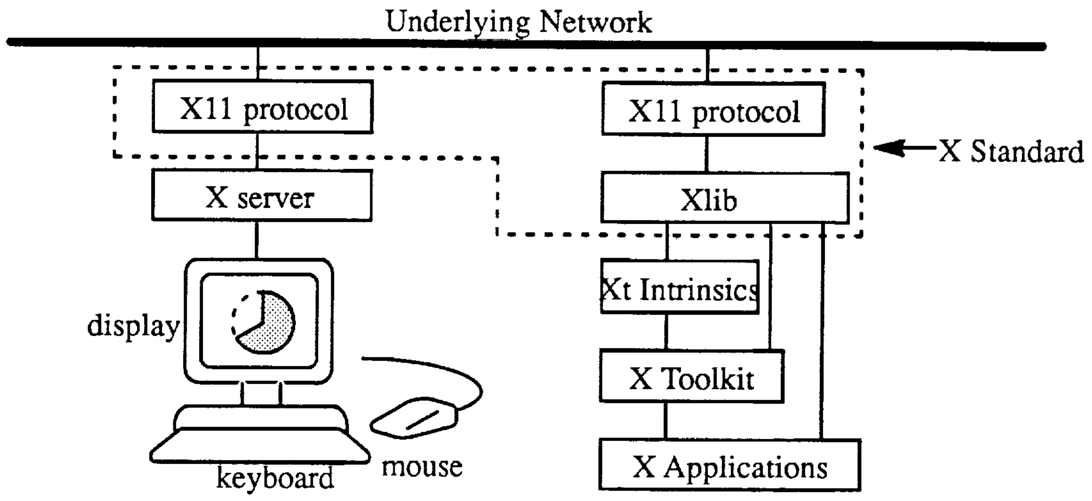


Figure 1

exists a data communication channel between the server and clients, the server can take the request and produce output for the client no matter what operating system the client is running on.

Some special client programs

There are two special client programs: the window manager and the display manager.

The standard release of the X window manager from MIT is called *twm*. Several other window managers, such as *mwm* (the Motif window manager from OSF), *awm* (Ardent window manager), *rtl* (tilted window manager, developed at Siemens Research and Technology Laboratories, RTL), and *olwm* (the OPENLOOK window manager from AT&T and Sun) are also widely used. The window manager takes care of the layout of the windows. It allows users to move and resize windows without client interactions. The reason for having the X window manager be a client program separated from the X server is to let users be able to choose the window manager they like, so that the design of the graphical user interface is freed from the manufacturer–selected graphics system. Another important functionality of a window manager is to provide a means for inter–client communications. A typical example is the *cut* and *paste* procedure between two windows on a display. The inter–client communication of X is directed by the ICCCM (Inter–Client Communications Conventions Manual) [3, 6].

The display manager is a client program designed to start the X server automatically and to keep it running. The *xdm* is a X display manager released from MIT. It is started by the root, not the user, at the system startup. When first run, *xdm* reads in some configuration parameters from a file named *xdm–config* in the xdm directory. This file indicates which servers *xdm* will manage. After reading the list of servers, *xdm* starts up these servers. For remote servers, *xdm* opens network connections. Then the display manager emulates the *getty* and *login* procedure on the displays, prompting for a user's name and password. When a user enters a name and password, *xdm* checks them and, if correct, executes initialization of a window application, creating the selected clients and a window manager defined in the file named .Xsession in the user's home directory. The file .Xsession is similar to .xinitrc. When the user logs out, *xdm* will destroys all windows by running a cleanup script called .Xreset and then goes back to the login prompt. The big difference when using an X display manager is that the X server is kept alive when the user logs off, waiting for a connection. In fact, when *xdm* is started up at system initialization, it brings up all the servers it manages. This is the typical situation of implementing X windows on X terminals with all clients running on a remote host.

Implementations of X

There are three ways to implement the X window system [7].

The first is to use workstations. The X server and all clients run locally. There is no extra network traffic introduced by X windows. If a distribution of load and high processing speed are needed, workstations are the choice.

The second way is to implement X on PCs running DOS or MS Windows. This requires adding PC X server and network protocol software (e.g. TCP/IP) to PCs. Client programs run remotely on some host server. The advantage of using PCs is to make full use of already installed PCs which still can run DOS or MS Windows applications. The PC hardware needs at least 4M bytes of RAM, an 80386SX or faster processor, and higher resolution graphics monitor/boards. High–performance graphics controller boards based on graphics processors (e.g., Texas Instruments TMS34010/20 products) can alleviate the PC CPU of running the server software.

The third implementation of X is by using X terminals. The strength of X terminals is that they are designed to run X windows. They have powerful processors and high–resolution graphics displays, and they are less expensive than workstations. The X terminals only run X servers and network protocol software. But the centralized host which runs all X applications needs lots of memory and processing power. One X terminal may require up to 1 MIPS of processing power, 1M byte of main memory, and 20M disk space on the host. In addition, each X terminal will produce an X traffic stream and load the underlying network. Therefore extra care should be taken during the design period. Figure 2 illustrates the three network implementations.

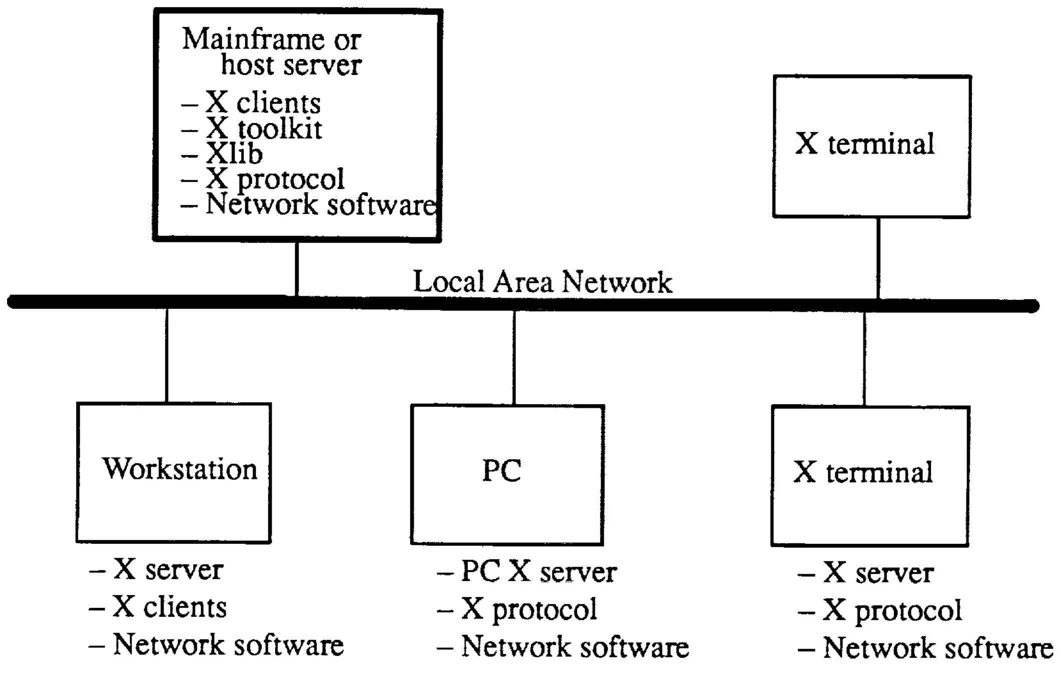


Figure 2 Implementations of X over LAN

In summary, the X window system provides a high–resolution, multitasking, graphics interface to end users. At the same time it imposes extra processing power requirements on the terminals and hosts, and data traffic on local area networks.

The following part of this report is divided into two sections. The first section classifies generally the X traffic flows over a local area network for various X implementations. The second section is an investigation of X traffic over TCP/IP and Ethernet.

Traffic Flows: Workstation vs. X Terminal

The X window system is a distributed system. Applications can be run at any node on a network provided that there exists a connection path between the application and the server. On the other hand, X supports a high–resolution, bit–mapped, multi–window interface to the end users. So it will produce extra data traffic compared with the conventional text–based, single screen terminal applications. The X protocol was developed for the data exchanges between the client and server. As previously stated, it has four primitives: *request, reply, event,* and *error.*

The *request* is the only primitive generated by the client and sent to the server. It may carry either an order to the server to draw, to change colors, etc. in a window, or an inquiry for some window information. Every *request* consists of a 4–byte header followed by optional additional data bytes. The maximum length of a request is $2^{18} - 1$ bytes. The server sends a *reply* to the client to reply to an information inquiry contained in a *request.* Each *reply* includes a 4–byte length field followed by none or more additional data bytes. The maximum length of a *reply* is $2^{32} - 1$ bytes. Both the *event* and *error* are 4–byte long, sent by the server. An *event* is used to indicate a device action taken by the user. An *error* is used to report errors.

Under the X protocol is the lower layers of the network protocol function, typically UNIX TCP/IP with Ethernet, or DEC's DECnet protocol. These lower layer protocols put additional overhead on each of X protocol data unit. The X protocol data unit is the data unit delivered as a packet by the underlying protocol. Since the X protocol tends to group several primitives together for transmission to reduce the overhead traffic, one X protocol data unit could comprise more than just one primitive. During a window application, each client will communicate with the server. If clients are distributed at various nodes on a network, there will be a network connection for each of them, as shown in Figure 3.

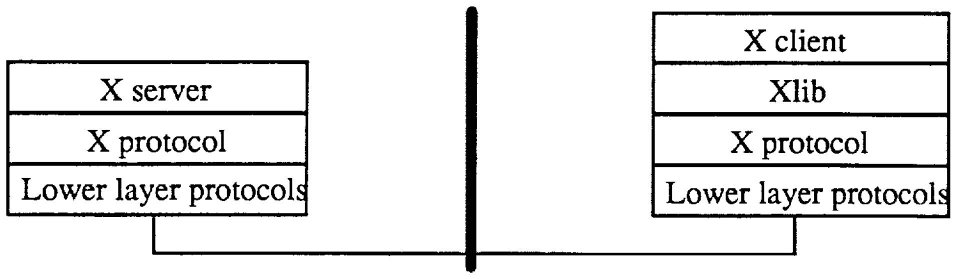


Figure 3. X client–server communication pair

There are several aspects of X traffic composition. The total X traffic over the network is dependent on the distribution of windowing functions. For a workstation version, the traffic may all be transmitted internally by the inter–process communication mechanism. While with an X terminal version, the network must accommodate all client–server data traffic. It is also application dependent. This includes how many terminals are running X windows and what kinds of tasks the users carry out under the X windows. For instance, the end users may be working in working cells equipped with X terminals to interact with a supercomputer running various applications. The end users only execute commands displayed on a selection menu and read the related information in the window. This centralized control does not need much processing power at the terminals, but depends on the speed of the host machine, the host machine interface, and the reliability of the network throughput.

Although X is intended to be network independent and encourages users to develop routines to port the X protocol to the network protocols they want to use, the TCP/IP protocol suite and the DECnet protocol suite are the most widely used. In this report we present an investigation of the actual X protocol traffic and the corresponding data traffic over Ethernet in some special cases. In our study, we assumed a number of identical terminals operating with a centralized host over an Ethernet through X windows and similar traffic statistics, so that the total amount of X protocol traffic on the network is simply the linear combination of traffic of all terminals. The investigation of the actual statistical behavior of aggregated traffic could lead to a further simulation study. We also assumed that each terminal has only one client–server connection active, since the user can only work within one window at a time.

X over Ethernet with TCP/IP

Measurement Configuration

In the study, a client–server connection was made over a 10–Mbits/s Ethernet LAN. Under the X protocol and on top of the Ethernet was the TCP/IP protocol suite. The layered structure is depicted in Figure 4 below. Both the machines running the server and client application were Sun SPARCstation 2. The X server was the *X11/NeWS* which supports both the X11 and NeWS protocols. The window manager was the *olwm*, the standard window manager from Sun OpenWindows product. We compared *olwm* with the M.I.T.'s standard window manager *twm*. They showed minimal difference in terms of X protocol data traffic.

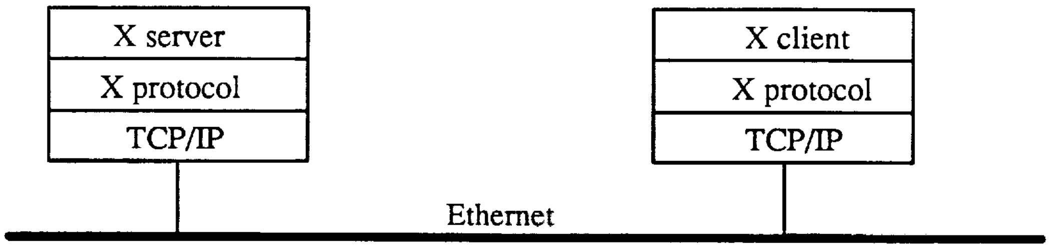


Figure 4

Traffic Monitors

The traffic monitor tools employed included a *perfmeter* and an *xscope*. The *perfmeter* measured the Ethernet packets to and from the server. It recorded the number of packets in each second during the client–server transaction. The *perfmeter* was not able to display the timed characteristics, such as packet delay. But since the measurement took place when the Ethernet LAN was almost idle, the network delay was negligible. The *xscope* utility was a useful program in monitoring X protocol traffic. It sat between the client and server, working as a relay station and at the same time observing and recording all protocol requests, events, error reports, and replies into an output file. Each entry in the output is tagged with a direction, a connection number and a time of day. The data extraction was done by a separate program. *Xscope* is an independent process. It may run on the same machine as the server, or as the client, or on a different host. It makes use of explicit TCP ports to connect the client and server. In our measurement, we setup the *xscope* at the server host and let the *xscope* and *perfmeter* monitor the X traffic in a synchronous manner. Thus the two traffic meters collected the same stream of traffic at two different levels. One displayed the traffic generated directly by the application programs, and the other showed the actual traffic on the network. The statistical differential of the two reflected the overhead of lower layer protocols (TCP/IP and Ethernet).

Application Load

We ran three different client applications.

The first was *xterm*, with which we generated two traffic patterns. One is the interactive UNIX command operations, such as listing directories, changing directories, copying files, cating files, and so on. The other traffic pattern was generated by continuously moving the pointer (mouse) into and out of the *xterm* window (named *xterm2* to distinguish). The second application was running *plaid*, an image pattern generation program. It kept sending drawing requests to the server to display the generated plaid patterns. It was an intensive load we used to investigate the extreme case. The last application was *ileaf5*, a desktop publishing software package from Interleaf, Inc.. This scenario included starting the *ileaf5*, opening a file, page scrolling in the file, closing the file, and closing *ileaf5*.

Measured Data

The data measured with the *xscope* and *perfmeter* are given in Tables 1–3. All entries are measured in bytes unless specially indicated.

Table 1. overall statistics

applications	*xterm*	*xterm2*	*plaid*	*ileaf5*
run time (sec.)	251.23	149.56	148.04	239.61
total bytes	1148533	51018	4490780	820129
total units	23919	1748	48822	17097
total X blocks	1237	1311	2287	2175
total pkts	5420	3342	27580	6569
mean pkts/sec.	21.11	21.58	179.12	25.67
max pkts/sec.	72	42	298	103

Table 2. clients statistics

applications	*xterm*	*xterm2*	*plaid*	*ileaf5*
total bytes sent	1130173	19010	4489776	755345
percentage (%)	98.40	37.26	99.98	92.10
total units sent	23503	779	48814	15232
percentage (%)	98.26	44.57	99.98	89.19
total blocks sent	835	471	2280	1105
percentage (%)	67.50	35.93	99.69	50.81
block size				
mean	1353.5	40.36	1969.2	683.57
std. dev.	928.09	32.19	260.77	855.91
maximum	2048	404	2048	2048
minimum	4	12	12	4

Table 3. server statistics

applications	*xterm*	*xterm2*	*plaid*	*ileaf5*
total bytes sent	18360	32008	1004	64784
percentage (%)	1.60	62.74	0.02	7.90
total units sent	416	969	8	1847
percentage (%)	1.74	55.43	0.02	10.81
total blocks sent	402	840	7	1070
percentage (%)	32.50	64.07	0.31	49.19
block size				
mean	45.67	38.11	143.43	60.55
std. dev.	143.35	23.63	152.94	84.00
maximum	2.32	476	476	2032
minimum	32	32	32	32

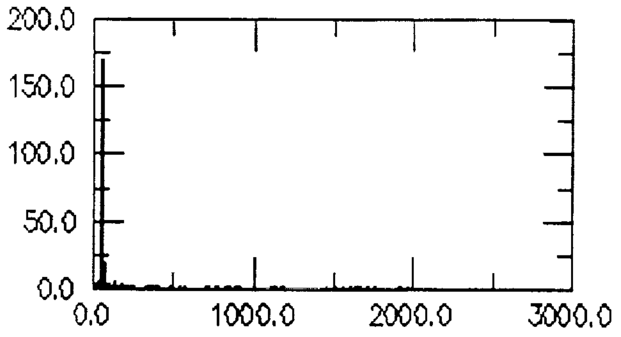
The units are the four X protocol primitives: request, reply, event, and error report. The X block (or just block) is defined as the data block the X protocol passes to the lower layer protocol (TCP in our case) for delivery. Since the X protocol works in an asynchronous manner and tries to send units in aggregation, one X data block may contains several units.

Data Analysis

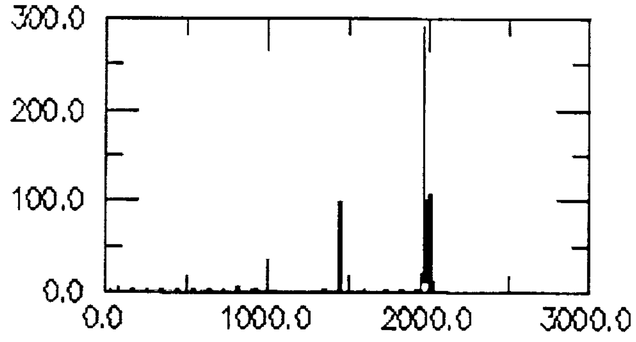
The traffic between the client and the server is highly skewed. Most of the data flows from the client to the server, except in the extreme case, like *xterm2*, where the user kept generating inputs to the server. Even so, the data flow from the server is less than 40 percent of the total.

The X protocol sends its data to the TCP/IP in blocks. Each block contains several data units. Different policies are reflected here. At the client side, one block consists of many data units (an average of 28.15, 21.41, and 13.78 for *xterm, plaid,* and *ileaf5*, respectively), so that a highly efficient use of the underlying network is assumed. At the server side, however, one block only carries one data unit or two. That is because on one hand, the data units sent by the server are mostly *events* whose sizes are very small so that grouping does improve the efficiency, but on the other hand, short blocks may help reduce the response delay, so that user input can be sent to the client application quickly.

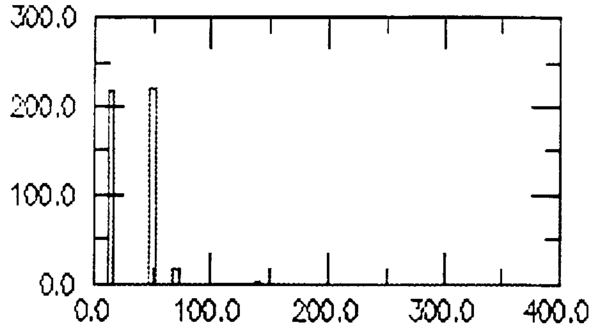
The block size depends largely on client applications. For the *xterm*2 case, most of the data was generated by the user moving the mouse pointer in and out of the client display window. So the X server kept sending the events EnterNotify, LeaveNotify, FocusOut, FocusIn. And the client responded with short requests specifying the (x,y) coordinates and the GC value (Graphic Context). While in the case of plaid drawing, the client program continued computing the plaid pattern and sending the corresponding drawing requests (PolyFillRectangle) to the server to update the plaid display. If taking the *xterm*2 and plaid as two extreme case data, the average block size of *requests* would be around several hundred to one thousand bytes per X block, indicated by the other two cases (1353.5 bytes for *xterm*, and 683.57 bytes for *ileaf*). But the large standard deviations indicate that the block sizes depart the means quite a lot, except in the *plaid* case. These can be visually illustrated by the distribution histograms in Figure 5. Most of the time, the block sizes are significantly shorter than the averages. With regards to the network delay performance, the shorter is better for the Ethernet technology.



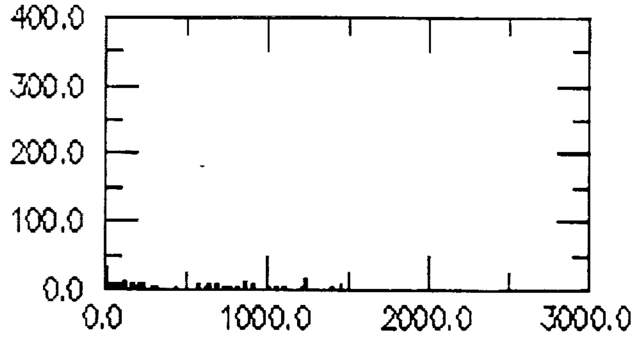
a. xterm



c. plaid



b. xterm2



d. ileaf5

Figure 5. block size distributions

The data block sizes from the server are much less than from the client, implying a much smaller impact on the network performance in terms of packet size.

There is shown an upper limit on the block size, 2048 bytes in our case. This limit is determined by the underlying protocol structure. The X protocol does not put a constraint on itself. It relies on the underlying network and assumes a reliable network connection is available. Once the connection is established, it sends as fast as possible data units without waiting for acknowledgements. Before being sent over the network each X data block is divided into several packets. The fragmentation is done by the TCP/IP protocol based on the lower–level network structures, protocol headers, etc.. The Ethernet data format allows only 1500 bytes per packet, certainly not enough for 2048 bytes. At the destination, an assembly procedure takes place to restore the X data blocks. The fragmentation adds some overhead to the data at both the TCP/IP and the Ethernet layer as shown below:

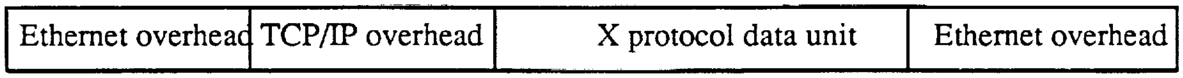


Figure 6. X data and overhead

On the other hand, the X window system is intended to be used in a general Internet environment. The client and server may communicate via several different networks. Also the IP datagram may take different routes to the destination point. The TCP protocol at the sending point does not have the knowledge of the physical network properties. Therefore it is hard for TCP to decide the best maximum packet size. The specification suggests a maximum size of 536 bytes for TCP packets, if it is used in a heterogeneous network environment. The Ethernet packet statistics are calculated in Table 4, where the bytes/pkt is the pure X protocol data per packet, excluding the overhead of TCP/IP and Ethernet. The standard overheads for TCP/IP and Ethernet are assumed, i.e., 40 bytes for TCP/IP and 26 bytes for Ethernet.

Table 4. Ethernet packet statistics (including both the client and server data)

applications	*xterm*	*xterm2*	*plaid*	*ileaf5*
mean pkts/block	4.38	2.55	12.06	3.03
mean bytes/pkt	211.98	15.26	162.82	124.45
mean pkt length	277.97	81.26	228.82	190.45
overhead %	24	81	29	35

The packet lengths are typically around a few hundred bytes, even under a heavy load (*plaid* case). It is worth noting that the packets transmitted for the *plaid* application are almost all *requests* to the server. This is to say, for large amounts of *request* traffic over the network the overhead is still high. In other words, the amount of overhead is independent of the application. This behavior may be explained by the fact that shorter network delay requires shorter packet size, which in turn results in higher overhead.

X traffic and Ethernet traffic

The generated X traffic has two patterns roughly. One is burst mode, like most interactive data traffic. The server tracks the user's random inputs with KeyPress, KeyRelease, etc., then sends events to the related client to trigger the transmission of drawing information of variable length. The other mode is when the client continuously executes a computation with some sort of algorithm and sends the results to the server to display, such as with the *plaid* program. The data rate is relatively constant, or with a small variance from the mean. To accurately characterize the bursty traffic is a difficult job. It may be left for further study. Here, we use the observations as well as the mean values to get some representative figures.

The instantaneous traffic intensity is determined by the amount of data arriving and the instant inter-arrival time. A detailed observation of the data captured by *xscope* shows that one X data block was generated and sent by the client side about every 0.103 seconds when displaying a large text file (in *xterm* case), and about every 0.06 seconds when displaying plaid patterns. In both cases, the number of bytes in a data block was very close to the upper limit, 2048 bytes, and with very small variance. This implies that the transmission rate was determined by the client program, not the protocol itself, in the measurement. The corresponding data rates are 159.068 kbits/s(*xterm*) and 273.067 kbits/s(*plaid*), respectively. This is calculated by multiplying the mean bytes per X data block with the mean data block generation rate. The mean values of X traffic are calculated in Table 5. The arrival rates in the first two cases indicate the user's interactive speed in front of the terminal. To determine an accurate measurement, the procedure must be conducted over a long period of time. For the machine–generated traffic, the arrival rate reflects the speed of the software and the hardware associated with the client and server.

Table 5. mean values of X traffic (including both client and server)

applications	*xterm*	*xterm*2	*plaid*	*ileaf*5
mean block arrival rate (blk/s)	4.93	8.77	15.38	9.09
mean block size	928.47	38.91	1963.60	377.07
traffic intensity	36.62 kbits/s	2.73 kbits/s	238.28 kbits/s	27.42 kbits/s

The resultant network traffic is calculated by multiplying the mean packet size, derived from the application level, with the packet rate measured at the network level. The results are summarized in Table 6. The traffic differences from the network level to the application level indicate the overhead of the TCP/IP and Ethernet protocols, plus some statistical errors.

Table 6. Ethernet traffic

applications	*xterm*	*xterm2*	*plaid*	*ileaf5*
traffic intensity	46.94 kbits/s	14.03 kbits/s	342.22 kbits/s	39.22 kbits/s
network util. %	0.47	0.14	3.42	0.39

The heaviest network traffic is 342.22 kbits/sec., which corresponds to 3.42 percent of the 10Mbits/sec Ethernet capacity. Assuming the Ethernet can handle an overall load up to 20 percent of its capacity without degrading performance, 6 *plaid* applications, or 50 *xterm* applications can run simultaneously over the network. Beyond that, the window performance constraints may shift from the machine to the network. It should be noted that since each X application is a random process with respect to its traffic, the mean values may not be sufficient for its description, especially for a widely spread distribution. In that case, simulation may be required. Another issue is that the traffic applied to the network is also a function of the machine speed. Software and hardware upgrades would cause varying increments of traffic intensity to the network.

Brief summary

Operating in the TCP/IP and Ethernet environment, X window applications do not generate much traffic to the network. The X performance is not restricted by the network. The traffic overhead due to the TCP/IP and Ethernet protocols is quite high, even though the X protocol aggregates data units into larger blocks before transmission. These blocks are broken down into small pieces at the lower layers to help reduce network delays. As long as the traffic volume is kept small, the overhead is not a big deal. Graphical plots of the four applications are given in Figures 6–9.

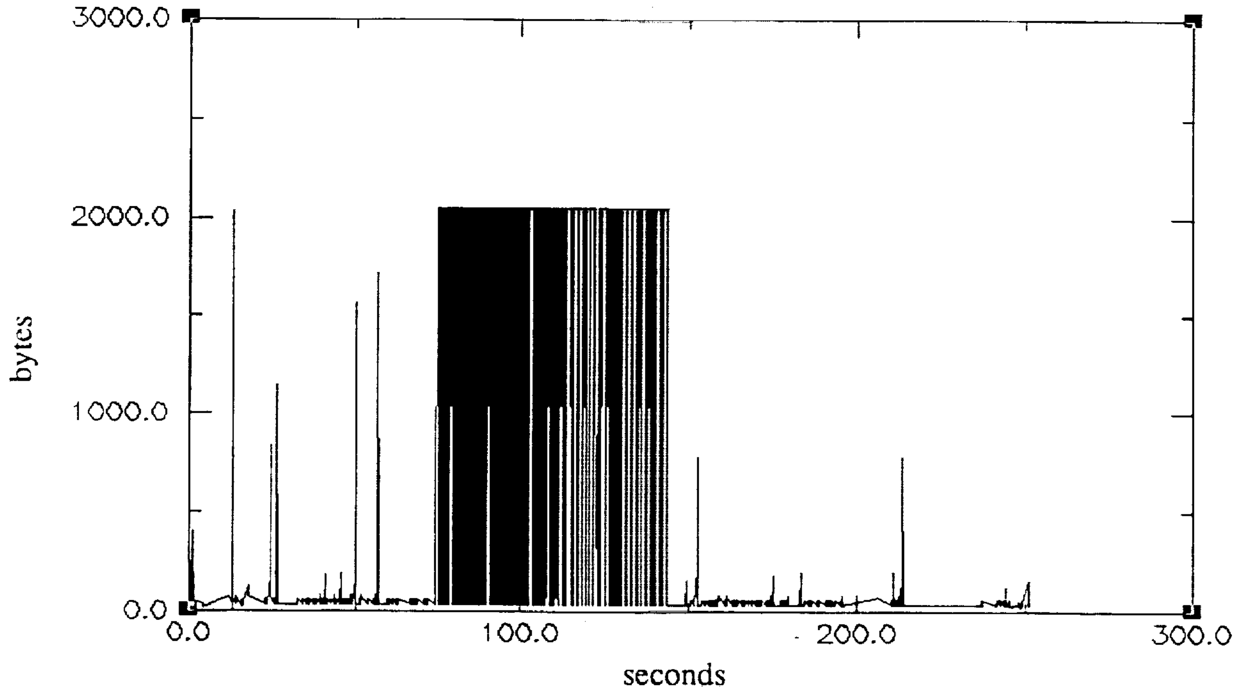


Figure 6.a xterm

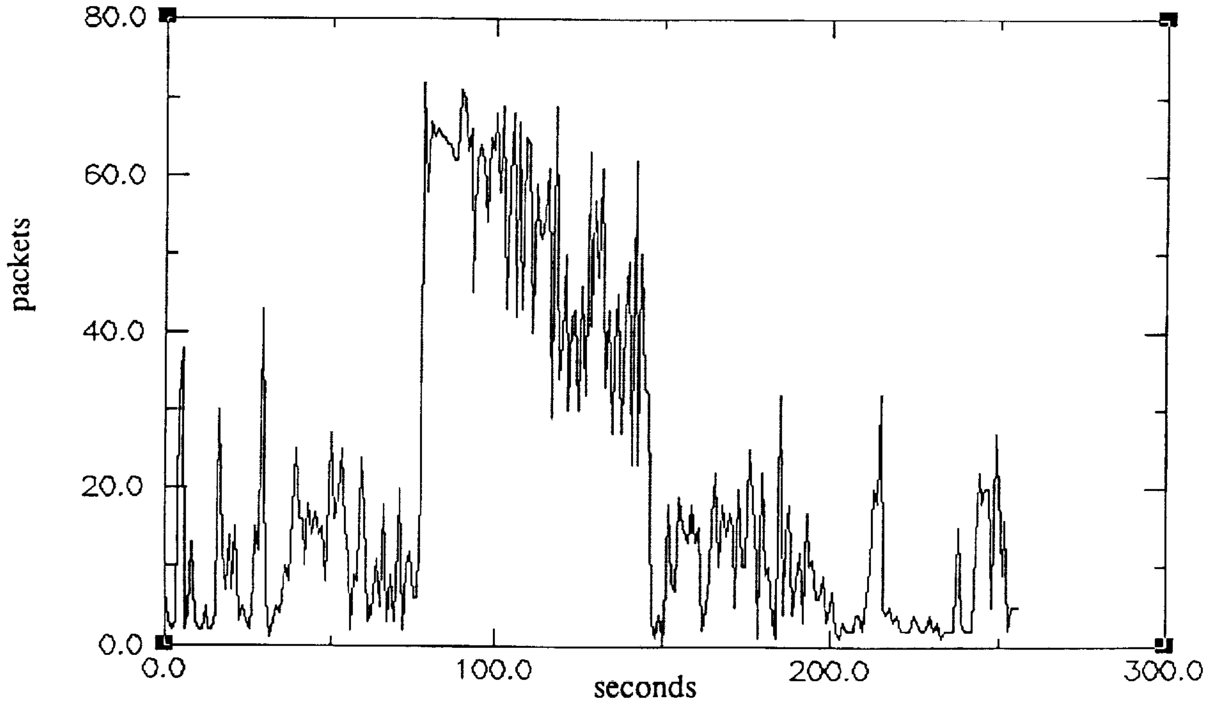


Figure 6.b xterm

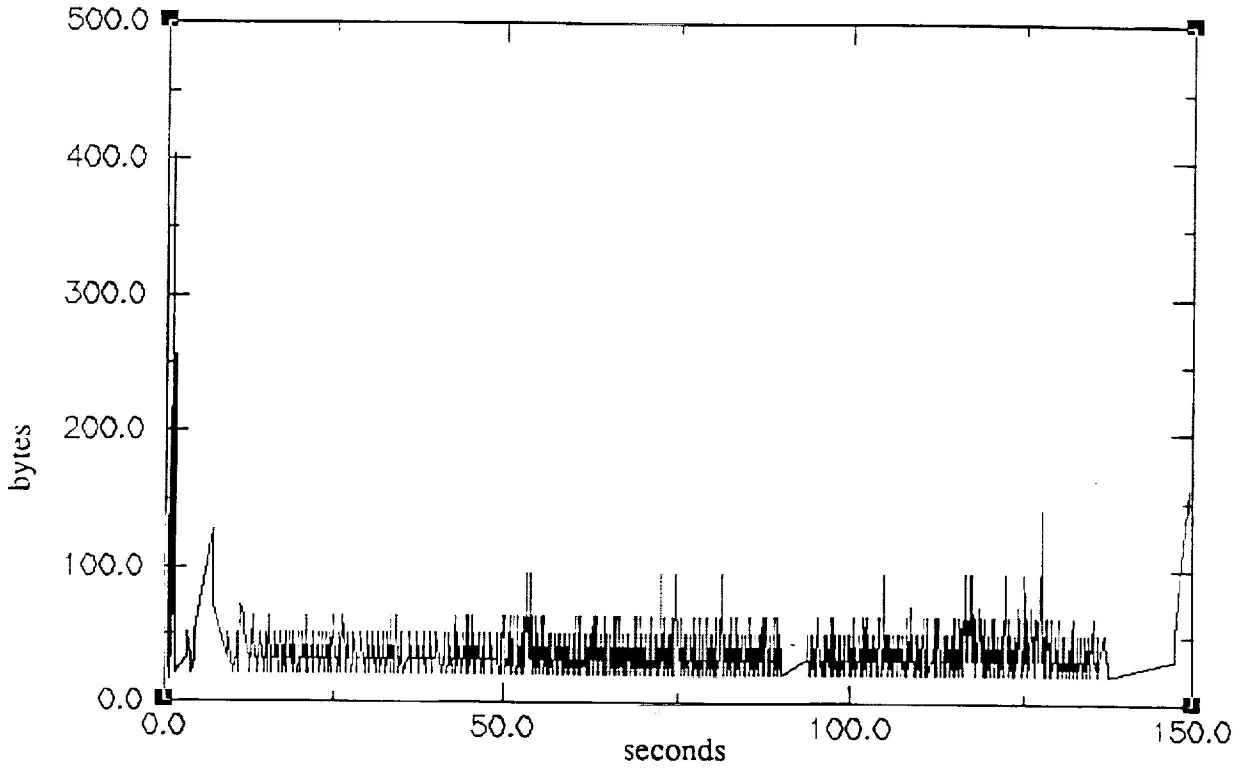


Figure 7.a xterm2

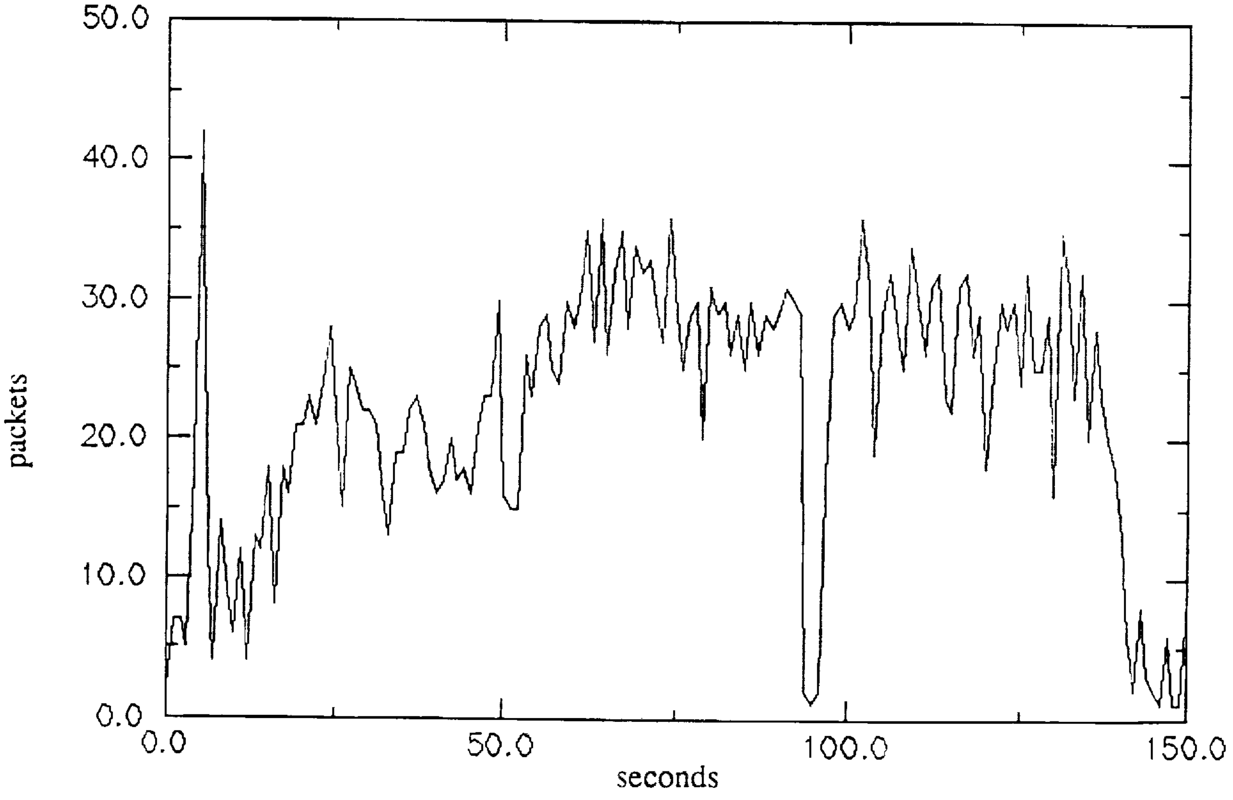


Figure 7.b xterm2

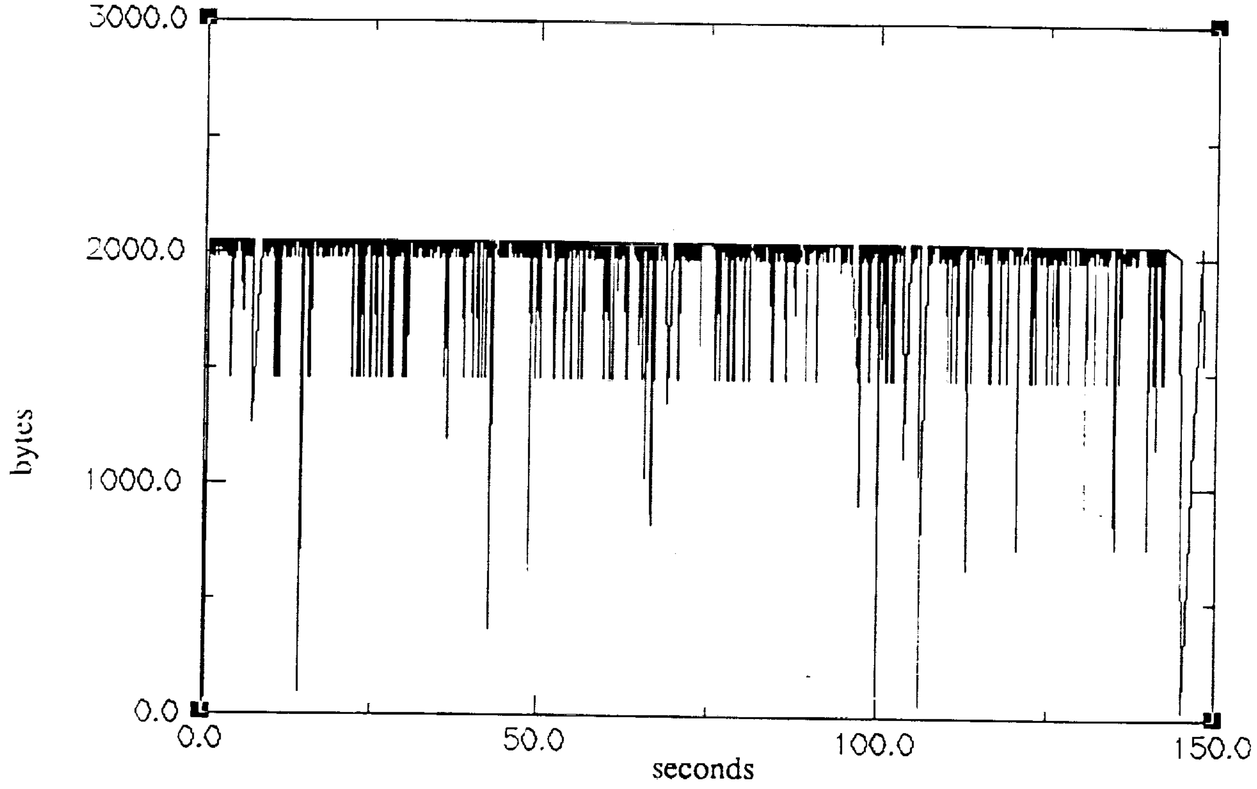


Figure 8.a plaid

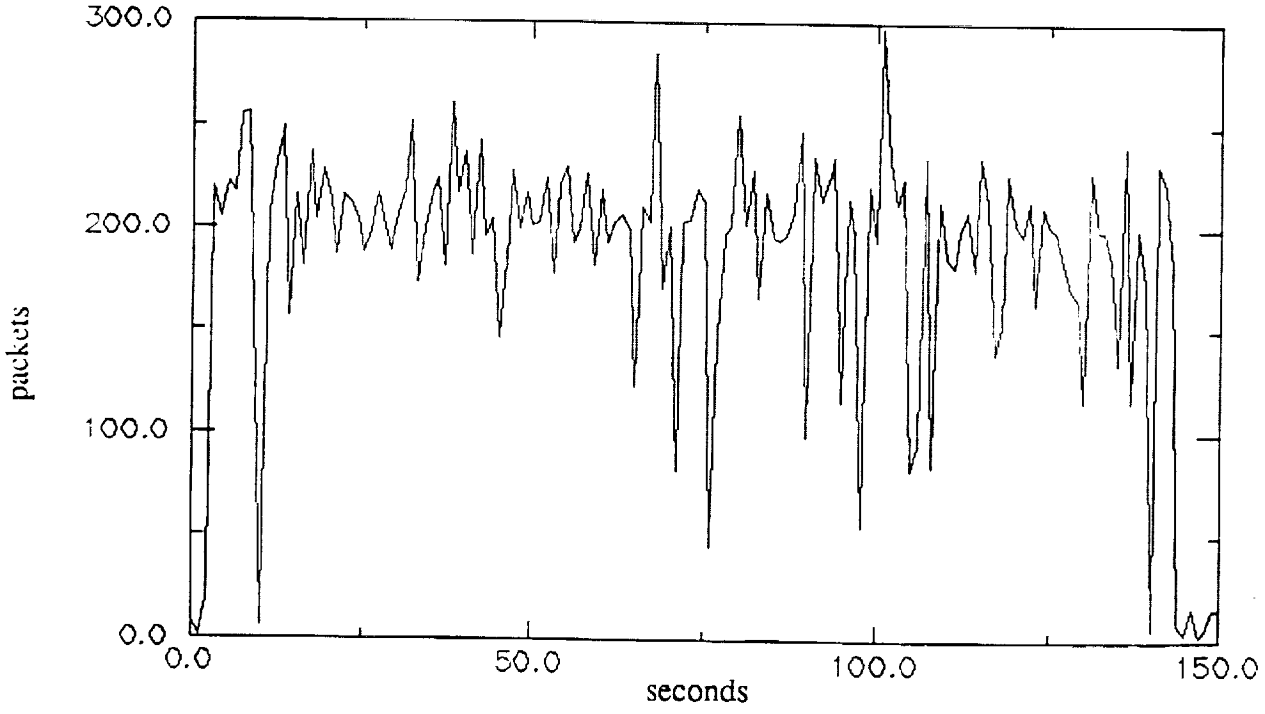


Figure 8.b plaid

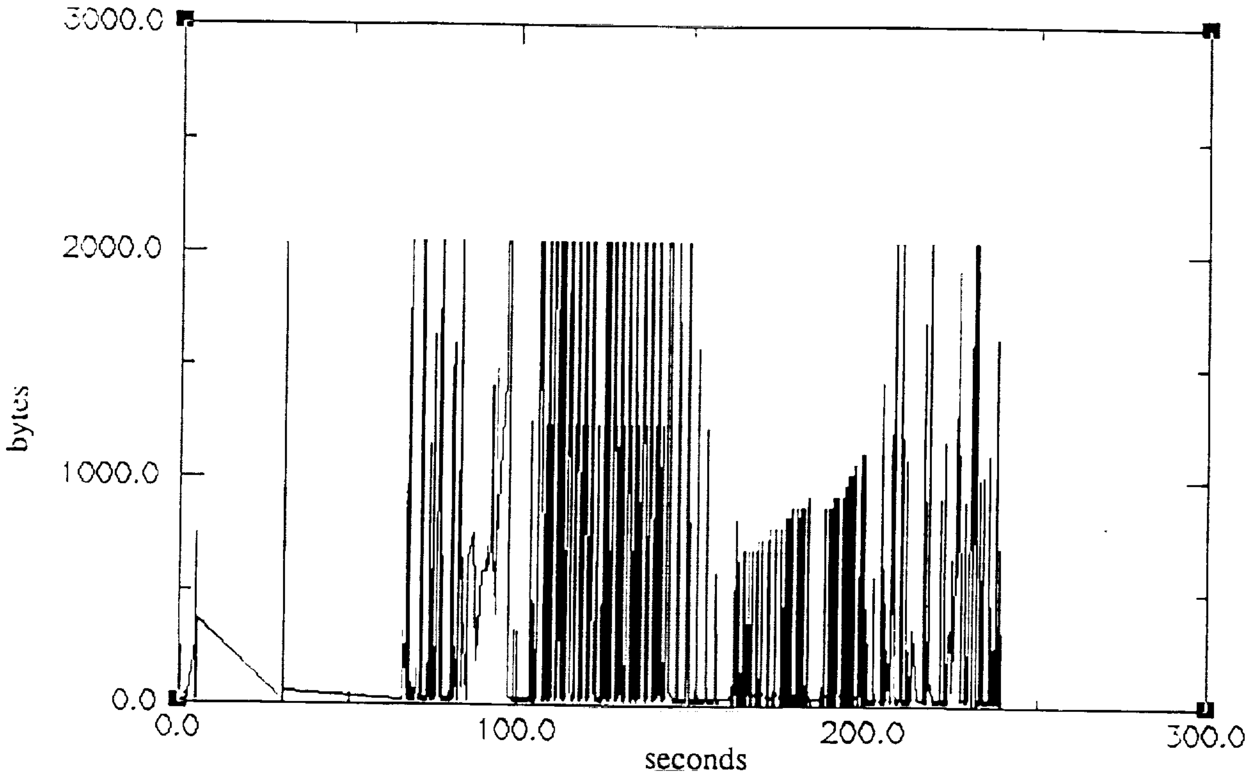


Figure 9.a ileaf

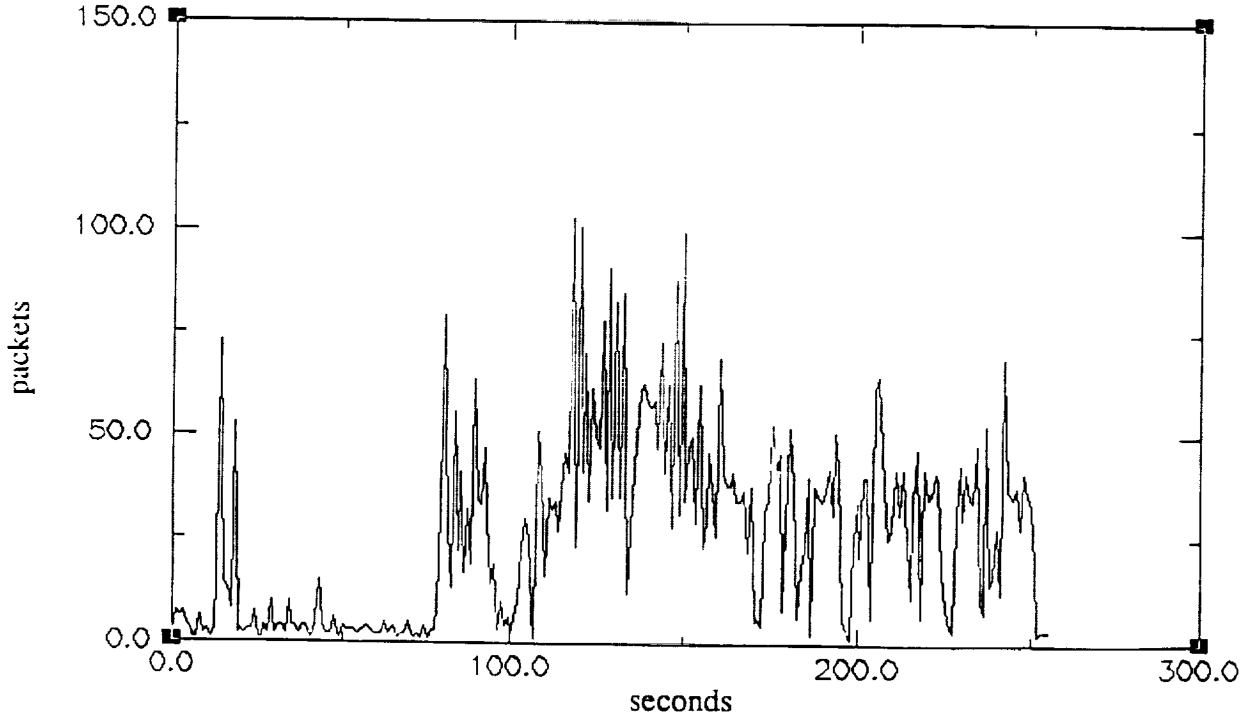


Figure 9.b ileaf

II. X APPLICATIONS WITH MACINTOSH

Much interest has been shown recently in optimizing X applications to make use of much less expensive personal computers as the X platforms instead of powerful but relatively expensive workstations. The advantage is that it provides PC users with the access to networked supercomputers while still maintaining all the PC's application features so that it takes full advantages of network resources and the PC's merits. Compared with X terminals, PC's do not have to live totally at the mercy of the central systems and are free of the inconveniences caused by the central systems maintenance schedules. In addition, the PC market offers a variety of selections for different users needs, such as fast computation, low cost, or high–resolution display. This wide range of choices makes it easier to achieve a good performance/cost ratio.

In this section, we present a survey study on Apple's Macintosh machines and different networking components with emphasize on performance aspects of X applications over Ethernet.

Network Configuration

The X applications with Macs include running Mac X servers on the Macintosh machines, and using X's cross–network capabilities to connect to remote clients, so that the Mac user can run applications on remotely connected workstations or minicomputers. Another possibility is the reverse situation: X applications (clients) are running on the Macs and results are displayed on remote workstations through the network. The first application is generally of greater interest.

Figure 10 presents a general networking situation when running X windows on networked Macs. The components involved include the X servers, network protocols, interfaces between the Macs and the network, and the network itself. A router/bridge may be used if the X client/server connection is being made to another network with different networking technology. Assuming that the machines running the X clients or other specific programs — with results being displayed on local Macs through the X window system — are much more powerful (such as DEC's VAXcluster, Sun or HP

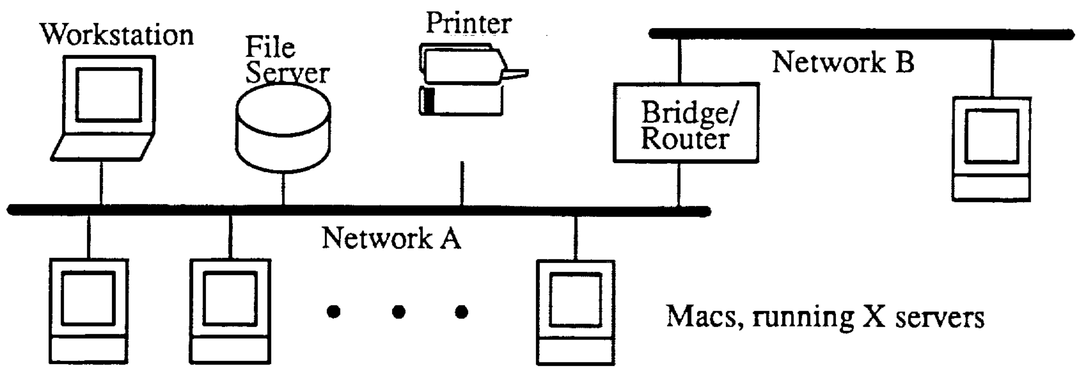


Figure 10. X windows on networked Macs

workstations), primary performance considerations of X applications should be given to the software and hardware elements in the Macs and in the network.

X server

There are two main X servers currently available for Macs: Apple's MacX and White Pines Software's eXodus. MacX complies with the MIT X Window System specification X11R4. It includes a built–in window manager that complies with the ICCCM standard. The window manager provides title bars, close boxes, and other Mac window controls. MacX also offers support for other window managers, such as OSF/Motif's *mwm*, or *twm*.

The latest version of MacX is MacX 1.1.7.

System requirements to use MacX by Apple include:

– Any Mac computer

– At least 2 megabytes memory

– At least two floppy disk drives (for Mac operating system 6.0.5 and later)

– A hard disk (for Mac operating system version 7.0)

– Mac operating system version 7.0, or version 6.0.5 and later

– A network connection (LocalTalk or an Ethernet connection)

Apple's UNIX system for Mac, A/UX 2.0, is an alternative operating system. In fact, A/UX has the MacX incorporated in it, so that it provides a native X environment.

To use Ethernet connection, it needs Apple's Ethernet NB card or equivalent products from other vendors. The transport protocols supported by MacX include MacTCP (Apple's implementation of TCP/IP protocols), AppleTalk ADSP (AppleTalk Data Stream Protocol), and DECnet. The shipment of MacX includes the MacTCP package.

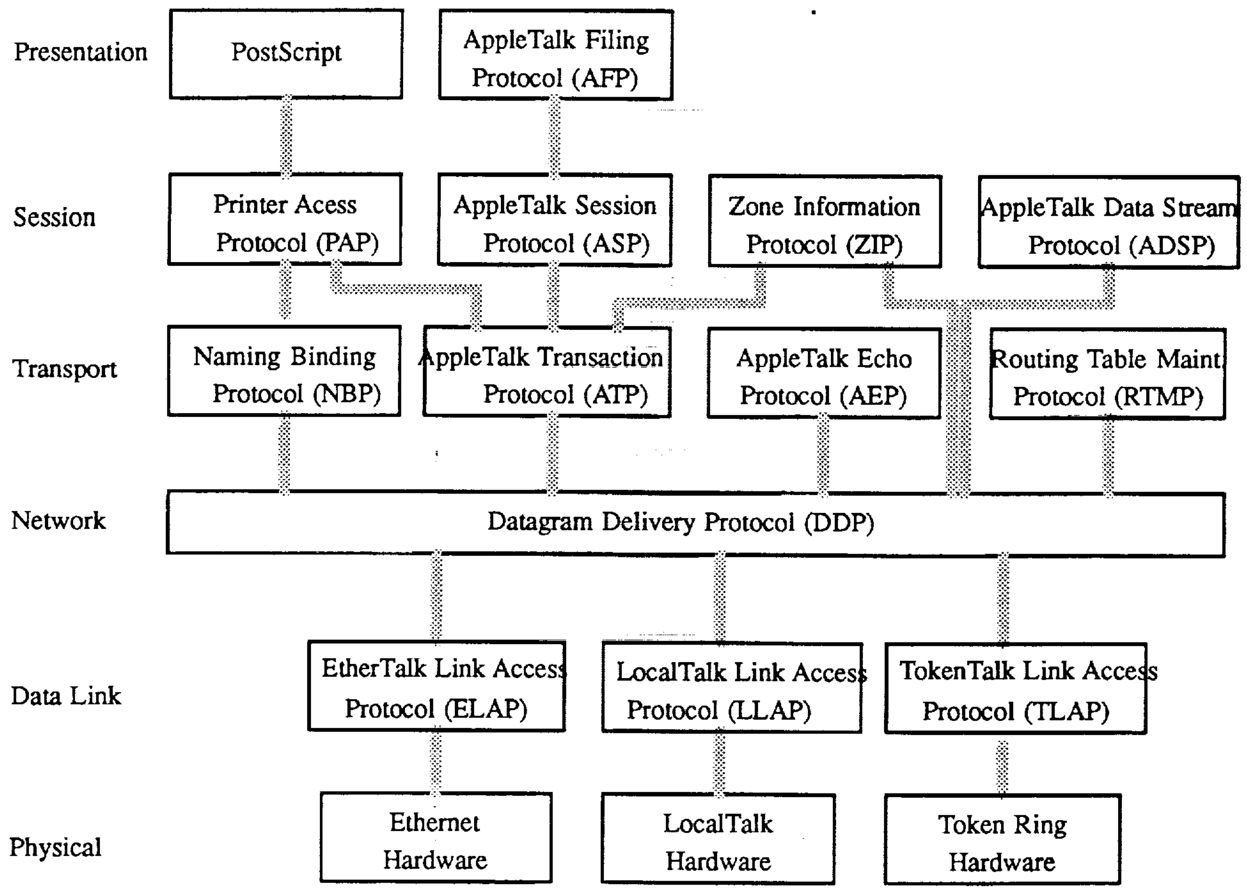
The performance of MacX is affected by its working environment, such as the speed of the Mac the MacX runs on and the operating system used. A test of performance efficiency for MacX and eXodus was conducted in [12] by using a standard X window benchmark program that ran all the integral X graphics algorithms in turn and then produced the statistical benchmark measured in Xstones. The Xstones represent the number of graphic operations per second. The MacX was running on a Mac IIci under the Mac operating system. The results are compared with those of MacX running under A/UX, and a Sun SPARCstation 2 using its own server:

	MacX under Mac OP	MacX under A/UX	Sun SPARCstation 2
Xstones	8285	13796	19374

Network protocols

AppleTalk is the network architecture Apple developed for Macs to talk to each other and to share printers and servers. AppleTalk is a set of protocols at different OSI layers. Originally AppleTalk only contained one data link protocol, called LocalTalk Link Access Protocol, which works on top of Apple's LocalTalk cabling, normally twisted–pair cables or phone wires. Each Mac product has the built–in functionality of LocalTalk. Thus it is easy for Macs to form a LAN over the existing phone wires in a building. The major drawback of LocalTalk is that its speed is rather slow: 230,400 bits per second. Some third–party products can be used to speed up the LocalTalk transmission. For example, Dayna Communication's DaynaTalk and TOPS's FlashTalk offer some performance improvements to running AppleTalk over LocalTalk cabling. The tests in [14] indicated an improvement of anywhere from 25 percent to 50 percent in file–transfer times. Even so, FlashTalk's top speed of 768 kbits per second and DaynaTalk's 850 kbits per second are still far below the Ethernet capability.

In 1989, Apple introduced AppleTalk Phase 2 to cope with the growing network [21]. AppleTalk Phase 2 did nothing for LocalTalk, but added Ethernet support and Token Ring support in the AppleTalk protocol suites. Also an extended addressing scheme was included to remove the original AppleTalk's 254–device limitation on the Mac network. To move to higher–speed networks, Ethernet has become an inevitable alternative to the slow LocalTalk. Ethernet provides the ways for users to quickly transfer intensive volume of data, such as large graphics files, so it is the natural solution for the bit–mapped X window applications. In addition, Ethernet is the most popular medium for networking UNIX workstations, IBM PCs, and DEC's VAX computers. Hooking Macs on the Ethernet makes it easier for Macs to access those other computers.



Presentation
PostScript
AppleTalk Filing Protocol (AFP)
Session
Printer Acess Protocol (PAP)
AppleTalk Session Protocol (ASP)
Zone Information Protocol (ZIP)
AppleTalk Data Stream Protocol (ADSP)
Transport
Naming Binding Protocol (NBP)
AppleTalk Transaction Protocol (ATP)
AppleTalk Echo Protocol (AEP)
Routing Table Maint Protocol (RTMP)
Network
Datagram Delivery Protocol (DDP)
Data Link
EtherTalk Link Access Protocol (ELAP)
LocalTalk Link Access Protocol (LLAP)
TokenTalk Link Access Protocol (TLAP)
Physical
Ethernet Hardware
LocalTalk Hardware
Token Ring Hardware

Figure 11. AppleTalk Protocol Suites

The transport protocols used in Macs to support MacX are AppleTalk ADSP, MacTCP, or DECnet. DECnet transport protocol is, of course, from another companies. MacTCP is shipped with each purchase of MacX. Therefore, MacTCP is the default transport protocol for MacX. It allows the Macs to connect to any existing TCP/IP network. To use MacTCP, each Mac needs an IP address from the network administration. MacTCP can run under AppleTalk with Ethernet connection.

Ethernet interfaces

In the past years, the number of Mac Ethernet interfaces has proliferated, and costs have come down. These Ethernet interfaces fall into two categories: cards that plug into a slot inside the Mac, and external adapters that plug into the Mac's SCSI port. The internal interface cards are connected in the Mac either to the NuBus slot or PDS (processor direct slot), depending on the Mac's type. SCSI stands for *Small Computer Systems Interface*. SCSI enables the Mac to communicate with high–speed peripherals, such as hard drives, image scanners, personal laser printers, CD ROM drives, and tape backup devices. Externally, SCSI is a port designed to attach additional hardware to the Mac. It works as a parallel bus, transmitting 8 bits at a time. A new version of the SCSI, SCSI–2, enhances the bus transmission performance by expanding SCSI's parallel data path from 8 bits to 16 bits, making a possible data transfer rates of 10–40 megabytes per second. The devices connected to SCSI are generally in a series configuration, called *daisy–chaining*. That means the SCSI bandwidth must be shared among these devices. Therefore, Mac's SCSI bus is generally slower than both the NuBus and PDS. For example, the 32–bit NuBus has a maximum quoted performance of 37.5 megabytes per second. However, the SCSI connection is compatible to all types of Macs provided that they have a SCSI port available.

There are four types of cables for Ethernet: 10BASE–5, 10BASE–2, 10BASE–T, and fiber–optic. 10BASE–5 is a thick coaxial cable with the segment length of no longer than 500 meters. It is best used as a network backbone. To attach a Mac to the thick coax usually requires a transceiver from the Ethernet interface card. 10BASE–2 stands for a thin coaxial cable with a segment length of up to 200 meters. The thin coax is less expensive and more flexible. Most Mac Ethernet interfaces have built–in transceivers for thin coax, so there is no need for external transceivers. But Apple's Ethernet LC card and Ethernet NB card have no transceivers on board. They include a special socket, called Apple Attachment Unit Interface (AAUI). So, using these two Ethernet cards does require an external transceiver. One of the advantages of using AAUI is that one does not need to change the Ethernet card when switching Ethernet cables, say, from 10BASE–2 to 10BASE–T. 10BASE–T runs over twisted–pair phone wire and usually uses a star topology. A 10BASE–T cable length (the distance allowed between a node and the star hub) is limited to 100 meters. The star topology makes 10BASE–T wiring easier to modify than its coaxial cousins, because the star hub (concentrator) resides at a central location. Each Mac must have either an Ethernet interface with a 10BASE–T transceiver built in or a built–in AUI (Attachment Unit Interface) port connected to a separate external 10BASE–T transceiver. The fiber–optic cable is the choice for the future. It outperforms the coax in many ways. But currently only a few Mac Ethernet interfaces support it, e.g., Mac2000 Board from Network Resources.

A list of main Ethernet connection interfaces is tabulated in the following tables [13,15,17,22].

Boards

Interface	Vendor	Mac	thick	thin	tw. pair
EtherLink/NB	3Com	IIs	x	x	x
EtherLink/NB TPX	3Com		x	x	x
Ethernet LC Card	Apple	LC	x	x	x
Ethernet NB Card	Apple	IIs	x	x	x
MacCon+	Asanté Technologies	IIsi, SE/30	x	x	x
FriendlyNet Card	ditto	LC		x	x
LanWay Ethernet	Avatar	LC	x	x	x
E4010–E6020	Cabletron Systems	IIs, SE/30	x	x	x
GatorCards/E	Cayman System	IIs	x	x	x
Ether2/Ether DS	Compatible Sys. Corp	IIs	x	x	x
DaynaPort E	Dayna Comm.	IIs, LC, SE/30, SE	x	x	x
FastNet	Dove Computer Corp.	IIs, SE/30, and SE	x	x	x
PhoneNet Card for Ethernet	Farallon Computing	IIs, SE/30	x		x
Magic Ethernet II	MacProducts USA	IIs	x	x	x
EtherNode 16	National Cemiconductor Corp.	IIs, SE	x	x	x
EtherNode 32	ditto	SE/30, IIsi	x	x	x
Mac1000	Network Resource	IIs, SE	x	x	x

NuvoLink II	Nuvotech	IIs	x	x	x
MacConnect	Racal–Datacom	SE, SE/30, IIsi	x	x	x
EtherPort	Shiva Corporation	IIs, SE, SE/30	x	x	x

Note: IIs means IIsi, II and up.

SCSI type

interface	vendor	mac	think	thin	tw. pair
Asanté EN/SC	Asanté Technologies	all	x	x	x
Ether+	Compatible Sys. Corp	all	x	x	x
DaynaPort	Dayna Comm.	all	x	x	x
NuvoLink SC	Nuvotech	all	x	x	x

Over the years as Mac networks moved from the slow LocalTalk to Ethernet, many studies have been made on the performance of Ethernet interfaces to investigate the possible potential bottleneck from the Mac to the Ethernet highway. Here we present the most interesting results from two tests [17, 22].

In [17] 15 twisted–pair Ethernet boards for Macs were tested. These boards include: 3Com's EtherLink/NB, Apple's Ethernet NB Card, Asanté Technology's MacCon3 for NuBus, Avatar's LanWay Ethernet, Cabletron's E6000 DNI, Compatible Systems's Ether2, Dayna Communications's DaynaPort, Dove Computer's FastNet III N, Farallon's PhoneNet Card for Ethernet, National Semiconductor's EtherNode 16 NB, Network Resources Corporation's Mac1000, Racal–Datacom's MacConnect Ethernet Card, Shiva's EtherPort II, Sonic Systems's Sonic Ethernet Series, Technology Works's Ethernet Kit. In the tests, two Mac IIci's were connected over Ethernet, and executed file transfers between themselves. The transfer rate was timed for each Ethernet board. To eliminate the limiting factor of the slower hard drive, a RAM disk was used instead to store the files. Thus the affecting elements are the Ethernet interface and the processor of the Mac. The results showed very little difference in the transfer rates among the tested boards when transferring a 5MB file. The average rate is about 1.5 Mbits per second. Smaller file needed relatively longer time to transfer, due to the fixed overhead of the control packets. A comparison between the RAM disk and hard drive was also tested. Not surprisingly, RAM disk provided much better Ethernet performance, which indicates that the hard drive, if it is involved in the connection, is probably the bottleneck. Although the 1.5 Mbits per second result does not separate the Ethernet board from the processor or prove that the Ethernet board might not be saturated, it does show a 1.5 MB rate is obtainable for the Ethernet boards.

Another test was carried in [22] about two years ago. It tested different Ethernet boards, and SCSI adapters on Mac IIs, SE/30, SE, and Plus. The results showed the performance of Mac IIs, and SE/30 with internal Ethernet boards (NuBus, PDS) is much better than SEs with Ethernet boards, or SE/30 with SCSI adapters. It reveals that the CPU or the SCSI port is the performance bottleneck, not the Ethernet interface. A further effort was made to test the interface raw performance. The raw performance is the fastest speed the interface can transfer data without being held up by the CPU or the SCSI. Network General's Sniffer together with a specially developed NetBasher! by MacUser NetWorkShop was used for this test. The maximum data transfer rates measured by the Sniffer were about 4.0–4.8 Mbits per second, where Asanté's MacCon+, Racal–Datacom's MacConnect, Cayman's GatorCard, and Cabletron's E6000 outperformed the others.

There are other tests on the Ethernet interfaces, some using client/server file transfer, some using special programs to generate data traffic to avoid accessing external devices. None of those tests observed the bottleneck at the Ethernet interfaces.

Mac machines

For an X server running on a Mac, choosing an appropriate Mac also means having better performance. The Macintosh family consists of many combinations at different prices to cater for different demands. The processor speed, memory size, bus configuration, etc. all contribute to the overall performance of a Mac. Most of the Macs in the current market include either M68030 or M68020 microprocessors, with a clock speed of 16 MHz to 32 MHz. The M68020 has an instruction cache, while the M68030 has both an instruction and a data cache. A cache is an area of memory that can be directly accessed by the CPU without the delays involved in reading from main memory, thereby increasing the effective processor speed. The profiles of various Macs are summarized in the Appendix.

Mac LC is the least expensive color Macintosh. It uses a 68020 microprocessor running at 16 MHz. It does not include a math coprocessor, the trade–off for the low price. But it has a built–in color video circuitry capable of generating 256 colors. To take advantage of the 256 colors, Apple introduced a sharp, low–cost 12–inch color RGB monitor, with 512x384 pixels. The LC has a standard 2 MB RAM expandable to 10 MB, to accommodate for the Mac operating system version 7.0. But the memory–management unit is not included. Without the MMU, the LC cannot utilize the virtual memory feature provided in Mac operating system version 7.0.

The LC II looks very much like the LC on its exterior. And it sells for nearly the same as or less than the LC. The big difference is that a 68030 microprocessor, instead of 68020, is included. There is not much of performance difference between LC and LC II. But the LC II offers virtual memory capability under the Mac operating system 7.0.

Quite a few tests of Ethernet interfaces used Mac II series computers, of which Mac IIsi is considered to have the better performance/cost ratio. The IIsi is faster than the LC. It can accept a NuBus expansion board, of which there are considerably more than there are LC boards. And the IIsi moves data between the memory and the processor in 32–bit portions instead of 16–bit as the LC II does. The IIsi can run Apple's A/UX version of the UNIX operating system. But IIsi costs hundreds of dollars more.

The fastest of Mac the II series is the IIfx. Its M68030 microprocessor runs at 40 MHz. However, is the most expensive.

Other factors

The operating system is another factor that needs to be considered for X window applications. Besides the Mac operating systems, Apple has a UNIX operating system, A/UX 2.0, for Macs. A/UX has the built–in MacX package, and tests in [12] indicate that running MacX under A/UX provides better performance. A/UX supports standard TCP/IP communication over Ethernet to other UNIX machines. To use TCP/IP protocols, there needs to be an Ethernet card installed in the Mac and configured to its TCP/IP mode. A/UX also provides AppleTalk connectivity over LocalTalk or EtherTalk networks. If both the LocalTalk and Ethernet cards are installed, you can choose between these networks. But A/UX requires a faster Mac, and at least 8 MB memory in order to achieve this better performance. If the Mac users do not need many UNIX features and networking services, the investment for A/UX would not be economically justified.

If the X clients are going to run on a different network than the one to which the Macs are connected, a router may be used to connect these networks. In this case, the X client/server connection involves data passing through the router, which should be taken into account in the overall performance assessment.

Brief summary

Multiple factors must be considered for running X servers on Macs. Tests show that the Ethernet network has sufficient bandwidth to support X applications. The Ethernet card should not be the bottleneck for connecting Macs to Ethernet given to the raw data traffic it can support, which means the real world applications must be able to generate enough traffic to saturate the Ethernet and the Ethernet cards. Using faster Macs may help improve the X window performance. And more memory size is generally a good choice to reduce the use of slower hard drives. A/UX provides a better environment than Mac O.S., but costs more.

Appendix to II: Mac Profiles

Mac II series

	Mac II	Mac IIx	Mac IIci	Mac IIsi	Mac IIcx	Mac IIfx
Processor	68020	68030	68030	68030	68030	68030
Math co-processor	68881	68882	68882	68882 (option)	68882	68882
Clock speed	16MHz	16MHZ	25 MHz	20MHz	16MHz	40 MHz
Max. RAM	8 MB	8 MB	32 MB	17 MB	8 MB	32 MB
Process cache	instruction	instruction and data	instruction and data, 32K cache	instruction and data	instruction and data	instruction and data, 32K cache
ROM	256K socketed chips	256K SIMM	512K SIMM	512K SIMM	256K replaceable	512K SIMM
Color/gray–scale support	yes	yes	yes	yes	yes	13–inch, 640x480 RGB, 24–bit color card
Internal expansion slot	6 NuBus	6 NuBus	3 NuBus	1 NuBus	3 NuBus	6 NuBus

Mac Plus, Mac SE, Mac SE/30, Mac Classic series, and Mac LC series

	Mac Plus	Mac SE	Mac SE/30	Mac Clas-sic	Mac Clas-sic II	Mac LC	Mac LC II
Processor	68000	68000	68030	68000	68030	68020	68030
Math co-processor	no	no	68882	no		no room	no room
Clock speed	7.83MHz	7.83MHZ	15.6672MHz	8 MHz	16 MHz	16 MHz	16 MHz
RAM	4 MB	4 MB	8 MB	4 MB	10 MB	2 – 10 MB	4 – 10 MB
Process cache	no	no	instruc-tion and data	no	instruc-tion and data	instruc-tion	instruc-tion and data
ROM	256K socketed	256K socketed	256K SIMM	512 K	512 K SIMM	512 K socketed	512 K socketed
Color/ gray–scale sup-port	no	no	yes	no	yes	yes, 16–bit color	yes, 8–bit, or 16–bit color

III. DECnet Capabilities

Introduction to DECnet

DECnet is a proprietary network facility developed by Digital Equipment Corporation. It has undergone five phases in development from phase I through the current phase V. Most of the earlier phases were developed around proprietary DEC protocols and services. Phase V has seen a migration to providing more Open Systems Interconnection (OSI) services and protocols in addition to an upgrading of the phase IV services. Since phase V was designed to remain compatible with earlier phase IV, a short look at phase IV makes it easier to understand the components of phase V. Phase V is also sometimes called Advantage–Networks.

Phase IV DECnet is generally centered on work groups that are connected together with Ethernet facilities. These work groups are then interconnected with wide area bridges or routers. Within the work groups, the standard Ethernet protocols are utilized. For wider distribution, Digital's Digital Data Communications Message Protocol (DDCMP) is usually used. Above these data link protocols are the DEC proprietary network, transport and session layers. On top of the session layer are several applications, two of the most common of which are the Data Access Protocol (DAP) and the Command Terminal (CTERM) protocol. DAP provides access to remote data bases, and CTERM is a remote terminal program, very similar in nature to the Telnet feature of TCP/IP.

These two services are supplemented by a number of other services, including a messaging system over DEC's Mailbus. Other services are provided for videotext, remote consoles, booting diskless nodes, bulletin boards, etc.

Phase V continues to include all the items from phase IV, including Ethernet, the IEEE 802.3 version of Ethernet, DDCMP, the permanent virtual circuit portion of X.25 networks, and the IEEE token bus protocol. In addition, phase V has added services supporting the ISO High–Level Data Link Control (HDLC) protocol, as well as the Fiber Distributed Data Interface (FDDI) protocol. Phase V also introduced two new proprietary DEC protocols that allow public data networks to be incorporated into DEC environments. These services are the Modem Control protocol (physical layer), and the Dynamically Established Data Link (network layer). These services permit dynamic establishment of modem connection through a public switched network using X.25 packet switched networks. For LAN's, DECnet phase V supports the Logical Link Control (LLC) protocols that may use either Ethernet, FDDI or token ring.

At the network layer, DEC uses ISO standard format 8473 for data and error packets. Hence interoperability with non–DEC equipment is maintained through this standard. Network routing is determined by a combination of the ISO 9542 standard and some additional DEC proprietary routing software. The 9542 protocol is primarily concerned with routing between End Systems and Intermediate Systems (ES–IS), while the DEC routing algorithms deal with Intermediate System – Intermediate System (IS–IS) routing. Non–DEC equipment that is not ISO 8473 compliant can be connected to a DECnet by use of a common subnet such as X.25.

At the transport layer, DEC uses the Network Services Protocol (NSP). This protocol is similar to TCP or the ISO TP4 protocols. Phase V also supports two other ISO protocols, TP0 and TP2. TP4 is used primarily for DEC applications, and the other two for non–DEC systems.

Up through the network layer, DECnet phase V is a fairly conventional OSI system with the inclusion of DDCMP to maintain compatibility with phase IV, and MCS for extra features. At the Session layer, there is a distinct split, and DECnet provides two separate session layers. The OSI session layer is provided to support OSI applications. A separate session layer, Digital Session Control protocol (DSC) is provided to support DEC services and to provide interconnection to proprietary DEC services at the lower levels of phase IV and phase V.

With all these different options at many levels, it is obvious that there is an almost bewildering combination of protocol stacks that are available to the users. Almost any of these combinations could be used in certain instances. For each application, DEC keeps track of the protocol stack in use through something called a tower. A tower is a set of addresses from the network layer on up. Each node keeps a list of towers, representing the possible combinations of protocols that might be used for communication. When communications is desired between two nodes or applications, the session layer examines the tower sets, and comes up with a common set that can be used for communication. A part of the DSC called the Domain Name Service (DNS) provides a distributed naming service that permits applications to communicate across the network using a logical name.

The towers are used only within the DECnet domain. For other connections, the OSI session layer is used. On top of this layer, DEC uses File Transfer Access and Management (FTAM) protocol and X.400. The FTAM implementation also has something called an FTAM/DAP gateway which allows DEC networks to access non–DEC FTAM systems. DEC has also implemented gateways for Telex and the TCP/IP SMTP.

Ethernet and DECnet

The major user of the data link layer in a DECnet system is the Digital Network Architecture (DNA) network layer. In addition, there are two other users that do not necessarily access the data link layer through the network layer. These users are the Local Area Transport (LAT) and the Maintenance Operations Protocol (MOP). LAT is a direct user of the data link layer that is used to provide terminal to host transfer. MOP is used to download operating systems to a remote host for booting diskless workstations and PC's. In addition, other network services may share the data link with these protocols. This can include services such as the naming service, the time protocol, remote procedure calls (RPC) and TCP/IP.

Digital originally used DDCMP as the major protocol on their networks. This protocol is still used for some wide area communications, but it has largely been replaced with Ethernet for most local area communications environments. Most Digital equipment is delivered with Ethernet interfaces, and this includes not only computers, but printers and disk servers as well. A very large percentage of the communications within a DECnet takes place over an Ethernet physical layer. This is particularly true for local traffic.

An appropriate model of a DECnet phase IV system would include a number of devices interconnected locally with Ethernet facilities. A set of these groups of workstations would then be interconnected by DDCMP facilities. For a phase V system, this model would be changed to add a number of different alternatives for the WAN interconnections. These would include FDDI, X.25 and HDLC. The move from DDCMP to HDLC places DEC into position to interface with the soon to be very important Integrated Services Digital Network (ISDN). Phase IV also supported HDLC, but it has become much more important in phase V.

In some cases, it is desirable to connect several devices to the Ethernet at a single point. When this is the case, a multiport transceiver that operates more or less as a data concentrator can be used. The Digital equivalent of a multiport transceiver is called a Digital Local Network Interconnection or DELNI. This device is sometimes referred to as "Ethernet in a can."

MOP

Within DECnet, there are several support protocols that are of interest in the development of the Yellow Creek network facility. Some of these services are significant from the network standpoint, but are of no special interest in this paper. These include the DNA naming service, the time service and remote procedure calls service (RPC). One support protocol of particular interest at this point is the Maintenance Operations Protocol or MOP.

As was noted earlier, at the session layer and above, there are two distinct architectures in use in a Digital network. These are DECnet and OSI. To these, a third must be added, and this is MOP. Technically, MOP is part of the DECnet protocol, but from a practical standpoint, it is a separate architecture. MOP is a direct user of the data link layer, and can operate without any of the DECnet protocols. Typically, MOP uses an Ethernet link, but it can operate over HDLC or DDCMP.

MOP is a fairly primitive protocol, and contains very limited security features. While MOP offers three levels of functionality, including testing the communications link and operating as a remote console, the main use of MOP is in download operating systems to a diskless workstation. While this can be initiated from a console, it more commonly takes place when the workstation is initialized.

When a diskless workstation is powered up, it utilizes the MOP to broadcast an appeal for help. A node on the network will then volunteer help. The diskless workstation responds with a memory load message, and data messages transfer the required data from the volunteer station into the memory of the workstation. The diskless workstation starts with a primitive primary loader that is used to load a secondary loader into the workstation. The secondary loader then loads a tertiary loader that can load the operating system into the workstation.

Instead of loading the normal operating system, the tertiary loader can also be used to load the network management initialization script (CMIP) into the workstation. This script is used to set network parameters such as maximum data packet sizes and retransmission timers, etc. Once the diskless workstation has been booted, it then performs its communications tasks over the network like any other node on the network.

LAT

An important concept in the Digital environment is that of the VAX cluster. The purpose of the VAX cluster is to take a disk drive (actually two) and make it available to multiple users simultaneously. This is accomplished through internal communications protocols that are transparent to the network users.

A VAX cluster typically consists of two or more VAX minicomputers. These computers communicate via a DEC 70 Mbps dedicated system backbone called the CI bus. The computers also communicate over the same bus with two Hierarchical Storage Controllers (HSC). The HSC is a specialized disk server that is connected to two disk drives. The disk drives are dual ported to multiple controllers.

All the storage facilities in the cluster are duplicated to provide redundancy and a measure of fault tolerance. Data written to the disk is shadow written to both disks simultaneously. This also improves fault tolerance. A lock manager running on the VAX system coordinates access to the disks from all users.

The combination of the minicomputers, the CI bus, the HSC's, the disks and the lock manager constitute the VAX cluster. To the users of the system, all the different VAX systems look the same. They all have access to the same data and the same peripheral facilities. Therefore, most users don't care which system they log into, except that they would like to log in on the computer that is likely to give them the best service.

The DEC Local Area Transport (LAT) was originally developed by Digital to solve the problem of which system VAX system to access. Like MOP, LAT is a relatively low level protocol that accesses the data link layer directly without reliance on many of the architectural features of DECnet. Since LAT provides the capability to connect many different clients to many different servers, it has a much broader application that first envisioned, and can be used in a number of different ways.

LAT attempts to balance the load on the VAX minicomputers through the use of a rating system. When a new session is established, LAT attempts to select the node that has the best rating at that time, and will, at least theoretically, provide the best service. Unfortunately, if service deteriorates, the load is not re–balanced.

LAT also provides the capability for multiplexing messages from several users (through the terminal server) into a single message. Because of this multiplexing capability, LAT makes an excellent platform for an X windows system. The terminal server can combine the many asynchronous events associated with an X session into a single packet for transmission to the computer. This multiplexing avoids a lot of the interrupts associated with an interactive system, and greatly reduces the load on the CPU.

Since LAT accesses the data link layer directly, it assumes an underlying IEEE 802.2 (data link) and a multicast capability. LAT can operate over either a CSMA/CD network or FDDI, although Ethernet is more common. LAT is intended to utilize only a small percentage of the bandwidth on the network.

LAT is a master/slave system based primarily on the use of timers. This philosophy greatly reduces the complexity of the slave component. Because LAT doesn't utilize the upper layers of the DECnet architecture, it is a relatively simple protocol, and provides little in the way of security, fault tolerance or transport service. LAT depends on the underlying data link layer to provide some of these facilities.

There are two major layers in LAT: the virtual circuit layer and the session layer. The virtual circuit layer is concerned with the establishment, maintenance and disconnection of the virtual circuits between the server and the host, and provides those services normally associated with a virtual circuit. The session layer provides the user interface to the system. The session layer performs the multiplexing of data from several terminals into a packet for transport over the network. LAT also provides a directory service that manages names for ports, nodes and services on the system.

In the LAT system, the terminal server normally assumes the role of the master, and the host is the slave. Whenever a timer in the master expires, the master transmits data to the slave. The slave is then able to transmit data back to the master by piggybacking this data onto the acknowledge packet. In the event that the master has no data to transmit, a path test message is used to maintain the virtual circuit and also enable data transmission from the slave to the master.

The LAT system usually operates in an unbalanced mode where the terminal server is the master and initiates all communications. One exception, however, is the case of a printer. Here, the printer is the slave and the host is the master. In this situation, the printer can initiate a connection and also initiate communications when necessary.

Flow control in LAT takes place at several levels. Both a credit system and a window–based flow control system are provided at different levels. There is also an X– on/X–off system for the user as well as some flow control in the underlying data link layer.

In essence, LAT is a fairly primitive communications system that is reasonably effective in communicating with the VAX cluster. By the use of the master/slave timer system and multiplexing, LAT places a large share of the communications load on the terminal server, and thereby frees up resources on the host.

Summary

In summary, DECnet V, like OSI, represents a philosophy as much as a communications system. It incorporates much of the architecture espoused by OSI. At the same time, it provides backwards compatibility with many features of the earlier DECnet systems. In many instances, DECnet has moved in to fill gaps in the OSI architecture with proprietary protocols. These protocols are usually coordinated with other interested parties in an attempt to gain widespread acceptance of these protocols.

All in all, DECnet is a flexible, expandable network that should serve the needs of a wide variety of users. With the impending inclusion of FDDI, it should serve well into the next decade. It is a worthwhile addition to the ASRM network system.

References

[1] *X Protocol Reference Manual,* O'Reilly & Associates, Inc., 1989.

[2] *Xlib Programming Manual,* O'Reilly & Associates, Inc., 1989.

[3] Nabajyoti Barkakati, *X Window System Programming,* SAMS, 1991.

[4] James Gettys, Philip L. Karlton, and Scott McGregor, *The X Window System, Version 11,* Software–Practice and Experience, Vol. 20(S2), Oct. 1990, S2/35–67.

[5] Ralph Droms and Wayne R. Dyksen, *Performance Measurements of the X Window System Communication Protocol,* Software–Practice and Experience, Vol. 20(S2), Oct. 1990, S2/119–136.

[6] Glenn Widener, *The X11 Inter–Client Communication Conventions Manual,* Software–Practice and Experience, Vol. 20(S2), Oct. 1990, S2/109–118.

[7] David Simpson, *3 Ways to Implement X,* Systems Integration, Sept. 1991, pp 54–58.

[8] David Simpson, *Windows into Networks,* Systems Integration, Jan. 1990, pp 39–45.

[9] Kevin Reichard and Eric F. Johnson, *X Performance,* Unix Review, Vol. 9, No. 9, Sept. 1991, pp 83–89.

[10] Rita Brennan, Kevin Thompson, and Rick Wilder, *Mapping the X Window onto Open Systems Interconnection Standards,* IEEE Network Magazine, May 1991, pp 32–40.

[11] Douglas E. Comer, *Internetworking with TCP/IP, Volume I,* 2nd Edition, Prentice Hall, 1991.

[12] Ian Bacon, *Opening the X Window,* MacUser, June, 1992, pp205.

[13] Dave Kosiur, *Network Connections,* Macworld, Nov., 1989, pp152

[14] Dave Kosiur, Kee Nethery, *Network Speed Trials,* Macworld, Dec. 1989, pp169.

[15] Dave Kosiur, *On the Ethernet Highway,* Macworld, March, 1990, pp133.

[16] Dave Kosiur, *Going the Ethernet Route,* Macworld, April, 1991, pp131.

[17] Dave Kosiur, *Moving up to Ethernet,* Marworld, May, 1992, pp150.

[18] Standford Diehl, *Mac Adapters Embrace Ethernet,* BYTE, Jan. 1990, pp203.

[19] Sharon Fisher, Dave Hull, *Macs on the Internet: Talking TCP/IP,* MacUser, July, 1990, pp210.

[20] Lon Poole, *Hubs: Connecting to Ethernet,* Marworld, June, 1992, pp166.

[21] Mark L. Van Name, Bill Catchings, *AppleTalk Phase 2 and You,* BYTE, Jan. 1990, pp145.

[22] *Ethernet Interface Tests By MacUser Lab,* MacUser, June 1990.