CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

EXPERIMENTAL ANALYSIS OF BROADCASTING ALGORITHMS TO PERFORM SET OPERATIONS

by

Enoch Oi-Kee Hwang

B.Sc. (Honours), University of British Columbia, 1982

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Computing Science

SIMON FRASER UNIVERSITY
September 1985

ISBN 0-315-30768-4

Approval

Name: Enoch Oi-Kee Hwang

Degree: Master of Science

Title of Thesis: Experimental Analysis Of Broadcasting Algorithms To Perform Set Operations

Examining Committee:

Chairman: Dr. Rob Cameron

Dr. Wo Shun Luk
Senior Supervisor

Dr. ~~Lou Hafer~~

Dr. Tiko Kameda

Dr. Jim Cavers
External Examiner
Faculty of Engineering Science
Simon Fraser University

July 12, 1985
Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Experimental Analysis Of Broadcasting Algorithms To Perform Set Operations

Author:

(signature)

Enoch Oi-Kee Hwang

(name)

September 3, 1985

(date)

Abstract

There has been a considerable number of algorithms developed for supporting a distributed database system in areas such as concurrency control, minimizing data transmission for set operations, crash recovery, and command processing. However, at the present time, most of these algorithms have only theoretical and/or simulated results concerning their performances. There is very little concrete knowledge concerning the actual performance of such algorithms in a real situation.

In order to obtain some empirical results, a set of these algorithms have been implemented. Specifically, they are algorithms for performing set intersections in a distributed database system based on a broadcast network. The result shows that for a small database, a static algorithm that fixes the order of the schedule at the start of processing performs better than a more sophisticated algorithm that orders the schedule dynamically.

In a distributed database system, the performance of the algorithms is also very much dependent on the method used to send the data. The method of broadcasting allows one processor to send its data out and all the rest of the processors attached to the same network will receive this data simultaneously. As a result, all the processors can now do parallel preprocessing of the data received. However, under the UNIX 4.2BSD operating system in which we did our experiments, broadcasting is only supported by the Internet User Datagram Protocol (UDP), and it turns out that

this protocol is unreliable. Thus, a high level reliable broadcasting protocol has to be designed on top of the UDP. Factors concerning the design of a high level reliable broadcasting protocol and the results for the protocols tested will also be presented in this thesis.

To my parents

Acknowledgement

Special thanks to my supervisor, Dr. W.S. Luk, for supporting and helping me throughout my work on this research. Also, thanks to my committee members, Dr. L. Hafer, Dr. T. Kameda, and Dr. J. Cavers for helping with the revision of the thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

Over the last few years, there has been a growing trend towards a more decentralized computer system. Instead of having one big main frame computer, several smaller computers located at different geographical areas but all connected together through a local area network are being used. These computers or workstations usually have their own resources such as processor, memory and disk storage. Each of the workstations can work independently or together by communicating with the others through the network.

We can quickly see the advantages in having a distributed database system in this environment where the data is stored on different computers. Such a system can therefore allow data to be physically stored close to the point where it is most frequently used - with obvious efficiency advantages - while at the same time permitting that same data to be shared by other, geographically remote users. These advantages are, however, balanced off with the numerous technical problems that also come with it. The main tradeoff is the problem of larger data transmission time required to send the data from one site to another over the network. For a non-distributed system, the main concern for a database system is often the number of disk accesses necessary in order to retrieve certain information from the database. When the database is distributed over a number of processing sites connected together

by a communication network, the amount of data transmission required to obtain the information is an additional consideration for system performance evaluation. In fact, many authors of papers on distributed databases have made the assumption that the disk access time is insignificant when compared with the data transmission time.

There has been a considerable number of algorithms developed for supporting a distributed database system in areas such as concurrency control [BeGo81], set operations such as joins, unions and intersections on a broadcast network [Luk84], crash recovery, minimizing the number of acknowledgements required for a broadcast network [ChMa84] and command processing. However, at the present time, most of these algorithms have only theoretical and/or simulated results concerning their performance. There is very little concrete knowledge concerning the actual performance of such algorithms in a real situation. The purpose of this research is to provide some empirical results for different algorithms to perform set intersections in a distributed database system based on a broadcast network.

Let us assume here that the entire database is a relation, which is basically a two-dimensional table [Codd71]. A row of a table represents a record and a column represents a set of values in the same field of all the records. In the distributed database environment, portions of the database, again as relations, are stored in various sites for the convenience of the users. Since the sites are all connected together by a communication network, the distributed nature of the database should be transparent to the users.

There may be requests for information that necessitate accesses to relations stored in more than one site. According to the relational model [Date81], there are three operations that may require data stored in two different relations, and hence

from two different sites. The first two operations are (row-wise) union and intersection of records belonging to different relations. If the records are long, it may be too expensive to move an entire record from site to site. By assigning each record a unique identifier, we may perform unions and intersections of these identifiers as preprocessing operations to select the records that are requested. The third operation, join, concatenates records from two relations together, if they have a common value in a common joining column. Semi-join [BeCh81] is an effective preprocessing operation for join to reduce data transmission. There are two steps in performing a semi-join operation of two relations in two different sites: 1. the joining column of a relation is sent to the other site, and 2. an intersection is performed between this column and the corresponding joining column to eliminate records (i.e. rows) that need not participate in the join operation. Thus in this sense, the semi-join operation and therefore the join operation is basically a set intersection operation. From the above discussion, we conclude that intersection and union of sets are important database operations, the sets being relations, groups of identifiers or joining columns.

1.2. The Problem Defined

From [Luk84], we have noted that the optimal algorithms for performing set intersection and union are very much similar. The data transmission is exactly the same. They differ in the local operation that they perform, either an intersection or a union. In fact, performing set union is simpler because it is unnecessary to reorder the sets according to their cardinalities in order to minimize the amount of data transmission, whereas for set intersection, this reordering makes a substantial difference in the amount of data transmitted.

In this thesis, we consider the problems of performing set intersections of data

sets distributed over a number of processing sites. To be more precise, it is a problem of gathering at a particular site, a set of data which satisfies a given set expression in a conjunctive normal form of a number of sets located in different sites. We will look at several algorithms for performing these queries in a distributed database environment and several transmission protocols for sending data between the sites. Since the data transmission time affects the overall performance of the algorithm a great deal, we want to look especially for ways to minimize this factor. The method of broadcasting provides just what is needed. It allows many processors to receive the same message at the same time when that message is broadcast out by one processor. As a result, all the sites can not only preprocess their data to eliminate redundant information, but do this preprocessing in parallel.

1.3. Objective of the Thesis

The optimal algorithm to perform set intersections on a broadcast network has been implemented in a real situation. The objective of this thesis is to compare the empirical results that have been collected from this implementation with the theoretical results presented in [Luk84]. The costs of the algorithms in terms of the transmission time, the local operation time, the handshake time, and the amount of data transmission are used as the comparison criteria. The empirical results from two other algorithms will also be compared. The first one requires no knowledge of the data sets and does not perform any preprocessing of the data, while the second one tries to reduce even more data transmission than the optimal algorithm. This is done at the expense of the overhead required to gather more information about the data sets and dynamically reordering the schedule. This analysis will provide us with the information about whether the theoretical results as presented in [Luk84] reflect real situations and whether the algorithm is optimal. If it is, then the algorithm is indeed

optimal, otherwise, analysis will be presented to explain the discrepancies between the theoretical results and the experimental findings.

Since the method of data transmission is such a vital part of the distributed set intersection algorithms, we have also analyzed several reliable broadcasting protocols for data transmission. The analysis of the empirical results for these protocols will also be presented and the criteria for a fast reliable broadcasting protocol will be described.

1.4. Organization of the Thesis

The organization of the rest of this thesis is as follows. In chapter 2, the broadcast network and its reliability are described. In chapter 3, we discuss the two internet protocols, UDP and TCP, and how they relate to broadcasting. Several reliable broadcasting protocols that we have tested are described in chapter 4. The results from these tests will also be shown in this chapter. In chapter 5, an optimal algorithm for performing set intersections in a broadcast network is described. This is followed by the description of three variations of the algorithm that we have implemented. The experimental setup in terms of the hardware and the software, together with a description of how the data is collected, is described in chapter 6. The rest of the chapter is devoted to the results and analysis of the three broadcasting algorithms to perform set intersections that we have implemented. Finally, chapter 7 contains the conclusion.

Chapter 2

The Broadcast Network

2.1. Introduction

In a distributed database system, the amount of data transmission over the network is dependent not only on the algorithm used but also very much on the topology of the network. In this research, we have selected to use the Ethernet network, which is a multiple-access broadcast network, as the basic communication architecture. In this chapter, we will discuss the components of the Ethernet, its reliability, and how it relates to our experiments.

2.2. The Ethernet

The Ethernet [MetBoggs76, DaBaPrSo79] is a system for local communication among computing stations. The shared portion of the Ethernet consists of the Ether, which is a passive medium for the propagation of digital signals and can be constructed using any number of media including coaxial cables, twisted pairs, and optical fibers. The Ethernet can be extended from any of its points in any direction by adding new segments which are joined together by repeaters. However, there must be only one path through the Ether between any source and destination; if more than one path were to exist, a transmission would interfere with itself. A station's Ethernet interface connects through an interface cable to a transceiver which in turn taps into the Ether. See figure 2-1. Our current experimental hardware configuration consists of five Sun Workstations, all connected together by an Ethernet network which operates at a speed of 10 megabits per second.

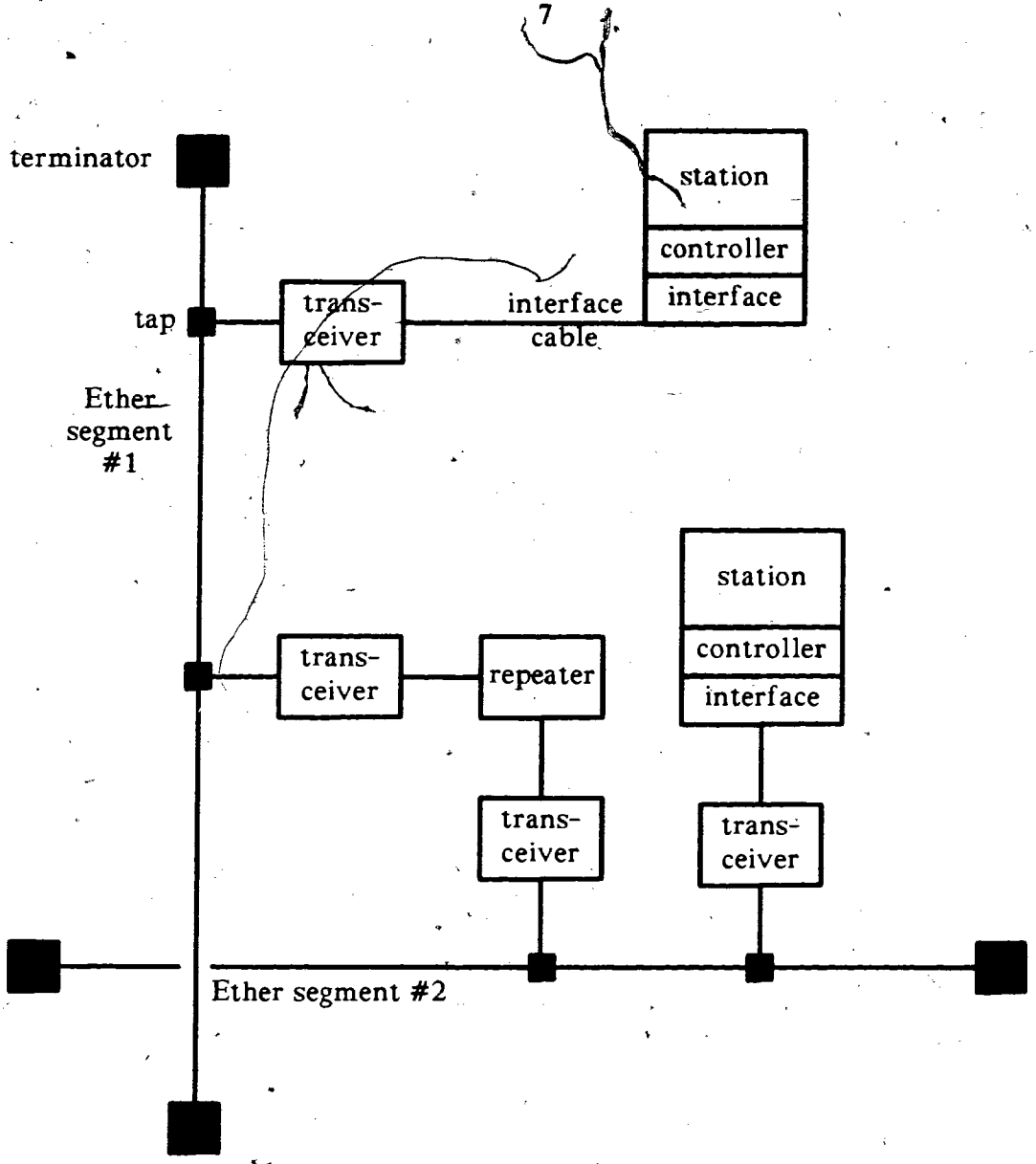


Figure 2-1: The Ethernet

2.2.1. Transceiver

The transceiver can be connected to or disconnected from the Ether at any point through the use of a tap, which is a simple device for physically connecting to the Ether. This procedure can be done at any time without disrupting any on-going communication on the network. Precautions must be taken to insure that likely failures in the transceiver do not result in pollution of the Ether. In particular, the transceiver should automatically be disconnected electrically from the Ether if power

is removed from it or if it acts suspiciously. A normal transceiver should be able to drive a kilometer of coaxial cable Ether tapped by up to 256 stations transmitting at 10 megabits per second.

2.2.2. Interface

The Ethernet interface is a device that basically performs three jobs. When a packet arrives at the transceiver, the interface hardware will check the address in the packet's header to see whether the packet is destined for this station. This hardware address filtering helps a station to avoid burdensome software packet processing when the Ether is very busy carrying traffic intended for other stations. If the packet is addressed for this station, it will be accepted.

After accepting a packet, the interface must convert the serial data from the network to parallel data used by the station (or vice versa if the station is sending out a packet).

Finally, a 16-bit cyclic redundancy checksum is calculated by the hardware on the serial data of the whole packet as it is transmitted or received.

2.2.3. Controller

The controller is a station specific low level firmware or software for getting packets onto and out of the Ether. It is responsible for the collision control of packets and the random delay time for the retransmission of collided packets.

In our research, the 3Com 3C400 Multibus Ethernet Controller is used. This controller combines the functions of both the interface and the controller into one, thus eliminating a separate interface.

2.2.4. Workstation

In our research, five Sun Workstations have been used, each representing a data node. The Sun Workstations are powerful general-purpose microcomputers using the 32-bit MC68010 CPU which operates at a speed of 10MHz. More will be said about the architecture of this machine in section 6.2.1.

2.2.5. Data Transmission

There is no central controller allocating access to the Ether, instead, a random access procedure is used in which each station independently decides when to transmit. A station with a packet to transmit first listens to the Ether using a carrier sense mechanism. If the Ether is idle, it immediately transmits the packet, otherwise, it waits until the transmission that is already in progress finishes. Once it hears the transmission cease, the station immediately transmits its packet. If no other station has been waiting, the station will acquire the Ether and the packet transmission should be successful. However, it is possible that two or more stations have been waiting and now they all sense the idle Ether and begin transmission simultaneously, producing a collision. Each sender, however, continues to monitor the Ether during transmission and detects collision when the signal on the Ether does not match its own output. Using a collision consensus enforcement procedure to ensure that all other colliding stations have seen the collision, the failure will be immediately apparent to all the transmitting stations and they can therefore abort the transmission immediately. The collided packets will be retransmitted after a random delay by the different stations in order to avoid repeated collisions. The use of a passive medium and the lack of any active elements in the shared portion combine to help provide a very reliable and flexible system.

2.3. Reliability of the Ethernet

One of the major objectives of any local network is to provide reliable communication facility, reflected both in the continued availability of the network itself and in the lowest possible error rate as seen by the individual hosts. Since the only shared component in the network is the passive coaxial cable with no active components, the overall reliability of the system is very high. However, packets are none the less subject to transmission errors. Thus, five mechanisms are provided by the Ethernet for reducing the probability and cost of losing a packet. These are (1) carrier detection, (2) interference detection, (3) packet error detection, (4) truncated packet filtering, and (5) collision consensus enforcement.

2.3.1. Carrier Detection

A packet's data is phase encoded on the carrier signal, thus a passing packet on the Ether can be detected by listening for its transitions. As a result, no station will start to transmit when there is a packet on the Ether. The only time when a collision can occur is when two or more stations find the Ether silent and begin transmitting simultaneously.

2.3.2. Interference Detection

Interference detection is done by the sending station and is indicated when the transceiver notices a difference between the value of the bit it is receiving from the Ether and the value of the bit it is attempting to transmit. The advantage of this is that the sender will know whether its packet has been damaged after a maximum of one round trip time. As a result, the packet can be scheduled for retransmission immediately without having to wait for an acknowledgement from the receiver. The frequency of detected interference is also used to estimate the Ether traffic for adjusting retransmission intervals and optimizing channel efficiency.

2.3.3. Packet Error Detection

A 16-bit cyclic redundancy checksum is computed and appended to each packet. Packets with unmatching checksums are discarded.

2.3.4. Truncated Packet Filtering

During transmission, packets may be truncated. Packets that are truncated by usually only a few bits are filtered out in hardware.

2.3.5. Collision Consensus Enforcement

When a station determines that its transmission is experiencing interference, it momentarily jams the Ether to insure that all other participants in the collision will detect interference and thus be forced to abort and retransmit after a random delay.

2.3.6. Reliability Results

With these five error reducing mechanisms, experiments have shown that the transmission error rate is about 1 in 2,000,000 packets [ShHu80]. It has also been shown that under normal load, 99.18 percent of the packets make it out with zero latency, and less than 0.03 percent of the packets are involved in collisions. This extremely small error and collision rate justifies our assumption that all the lost packets are due to problems in flow control and not in the data transmission itself. Hence, solving the problem of lost packets involves the design of a high level reliable protocol that manages the flow control.

2.4. Broadcasting and Multicasting

The significance of using the Ethernet local area network is that it is a multiple-access broadcast network. The Ethernet uses a bus or tree topology (see figure 2-2) and therefore, can support both broadcasting and multicasting.

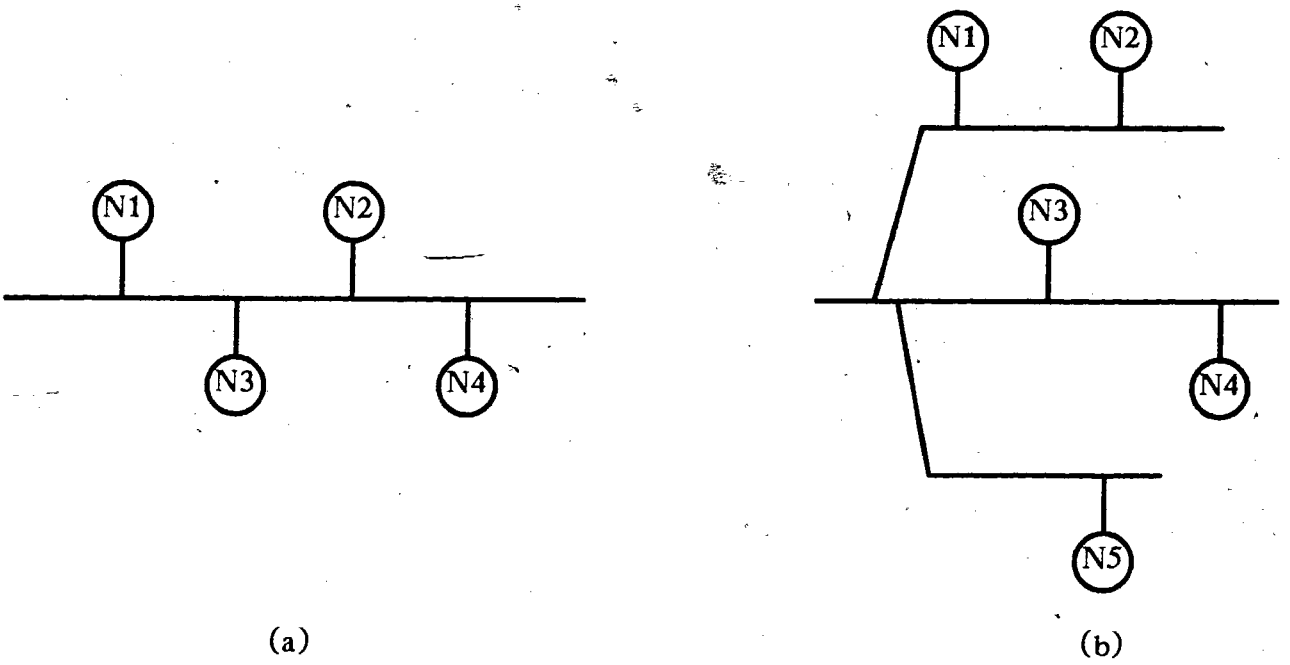


Figure 2-2: (a) The Bus and (b) Tree Topology of the Ethernet

In broadcasting, a transmitted packet will propagate the length of the medium and thus, all other stations connected to the network can receive this packet. Multicasting is very similar to broadcasting except that the packet being transmitted is received by only some subset of all the stations connected to the network. In both cases, the destination address in the packet header will, instead of containing a unique station address, contain a "wildcard address" which will match all or a subset of the stations' addresses.

If a message is to be sent to all stations on the network, a broadcast of this

message will provide a lower variance of transmission arrival time over separate single-destination transmissions. Since all the stations can now receive this message at the same time (with negligible transmission delays), the processors can now proceed to process the data simultaneously. In our distributed database application, preprocessing of the data is performed in this manner and will hopefully eliminate some redundant information. The eliminated information need not be transmitted later on in the process of doing the query.

Chapter 3

Internet Protocols

3.1. Introduction

The Internet network supports two protocols: the Internet User Datagram Protocol (UDP) and the Internet Transmission Control Protocol (TCP). The main difference between the UDP and the TCP is that the TCP provides a reliable, flow-controlled, two-way transmission of data through a connected socket, whereas the UDP is a simple, unreliable datagram protocol using a connectionless socket. The issues that relate to these two protocols in connection with our research will be discussed in this chapter.

3.2. Socket Types and Protocols

Within the Internet network, communication between two processes or nodes takes place between communication endpoints known as sockets. Each socket has the potential to exchange information with other sockets within the network. Several methods of communication are available and each is associated with a different socket type.

The two main socket types that the Internet supports are virtual circuit socket and datagram socket. These two socket types correspond to the two protocols TCP and UDP respectively. Thus, the way in which TCP and UDP work is inherent to the communication methods used by these two different sockets.

3.2.1. Datagram Packet Switching

For the datagram packet switching technique, a message that is to be sent is broken into smaller units called packets. The reason for this is that the length of the data that may be transmitted is limited in the packet-switched network. A typical maximum length is one to several thousand bits. In our experimental network, the maximum length is 1518 bytes with 46 bytes being the packet header and 1472 bytes for data. A message that is of length greater than the maximum packet size will be sent one packet at a time. Each packet is treated independently by the network. In addition, no dedicated path is established between the two communicating stations. As a result, each packet must have a destination address appended to it.

One advantage of the datagram approach is that no connection overhead is required. Thus if a station wishes to send only one or a few packets, this service is very fast.

3.3. TCP vs UDP

Our initial task is to investigate what the difference is in the transmission time between the two protocols - TCP and UDP. The experiment is very simple; we just check the time required to send a fixed number of fixed size packets using the two different protocols. The packet size used is four bytes and the time is the average elapsed time in seconds in sending 1000 packets to one node. The result is shown in table 3-1.

From this result, we can draw the following conclusions:

1. Connection time for TCP is long.

2. A small routing overhead for UDP.

3. TCP is reliable.

4. UDP is unreliable.

	Elapsed Time (seconds)		
Method	**Connection**	**Transmission**	**Total**
TCP	13.00	4.15	17.15
TCP with additional acknowledgement/packet	13.00	16.89	29.89
UDP without acknowledgement	0	6.32	error
UDP with acknowledgement/packet	0	17.04	17.04

Table 3-1: Transmission Time for TCP and UDP

3.3.1. Connection and Transmission Times

TCP uses the virtual circuit socket which requires an initial connection time for establishing the virtual circuit. The result shows that indeed the connection time for TCP is very long as compared to that of UDP. In fact, UDP does not require any connection time at all simply because the datagram socket is connectionless.

However, the transmission time for UDP without acknowledgement is 2.17 seconds longer than that of TCP. This is explained by the fact that for TCP, there are no record boundaries, so data can be packed tighter together. Thus, less packets are needed for the transmission of the file. For UDP, different records cannot be packed together because there are record boundaries, and so, more packets are needed for the transmission of the file.

3.3.2. Error Free Data Transmission

For error free data transmission, we can use either the TCP without acknowledgement scheme or the UDP with acknowledgement scheme. The result in table 3-1 shows that the transmission time for the TCP scheme is about four times less than that of the UDP scheme. However, with the large connection time for the TCP scheme, the total elapsed times for the two schemes are about the same. Figure 3-1 shows a plot of the total transmission time for sending different numbers of four bytes packets. From this, it is clear that the UDP scheme is much faster for sending less than 4K bytes of data. However, when more than 4K bytes of data is to be transmitted, the TCP scheme is faster. We might ask whether this result is dependent on the packet size at all. The answer is yes. When a packet size of 1472 bytes is used, we get the result as plotted in figure 3-2. Using a large packet size, it does not seem like that there is any crossover point between the two curves. Thus, the UDP scheme is much faster for sending any amount of data when a large packet is used. Moreover, we will see in the next section that it is better to use a larger packet size.

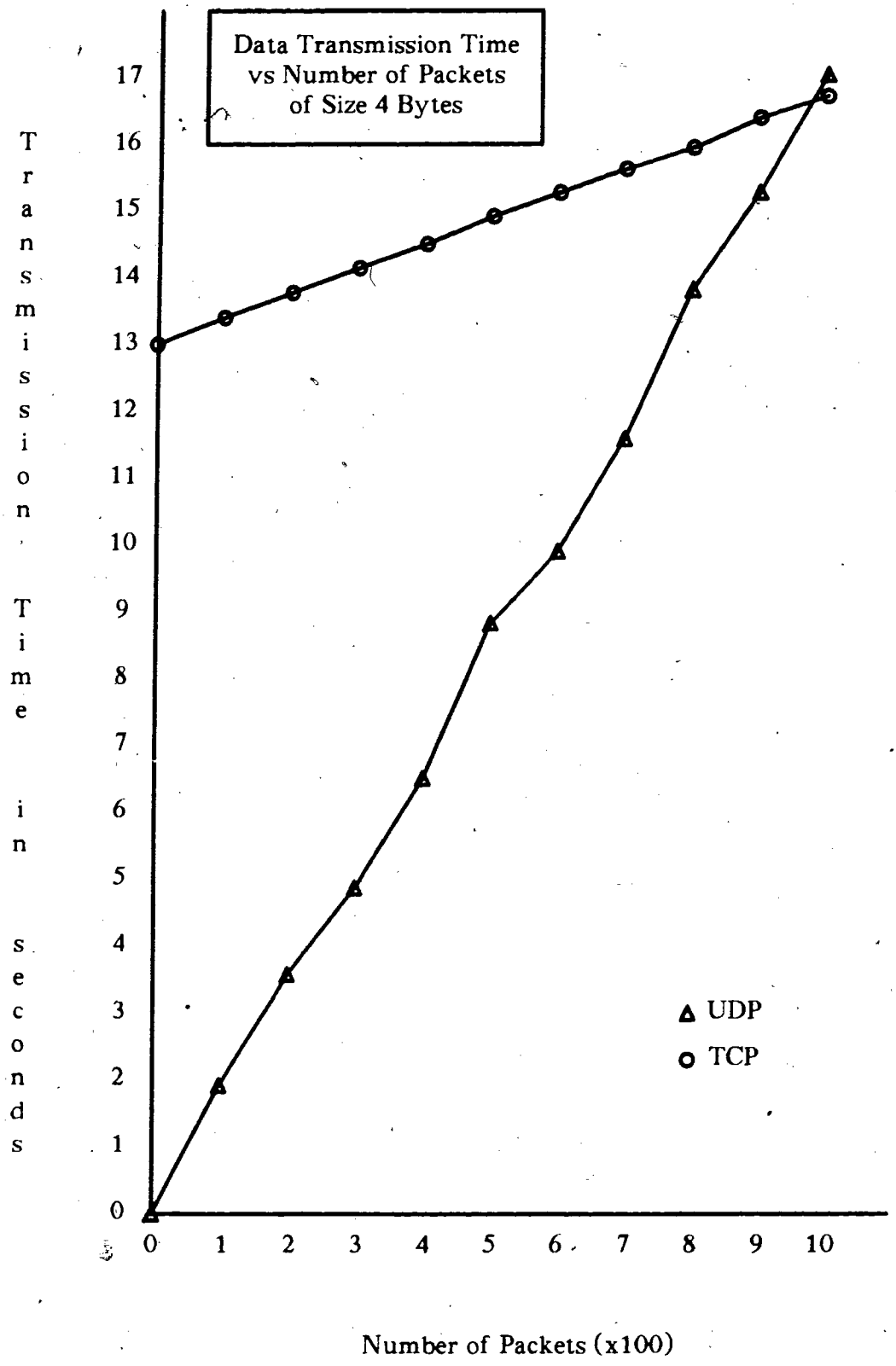


Figure 3-1: Transmission Time for Different Number of Packets of Size 4 Bytes

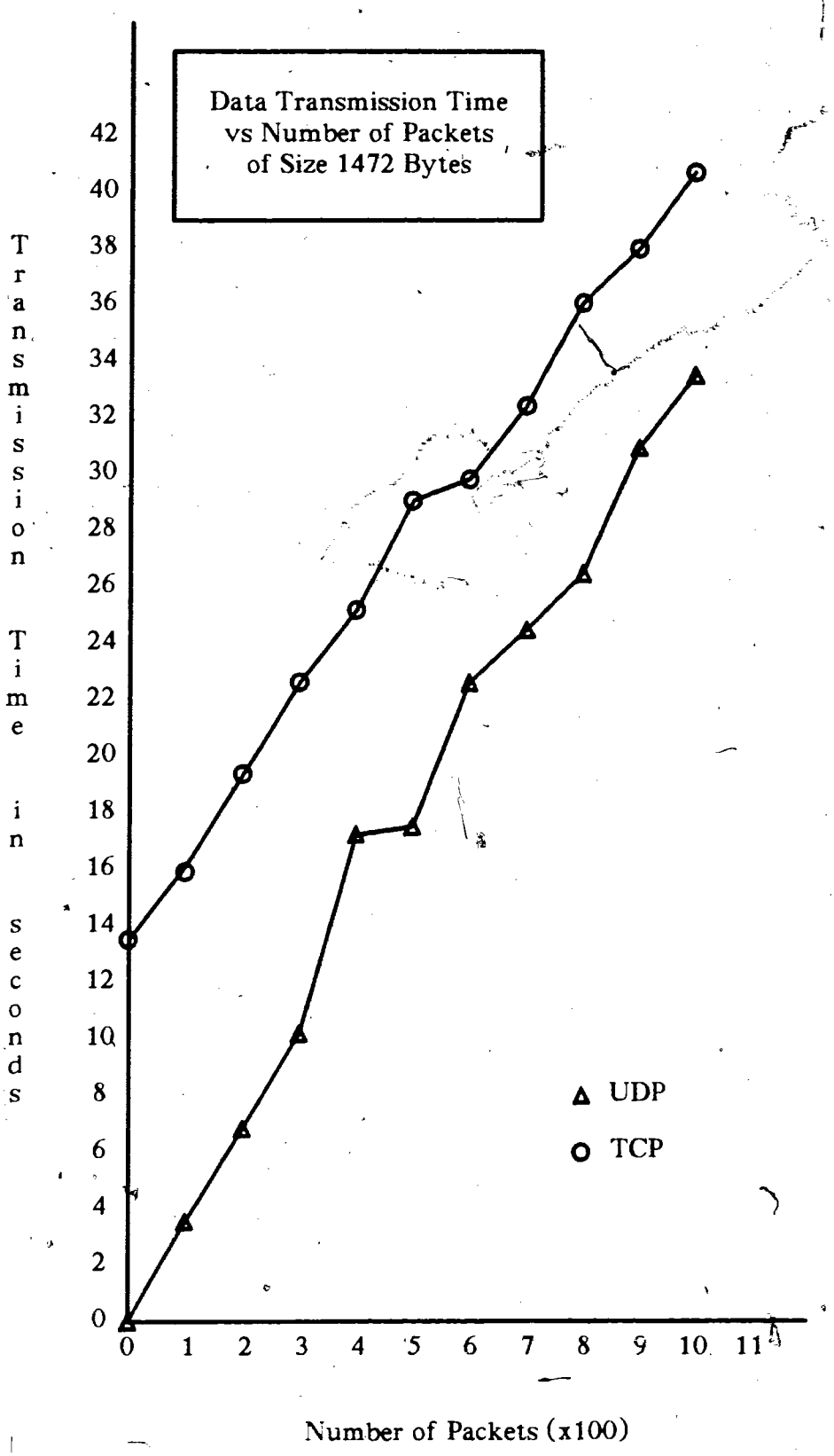


Figure 3-2: Transmission Time for Different Number of Packets of Size 1472 Bytes

3.4. Optimal Packet Size

In our experiments, we have found that even though the transmission time increases as the packet size gets larger (see figure 3-3), it is still better to use the largest possible packet size. The reason is that when we calculate the transmission time per byte, the time decreases as the packet size gets larger. This result is depicted in figure 3-4. This graph shows that for a data packet size of 400 bytes, the throughput rate for both the UDP and the TCP is the same. However, as the data packet size increases, the throughput rate of the UDP improves over the TCP throughput rate.

The dips and peaks in figure 3-3 are explained by the fact that the buffer size in the workstation is 512 bytes. In order to send a packet of size over 512 bytes, more than one buffer must be acquired. Thus, for the UDP case, the time increases as a buffer is being filled. When the buffer is full and a new buffer is acquired, the time decreases again because of an almost empty buffer. For the TCP case, the time changes are not as drastic as the UDP case. Note that the buffer holds, not only data but also header information such as the source and destination addresses. This is why the dips in the figure do not occur exactly at the 512 byte boundary.

Moreover, other experiments done by Shoch and Hupp [ShHu80] have shown that the larger the packet size, the better the utilization of the network.

There is, however, a limit to the size of a UDP packet. This restriction is due to the datagram packet-switch network itself. This maximum packet size is fixed at 1518 bytes including the packet header and 1472 bytes of data. Hence, all our experiments are done using this maximum packet size.

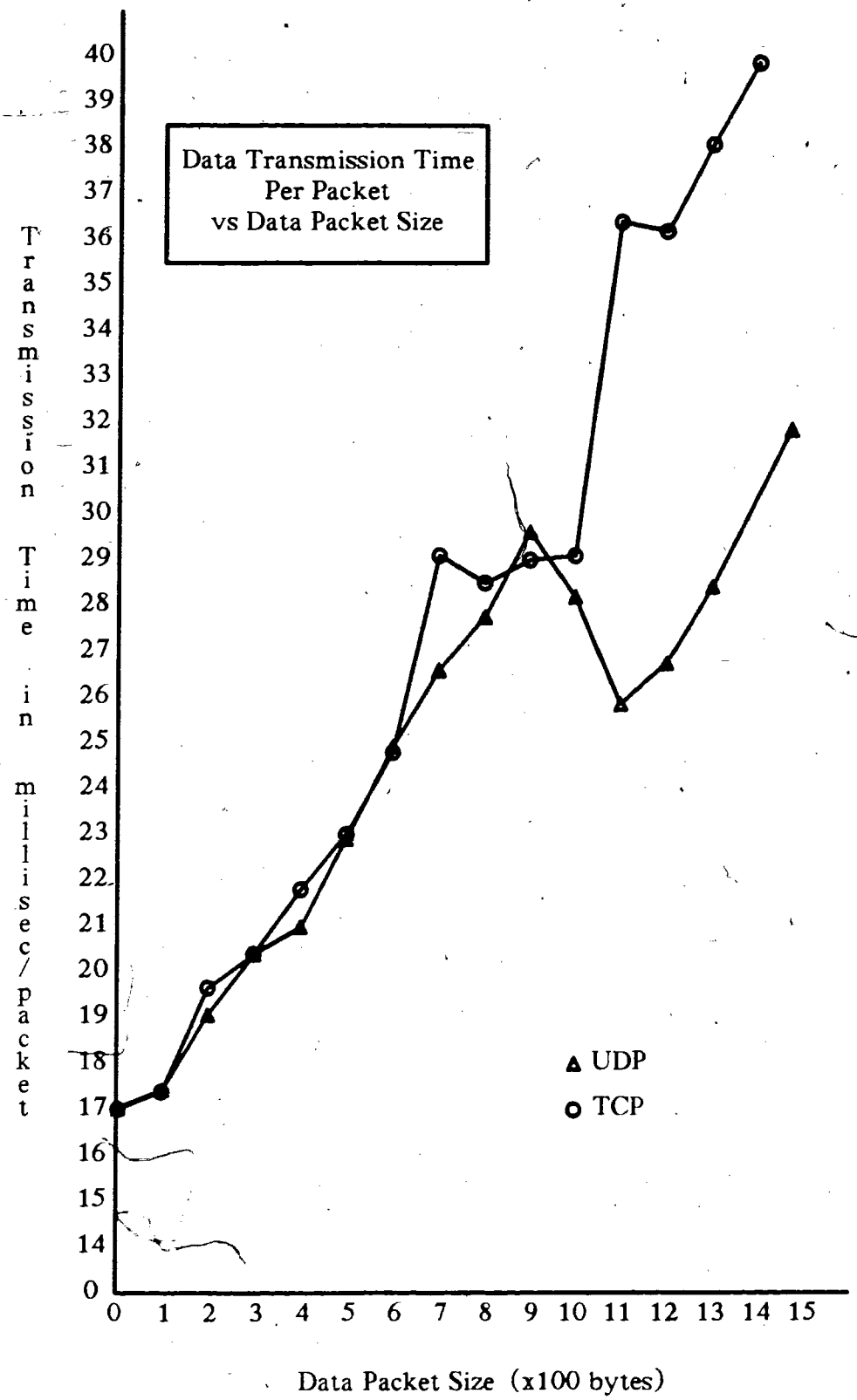


Figure 3-3: Transmission Time Per Packet vs Data Packet Size

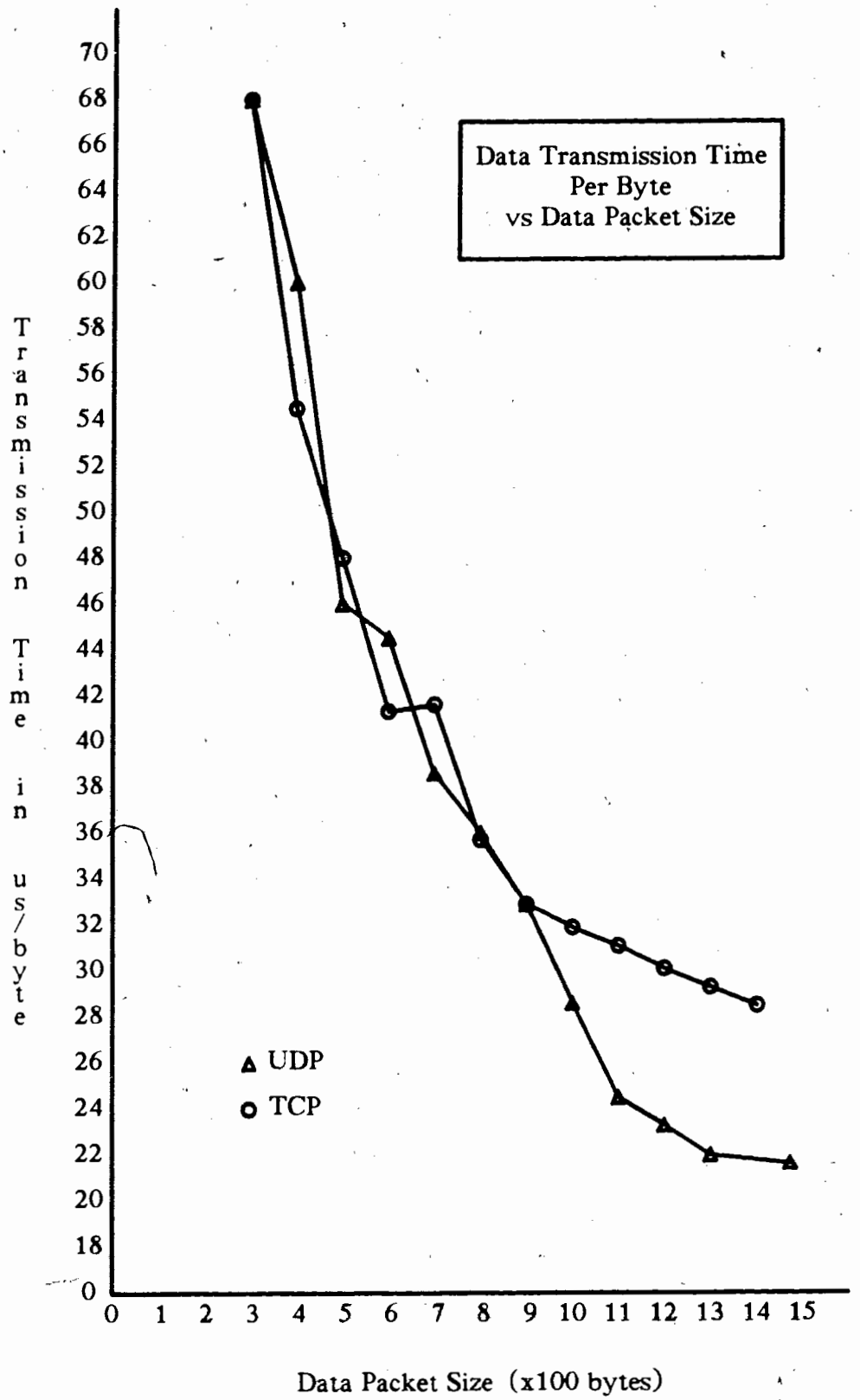


Figure 3-4: Transmission Time Per Byte vs Data Packet Size

3.5. Reliability of TCP

TCP is reliable in itself because the protocol has already incorporated a flow-control and window mechanism to guarantee delivery of packets. Thus, TCP without acknowledgement is sufficient and an extra acknowledgement scheme built on top of TCP is unnecessary.

3.5.1. Flow Control in TCP

Flow control is concerned with ensuring that the rate of transmission of packets from the source shall not exceed the capacity of the destination to receive packets. The flow control mechanism in TCP is based on a multi-packet acknowledgement scheme. For a single packet acknowledgement scheme, after the sender sends out a packet, it will wait for an acknowledgement from the receiver before sending out the next packet. If, after a certain time delay, the sender has not received the acknowledgement, it will retransmit the packet. This scheme can be highly inefficient since the network can be idle for a great part of the time while acknowledgements are awaited after each packet. To improve the efficiency of the line, several packets can be transmitted by the sender before waiting for an acknowledgement. In doing this, it will be necessary to provide some means of distinguishing individual packets. An acknowledgement also must now be able to specify which packets it is acknowledging. Packets can be distinguished simply by using a sequence number carried in the packet header. An acknowledgement can thus use these packet sequence numbers to specify which packets it is acknowledging.

The number of packets that may be transmitted before an acknowledgement is received is determined by several factors such as the line bandwidth and the availability of the receiver's buffers. This number, known as the "window width", is

agreed upon by all the nodes at initialization time. At any point, the transmitter can transmit packets that are within the width of the window continuously. Once this is done, it will wait for an acknowledgement for these packets. The window is moved forward as the acknowledgements are received, thus, allowing new packets to be transmitted. In this way, the scheme guarantees that all the packets will be received.

3.6. Unreliability of UDP

In section 2.3, we saw that the possibility of transmission error due to the network itself is very small. However, our experiment shows that lost packets are very common with UDP. The reason, therefore, for this unreliability is due to the fact that the UDP does not have any flow control as does the TCP. As a result, the protocol can only give its best effort to deliver the internet packets, but it cannot guarantee that they are delivered once and only once, nor that they will be delivered in the same order that they were transmitted. Packets may also be lost due to the transceiver's buffer overflowing. If data arrives at a controller which is unable to accept it into its buffers, that data can simply be thrown away with the complete assurance that it will be retransmitted eventually. However, with just the bare UDP, no retransmission will be called for.

Packets lost due to buffer overflow happens quite frequently when a sender is sending out packets at a very high rate without any interruption, but the receiver is picking up the packets at a much slower speed. Another cause for buffer overflow is when messages from various stations arrive simultaneously or within a very short period of time and again the receiver is unable to process them as fast. In order to understand this problem of buffers overflowing more fully, we need to look at the hardware aspect of the multibus Ethernet controller where the packets are being picked up and buffered from the network.

3.6.1. Multibus Ethernet Controller

Our experimental Ethernet uses the 3Com 3C400 Multibus Ethernet Controller as depicted in figure 3-5. The controller is a device that is responsible for the carrier sensing, collision detecting, and the buffering of encoded data for transmission and reception. Part of its internal memory is allocated to two buffers each of size 2K bytes for the storing of data.

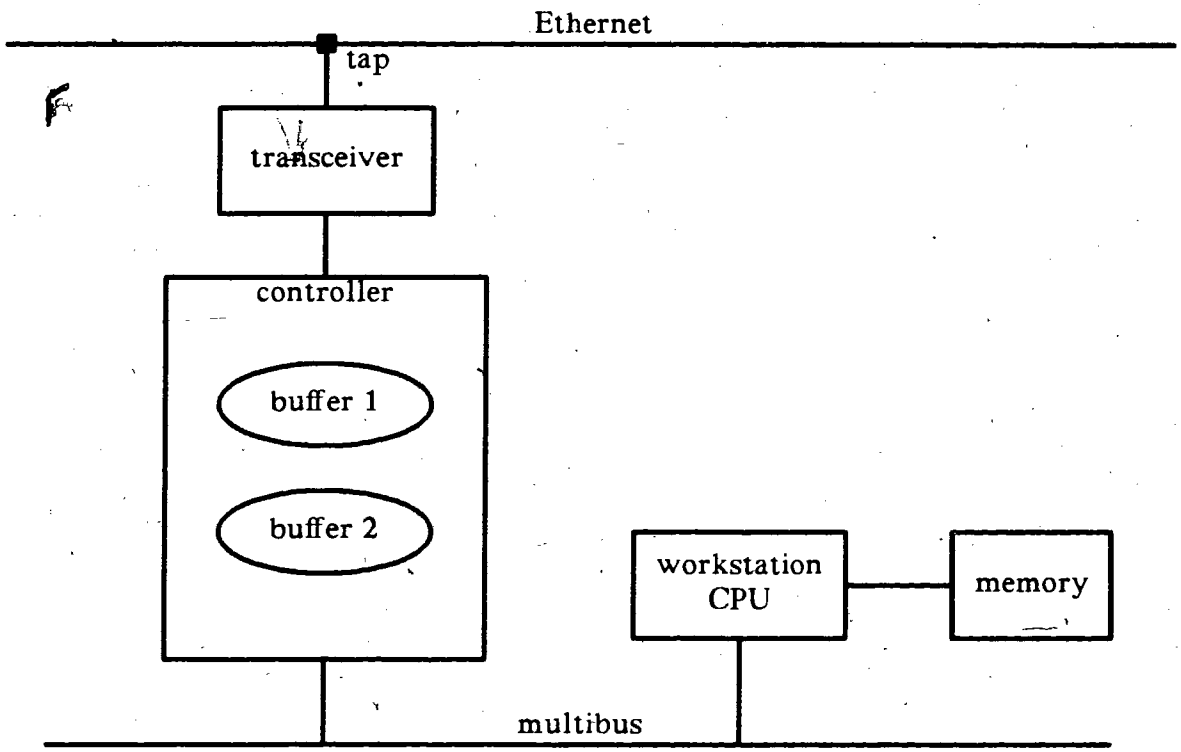


Figure 3-5: The 3Com Multibus Ethernet Controller

A packet of encoded data placed on the Ether will be picked up by the controller if it is addressed to it, and will temporarily place it in one of its two internal receive buffers. Following this, the controller will then try to interrupt the workstation. If the workstation is not manipulating the message queue in its own

memory when the interrupt occurs, it will service the interrupt by transferring the packet from the controller's buffer to the message queue in its own memory. However, while the workstation is manipulating the message queue, the interrupt is disabled for concurrency control reasons. While the interrupt is disabled by the workstation and more than two packets of information arrive at the controller, the first two will be stored in the two buffers, but the subsequent packets will be ignored. The problem is that the controller has only two buffers and each buffer can only hold one packet no matter how small the packet is.

This problem of buffer overflow happens frequently in two situations. The first situation is where a sender sends out a continuous stream of packets with no pauses to a receiver who is unable to empty the buffers at this fast rate. The second situation is where there are several receivers who need to acknowledge the arrival of a packet to the broadcaster at the same time. Since the packet was broadcast, all the receivers will receive the packet at the same time with negligible transmission delay, and if they all work at about the same speed, they will all acknowledge at about the same time. As a result, the third and subsequent acknowledgements arriving at the broadcaster will be lost.

3.6.2. Experiment

The experiment whose result is summarized in table 3-1 shows that the first situation does indeed happen. In the case of UDP without acknowledgement, packets are lost due to the sender sending out packets faster than the receiver can handle them. For the second situation, if there are only two buffers, then acknowledgement packets are lost if more than two receivers acknowledge a packet at the same time.

The experiment uses one broadcaster and four receivers. The broadcaster

broadcasts a fixed number of fixed size packets to the four receivers. Upon receiving a packet, the four receivers will send an acknowledgement back to the broadcaster. Only after receiving all of the acknowledgements (or a time-out occurs if some acknowledgements are lost) will the broadcaster broadcast the next packet. The experiment is set up in such a way that either the four receivers can acknowledge the receipt of a packet immediately or some of the receivers wait for one second before acknowledging. The result for this experiment (as summarized in table 3-2) shows that when more than two receivers reply at the same time, acknowledgements are indeed lost. When only two receivers acknowledge at the same time, no acknowledgements are lost. This is also the case when three receivers delay even though there are only two buffers. This is because when the first packet arrives, the workstation can immediately transfer this packet from the controller to its own memory. Thus there are still two empty buffers for storing the next two packets. This also explains why the number of acknowledgements lost for the one receiver delay case is less than the no receiver delay case. When a workstation is manipulating the message queue, which is a time consuming process, the interrupt from the controller is disabled. If both buffers already contain packets and another packet arrives in the mean time, it will be thrown away. This is just the case when no receiver or four receivers wait in our experiment.

When three receivers or four receivers delay for the same time, we would expect the results to be respectively similar to one receiver and no receiver delay, but this is not the case. The reason is that the experiment is done in a multi-tasking environment. Thus, when the acknowledgements are delayed for one second, the process could very well be swapped in and out several times. As a result, the acknowledgements will be sent at different times and thus arrive at the broadcaster at different times. This in effect, is almost like using randomized delays for the

acknowledgements. Thus, the percentages of lost acknowledgements for the three and four receivers delay cases are much smaller than the one and no receiver delay cases.

number of receivers that wait	percentage of acknowledgements lost
0	25%
1	13%
2	0%
3	0%
4	0.03%

Table 3-2: Percentage of lost acknowledgements

3.7. UDP Broadcast vs TCP

For error free data transmission, we can use either TCP without acknowledgement or UDP with acknowledgement. We have seen that the UDP scheme is much faster than the TCP scheme. Moreover, if we want to send the same message to all the nodes, we can do still better than either of the two schemes by using the broadcast method that is supported by the Ethernet. In order to send the same packet to n nodes using TCP, one will have to do this sequentially by sending the packet to one node at a time, whereas if we use broadcasting, we need only broadcast the packet once and all the nodes will receive the packet simultaneously. This means that to send the same data to n nodes, TCP will take n times longer than broadcasting.

There is, however, one draw-back in using broadcasting because it is only supported by UDP and not by TCP. The result shows that UDP by itself is unreliable because it does not have a built in flow control mechanism like TCP. Therefore, if we are to use the UDP broadcast for data transmission, we will have to build a higher level reliable protocol on top of UDP.

3.8. Conclusion

To send a packet to many nodes using the broadcast method is definitely faster than sending a packet using either TCP without acknowledgement or UDP with acknowledgement. However, the broadcast method is only supported by UDP, and UDP is unreliable. Thus, if we are to use the UDP broadcast for data transmission, we will have to build a higher level reliable protocol on top of UDP. This protocol must guarantee that all the processors involved must receive all the data correctly. This would imply that when a packet is lost, the receivers must be able to let the sender know that the packet is lost and request the sender for a retransmission.

Chapter 4

Reliable Broadcasting Protocols

4.1. Introduction

Since data transmission is a critical factor in a distributed database system, we need not only a reliable but also a fast broadcasting protocol. That is, one which will spend the least amount of time in sending data, and yet guarantees that all the data will be received by all of the participating nodes. We have noted that the UDP broadcast is unreliable and therefore needs to have a higher level reliable broadcast protocol built on top of it. In this research, several broadcasting protocols built on top of the Internet User Datagram Protocol have been tested. They differ in the way in which the acknowledgements are made and in the method used for the retransmission of lost packets. The protocols are: acknowledgement per packet, acknowledgement per file with file retransmission, acknowledgement per file with selective packet retransmission, and no acknowledgements or retransmissions. Note that the last scheme is not really a protocol, but only for the purpose of finding the lower bound for data transmission and the overhead required for acknowledgements. Several variations of the above protocols have also been tested. We will now describe these protocols in detail. A pseudo code listing of these protocols can be found in appendix A.

4.2. Acknowledgement Per Packet With Packet Retransmission

For this protocol, a file is broadcast from a sender to the receivers in data packets consecutively numbered from one. A data packet, as depicted in figure 4-1, has a sequence number and a data field.

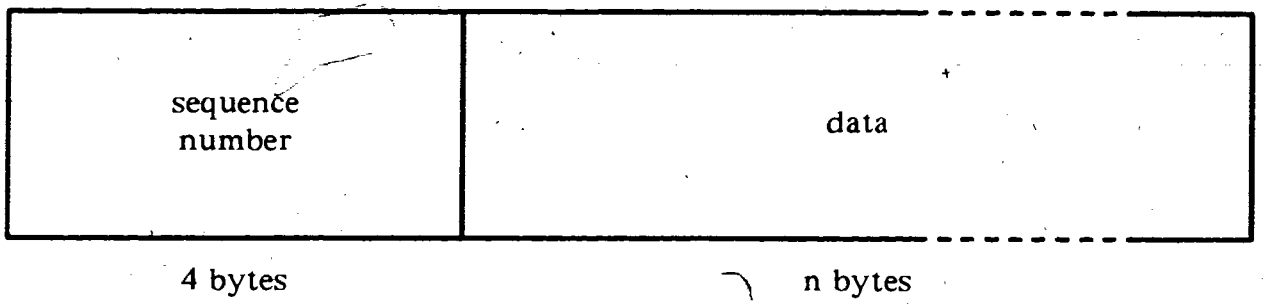


Figure 4-1: Data Packet

The sequence numbers begin with one and increase by one for each new data packet, thus allowing the receivers to discriminate between old and new packets. The data field has a length of from one to 1456 bytes making the maximum packet length to be 1472 bytes.[1] All the packets, except possibly for the last one, have the maximum length size. If the packet is 1472 bytes long, the packet is not the last data packet; if it is from one to 1471 bytes long, it acts as the end packet and signals the end of the file transfer.

After broadcasting a packet, the broadcaster has to wait for an acknowledgement from each of the receivers. An acknowledgement packet is shown in figure 4-2. The sequence number is the number of the packet that it is acknowledging and the message is always OK. The broadcaster will miss an acknowledgement if either the acknowledgement was lost or the data

[1] See section 3.4 for a discussion of why we have selected to use a maximum packet size of 1472 bytes.

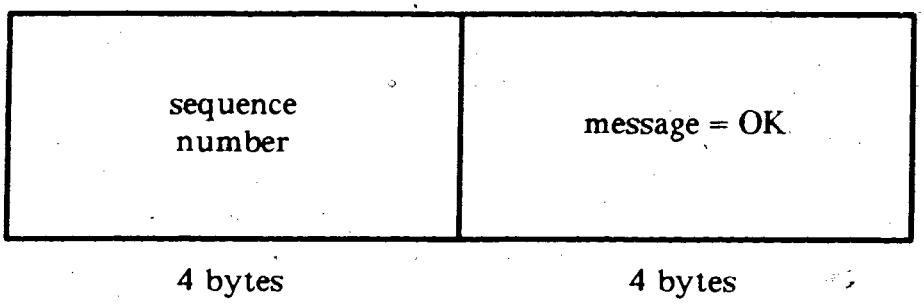


Figure 4-2: Acknowledgement Packet

packet never arrived at the receiver and so no acknowledgement was made. In either case, the broadcaster will timeout and will rebroadcast that data packet. The broadcaster will not go on to send the next packet until the acknowledgements have been received from all the receivers. All packets received by the receivers will be acknowledged with the message OK, but only the ones with the same sequence number as that of its own internal sequence number count will be taken. The others with the previous sequence number will be ignored. This acknowledgement scheme guarantees that at each broadcast, all previous packets will have been received by all the receivers.

The end packet, as mentioned before, contains less than 1472 bytes of data.

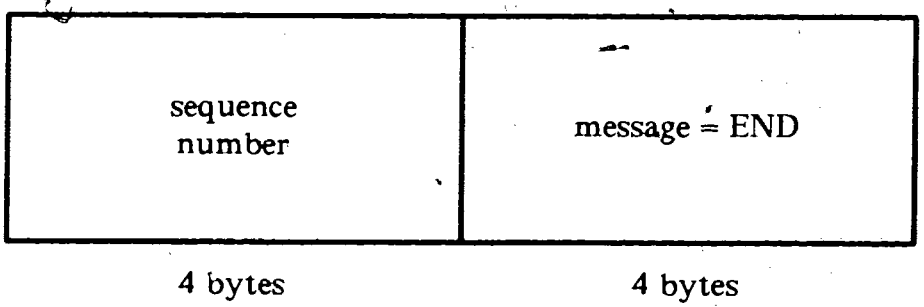


Figure 4-3: End Packet

If the file happens to divide evenly into the packets, thus filling up the last packet, then an extra end packet will be broadcast. This end packet has the same format as

the acknowledgement packet (see figure 4-3) except that it has the next data packet sequence number and the message is END. After receiving the end packet, the receivers will acknowledge it as before. They will then wait for an end-reply packet from the broadcaster, upon which they will terminate. Upon receiving all the acknowledgements for the end packet, the broadcaster will broadcast out an end-reply packet and then is free to go off with the assurance that the file has been transferred successfully. However, if an acknowledgement for the end packet is lost, the broadcaster will rebroadcast the end packet just like the data packets.

The purpose of the end-reply packet is to make it practically certain that the sender and receivers of a file will agree on whether the file has been transmitted correctly. If the broadcaster lost an acknowledgement for the end packet, it will rebroadcast the end packet. With the above end sequence, the receivers will still be around to acknowledge an end packet if the original one was lost. Thus the broadcaster and receivers can all terminate together after being assured of a successful file transfer.

4.3. Acknowledgement Per File With File Retransmission

The second broadcast protocol is very much like the first one. The data packet, like the first one, has a length of 1472 bytes with a sequence number field and a data field. The end packet is a data packet that is shorter than 1472 bytes or one with an END message. The data packets are broadcast out consecutively numbered from one. The difference is that instead of the receivers acknowledging every packet that is broadcast, they will wait until receiving the end packet. After receiving the end packet, each receiver will send an acknowledgement packet back to the broadcaster. The acknowledgement packet is similar to the previous one with a

sequence number followed by a four byte message. The message, however, can be either OK (if all the packets were received) or RETRANSMISSION (if some packets were lost and retransmission is required). If one of the acknowledgement message is RETRANSMISSION or if an acknowledgement packet is lost, the broadcaster will retransmit the whole file starting from packet one; in effect, restarting the whole process. However, if all the acknowledgement messages are OK, then it will broadcast the end-reply packet and then terminate. The receivers, after sending out the acknowledgement for the end packet will either wait for the retransmission of the file if it has lost some packets, or wait for the end-reply packet. If it is waiting for the end-reply packet but receives data packets, they will be ignored.

4.4. Acknowledgement Per File With Selective Packet Retransmission

As the name suggests, the procedure for this third protocol is much the same as the second protocol. The detailed operations for dealing with the packets and the acknowledgements, however, are quite different. Instead of retransmitting the whole file when some packets are lost, only those packets that are lost are retransmitted. Thus the acknowledgements will need to specify which data packets have been lost and then the broadcaster needs to be able to selectively pick out a portion of the file corresponding to those lost data packets for retransmission. This scheme is very much like the flow control and window mechanism used by the TCP protocol with the window width being the size of the file.

The broadcaster begins with broadcasting the data packets consecutively numbered from zero to the receivers. The data packet uses the same format as described previously with a total length of 1472 bytes. The last data packet, even

with a length of less than 1472 bytes is not considered as the end packet. An extra end packet using the next sequence number with the message END is always broadcast to signal the end of a file transfer. The reason is that this extra end packet must be there to signal the end when several packets all having the maximum length size are being rebroadcast. This situation is similar to the case where the file divides evenly into the packets and so all the packets have the same maximum length.

The acknowledgement packet has a variable size as shown in figure 4-4.

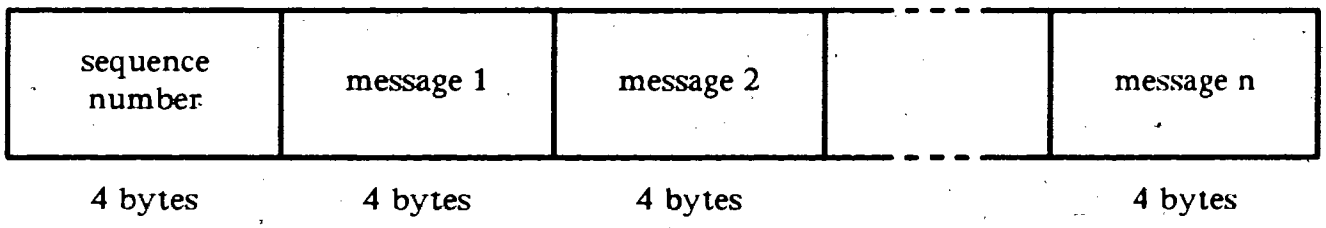


Figure 4-4: Variable Format Acknowledgement Packet

There are as many messages in the acknowledgement packet as there are data packets: the first message corresponds to the first data packet, the second message to the second data packet, etc.. Each of the messages can be either OK if the corresponding data packet was received or RETRANSMISSION if that packet was lost. With this acknowledgement scheme, there is a limit to the number of packets that can be sent for each file transfer. This limit of 364 data packets or a file size of about 530K bytes is good enough for our purpose. Notice that we can use a message size of only one bit, thus increasing our limit by 32 times.

Upon receiving a packet, the receiver will check whether or not it has already received this packet. If it has already received this packet, the packet will be ignored, otherwise, it will store the packet and notes down that this packet has been

received. After receiving the end packet, the receiver will construct the acknowledgement packet according to whether a data packet has or has not been received. After sending the acknowledgement packet back to the broadcaster, the receiver will either wait for the end-reply packet if it has received all the data packets or it will wait for the retransmission of lost packets. If it is waiting for the end-reply packet, it will ignore all other packets that it receives. It will continue to wait until it receives the end-reply packet upon which it will terminate.

The broadcaster, upon receiving all the acknowledgements will decide on which packets need to be retransmitted. This means that all the packets will need to be buffered. All lost data packets from any receiver will be rebroadcast sequentially with no interruption. The last data packet rebroadcast will be followed by the end packet. This rebroadcasting of lost packets will continue until all the messages in all the acknowledgement packets are OK, after which the broadcaster will broadcast out the end-reply packet and will then terminate.

4.5. No Acknowledgements Or Retransmissions

This last protocol with no acknowledgements or retransmissions is really not a protocol at all. The broadcaster simply assumes that all data packets will be received by all the receivers. Thus, all the data packets will be broadcast out consecutively numbered from one with no interruption until the end packet. The end packet, as in the first protocol, is either a data packet with less than 1472 bytes long or one with the END message. After the end packet, the broadcaster will just terminate without any further work. All that the receivers can do is just hope for the best. If there are lost data packets, then the result will be erroneous.

The purpose of this is to find out what is the overhead for the

acknowledgements and retransmissions required by the previous three protocols. When there are lost packets or incomplete file transfer (which happens quite often), no data is gathered. Nevertheless, when all the file transfers run to completion, we will have the lower bound for the transmission time.

4.6. Variant Of The Third Protocol

Of the four protocols that we have discussed so far, the acknowledgement per file with selective packet retransmission protocol described in section 4.4 is probably the best one. However, we might still be able to improve on it by putting a time delay between the broadcasting of the data packets. This follows from the fact discussed in section 3.6 that when packets are arriving too fast at a transceiver, they will be ignored. Thus, if we slow down the broadcasting process, fewer data packets will be lost by the receivers, and so fewer retransmissions will be required.

If we also make the receivers wait before making the acknowledgements, then the broadcaster will not lose so many acknowledgements and therefore will not have to wait for a timeout. This will not work if we use a constant time delay t for all the receivers. We need to have the first two receivers reply with no delay, the second two with a delay of t, the third two with a delay of $2t$, etc..

4.7. The Initial Connection

The initial connection routine is common to all of the protocols. It basically checks whether all the nodes specified in the query are responding or not. If all the nodes respond, then the processing can continue, otherwise, the intersection cannot be completed.

The node where the query is issued initiates this connection by broadcasting out

the query to all the nodes. Upon receiving the query, all the nodes will check whether its data set is being requested for the intersection. If it is, then the node will respond by sending its data set size back to the requesting node, otherwise, nothing is done. When the requesting node has received all the data set sizes from the participating nodes, then the processing of the query will start. If one of the requested data nodes did not respond, it is assumed that that node is down and its data set is not available, therefore, the query cannot be processed.

This process of gathering the data set sizes is referred to as the "handshake" in later chapters. The initial connection, therefore, involves the setting up of the sockets and their addresses, followed by the handshake.

4.8. Empirical Results

The transmission times for the different protocols are summarized in table 4-1. The lower bound for data transmission is 0.25 milliseconds per byte. The two protocols with delay, which are variations of the acknowledgement per file with selective retransmission protocol were expected to perform better. However, they performed the worst among all the protocols tested. The reason is that the delay used is one second which is too long for any practical purposes. Most of the time is spent on waiting. Thus, if we are to use delay, it will have to be less than one second. The acknowledgement per packet scheme has a transmission overhead of 0.19 milliseconds. This scheme requires too many acknowledgements, which are time consuming.

The 0.39 millisecond transmission time for the acknowledgement per file with whole file retransmission protocol is a little misleading. This protocol runs indefinitely for a very long time for many file transfers and this is not reflected in

Method	Transmission Time (ms/byte)	Acknowledgement Overhead (ms)
No acknowledgements	0.25	0.00
Acknowledgement per file with selective retransmission	0.36	0.11
Acknowledgement per file with whole file retransmission	0.39	0.14
Acknowledgement per packet	0.44	0.19
Acknowledgement per file with delay in sending acknowledgement	0.72	0.47
Acknowledgement per file with delay in broadcast	0.95	0.70

Table 4-1: Transmission Time For Different Protocols

the summary. The reason why this happens is that the whole file is retransmitted when there is a lost packet. This in effect, is the same as restarting the whole process. If all the processes operate at the same speed, then the same result will happen again and the same packets will be lost. Thus, repeating cycles are formed, and the process goes on indefinitely.

The best of all the protocols tested is the acknowledgement per file with selective retransmission scheme. This protocol requires a transmission time of 0.36 milliseconds per byte and has an overhead of only 0.11 milliseconds. The comparison experiments for the three set intersection algorithms use this protocol.

4.9. Criteria for a Fast Reliable Broadcasting Protocol

From our experiments, we have found several criteria for a fast reliable broadcasting protocol. In order to have a reliable protocol, some kind of acknowledgement is necessary. However, we must minimize the number of acknowledgements required. Our protocol which uses only one acknowledgement after each file transfer is the fastest.

Another criterion is the packet size used. We have found that the larger the packet size, the faster the rate of data transmission.

Both the delay in broadcasting a packet and the delay in sending acknowledgements must be less than one second. In using a one second delay, most of the transmission time is spent in useless waiting.

Chapter 5

Set Intersection Algorithms

5.1. Introduction

A straightforward solution to obtain an answer for a given set expression is to require all the data sets specified in the expression to be transmitted to the site where the answer is to be presented and then compute the answer there. The problem with this solution is that it ignores the distributed nature of the problem and the properties of the broadcast network. It will therefore result in a tremendous amount of data that needs to be transmitted. Since in a broadcast network, every set transmitted over the network is available to all nodes, each node can make use of this information to process its local set and hopefully to eliminate some elements from the set which have been rendered redundant. This will result in a smaller set of data that needs to be transmitted in the future. In addition, parallel processing in each node may reduce the response time in deriving the answer.

In this chapter, we will describe three different algorithms to perform set intersections on a distributed database: Simple, Static, and Dynamic. The Simple algorithm does not take into consideration the properties of the broadcasting network, while the Static and the Dynamic algorithms are based on the theoretical results described in [Luk84]. We will briefly summarize the theory developed in [Luk84] and then describe the three algorithms in detail.

5.2. Theory

5.2.1. Definitions And Notations

The processors or data nodes in the broadcast network are denoted by $N_1, N_2, \ldots, N_n$. Stored in each data node N_i, $1 \leq i \leq n$, is a set of data objects denoted by S_i, $1 \leq i \leq n$. Any one of the data nodes can be a request node R and can submit a request Q. A request Q is a set expression of intersections of the data objects S_i, and has the form

$$S_{q(1)} \cap S_{q(2)} \cap \ldots \cap S_{q(m)} \quad \text{for } m \leq n$$

where each S_q is one of the sets which are all distinct from each other. Once Q has been submitted, it is interpreted within R, which acts as the scheduler, and a schedule to perform all necessary operations is derived and broadcast to all the data nodes.

5.2.2. The Schedule

The schedule is a sequence of operations which are arranged in the order of execution. At every time step in a schedule, a node is designated to be the transmitter of its data set and the other nodes will be the receivers. The local operations to be performed in each node after reception of the transmitted set are also specified.

After the schedule has been formulated by the scheduler R, it is broadcast to all the nodes. All the participating nodes will then follow this schedule in their processing. After the completion of all the operations specified in the schedule, the result T, which is a set of objects satisfying the request Q, will be available to the result node R.

5.2.3. Node Operations

There are two kinds of operations specified in a schedule that the processor in each data node performs: data transmission and local operation. For data transmission, if node N_i is the transmitter, it simply broadcasts its own data set S_i to all the other nodes. In the schedule, it is denoted by

$$N_i: S_i \rightarrow \text{network}$$

A local operation is an intersection of the local set with the received set. When S_j is received, the node N_i will perform the local operation denoted by

$$N_i: S_i \leftarrow S_i \cap S_j$$

This means that the local set S_i is replaced by the intersection of the two sets S_i and S_j. Operationally, the part of the set S_i which cannot be found in the received set S_j will be removed from S_i.

5.2.4. Optimization

As in many optimization problems, the kind of optimization achievable depends very much on the amount of information that is available to the scheduler. In order to reach an optimal solution, the scheduler must have complete knowledge of all the sets stored in all the nodes. But in an environment where the data is constantly changing, it would be very costly for the scheduler to have a complete knowledge of all the data sets. This is especially so when the scheduler is not fixed at one particular node. Any node in the network can be a request node and therefore is the scheduler. This would imply that all the nodes in the network must have a complete knowledge of all the sets stored in all the nodes. Even if this information is available, and the scheduler has determined the optimal schedule, once the nodes start to process the schedule, the data would be changed, and an optimal schedule may no longer be optimal any more.

Since it seems highly unrealistic to expect the scheduler to have complete knowledge of all the data sets, we will assume that only the cardinalities of all the sets are known at the start to the scheduler. This is not an uncommon assumption in a database environment. Due to this limitation, it is therefore only possible for the scheduler to derive an optimal solution in the average case, i.e., the solution with the least expected cost. The amount of data transmission is calculated as the expected value of what will be eventually sent. For example, if $S_1 \cap S_2$ is to be transmitted, the amount of data transmission is estimated to be

$$\frac{cardinality(S_1) \times cardinality(S_2)}{cardinality\ of\ the\ universal\ set}$$

For a more detail treatment of this, the reader is referred to [Luk84].

The cost of a schedule is the total cost of all data transmission. The cost of the local operation is omitted because it is assumed to be negligible. (This is an assumption not to be adopted in our experiment.) From the viewpoint of transmission, an object is one unit of data and transmission of an object incurs one unit of transmission cost. Thus the cost of a schedule is the total number of objects that are sent out on the network.

An optimal schedule is defined to be one by which the answer to the request Q will be derived with minimum cost.

We will now derive an optimal schedule for Q where Q is an intersection of n sets, i.e., $S_1 \cap S_2 \cap \ldots \cap S_n$ Let us re-label, for the time being, the sets according to their cardinalities in increasing order, so that S_1 is the set with the least cardinality and

$$|S_1| \leqslant |S_2| \leqslant \ldots \leqslant |S_n|$$

This reordering of the sets is denoted in the schedule by

$$Q \leftarrow \text{reorder } Q$$

The optimal schedule is as follows:

The Optimal Schedule

$$\begin{array}{lll}
(0) & R: & Q \leftarrow \textit{reorder } Q \\
 & & Q \rightarrow \textit{network} \\
 & & \\
(1) & N_1: & S_1 \rightarrow \textit{network} \\
 & N_2: & S_2 \leftarrow S_1 \cap S_2 \ , \ \ldots \ , \ N_n: \ S_n \leftarrow S_1 \cap S_n \\
 & & \\
(2) & N_2: & S_2 \rightarrow \textit{network} \\
 & N_3: & S_3 \leftarrow S_2 \cap S_3 \ , \ \ldots \ , \ N_n: \ S_n \leftarrow S_2 \cap S_n \\
 & & \\
\ldots & & \\
 & & \\
(n) & N_n: & S_n \rightarrow \textit{network} \\
 & R: & T \leftarrow S_n
\end{array}$$

T is thus the answer to Q and is available not only to R, but also to all the other nodes. The proof of optimality for the above schedule is given in [Luk84].

5.3. The Simple Algorithm

This first algorithm, referred to as the Simple Algorithm, does not make use of the parallel processing that is possible with a broadcasting network. Neither do the nodes perform any preprocessing to its database. When given the query Q, the request node R will initiate a connection with all the N_i's as specified in Q by broadcasting Q. This procedure is referred to as the initial connection and is described in detail in section 4.7. The request node R does not rearrange the order of Q at all. R simply uses the order of Q as the schedule for operation.

After the initial connection has been established with all the participating nodes, the request node R will wait for all the data nodes to send their entire data set sequentially following the order of the schedule. Upon receiving S_j , R will replace its own data set S_i with the intersection of S_i and S_j. This is the local operation denoted by

$$S_i \leftarrow S_i \cap S_j$$

The data nodes, upon receiving Q, will follow the order as specified by Q in which to send the data. The data is broadcast by each node sequentially to the rest of the nodes. However, all the nodes except for R, simply ignore the data being received and no local operations are done whatsoever.

5.3.1. The Schedule

Given the query

$$Q = S_1 * S_2 * \ \ldots * S_n$$

the nodes will follow schedule 1 below

Schedule 1

(0) $R : Q \rightarrow network$

(1) $N_1 : S_1 \rightarrow network$
$R : T \leftarrow S_R \cap S_1$ *if* S_R *is in* Q
$T \leftarrow S_1$ *otherwise*

(2) $N_2 : S_2 \rightarrow network$
$R : T \leftarrow T \cap S_2$

...

(n) $N_n : S_n \rightarrow network$
$R : T \leftarrow T \cap S_n$

Note that all the nodes do not make use of the information that is received from the broadcast. They can certainly make use of it by doing some preprocessing and hopefully, they can reduce the size of their data set by eliminating the redundant information. This is just the approach taken by our next algorithm.

5.4. The Static Algorithm

In this algorithm, we want to make use of the information that is broadcast to each of the nodes. Since the data is broadcast, all the nodes will receive the data at the same time. This means that the processing of the data can be done in parallel by all the nodes.

The algorithm starts, when given a query, by performing the initial connection steps. This involves the broadcasting of the query to the other nodes. All the participating nodes will reply by sending the cardinality of their database to the request node. Upon receiving all the cardinalities, the request node will reorder the query sequence according to the cardinalities in the increasing order. This reordered query becomes the schedule of operations and is broadcast to all the nodes. This schedule remains fixed for the duration of the whole process and the nodes will use this schedule to determine which of the two operations, local or data transmission, to perform. This algorithm is just the theoretically optimal algorithm as described in section 5.2.4.

5.4.1. The Schedule

Given the query

$$Q = S_1 * S_2 * \ \ldots * S_n$$

R will reorder Q according to the cardinalities in increasing order. Let us, for the time being, re-label the sets according to their cardinalities so that

$$|S_1| \leqslant |S_2| \leqslant \ldots \leqslant |S_n|$$

This reordering of the sets is denoted in the schedule by

$$R : Q \leftarrow \text{reorder } Q$$

The static schedule is as follows.

Schedule 2

$$
\begin{array}{lll}
(0) & R : Q & \leftarrow \textit{reorder } Q \\
 & Q & \rightarrow \textit{network} \\
 & & \\
(1) & N_1 : S_1 & \rightarrow \textit{network} \\
 & N_2 : S_2 & \leftarrow S_1 \cap S_2 \ , \ \ldots , \ N_n : S_n \leftarrow S_1 \cap S_n \\
 & & \\
(2) & N_2 : S_2 & \rightarrow \textit{network} \\
 & N_3 : S_3 & \leftarrow S_2 \cap S_3 \ , \ \ldots , \ N_n : S_n \leftarrow S_2 \cap S_n \\
 & & \\
\ldots & & \\
 & & \\
(n) & N_n : S_n & \rightarrow \textit{network} \\
 & R : T & \leftarrow S_n
\end{array}
$$

Operationally, we select the set with the smallest cardinality first and broadcast the set to the other nodes. Each node then calculates the intersection of the received set with its own set. The resulting set size will therefore be either the same or smaller than the original set size. The next smaller set in the original schedule is then broadcast out. Each node again calculates the intersection of the received set with its own set. This sequence is repeated until all the nodes containing sets in the expression Q have broadcast their own set once. Whatever the last node broadcast will be the answer to Q.

5.5. The Dynamic Algorithm

The Dynamic algorithm is very much similar to the Static algorithm. The only difference is that after each set transfer, the request node will wait for the other nodes to send their new data set sizes to it. Only those nodes that have not sent their data set need to reply, the rest of them that have already sent their data set need not do so. Having received all the new data set sizes, the request node will recalculate a new order for the schedule. Just like the initial construction of the schedule, this reordering is done according to the cardinalities of the data sets in the increasing order. Once this is done, the new schedule is again broadcast to all the nodes, and the nodes will follow this new schedule. The last set broadcast will be the answer to Q.

This process of dynamically reconstructing the schedule after each node has transmitted its file should reduce the amount of data transfer even more than the Static algorithm. However, there is a price to be paid for this. First, more operational time will be needed to recalculate a new order after each file transfer, and secondly, more data transfers are necessary from the data nodes so that the request node can have the new data set sizes for doing the rescheduling. This trade-off is confirmed by our experimental results.

5.5.1. The Schedule

The dynamic schedule is as follows.

Schedule 3

$$
\begin{array}{lll}
(0) & R: Q \leftarrow \textit{reorder } Q & \\
 & \quad Q \rightarrow \textit{network} & \\
(1) & N_1: S_1 \rightarrow \textit{network} & \\
 & N_2: S_2 \leftarrow S_1 \cap S_2,\ |S_2| \rightarrow \textit{network} & \\
 & \ldots & \\
 & N_n: S_n \leftarrow S_1 \cap S_n,\ |S_n| \rightarrow \textit{network} & \\
 & R: Q \leftarrow \textit{reorder } (Q - N_1) & \\
 & \quad Q \rightarrow \textit{network} & \\
(2) & N_2: S_2 \rightarrow \textit{network} & \\
 & N_3: S_3 \leftarrow S_2 \cap S_3,\ |S_3| \rightarrow \textit{network} & \\
 & \ldots & \\
 & N_n: S_n \leftarrow S_2 \cap S_n,\ |S_n| \rightarrow \textit{network} & \\
 & R: Q \leftarrow \textit{reorder } (Q - N_2) & \\
 & \quad Q \rightarrow \textit{network} & \\
\ldots & & \\
(n) & N_n: S_n \rightarrow \textit{network} & \\
 & R: T \leftarrow S_n &
\end{array}
$$

Chapter 6

Results And Analysis

6.1. Introduction

In this chapter, we will first describe the setup for our experiments and how the data is collected. The rest of the chapter will be devoted to the presentation and analysis of the results for the three set intersection algorithms described in chapter 5.

6.2. Experimental Setup

6.2.1. The Hardware

All the experiments performed in this research were carried out on the local SFU Ethernet network which operates at a speed of 10 megabits per second. The current hardware configuration consists of five Sun Workstations, all connected together by the Ethernet and all running under the UNIX 4.2BSD operating system.

The Sun Workstations are powerful general-purpose microcomputers providing state of the art computing environment. Operating in a distributed network, each workstation supplies its user with a dedicated 32-bit architecture CPU and memory, while using high-speed local area communications to share other network resources and services. The computational power of the workstations begin with the most advanced microprocessor available today: the MC68010, a recent improvement to the Motorola 68000. The MC68010 operates with a clock speed of 10MHz and is able to address

up to 4 megabytes of physical memory with no wait states. It can also support up to 16 megabytes of virtual memory per process.

All the Sun Workstations run the most advanced version of the UNIX operating system, as enhanced at the University of California at Berkeley. The UNIX 4.2BSD version supports remote interprocess communication and remote execution of a process. Thus, a local area network communication facility is a standard capability of every workstation.

Of the five Sun Workstations that we have, one is the file server for the others. Hence, the work load on this particular machine is heavier than the other four. During most of our experiments, most of the machines have no more than two processes running except for the file server. This fact is reflected in our results by the higher operation time required by the file server as compared to the other four machines.

The five Sun Workstations, together with the multiple-access broadcasting Ethernet network, provide all the hardware equipment necessary for our experiments. The machines, connected together, can thus serve as data nodes in our model, with the database distributed among them.

6.2.2. The Database

In our experiment we have treated the data sets as groups of identifiers. These identifiers are simply integers. We have, thus, made the assumption that the record size is equal to the key size. We will see in later sections that this assumption will affect the ratio of the transmission time to the handshake time for the different algorithms.

Our database currently consists of 15000 integers distributed evenly among the five nodes. Within one data set, no two integers are the same. These integers have been randomly generated and are uniformly distributed over a given range. This assumption of a uniform distribution in the data sets is very common and is used in many modeling situations. This is especially the case when no prior knowledge of the database is known. However, we must realize that the results obtained based on this assumption will have biases and therefore will not apply, in general, to other modeling situations. We will discuss the implication of this assumption in this work later on. For a more general discussion, the reader is referred to the Ph.D. thesis of Christodoulakis [Chris81].

By using a different range, a different selectivity of the data set can be obtained.

Definition 1: The selectivity of a data set is defined as

$$\frac{\textit{the cardinality of the data set}}{\textit{the range of numbers used}}$$

For example, if we generate 3000 numbers over a range of numbers from 1 to 12000, then the selectivity of this particular data set is

$$\frac{3000}{12000} = 0.25$$

Thus, the larger the selectivity, the more chances there are of having duplicates in the different data sets, and thus, the larger will be the resulting intersection of the sets. This method of data generation is consistent with the assumption [Luk84] used to estimate the size of overlap of two sets.

We have also made the assumption that initially, there is a uniform selectivity and data set size throughout all the data sets. In the experiment, four different selectivities are used: 0.10, 0.25, 0.50, and 0.75 . All of these use a data set size of 3000 numbers. These numbers are generated by the different machines at initialization time and remain in central memory for the duration of the whole process. Thus, the time measurements in the results do not include any data transfer time to and from secondary storage.

6.2.3. Data Collection

The data that we have collected from our experiments falls into two categories. These two categories represent the two ways in which we can look at the cost of the algorithms. The first category contains data that deals with the amount of data transmitted. The second category deals with the time required for the data transmissions and the local operations. All the data is collected for the three algorithms using the four different selectivities mentioned in section 6.2.2. Using the five Sun Workstations as the data nodes, each containing 3000 integers, the query given is always the intersection of the five data sets

$$S_1 * S_2 * S_3 * S_4 * S_5$$

Since our data sets contain integers, we can simply count the number of integers that are transmitted for the amount of data transmission. This will give us an exact cost for the data transmissions of the algorithms.

The time measurements collected are the elapsed times for the data transmissions, the local operations and the handshakes. These times are obtained by starting a timer at the start of the operation and stopping it when the operation finishes. Due to the constraints of our hardware timer, the times are rounded off to

the nearest 10 milliseconds. Since we are working in a multi-processing environment, the times do fluctuate depending on the usage of the system. Thus, the data transmission time is the average elapsed time required to transmit a data set obtained from several runs. The local operation time is the average elapsed time required by a node to process the received data, i.e. to perform the operation

$$S_i \leftarrow S_i \cap S_j$$

where S_i is the local data set and S_j is the received set.

The handshake time is the average time required for the scheduler to obtain the data set sizes from the participating data nodes and then to reconstruct the schedule. The process of obtaining the data set sizes involves two steps. First, the scheduler will broadcast out a message specifying which node needs to respond. Upon receiving this message, the specified node will reply with its own data set size. For the Static and the Dynamic algorithms, there is a handshake during the initial connection. This initial handshake time is the same for both of them. For the Dynamic algorithm, there is also a handshake after every file transfer. After obtaining the data set sizes, the scheduler will reconstruct the schedule. The Simple algorithm does not require any handshake time or schedule reconstructing time.

The relationship of these three different times are diagramed in appendix B. It also shows the synchronization of the five data nodes for the three algorithms.

6.3. Analysis

In this section, we will present and analyze our main results from the research.

6.3.1. Amount of Data Transmitted

The amounts of data involved in the data transmissions for the three algorithms are tabulated in table 6-1. As we can see from these results, the amounts of data that need to be transmitted for the Static and the Dynamic algorithms are greatly reduced over the Simple algorithm. This is especially true when the selectivity is low. The reason is that there are few duplicates in the integers between the sets when a low selectivity is used. In fact, when the selectivity is 0.10, the cardinality of the resulting set T is zero. Thus, more than 90 percent of the integers are eliminated after the first node has transmitted its data for both the Static and the Dynamic algorithms. When the selectivity is 0.75, the resulting set size is 950, and so less data can be eliminated, which means more data needs to be transmitted. Even then, there is a saving of more than 23 percent.

The results for the Static and the Dynamic algorithms are not too much different from each other. This observation depends very much on the way the data is generated. With a uniformly distributed database, there is very little variation in the size of the intersection of any two data sets.[2] In other words, the probability of two different intersections having approximately the same size is very high. On the average, over 90 percent of the times, the intersection set size is within the range of $\pm$ 30 elements from its mean for the various selectivities.[3] As a result of all the

---

[2]See appendix C for an analysis of the size of the intersection of two uniformly distributed data sets.

[3]See table C-1 in appendix C.

sets having almost the same size, it will not make much of a difference whether the smallest set is selected for transmission (as in the Dynamic algorithm) or just a random set is selected for transmission (as in the Static algorithm).

If we remove the constraint of a uniformly distributed database, the Dynamic algorithm might perform drastically better in terms of data transmission depending on the data distribution. As an example, it is possible to get three overlapping sets of data as depicted in figure 6-1.

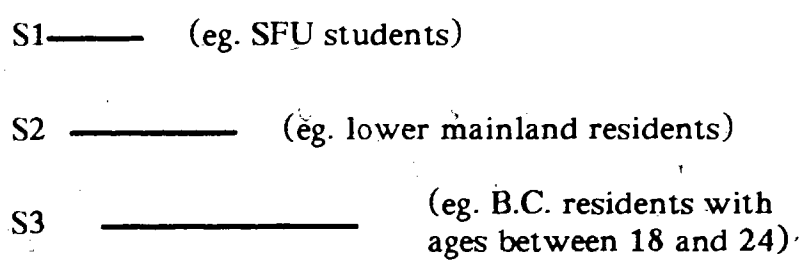
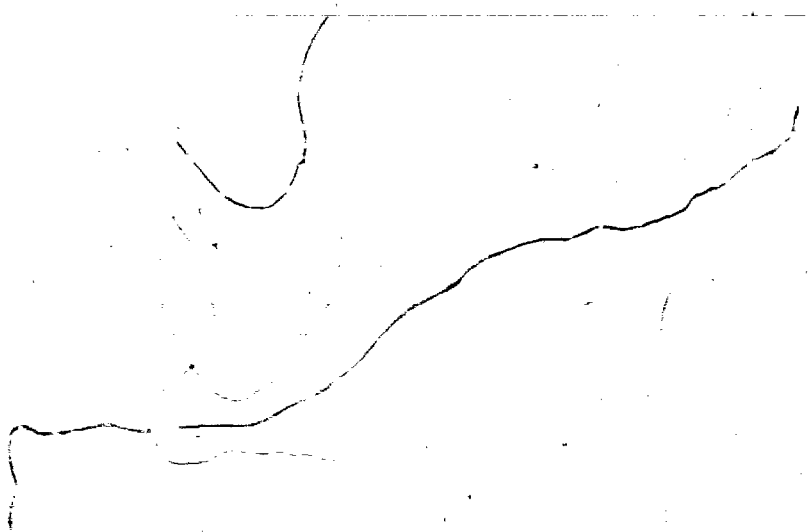


Figure 6-1: Example of Overlapping Sets

The Static algorithm will construct the schedule $S_1 * S_2 * S_3$, whereas the Dynamic algorithm will construct the schedule $S_1 * S_3 * S_2$. From the figure, it can be easily seen that the schedule $S_1 * S_3 * S_2$ will transmit much less data.

Regardless of the database distribution, however, if substantial elimination of redundant information occurs after the first file transfer, the Dynamic algorithm might not out-perform the Static algorithm. This is because after each file transfer, subsequent reordering of the schedule will not bring much improvement. Consider the two cases when the selectivities are 0.1 and 0.25 respectively. The data transmitted in the first round already accounts for 90 percent and 75 percent respectively of all the data transmitted. Thus, for low selectivities, the Dynamic algorithm may not do any better even if the constraint of the uniform distribution is removed.

The restriction of a uniform size (and hence selectivity) for all the data sets provides the worst case for the Static algorithm. Because of this, the algorithm has no knowledge, whatsoever, of how to construct the schedule. Thus, the schedule is constructed at random. Removing this restriction will improve the performance of the algorithm. The reason is that in the absence of any prior knowledge of the database, we can expect that a small data set will remain small after the intersection. Thus, the smallest data set identified by the Static algorithm will very likely still be the smallest data set after the intersection.



Node Order for Simple & Static	Simple	Static	Node Order for Dynamic	Dynamic
1	0	3000	1	3000
2	3000	289	2	289
3	3000	27	5	22
4	3000	0	3	0
5	3000	0	4	0
Total	12000	3316		3311

(a) Selectivity = 0.10

Node Order for Simple & Static	Simple	Static	Node Order for Dynamic	Dynamic
1	0	3000	1	3000
2	3000	759	5	735
3	3000	183	3	179
4	3000	42	2	42
5	3000	7	4	7
Total	12000	3991		3963

(b) Selectivity = 0.25

Node Order for Simple & Static	Simple	Static	Node Order for Dynamic	Dynamic
1	0	3000	1	3000
2	3000	1501	3	1500
3	3000	775	5	744
4	3000	390	4	378
5	3000	193	2	193
Total	12000	5859		5815

(c) Selectivity = 0.50

Node Order for Simple & Static	Simple	Static	Node Order for Dynamic	Dynamic
1	0	3000	1	3000
2	3000	2252	5	2239
3	3000	1681	3	1670
4	3000	1250	2	1250
5	3000	950	4	950
Total	12000	9133		9109

(d) Selectivity = 0.75

Table 6-1: Amounts of Data Transmission for the Different Algorithms

A comparison of the savings of the amounts of data transmitted as obtained from the Static and the Dynamic algorithms over the Simple algorithm is summarized in table 6-2.

	Data Transmission Savings	
Selectivity	**Static**	**Dynamic**
0.10	72.37%	72.41%
0.25	66.74%	66.97%
0.50	51.18%	51.54%
0.75	23.89%	24.09%

Table 6-2: Savings of the Static and Dynamic Over the Simple Algorithm

6.3.2. Data Transmission Time

The total data transmission time required by each algorithm to perform the intersection of 15000 records distributed evenly among five nodes is shown in table 6-3. These times, which are the averages from different runs, are also plotted in figure 6-2.

	Selectivity			
Algorithm	**0.10**	**0.25**	**0.50**	**0.75**
Simple	2.661	2.661	2.661	2.661
Static	0.996	1.256	1.525	1.808
Dynamic	0.895	1.240	1.395	1.675

Table 6-3: Data Transmission Time in Seconds

For the Simple algorithm, the amount of data transmission is not dependent on the selectivity. Whatever the selectivity is, the whole data set has to be transmitted, thus, the transmission time remains constant. For the Static and the Dynamic algorithms, less data needs to be transmitted for smaller selectivity, but the data

transmission time increases rapidly as the selectivity increases. When the selectivity approaches to one, the amount of data transmission will approach 100 percent of the data set, and therefore will require the same amount of transmission time as the Simple algorithm. This demonstrates the fact that for high selectivities, the Simple algorithm may have comparable performance.

The performances of the Static and the Dynamic algorithms are about the same. As we have seen in the previous section, the amounts of data transmitted is very much the same, and this is directly related to the transmission time. Moreover, the few extra numbers needed for data transmission with the Static algorithm, for most cases, can fit into the same data packet anyway.

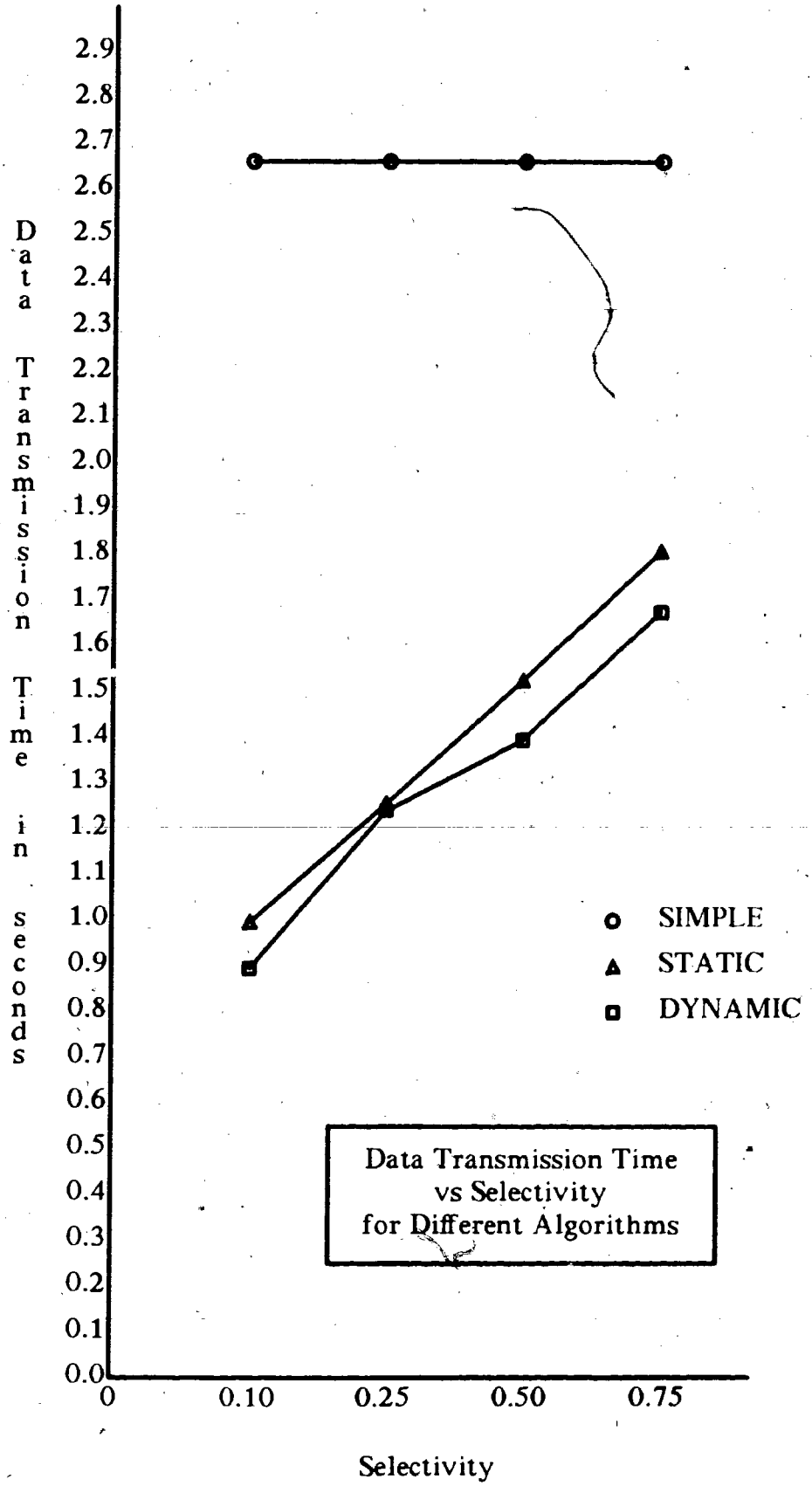


Figure 6-2: Data Transmission Time vs Selectivity

6.3.3. Local Operation Time

The result for the local operation time is shown in table 6-4 and is plotted in figure 6-3.

	Selectivity			
Algorithm	**0.10**	**0.25**	**0.50**	**0.75**
Simple	0.909	1.051	1.295	1.415
Static	0.420	0.460	0.620	0.848
Dynamic	0.450	0.460	0.610	0.835

Table 6-4: Local Operation Time in Seconds

As we can see, the operation time for the Simple algorithm increases just a little with increasing selectivity. This should be the case because the nodes are transmitting the same amount of data. The only difference is that for higher selectivity, the intersection set is larger, and thus, a longer time is required for the operation. The operation times for the Static and the Dynamic algorithms are very similar. This shows that not much is gained from the Dynamic algorithm. In fact, for small selectivity, the Dynamic algorithm takes longer than the Static algorithm because of the extra overhead involved. Only when the selectivity is large does the Dynamic algorithm performs better. Even then, the improvement is not that great.

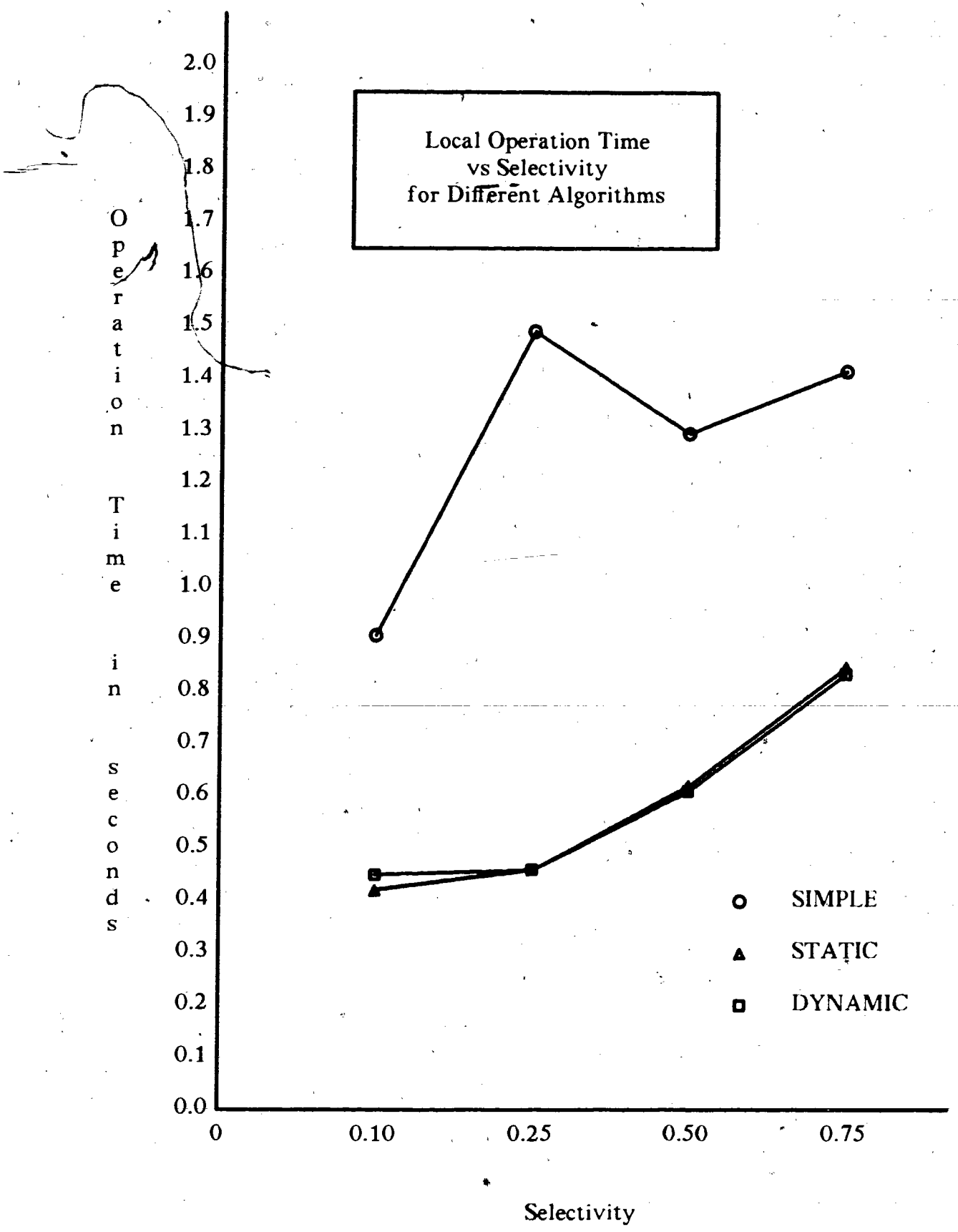


Figure 6-3: Local Operation Time vs Selectivity

6.3.4. Transmission vs Local Operation Time

When we derived our theoretically optimal algorithm in section 5.2.4, we assumed that the local operation time was very insignificant when compared to the data transmission time. We now want to look at our empirical results to see whether this assumption is indeed correct.

The ratio of data transmission to local operation is dependent on the particular algorithm used. We have listed in table 6-5 these ratios for the different algorithms as obtained from our experiment.

	Selectivity				
Algorithm	**0.10**	**0.25**	**0.50**	**0.75**	**Average**
Simple	2.93:1	2.53:1	2.05:1	1.88:1	2.35:1
Static	2.37:1	2.73:1	2.56:1	2.13:1	2.42:1
Dynamic	1.99:1	2.70:1	2.29:1	2.01:1	2.24:1

Table 6-5: Ratios of Data Transmission to Local Operation

These ratios show that the data transmission time is just a little over two times that of the local operation time. The local operation time is therefore very significant and thus cannot be ignored.

6.3.5. Handshake Time

The Simple algorithm does not require any handshake time, whereas the Static and the Dynamic algorithms require one handshake during the initial connection. In addition, the Dynamic algorithm requires one extra handshake per file transfer and each handshake uses a considerable amount of time. This is because the handshake

requires all the nodes to transmit their data set sizes to the scheduler and data transmission is time consuming. The handshake time also includes the construction of the schedule for the Static and the Dynamic algorithms. The handshake time is not dependent on the selectivities of the data, but rather on the algorithm used. Thus, it remains constant for the different selectivities as listed in table 6-6.

	Selectivity			
Algorithm	**0.10**	**0.25**	**0.50**	**0.75**
Simple	0.000	0.000	0.000	0.000
Static	0.059	0.059	0.059	0.059
Dynamic	0.511	0.511	0.511	0.511

Table 6-6: Handshake Time in Seconds

Notice that the handshake time for the Dynamic algorithm is much larger than that of the other two algorithms. When we compare the local operation time plus the handshake time with the three algorithms, (see figure 6-4), we find that the Dynamic algorithm is now much worse than the Static algorithm. The Dynamic algorithm now takes about the same amount of time as the Simple algorithm. In fact, when the selectivity is low, it is worse than the Simple algorithm, but gradually improves over the Simple algorithm when the selectivity increases. The reason is that the overhead for the many handshakes required by the Dynamic algorithm is too great to absorb the savings gained from the data reduction.

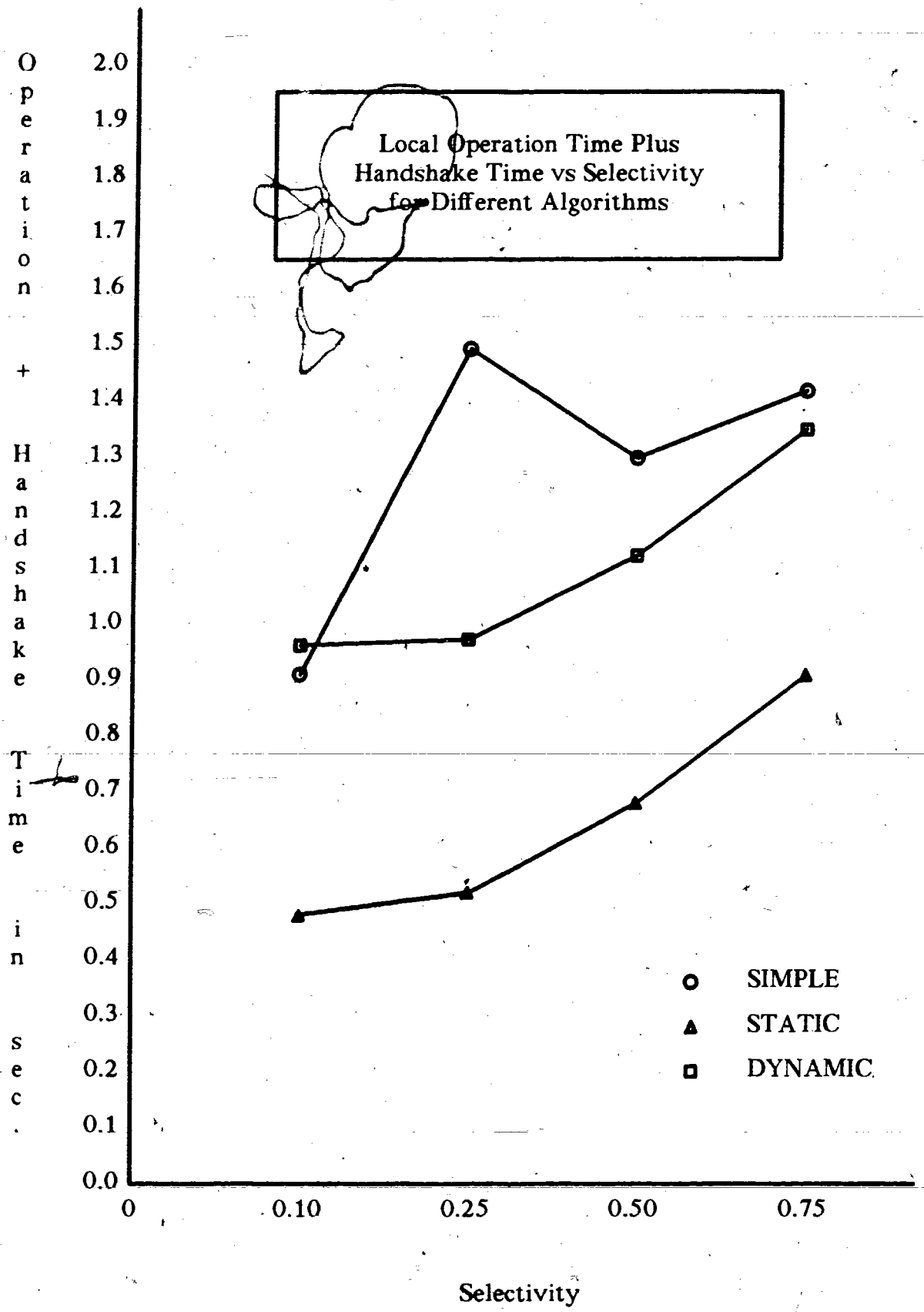


Figure 6-4: Local Operation Plus Handshake Time vs Selectivity

6.3.6. Handshake Time vs Transmission Time

The handshake time is constant and is independent of the data set size. It is, therefore, expected that as the data set size increases, the handshake time will be proportionally reduced relative to the transmission time or the local processing time. There are two ways in which the data set size can increase: (1) by increasing the size of the elements in the data set, and (2) by increasing the number of elements in the data set.

In the first case, we can have each element of the data set as a long data record instead of an integer. In this situation, the high percentage of handshake time of the Dynamic algorithm (see table 6-7), would be reduced, thus improving the performance of the Dynamic algorithm vis-a-vis the Static one. Note, however, that with larger record size, we need not transmit the whole record for doing the intersection. Just transmitting the keys of the records would be a good preprocessing strategy, because only the data records in the intersection need to be transmitted and transmitted once. This is basically what we are doing in our experiments. Moreover, as we have seen in section 6.3.1, the Dynamic algorithm will not out-perform the Static algorithm by much in terms of data transmission. Thus, taking everything into consideration, the Static algorithm will still have comparable results.

The second way to increase the data set size is to increase the number of elements in the data set. We expect that in this situation, when the data set is increased to over a certain size, the Dynamic algorithm will be better. We have, therefore, interpolated the results of table 6-7 for larger data sets. We will assume that with increasing data set size, the handshake time and the overhead remain constant, while the data transmission and the local operation time increase

proportionally. For each selectivity, let V_s be the sum of the transmission time and the operation time required by each byte in the data set by the Static algorithm and V_d for that of the Dynamic algorithm. Let C_s be the sum of the handshake time and the overhead time for the Static algorithm and C_d for the Dynamic algorithm. Let k denote the size of the data set such that the Static and the Dynamic algorithms produce identical elapsed time. Thus

$$V_s k + C_s = V_d k + C_d$$

or

$$k = \frac{C_d - C_s}{V_s - V_d}$$

The result from the interpolation is plotted in figure 6-5. In the area below the curve, the Static algorithm is better, and in the area above the curve, the Dynamic algorithm is better. We believe that the peak should not be there, but instead, a monotonically decreasing curve connecting the points. The peak in the curve may have been caused for two reasons. The first plausible reason is that, as our analysis of the data sets used in the experiment has shown, when the selectivity is 0.25, the deviation from the mean size of the intersection set of two random sets is larger than that for the other selectivities. Thus, the result from the experiment fluctuates more at this selectivity. The second possible reason is that the curve is an interpolation of the results for database sizes of 3000 bytes. Because we are interpolating these results for larger database sizes, the fluctuation in the experimental data is increased.

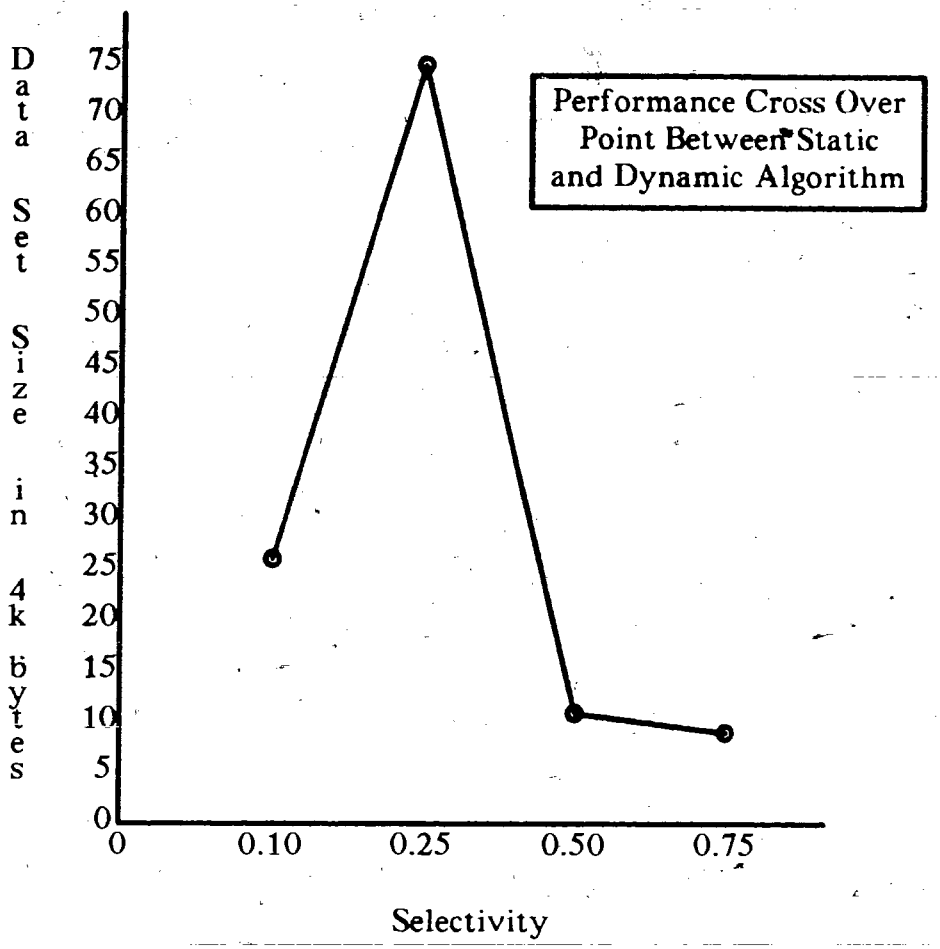


Figure 6-5: Performance Cross Over Point for the Algorithms

6.3.7. Total Elapsed Time

Finally, when we combine all the factors together, i.e., the transmission time, the local operation time, and the handshake time, we obtain the results as plotted in figure 6-6. The results are also summarized in table 6-7. The timings in the table are all in seconds. From this, it is clear that indeed when the database is small, the Static algorithm is the best algorithm among the three that we have implemented.

Simple	Selectivity			
	0.10	**0.25**	**0.50**	**0.75**
Bytes broadcast	48000	48000	48000	48000
Transmission time	2.66 (74%)	2.66 (71%)	2.66 (67%)	2.66 (65%)
Local operation time	0.91 (25%)	1.05 (28%)	1.29 (32%)	1.42 (34%)
Handshake time	0.00 (0%)	0.00 (0%)	0.00 (0%)	0.00 (0%)
Overhead	0.04 (1%)	0.03 (1%)	0.05 (1%)	0.04 (1%)
Total elapse time	3.61	3.74	4.00	4.12

Static	Selectivity			
	0.10	**0.25**	**0.50**	**0.75**
Bytes broadcast	13264	15964	23436	36532
Transmission time	1.00 (65%)	1.26 (69%)	1.53 (69%)	1.81 (66%)
Local operation time	0.42 (27%)	0.46 (25%)	0.62 (28%)	0.85 (31%)
Handshake time	0.06 (4%)	0.06 (3%)	0.06 (3%)	0.06 (2%)
Overhead	0.07 (4%)	0.06 (3%)	0.01 (.4%)	0.04 (1%)
Total elapse time	1.55	1.84	2.22	2.76

Dynamic	Selectivity			
	0.10	**0.25**	**0.50**	**0.75**
Bytes broadcast	13244	15852	23260	36436
Transmission time	0.90 (43%)	1.24 (54%)	1.40 (54%)	1.68 (55%)
Local operation time	0.45 (22%)	0.46 (20%)	0.61 (23%)	0.84 (28%)
Handshake time	0.51 (24%)	0.51 (22%)	0.51 (20%)	0.51 (17%)
Overhead	0.23 (11%)	0.11 (4%)	0.07 (3%)	0.01 (.3%)
Total elapse time	2.09	2.28	2.59	3.04

Table 6-7: Comparison Summary of the Three Algorithms

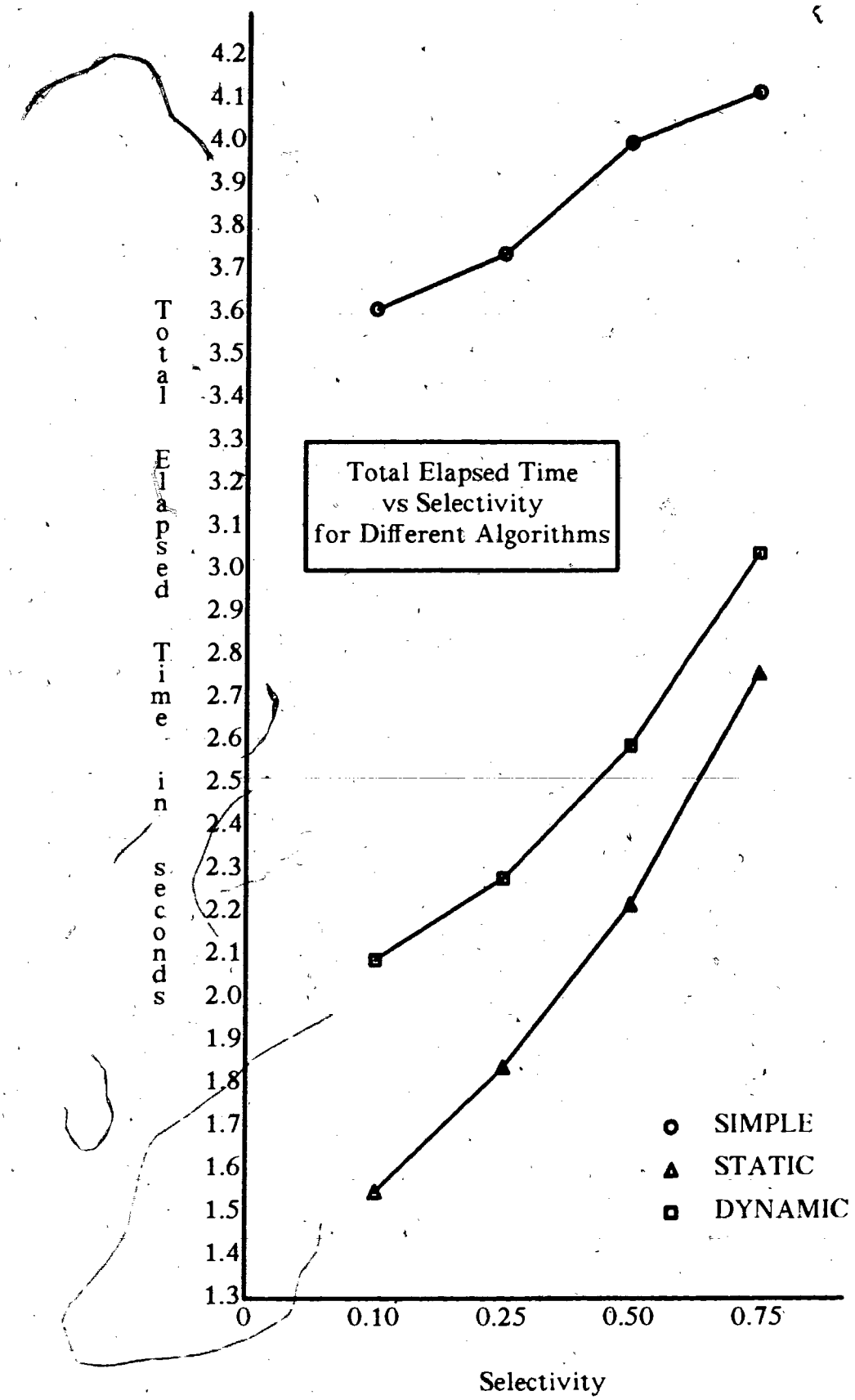


Figure 6-6: Total Elapsed Time vs Selectivity

Chapter 7
Conclusion

The objective of this thesis is to provide empirical results for algorithms to perform set intersections in a distributed database environment. We have noted that the set intersection operation is a very important database operation both in its own right and as a sub-operation for the join operation.

In searching for a good algorithm to perform set intersections, we have made use of the properties of a multiple-access broadcast network. For a broadcast network, it is possible for one node to broadcast a message and have all the nodes connected to this network receive this message simultaneously. As a result, all the nodes can proceed to process the received data in parallel. Moreover, this preprocessing of the data can eliminate all the redundant information from data to be transmitted in the future by these nodes. The data reduction process is very important in a distributed database system because the data transmission time is significant when compared with the local processing time. However, the broadcast method, which is only supported by the User Datagram Protocol, is unreliable. The reason is that UDP does not have any flow control mechanism, and so, packets are thrown away when the receiver is unable to buffer them.

The unreliability of the UDP can be solved if a flow control mechanism is implemented in the networking software, or it can be decreased if more buffers are available in our Ethernet controller and if the usage of the buffers can be improved.

Thus, our result may be slightly different if an Ethernet Controller different from the popular 3Com Ethernet Controller is used. We argue, however, that this alone may not be sufficient to alleviate the unreliability problem substantially. For example, if the number of nodes attached to the network increases, while the number of buffers in the controller is fixed, the buffer overflow problem will persist. We suspect that the heart of the problem lies in the BSD 4.2 networking software which disables interrupts from the Ethernet controller when manipulating the network data structures. However, further discussion on the modification of the networking software is beyond the scope of this thesis.

Given the networking software as it exists, a higher level broadcast protocol has to be designed on top of the UDP which guarantees that all the packets will be received by all the receivers. We have found that the protocol which uses only one acknowledgement after a file transfer with selective retransmission is the most efficient. The acknowledgement will specify which packets need to be retransmitted. The sender will selectively retransmit these lost packets. Moreover, since the transmission overhead per packet cannot be reduced, it is more efficient in using a large packet size as opposed to using a small packet size. The maximum packet size allowed is 1472 bytes, excluding the Internet address header.

The reduction of data transmission is dependent not only on the broadcast protocol used, but also on the algorithm selected for performing the set intersection. Of the three algorithms, Simple, Static, and Dynamic, that we have implemented in our experiments, only the Static and the Dynamic algorithms make use of the broadcasting properties. The Simple algorithm just ignores the broadcasting advantages. Because of this, we find that indeed, the Static and the Dynamic algorithms perform much better. The amount of data transmitted is between 23 and 72 percent less than

the Simple algorithm depending on the selectivity of the data. Between the Static and the Dynamic algorithms, we would expect the Dynamic algorithm to be better because it tries to reduce the amount of data transmission even more. However, it has turned out that this is not the case when the data sets are small (less than 36000 bytes each). With our assumptions of a uniformly distributed database, having initially the same size (and hence selectivity), the many reconstructions of the schedule done by the Dynamic algorithm do not reduce the amount of data transmission by much. The reason is that there is very little variation in the intersection set size, whether the set was chosen randomly, or the smallest set was chosen. Also, substantial elimination of redundant information occurs after the first file transfer, after which the reduction of data drastically decreases. Thus, the subsequent reconstruction of the schedule does not bring about much data reduction. On the other hand, the overhead required for the extra reductions far exceeds the benefit derived from the little amount of savings. The Static algorithm uses only the first ordering but not the subsequent ones, and so the overall performance is much better than the other two algorithms for small databases. We must caution the reader that the tradeoff between the reordering overhead and the possible data reduction must be carefully studied given prior knowledge of the characteristics of the database distribution over the network.

Appendix A

Pseudo Codes for the Broadcasting Protocols

A.1. Acknowledgement Per Packet With Packet Retransmission

A.1.1. Broadcaster

```
          Set sequence_number S = 1;
          DO{
Send:        Send packet S;
             Set timer;
             FOR C = 1 TO number_of_receivers{
                IF (timer interrupts)
                   THEN GOTO Send;          /* retransmit packet */
                Receive acknowledgement;
                }
             Reset timer;
             /* If received all acknowledgements, the program   */
             /* will reach this point without being interrupted */
             S = S + 1;
             } WHILE (more packets to be sent);
          Send end-reply packet;
```

A.1.2. Receiver

```
          Set sequence_number S = 1;
          DO{
             Receive packet P;
             Send acknowledgement for packet P;
             IF (P = S)
                THEN{
                   process packet P;
                   S = S + 1;
                   }
             } WHILE (P is not the end packet);
          Receive the end-reply packet;
```

A.2. Acknowledgement Per File With File Retransmission

A.2.1. Broadcaster

```
Send:   Set sequence_number S = 1;
        DO{
           Send packet S;
           S = S + 1;
           } WHILE (more packets to be sent);
        Set timer;
        FOR C = 1 TO number_of_receivers{
           IF (timer interrupts OR acknowledgement is not ok)
              THEN GOTO Send;    /* retransmit packet */
           Receive acknowledgement;
           }
        Reset timer;
        Send end-reply packet;
```

A.2.2. Receiver

```
Repeat: Set sequence_number S = 1;
        Set flag ACK = ok;
        DO{
           Receive packet P;
           IF (P = S)
              THEN process packet P;
           IF (P > S)
              THEN ACK = not ok;
           S = S + 1;
           } WHILE (P is not the end packet);
        Send acknowledgement ACK;
        IF (ACK = not ok)
           THEN GOTO Repeat;
        Wait for the end-reply packet;
```

A.3. Acknowledgement Per File With Selective Packet Retransmission

A.3.1. Broadcaster

```
        Set flag RETRANSMIT = false;
        Set sequence_number S = 1;
Send:   DO{
           IF (RETRANSMIT = true)
              THEN{
                 Select next lost packet S;
                 Send packet S;
                 }
              ELSE{
                 Send packet S;
                 S = S + 1;
                 }
           } WHILE (more packets to be sent);
        Set timer;
        FOR C = 1 TO number_of_receivers{
           IF (timer interrupts OR acknowledgement is not ok)
              THEN{
                 RETRANSMIT = true;
                 GOTO Send;      /* retransmit packet */
                 }
           Receive acknowledgement;
           }
        Reset timer;
        Send end-reply packet;
```

A.3.2. Receiver

```
        Set array ACK(i) = not ok for all i;
Repeat: DO{
           Receive packet P;
           IF (ACK(P) = not ok)
              THEN{
                 Process packet P;
                 ACK(P) = ok;
                 }
           } WHILE (P is not the end packet);
        Send acknowledgement ACK;
        IF (ACK(i) = not ok for any i)
           THEN GOTO Repeat;
        Wait for the end-reply packet;
```

A.4. No Acknowledgements Or Retransmissions

A.4.1. Broadcaster

```
Set sequence_number S = 1;
DO{
   Send packet S;
   S = S + 1;
   } WHILE (more packets to be sent);
Send end-reply packet;
```

A.4.2. Receiver

```
DO{
   Receive packet P;
   Process P;
   } WHILE (P is not the end packet);
Receive end-reply packet;
```

Appendix B

Timing Synchronization

B.1. Simple

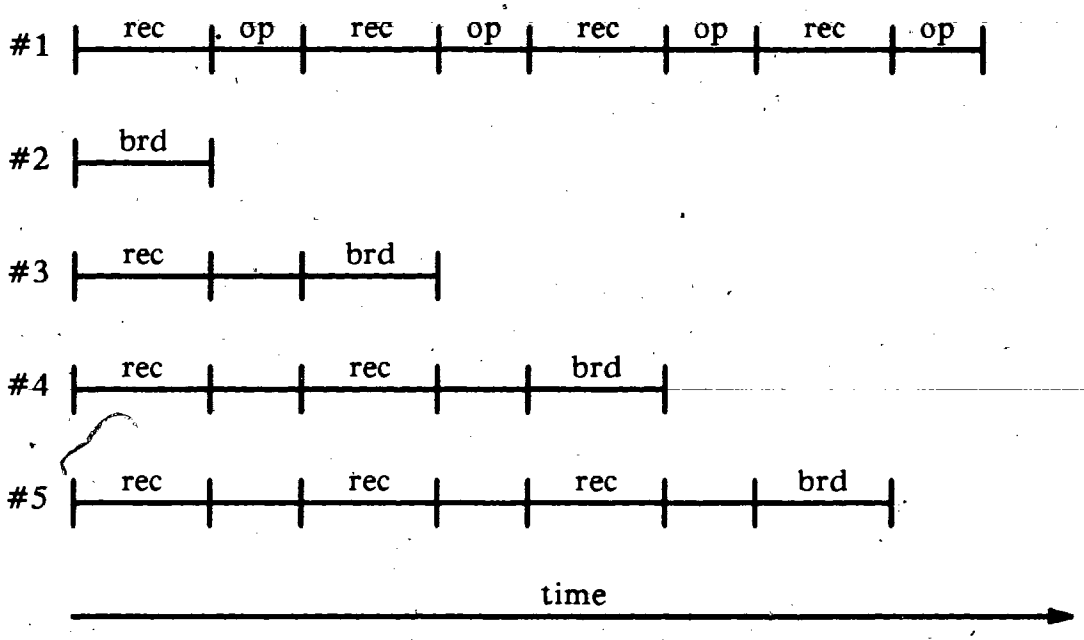


Figure B-1: Timing Synchronization for the Simple Algorithm

hnk = handshake time
brd = broadcast time
rec = receive time
op = local operation time

B.2. Static

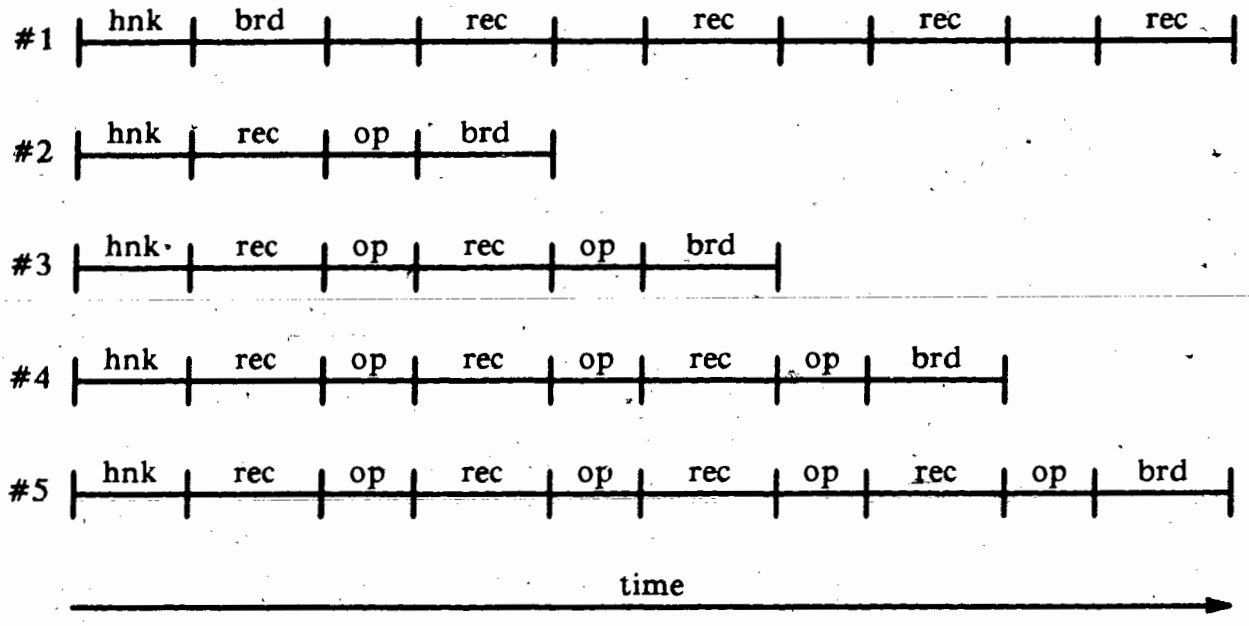


Figure B-2: Timing Synchronization for the Static Algorithm

B.3. Dynamic

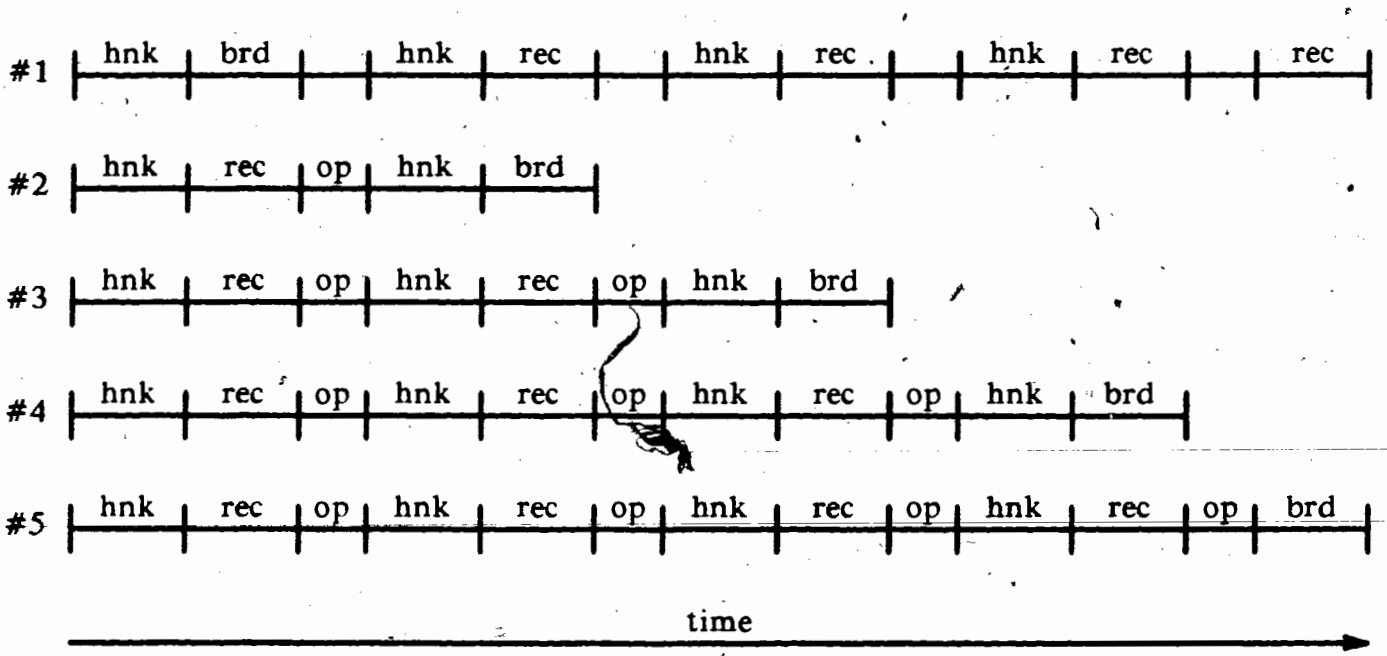


Figure B-3: Timing Synchronization for the Dynamic Algorithm

Appendix C

Analysis Of The Intersection Set Size

In this analysis, we want to show that given any two independent sets containing uniform distributed data and uniform selectivity, the intersection set size k is very close to a certain mean. We will show this, first, by calculating the probability of k inside a certain range of the mean, and then, we will show this with some empirical results.

C.1. Calculation Of The Probability

Let A and B be two sets containing uniformly distributed data, each with N elements selected randomly from a set of M elements. Let k be the intersection set size of the two sets A and B, i.e., $k = |A \cap B|$, where $0 \leqslant k \leqslant N$. Below, we will derive the probability function of the random variable k.

For $k = 0$, the number of possible ways of selecting two sets of N elements each from a set of size M is

$$\binom{M}{N}\binom{M-N}{N}$$

For $k = 1$, we get

$$\binom{M}{1}\binom{M-1}{N-1}\binom{M-N}{N-1}$$

In general, for $k = i$, we get

$$\binom{M}{i}\binom{M-i}{N-i}\binom{M-N}{N-i}$$

Thus, the general probability function for any k, where $0 \leq k \leq N$ is

$$\frac{\binom{M}{k}\binom{M-k}{N-k}\binom{M-N}{N-k}}{\binom{M}{N}\binom{M}{N}} \qquad \text{for } k \geq 2N - M$$

and 0 for $k < 2N - M$

Using this probability function equation, the probabilities for different values of k and M have been calculated. N is fixed at 3000. Table C-1 lists the probabilities of getting a certain k within a certain range of the mean for four different values of M.

Selectivity	Mean ± Range	Probability
0.10	300 ± 1%	15%
0.10	300 ± 3%	48%
0.10	300 ± 5%	66%
0.10	300 ± 10%	95%
0.25	750 ± 1%	29%
0.25	750 ± 3%	72%
0.25	750 ± 5%	93%
0.50	1500 ± 1%	56%
0.50	1500 ± 3%	98%
0.75	2250 ± 1%	94%

Table C-1: Probabilities of Getting k Within a Certain Range of the Mean

On the average, over 90 percent of the times, k is within the range of ± 30 elements from its mean for the various selectivities. This conclusively shows that given two uniformly distributed sets, the probability of getting the intersection set size of these two sets to be close to the mean is very high.

C.2. Empirical Results

Table C-2 below shows some empirical results in order to justify the apparent lack of variations between columns 3 and 5 in table 6-1. For each selectivity, we generate six random sets of size 3000 each from a given range of integers. (The ranges are 1-30000, 1-12000, 1-6000, and 1-4000 for selectivities 0.10, 0.25, 0.50, and 0.75 respectively). Then intersections are performed between one chosen set and all other five sets. The columns RAND1 to RAND5 contain the intersection set size of a common set with the other five sets. The MIN column contains the minimum set size among the five intersections. The AVG column shows the expected difference in size, between a randomly chosen set from RAND1 to RAND5 and the minimum of the five sets.

Selectivity = 0.10

MIN	RAND1	RAND2	RAND3	RAND4	RAND5	AVG
296	303	296	324	303	313	11
283	324	283	309	301	304	21
285	316	299	285	301	310	17
276	296	320	276	301	314	25
296	296	304	296	312	303	6
292	307	334	345	292	296	22
270	283	307	300	270	285	19
289	289	296	294	303	300	7

Selectivity = 0.25

MIN	RAND1	RAND2	RAND3	RAND4	RAND5	AVG
724	724	776	733	741	750	20
742	767	742	743	749	791	16
730	760	753	769	730	756	23
718	762	718	731	788	739	29
751	766	770	751	755	786	14
741	766	761	741	754	762	15
728	769	731	728	763	743	18
749	762	752	757	751	749	5

Selectivity = 0.50

MIN	RAND1	RAND2	RAND3	RAND4	RAND5	AVG
1474	1474	1503	1481	1523	1512	24
1461	1499	1507	1480	1532	1461	34
1477	1516	1477	1514	1511	1504	27
1502	1502	1512	1518	1508	1529	11
1496	1515	1552	1519	1496	1506	21
1487	1504	1487	1503	1503	1528	18
1469	1508	1469	1478	1522	1491	24
1480	1485	1510	1529	1480	1500	20

Selectivity = 0.75

MIN	RAND1	RAND2	RAND3	RAND4	RAND5	AVG
2243	2243	2261	2262	2255	2250	11
2242	2242	2247	2273	2243	2247	8
2236	2258	2244	2236	2256	2246	12
2237	2251	2237	2246	2254	2270	14
2242	2258	2242	2264	2250	2248	10
2246	2246	2264	2258	2252	2246	7
2241	2260	2241	2261	2246	2242	9
2239	2240	2239	2260	2244	2249	7

Table C-2: Empirical Results of the Intersection Set Size

References

[BaDr84] Babaoglu, O. and Drummond, R., "Communication Architectures for Fast Reliable Broadcasts", Computer Science Dept., Cornell University, Ithaca, N.Y., Oct. 1984.

[BeGo81] Bernstein, P.A., and Goodman, N. "Concurrency Control in Distributed Database Systems", ACM Comput. Surv., June 1981, Volume 13, Number 2, pp. 185-221.

[Chang84] Chang, Jo-Mei, "Simplifying Distributed Database Systems Design by Using a Broadcast Network", ACM Sigmod, June 1984, pp. 223-233.

[ChMa84] Ghang, J.M., and Maxemchuk,N.F., "Reliable Broadcast Protocols" , ACM Transactions on Computer Systems, August 1984, Volume 2, Number 3, pp. 251-273.

[Chris81] Christodoulakis, S., "Estimating Selectivities in Databases", Technical Report CSRG-136, Computer Science Dept., University of Toronto, 1981.

[Codd71] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", CACM, Volume 13, Number 6, pp. 377-387.

[Date81] Date, C.J., "An Introduction to Database Systems", 3rd Ed., Addison-Welsey, 1981.

[DaBaPrSo79] Davies, D.W., Barber, D.L.A., Price, W.L. and Solomonides, C.M., "Computer Networks and Their Protocols", John Wiley & Sons, 1979.

[Luk84] Luk, W.S., "An Optimal Algorithm to Perform Set Intersections and/or Unions on a Broadcast Network", SFU Technical Report TR 84-12, 1984.

[LkBk81] Luk, W.S. and Black, P.A., "On Cost Estimation in Processing a

Query in a Distributed Database System", Proceedings of IEEE COMPSAC 1981, pp. 24-32

[MetBoggs76] Metcalfe, Robert M. and Boggs, David R., "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, July 1976, Volume 19, Number 7, pp. 395-404.

[PaPo85] Page, Thomas W. Jr. and Popek, Gerald J., "Distributed Data Management In Local Area Networks", 4th ACM Sigact-Sigmod Symposium on Principles of Database Systems, March 25-27, 1985, pp. 135-142.

[ShHu80] Shoch, John F. and Hupp, Jon A., "Measured Performance of an Ethernet Local Network", Communications of the ACM, December 1980, Volume 23, Number 12, pp. 711-720.

[Spector82] Spector Alfred Z., "Performing Remote Operations Efficiently on a Local Computer Network" Communications of the ACM, April 1982, Volume 25, Number 4, pp. 246-259.

[Stallings84] Stallings William, "Local Networks", Macmillan Publishing Company, New York, 1984.

[Stonebraker77] Stonebraker, M.R. and Neuhold, E., "A Distributed Database Version Of Ingres", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

[Tanenbaum81] Tanenbaum, Andrew S., "Network Protocols", Computing Surveys, December 1981, Volume 13, Number 4, pp. 453-489.