Performance Aspects of Single-User Systems on a Local Network

Francisco Aurtenechea

M. Phil.
University of Edinburgh
November 1982
Abstract

This research is concerned with the study of performance of a local network environment being developed at Edinburgh University. The investigation covers three principal areas: a) performance of present, revised and replacement filestores, b) performance of existing communication links and Ethernet local area network, and c) the impact of network performance on overall system performance.

These areas represent different levels of protocol interaction upon network communication. The intention of this study is to ascertain the impact of these levels of interaction upon the performance of network-based services as a whole. The analysis comprises measurements like: time distributions of the client/filestore dialog through the network, different performance levels on the Ethernet local area network, and time distribution and pattern of network usage from individual clients and user applications and their relationship to network traffic.
Acknowledgements

I would like to express my gratitude

To Hamish Dewar, the supervisor of this thesis, for his constant assistance and encouragement throughout the course of this research work. He also made boundless suggestions for improving both the content and presentation of this thesis.

To Igor Hansen and Reiner Thonnes for their advice and frequent stimulating discussions during the last year of the present work.

To Vera Noethe, Pedro Hepp, Krishna Kulkarni and Brian Monohan, my room companions during the last two years at Edinburgh University, for many interesting discussions.

To my wife for her patience to tolerate my long absences from home, and to my son for providing a lively atmosphere during our two years in Edinburgh.

To the Pontificia Universidad Catolica de Chile, for the award of the studentship.

To all the people at Edinburgh University, who helped me during the progress of the present work.
Declaration

I declare that this thesis was composed by myself and that the research work described was performed by myself, unless specifically stated otherwise within the context.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td></td>
</tr>
<tr>
<td>Acknowledgements</td>
<td></td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1-1</td>
</tr>
<tr>
<td>2 Preparatory Topics</td>
<td></td>
</tr>
<tr>
<td>2.1 Network Filing Systems</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2 Local Area Networks</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3 Ethernet-type Networks</td>
<td>2-13</td>
</tr>
<tr>
<td>2.4 Performance in Context</td>
<td>2-26</td>
</tr>
<tr>
<td>3 Systems Used for Evaluations</td>
<td></td>
</tr>
<tr>
<td>3.1 Edinburgh Departmental Links</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2 Existing Filestores</td>
<td>3-5</td>
</tr>
<tr>
<td>3.3 ISYS80: A Filestore Based Operating System</td>
<td>3-9</td>
</tr>
<tr>
<td>3.4 Edinburgh CSMA/CD Local Area Network</td>
<td>3-11</td>
</tr>
<tr>
<td>4 Evaluation Methods</td>
<td>4-1</td>
</tr>
<tr>
<td>5 Performance Evaluations</td>
<td></td>
</tr>
<tr>
<td>5.1 Performance of Departmental Links</td>
<td>5-1</td>
</tr>
<tr>
<td>5.2 Performance of Departmental Ethernet</td>
<td>5-20</td>
</tr>
<tr>
<td>5.3 Performance of Filestores</td>
<td>5-47</td>
</tr>
<tr>
<td>5.4 Impact of Network Performance on System Performance</td>
<td>5-79</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>6-1</td>
</tr>
<tr>
<td>7 References</td>
<td>7-1</td>
</tr>
</tbody>
</table>
1 Introduction

The area of local computer networks has emerged as a major focus of research activity, in parallel with work in public networks. Performance evaluations of these communication systems come into prominence as a vital concern in the evolution of this important computation area. As local networks expand and evolve and as user traffic demands increase, data from performance studies becomes important evidence for design-tradeoffs and capacity planning.

This thesis reports on the evaluation of the performance of a particular local network environment. The focus of attention is to find out how different performance levels impact on single-user systems embedded in the network. Our aim is to look at total performance characteristics in an environment where diverse levels of activities interact: file management service, communication channels and protocols, client's data transfer transactions, pattern of demand from user's applications etc. By enlarging the scope of evaluations, the significance of individual levels of system performance can be better understood. The extent to which particular system enhancements affect the overall communication system performance is analysed. The intention is also to establish what time limiting factors
are involved in the local network entities that interact through the different communication pathways.

The dissertation is organized as follows:

In Chapter 2, an overview of the communication and computer systems related to the topic of this research work is presented. Details about the general context of evaluations pursued by this study are also presented here.

Chapter 3 describes the particular system environment used for the evaluations. Existing computer and communication systems are characterized, focusing attention on those issues that are relevant to the performance levels subsequently analysed.

The approach and methods employed for carrying out the performance analysis are described in chapter 4. Here, we present the main tools used to quantify the performance parameters to be analysed.

The principal topic of the thesis is described in chapter 5. This chapter, comprising four sections, outlines the series of measurements and analysis of performance carried out on the systems identified in chapter 3. The first section is devoted to describing timing experiments on the point-to-point Departmental communication links. A series of performance measurements on the Departmental Ethernet local network
is described in section two. Evaluations are aimed at revealing those network parameters that may impact the network performance. In section three, we present a quantitative evaluation of the Departmental filestore. Its performance characteristics are measured, as seen by the clients to which the filestore provides filing service. In the last section, a number of single-user systems are selected to undergo timing experiments upon individual applications to establish the way in which these relate to network resources.

Overall system performance is always kept in mind throughout the different evaluations carried out in this dissertation.
2 Preparatory Topics

2.1 Network Filing Systems

The use of Distributed Computer Systems [Boc79] has dramatically increased in the last decade. A major contribution to the operability and feasibility of distributed computation has been the incorporation of local computer networks [Met76, Cla78], and the increase of microprocessor power together with a decrease in their prices. Thus, the use of dedicated processors to implement specific functions in the network has allowed a more rational use of resources and the sharing of expensive peripherals such as disks and printers.

The handling and transport of files are a fundamental aspect of network usage. To take full advantage of the network in meeting the need for storing and transferring of files, the use of dedicated processors to provide general storage service for the network community (the clients) has become very popular. Systems of this kind are commonly known as Network Filing Systems (NFS) or Network-based File Servers [Dew78f, Mc181, Mit82, Swi79].

A network filing system is a filing system that is implemented in one or more cooperating computers.
connected by a communication network. The interfaces that a file server presents to its clients resemble those of a filing system embedded in an operating system (a co-resident filing system). In a co-resident filing system, the clients are processes resident in the same computer. By contrast, in a network filing system the clients are processes belonging to different computers, often single-user systems. A client cannot expect the same performance from the two approaches because, in a network filing system, some time is inevitably added by the network communication protocols and transmission delays through a number of interfaces. One of the questions which this thesis seeks to answer is how significant this extra delay is in relation to total response delay.

Basically, a network filing system can be seen as a repository of data on behalf of its clients, hiding the physical characteristics of the disk storage. The basic responsibilities of a NFS are to create, store, protect and administer client and system files (i.e. those shared amongst the clients), and to provide for a number of file management service operations (e.g. data transfer primitives). The main operations a file server includes are: Open a file either for reading or writing, Close a file, Read and Write entire files or specific pages within a file either sequentially or randomly, Copy a file into another, Delete a file etc.
Transactions

Typically, a network filing system encapsulates the state context of read and write request sequences in a mechanism called a transaction. In a transaction, a unique identifier coexists between the operations "open transaction" and "close transaction" and is used as a token that individualizes a sequence of exchanges (on a single file) between client and server. Also, this identifier may define a capability for an object (e.g. a file or group of bytes) which is to be interlocked in some way (e.g. single-writer multiple-reader).

Protocols

Network filing systems use higher and lower levels of protocol to govern the sequence of data and control exchanges involved in the transactions. These establish a context of dialog between the server and its clients. By means of the protocol context, clients implement request packets, usually containing the operation code, user identification and any necessary parameter (e.g. file-name). The higher level of protocols handles issues like error and flow control, the acceptance or rejection of requests from the file server, the presence or absence of acknowledgement from the client. In implementing flow control, the file server may reject a packet if there is no buffer available or may abort an already-in-process
client if that request is holding resources the server needs. The protocol may specify that no explicit packet rejection will take place, requiring the client to retry upon failing to receive a response within a certain predefined amount of time. If the communication medium is considered to be reliable, the state held by the server (e.g. data buffers) may be discarded immediately after it has sent a reply, rather than having to wait for an acknowledgement from the client to do so.

Low and intermediate tasks (e.g. transmission error checking, address recognition, retransmission, network flow control etc.), are preferably offloaded from the host processor to the hardware circuitry and firmware of the network interface. This level of protocol also defines the basic unit of transmission, which may be fixed to a certain number of bytes (as in the Cambridge Ring), or may consist of variable-sized packets (as in Ethernet).

The requirements of a NFS may range from simple file access to elaborate file management services, incorporating functions like sharing, protection, backing storage and distributed storage. These functional requirements lead to different degrees of sophistication in the implementation of a network filing system. Among the implementations of network filing systems [Dew77f, Dio80, Swi78], we concentrate on those earlier file
servers, consisting of a fixed stand-alone filing system in a single computer. These first generation network filing systems exhibit a simple file system organization with a level of generality equivalent to a conventional filing system embedded in the operating system. The principal aim of the servers of the type investigated is to provide simple but complete facilities for clients, consisting in general of minicomputer systems restricted in memory size and/or processor capacity, and with no backing storage. The centralization of filing facilities and resources is advantageous in terms of reducing overall costs and simplifying the operating systems of the clients, as well as in terms of the convenience of having files accessible from any of the clients. A system of this kind [Dew77f] was used to conduct a number of timing experiments (see sections 3.2 and 5.3).
2.2 Local Area Networks

A number of factors have recently brought into prominence a concept in computer networking termed the local area network (or LAN). Local area networks lie midway between multi-processing within one computer system on the one hand and remote computer networking on the other.

While long-haul packet switching networks span from a number of kilometers (e.g. along a city) to thousands of kilometers (e.g. across a continent via satellite), LANs interconnect from tens of meters (e.g. in a room or building) to a few kms. (e.g. in adjacent buildings). Another major distinguishing feature in local area networks is the use of high data rates (between 1 and 10 Mbps) for bit serial transmission. In addition, because the network is geographically limited, the propagation delay of the signal is very small. This ensures better control over transmissions because, since the propagation delay of the signal is much smaller than the packet transmission time, a network station (see below) can listen to the channel before transmitting in order to recognize the idleness of the channel. Again, in practical terms, an important aspect of local area networks is that they are free to choose their own transmission technology without the need to be regulated.
by a common carrier. This avoids the extra expenses incurred by long-haul networks in the need to attach to a public communication system (e.g. a telephone network). Summarising, a local area network is a network limited in geographical scope, that uses high data rates over inexpensive privately owned transmission media.

The term station used above, refers to the software/hardware network related components, whose tasks consist of providing network access control and of maintaining intermediate protocol levels between higher communication entities in the network. The terms station, network adapter and network interface will be used equivalently.

Topologies

One of the benefits of local area networks is that they can achieve their aims while using a simple pattern of node interconnection. The term node will be used for a complete set of hardware/software components attached to the network transmission medium. The unrestrained graph structure (i.e. every node connects with an undefined number of nodes in the network) commonly used in long-haul networks may be impractical in local networks. One of the aims of this topology is to balance the pattern of traffic on the network with the pattern of interconnections, in order to minimize the cost of
interconnections. This structure introduces the unavoidable cost of making a routing decision at each node a message traverses, adding complexity to the protocols of communications at each node. In a local network, which uses high bandwidth in a geographically restricted environment, such optimisation at the expense of complexity is irrelevant. Instead, the pattern of node interconnection in LANs is typically constrained to some regular structure. Figure 2.2.1 illustrates the main topologies used in LANs.

Figure 2.2.1: Local Network Topologies

a) Star  b) Ring  c) Bus

The conventional star topology although not widespread in local networks, is sometimes useful to accommodate particular environments. Here, a central node maintains a point-to-point interconnection with a number of
computers. The central node focuses and routes every message on the network. This topology imposes a number of restrictions (e.g. it depends totally on the reliability of the central node) but is useful if the pattern of communication resembles the physical connection (e.g. terminal access to a time-sharing computer). This topology may also typify the case of a community of small mini-computers which are provided with file access to a simple central file server (the central node).

The most widely used LAN topologies comprise those using a decentralized network control mechanism. These, unlike the star network, must provide a mechanism to synchronize the distribution of messages (i.e. which node may transmit at any given time). One of these local network topologies is the Ring. The Ring is a unidirectional point-to-point interconnection along consecutive nodes arranged in a circular configuration. Every packet passes around the ring one node at a time until it reaches the destination node. A number of different strategies may be used to synchronize the distribution of messages. One of these, known as the Slotted Ring, is the control mechanism used by the Cambridge Ring [Wil80]. Here, a number of fixed-size packets (message slots) circulate around the ring. Each message slot can be full or empty and a node wanting to transmit simply waits for an empty slot, marks it as full.
and places data on it. One characteristic of this type of strategy is that the size of the slot, and hence of the packet, is small.

The ring topology can be regarded as a series of point-to-point interconnections between active elements. Another approach to local network topologies is to incorporate passive elements using broadcast-based communication. This topology is widely known as the Bus architecture. Here, a shared communication medium, the bus, transports messages from the originating node to the end of both extremes of the bus. As in ring networks, bus networks must provide control of access for message transmission. The best known technique is the contention mechanism whereby every station on the network listens to the channel before transmitting. Thus, a station delays its own transmission while a signal remains on the cable: the principle of deference. Other approaches to network control can be used and are intended basically to avoid collisions and possible unfairness in channel acquisition. Nevertheless, the distributed contention mechanism is still the most widely found control mechanism in bus architectures.

Several benefits have been achieved by the use of local area networks. Some of them are:
Simplicity of components and control structures. The technologies available in LANs for high-speed bit serial transmission (e.g. twisted pairs, coaxial cable), remove the need to add hardware complexity in the network components to make the most effective use of the available bandwidth.

Flexibility and low cost in supporting distributed computation around an establishment. The relatively low cost and simplicity of the network-oriented components of a local network have facilitated and reduced the cost of tapping function-oriented computing facilities onto the network. Thus, high incremental growth in a distributed computing environment can be achieved at a cost which is basically related to the host oriented parts and software, rather than the network oriented parts.

Simplicity of Protocols. The high data rates achieved in local networks avoid the need to impose awkward restrictions aimed at minimizing the control context of a frame. The header in a frame can be used to address directly ports, processors, buffers etc., and consequently simplify the control structure of the communication protocols.

The fundamental reason for the increased emphasis on local area networks is that by supporting the requirements of functionally distributed processing, they
can enable a new approach to computer utilisation. Instead of having a large but not always reliable mainframe to support time sharing service, it becomes more economical and productive to have several minicomputers attached to the network. A group of specialised servers may provide for the extra requirements that a mini-frame could not sustain (e.g. file administration, access to expensive peripherals, running large programs etc.).
2.3 Ethernet-type Networks

The major success in the designing of broadcast-based local networks came in 1976 with the implementation, at Xerox PARC, of a system known as the experimental Ethernet [Met76]. In the late 70s Xerox, joined by Digital Equipment Corporation and Intel Corporation, developed a final specification for the Ethernet [Dix80]. This communication technique uses a simple, passive communication medium, the Ether (usually a coaxial cable), which is shared among distributed computing stations. Control of access to the ether is distributed among the stations, which delay their own transmissions while the channel is busy (i.e. a signal is sensed on the cable), and attempt transmission only when the cable is sensed idle. Nevertheless, the signal does not propagate instantaneously and two or more stations could erroneously sense the channel idle during the beginning of another transmission, seize the channel and collide. To overcome this difficulty, Ethernet incorporates the requirement for a transmitting node to continue to listen while transmitting. The detection of a mismatch between the transmitted message and the monitored signal permits colliding stations to detect the interference promptly and to stop transmitting. This technique is known as "Carrier Sense Multiple Access with Collision Detection" (CSMA/CD). Furthermore, in order to guarantee that the
duration of the collision is sufficient to ensure its detection by all transmitting stations on the network, each station uses a Collision Consensus Reinforcement procedure that jams the Ether for a short time interval. This ensures only a minimum waste of bandwidth after a collision occurs. Furthermore, following a clash the "guilty" stations defer their own transmissions for a random period of time, whose mean is incremented twice if a consecutive collision occurs. This procedure is known as the Binary Exponential Backoff algorithm.

**Ethernet Components**

One of the objectives of the Ethernet is to provide full compatibility among the universe of Ethernet implementations. To attain this, several specifications for the lowest layers of the network architecture, concerning network components and communication protocols, have been standardised [Dix80]. These standards relate to the Data Link layer and the Physical layer. No standard specification has so far been provided for higher level protocols, which the Ethernet specifications refer to as the Client layer. This functional level of the network defines the higher layers of the ISO model [Zim80] (Network layer, Transport layer etc.), and it is Ethernet's intention to provide full compatibility at that level of protocols.
The Physical Layer provides a means for transmitting and receiving serial-bit streams between the communication medium and the Data Link Layer. This function is accomplished with clock mechanisms for synchronization and timing, carrier and collision detection, serialization and deserialization of bit streams and generation and removal of preamble information (i.e. for packet synchronization).

Table 2.3.1 presents some of the standard timing specification for this layer. The complete set of timing components, (including propagation times, synchronization for all intervening electronics and signal rise time degradation), specify a total worst-case round trip signal propagation delay of 450 bit-times (equal to 45 usec. at 10 Mbps). This time also includes a maximum delay of 10.26 usec, incurred in a maximum point to point link anywhere in the system, that may be used as a way of linking cable segments, typically in different buildings, using repeaters.
Table 2.3.1 Components of Physical Channel Propagation Delay
10 Mbps Ethernet (Total 45 usec)

<table>
<thead>
<tr>
<th>Element</th>
<th>delay (us)</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transceiver cable</td>
<td>3.08</td>
<td>5.13nS/M, 2 300M paths (1)</td>
</tr>
<tr>
<td>Transceiver (transmit and receive path)</td>
<td>3.10</td>
<td></td>
</tr>
<tr>
<td>Coaxial cable</td>
<td>13.00</td>
<td>4.33nS/M, 2 1500M paths (2)</td>
</tr>
<tr>
<td>Carrier sense signal</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>Collision detect signal</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>Signal rise time</td>
<td>0.30</td>
<td>forward (carrier sense) path</td>
</tr>
<tr>
<td>Signal rise time</td>
<td>8.10</td>
<td>return (coll. detect.) path</td>
</tr>
</tbody>
</table>

(1) assuming 0.65c
(2) assuming 0.77c, c: speed of the light

The Data Link layer provides control mechanisms for address recognition, the framing of incoming and outgoing packets, frame check sequence generation and validation, carrier deference, collision handling and the scheduling of retransmissions.

By monitoring the physical channel (i.e. by watching the carrier sense signal), the Data Link layer controller defers to any passing frame by delaying any pending transmission of its own. After the end of the passing frame, the data link controller continues to defer for 9.6 usec to provide proper interframe recovery time for other data link controllers and for the physical channel.

The number of bits transmitted after a collision has been detected (the magnitude of the jam), is defined to be at least 32 (but not more than 48) bit times. This amount, added to the sum of the physical layer round trip propagation time (450 bit times), defines the upper bound
on the acquisition time of the network. This value also defines the **slot-time**, which is used as the magnitude for the scheduling of retransmissions during the contention period (i.e in the backoff algorithm). Thus, the slot-time in the standard specifications is defined to be 512 bit times -a value a little greater than the sum of the jam and the maximum worst-case round trip signal propagation time of the network.

**Performance Considerations**

Before considering the effects of contention in the performance of Ethernet, let us see what the channel utilization would be in an ideal case where one or more transmitters drive the medium as fast as it will go. We would like to know, in this case, what is the maximum channel efficiency relative to, for example, nominal 10 Mbps, for various packet sizes. Even in this case, the efficiency of the channel is below 100% since, due to the minimum spacing that must be provided as recovery time for the Data Link, the sender cannot keep sending full blast. Hence, using the following notation:

\[ I = \text{interframe recovery time, in usec} \]
\[ C = \text{channel capacity, in bits/usec} \]
\[ P = \text{the number of bits per packet} \]
\[ ME = \text{Maximum channel efficiency of the Ethernet,} \]

we can simply denote,

\[ ME = \frac{P}{C} / (\frac{P}{C} + I) \]
In the relation above, the effect of transmission errors (known to be very infrequent in Ethernet) is not considered. Different values of $ME$, for various packet sizes, are shown in table 2.3.2, according with two different specifications of the channel capacity and interframe recovery time. The first (left) corresponds to the standard Ethernet specifications and the second (right) to the current Edinburgh Ethernet specifications (see table 3.1.1). As can be seen, as the size of the packet decreases the influence of the interframe recovery time in reducing channel efficiency becomes significant.

Table 2.3.2 : Ethernet maximum channel efficiency (no contention)

<table>
<thead>
<tr>
<th>Packet Size (8bit bytes)</th>
<th>$C=10$, $I=9.6$</th>
<th>$C=2.1$, $I=3.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.571</td>
<td>0.953</td>
</tr>
<tr>
<td>32</td>
<td>0.727</td>
<td>0.976</td>
</tr>
<tr>
<td>64</td>
<td>0.842</td>
<td>0.988</td>
</tr>
<tr>
<td>128</td>
<td>0.914</td>
<td>0.994</td>
</tr>
<tr>
<td>256</td>
<td>0.955</td>
<td>0.997</td>
</tr>
<tr>
<td>512</td>
<td>0.977</td>
<td>0.998</td>
</tr>
<tr>
<td>1024</td>
<td>0.988</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Several performance analyses have been carried out to ascertain the behaviour of the Ethernet distributed network access mechanism (the contention mechanism) under normal and high loads [Sho80]. The measurements were confined to the lowest level bus contention mechanism, isolated from any possible interaction with higher level protocols and communication interfaces. The measurements confirmed the high performance predicted for the
Ethernet's network access technique. The results obtained give a useful picture of how the performance of the network is bounded at the lowest level of interfaces. Nevertheless, buffering strategies, mismatches of speed between network and host interfaces, and interaction with the higher protocols may substantially decrease network performance.

A simple model that analyses the efficiency of this low level of the network under high loads [Met76], takes into account apart from the two parameters P and C described above, the following:

T: the slot-time, in microseconds, and
W: the mean time for acquisition of the channel.

The last parameter is dependent on the instantaneous load k of the network (i.e. the number of stations ready to transmit a packet). Assuming an ideal control mechanism that independently enables each of the k stations, during the contention slot, to transmit with equal probability 1/k then, the probability A of acquisition of the channel is,

\[ A = \frac{k-1}{1-1/k} \]  

The probability that the Ether is acquired on the i-th slot (i.e. the previous i-1 slots have been wasted during
the contention period) is equal to $A^*(A-1)^*i$, whose mean 
$W$ is given by,

$$W = \frac{(1-A)}{A} \quad (2.3.2)$$

Under this assumption of a constant instantaneous load 
k, there is no idle time, only packet transmission times 
plus contention times. Then, the efficiency $E$ of the 
channel (i.e. the proportion of time the network is 
successfully carrying packets to the total elapsed time), 
can be expressed as,

$$E = \frac{P}{C} / (\frac{P}{C} + W*T) \quad (2.3.3)$$

Table 2.3.3 summarizes performance figures, obtained 
from the function $E$ above, for various packet sizes and 
numbers of continuously queued stations. These figures 
characterize a 10 Mbps network with a slot-time of 45 
usec (i.e. the standard Ethernet).

Table 2.3.3: Ethernet efficiency with continuously 
queued sources. $C=10$ Mbps, $T=45$ usec

<table>
<thead>
<tr>
<th>$k$</th>
<th>1024</th>
<th>512</th>
<th>256</th>
<th>128</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.948</td>
<td>0.901</td>
<td>0.820</td>
<td>0.695</td>
<td>0.363</td>
</tr>
<tr>
<td>3</td>
<td>0.936</td>
<td>0.879</td>
<td>0.785</td>
<td>0.645</td>
<td>0.313</td>
</tr>
<tr>
<td>4</td>
<td>0.930</td>
<td>0.869</td>
<td>0.769</td>
<td>0.624</td>
<td>0.293</td>
</tr>
<tr>
<td>7</td>
<td>0.923</td>
<td>0.857</td>
<td>0.749</td>
<td>0.599</td>
<td>0.272</td>
</tr>
<tr>
<td>10</td>
<td>0.920</td>
<td>0.852</td>
<td>0.742</td>
<td>0.590</td>
<td>0.265</td>
</tr>
<tr>
<td>32</td>
<td>0.916</td>
<td>0.845</td>
<td>0.731</td>
<td>0.576</td>
<td>0.253</td>
</tr>
<tr>
<td>64</td>
<td>0.915</td>
<td>0.843</td>
<td>0.728</td>
<td>0.573</td>
<td>0.251</td>
</tr>
<tr>
<td>128</td>
<td>0.914</td>
<td>0.842</td>
<td>0.727</td>
<td>0.571</td>
<td>0.250</td>
</tr>
</tbody>
</table>

2-20
The measurement results obtained on the 3 Mbps experimental Ethernet, for several values of P, being very close to the function E above, showed that even under extremely high loads the network utilisation at the contention level remains high and the system does not become unstable. Utilization decrease with smaller packet size is due to the fact that a greater proportion of time is lost due to collisions when packet size is small.

The network access mechanism has also been evaluated when the network is submitted to an average load (i.e. as a proportion of the network's carrying capacity). Measurement has also proved the stability of the network, that is, that network throughput is a non-decreasing function of the offered load.

Nevertheless, any particular Ethernet-like system does not match exactly the ideal 1/k control policy. In addition, subtle details in the network design can affect the efficiency of the contention control mechanism [Alm79]. Ethernet's control mechanism is unable to determine exactly the value of k following a collision; it can only be estimated. The Backoff algorithm, which doubles the discrete interval of retransmissions (i.e. the clock ticks during which a retransmission can proceed) every time a successive collision occurs, is used to approach the 1/k retransmission probability.
After the first collision, the retransmission interval is set to 2 clock ticks, after the second collision to 4 clock ticks, and so on. To ensure that no progressive collisions will occur, the granularity of the clock (i.e. the interval between ticks) has to be greater than the slot-time. This guarantees that the transmission signal of the message which seized the channel reaches all the active stations before another retransmission is attempted. If, on a particular design, the granularity of an available clock is substantially greater than the slot-time, the channel efficiency may decrease significantly from the optimal 1/k model.

Figure 2.3.1: Response Time Mean for Variants [A1m79]

The results of a simulation on a 3 Mbps Ethernet-like network, with a slot-time of 10 usec and a clock granularity of 38.08 usec, have been compared with a
pseudo 1/k scheme [Alm79]. Figure 2.3.1 shows the results obtained upon the average response time (i.e. the time between when a station desires to transmit a packet and when the packet is successfully transmitted), as a function of the average load imposed on the network, for a fixed 512-bit packet size.

Apart from the workload imposed on the network, the low level hardware and the contention mechanism, other factors influence the performance of computer systems interacting through the local network. These can be broken down into the categories of: network adapters, data-movement capabilities and network protocols. A brief discussion of each factor follows.

Under the heading of network adapters fall factors relating to the way in which hosts are interfaced to the medium. The network under investigation uses intelligent adapters, incorporating a microprocessor running under the control of firmware as well as special-purpose hardware. Some relevant considerations are: the buffering capacity, the number of ports, the cpu capability, the I/O channels, and the extent of custom hardware that can relieve the cpu of individual processing tasks. The buffering capacity is the space the station uses to store incoming packets either arriving from the host or off the ether. If the rate at which packets arrive at the station is high, the buffer
capacity can be exhausted. In this case, delays are introduced through the source station having to undertake packet retransmission (i.e. after a predefined amount of time without receiving an acknowledgement). The number of ports in the network interface defines the number of virtual circuits that a station can concurrently maintain with one or a number of other stations in the network. The lack of ports in a station supporting high data traffic can cause significant delays at the source stations, for they could have to repeatedly retry their own transmissions waiting for the availability of ports at the destination station. The CPU resource is divided among the activities that are concurrently running in the low-level firmware of the station. A slice of time is given to each active port in order to ensure that each of them makes forward progress. The throughput of the adapter is bounded by the speed at which these activities are carried out (i.e. the speed of the CPU). Particular function-oriented hardware may relieve the CPU of certain processing and I/O tasks and consequently increase the throughput of the adapter in proportion to the task that is offloaded. Examples of such tasks are: address recognition (which relieves the controller of processing every passing packet), CRC generation and checking, and the transmit deference procedure.

Host-to-host communication performance is also affected by the data-movement capabilities available at
both the station and host levels. The total message delay can be significantly increased by the time it takes a packet to be passed across the interface with the host. At this level of communication, the time involved in Input/Output activities is dependent upon the speed of the cpu in both the host and the station, and the I/O devices involved, which include: a) the transfer method the host uses to communicate with external devices, b) the Host/DMA interface that physically connects the station with its host, and c) the speed and mode of operation of the DMA at the network interface. The parallel interface as such can operate at any convenient rate to pass data bytes to/from the host. The DMA usually does not operate with the host in burst mode, which means that every byte transferred must be granted by the cpu (i.e. on a cycle stealing basis).

The higher levels of communication protocols of the network embedded in the controller firmware make use of most of the above mentioned network resources to enable the communication transactions to be carried out. This control mechanism basically accounts for overheads like packet building time, virtual circuit coalescence, retransmissions, acknowledgements etc. The performance of the network, as seen by a host, can be heavily influenced by the amount of activity required at this level (i.e. how many instructions are executed to carry out each task).
2.4 Performance in Context

In a communication network, diverse computers interact with each other through the network components, typically network servers giving resource access to the clients, which in turn support the users. Users have the last judgement in the performance achieved by the application programs requesting resources through the network.

Every request/response dialog between a pair of hosts in a network requires that a number of packets (perhaps including acknowledgements) flow in both directions, with the average message turnaround time depending on a large number of factors concerning configuration, traffic and strategies employed. These factors include: the communication medium, intercommunication interfaces, usage pattern of the network, buffering strategies, end-to-end protocols, cpu(s) involvement etc. The overall performance of the network is dependent upon these different network constituents and upon their mutual interaction. To the extent that these factors can be identified and taken into account, the performance analysis may better quantify which are the critical components that may degrade network performance and which are irrelevant.

In a local network, the end-to-end protocols
correspond to a multilayer structure that enables data exchange between any two hosts in the network. Every layer supports specific functions and through interlayer interfaces ties up with higher level protocols until finally user applications are reached. Concerning this level of communication, some questions arise:

- How is network performance affected by the buffering strategies and transfer protocols through the various functional layers supporting the dialog between two entities in the network?

- How sensitive is network performance to the kind of operations performed by the low-level firmware or software which controls the network traffic, and the speed of the processor(s) that execute(s) these instructions?

- How important is it that certain processing and communication activities can be off-loaded into function-specific low level hardware (e.g. address recognition)?

Secondly, as has been discussed above, the use of high bandwidth for bit-serial transmission is one of the major attraction of local networks. To the techniques already in use (e.g. twisted pairs, coax cable), are likely to be added techniques that make more bandwidth available, like
CATV and fibre optics. This leads to the posing of some further questions:

- How significant is bandwidth in the total delay of a simple request/response dialog in a process-to-process communication through the network?

- How does the message overhead (i.e. headers, acknowledgements) affect the total end-to-end delay?

Thirdly, we may consider factors outside the network itself that directly or indirectly affect user performance, like the applications programs using the network, host configurations and interfaces to the network. The efficiency of the lowest level protocols of the network does not guarantee fast response. Messages over the network have to deal with potential mismatches of speed at which data is processed within the different levels of the network. For example, a request that arrives at a shared server (e.g. a file server) may require that one or more processes have to be rescheduled, or may have to wait for the availability of disk controllers etc. This delay can be significant as the load on the network increases. Hence, for a particular configuration, it is relevant to establish,

- What is the mismatch between the rate at which data is delivered to the host, and the rate at which the
host can absorb it? Which are the time limiting factors?

- In a typical read or write request, what percentage of the average response time is due to the server CPU involvement, and what part corresponds to disk activities (seek time, latency time etc.)?

- To what extent can the performance of application programs through the network be improved by modifying strategies of data transfer within the file server or the client system?

- How is the response time perceived at the terminal influenced by the configuration of the client, such as size and management of memory and I/O capabilities? How does this affect the total response time distributions of identical user applications on the network?

- How do these configuration aspects impact on the traffic imposed over the network?

- For a typical application how significant is local computation versus I/O activity?

Throughout this investigation, an attempt has been made to keep these overall factors in mind in order to
provide a perspective and framework for the relevance of the evaluations carried out. It is of course difficult to discern all the relevant factors that may influence overall performance, because of the wide variability of application systems, intercommunication interfaces and traffic likely to exist in a distributed environment supported by a local network. However, we have been able in the course of this research to isolate a number of the factors mentioned above and carry out systematically varied experiments which have made it possible to reach conclusions about the importance of some of them.

In some cases the findings were entirely as predicted. In others, they were surprising, indicating either that the a priori analysis had been too naive or, in a few cases, that a system component was not functioning as its designers had intended.

It is also hoped that the particular results presented and the tools developed may enable others to derive information on aspects of particular interest to them.
3 Systems Used for Evaluations

This chapter describes the environment in which evaluations have been carried out, namely the local area network being developed in the Computer Science Department at Edinburgh University.

Figure 3.1 illustrates the configuration comprising the computer systems and communication media currently integrated in the network. We selected for evaluation all combinations of communication media and interfaces, and a subset of the computer systems comprising the application environment of the local network. The narrative which follows is a description of each pertinent configuration.

3.1 Edinburgh Departmental Links

Departmental links at Edinburgh University are interfacing techniques which enable the interconnection of computers and the attaching of peripherals, using serial asynchronous character transmission along a single standard coaxial cable. Basically, a Link consists of a pair of independent ports within one device—a transmitter and receiver pair.
Figure 3.1: Local Area Network Configuration (EUCSD)
The first Departmental link was conceived in early 1971, and its design was principally aimed at providing a basis for peripheral resource sharing for an increasing number of machines in the Department. Through time, these links became the principal method of attaching new peripherals, and of inter-connecting processors on a point-to-point basis [Lnd76].

The main feature of the links is automatic flow control at the character level, whereby the speed at which a recipient accepts bytes from one end determines the rate at which they may be inserted at the other.

Timing

The data format is similar to the serial asynchronous character standard used by teleprinters. It consists of one start bit, 8 data bits, a parity (or 9th data) bit and a stop bit. A basic diagram for the transmission of a character is shown in figure 3.1.1. Flow control is implemented by an acknowledge pulse in the reverse direction lasting a minimum of 1 bit-time and starting at least one bit time after the middle of the last bit of the transmitted character. Another character can be transmitted after a minimum of one half-bit time following the acknowledge pulse. The length of the acknowledge pulse may be arbitrarily long due to delay in disposing of the character. The minimum transmission
time in the absence of delay is thus 12 bit times.

![Timing Diagram](image)

Earlier versions of the link operated at 256 Kilobaud (0.26 Mbps) and more recent ones at 1 megabaud (1 Mbps). This has provided a way of isolating the effects of pure bandwidth considerations for this mode of connection.
The first Network Filing System implemented in the Department, the Departmental filestore [Dew77f], was a simple file server, aimed basically at providing a set of primitives for data transfer and file management, including a protection mechanism and naming of files. The local area network for communication consisted of a star configuration with the filestore as the central node, providing point-to-point communication with a number of clients via Departmental links (see figure 3.1). The filing system was subsequently rewritten to simplify its structure and to accommodate the incorporation of Departmental Ethernet links as another communication medium. Both the old and new filestores, which were the subject of timing experiments, have similar characteristics regarding related hardware, protocols and disk organisation. The description that follows outlines these common features.

The filestore was written using the HAL assembly language [Dew78], and runs on an Interdata minicomputer with 64 Kbytes 16-bit words of memory. The permanent storage comprises two disk units, using CDC 9762 drives with a formatted capacity of 67.4 Mbytes each. The disk units have an average access time of 30 ms, an average latency time of 8.3 ms and a transfer time of about 1.2
Protocol

The filestore is a passive server, and to access its services a client sends a request message (normally a command line), which elicits a response message in the opposite direction. Thus every operation requires a simple two-way exchange: request from client to filestore, response from filestore to client. In the case of a Write command (that is, the client writing data to a file) a sequence of data bytes (usually 512 bytes) accompanies the request message, and similarly a Read command elicits a response which is accompanied by the data. For every opened file, a temporary unique identifier, a transaction number, comprises the basic context by which clients access files. Each transaction number represents an object, the file, which is interlocked in the standard multiple-reader/single-writer fashion. While an individual request/response dialog is in progress, the same client cannot activate another filestore request. However, in the meantime, exchanges with other clients can be carried out, but any read/write request will be restricted by the availability of the disk controller. Each operation runs to completion before the filestore acknowledges it, except that the Write operation is acknowledged once the data from the client reaches the filestore and does not wait for the Mbits/sec.
disk write to be completed.

**Organization of the Disk Space**

Each disk is divided into four partitions that hold different kinds of file information. The first partition (small), holds core-image loadable programs, the second and fourth partitions (large) are used for the actual file storage. The third contains the directory blocks and bit-map tables. Each partition can store up to $2^{16}$ blocks, a **block** (512 bytes) being the basic unit of a partition. Each block is in exactly one file and each file is contained in a single partition. Directories are 4 disk blocks (2048 bytes) each, only one being in store at a time. They are identified by indirection through a resident owner register table, using an owner-number to determine disk and owner.

A file consists of a number of adjacent areas called **extents**, an extent being the basic unit of space allocation. When a file is being created, the first extent is allocated at the time that the file is opened, and extra extents are allocated as the file expands beyond extent boundaries. Any overallocated space in the last extent is deallocated when the file is closed. The allocation of extents involves searching and updating a bit-map table. One bit-map block (512 bytes) is held in memory at a time, and to each disk block corresponds a
single bit in the bit-map table. Thus, each 512-byte bit map block accounts for 4096 disk blocks (the largest possible extent). Hence, to cope with the maximum number of files in a partition ($2^{16}$ blocks), 16 bitmap blocks are required.
3.3 ISYS80: A Filestore Based Operating System

ISYS80 is an Operating System, implemented in the Department [Dew77i], for the Interdata 16-bit mini-computer family. The system, which is essentially a filestore-based system, provides the usual functions of a mini-computer multi-tasking operating system.

Although some of the commands are available to any user (i.e. any person) not logged on, to have full access to all the filestore capabilities and system files, the user has to be accredited as an 'owner' and requires to logon to the filestore. A user-session involves all the operations between the commands logon and logoff.

The user usually starts the session by editing some of his files. This elicits a call to the Isys80 loader to load the system editor (approximately 10 blocks) from the filestore to the client's memory. The capacity of the Isys80 machines for storing files to be edited (equal to the capacity of the client's memory background partition minus the size of the Isys80 editor) is in general restricted. An edit session normally involves operations in three files: the file being edited (old file), the file being created (new file), and a temporary file. If the client's memory becomes full then, whenever a block not held in memory is required either forward or backward
in the file, a block needs to be written onto the new file or the temporary file respectively.

The user session normally continues by invoking one of the utility programs, assemblers, micro assemblers or compilers, which will usually take the file just edited as the input file.

The HAL assembler used on the Interdatas is two-pass, with the source file being read twice. The code size is 21 blocks and it requires to read a 1 block predefinition file as well as the user's program. Assemblers for the Departmental micro-processor kits are also available.

The main high-level language supported is IMP. The Compiler is multi-pass with temporary files conveying information between the passes.
3.4 Edinburgh CSMA/CD Local Area Network

Edinburgh's CSMA/CD local network is a 2.1 Mbps Ethernet-like network. Its implementation will gradually replace the current point-to-point communication facilities in the Department. The network is aimed at accommodating the integration of diverse independent single-user machines and a number of function-oriented computing facilities (e.g. remote compilation). This communication system will constitute the ground for future hardware and software experimentation and design.

The Network Controller

User machines (hosts) are interfaced to the network by intelligent controllers (ethernet stations). Each station consists of a single-board device assembled around a 4Mhz Z-80 processor with 4K bytes of PROM supporting the firmware and 4K bytes of writable memory handling 6548-bytes buffers to accommodate the exchange of incoming data from the Ether and from the host. A 2Mhz 4-channel Direct Memory Access (DMA) controller handles the bulk of the data that is transferred to/from the ether transceiver and the host. Here, data is transferred up to 0.5 Mbytes/sec through channels designated: Ether-Receiver, Ether-Transmitter, From-Host and To-Host. The exchange of data between the Ether and
the station is carried out through FIFO buffers, which reduce the effects of any delays in access to the station data bus. The receiver FIFO comprises 64 bytes, enough room to give the controller time to set up new buffers for packets arriving closely one after another. The transmitter FIFO incorporates 16 bytes, which provides a suitable window for any irregularities in transfer on the bus. The tasks of handling checksum, address recognition, transmit deference and serialisation and deserialisation of bits, are offloaded into custom hardware.

Host Interfaces

The interface between the host and the station is a 9 bits wide parallel bidirectional interface, driven by the user as two independent unidirectional 9 bits wide logical channels—a transmitter and receiver pair. The 8 least significant bits (0:7) are used for information data and the ninth bit (bit 8) to distinguish between data bytes and control characters. Examples of currently available interfaces are: parallel Interdata mini-computer interface and a single channel DMA interface connected to a PDP-11 UNIBUS.

Network parameters

All the fundamental facilities and control structures
inherent to Ethernet networks (see section 2.3) are included in Edinburgh's Ethernet. However, it differs from the proposed Ethernet at Xerox in a number of network parameters and functional specifications. A summary of various specifications on both networks, is given in Table 3.4.1.

Table 3.4.1. Xerox and Edinburgh CSMA/CD LAN.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Xerox</th>
<th>Edinburgh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Rate [Mbps]</td>
<td>10</td>
<td>2.1</td>
</tr>
<tr>
<td>Min. packet size [bytes]</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Max. packet size [bytes]</td>
<td>1500</td>
<td>548</td>
</tr>
<tr>
<td>Header [bytes]</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>CRC [bytes]</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Interframe spacing [usec]</td>
<td>9.6</td>
<td>3.0</td>
</tr>
<tr>
<td>Address recognition</td>
<td>Data Link Cont. Hardw. Filter</td>
<td></td>
</tr>
<tr>
<td>Checksum computation</td>
<td>Data Link Cont. Custom Hardware</td>
<td></td>
</tr>
<tr>
<td>Transmit deference</td>
<td>Data Link Cont. Custom Hardware</td>
<td></td>
</tr>
</tbody>
</table>

Compared with the Xerox et al. proposal, Edinburgh's Ethernet presents the disadvantage of having a lower data rate (2.1 Mbits/sec), and a lower maximum packet size (548 bytes). This difference can be manifested in a lower achievable throughput in presence of high load conditions. However, Edinburgh's Ethernet, being comparatively a lower scale network, allows a lower interframe recovery time, that could partly compensate this lower channel utilisation (see Table 2.3.2). On the other hand, the fact that on the Departmental Ethernet, some tasks are offloaded into special function-oriented low level hardware, relieves the processor from many of the tasks of the Data Link layer. This offloading
potentially enhances the realisable power of Edinburgh's Ethernet, because the processor is able to absorb the overhead of higher level service from the host.

**Services**

The Network offers several different kinds of services through a set of dynamically assignable ports. These enable different contexts of interconnectivity to coexist between the station and its attached host, and also define the kind of internetwork communication between the hosts. Two principal modes of information exchanging between host-station entities in the network are available: virtual point-to-point and broadcast.

a) **Point-to-point**

This service establishes a liaison between any two (station, port) pairs in the network. Two different services are distinguished here: single node to many nodes and single node to single node.

The first service is offered solely by a permanent open port (port 0), and the second type offered by any of the other available ports on the station (1:15). On both services, packets received from the network are immediately acknowledged on a station-to-station level and the resulting data frame delivered to the host. When
transmitting, the port maintains in its context the number and spacing of (possible) retransmissions. The transmitting station sends an acknowledgement to its attached host upon receiving the station-to-station acknowledgement. Apart from these similarities the two services are distinguished in some operational aspects.

a1) Single node to many nodes service

This service can be characterized as a datagram service. The only port that gives this service, port 0, maintains no context concerning the destination of messages submitted by the host. Hence, every submitted message must contain the destination address (6 bytes). For every packet received from the ether, no check is performed on the source address (6 bytes) which is passed to the host together with the data.

a2) Single node to single node service

This service can be regarded as a virtual circuit connection between two nodes in the network. It is accessed by opening one of the available ports (1:15) and by passing to it the remote address (node and port) to which future data exchanges refer. The remote address is held in the context of the port for subsequent data exchanges addressing this port.
A special form of this service permits a host to treat its Ethernet interface as a direct connection after establishing a virtual circuit. In this mode no protocol information is included in the messages passed, these being simply data byte streams. This mode, usually called Protocol-less, has been implemented mainly to simplify the software implications of switching to Ethernet communication from direct connection. Using an available buffer space of 532 bytes, the station frames the incoming data implementing a time out scheme to recognize the end-of-frame. Protocol-less mode is completed once another control character addressing the same port, is passed by the host to the station.

b) Broadcast

When using this service, a packet is received through port 0 by every station on the network. A 256-bit masking register held on the context of port 0, is used to enable or disable the final delivering of the received packet to the host. A broadcast packet can be sent through any port (0:15), and is recognised solely on the basis of the first byte of the packet (destination address) being 0. Unlike the other services, there is no station-to-station acknowledgement of broadcast packets.
Host/Station Traffic

The transfer of data frames between the host and the associated port in the station, is accomplished by means of a transaction mechanism. This consists of general and port specific control exchanges of fixed size and format, and data streams. All services above described, except Protocol-less, use the same protocol of transactions.

Figure 3.11.1: Transactions for Information Exchange

<table>
<thead>
<tr>
<th>a) transmitting</th>
<th>b) receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>station</td>
</tr>
<tr>
<td>( ------dtx.m------ )</td>
<td></td>
</tr>
<tr>
<td>( &lt;------rdy.m------ )</td>
<td></td>
</tr>
<tr>
<td>( ------stx.m------ )</td>
<td></td>
</tr>
<tr>
<td>( ====data==== )</td>
<td></td>
</tr>
<tr>
<td>( ------etx------ )</td>
<td></td>
</tr>
<tr>
<td>( &lt;----ack.m------- )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4.1 illustrates the protocol of a successful transaction between the station and its host, when transmitting and receiving messages. The control exchanges: dtx (transmit data packet), rdy (ready to accept a data frame), stx (start of data frame) and ack (packet successfully delivered) represent port specific (port m) control characters. The etx (end of frame) control exchange is a port independent control character. Data consist of up to 532 bytes, and when transmitting using the datagram service (through port 0) the remote
address (6 bytes) must precede the stream of data. When receiving, the transaction may be rejected by the host, by sending a positive acknowledgement upon reception of dtx. When transmitting from the host, the station sends the acknowledgement to the host upon reception of the station-to-station acknowledgement for the transmitted packet.

**Controller firmware**

Most of the protocol handling required for data transfer across the Departmental network is provided by the controller's stored program (firmware) embedded in the network interface. Current operations at this network level can be summarized, briefly, as follow.

a) Packet arriving off the ether:
- Validates the packet: Reads the packet status byte (generated by the low level hardware) to ascertain the occurrence of any error condition (collision, overflow, checksum error); reads the packet header checking the internetwork protocol parameters (addressee port, destination address, sequence number, packet's type). Sends a NAK (negative acknowledgement) in case an error condition arises.
- Reprograms the DMA (Ether-receiver channel) with a free buffer, to enable storing of subsequent incoming packets from the Ether.
- Links the packet into the appropriate queue, to be
subsequently delivered to the host.
- If the arriving packet was not a broadcast or an
acknowledgement packet, sends an acknowledgement
packet to the source station.

b) Interrupt from the host:
- If the interrupting character is a control
character, checks it for compliance with control
conventions.
- If the character is a DTX, sends RDY, waits for STX
and the subsequent data frame
- Upon reception of the frame completion signal (ETX,
time out or buffer overflow), assigns the frontmost
buffer in the output queues to the To-Ether channel
and enables it. Waits for an acknowledgement or
retransmits the packet if the time out condition
arises.

Between each of the above operations, a series of
miscellaneous activities must be carried out to set up
proper conditions for operation (i.e. claiming of I/O
channels, enabling or disabling the DMA, rescheduling of
processes, finding active processes, updating the context
of active ports etc).
4 Evaluation Methods

Here, we describe the main feature of the approach used to carry out evaluations of the performance of the computer communication environment formerly outlined.

Dealing with evaluations on a number of interacting systems requires the understanding of each performance level in some detail. The degree of specification at each performance level should be in relation to the significance that it has on the performance of successive system levels interacting with it.

If the system being evaluated is complicated, one approach to reproduce its performance levels is to develop simple functional specifications of the most important operations involved. The use of simple models to represent very complex systems finds increasing applications in computer system performance work [McF73]. If the system to be evaluated is relatively simple, then adding complexity to its specifications would probably hide the relevant performance issues being investigated.

The systems upon which evaluations were undertaken in this research work represent rather simple computing systems whose individual behaviour is relatively well
understood. Our task is to carry out evaluations on each of them, aiming to reproduce those performance issues that may have a real impact upon overall system performance (i.e. at the user level). Our purpose is to obtain a proper identification of the gravity of a number of performance parameters involved in computer system communication in a local area network.

An important part of the evaluation tasks is based on direct measurements upon the actual implemented systems. The availability of different versions of similar computer and communication systems enabled us to carry out direct measurements to ascertain real system behaviour under changes either in software or hardware.

The measurement data obtained were validated using very simple analytical specifications which were deterministic representations of the fundamental operations involved. They were used to improve the understanding of the systems being evaluated and to provide a mechanism for system evaluation under some proposed modifications to the actual system strategies or the incorporation of new facilities. Previous performance analysis [Alm79, Mar81] has, in some cases, been used to predict and reproduce performance levels of particular systems being evaluated.

A simple but fundamental task when undertaking
performance measurements concerns the choice of the timing devices. Measurement tasks imply dealing with the level of accuracy of the measurement data obtained. The accuracy of the devices used to carry out timing experiments will depend on the level of timing granularity to be inferred from the system being measured. For example, when measuring response times at the terminal level in operations occupying a number of seconds, time measurements can be carried out by simply using an accurate chronometer. In activities expiring in time magnitudes of the order of milliseconds or microseconds, a more specialised device needs to be provided. If operations can be measured repeatedly a number of times, a rough timing device can still be used provided the total elapsed time of the repeated process can fit in the granularity of that device.

The tasks of timing the different operation levels of the systems being evaluated, and of synchronizing certain events, were carried out by incorporating software implemented clocks at the client level. These consisted of programmed loops of simple machine instructions controlled by the protocol of the event being timed (e.g. a real time counter or a device status flag). As timing experiments were required to be measured from different machines, clocks had to be provided for any of them. It turned out that the same clock program even running on similar machines can render substantially different
results due to subtle differences in the instruction timing between the systems. Hence, each clock requires to be calibrated against another timing device (e.g. an accurate chronometer). This can be done, for example, by providing a stand-alone program and a chronometer, and using the control console switches interacting with the cpu. These can be used to start the clock (loop) approximately simultaneously with the chronometer. After a sufficient real time span has elapsed (e.g. 60 seconds of chronometer time), the cpu is halted setting up the clock's counters which can be used to gauge the loop against the chronometer time. Repeating this procedure a number of times on each pertinent machine, proper adjustment factors for each individual machine (i.e. for each clock) can be derived.

On any experiment, different clocks can be used to check the accuracy of the results being measured. We used this method on most of the timing experiments, provided that external clocks (i.e. events running on one machine being measured on another machine) can be incorporated by using the facilities of point-to-point connection available on the Interdata local area network. If the elapsed time of the event being measured is small, care must be taken to include, if the case arises, the time it takes the signal to propagate through the links that connect machines with external clocks (see section 5.1).
5 Performance Evaluations

5.1 Performance of Departmental links

This section is concerned with evaluation of the performance of the direct co-axial link connections (see section 3.1), which were initially used on the Interdata mini-computer network. The purpose is to measure the byte transmission overhead due to the line handling protocols over the Departmental link interfaces. Data measurements obtained are then used on subsequent evaluations of client/filestore communication levels.

Evaluations incorporate two different performance levels: clients and communication interfaces. The client level comprises two different type of Interdata processors (74 and 70), and the communication level one topology: star (see figure 3.1). In this topology, clients connect directly to the filestore using coaxial cable interfaced with 1 Mbps links (fast). Currently, 10 pairs of these links are available for direct point-to-point connection with the filestore. The small machines (16kb) connect to the filestore via a multiplexer, most of them using 0.26 Mbps links (slow). The multiplexer, an Interdata 70, which stands beside the filestore, multiplexes file access requests from the small machines to the filestore, and also provides them
with assembly and compiling facilities. Point-to-point communication between any two Interdata clients is also available by making explicit connections using separate links (fast or slow).

**Point-to-point connection between clients**

The first experiment is concerned with obtaining performance figures on the transfer of data between pairs of client machines.

**Description of the experiment**

Basic transmission speed evaluations were conducted as follows: two nodes were connected through their link-interfaces, one being the transmitter and the other the receiver. A third node, connected to the receiver, was used as a clock (see chapter 4). Communications between two Interdata 70s, and two Interdata 74s were selected for the evaluations of 1 Mbps links. Interfaces with 0.26 Mbps links were available only on Interdata 74 machines. Clocks were selected at random from any of the Interdata machines. To estimate the 1-byte transmission delay, a 1 Mbyte transmitter-to-receiver data exchange was used.

Several timing experiments were made on both link interfaces. On detecting the arrival of the first byte
transmitted, the receiver signals the clock machine to start timing. Thereafter data bytes are received at full speed until the last byte arrives, at which point the receiver signals the clock to stop.

At the host level, two different methods of programming were used for byte transmission: Programmed Loop (Pl) and Block Transfer (Bt). In the first method, the processor executes a loop made up of simple Interdata Input/Output instructions. For each byte transferred there is an inner loop sensing status to detect that the interface is ready to accept/deliver another data byte, then the instruction to carry out the actual transfer of the data byte, then the housekeeping instructions required to count bytes. In the second method, the processor makes use of the Interdata's Block Transfer instructions. These are single non-interruptable instructions which provide for the transfer of complete blocks of data between an I/O device and memory, and are effectively a micro-coded version of the programmed loop.

**Expected Times**

Table 5.1.1 shows the processor per-byte execution times, in usec, for both transfer methods above mentioned, computed on the assumption that the transmitter/receiver device is always ready. These values may vary from one machine to another due to
possible speed dissimilarities on particular instructions. $T_{tx}(M)$ and $T_{rx}(M)$ correspond to the byte-transfer time at the transmitter and receiver end respectively, when using the method of transfer $M$.

Table 5.1.1: Processor time for byte-transfer (usec)

<table>
<thead>
<tr>
<th>Processor</th>
<th>$T_{tx}(P1)$</th>
<th>$T_{rx}(P1)$</th>
<th>$T_{tx}(Bt)$</th>
<th>$T_{rx}(Bt)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I74</td>
<td>20.5</td>
<td>22.3</td>
<td>3.50</td>
<td>3.50</td>
</tr>
<tr>
<td>I70</td>
<td>16.5</td>
<td>16.8</td>
<td>3.00</td>
<td>3.00</td>
</tr>
</tbody>
</table>

The expected byte-transmission time of the medium, in microseconds, for link $L$, is given by,

$$T_{link}(L) = \frac{12}{B_{dr}(L)}$$

where $B_{dr}(L)$ is the transfer rate of link $L$ in bits per second. According to the speed of the links (0.26 and 1 Mbps), the expected byte-transmission time, in microseconds per byte is,

$$T_{link}(Slw) = 45.45 \quad (5.1.1)$$

$$T_{link}(Fst) = 12.00 \quad (5.1.2)$$

Where $Slw$ and $Fst$ stand for the slow and fast links respectively.

These values represent the minimum byte transmission
time of the medium. In practice, this will be extended by any delay on the transmitter in sending the next character, or the receiver in disposing of the acknowledged character. At the lowest level these processing overheads depend on the basic speed of the processor involved and the method of programming as discussed above.

Departmental link interfaces, while transmitting a byte along the cable, can buffer another byte from the transmitting host. Consequently, transmission and reception times overlap when more than one byte is transferred between a pair of hosts connected via Departmental links. A time diagram illustrating the timing in the transmission sequence of n bytes, is shown in figure 5.1.1. Hence, apart from the first two time units and the last two, the per-byte time should be the maximum of the three relevant times, that is:

\[ Teed(M,L) = \text{Max} \{ Ttx(M), Tlink(L), Trx(M) \} \]  

(5.1.3)
For packets of any length, the extra start and end components are negligible.

According to relation 5.1.3 and table 5.1.1, we can infer what end-to-end delays are expected when using the different software/hardware configurations above mentioned. Table 5.1.2 gives a summary of some of these expected times, in seconds, when transferring 1 Mbyte. The figures, can also be regarded as the byte-transfer time in microseconds.

Table 5.1.2: End-to-end Delay (expected values)
1 Mbyte transfer times (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Teed(Pl,Slw)</th>
<th>Teed(Bt,Slw)</th>
<th>Teed(Pl,Fst)</th>
<th>Teed(Bt,Fst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I74</td>
<td>45.45</td>
<td>45.45</td>
<td>22.30</td>
<td>12.00</td>
</tr>
<tr>
<td>I70</td>
<td>45.45</td>
<td>45.45</td>
<td>16.80</td>
<td>12.00</td>
</tr>
</tbody>
</table>

Experimental Timings

The measurement data obtained from the timing experiments are shown in table 5.1.3. These figures represent average times whose corresponding standard deviation were almost negligible.

Table 5.1.3: End-to-end Delay (Average observed values)
1 Mbyte transfer times (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Teed(Pl,Slw)</th>
<th>Teed(Bt,Slw)</th>
<th>Teed(Pl,Fst)</th>
<th>Teed(Bt,Fst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I74</td>
<td>53.3</td>
<td>45.5</td>
<td>22.2</td>
<td>12.3</td>
</tr>
<tr>
<td>I70</td>
<td>----</td>
<td>----</td>
<td>17.8</td>
<td>12.1</td>
</tr>
</tbody>
</table>
As can be seen, most of the observed times match fairly close the expected times shown in table 5.1.2. The small mismatch observed in the case when driving fast links using a programmed loop at both ends (Pl,Fst), is due to differences between the actual and nominal values of the instruction times included in the loop. When links are driven using Block Transfer instructions at both ends, the transmission time of the medium represents the maximum time. In these two cases, the observed times closely matched the expected times for the speed of the links (see relations 5.1.1 and 5.1.2). A significant difference was observed in the case when the slow links were driven using a programmed loop (Pl,Slw). This can be explained as follow:

Before each byte is delivered to the receiver, a delay is introduced at the loop control that is testing the device status flag that signals the byte reception. This loop comprises two instructions corresponding to the sequence: "sense status" and "branch if not available". The extent of the delay depends on the degree of synchronization that exists between when the byte is ready to be read from the link interface, and when the device status flag is checked. If the device status flag is checked just when the character is ready, no extra time is lost. Otherwise, the extra delay can be as much as the loop time. In addition, further time is inevitably wasted between the branch and the point in the Read Data instruction at which the byte is actually
transferred. Considering the instructions used in the experiments we can estimate that, for the Interdata 74, this total delay (i.e. including the loop control time) lies between a minimum of 7.5 to a maximum of 11.5 usec, which explains the difference between observed and estimated times in this case. In fact, a similar delay is also produced when using Block Transfer instructions, but this is much smaller because in this case the sensing status loop is microprogrammed. Hence, in relation 5.1.3, there should be a term representing this extra time; although it would be almost impossible to estimate it a priori. This should be small in the case that the processor time at the receiver end is greater than the speed of the medium.

Client/Filestore Communication

In this part of the investigation, the evaluation of the performance of Departmental links was carried out by undertaking direct time measurements on the different client/filestore configurations. Both the old and new versions of the filestore were evaluated to ascertain the impact of different data transfer methods on the end-to-end delay for the typical case of 512-byte packets.
Experiment Description

The single specially-programmed client used to evaluate the file management service at the filestore (see section 5.3), was used to obtain the actual performance figures upon client/filestore block transfer times.

Block transfer times to/from the filestore were measured as follows: once the request to the file server (i.e. Read or Write) was set up, and as soon as the first data byte was ready to be received (sent) from (to) the filestore, an external clock was signalled to start timing. Once the last byte of the block was received (sent), the clock was signaled to stop. Average times were obtained by measuring block transfer times on several successive Read or Write requests.

Expected Times

The end-to-end transfer delay concerning the transit over the two client/filestore configurations, namely the time it takes a block to be transfered from the client's buffer to the filestore buffer (or vice-versa) is given by,

\[ T_{\text{blk}(D_1)} = 512 \times \max \{ T_{\text{cl}}, T_{\text{link}(CF)}, T_{\text{fs}} \} \]  
\[ T_{\text{blk}(M_\text{x})} = 512 \times \max \{ T_{\text{cl}}, T_{\text{link}(CM)}, T_{\text{mx}}, T_{\text{link}(MF)}, T_{\text{fs}} \} \]
where, Dl and Mux stand for Direct link and Multiplexed connection respectively.

The relations above are specifications of relation 5.1.3. CF, CM and MF represent the type of link at: the client/filestore, client/multiplexor and multiplexor/filestore respectively. When transmitting, Tcl and Tfs resemble time Ttx at the client and filestore respectively, and when receiving they resemble time Trx (see relation 5.1.3). Tmx represents Trx plus Ttx at the multiplexor since it first reads bytes from one end of the link interface and subsequently writes these to the other.

Currently, clients make use of the tight programmed loop, described earlier, for the exchanging of data between the memory bus and the links. Byte transfer in the old version of the filestore was accomplished by using a special feature of the Interdata called Automatic I/O service, which allows interrupts to be serviced automatically. The new version of the filestore utilises a form of programmed loop incorporating a time-out scheme. This avoids the possibility of the filestore being hung up in case a client should crash or otherwise mis-behave while data transfer is in progress. Data blocks through the multiplexor are transferred to/from the filestore at full speed (i.e. without buffering), using a programmed loop. Thus, every byte read either
from the client or the filestore is delivered to the parallel interface as soon as the latter is available to receive (i.e. it has handed over the previous byte). This explains the fact that, in relation 5.1.5, Tblk(Mux) also correspond to a maximum time rather than to accumulated values.

Table 5.1.4, gives a timing summary of different values of Tfs, Tmx and Tcl. They represent actual byte transfer times incurred at the corresponding processors.

Table 5.1.4: Timing summary

<table>
<thead>
<tr>
<th>per byte per block</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[usec] [msec]</td>
<td></td>
</tr>
<tr>
<td>Tcl</td>
<td></td>
</tr>
<tr>
<td>20.00 10.40</td>
<td>Programmed Loop (read, I74)</td>
</tr>
<tr>
<td>18.25 9.50</td>
<td>Programmed Loop (write, I74)</td>
</tr>
<tr>
<td>15.00 7.60</td>
<td>Programmed Loop (read, I70)</td>
</tr>
<tr>
<td>14.75 7.40</td>
<td>Programmed Loop (write, I70)</td>
</tr>
<tr>
<td>Tfs</td>
<td></td>
</tr>
<tr>
<td>3.50 1.80</td>
<td>Block Transfer (read, I74)</td>
</tr>
<tr>
<td>3.50 1.80</td>
<td>Block Transfer (write, I74)</td>
</tr>
<tr>
<td>3.00 1.54</td>
<td>Block Transfer (read, I70)</td>
</tr>
<tr>
<td>3.00 1.54</td>
<td>Block Transfer (write, I70)</td>
</tr>
<tr>
<td>Tfs</td>
<td></td>
</tr>
<tr>
<td>23.25 11.90</td>
<td>Automatic I/O (read, I74)</td>
</tr>
<tr>
<td>23.75 12.16</td>
<td>Automatic I/O (write, I74)</td>
</tr>
<tr>
<td>16.50 8.45</td>
<td>Programmed Loop (read, I70)</td>
</tr>
<tr>
<td>16.25 8.30</td>
<td>Programmed Loop (write, I70)</td>
</tr>
<tr>
<td>24.00 12.30</td>
<td>Programmed Loop (read, Time-out, I70)</td>
</tr>
<tr>
<td>23.75 12.20</td>
<td>Programmed Loop (write, Time-out, I70)</td>
</tr>
<tr>
<td>Tmx</td>
<td></td>
</tr>
<tr>
<td>24.00 12.30</td>
<td>Progr. loop (read+write, Time-out, I70)</td>
</tr>
</tbody>
</table>
**Figure 5.1.2: Read Block Diagram (Multiplexer, slow links)**

<table>
<thead>
<tr>
<th>Client (Block I/O)</th>
<th>Filestore (Prog. Loop)</th>
<th>(Fast Links) (1 Mbps)</th>
<th>(Multiplexer) (Prog. Loop)</th>
<th>(Slow Links) (26 Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AA</td>
<td>AA</td>
<td>AA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RD PL VD PL</td>
<td>RD PL VD PL</td>
<td>RD PL TOL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RD PL VD PL</td>
<td>DLY VD PL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RD PL TOL</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.1.3: Write Block Diagram (Multiplexer, Slow Links)**

<table>
<thead>
<tr>
<th>Client (Block I/O)</th>
<th>Filestore (Prog. Loop)</th>
<th>(Slow Links) (26 Mbps)</th>
<th>(Multiplexer) (Prog. Loop)</th>
<th>(Fast Links) (1 Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AA</td>
<td>AA</td>
<td>AA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RD PL VD TOL</td>
<td>DLY RD PL VD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DLY RD TOL</td>
<td></td>
</tr>
</tbody>
</table>

**Annotations:**
- WD: Write Byte
- AA: Automatic Ack.
- PL: Programmed Loop
- TOL: Time Out Loop
- DLY: Delay (Loop Control)
Figure 5.1.4: Read Block Diagram (Multiplexer, Fast links)

(Filestore
Prog. Loop)  

(Fast Links)  
(1 Mbps)

(Multiplexer
Prog. Loop)

(Slow Links)  
(1 Mbps)

(Client
Block I/O)

---

Figure 5.1.5: Write Block Diagram (Multiplexer, Fast Links)

(Client
Block I/O)

(Slow Links)  
(1 Mbps)

(Multiplexer
Prog. Loop)

(Fast Links)  
(1 Mbps)

(Filestore
Prog. Loop)
Figure 5.1.6: Read Block Diagram (Direct, Fast links)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Prog. Loop)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Fast Links)</td>
<td>AA</td>
<td>AA</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 Mbps)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Block I/O)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.1.7: Write Block Diagram (Direct, Fast links)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Prog. Loop)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Fast Links)</td>
<td>AA</td>
<td>AA</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 Mbps)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Block I/O)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To clarify the estimates of block-transfer times, we have illustrated (see figures 5.1.2 to 5.1.7) a number of pertinent timing diagrams. The timing constituents represent a particular transfer method at the host level: multiplexor and filestore using a programmed loop, and the client using Block Transfer I/O instructions.

On each figure, the vertical lines separate the maximum time components of each byte being transferred (i.e. the critical component). Times are displayed in accordance with actual processing time characteristics (i.e. the values given in table 5.1.4). We can regard a programmed loop as consisting of three components: a transfer instruction (Read (RD) or Write (WD)), a simple programmed loop (PL) and possible a time-out loop (TOL). Currently clients do not make use of a time-out scheme. In a particular block transfer, this last component may not require to be executed, depending on whether the byte is ready to be received or transmitted as soon as the device status is tested. If the byte is ready, the time-out loop is not executed.

As can be seen in figures 5.1.6 and 5.1.7 (direct link connections), clients can attain better performance when reading blocks from the filestore than when writing blocks to it. This is due to the fact that in a normal transfer, when reading from the filestore, the time-out loop (TOL) is never executed because the byte
transmission from the filestore is smoothed by the buffering capacity of the link. When writing a block to the filestore, the time-out loop is entered for each byte read. We observe that this could be avoided if the programmed loop (PL) at the filestore were slightly increased (say by one 4-usec instruction) so as to allow a closer synchronization of the receiver with the arrival of each byte.

Considering the case of clients connected to the multiplexor via 0.26 Mbps links (see figures 5.1.2 and 5.1.3), the writing operation can render worse performance than the reading. This, as can be seen in figure 5.1.2, is due to the fact that the delay (DLY) incurred when the multiplexor reads bytes from the slow interface, affects the critical component (slow links) of the transfer time.

Regarding clients connected to the multiplexor through 1 Mbps links (figures 5.1.4 and 5.1.5), the critical time component is at the multiplexor, and no delay is expected to occur at the byte reception level. In this case, block reading and writing should be expected to show a similar time performance (the time of the loop at the multiplexor).

The estimates of block transfer times for the various interfaces, considering different transfer methods at the
client and filestore, are summarized in table 5.1.5. Times concerning the transfer methods not included in figures 5.1.2 to 5.1.7, were also worked out from table 5.1.4, and using the same pictorial scheme above illustrated. Those times including two values indicate the range of times which can be present due to loop control delays.

Table 5.1.5: Estimated block-transfer times (msec)

<table>
<thead>
<tr>
<th>Transfer Method</th>
<th>Oper.</th>
<th>Connection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Filest.</td>
<td></td>
<td>Mux. (0.26 Mbps)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read</td>
</tr>
<tr>
<td>Blk. I/O P. loop</td>
<td></td>
<td>23.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25.0-28.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12.3</td>
</tr>
</tbody>
</table>

Experimental Timings

The measurement data obtained from the experiments are summarized in table 5.1.6. As in the previous experiment, they represent average values. Minimum and maximum times were very close to the average values.
Table 5.1.6: Observed block-transfer times (msec).

<table>
<thead>
<tr>
<th>Transfer Method</th>
<th>Oper.</th>
<th>Connection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mux. (0.26 Mbps)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Client Filest.</td>
</tr>
<tr>
<td>Block I/O Auto I/O</td>
<td>Read</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>23.5</td>
</tr>
<tr>
<td>Block I/O P. Loop</td>
<td>Read</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>24.5</td>
</tr>
<tr>
<td>P. loop P. loop</td>
<td>Read</td>
<td>26.5</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>24.5</td>
</tr>
</tbody>
</table>

The observed times displayed above, nearly match the values of the estimated times given in table 5.1.5. As in the first experiment, small differences are attributable to the real speed of the processor involved in relation to its nominal speed, and the uncertainties of delays (software and hardware) for byte transfer synchronization. In general, in these experiments, this last source of overhead was relatively small, reaching at most about 30% of the maximum expected delays.

To put the performance figures obtained in context, it may be observed that there is something of a mismatch between link capability and processor capability. In the case of the slow links, the transmission time of the medium dominates, so that there is a substantial slack in the cpu involvement, but in such small amounts as to be useless for other purposes. In the case of the fast links, the processor time dominates if programmed loop is used, leading to under-utilisation of the available...
bandwidth. While this is not true of the Block Transfer mode, the Block Transfer instructions are regarded as "dangerous" since any mis-behaviour at the other end can hang the processor.

In summary, for a 512-byte block, the best case figure is approximately 6.2 msec and the worst case figure (on the factors considered) is approximately 27 msec.
5.2 Performance of Departmental Ethernet

The aim of this section is to discern the relevance of individual hardware/software network factors in the host-to-host dialog responsiveness of the local network. Direct measurements on the current Departmental Ethernet system were carried out to establish the contribution of each individual component in the total time involved in the interchange of messages between hosts.

The factors that may affect the overall network performance of the Ethernet have been described in section 2.3. Some of them are analysed in the timing experiments carried out here.

Experiment Description

The elements we selected for evaluation comprised a number of stations and their associated hosts, and direct links interfaces interconnecting hosts. The latter were used as a mean of synchronizing external events (e.g. timing).

Initially, network traffic generators were implemented to evaluate some user-oriented network parameters, i.e. network throughput, message delay, under different loads. Here, the load imposed was considered as a function of
the packet size, packet rate, and the number of transmitting stations. This was done by modifying the station firmware.

The special routine in the controller firmware was very simple: allowing just interrupts from the host (for load changing), the station waited for a random period of time (signalled by the host) and then attempted a frame transmission. Based on monitoring the state of the DMA, packet completion was signalled to the host where statistics were accumulated (e.g. load imposed, effective packet transmission). By counting the number of packets successfully delivered through the Ether, the trunk throughput can be assessed (i.e. the time that the Ether is busy carrying packets, as a function of the load imposed). To ascertain the real load imposed and the effective utilization of the low level trunk, a sampling station was used. A slight modification in the hardware circuitry at one station was carried out, to allow for the sampling of the signal on the cable. Using a tight firmware-implemented loop the station was able to enquire the state of the trunk (busy or idle). At regular intervals the information collected was passed on to the host where statistics were accumulated for a given period -large enough to ensure the accuracy of evaluations.

It was discovered that the maximum load imposed by one node could never reach more than approximately 80% of the
capacity of the trunk (The maximum was obtained for 1-kbyte packets). This was due firstly to the nonzero transmitter-to-transmitter turnaround time and secondly, to the fact that each station is provided with an electromagnetic switching device (relay) in the tap that protects the hardware circuitry and avoids gross overloading from one node. This device, energized by electrical current through its coil, breaks the electrical circuit to the ether upon detection of the transmitter activity rising above the permitted level.

A second system for evaluation was implemented to assess the network throughput including the high level protocols (i.e. interaction of the network access control and the low level firmware at the station controller). A loading system was implemented at the host level, the load synchronization being signaled by an external machine. The purpose of this experiment was to measure the throughput in a multi-client/server environment. This comprised the initiation from one station of a virtual connection with a predefined number of loaders. To avoid unnecessary delay at the receiver, each packet delivered to the host from the station could be "dropped to the floor" (i.e. acknowledged by the host but not received).

However, at the time these experiments were undertaken, the Departmental Ethernet was not operational
enough to cope with a sufficient level of load to allow the experiment be carried out with a significant traffic load. However, the above system is available for future evaluations. The influence of different workloads upon the performance of Ethernet-like systems is analysed subsequently. Instead, we have undertaken timing experiments for the evaluation of different levels of message delays upon internetwork communication between hosts, under no congestion. The experiment can be described briefly as follows:

The test configuration for evaluations included a host-station pair (transmitter - receiver) which as well as being physically connected through the Ethernet were point-to-point interconnected for timing purposes. Provision for loader synchronisation was also incorporated in the event that time measurements under congestion were possible.

After initiating the protocol of interconnection between the two stations, the transmitter was "woken up" at random intervals, and initiated the transmission of a frame to the destination host. Different levels of message delay were measured, depending on the instant the transmitter signalled the receiver host to start timing. At the receiver end, timing was completed at one of two points, either as soon as the first byte was ready to be received from the station or when the last byte of the
packet had reached the host's buffer. The timing experiments incorporated, at the host level, two different processors (I74 and I70), and two data transfer capabilities (programmed loop and Block I/O instructions). All network services were included in the evaluations: virtual circuit (including Protocol and Protocol-less mode) and Broadcast.

**Expected Times**

Data coming to/from the Ether, is buffered into RAM. Accordingly, the total block transfer delay through the Ethernet is accumulated in three communication entities: the Ether, and the two host/station interface pairs. This time, without considering the time involved at the station controller, can be expressed as,

\[
T_{blk}(\text{eth}) = 512 \times \{ \text{Max}(T_{tx}, T_{dma}) + \text{Max}(T_{eth}, T_{dma}) + \text{Max}(T_{dma}, T_{rx}) \} \quad (5.2.1)
\]

where,

- \(T_{dma}\) = Byte transfer time over the station's DMA interface
- \(T_{eth}\) = Byte transfer time over the Ether

Times \(T_{tx}\) and \(T_{rx}\) were defined earlier (see section 5.1).
**Figure 5.2.1: Read Block Diagram (Ethernet Links)**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filestore (Prog. loop)</td>
<td></td>
</tr>
<tr>
<td>Parallel Interface</td>
<td></td>
</tr>
<tr>
<td>DMA (from host)</td>
<td></td>
</tr>
<tr>
<td>Disk Drive</td>
<td></td>
</tr>
<tr>
<td>DMA (from Tx. FIFO)</td>
<td></td>
</tr>
<tr>
<td>Ether</td>
<td></td>
</tr>
<tr>
<td>DMA (to Rx FIFO)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WD</td>
<td>Write Byte</td>
</tr>
<tr>
<td>PL</td>
<td>Programmed Loop</td>
</tr>
<tr>
<td>CTFM</td>
<td>Controller Firmware Time</td>
</tr>
<tr>
<td>Lt</td>
<td>DMA Latency Time</td>
</tr>
<tr>
<td>Dt</td>
<td>DMA Byte Transfer Time</td>
</tr>
<tr>
<td>Et</td>
<td>Ether Byte Transfer Time</td>
</tr>
<tr>
<td>RD</td>
<td>Read Byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA (from Ether)</td>
<td></td>
</tr>
<tr>
<td>Parallel Interface</td>
<td></td>
</tr>
<tr>
<td>Client (Block I/O)</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- WD: Write Byte
- PL: Programmed Loop
- CTFM: Controller Firmware Time
- Lt: DMA Latency Time
- Dt: DMA Byte Transfer Time
- Et: Ether Byte Transfer Time
- RD: Read Byte
Using a similar pictorial scheme to that given for the block transfer times over the Departmental links, this time is illustrated for the Ethernet configuration in figure 5.2.1. Again, the transmitter host uses a programmed loop, and the receiver host Block Transfer instructions. Each time component in turn will be analysed through the data measurement obtained.

Experiment Results

In figure 5.2.2, for a particular configuration, different time constituents of host-to-host message delay, obtained from direct measurements, have been broken down, to illustrate delays at various protocol levels. The different sources of overhead can be explained as follow:

(a) Packet transmission time through the Ether. This time is defined by the bandwidth availability of the network (3.79 usec/byte), and the size of the packet being transmitted. For every data packet being transmitted a constant time of about 60 micro-seconds is included, which corresponds to the data control transmission time (16 bytes time).

(b-a) Overhead at both Ethernet station controllers. This overhead relates to both hardware and software delays involved in the transmission and reception of
packets. It includes the transfer of the station-to-station acknowledgement at the receiver end, corresponding to approximately 1 msec. It is clear that hardware delays (e.g. packet serialization and deserialization, deference time, carrier detection, CRC generation and checking, address filtering) can be considered negligible compared with the software delays.

(c-b) Host/station transmission time. This is the time it takes a packet to be transmitted from the host's buffer to the Ethernet's RAM memory. At this level of transmission the DMA does not operate in burst mode, but instead data transmission is carried out using single byte transfers. Three time components are included here: The DMA latency time, the DMA transfer speed, and the host/interface transfer and byte synchronization. The first time component corresponds to the bus request/acknowledge cycle between DMA and the Z80 processor. This is the time it takes the Z80 processor to respond to a bus request from the DMA, which can be as much as one Z80 machine cycle (approximately between 3 and 6 clock cycles). Currently, the speed of the Z80 cpu does not match the nominal 2Mhz speed. From direct measurements we obtained that its actual speed is approximately 1.5 Mhz. Hence, the latency time can range from 2 to 4 usec. The second time component corresponds to four cycles of the Intel 8257 DMA controller, which provided with a 2MHz clock input allows the DMA to
transfer at a rate of 0.5 Mbytes/second (2 usec/byte). The third level of delay relates to the time it takes the byte to arrive at the interface hardware from the host. According to the characteristics of the host transmitter (an Interdata 74), this time (see table 5.1.1) represents a maximum of 3.5 usec/byte.

(d-c): This is the time required to set up the transaction mechanism for the transfer of data packets at both host/station pairs (see section 3.4). Ethernet boards handle control information one byte at a time. If the host/station mechanism of transmission is carried out using the protocol-less facility, this control exchange time is not present, but instead the transaction is completed by means of a time-out scheme. This mode of transmission, as we will show later, can significantly increase the host-to-station transmission delay.

(e-d) Station/host transmission time. The time it takes to flush out the data packet from the station buffer to the host's buffer. This time is equally affected by the timing factors indicated for the time c-b above.

Figure 5.2.3 compares total host-to-host message delays (e.g. time e in figure 5.2.2) for the three types of services sustained by the network. Obviously, it is not relevant to compare performance between the simple two-message exchange scheme and the broadcast mode,
because they are intended for different purposes. The former involves message exchanges between a well identified pair engaged in a dialog, while the latter involves one source node sending messages to a set of potential recipients in the network. Examples of the last service might be: a station that broadcasts the time of day at regular intervals, or a node which announces its presence to the other nodes in the network when it is first switched on. The constant time difference observed between broadcast and protocol mode is due solely to the extra overhead incurred at the receiver station by preparing and sending the acknowledgement packet to the transmitting station. Time dissimilarities obtained for the protocol and protocol-less mode of operation are explained below.

Whenever data is being received from the host, the station checks its completion either by the reception of an ETX control character, a time-out event, or by the occurrence of a buffer overflow condition. The first event occurs when the station and its host exchange information in protocol mode. The last two characterize the protocol-less mode of transmission. In this case, the station has to divide a continuous data stream into packets: a packet boundary is imposed either when a significant delay is observed (the normal case) or when the buffer into which data is being transferred by the DMA overflows.
Figure 5.2.2: Message Delay (Host to Host) (Time Components)

Connection: Virtual Circuit; Mode: Protocol
Data Transfer (Host): Block I/O Instructions
Processors (Host): (Receiver: I70, Transmitter: I74)

[Msec.]

(a) : Packet transmission (Ether)
(b-a) : Network/Software Delays
(c-b) : Host/Station Data Transfer
(d-c) : Control Exchange
(e-d) : Station/Host Data Transfer

Packet Size [Bytes]
Figure 5.2.3: Message Delay (From Host to Host)

Data Transfer (Host): Block I/O Instructions
Processors (Host): Receiver: 170, Transmitter: 174

[Graph showing message delay versus packet size in bytes with lines for different modes: P-less Mode (1.00), Broadcast (0.20), Protocol Mode (0.25).]
Figure 5.2.4: Message Delay
(From Transmitter Host to Receiver Station)

Data Transfer (Host): Block I/O Instructions

[Msec.]

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
</tr>
<tr>
<td>280</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td></td>
</tr>
<tr>
<td>360</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>440</td>
<td></td>
</tr>
<tr>
<td>480</td>
<td></td>
</tr>
<tr>
<td>520</td>
<td></td>
</tr>
<tr>
<td>560</td>
<td></td>
</tr>
</tbody>
</table>

---

10  11  12  13  14  15  16  17  18  19  20

P-less Mode (I74) (0.90)
P-less Mode (I70) (0.80)
Protocol Mode (I74) (0.25)
Protocol Mode (I70) (0.20)
Figure 5.2.5: Message Delay
(From Transmitter Host to Receiver Station)

Data Transfer (Host): Programmed Loop

[Msec.]  St. Dev.

- P-less Mode (174) (2.30)
- P-less Mode (170) (2.00)
- Protocol Mode (174) (0.22)
- Protocol Mode (170) (0.20)

Packet Size [Bytes]
As no external timing device is provided for the purposes of signaling time-out events, the controller makes use of a low-level process embedded in the firmware (watchdog) for this purpose.

Occasionally, the host/station transmission time may be greater than the time-out window being used, so that the packet may be timed out before its transmission finishes. This situation is more likely to occur if the packet transmission time at the host level is high (e.g. when using programmed loop), because there is more probability of timing out the packet in the middle of its transmission. This situation can be observed when comparing the average times illustrated in figure 5.2.4 and 5.2.5, for the protocol-less mode of transmission. In these figures, the average times represent host/station packet delays, namely the time lapses from when the first byte was ready to be sent at the transmitting host until the packet was ready to be delivered to the receiving host.

When using Block Transfer I/O instructions (figure 5.2.4), times increase constantly as the packet size does. But when a programmed loop is used, (figure 5.2.5), the average observed times are less uniform. As can be seen from both figures, protocol-less mode of transmission entails much higher delays than transmission in protocol mode. These time differences become less as
the size of the packet being transmitted increases. This is due to the fact that the proportion of time wasted by the time-out procedure is less when the packet transmission time is greater (i.e. as the packet size increases). The figures also illustrate the impact of different processor speeds in the total packet delay.

At the receiver host, the protocol-less mode of transmission does not produce the delays that are incurred when transmitting. This can be observed by comparing the corresponding times in figure 5.2.3 and 5.2.4. In the former, where the host reception time is included, the time differences between protocol and protocol-less mode remain similar to those of the latter.

Host-to-Host Timing in a Two-Stroke Exchange Protocol

A number of timing experiments were carried out emulating the two-stroke message exchange by which client and filestore communicate (see section 5.3). The experiments included the read/write exchange mechanism. The aim was to ascertain the different impact of I/O capabilities at the host level and protocols at the network interface upon this mode of operation. These evaluations provide timing information about the client/filestore communication overheads incurred when excluding processing and disk transfers at the server (i.e. the times the user would expect in the hypothetical
case that the filestore did not have to be involved in its normal processing and disk activities).

The protocol followed when reading and writing operations between the client and the filestore are carried out, consists of:

**Reading**, the client sends: a request message including three or four characters containing the operation code, a transaction number and a terminator character. The filestore responds with: a response message whose first two bytes contain the number of bytes in the block (usually 512), followed by the block of data requested by the client.

**Writing**, the client sends: a message whose first two or three bytes contain the operation code, the transaction number, a byte count and a terminator character. These bytes are followed, in the same packet, by the block of data (usually 512 bytes) to be written out to disk.

The filestore responds with: an acknowledgement message which is sent immediately after reception of the block of data from the client.

The above description relates to the messages as defined in the filestore protocol, ignoring the standard Ethernet packaging.
Experiment Description

Even though the above client/filestore message exchange was simulated on two clients, we will still continue to use the terms client and filestore.

The timing experiments involved a similar configuration to that employed for the link experiments. Each of them required three nodes: the two nodes engaged in the data exchange and a clock attached to the client. The client signals the clock to start timing as soon as it is ready to transmit a packet. Upon reception of the filestore response, the clock is signaled to stop. In a reading operation, timing is completed once the filestore response (data packet) has reached the client's buffer. In both reading and writing operations this procedure was repeated a number of times. Concerning the network protocols, we included for evaluation both protocol and protocol-less mode for the client but only protocol mode for the filestore (this being the normal mode of operation used by the filestore). Broadcast mode was discarded because under normal circumstances a client/filestore dialog would not be carried out using this type of service.

The expected times in this host-to-host request/response dialog can be specified as follow:

For a reading operation,
Tdlread = Tclst(3) + Tstfs(3) + Teth(550) + 2Tstfm
+ Tfsst(515) + Tstcl(515) + Tcecl + Tcefs

For a writing operation,
Tdlwrit = Tclst(515) + Tstfs(515) + Teth(548) + 2Tstfm
+ Tfsst(1) + Tstcl(1) + Tcecl + Tcefs

Tclst(n) represents the time for the transfer of n bytes from client to station and Tstcl(n) the client's reception time of n bytes from the station. Similarly, Tfsst(n) and Tstfs(n) are times involving the station and the filestore. Tclst and Tfsst represent host/station data block transfer overheads (e.g. (c-b) in figure 5.2.2). Tstcl and Tstfs represent station/host block transfer overheads (e.g. (e-d) in figure 5.2.2). All these times are dependent on the type of protocol and I/O capabilities. The first two can be obtained by taking the difference between times in figure 5.2.4 and 5.2.5 respectively, and time b in figure 5.2.2. The value n in Teth(n) (the byte transmission across the Ether) takes into account the header (twice, as there are two internetwork message exchanges) of the internetwork protocol (16 bytes). The time involved in the exchange of control information between the host and the station (Tcecl), and the filestore and the station (Tcefs), are present only in the protocol mode of transmission (e.g. (d-c) in figure 5.2.2). Time Tstfm is the overhead involved at the station's controller firmware (i.e. time (b-a) in figure 5.2.2). The constant 2 multiplying this term is because there are two messages involved in each exchange.
Figure 5.2.6: Two-stroke Host/Host Protocol
(Measurements on Ethernet Links)

Transmitter          Receiver
A: (P. Mode, Block I/O) -- (P. Mode, Block I/O)
B: (P. Mode, Pr. Loop) -- (P. Mode, Pr. Loop)
C: (P-less, Block I/O) -- (P. Mode, Pr. Loop)
D: (P-less, Block I/C) -- (P. Mode, Block I/D)
E: (P-less, Pr. Loop) -- (P. Mode, Pr. Loop)

[ms]  

--- Transmitter: 174
..... Transmitter: 170
Measurement data obtained from the experiments are displayed in figure 5.2.6 for the Reading operation (similar results were obtained for the Writing operation). As can be seen, the mode of liaison currently being used in service for the server and its clients (bar E) entailed the greater delay. Difference between liaison E and B was due solely to the time wasted in the watchdog process at the controller firmware, when receiving the request packet (3 bytes) from the client. Liaison D is likely to constitute the client and filestore mode of communication in the future. This will represent a time reduction of about 37% with regard to the current message exchange time.

Performance Analysis on Ethernet-like Networks

The performance characteristics of Ethernet-like systems under different loads have been extensively measured and analysed [A1m79, Sho80, Wat80, Yeh79]. Based on these studies, and our timing experiments (see also 5.3), it is possible to quantify what real time constraints the network access mechanism imposes upon clients communicating with the filestore through the Ethernet. In other words, we can estimate how significant is the response time degradation as average load increases, in the time that elapses in a typical two-stroke message exchange between the server and its clients.
Based on queuing theory and specifically on embedded Markov processes [Kle75], performance analysis of network responsiveness (i.e. network service time) under different loads has been carried out [Alm79, Mar81]. The network service time is analysed in terms of the average packet delay, measured as the time lapse from when the packet is first generated by the transmitting adapter until it is successfully delivered to the destination adapter. The load is expressed as a proportion of the network carrying capacity, measured in bits/second. The service time can be broken down into two components: packet transmission time and contention resolution time. The former is independent of the traffic being imposed, but the latter depends on the number of packets queuing for service (instantaneous load), and is equal to the slot-time multiplied by the number of slots devoted to contention (see relation 2.3.2). The instantaneous load changes according to the rate at which packets arrive (average load) and the rate at which packets are given service (i.e. delivered to the destination). Under these assumptions the average message delay, for a given average load, can be obtained by taking the quotient between the average instantaneous load (i.e. the average number of packets waiting for service) and the arrival rate (Little's equation).
Figure 5.2.7: Average Packet Delay
(Rate: 2 Mbps; Packet Size: 32 Bytes)

Figure 5.2.8: Average Packet Delay
(Rate: 2 Mbps; Packet Size: 150 Bytes)

Figure 5.2.9: Average Packet Delay
(Rate: 2 Mbps; Packet Size: 512 Bytes)

Figure 5.2.10: Average Packet Delay
(Slot: 30 usec; Packet Size: 512 Bytes)
Based on this approach, we have derived message delay distributions (see figure 5.2.7 to 5.2.10) for different network configurations (i.e. slot-time, network rate and average packet size), assuming some of the characteristics of the Departmental network. The slot-time must be considered as the worst case collision detection time (see section 2.3), which apart from the network's dimensions (length of the communicating pathways), is dependent on the time involved in the intervening electronics (see table 2.3.1). For a small-scale network like the current Edinburgh network, a slot-time between 10 to 15 usec would fit reasonably well (assuming that it would expand to no more than about 500 meters).

For a given network rate, we can quantify how significant is the slot-time in the average packet delay. For relatively big packets (see figure 5.2.8 and 5.2.9) packet delays are almost insensitive to the slot-time but for small packets (figure 5.2.7), increase in the slot time can significantly increase the waiting times as the load increases. In the client/filestore network environment, the expected pattern of messages over the network is: a small packet (of 20 bytes) is followed by a big packet (532 bytes), or vice versa. But also, every data packet is followed by an acknowledgement packet (16 bytes). Thus, this network configuration can be considered as having an average packet size of about 150
bytes. In figure 5.2.8 we have displayed average packet delays relating to this average size packet. Simulation experiments carried out [Alm79], have demonstrated that Ethernet performance with a variety of packet sizes compared to its performance with a fixed size for the same average packet size, can evidence greater delays in the former case (approximately from 7% to 12%).

Figure 5.2.10 shows how the network response time is improved as the carrying capacity of the physical network medium is increased, for a given average load. The reference load in this case was considered as a percentage of the 2 Mbps network (the speed of the Departmental Ethernet). As can be seen, if the load maintains the levels imposed for the 2 Mbps network, increasing the network rate would lead to an almost negligible decrease in network delays.

The higher level protocols impose a number of limiting factors on network performance. We analyse some of these on the basis of evaluation of the network.

One of the real-time constraints that the current controller firmware imposes is related to the depth of the receiver FIFO buffer (64 bytes). Packets may arrive off the Ether at a rate that can be as close as 3 usec (the interframe recovery time). Considering the current bandwidth of the network (2.1 Mbps), the receiver buffer
gives the controller a real time span of about 243 usec, to validate the packet, to link the current packet into the appropriate (To-host) queue, and to reprogram the DMA Ether-receiver channel with another free buffer. Considering the actual speed of the Z80 cpu (1.5 Mhz), for an average machine cycle this real time span of 243 usec would correspond, approximately, to 80 Z80 machine instructions. If the network carrying capacity is incremented in some proportion, this hypothetical limit number of instructions would decrease in the same proportion. To avoid intolerable delays or deadlock conditions (i.e. source adapter continuously retrying due to packets being lost at the receiver adapter) under high loads, the adapter's cpu speed would have to be increased.

For similar reasons, under high loads it is important that tasks like address recognition and CRC checking are offloaded into customised hardware (as in the Edinburgh Ethernet).

The interfaces at the Host level can also impose substantial delays if the adapter is receiving packets at a high rate. The rate at which a host (e.g. the filestore) interacting with the DMA may flush out packets from the station's buffer is approximately 7.5 usec/byte when using Block Transfer instructions (see figure 5.2.2), and approximately 17 usec/byte when using
programmed loop. Thus, using the fastest I/O capabilities of the Interdata, the mismatch between the speed of packet reception from the Ether and packet delivery to the host is approximately a factor of two. Under high loads this mismatch could decrease network throughput and increase packet delay at the source station. Currently, packet delivery to the host is postponed until after the sending of the acknowledgement to the source station. When comparing Broadcast and Protocol mode service times (see figure 5.2.3), we can observe that a non-trivial amount of time is wasted in preparing and sending the acknowledgement packet. This time could be gained by the host if packet delivery to it and the dispatch of the acknowledgement packet were done concurrently.
5.3 Performance of Filestores

The present section is devoted to a quantitative evaluation of the filestore described in section 3.2, related to disk characteristics and file management service, as seen by the clients to which the filestore provides service through the local network.

The aim is to establish which are the limiting factors, having regard to transfer time through the communication channels, cpu(s) and disks. Thus a measure is established of the average delay that a client can expect when using the file server capabilities. The evaluation of the above factors is aimed at obtaining estimates like maximum, minimum and average delays in the process of transferring data files back and forth among the file server and its clients. Performance evaluations on the communication side, namely the line transmission overheads, have already being carried out (see sections 5.1 and 5.2). The performance figures obtained will be used to find total time profiles in the client/filestore dialog through the different communication pathways.

Configuration

The environment in which experiments were conducted, comprised a group of Interdata machines which obtain
general file support from the filestore (an Interdata 70). Two different software implementations of the filestore were evaluated, both of them mounted on identical hardware components (see section 3.2). The second implementation incorporates facilities for communication using the Departmental Ethernet. This new version was also aimed at reducing the complexity of the old system, which was based on a transaction-processing scheme. Instead, the new version uses a proper multiprocessing regime. Details relating to different approaches to filing service operations on the two filestores are given subsequently, when analysing particular capabilities.

To evaluate the time limiting factors, as far as the file management service is concerned, we selected five of the filestore capabilities which are commonly required in a user session -CLOSE, OPENR, OPENW, READSQ and WRITESQ. The first two operations represent user-level requests (i.e. require a user-number as identification), and the rest correspond to transaction-level requests (i.e. require a transaction-number as identification). The following is a description of the activities carried out by each of them.

**OPENR** - Initiates a transaction to read a file sequentially. The system checks: the validity of the request, the existence of the file, and the user permission (to read the file); in general this will
involve the user's directory being read into store. A transaction number is allocated, and the transaction record assigned to the transfer is initialised (setting the in-core pointers at the beginning of the file). A transaction number (Txno) is sent to the user as a token for subsequent data transfer requests upon the file.

Openw - Initiates a transaction to write a file sequentially. The system checks the validity of the request, the user permission and the remaining quota. As in Openr operation the transaction mechanism is initialised and a Txno sent to the user. Currently, a first free space (extent) is allocated to the file if a size is specified or can be estimated from an existing file of the same name; otherwise allocation is carried out during the write operation.

Readsq - Reads the next block on the file associated with a given Txno. The block is 512 bytes for all except the last in the file. The block required must be read from the disk (these being no pre-reading in the existing filestore) and a directory read is also required if a switch is being made to a new extent. When the data is available in filestore memory, it can be sent as the response to the client.

Writesa - Writes the next block in the file associated with a given Txno. The user specifies the number of bytes to be written (512 bytes unless it is the last
block of the file). The file is dynamically created through successive \texttt{Writeq} operations. An acknowledgement is sent immediately after the block has been read from the user's buffer. Thus, the user does not have to wait for the data to be written on disk to activate another operation. The filestore then proceeds to write the block to disk. If space allocation is required it may be necessary to read and update directory and bitmap information.

\textbf{Close} - The only case of interest here is when this operation terminates a transaction previously opened for sequential writing: \texttt{Closew}. Here, all unused blocks, if there are any in the last extent, are deallocated and if a file with the same name already exists, it is deleted.

\textbf{Experiment Description}

The timing experiments carried out made use of a single specially-programmed client system including all the facilities for direct evaluation of a number of different alternatives on each filestore operation.

The process of timing was carried out by using either an internal or external clock. The external clock was always necessary in cases when computing total times (i.e. including the client and communication overheads).
Timing results were collected by sending a number of successive client-filestore requests (usually 100). Average response times were measured repeating this procedure several times. The first experiments were carried out at times when there was no contention (i.e. no other client interrupting the filestore). The absence of congestion has a number of effects on performance, not simply a matter of the absence of queuing. For example, fewer directory reads and disk seeks are to be expected.

The time interval between the client's requests to the filestore, was chosen to be uniformly distributed between the minimum and maximum latency time of the disk (see table 5.3.1). The reason was to avoid fixed bias in the results due to the position of the disk head while making a request. For example, if the interval between requests had been constant, the disk would have rotated a constant distance between each request. Given the difference in processor speed among clients, the results would be distorted by the specific disk rotation times incurred by each client. For each group of requests, a different seed was given to a random number generator routine. To set up the initial inter-request interval, the seed was obtained as a function of the number of microseconds the user delayed in providing one of the requested parameters of the experiment. The subsequent seeds were taken as a function of numerical values (times) obtained from the experiment itself.
Simple Models of Filestore Operations

Before going on to describe the timing experiment results, we develop very simple models to reproduce two factors that affect performance levels at the filestore: device characteristics and file management service.

Disk Characteristics

Two CDC 9762 67.4 megabyte replaceable magnetic disks comprise the storage modules used by the filestore. One controller governs the data exchange between the server and disks. Table 5.3.1 presents some parameters of this secondary storage device.

Table 5.3.1: Speeds of CDC 9762 disk.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Conditions</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seek Timing</td>
<td>822-Track Seek</td>
<td>55 ms (maximum)</td>
</tr>
<tr>
<td></td>
<td>Average Seek</td>
<td>30 ms</td>
</tr>
<tr>
<td></td>
<td>1-Track Seek</td>
<td>7 ms (maximum)</td>
</tr>
<tr>
<td>Latency Time</td>
<td>Maximum</td>
<td>17.2 ms</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>8.33 ms</td>
</tr>
<tr>
<td>Recording Rate</td>
<td></td>
<td>9.677 MHz (nominal)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1209600 bytes/sec.</td>
</tr>
</tbody>
</table>

The transfer unit used by the filestore for all file data transfers is a 512-byte block. The disk access time for a block, in milliseconds, is therefore:

\[ Da = Sk + Dlt + 512 \times RR \] (5.3.1)
where, Sk is the time required to seek for the required track, Dit the latency time of the disk to reach the required sector, and RR the recording rate of the disk, in milliseconds/byte. According to the values in table 5.3.1, a summary of some characteristics disk access times is given in table 5.3.2.

Table 5.3.2: Disk access times (for a disk block). (msec)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum:</td>
<td>72.62</td>
</tr>
<tr>
<td>Average:</td>
<td>38.75</td>
</tr>
<tr>
<td>Average (no seek):</td>
<td>8.75</td>
</tr>
<tr>
<td>Minimum:</td>
<td>0.42</td>
</tr>
</tbody>
</table>

File Management Service

Here, we analyse the performance characteristics of the filing system itself, that is concentrating on filing service overheads. For this purpose, we develop very simple models of the filestore capabilities above mentioned. We do not intend to reproduce completely all timing issues included in the file system service operations, but rather simplified analytical specifications of the essential operations involved. The purpose is to improve understanding of the file management system and to provide a technique for evaluating some proposed modifications to the file management strategy. After the analysis, a timing summary is given for the expected times in a number of operations.
**Openr** - A simple analytical specification for the time required to open a file for sequential reading is,

\[ T_{openr} = T_{prq(or)} + T_{rddir} + T_{idfil} \]  

(5.3.2)

where,

- \( T_{prq(or)} \) = Time to process the Openr request.
- \( T_{rddir} \) = Time to read the directory (4 blocks) from disk.
- \( T_{idfil} \) = Time to identify the file in the directory.

The time \( T_{prq(or)} \) includes all miscellaneous issues involved in the processing of the command, like the identification of the request, rescheduling of processes, and the sending of a reply to the client. \( T_{rddir} \) is obviously zero if the directory is already in store at the time it is required. \( T_{idfil} \) may involve a search in the owner register table for the file's owner, and always involves a search in the directory for the file being requested.

**Openw** - The time involved in this operation can be expressed as,

\[ T_{openw} = T_{prq(ow)} + T_{rddir} + T_{idfil} + T_{all(n)} + T_{wrdir} \]  

(5.3.3)

Where,

- \( T_{prq(ow)} \) = Time to process Openw request.
- \( T_{all(n)} \) = Time to allocate an extent of \( n \) blocks.
- \( T_{wrdir} \) = Time to write the directory (4 blocks) onto disk.

In the current version of the filestore, space is allocated at the time the file is opened only if the size
of the extent (n blocks) is specified as a parameter in the command or if a file with the same name already exist. In this case, the size of the extent sought is the size of the previous file. In the old version of the filestore, an initial allocation of space was always made at the time of opening the file. Timing specifications involved in the space allocation activity are included below, in the Writesq operation.

**Writesq** - The average time to write one block out of a total of n can be expressed as,

\[
T_{\text{writesq}} = T_{\text{prq}}(\text{wr}) + \frac{1}{n} \sum_{i=1}^{i=k} T_{\text{all}}^{(2)} + T_{\text{wrblk}} + T_{\text{verif}} \quad (5.3.4)
\]

where,

- \( T_{\text{prq}}(\text{wr}) \) = Time to process the Writesq request.
- \( T_{\text{wrblk}} \) = Time to write a block onto disk.
- \( T_{\text{verif}} \) = Time to verify (write) the previous written block.
- \( k \) = The number of extents allocated to the file.

The space management of the filestore system utilises a conventional bitmap approach to keep track of all the free blocks on the disk. Only one bitmap page is held in store at a time. The allocation of a contiguous extent is done on a strict first-fit basis. To implement that, a record is kept of the largest extent to be found in each bitmap page. This array of maximum extents is searched sequentially to find the first page which contains an extent big enough to satisfy a given
requirement. If the bitmap thus identified is not in memory, then two bitmap transfers are required: the current bitmap held in memory is written out first and then the requested bitmap is transferred from disk. The relevant page is then searched according to a first-fit procedure, and a new maximum extent must be computed. When writing files for which a size is not given or deducible at the time of opening, the policy for allocating extents is as follows: when the first block is written, if it is less than 512 bytes (i.e. a one-block file) a one-block extent is requested; otherwise a two-block extent is requested. Thereafter, as extents are exhausted, double the size of the previous extents is requested. In all cases, the extent allocated is the whole of the first extent which is big enough (therefore possible bigger than the request). Hence, the maximum number of extents ($K_m$) that will be allocated for a file of size $n$ (where, $2^{k_1} <= n <= 2^{(k+1)-1}$), is,

$$K_m = \text{int}(\log_2(n+1)) \quad \text{or} \quad K_m = \text{int}(\log_2(n+2)) - 1 \quad \text{(5.3.5)}$$

Where $\text{int}(x)$ denotes the integer part of $x$.

When a extent is found, it is marked as in use and an extent entry is added to the file directory, which may require to be read from disk in case it is not in memory. In any case, the directory is written back to disk (write-through). In the old filestore, it was also necessary to write back a bitmap page as soon as it had
been altered, to avoid inconsistency, but this is not necessary in the new filestore as in it the bit-maps are constructed on system load.

Ignoring the first sequential search of the maximum extent array, the time to allocate a j-blocks extent is given by,

\[ T_{all}(j) = T_{fit}(j) + (T_{dbmap} + T_{wbmap}) + T_{dir} + T_{updbm}(j) + T_{wdir} \]  
\[ (5.3.6) \]

where,

\[ T_{fit}(j) = \text{average time to search in a bitmap page (4096 bits) for } j \text{ consecutive "free" bits, using a first-fit scheme.} \]
\[ T_{dbmap} = \text{Time to read a bitmap block from disk (}=0 \text{ if it is already in memory).} \]
\[ T_{wbmap} = \text{Time to write a bitmap block to disk (}=0 \text{ in case requested bitmap is already in memory).} \]
\[ T_{updbm}(j) = \text{Time to update the j-blocks extent into both the bitmap and the directory.} \]

\( T_{fit} \) depends on the current availability and distribution of blocks on disk. Times \( T_{dbmap} \) and \( T_{wbmap} \) are similar to \( T_{dblk} \) and \( T_{wbblk} \) respectively.

Reads -The average time to read a block, over a total of \( n \) blocks to be read, can be expressed as,

\[ T_{read}(n) = T_{prq}(rd) + k/n(T_{dir} + T_{slot}) + T_{dblk} \]  
\[ (5.3.7) \]

where,

\[ T_{prq}(rd) = \text{Time to process Reads request.} \]
\[ k = \text{The number of extents allocated to the file.} \]
\[ T_{slot} = \text{Time to extract the next extent slot from the directory.} \]
\[ T_{dblk} = \text{Time to read a block from disk.} \]
Tsslot is negligible if the directory required is already in store. Trdblk is dependent only upon the characteristics of the disk (see table 5.3.1). Tprq(rd), being similar to Tprq(wd) above, apart from the activities mentioned for the Openr operation, includes the overhead incurred in the disk handler process.

Closew - The time required to close a new file on disk is given by,

\[ T_{\text{closew}} = T_{\text{prq(cl)}} + T_{\text{rddir}} + T_{\text{dall}(j)} + T_{\text{idfil}} + T_{\text{del}} + T_{\text{wrdir}} \]  

(5.3.8)

where,

- \( T_{\text{dall}(j)} \) = Time to deallocate \( j \) unused blocks in the last extent.
- \( T_{\text{idfil}} \) = Time to identify if a file with the same name already exists.
- \( T_{\text{del}} \) = Time to delete the previous file (if it exists).

We can assume that, in the average, the last extent can be expected to be half empty. Then, \( j = 2^{\frac{k}{2}} \), where \( k \) is the number of extents allocated to the file. \( T_{\text{del}} \) is 0 if no file with the same name to the one being closed exists previously. Otherwise, \( T_{\text{del}} \) accounts for the time needed to deallocate the number of extents allocated for the previous file. \( T_{\text{dall}} \) includes the time to update, in the bitmap table, the current maximum extent size of the page whose extent was deallocated. A search through the whole page needs to be carried out.

Table 5.3.3 contains the time values (in msec.) of

5-58
estimated parameters included in the capabilities modelled above. They represent times only for the current version of the filestore. Differences between the file management service on both filestores are discussed later.

Table 5.3.3: Estimated file management service parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trqp(rd,wr)</td>
<td>2.5</td>
</tr>
<tr>
<td>Tidfil</td>
<td>0.6</td>
</tr>
<tr>
<td>Trddir</td>
<td>40.0</td>
</tr>
<tr>
<td>Twrdir</td>
<td>58.9</td>
</tr>
<tr>
<td>Tali</td>
<td>116.4</td>
</tr>
<tr>
<td>Trdblk</td>
<td>38.75</td>
</tr>
<tr>
<td>Twrbk</td>
<td>38.75</td>
</tr>
<tr>
<td>Tverif</td>
<td>17.6</td>
</tr>
<tr>
<td>Tdall(n)</td>
<td>20 + 0.003n</td>
</tr>
<tr>
<td>Topenr</td>
<td>42.2</td>
</tr>
<tr>
<td>Topenw</td>
<td>60.8</td>
</tr>
<tr>
<td>Twritesq</td>
<td>58.8</td>
</tr>
<tr>
<td>Treadsq</td>
<td>41.2</td>
</tr>
<tr>
<td>Tclosew</td>
<td>109.0</td>
</tr>
<tr>
<td>Tffit(n) (0.018n - 20)</td>
<td></td>
</tr>
</tbody>
</table>

(1): average time, in a 70 files directory. (2): does not includes Tffit.

All times above involving disk access include average seek times (see table 5.3.1) for each disk transfer operation. As the time Tffit is rather difficult to obtain, since it depends on the distribution and availability of free space on the selected page, we have not displayed total expected times including space allocation time activities. We analyse these issues from direct measurements. To have an idea of the level of cpu activities involved in the extent searching operation (Tffit), we have displayed its approximated expected minimum and maximum times. The latter is taken to be independent of n, because it assumes the situation whereby the search goes through the whole bitmap page (256 16-byte words), and that half of the page is empty.
Obviously, the bigger the size of the extent to be allocated the bigger is the probability that $T_{fit}$ approaches this maximum time.

**Performance Measurements**

The analysis of the measurement data obtained from the timing experiments described earlier follows. All operation times were measured from the time that the client program issued the requests until it had received announcement of completion.

The relative performance of the two file servers is quantified in table 5.3.4, which summarizes times (in milliseconds) excluding client/filestore communication overheads. Thus, they correspond to disk times and software overheads incurred at the server. Times in operations involving data transfer (readsq and writesq) represent one block transaction times.

Table 5.3.4: Measured Filestore Management Service Times.

<table>
<thead>
<tr>
<th></th>
<th>Old Filestore</th>
<th></th>
<th>New Filestore</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Average</td>
<td>Max.</td>
<td>Min.</td>
</tr>
<tr>
<td>Opener</td>
<td>28.0</td>
<td>37.0</td>
<td>45.0</td>
<td>2.2</td>
</tr>
<tr>
<td>Openw</td>
<td>139.0</td>
<td>147.0</td>
<td>155.0</td>
<td>23.0</td>
</tr>
<tr>
<td>Closew</td>
<td>140.0</td>
<td>148.0</td>
<td>156.3</td>
<td>67.0</td>
</tr>
<tr>
<td>Readsq</td>
<td>6.0</td>
<td>16.5</td>
<td>23.0</td>
<td>3.1</td>
</tr>
<tr>
<td>Writesq</td>
<td>24.3</td>
<td>31.0</td>
<td>40.0</td>
<td>20.0</td>
</tr>
</tbody>
</table>

Times illustrated above were obtained in absence of
congestion, so that in general no seek time was present. Also, times for the Readsq operation do not include any possible source of overhead added by the presence of multiple extents in the file being read, and in the Openw and Writesq operations the overhead added when allocating of space is required. Times involving space allocation are analysed below. As can be seen, the observed times match fairly well the expected times given in table 5.3.3 (excluding average seek times).

As expected, the range between maximum and minimum times corresponded approximately to the maximum rotational latency of the disk. Minimum times for the operations involving file opening represent times when the directory was held in memory (i.e. it did not need to be read from disk). In the old filestore, usage information in the directory was always updated and written back to disk during the Openr operation (i.e. to indicate that the file was read). In the new version this is only done in the case that the usage information has changed, which explains the greater time obtained for this operation in the old version.

In the old filestore, the level of cpu activity in the above operations was greater than in the new version due to the fact that in the former a high proportion of the I/O processing tasks were carried out by using general purpose supervisor calls, which require more cpu time.
than the special-purpose sequences employed in the new version. This partly explains the greater times obtained in the old filestore for all the operations here evaluated.

Prior to writing data into a file, free space must be allocated. As previously explained, in the old filestore this task was always carried out using an invariant algorithm. However, it is often the case that a file being written corresponds to an updated version of an already existing file. In this case, the size of the new file can be a priori estimated by the size of the previous file. This option has been incorporated in the new filestore, and in such a case the task of space allocation is undertaken when the file is opened. In this way, any extra directory and/or bitmap disk transfers during the write operation are avoided. We have also run timing experiments, in the new filestore, forcing the allocation of space to take place as the file is written. This was easily achieved by deleting any previous existing file before each writing operation was started. The results obtained for the average time required to write a block on disk, as a function of the size of the file being written, are illustrated in figure 5.3.1. Curves show that the average time is a decreasing function of the file size. This characteristic reflects the policy of space allocation (see relation 5.3.6). As the size of the file increases, the number of times
blocks are allocated is less in proportion to the size of the file.

The Readsq operation was also measured on files previously created with more than one extent. In this case, the times were similar to those obtained when measuring this operation on files having no multiple extents.

Total time measurements in the operations involving data transfer (Readsq and Writesq), are summarized in table 5.3.5 (excluding seek times). They include the request building time in the client's program, data transfer overheads (i.e. block transfers over the links and disks) and the average basic time for the operations (see table 5.3.4). Time variation can be seen to be related to the different configurations used to communicate with the filestore. The test configuration for evaluation of the two filestores included two main dissimilarities: firstly, different facilities were incorporated for the transfer of data between filestore memory and the external interfaces. The old filestore made use of the Automatic I/O facility, while the new filestore uses a programmed loop; secondly, a number of clients can communicate with the new filestore via Departmental Ethernet links, a facility that was not available on the previous network configuration.
Figure 5.3.1: Writesq (Average Response Time)

[Diagram showing response time for different file sizes and filestores, with 'old filestore' and 'new filestore' curves, and an 'average (no space allocation)' line.]

File size [blocks]

[10 30 50 70 90 110 130 150 170 190 210 230 250 270 290]
<table>
<thead>
<tr>
<th>Filestore (Client)</th>
<th>Operation</th>
<th>Link</th>
<th>Readsq</th>
<th>Writesq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block I/O</td>
<td>Mux (slow)</td>
<td>41.5</td>
<td>55.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mux (Fast)</td>
<td>30.0</td>
<td>45.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>29.5</td>
<td>44.0</td>
<td></td>
</tr>
<tr>
<td>Programmed Loop</td>
<td>Mux (slow)</td>
<td>42.5</td>
<td>56.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mux (Fast)</td>
<td>29.3</td>
<td>45.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>25.0</td>
<td>44.3</td>
<td></td>
</tr>
<tr>
<td>Block I/O</td>
<td>Mux (slow)</td>
<td>36.5</td>
<td>54.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mux (Fast)</td>
<td>25.5</td>
<td>42.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>20.5</td>
<td>39.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ethernet</td>
<td>44.4</td>
<td>61.0</td>
<td></td>
</tr>
<tr>
<td>Programmed Loop</td>
<td>Mux (slow)</td>
<td>38.4</td>
<td>54.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mux (Fast)</td>
<td>27.0</td>
<td>43.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>22.7</td>
<td>40.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ethernet</td>
<td>50.5</td>
<td>67.0</td>
<td></td>
</tr>
</tbody>
</table>

The request building time included in the total operation times of table 5.3.5, corresponds to the time between when the client's program started to build the command request and the time when the last part of the request arrived at the server. For those clients connected to the multiplexor, this time was approximately 1 msec and for clients connected directly to the filestore, approximately 0.5 msec. Concerning nodes connected via Departmental Ethernet, this time represented a much greater overhead (approximately 8.5 msec) due to the extra software delays added at the Ethernet controllers at both ends of the communicating entities. This includes factors like packet building times, the protocol of communication between the host and
the station (i.e. Protocol-less mode for the client, and Protocol mode for the filestore), the station-to-station acknowledgement and the transmission time of the request packet through the Ether. Timing details describing these different sources of overhead, on different client/filestore liaisons through the Ethernet, were given in section 5.2. Considering these performance figures (see figure 5.2.6) and the Read/Write operation times obtained in the current two-stroke exchange between client and filestore (illustrated in figures 5.3.2 and 5.3.3), we can derive what percentage improvements can be expected on these operations from the different liaisons.

In table 5.3.6 we have summarized these percentages.

Table 5.3.6: % Time saved from current client/server Ethernet liaison (E in figure 5.2.6)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readsq</td>
<td>35.6</td>
<td>10.6</td>
<td>17.3</td>
<td>28.7</td>
<td>0.0</td>
</tr>
<tr>
<td>Writesq</td>
<td>26.8</td>
<td>8.0</td>
<td>13.0</td>
<td>21.6</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The significance of delays introduced by the Ethernet network access mechanism, may also be analysed on the basis of the total times obtained for the client/filestore data transfer operations. Taking into account the current characteristics of the Edinburgh Ethernet, i.e. 2 Mbps, about 15 usec slot-time, and an average packet of 150 bytes (see figure 5.2.8), we can derive the following. If the load imposed over the network is below 50% of the carrying capacity of the
physical network medium, message delays incurred at the network access level can be considered negligible, compared with the times that elapse at disk(s), cpu(s) and communication interfaces (see figures 5.3.2 and 5.3.3). Only if the load increases beyond about 85%, can we expect that the response time degradation due to contention will be noticeable to the terminal user.

The file management service time was also measured in the presence of a controlled offered load. The filestore was not intended to support specialised data management services like data base access or backing storage. Thus, the level of processing activities at the filestore under congestion, does not require sophisticated procedures to cope with the concurrency of processing. The processing activities at the filestore, i.e. cpu times, represent a small proportion of the elapsed time involved in read/write requests (see figure 5.3.2 and 5.3.3). Thus, as the load increases, the waiting times at the client will increase principally due to the greater disk activity involved, and when using the Ethernet, due to delays at the network contention level and the network interfaces (e.g. possible buffer unavailability). As there is just one disk controller, the impossibility of concurrency of disk operations represents a major bottleneck when demand for disk resources becomes high.
Figure 5.3.2: READSQ (Time Components)

Transfer Method: Filestore (Programmed Loop)
Client (Block I/O)
Processor (Client): (1) and (4): 174
(2) and (3): 170

A: Filestore Processing Time
B: Average Disk Transfer Time (no seek)
C: Maximum Disk Transfer Time (no seek)
D: Request Time
E: Data Communication overhead
F: E (if Client uses Programmed Loop)

[ Msec. ]

58
54
50
46
42
38
34
30
26
22
18
14
10
6
2

ETHERNET

(MUX)

Slow Links

(DIRECT)

Fast Links

14

10

6

2

(1) (2) (3) (4)

5-68
Figure 5.3.3: WRITESQ (Time Components)

Transfer Method: Filestore (Programmed Loop)
Client (Block I/O)
Processor (Client): (1) and (4): 174
(2) and (3): 170

A: Filestore Processing Time
B: Average Disk Transfer Time (no seek)
C: Maximum Disk Transfer Time (no seek)
D: Request Time
E: Data Communication overhead
F: E (if Client uses Programed Loop)

[ Msec. ]

(MUX)
Slow Link

(MUX)
Fast Link

(DIRECT)
Fast Link

ETHERNET

5-69
The level of congestion that typical client applications produce is analysed in section 5.4. The pattern of demand consists basically of Read/Think/Read or Write/Think/Read sequences, where "Think" refers to the client's processing time between I/O requests. A system designed to generate a composite of several of these request patterns was implemented. The load was assembled by changing the request frequency from a variable number of clients (loaders). The loaders and a clock providing the timing for the requests were linked together in a daisy chain scheme, using Departmental links, to allow the load changing to be synchronized (i.e. by sending from one machine to another signals at regular intervals controlled by the clock).

We have displayed time distributions under different load levels, consisting of just Read requests (figure 5.3.4) or alternate Read/Write requests (figure 5.3.5). The observed times in the latter case were greater than in the former due to the presence of disk verification on each write operation (see relation 5.3.4). The observed times do not include client/filestore data communication delays, which in this case are independent of the load imposed.
Figure 5.3.4: READSQ
(Time Measurements Under Various Loads)

(Excluding Client/Filestore Data Transfer Time)
Pattern of Requests From Loaders: Read

<table>
<thead>
<tr>
<th>[Msec.]</th>
<th>#Loaders</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Disk1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Disk1, Disk2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Two in Disk2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Three in Disk1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Request Frequency [Msec.]
Figure 5.3.5: READSQ
(Time Measurements Under Various Loads)
(Excluding Client/Filestore Data Transfer Time)
Pattern of Requests From Loaders: Read/Write

<table>
<thead>
<tr>
<th>[ Msec. ]</th>
<th>#Loaders</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>_______</td>
<td>1</td>
<td>Disk1</td>
</tr>
<tr>
<td>_______</td>
<td>2</td>
<td>Disk1, Disk2</td>
</tr>
<tr>
<td>_______</td>
<td>3</td>
<td>Two in Disk2</td>
</tr>
<tr>
<td>_______</td>
<td>4</td>
<td>Three in Disk1</td>
</tr>
</tbody>
</table>

120 110 100 90 80 70 60 50 40 30 20 10

10 20 30 40 50 60 70 80 90 100 110
Request Frequency [ Msec ]
As the load increases, that is as the inter-request interval decreases, times approach the sum of the average individual times from the loaders, plus seek times if the case arises. For example, in figure 5.3.4, consider the case involving just two loaders making requests involving different disks (i.e. no track seeking is expected). In this case, the observed times were always less than the average Readsq operation time in absence of seek (see (3) in figure 5.3.2). The difference is about 3 msec, which is approximately the filestore Readsq/Writesq cpu processing time. This is explained by the fact that when the filestore is engaged in disk activities, an overlapping of operations (cpu and disk) is possible. The maximum time overlapping is precisely the above cpu processing time.

In figure 5.3.5, times for the same case above approach the average of the sum of the Readsq and Writsq operations (without including seek time).

Analysis of File Management Service Proposals

Here, we discuss some possible alternative strategies to the current file management service operations. The feasibility of any new strategy is discussed, aiming for improvements upon client service performance.
a) Buffering policies for space allocation activities

Space allocation activities impact the performance of operations involving reading and writing of files (see relations 5.3.4 to 5.3.6). Overheads refer to the extra activities upon user-directories and bitmap tables.

At present, bitmap blocks are allocated to one buffer in memory. Bitmap transfers to/from the disks may be required either during the Openw or Writesq operations. The first operation does not represent a significant overhead compared with total times involved in typical client applications (see section 5.4). If normally the task for allocating space is carried out in the Openw operation, then increasing the buffering capacity to hold bitmap blocks in memory will not offer a significant improvement.

When space is allocated at the time the file is being written, the maximum number of bitmap transfers to/from disk that might be required for an n-blocks file is equal to $2^*K_m$ (see relation 5.3.5). The constant 2 refers, as formerly explained, to the fact that two bitmap transfers (reading and writing) may be involved whenever bitmap activities are required. The relation above represents a worst case bit-map disk transfer, for it assumes that whenever an extent is needed, the required bitmap is not in memory. At present, within one partition, bitmap
activities are normally concentrated in just one of the potential 16 pages available per partition. Under these circumstances, the minimum probability that the requested bitmap is in memory at the moment it is required, is \(1/4\) (as there are 4 partitions). Then, considering that a bitmap activity may include two disk transfer times (Trdblk and Twrblk in table 5.3.3), which represent approximately 77.5 msec, we can estimate that the maximum time a Writesq operation can expend due solely to bitmap activities, may approach the (probabilistic) value, 
\[
(77.5\times Km(1-B/4))/n \]
where \(B\) is the number of in-memory bitmap blocks, and \(n\) the size, in blocks, of the file to be written. Under these assumptions, we can predict what maximum performance improvement might be obtained if the bitmap buffer capacity were enlarged by a given amount.

<table>
<thead>
<tr>
<th>File Size</th>
<th>Km</th>
<th>in-memory bitmap blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>(Km)</td>
<td>(2)</td>
</tr>
<tr>
<td>3 - 6</td>
<td>2</td>
<td>8.6</td>
</tr>
<tr>
<td>7 - 14</td>
<td>3</td>
<td>5.5</td>
</tr>
<tr>
<td>15 - 30</td>
<td>4</td>
<td>3.4</td>
</tr>
<tr>
<td>31 - 62</td>
<td>5</td>
<td>2.1</td>
</tr>
<tr>
<td>63 - 126</td>
<td>6</td>
<td>1.2</td>
</tr>
<tr>
<td>127 - 254</td>
<td>7</td>
<td>0.7</td>
</tr>
<tr>
<td>255 -&gt;</td>
<td>8</td>
<td>0.4 -&gt;</td>
</tr>
</tbody>
</table>

Table 5.3.7 displays, for different numbers of in-memory bitmap blocks, the maximum amount of time, in msec, expected to be saved in the Writesq operation when it involves space allocation. The file size (value \(n\))

5-75
was taken as the average of the two extreme values of table 5.3.7. Certainly a Writesq operation will not always require to undertake the task of allocation of space, and also under normal loads the concurrency of processes will not necessarily lead to bitmap activities every time an extent is needed. Times above represent the maximum expected time improvements, for the total writing operation, as the bitmap buffer capacity is enlarged. The conclusion is that these time enhancements can be considered relatively important only for small files. This is in relation to the total time of a Writesq operation when it involves space allocation (see figure 5.3.1).

b) Prefetching

At the filestore, a pool of dedicated processes deal with client requests, sharing a set of buffers for exchanging data with the clients. From the context of client's applications (Isys80) it will be observed (see section 5.4) that many of them require consecutive use of a substantial number of contiguous blocks from the filestore. This fact could be used to provide for response time enhancements, by making an extra transfer of blocks from disk to memory in advance of their use. The filestore could handle a pool of memory blocks, aiming to favour those applications (i.e. transactions) requiring high transfer rates. The feasibility of this
strategy has to be analysed under the possible constraints of buffering to maintain the extra context of non-volatile data.

The filestore could take advantage of the idle states to read, after each ReadSq operation, the next block in the file in advance of its requirement. The full advantages of prefetching would occur in situations when, while engaged in this operation, no other request is pending. From direct measurements (see section 5.4), we have obtained that in many of the client's applications the request frequency of filestore resources is higher than the disk time activities at the filestore (about 40 msec considering seek time). Under low loads, the prefetching scheme could produce noticeable enhancements of response time at the terminal level.

In a state of high loads, if the filestore buffering capacity is enough to sustain the volume of data generated by prefetching, times involving this operation can be regarded as normal disk transfer times (i.e. in absence of prefetching). The maximum time that can be saved by this strategy is the disk access time, and this is achieved when the inter-arrival time is greater than or equal to the disk access.

In order to have a real perspective of how prefetching would enhance the performance of reading operations, it
is necessary to look at the time proportion involved in disk access compared with communication overheads and cpu times. These time percentages are summarized in table 5.3.8, considering two client/filestore network interfaces, according to the results obtained from previous experiments (see figure 5.3.2). They were obtained assuming the current mode of programming used by both the client and the filestore (i.e. a programmed loop). Values in brackets summarize the case when the average seek time is included in the access time of the disk. The time percentage devoted to disk access could be regarded as the proportion of saved time, for each Read request, if a 1-block prefetching scheme were to be followed. We are assuming the case when the average frequency of I/O requests is less than the average disk access time.

Table 5.3.8: Readsq Operation (Time Percentages)

<table>
<thead>
<tr>
<th></th>
<th>Direct Link</th>
<th>Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Building Time</td>
<td>2.2 (1.0)</td>
<td>17.8 (11.2)</td>
</tr>
<tr>
<td>Data Communication</td>
<td>46.6 (20.0)</td>
<td>59.4 (37.3)</td>
</tr>
<tr>
<td>C.P.U Time (Filestore)</td>
<td>13.8 (5.9)</td>
<td>6.1 (3.8)</td>
</tr>
<tr>
<td>Average Disk Access</td>
<td>37.4 (73.1)</td>
<td>16.7 (47.7)</td>
</tr>
</tbody>
</table>

Hence, under these assumptions, we can predict how the prefetching scheme would improve the performance of user's applications (e.g. assembling, editing). Evaluations upon a number of them were undertaken and are reported in the following section.
5.4 Impact of Network Performance on System Performance

This section is devoted to describing a series of timing experiments related to user applications upon single-user systems, carried out on the Interdata cluster which form part of the Edinburgh local area network. On these machines, most of the user applications demand a series of Input/Output requests from the central filestore. Here, we deal with the problem of finding time distributions in such a local network environment for several of the capabilities that this system provides to the user. The principal aim is firstly, to give a comparative performance analysis between different configurations as seen by users and secondly, to obtain time profiles of computations versus data transfers and pattern of traffic demand. Through this analysis we expect to obtain a comprehensive view of the pattern of access to resources originated by small-memory single-user systems. This kind of system can be expected to generate a high load on the network. Subsequently, consideration is given to the likely implications for machines with larger memories and greater processing power.

Low-memory Single-user Systems

In previous experiments (see section 5.3) we have
estimated what time profiles the client can expect in its
dialog with the filestore. This analysis was conducted
within a context comprising various configurations (speed
and type of communication paths). Through evaluations on
some of the filestore capabilities, we obtained time
distributions on the client-filestore communication. As
far as the client was concerned, the fastest interface to
the filestore was direct-connection through Departmental
1 Mbps links. Now some questions arise: what time
distributions are expected by the user? What computer
configuration issues impact upon these time distributions
and how? What is the effect upon the traffic generated
over the network? Before attempting an answer, it is
useful to establish the interface boundaries that are
presented to the user. The server, in this case the
central filestore, communicates with the client (i.e. the
operating system), through several communication
interfaces and protocols. The client, in turn, offers to
the user (i.e. at the terminal attached to the computer)
different capabilities depending on the hardware
environment of the machine. All Interdatas in the
Department use the same operating system, so that the
main variables in the user-client interface are: memory
size and machine model, the latter affecting instruction
repertoire and execution time, with the Interdata 70
being the fastest and the Interdata 7/16 the slowest of
the machines.
Table 5.4.1 depicts the possible configurations of the Interdatas and their connection to the network. Of the 54 possible interface combinations to the filestore, five were made the subject of the first evaluations (Jan 82) and after the incorporation of the experimental Departmental Ethernet, we selected a sixth configuration. These six configurations are summarized in table 5.4.2.

Table 5.4.1: Interdata local area network configuration

<table>
<thead>
<tr>
<th>Machine Model</th>
<th>Memory Size</th>
<th>Connection Type</th>
<th>Transmission Speed (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/16</td>
<td>64 k</td>
<td>via MUX</td>
<td>0.26</td>
</tr>
<tr>
<td>74</td>
<td>16 k</td>
<td>Direct</td>
<td>1.00</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>Ethernet</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Table 5.4.2: Selected configuration for evaluations

C1: (74, 16 k, Via MUX, 0.26)
C2: (70, 64 k, Via MUX, 0.26)
C3: (70, 16 k, Via MUX, 1.00)
C4: (70, 64 k, Direct, 1.00)
C5: (7/16, 64 k, Direct, 1.00)
C6: (74, 16 k, Ethernet, 2.00)

These experiments involved selecting an appropriate activity-context for the client and establishing the execution times and other performance figures for that activity. In this way, we derived time profiles for typical applications on the different configurations. The major activities on Isys80 lie principally in the area of assembling and editing (see section 3.3). We
selected for evaluation these two operations together with compilation and program loading. Thus, for a file of a certain size, we established how long it takes, for example, to assemble it, to search through it from top to bottom, and so forth.

Experiment Environment

Every client activity generates a series of Input/Output requests to and from the filestore. These are mainly requests for opening and closing files, and requests for reading and writing sequentially stored blocks, the last two being the most frequently used operations. Knowing the time attributable to the sequence of I/O requests and the total time (i.e. chronometer time) of a specific client activity, then the distribution of client execution times can be estimated. On these machines there is no facility to measure cpu time directly for a running program.

The specific set of Isys80 applications (commands) upon which we carried out the timing experiments were,

Editing:

a) Find: String searching. Search the rest of the file, starting at the file pointer, for the first occurrence of a specific string. In the evaluations we considered the case when the searching goes through the entire file, for a non-existent string.
b) **Bott:** Move the file pointer to the end of the file starting at the beginning of the file.

c) **Top:** Move the file pointer back to the start of the file starting at the end of the file.

d) **Close:** Close File. The file being edited is written on disk. Here, the timing experiments always start with the file pointer exactly at the middle of the file.

**Loading:**

a) **Load:** Read information from a source stream (file) into store. We simulated the pattern of activities required for program loading by involving a file transfer with the destination specified as null.

**Program Translation:**

a) **Hal:** Assembling in Hal. Carry out an assembly of a program in the HAL-70 programming language, which is a high-level assembly language. The assembler is two-pass. The timing experiments consider separately the cases with and without an output listing (**Hal/L**).

b) **M68:** Assembling in M6809. Call the Motorola M6809 microprocessor assembler.

c) **IMP:** Compilation in IMP. Compile a program with the Interdata version of the IMP77 compiler. Both the compilation and linking processes were included in evaluations.

As we are not measuring user behaviour, the activity-context selected may not necessarily represent
normal user operations. However, they include the most heavily used Isys80 applications which, for our purposes of establishing comparative performance analysis between different configurations, can be considered satisfactory.

The editing commands represent the subset of the editor facilities which generate the largest amount of data traffic with the filestore. The Load command here represents the activity of the Isys80 loader, which is often required in a user session. Both HAL-70 and M6809 are assembly facilities widely used in the Department. The assemblers and compilers cannot be run in the 16-kbyte machines, which use the multiplexor for this purpose.

Times were computed using a chronometer or Interdata machines as clocks, the machines being operated from the control console switches. To get real time profiles from each application, the timing experiments were carried out in the absence of congestion (i.e. no other clients interacting with the filestore).

Evaluations were carried out by using a set of test files with sizes ranging from about 30 blocks to as much as 280 blocks (that is 15 to 140 kilobytes). This set was used on all the different clients. We estimate that each measurement may have been subject to an error of about 300 milli-seconds. This could represent a maximum
error of about 5% in times derived from the observed times.

**The Operation Context**

The Interdata screen editor maintains in local memory only as much of the file being edited as will fit in the available space, which is about 3 kilobytes for a 16k machine and about 20 kilobytes for a 64k machine. Moving from one end of the file to the other involves the writing and reading of all the blocks in the file, less the number which can be held in local memory. Thus, on starting, the client's available memory is filled sequentially as the demand arises, each earlier block being moved up in memory (this strategy minimising space overheads at the expense of cpu). When the memory portion becomes full, and a new block is required from disk, the top block is written back to disk. Hence, when this position is reached, the traffic with the filestore is increased as extra requests (writes) are generated for each block read.

**Estimates of cpu**

If Fs and Ms are the size, in blocks, of the file and the client's memory respectively, then, for example, for the operations Find and Bott, if Fs≤Ms, a total of \((Fs(Fs+1)/2)-1\) in-memory block transfers will be
generated. On the other hand, if $F_s > M_s$, a total of $M_s(M_s+1)/2-1 + M_s(F_s-M_s)$ blocks will have to be shifted in memory. The client's execution time, for all the editing operations, can be stated as:

If $F_s \leq M_s$,

- $CLEX(\text{Find}, F_s) = (F_s(F_s+1)/2-1)T_{mov} + F_s T_{find}$ \hspace{1cm} (5.4.1)
- $CLEX(\text{Bott}, F_s) = (F_s(F_s+1)/2-1)T_{mov} + F_s T_{bott}$ \hspace{1cm} (5.4.2)
- $CLEX(\text{Top}, F_s) = F_s T_{top}$ \hspace{1cm} (5.4.3)
- $CLEX(\text{Close}, F_s) = ((F_s-F_p)(F_s-F_p+1)/2-1)T_{mov} + F_s T_{close}$ \hspace{1cm} (5.4.4)

And if $F_s > M_s$,

- $CLEX(\text{Find}, F_s) = (M_s(M_s+1)/2-1+M_s(F_s-M_s))T_{mov} + F_s T_{find}$ \hspace{1cm} (5.4.5)
- $CLEX(\text{Bott}, F_s) = (M_s(M_s+1)/2-1+M_s(F_s-M_s))T_{mov} + F_s T_{bott}$ \hspace{1cm} (5.4.6)
- $CLEX(\text{Top}, F_s) = M_s(F_s-M_s)T_{mov} + F_s T_{top}$ \hspace{1cm} (5.4.7)
- $CLEX(\text{Close}, F_s) = M_s(F_s-F_p)T_{mov} + F_s T_{close}$ \hspace{1cm} (5.4.8)

where, $T_{mov}$ is the processing time involved in shifting one block in memory, and $T_p$ the processing time, per block, for the operation $p$ ($p = \text{Find, Bott, Top or Close}$). $F_p$ corresponds to the number of blocks from the top of the file to the file pointer.

The Load command involves no in-memory transfers so that in this case,

- $CLEX(\text{Load}, F_s) = F_s T_{load}$ \hspace{1cm} (5.4.9)

In each application, the whole operation includes the
activities related to the exchange of data between the client and the filestore. Hence, the total time \((T_{tot})\) for the operation \(p\), is given by,

\[
T_{tot}(p,Fs) = CLEX(p,Fs) + IOEX(p,Fs) \quad (5.4.10)
\]

where, \(IOEX(p,Fs)\) is the time involved in Input/Output exchanges between the filestore and its client (see relations 5.4.11 to 5.4.16). Total times for all the user applications evaluated here follow the relation 5.4.10.

It can be observed that when editing a file whose size is bigger than the capacity of the larger memory partition (34 blocks in this case), a small Interdata may have a better overall time performance compared with a larger one. This difference increases as the file size increases.

**The Measurements**

Figures 5.4.1 to 5.4.6 illustrate the measurement data obtained for total operation times for the user applications outlined above. As can be seen on the editing commands, the slope of the lines changes at the point where the size of the file becomes equal to the size of the client's memory partition. This is more clearly visible on those configurations having a bigger
memory partition (C2, C4 and C5).

As the operation time for the Load command (see figure 5.4.1), does not depend upon the client's memory size, differences in the observed times are related solely to the speed of the client's processor involved and the transmission rate through the different communication interfaces. For example, configuration C5 which connects directly to the filestore, gives a worse response time than configuration C2 which connects to the filestore via the multiplexor through a slower link (0.26 Mbits/sec). This is attributable to the difference between the client's processor speed on the two configurations. C6 includes the Departmental Ethernet links, and the same host processor as C1. The observed values show how significant are software delays at the Ethernet controller station (C6) in the total loading operation.

In figure 5.4.3, the reference points a to f illustrate time response variability for a given file size, in the Bott operation. Some of these time differences are analysed below. Calling the difference between any two points x and y, for a given file size Fs, \( D_{xy}(Fs) \) we have,

Factor \( D_{pd}(Fs) \) denotes the difference between the processor speed in both clients, which from relation 5.4.6 can be expressed as,
\[ D_{ed}(Fs) = Ms(Fs-Ms)[T_{mov}(C5)-T_{mov}(C4)] + Fs[T_{bott}(C5)-T_{bott}(C4)] \]

where, \( T_{mov}(Cx) \) and \( T_{bott}(Cx) \) are the corresponding processing times at the client in configuration \( Cx \).

Factor \( D_{fd}(Fs) \), denotes only the different proportion of time that elapses in the transfer of bytes through the communication components on the two configurations \((C2, C4)\). The function CLEX (see relation 5.4.6), should be the same on both configurations, so that we can derive from relation 5.4.12 and table 5.4.4 in both pertinent cases (Direct and Mux), that when \( Fs > Ms \), \( D_{ed} \) is equal to \( 15.9 + Fs + (Fs-Ms) 13.5 \).

Factor \( D_{fe}(Fs) \) is concerned with both sources of variation outlined above because both the processor speed and the type of linkage to the filestore differ in configurations \( C5 \) and \( C2 \).

Points a, b and c are a common reference for the smaller machines. As above, in factor \( D_{fd}(Fs) \), factor \( D_{ba}(Fs) \) can be derived in a similar fashion.

As configurations \( C1 \) and \( C6 \) relate to the same client machine, factor \( D_{ob} \) can also be worked out from table 5.4.4 on the pertinent interfaces. In this case, \( D_{ob}(Fs) \) is equal to \( 12.1Fs + (Fs-Ms) * 13 \).
Figure 5.4.1: Load (Elapsed Time)

File Size [Blocks]

Figure 5.4.2: Find (Elapsed Time)

File Size [Blocks]
**Figure 5.4.3: Bott (Elapsed Time)**

**File Size [Blocks]**

```
<table>
<thead>
<tr>
<th>File Size [Blocks]</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```

**Figure 5.4.4: Top (Elapsed Time)**

**File Size [Blocks]**

```
<table>
<thead>
<tr>
<th>File Size [Blocks]</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```
Figure 5.4.5: Close (Elapsed Time)
(file pointer at the middle of the file)

File Size (Blocks)

Figure 5.4.6: Assembling & Compilation
(Elapsed Time, Configuration C4)

File Size (Blocks)
Concerning the differences between the set of points (d,e,f) and (a,b,c), they relate, as explained above, not only to dissimilarities in processor speeds and communication interfaces, but also the size of the client's memory. Figures 5.4.2 to 5.4.4, corroborate that, when editing, response time is strongly dependent on the client's memory size. Similar functions can be obtained for the factors relating the differences between these points, using the same relations and table 5.4.4.

The figures above also illustrate those coordinates where the small-memory clients gain in performance compared with the larger ones. These performance characteristics are dependent upon the pattern of data transfer generated by each command and upon the speed at which the data flows through the interfaces. For example, in command Bott, for a file size above approximately 70 blocks, configuration C1 offers a better response time than configuration C4. Concerning command Top, the same occurs but for a file size greater than 100 blocks.

As can be derived from the figures above, relation 5.4.10 associates the total time $T_{tot}$ and the size of the file as, $T_{tot} = A \cdot F_s + B$.

Relating $T_{tot}$ and $F_s$ as the dependent and independent variables respectively, of a linear regression, by least
squares, the coefficients of the regression can be estimated. Table 5.4.3 summarizes these coefficients upon each interface. They represent, in figures 5.4.1 to 5.4.4 the lines above the point where the gradient of the lines changes (i.e. \(F_s > 3\) in configurations C2, C4 and C5, and \(F_s > 6\) in configurations C1, C3 and C6)

Table 5.4.3: Regression Coefficients in \(T_{tot} = A F_s + B\)

<table>
<thead>
<tr>
<th>Oper. Coeffs.</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>102.1</td>
<td>200.0</td>
<td>85.9</td>
<td>183.9</td>
<td>208.0</td>
<td>116.0</td>
</tr>
<tr>
<td>B</td>
<td>-300</td>
<td>-4500</td>
<td>-200</td>
<td>-4300</td>
<td>-5000</td>
<td>-250</td>
</tr>
<tr>
<td>Bott</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>135.0</td>
<td>230.0</td>
<td>112.0</td>
<td>200.3</td>
<td>224.0</td>
<td>156.0</td>
</tr>
<tr>
<td>B</td>
<td>-550</td>
<td>-5200</td>
<td>-700</td>
<td>-5000</td>
<td>-5200</td>
<td>-500</td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>152.2</td>
<td>235.0</td>
<td>140.7</td>
<td>213.7</td>
<td>221.6</td>
<td>168.0</td>
</tr>
<tr>
<td>B</td>
<td>-2000</td>
<td>-7100</td>
<td>-2000</td>
<td>-6500</td>
<td>-5700</td>
<td>-2200</td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>134.2</td>
<td>115.9</td>
<td>99.7</td>
<td>99.5</td>
<td>132.3</td>
<td>150.0</td>
</tr>
</tbody>
</table>
| B            | neglig.| neglig.| neglig.| neglig.| neglig.| neglig.

Coefficient A quantifies, for each configuration, the time increment (in msec) that can be expected for each processed block. Coefficient B, for the first three commands, is negative because the lines that are being represented do not include the values of \(F_s\) between 0 and \(M_s\) blocks.

It should be pointed out that, in general, user files tend to be small (say below 60 or 50 blocks). On this basis, in figures 5.4.2 to 5.4.4, we should probably place more emphasis on those points that represent small files rather than big ones. At these, in general, the
smaller machines offer worse performance than the bigger.

Data Transfer Versus Client's Computation

The previous paragraph was confined to the description of total time profiles, as seen by users, on a number of client's applications which converse with the filestore through different communication pathways. Here, based on these measurement results, we derive time profiles of client's computation versus data transfer.

Function IOEX (see relation 5.4.10) was derived by monitoring the request/response sequences generated by each operation. This was done through the system control terminal attached to the filestore. The pattern of exchange sequences was obtained for each of the applications being evaluated. The following is a simple analytical specification of the time involved in the I/O exchanges between the filestore and its clients, on each application.

\[
\begin{align*}
\text{IOEX}(\text{Find}, F_s) &= (F_s)T_{\text{read}} + q[(F_s-M_s)(T_{\text{read}}+T_{\text{write}})] \quad (5.4.11) \\
\text{IOEX}(\text{Bott}, F_s) &= \text{IOEX}(\text{Find}, F_s) \quad (5.4.12) \\
\text{IOEX}(\text{Top}, F_s) &= q[(F_s-M_s)(T_{\text{read}}+T_{\text{write}})+T_{\text{read}}+T_{\text{Topw}}] \quad (5.4.13) \\
\text{IOEX}(\text{Close}, F_s) &= (F_s-F_p)T_{\text{read}} + (F_s-q[F_p-M_s])T_{\text{write}} \quad (5.4.14) \\
\text{IOEX}(\text{Load}, F_s) &= F_sT_{\text{read}} + T_{\text{Openr}} \quad (5.4.15)
\end{align*}
\]

where, \( q \) is equal to 0 if \( F_s \leq M_s \) and equal to 1.
otherwise.

Regarding assembling and compilation, factor IOEX involves operations not only related with one file, but with a number of them. Below, we summarize this function for the assembly operations (HAL and M68), considering the case without an input listing.

\[
\text{De}(\text{Ass}, F_s) = 3T_{\text{openr}} + 3T_{\text{closer}} + T_{\text{openw}} + T_{\text{closew}} + (A_{sz} + A_{xz} + nF_s)T_{\text{read}} + F_{so}T_{\text{write}}
\]  

(5.4.16)

where, Asz is the size of the n-pass assembler (21 and 75 blocks for the HAL and M6809 assemblers respectively), Axz the size of an auxiliary file (1 block for both assemblers), and Fso the size of the generated object file. For the Hal assembler, n is equal to 2, and for the M6809 assembler n depends on the complexity of the source code (generally n=6).

Topenr, Topenw, Tread and Twrite correspond to the filestore operation times, as seen by clients, required to open a file for sequential reading, to open a file for sequential writing, to read a block from the filestore, and to write a block to the filestore respectively. Time measurements upon these operations, carried out concurrently with these evaluations led to the times shown in table 5.4.4 (see also table 5.3.5)
As can be observed, times in the \textit{Writesq} operation are much less than the corresponding total times displayed in table 5.3.5 (on clients using a programmed loop). In fact, the former only include client/filestore data exchange times (D+E in figure 5.3.3). The reason for considering just this I/O overhead for deriving the real time profile of client's execution times is as follows. For a write operation, the acknowledgement from the server occurs before it starts to undertake the disk transfer operation itself. Hence, disk transfer and client's processing times overlap. And in fact, the latter (being precisely Tmov) is greater than the former (this was obtained, a priori, by a rough calculation of the actual times from the instructions in the Editor).

\begin{table}[h]
\centering
\begin{tabular}{lcccccc}
\hline
 & C1 & C2 & C3 & C4 & C5 & C6 \\
\hline
Topenr & 2.6 & 2.6 & 2.6 & 2.1 & 2.1 & 2.1 \\
Topenw & 32.2 & 32.2 & 32.2 & 31.4 & 31.4 & 60.1 \\
Tread & 38.4 & 38.4 & 27.0 & 22.5 & 22.5 & 50.5 \\
Twrite & 25.5 & 25.5 & 14.2 & 12.5 & 12.5 & 38.5 \\
Tclosew & 77.6 & 77.6 & 77.6 & 77.0 & 77.0 & 90.0 \\
\hline
\end{tabular}
\caption{Filestore operation times. Configurations (see table 5.4.2)}
\end{table}

The pattern of demand from the user applications consists of either Read request or Write/Read request sequences. In most of the editing commands, the latter occurs if the file size (Fs) is greater than the client's memory available for the editing process (Ms). Then, since, in general, the read and write sites will be on
different cylinders of the disk, a seek time (Tsk) is likely to be present between each sequence of reading and writing. Thus, the factor (Fs-Ms)Tsk should be added in relations 5.4.11 to 5.4.13. Concerning compilation and assembling, this time is also present, at least every time successive requests are carried out on different files. However, it is reasonable to assume that this component in the timing figures was much smaller than the average seek time of the disk, because files were always confined to be in one partition.

In the previous section, we estimated what percentage improvement could be obtained, on the Read operation, if a prefetching scheme were to be used (see average disk access in table 5.3.8). Then, on the basis of the total elapsed times obtained in this section (see figures 5.4.1 to 5.4.6), we can estimate what percentage of the time, for any of the user applications, could be reduced if prefetching were in operation. To attain this we should also establish what proportion of the data transfer time is devoted to the reading operation (see relations 5.4.11 to 5.4.16). For example, for the assembly process in HAL (see relation 5.4.16), on a client using direct links to the filestore, without taking into account seek times (configuration C4 in table 5.4.4), this percentage is approximately 3.2%. Adding seek times, this is approximately equal to 12.5%. Concerning the editing operations (Find, Bott and Top), these percentages, being
about the same for each, correspond approximately to 5% and 17% respectively.

Figures 5.4.7 to 5.4.12, represent the function CLEX (see relation 5.4.10), divided by the number of I/O blocks involved in each operation. Therefore, these figures illustrate the average frequency of requests that are generated by the different operations from each individual machine. They quantify the average network load expected from an individual single-user system, expressed as the average inter-arrival of requests. These, from the client's point of view, represent either a 516-byte message request (a Write request) or a 3-byte message request (a Read request). Thus, for example, an assembly process in HAL, generates a load over the network of approximately 0.5% of the current Ethernet carrying capacity (2.1 bits/usec). Similarly, a small Interdata executing some of the editing operations generates a load that represents about 3% of the network bandwidth. Hence, under the current pattern of user-activities and current number of clients, the network can be expected to be free of congestion.

For each individual command, the pattern of demand differs for each configuration due to differences either in the client's processor speed and/or memory size. Concerning commands Find, Bott and Close, the average inter-request time increases with the size of the file,
because the proportion of in-memory block transfers to the size of the file increases. But, this rise drops gradually after Fs becomes equal to Ms as write requests are generated. In command Top, when Fs≤Ms, no I/O requests are generated, which explains the different shape of the curves in figure 5.4.10 compared with those of the other editing commands.

Regarding assembling and compilation, the function CLEX is illustrated in Figure 5.4.12, for the configuration C4 only. In Table 5.4.5 we summarize the relative differences in processing speed obtained for the different configurations, expressed as fractions of the slowest speed.

<table>
<thead>
<tr>
<th>Config. command</th>
<th>C1 or C6</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hal</td>
<td>0.94</td>
<td>0.90</td>
<td>0.94</td>
<td>0.87</td>
<td>1.00</td>
</tr>
<tr>
<td>M68</td>
<td>0.83</td>
<td>0.80</td>
<td>0.83</td>
<td>0.77</td>
<td>1.00</td>
</tr>
<tr>
<td>IMP</td>
<td>0.81</td>
<td>0.79</td>
<td>0.81</td>
<td>0.76</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Nearly 55% to 58% of the data in the test files used for assembling in Hal corresponded to symbolic code, the rest being comments or blank lines. For the files used to evaluate M68 and IMP, this percentage represented approximately 65% to 70%.
Figure 5.4.7: Load
(Average Frequency Between I/O Requests)

File Size [Blocks]

Figure 5.4.8: Find
(Average Frequency Between I/O Requests)

File Size [Blocks]
Figure 5.4.9: Bott
(Average Frequency Between I/O Requests)

File Size [Blocks]

Figure 5.4.10: Top
(Average Frequency Between I/O Requests)

File Size [Blocks]
Figure 5.4.11: Close
(Average Frequency Between I/O Requests)

File Size [Blocks]

Figure 5.4.12: Assembling & Compilation
(Average Frequency Between I/O Requests)
Configurations C4

File Size [Blocks]
Figure 5.4.13: Load (% I/O Activities)
(i.e. excluding client's computations)
Including average seek time

File Size [Blocks]

Figure 5.4.14: Find (% I/O Activities)
(i.e. excluding client's computations)
Including average seek time

File Size [Blocks]
Figure 5.4.15: Bott (% I/O Activities) (i.e. Excluding client's computations) Including average seek time

File Size [Blocks]

Figure 5.4.16: Top (% I/O Activities) (i.e. Excluding client's computations) Including average seek time

File Size [Blocks]
Figure 5.4.17: Close (% I/O Activities)
(i.e. Excluding client's computations)
(1ncluding Average Seek Time)

File Size [Blocks]

Figure 5.4.18: Assembling & Compilation
(% I/O Activities; Configuration C4)
Including average seek time

File Size [Blocks]
The percentage of data transfer time to total time, which can be expressed as \(100 \times \frac{IOEX}{T_{tot}}\), is depicted in figures 5.4.13 to 5.4.18. To obtain these percentages we have included in the total times the average seek time of the disk for each I/O request generated. This would represent the extreme case regarding access time under presence of interfering or competing operations from the same or other activities.

The figures, showing another view of the pattern of requests from individual users, also serve to provide a source of information regarding the relative contribution to congestion that individual applications may generate. On the other hand, we can infer from the different client's activities, how many of them might be occurring simultaneously without apparent disruption of individual user-performance. For example, looking at figure 5.4.15, on configuration C4, we might predict that two of these single-user systems involved in that particular operation (Bott), will not degrade individual performance if they are executed simultaneously. On the other hand, as can be deduced from figure 5.4.18, approximately five HAL assembly processes running on different single-user systems, could coexist without noticeable degradation in response time. This is valid assuming that the performance at the processing level in the server does not decrease by the presence of these simultaneous applications. As was analysed in the previous section,
the level of performance degradation of the present filestore under congestion can be considered low.

Conversely, taking the complement of the percentages (i.e. 100-%) of figures 5.4.13 to 5.4.18, we can establish how important computing time is in the total response time of the different user applications. Concerning the editing operations, in the 16 kbyte machines (configurations C1, C3 and C6), this figure ranges approximately from 20% (for the Bott operation) to 25% (for the Find operation). In the 64 kbyte machines (Configurations C2, C4 and C5), it ranges from about 50% to 60%. Regarding assembling and compilation (see figure 5.4.18), computing time ranges from 80% (for the Hal operation) to 88% (for the M68 operation). So that computing time is clearly significant in many cases relative to the time involved in I/O activities.

Large-memory Single-user Systems

The previous evaluations focused upon single-user systems with very limited main memory, accessing filestore files on a block by block basis. Here, based on the previous measurement results, we extend the analysis of the performance on individual applications, to single-users with relative high memory capacity.
Large memory mini-computer systems are likely, in the future, to constitute the typical client accessing the Departmental network resources. Presently, the Department is developing a powerful multi-purpose single-user system, with a memory capacity of 0.5 Mbytes, built around a M68000 processor. A system of this kind is able to hold entire user or system files in memory. Hence, an in-progress application can be much more independent from a network filing system, compared with a smaller system. Similar applications on both kind of systems, conversing with a central file server, may lead to substantial differences in both the response time and the pattern of demand generated. We can expect that the larger the client's memory, the less is the frequency at which clients access resources through the network. This kind of system can retain much of the data needed as the operation (e.g. editing, assembling) is in progress, and file updating onto disk can be postponed until the file is closed. For example, a file being edited will generate only read requests as the user moves forward through the file and when eventually the file is closed, the whole file already read into memory will be written back on disk. As a result, the alternate Read/Write pattern of requests that low-memory systems commonly generate is less likely to be demonstrated in large-memory systems. These would tend to originate request patterns consisting of read sequences followed by write sequences, but not alternately.
We now analyse, for the user-applications already evaluated, the pattern of data exchange expected on large-memory systems, when running similar user-applications. According to the relations 5.4.11 to 5.4.16, we can derive the following simple specifications,

\[
\text{IOEX(Find,Fs)} = \text{Fs} \times \text{Tread} \quad (5.4.17)
\]

\[
\text{IOEX(Bott)} = \text{IOEX(Find,Fs)} \quad (5.4.18)
\]

\[
\text{IOEX(Top,Fs)} = 0 \quad (5.4.19)
\]

\[
\text{IOEX(Close,Fs)} = (\text{Fs-Fp})\text{Tread} + \text{FsTwrite} \quad (5.4.20)
\]

\[
\text{IOEX(Load,Fs)} = \text{Fs Tread} + \text{Topenr} \quad (5.4.21)
\]

Turning to assembly operations, for an n-pass assembler that is able to maintain in memory the source file during the entire operation, the function IOEX, will not depend upon n (see relation 5.4.16). In this case, assuming similarity of the assembling systems, this function would be,

\[
\text{De(Ass,Fs)} = 3\text{Topenr} + 3\text{Tcloser} + \text{Topenw} + \text{Tclosew} + \\
(\text{Asz + Xsz + Fs})\text{Tread} + \text{FsoTwrite} \quad (5.4.22)
\]

By subtracting time IOEX in both systems, on the corresponding operations, we can derive specifications for the time saved in I/O exchanges (SIOEX) in a large-memory system compared with a low-memory system. We have,
According to the above specifications, we can also derive how the number of requests (Reads and Writes) in a large-memory system, would decrease relative to a small system. This amount \( RQR \) is summarized below for each application.

\[
\begin{align*}
RQR(\text{Find}, Fs) &= (Fs - Ms) \\
RQR(\text{Bott}, Fs) &= RQR(\text{Find}, Fs) \\
RQR(\text{Top}, Fs) &= 2(Fs - Ms) \\
RQR(\text{Close}, Fs) &= -(F_p - Ms) \\
RQR(\text{Load}, Fs) &= 0 \\
RQR(\text{Ass}, Fs) &= (n-1)Fs
\end{align*}
\]

Functions \text{SIOEX} and \text{RQR} have been quantified in table 5.4.6, for several file sizes, assuming applications running on a system connected to the filestore through the Ethernet, and considering \( Ms=6 \) for the small system (e.g. a typical Interdata 74). The negative values in the Close operation are due to the fact that in this operation, a large-memory system increases its I/O activities rather than decreasing them compared with the
small system.

Table 5.4.6: I/O parameters reduction (large-memory system)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Ethernet Configuration</th>
<th>(Fs)</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>£ requests</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Fs)-&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find, Bott</td>
<td></td>
<td></td>
<td>1.7</td>
<td>3.6</td>
<td>5.5</td>
<td>7.5</td>
<td>44</td>
<td>94</td>
<td>144</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>Top</td>
<td></td>
<td></td>
<td>3.9</td>
<td>8.4</td>
<td>12.8</td>
<td>17.3</td>
<td>89</td>
<td>189</td>
<td>289</td>
<td>389</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td></td>
<td></td>
<td>-0.7</td>
<td>-1.1</td>
<td>-2.1</td>
<td>-3.6</td>
<td>-19</td>
<td>-44</td>
<td>-69</td>
<td>-94</td>
<td></td>
</tr>
<tr>
<td>Assembling</td>
<td>(n=2)</td>
<td></td>
<td>2.3</td>
<td>4.5</td>
<td>6.7</td>
<td>8.9</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>
6 Conclusions

Throughout this dissertation, we have given a quantitative performance analysis, using measurements on actually implemented systems, comprising a number of cooperating computer and communication systems integrated in a local network.

In the local area network evaluated, where a number of single-user systems compete for file resources, a major bottleneck is disk access. Disk activities like track seeking and rotational latency may involve delays that represent almost 50% of the total client I/O request times. Given the actual pattern of resource demand of filestore clients, a simple block-at-a-time prefetching scheme could partially overcome this major overhead. This policy could lead to improving total response times by approximately 17%, in some cases.

By comparison, the current load on the communication network generated by typical user-applications, on the average, consumes only a small percentage of the available bandwidth and hence, on the Ethernet, the number of collisions and retransmissions is expected to be small. Thus, we can assume that network throughput is not a limiting factor at significant levels. Hence, the current network configuration (e.g. bandwidth, maximum
packet size, station packet buffer storage etc.) can be considered satisfactory. The incorporation of large-memory single-user systems to the network would perhaps not increase average network traffic levels beyond those currently generated. However, the pattern of traffic of such an evolving network may well differ from the current pattern.

The processing overhead at the file-server does not represent significant delays in the process of accessing files as seen by the user. But aspects of client's configurations like processor speed, size and management of memory add considerably to total delays of some user-applications.

Considering the current performance characteristics of the higher level protocols of the network, the importance of the low level access mechanism in the overall performance of clients accessing network resources, can be considered to be significant only for traffic loads beyond approximately 85%. Thus, if the current hardware and control mechanisms (e.g. Backoff algorithm, Collision Consensus Reinforcement procedure, clock granularity) can sustain the actual slot-time at levels given by this low-scale network (i.e. 10 to 15 usec), the contention level will not add significant access restrictions at high loads.
Any attempt to increase network bandwidth, apart from being currently unnecessary, would have to be considered with care. To get the full benefits of this increase, in case of high load conditions, the maximum internetwork packet size, as well as the station buffer capacity would also have to be increased. Moreover, a higher cpu speed at the network interface would have to be provided to avoid packet losses when these arrive closely after each other. Current client/filestore protocol packet boundaries may have to be reorganized to get the full benefits of increasing buffer and cpu network interface capabilities.

The mismatch that exists between the rate at which data can be delivered to the host and the rate at which the host can absorb it, is relatively high. This represents about a factor of four considering the host I/O capabilities currently being used. The current buffer availability at the network adapters is not high enough to avoid overflows under conditions of heavy load. Concurrency of processing activities at the station controller and data transfer at the host, while the former is engaged in preparing an acknowledgement, could partially overcome this mismatch.

In addition, it is clear that the current firmware on the existing processor introduces additional delays which, though not major, are non-trivial. As a
consequence of all these factors, it is clear that the overhead for each packet is high. One aspect of this is that even though the low-level acknowledgement policy does not add considerably to byte volumes, it doubles the number of packets over the network with obvious consequences for network responsiveness at high loads. It would seem worthwhile to consider the possibility of handling acknowledgement at the higher level or other strategies aimed at reducing packet volumes over the network.
7 References

[Aim79] Guy T. Aimes & Edward D. Lazowska
"The Behaviour of Ethernet-Like Computer Communication Networks"

"Architecture of Distributed Computer System"

[Car79] Robert J. Carpenter and Joseph Sokol
"Serving Users With a Local Area Network"

[Cla78] D.D. Clark, K.T. Program, and D.P. Reed
"An Introduction to Local Area Networks"

[Cra80] R. C. Crane and E. A. Taft
"Practical Considerations in Ethernet Local Network Design"

[Dew77i] H. Dewar
"ISYS80 - Reference Manual",
Department of Computer Science, Edinburgh University, 1977.

"The Filestore"
Department of Computer Science, Edinburgh University, 1977.

[Dew78] H. Dewar
"HAL 70, a High Level Assembly Language for Interdata Series 70 Minicomputers"
Department of Computer Science, Edinburgh University 1978.

"The Ethernet: Data Link Layer and Physical Layer Specifications"
[Dio80] J. Dion
"The Cambridge File Server"

[Fre80] H. A. Freeman & K. J. Thurber
"An Updated Bibliography on Local Computer Networks"

[Ge178] Erol Gelenbe, Jean L. Grange and Pratice Mussard
"Performance Limits of the TMM Protocol: Modeling and Measurements"

FIPNET: A 10 MBPS Fiber Optics Local Network

[Gru77] Dana S. Gruff & Ira W. Cotton
"Criteria for Evaluation of Data Communication Services"

[Han82] I. B. Hansen & W. D. Enos
Edinburgh CSMA/CD Local Area Network
Department of Computer Science, Edinburgh University, 1982.

[Int73] Interdata INC

[Kle75] Leonard Kleinrock
"Queueing Systems"

[Lnd76] P. J. Lindsay & H. Dewar
"AA Link Coaxial Line Standards", Draft Department of Computer Science, Edinburgh University, 1976.

"Simulation of a Local Computer Network"

[Mar81] M. Marathe & S. Kumar
"Analytical Models for an Ethernet-like Local Area Network Link"
[Pax79] William H. Paxton
"A Client-Based Transaction System to Maintain Data Integrity"

[Mag82] Peter S. Mager
"Toward a Parametric Approach for Modeling Local Area Network Performance"

[McL81] Paul L. McInerney
"The design of a Network Filing System"
(Edinburgh Ph.d Thesis) Department of Computer Science, Edinburgh University, 1981.

[McF73] T. F. McFadden and J. C. Strauss
"Quantitative Evaluation of File Management Performance Improvements"

[Mit82] James G. Mitchell & Jeremy Dion
"A comparison of Two Network-Based File Servers"

[Met76] Robert M. Metcalfe and R. Boggs
"Ethernet: Distributed Packet Switching for Local Computer Networks"

[Sal81] Jerome H. Saltzer and David D. Clark
"Why a Ring?"
Seventh Data Communications Symposium, Vol. 11, No. 4, pp. 211-217, 1981.

[Sho80] John F. Shoch
"An Annotated Bibliography on Local Computer Networks"

[Sho80] John F. Shoch and Jon A. Hupp,
"Measured Performance of an Ethernet Local Network"

[Sho81] John F. Shoch
"An Introduction to the Ethernet Specification"

[Spa79] Otto Spaniol
"Modelling of Local Computer Networks"
[Stu80] H. Sturgis, J. Mitchell & J. Israel  
"Issues in the Design and Use of a Distributed File System"  

"WFS: a Simple Centralized File System for a Distributed Environment"  

[Tan76] Andrew S. Tanenbaum  
"Structure Computer Organisation"  

[Tan81] Andrew S. Tanenbaum  
"Computer Networks"  
Prentice-Hall, 1981.

[Wat79] W. B. Watson  
"Simulation Study of the Traffic Dependent Performance of a Prioritized, CSMA Broadcast Network"  

[Wat80] W. B. Watson  
"Performance in Contention Bus Local Network Interconnection"  

[Wat81] W. B. Watson  
"Clarifications on Performance in Contention Bus Local Network Interconnection"  

[Wil80] Maurice V. Wilkes and Roger M. Needham  
"The Cambridge Model Distributed System"  

[Yeh79] Jeffry W. Yeh  
"Simulation of Local Computer Networks - a case Study"  

[Zim80] Hubert Zimmerman  
"O.S.I. Reference Model: The I.S.O. Model of Architecture for Open Systems Interconnection"  
Appendix

Programs used for Evaluations (brief description)

The programs that are listed below (in alphabetic order), can be found in the filestore user directory FRA. They constitute the main part of the systems used for evaluations throughout this dissertation.

CLK. This is a software implemented clock, that permits timing of operations running in another machine (transmitter) to which the clock (receiver) is point-to-point interconnected. This program can run in any Interdata client machine.

LDRETH. Used in cooperation with a low-level firmware packet generator (see LDRFW), this is used to set up different loads through the Ethernet. The load is considered as a proportion of the available bandwidth and it is computed as a function of the packet size and the inter-packet transmission time. Individual throughput from each loader can be obtained by considering the total number of bits delivered and the total sampling time. Load from different loaders can be synchronized by connecting the loaders directly via Departmental links in a daisy chain scheme.

LDRFW. This is a low-level firmware packet generator for the Ethernet network interface. It allows only
interrupts from its host (see LDRETH) for load changing (inter-packet delivery time and packet size). Successfully transferred packets and collisions are reported to the host.

MBLR. Enables the reception of a large amount of data (e.g. 1 Mbytes) from another machine (see MBLT) connected via Departmental links. It can be used to measure byte transfer times through these links. Once the reception has been completed it lights the lights at the control console. Hence, by using a chronometer, the transfer time can be obtained. This time can also be obtained by providing point-to-point connection with a clock (see CLK). Two different data transfer methods at the processor level can be selected.

MBLT. This is used to send a large amount of data to another machine (see MBLR) connected via Departmental links. Two different data transfer methods at the processor level can be selected.

MSGDLY:R. Enables the measurement of host-to-host or station-to-station message delays through the Ethernet. Packets and clock signals are provided by another machine (see MSGDLY:T), connected point-to-point to the machine running this program.

MSGDLY:T. Provides for packet sequences through the Ethernet, for the measurement of message delays at the host and network interface levels. Different host I/O capabilities and network services can be set up. A direct link connection with a receiver machine (see
MSGDLY:R) should be provided.

**FCM.** Allows the direct timing of a number of filestore capabilities like Readsq, Writesq, Openw etc. Some options for each capability can be evaluated too. It can also be used to measure client/filestore data transfer times through the different communication pathways (i.e. Ethernet and Departmental links) in which case, a point-to-point connection using Departmental links with a receiver (clock) machine (see CLK) should be provided.

**FCMLOAD.** Permits generation of various patterns of filestore requests from one client machine. Traffic may consist of sequences of Read, Read/Write or Write filestore requests. The load is changed by varying the request frequency (in msec), either automatically (signaled by a clock) or manually. In the former case, the clock (see CLK) and the loaders should be connected point-to-point in a daisy chain scheme.

**PPEETH.** Simulates the two-stroke protocol the filestore provides to its clients for Reading and Writing of sequentially stored blocks. Thus it can be used to measure the delays involved in this exchange (i.e. excluding the processing time at the filestore and disk access time). Three machines should be supplied. Two of them run this program, the first used for the "filestore" and the other for the client. The last should be a clock (see CLK), connected point-to-point with the client.