University of
Waterloo Research Institute

THE ESTABLISHMENT OF A TELIDON

TECHNICAL CENTRE

FINAL REPORT

APRIL, 1982

WATERLOO RESEARCH INSTITUTE
OFFICE OF RESEARCH ADMINISTRATION
UNIVERSITY OF WATERLOO

PROJECT NO. 908-01

THE ESTABLISHMENT OF A TELIDON TECHNICAL CENTRE

FINAL REPORT

by

Eric Manning
F.W. Tompa
G.H. Gonnet
R.R. Williams
V.F. DiCiccio

Sponsored by

Department of Communications
Canada

under

Contract with

Department of Supply and Services
No. OER80-00099

Mr. R. Baser
Scientific Authority

April, 1982

EXECUTIVE SUMMARY

OVERVIEW

This is the final report for DSS contract no. OER80-00099 performed by the Computer Communications Networks Group at the University of Waterloo. The study activity and this report can be partitioned into three main areas: general familiarization, videotex network architectures and database architectures.

The familiarization activities have included contact with other researchers, experience with database management software and the creation and display of Telidon pages. An important result of this direct experimentation has been the insight gained into the communication, storage and retrieval of Telidon information.

The study of Telidon network architectures has dealt primarily with the distribution network between the database and the user terminals. Initially, the various designs for videotex and teletext networks were enumerated and compared. In particular, the Omnitel integrated services data network has been modelled in considerable detail and its performance has been determined using both analytic and simulation techniques. The results indicate that Omnitel performs well for videotex traffic when mixed with traffic from low data rate digital services. However, there may be significant congestion if high levels of non-videotex traffic from sources such as digital telephony are introduced.

Also, a technique called "encryption-switching" has been devised for the delivery of Telidon and Telidon-related services over a CATV network. Local Area Network techniques are used to transmit the data in broadcast mode, but encryption methods prevent subscribers from intercepting data which they are not entitled to receive. Preliminary performance calculations suggest that an encryption-switching network based on state-of-the-art hardware may be capable of handling significant numbers of subscribers (a large fraction of the CATV subscribers) and is incrementally expandable.

The database architecture study has been divided into two main topic areas: the database structure and retrieval facilities as seen by the user and the database architecture used at the system level. Page labels have been introduced as a means for users to access pages, and implemented in the Waterloo Telidon software package; several problems relating to the use of labels and menus have been identified and solutions have been devised.

A generic model at the system level for the maintenance of a Telidon database has been proposed. This model includes a cache memory, secondary storage (disk) and an index to translate an absolute page identifier into a disk address for each page. Simulations of the system based on this model were used to investigate the effects of page cache strategies, cache size and index strategies on the system

performance. It was found that the use of a cache and an appropriate indexing strategy can reduce the number of disk accesses to less than one per page displayed. This result has major economic significance because the number of disk accesses required per page usually determines the number of simultaneous users a system can support with a given hardware cost. Furthermore, the performance of current Telidon systesm, such as Waterloo Telidon, can be significantly improved through the use of an appropriate cache.

The following sections summarize the important activities and results of each aspect of this study in more detail.

FAMILIARIZATION

Throughout this study we have had a great deal of contact with other researchers and field trial administrators in Canada, the United States, England, France and Germany, through field trips and conferences. Initially, our sole purpose was to gather information about current and future thinking; however, the emphasis began to shift towards the coordination of research activities and the sharing of ideas.

At the University of Waterloo there now exists a complete Telidon system as required to offer Telidon services campus-wide. Approximately thirty user terminals and a Norpak IPS-2 have been obtained (which means that UW's field trial is one of Canada's largest at this time). This has

permitted various faculties, departments and groups to create pages for installation on our two databases, both of which use locally written database software. The Waterloo Telidon database software has adopted the features of the DOC user interface plus some enhancements such as page labels of various kinds. The software has been running reliably for some time on a PDP-11/45 in CCNG, a VAX-11/750 (at Loyalist College in Belleville) and a VAX-11/780 in the UW Faculty of Mathematics. The availability of database software which is thoroughly understood and can be easily modified by local researchers has been a great help in experimenting with new database access techniques for this contract, as well as in providing an advanced Telidon service to UW users. Finally, we hope to have remote users from High Schools, the Ontario Government field trial, and the UW Correspondence Program.

NETWORK ARCHITECTURES

A general reference model for videotex network architectures was developed. It contains several data networks including the distribution or delivery network which connects the users' terminals to the point of access for the database. Our attention has focussed primarily on architectures for the delivery network.

The strategies which have been proposed or implemented for delivery networks can be divided into two groups: one-way distribution (i.e. teletext) networks and two-way

distribution (i.e. interactive Videotex) networks. Ceefax, Antiope and the CBS St. Louis field trial are examples of one-way networks which were examined in detail.

Two-way network schemes make use of a variety of communications media and signalling techniques. Prestel, Vista, Bildschirmtext and Captain all use two-way telephone delivery networks. QUBE and the London (Ontario) Cable TV system both use two-way cable TV data transmission as does the encryption-switching system proposed in this report. Hybrid systems such as data broadcast in the vertical blanking interval of the TV signal for downstream data, and telephone for upstream data, have been proposed. Such a scheme was considered for field trials of Antiope and for a system at KSL-TV. A more ambitious alternative is the Integrated Services Network typified by Omnitel, which was chosen as the distribution network for Project Ida; Omnitel is based on cable with intelligent switching nodes distributed throughout the network.

In general, two-way cable networks offer the advantage of potentially higher data rates, as compared to two-way telephone data transmission using the ubiquitous 1200 bps modem. Integrated Service Networks allow both data and video transmission, unlike the telephone. However, the examples of two-way data transmission on existing CATV networks have used low data rates and have had difficulties with the ingress of RF noise into the Cable TV plant.

Ingress of RF noise is likely to be expensive to correct; it should not be overlooked when videotex policy is formulated.

OMNITEL PERFORMANCE

The performance of the Omnitel network for the delivery of videotex service has been investigated in detail. A model of the delay of messages in the upstream and downstream directions was constructed using servers and queues. Servers were used to represent the processing and switching of messages and queues were used to model messages waiting for processing at switching nodes.

A queueing-theoretic analysis of the model was was used to generate the majority of the performance results, such as response time, waiting time and location of system bottlenecks. However, since certain assumptions are required to make the analysis tractable, a simulation program was developed based on the model, and the simulation results were used to verify the assumptions made in the analytic solutions. An exact match between simulation and analysis was obtained at low traffic levels, however, divergence occurred at higher traffic levels near the saturation point for any of the network components. This was due to the breakdown of the analytic assumptions about the traffic distribution and the more accurate nature of the simulation. However, both solutions exhibit saturation at similar input loads or traffic levels. Therefore, the analytic results were judged reliable.

The analytic results indicate that Omnitel performs well for videotex traffic mixed with other fairly low data rate digital services such as remote alarm detection and meter reading. However, congestion will occur if high levels of non-videotex traffic, such as digital telephony without data compression, are introduced. As a general rule, the first multiserver switching unit encountered in the direction of data flow under consideration (i.e. upstream or downstream) will act as the system bottleneck. (These components are the DCT in the downstream direction and the IDT line for upstream traffic.) Under some traffic conditions, the RVDM unit may also act as a system bottleneck.

ENCRYPTION-SWITCHING

CATV networks provide an attractive alternative to the telephone network or public packet-switched networks for the delivery of Telidon traffic. Local area network techniques, especially those developed for broadband bus networks, can be adopted for the provision of data services on CATV. This strategy differs significantly from more conventional approaches such as Omnitel, where a hierarchy of switching nodes is distributed throughout the network to switch circuits, data streams or data packets in order to conserve bandwidth on the network trunks. In a bus type local area network, such as we can build on the CATV plant, all downstream data packets are broadcast to all subscribers.

Each subscriber has a microprocessor-controlled interface box to examine all the downstream packets, and accept only those addressed to or of interest to the subscriber. However, since all downstream data packets are available at all subscriber interface boxes, the data should be encrypted to provide security and protect the service providers' revenues. The encryption and decryption can be provided at the interface box using the Data Encryption Standard (DES) algorithm which is available as a single chip. A 56 bit key is required for this algorithm; the same key is used for both encryption and decryption.

Four classes of traffic have been identified and techniques for controlling access to these services through the use of keys have been devised:

a) "bubble-pack" information - This is the news, weather and sports type of information, of interest to all subscribers and typically chosen for transmission on teletext systems. A single key can be used system-wide to protect this service. The key is issued to every subscriber at subscription time.

b) interactive videotex and transaction services - This information should be encrypted, using individual keys for each subscriber which are issued at subscription time. If transaction services such as EFTS warrant additional security, a session key can be negotiated under the protection of the per subscriber key.

c) closed user groups or secondary interactive videotex services - This information can be encrypted with a group key or a per subscriber key. In either case the key is issued at subscription time.

d) point-to-point subscriber service - This service allow subscribers to communicate directly. A session key is required for security. The session key may be negotiated between the subscribers using

Merkle's method or it can be requested from a central key server.

A principle, which we have termed the Accountability Requirement, states that each subscriber must take a deliberate, verifiable action to request a class of information, so that he can be billed for such accesses and so that it can be proved that he requested such access. This accountability requirement is satisfied for the first three classes of traffic using the key distribution techniques described above. If Merkle's method is used for determining the session key for subscriber-to-subscriber service, the accountability requirement is not satisfied, however, the central key server provides the mechanism for accountability.

Calculations of the number of subscribers an encryption-switching CATV network can support were done using realistic traffic estimates and the capacity of the Sytek LocalNet 20 broadband local area network. This is a commercial product, in service at UW, which provides broadband data service on a Cable TV plant. The Sytek network has 120 channels, each using 300 Khz of bandwidth. In an encryption-switching system each channel may support one of the following three activities:

a) approximately 700 pages of "bubble-pack" information accessible to all subscribers, or

b) approximately 300 active subscribers performing videotex information retrieval or transaction services; or

c) approximately 100 subscriber-to-subscriber "calls".

DATABASE ARCHITECTURE

One of the main requirements of a database architecture is convenience for the user. This study has determined that the tree structure of Telidon pages should be eliminated since its usefulness is outweighed by its apparent restrictions and the resulting confusion to users. Furthermore, the use of absolute, numeric page identifiers has at least two drawbacks: users tend to make mistakes, and the use of page identifiers tied directly to page addresses makes reorganization of the database awkward. ("Where did my pages go?")

The solution is to adopt a menu-based interface using labels such that the users are not aware of either the numeric page identifiers or the underlying tree structure of pages. However, the direct substitution of alphanumeric labels for numeric page identifiers does not solve the problem; the function of labels and the ways that they are used must be examined.

A taxonomy was developed for the characteristics of menus. Menus do not have to be explicitly displayed on the screen. A "page-specific" menu is only usable at a particular page (e.g. the DOC single digit menus). A "page-independent" menu is usable from everywhere in the database (e.g. Waterloo Telidon page labels).

Based on a study of the use of labels, the following types of menus are necessary:

a) a universal or page-independent menu of alphanumeric

labels, although not every page needs its own label;

b) page-specific menus of arbitrary size (i.e. more than 10 labels); and

c) private page-independent menus created by each subscriber to provide convenient direct access to his favourite pages.

When multiple menus can exist simultaneously, if two or more menus contain identical labels attached to different pages, a method must be adopted for resolving ambiguity. There are at least three possible solutions. The DOC Telidon software uses a syntactical distinction in the labels of page-specific and page-independent menus so that the ambiguity does not arise. Waterloo Telidon has relatively few syntactical restrictions for labels but enforces a fixed order for searching menus, so that the page is displayed which is attached to the ambiguous label, in the first menu in which the label is encountered. As a third possibility, the system could request and act on further discriminating input from the user. The last alternative is potentially the most satisfactory solution, and requires further study.

The combination of labels in a single request is a natural extension to the use of labels for page access. This may take the form of a Boolean expression of labels, or, the user may fill in a form electronically to specify the information that he wishes to receive. The study of form creation, maintenance and use may be crucial for providing an acceptable user interface for advanced videotex applications.

A set of commands has been devised to permit the information provider to attach a new page to the database and link it both to and from existing pages. The concept of a page frame, which is the part of a page which holds the interpage linking information, was introduced. Commands for removing pages and deleting labels have also been created.

Some issues relating to page creation and access have been identified and potential solutions have been suggested as guidelines for future work. These include:

a) dynamic menu maintenance and concurrency control (What if two inputters attach the same label in a page-independent menu to two different pages simultaneously?);

b) true keyword processing including a means for handling multiple page retrieval (Keyword access allows the user to retrieve all pages which contain certain words or strings of words specified by the user.);

c) the management of sets of related pages as a unit, perhaps in structures other than the tree; and

d) various techniques to expand the range of information or services for distributed videotex databases.

FILE STRUCTURES FOR DATABASE STORAGE

Alternative techniques for maintaining Telidon databases at the system level have been studied. This was done by developing a generic model for videotex file structures. The model included alternative search strategies to translate from numeric page identifiers to disk addresses and alternative page cache strategies. The model was simulated to determine the effects of these alternatives on

performance. Here is some more detail about these activities.

The generic model for videotex file structures included a page cache memory (in fast primary storage), and secondary storage (disk). It was assumed that a disk address is necessary to retrieve a page from the disk and that each page contained the disk addresses of the page-specific menu choices and nearest neighbours in the tree which were accessible from that page. Therefore, the disk address of a requested page could frequently be obtained from a page in the cache. If not, then a large, dictionary-like table which translates numeric page identifiers into disk addresses was consulted. Because of its size, this index table must be stored in the disk, although frequently used portions of the table can be stored in the cache.

Three alternative indexing techniques were proposed for this translation table: B-trees, digital trees and hashing schemes using buckets. The first and last of these schemes were incorporated in the model.

The model was simulated to determine the sensitivity of system performance to cache size and indexing strategy. It was found that the use of a cache is valuable and can reduce the number of disk accesses to an average of less than one per page request. Also, if a cache is used, both B*-trees and hashing tables when used as indexing strategies yield equivalent performance. However, the model used for the

simulations contains some approximations, and should be validated in particular cases by the comparison of predicted and actual behaviour before the results are used for the design of a production system.

TABLE OF CONTENTS

LIST OF FIGURES

I. Introduction

This is the final report for DSS contract no. OER80-00099. It describes the progress we have made and the activities we have undertaken at the University of Waterloo to estalish a centre for Telidon technology.

This report is organized along the lines of the activities described in the Statement of Work, as revised in March 1981. Section II describes the contact we have had with other researchers, the Telidon equipment we have acquired and the status of Telidon activities at the University of Waterloo which are not specifically a part of this contract.

Section III, entitled "Network Architectures", describes various teletext and videotex networks which we examined as an information-gathering excercise early in this study. It also presents the results of a detailed study of the performance of Omnitel and describes a technique we have developed called "encryption-switching" for the delivery of Telidon service on CATV systems.

Section IV contains the results of our study of database architectures for Telidon. Issues pertaining to a menu-based interface based on labels to a Telidon database are discussed and alternatives and recommendations are presented. Details of a model for the file structure and retrieval of Telidon pages are given and recommendations are made based on the results of a simulation study.

II. Familiarization and Telidon Promotion

In the contract work statement, "familiarization" was broken down into four areas: a literature search; contact with other researchers; the procurement of two Telidon terminals; and the installation of Telidon server software in our laboratory. All four areas have been successfully completed, with accomplishments in some areas which far exceeded our original goals.

The literature search was the first activity under this contract, but it has continued up to the present time. An initial list of documents was listed in the first interim report (March 31, 1980), and this has been the basis of the literature collection. Since then, literature has been collected from various conferences and workshops that we have attended, and we have maintained awareness of current videotex publications and events.

Contact with other researchers has also been an ongoing effort. Initially, Dr. David Morgan visited many researchers in Canada, the United States, and in Europe. (A list of researchers contacted by Dr. Morgan was included in the first interim report.) Dr. Morgan also attended Viewdata '80 in London, England, in March 1980, and contacted researchers there. Since then, Dr. Frank Tompa has travelled to a number of conferences and workshops, making valuable contacts. He has attended "Inside Videotex" in March 1980, the "First Montreal Workshop on Videotex Tech-

nology" in June 1980, and "Videotex '81" in May, 1981. As well, we hosted the "Second International Videotex Workshop" at the University of Waterloo, which brought researchers together from Canada, Britain, Germany, and Japan. Most recently, Dr. Tompa travelled to Germany and Rennes, France in early December, partly funded by this contract. While in Rennes, Dr. Tompa talked to D. Le Moign and Y. Yclon of CCETT to discuss the current status of Teletel and the "electronic directory". In addition several researchers have visited CCNG to talk to us. These have included Keith Clarke of the British Post Office, Roger Woolfe of Butler Cox and Partners, Ltd., and John Wicklein. We also met with Jan Gecsei and Gregor Bochmann from the University of Montreal in late 1980, with whom we discussed Telidon research, and attempted to coordinate our work.

The last two sections of the familiarization process involved obtaining two Telidon terminals, and installing the DOC Telidon host software in our lab. The procurement of terminals has been quite successful. Initially, we borrowed a Telidon decoder, monitor, and keypad from OECA as part of their trial. We used this terminal to explore the DOC Telidon system in Ottawa, and to get a "feel" for the capabilities and functions of the equipment. After some time, a second (integrated) terminal, equipped with a simple keypad/keyboard, was borrowed from Electrohome Ltd. Finally, in mid 1981, the University of Waterloo made an

agreement with Electrohome whereby we would acquire thirty terminals in exchange for computing services. These terminals have already been delivered, and many have been put for use in various sites at the University.

We unfortunately were unable to install the Telidon host software written at DOC. Our problems basically stemmed from a lack of a proper hardware and software environment, and our lack of RSX-11 experience. We did not have the correct version of RSX-11M or of the various pieces of software needed to support the Telidon system. After many delays, we eventually got the necessary permission from the Digital Equipment Corporation to run the software, and we received the complete software from DOC. At this point, our problems seem to have been caused by our hardware environment, which includes some non-DEC equipment not anticipated in the operating system.

After much frustration, we decided to write our own version of Telidon, (called Waterloo Telidon), independent of this contract. This software was written using the Unix operating system, where most of our operating system expertise lies. We endeavoured to make this version of a Telidon host compatible with the original DOC version (there are some minor differences), and as well, we have enhanced it to include some of the ideas presented in section IV of this report. In particular, instead of restricting index (or menu) choices to the digits 1 through 9, we also allow ar-

bitrary alphanumeric labels. In addition, we have three levels of labels for pages: local labels (which are like index choices); user labels, that allow each user to identify a set of chosen pages; and global labels, set by the system operator to identify a set of pages for the entire user community. Waterloo Telidon has been running reliably for some time in our lab on a PDP-11/45, and has been installed successfully on the Math faculty's VAX-11/780, and at Loyalist College in Belleville on a VAX-11/750. Documentation for Waterloo Telidon is available on request.

In addition to acquiring Telidon terminals and developing the CCNG Telidon software, we acquired a Norpak IPS-2 Information Provider System as part of the modified work statement. This machine has been in steady use since it was acquired and is presently in use twelve hours per day. Various groups on campus, including the Faculty of Mathematics, the Faculty of Human Kinetics and Leisure Studies (HKLS) and the University of Waterloo Correspondence Program have provided funds for the development of pages for our database. Since the summer of 1981, we have had between one and three co-op students (from UW's Department of English) producing pages on the IPS and on Unix using locally written versions of Textcon and Pdicon (these programs translate PDIs to and from an english-like equivalent). The pages produced will be used during the University's twenty-fifth anniversary celebrations, in an Ontario

Government field trial, and hopefully for remote access by high schools and Correspondence students.

With the addition of an IPS, we now have a complete on-campus Telidon network. Having all three major parts (terminals, a host database, and an IPS) has helped immeasurably in our understanding of Telidon systems. The process of writing the database software contributed significantly to our knowledge, and has enabled us to try out some of the ideas on information retrieval discussed in section IV. The IPS helped increase our technical knowledge in this area, as well as letting us gain expertise in designing pages and using PDIs to their fullest extent. The acquisition of the terminals has made everything discussed so far possible, and by testing Electrohome terminals thoroughly, we have been able to make constructive criticisms on their design to Electrohome. Finally, our activity constitutes one of the larger Telidon field trials in existence at this time.

III. Network Architectures

III.1 Introduction

This section describes three different activities in our study of network architectures. Section III.2 presents a survey of existing subscriber distribution schemes, and it represents our initial efforts to familiarize ourselves with the videotex world. The second section describes a detailed study of the performance of an existing distribution scheme, Omnitel, an integrated services network designed by John Coyne for Project Ida. A queueing model for this network was developed, and analytical and simulation techniques were used to obtain performance results. Section III.4 proposes a new method for delivering Telidon on CATV systems, which we call "encryption-switching." This method uses a combination of digital network broadcast techniques (commonly used in local area networks), and encryption.

Throughout section III, we have attempted to use a uniform terminology, which is summarized in the Videotex Network Architecture Reference Model in figure 3.1. In this model, the network is broken up into four separate components: the server network, the information provider network, the "third-party" network, and the distribution network.

With potentially thousands of subscribers wishing simultaneous access to the database, a large videotex network

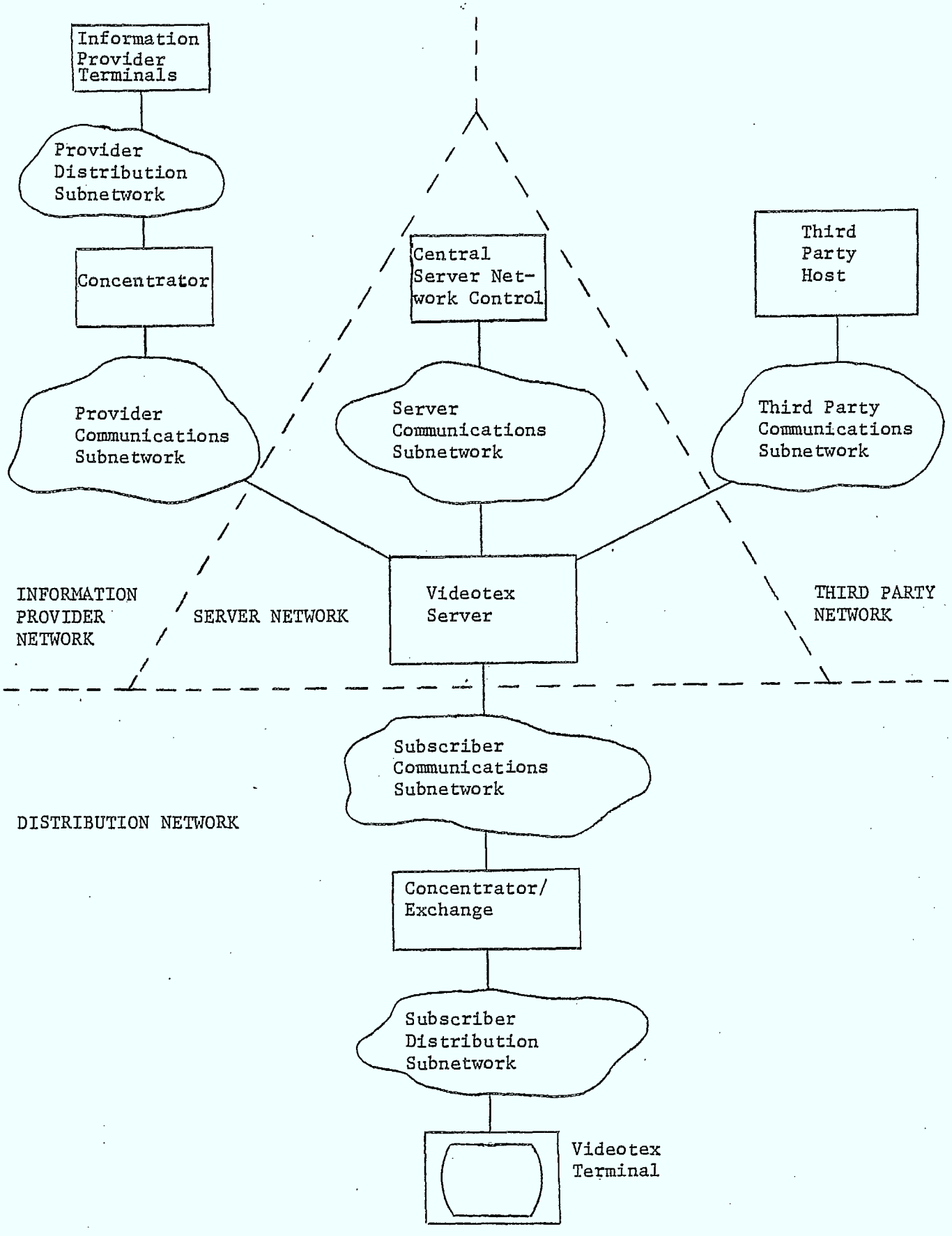


Figure 3.1 Videotex Network Architecture

will of necessity require several networked computers. The server network will provide access to the complete database for all subscribers. This may involve sharing the database or replicating it at each videotex server, and it may require sharing the load of the customers. Regardless of its implementation, though, the server network must provide a reliable, fast service.

Videotex pages can be created either on a server computer, or at an external computer designed for this function, commonly called an Information Provider computer. In either case, there is a need to connect remote terminals or computers to the server for the purpose of updating the videotex database. There may also be a need to interconnect IP computers. These functions are provided by the information provider network. Eventually, it may be desirable to allow every subscriber to also be an IP -- that is, everyone will be able to create information for the database. In this case, the information provider network will be merged with the distribution network.

The third party network exists to provide access to external computers (so-called "third parties"), which house non-Telidon databases or provide interactive services such as computer-aided instruction, airline reservations, or banking. In these cases, the videotex server does not retrieve information, but will reformat the data for subscriber terminals, correlate the data with the contents of

its own database, and so on.

In some circles, the term service provider network is now being used to refer to a network of machines that provide services of some sort to the customers. These services include the creation and updating of information, and the provision of electronic services. The term "service provider network" therefore, will be used to include both the information provider network, and the third party network.

Finally, the distribution network provides the important job of connecting the subscriber terminals (of which there may be thousands) to the individual servers in the server network. The characteristics of this network most closely affect the nature of the videotex system. For example, broadcast distribution schemes do not allow customer interaction with the server computer; this fact in turn drastically alters the kind of service provided.

The various different kinds of distribution schemes will be described in the next section. Section III.3 will discuss the performance of a specific distribution network (Omnitel), and section III.4 will propose a new distribution method based on CATV and encryption.

III.2 Some Current Distribution Schemes

In this section, some of the various distribution schemes currently being used or contemplated will be described. As well, a general description of integrated service networks will be given, as videotex systems will likely be incorporated into these networks.

Videotex systems have been divided into two main categories: one-way networks, which use television broadcast technology to distribute the information; and two-way networks, which require a subscriber-to-host connection as well as a host-to-subscriber connection to enable interactive communications. The one-way networks described here are Ceefax and Antiope, both developed in Europe. The two-way networks are further divided into different classes: telephone-based systems, namely Prestel from Britain, Vista from Canada, Captain from Japan, and Bildschirmtext from Germany; Broadcast-Out/Telephone-In networks, being considered in some field trials; Cable-Out/Telephone-In networks; two-way cable systems, namely QUBE; and Integrated Services Networks (ISNs), with the Omnitel network as an example.

III.2.1. One-way Distribution Networks

One-way videotex systems (or "teletext" systems) use television broadcast signals to distribute videotex information to subscriber terminals. A small set of videotex pages are broadcast cyclically over the air, embedded in a TV signal. The videotex terminals wait for a particular page, or set of pages to be broadcast, and then capture them for display to the user. Since these systems are not interactive, they are used only to distribute information to subscribers, and interactive services (like teleshopping or electronic mail) cannot be made available.

This section will describe two major one-way services currently in existence; Ceefax, the commerically operating system developed by the British Broadcasting Corporation, and Antiope, the system developed in France.

III.2.1.1 Ceefax

The British Broadcasting Corporation introduced Ceefax in 1972 to provide a method to caption television programs for the deaf [MORG80], but it has since been expanded to include videotex data. An experimental system began operating in 1973-74, and the commercial system began operating in November, 1976. There were a total of 150,000 Ceefax sets as of March 1981, with 10,000 new sets added every month.

The Ceefax Delivery System

Currently, the data is sent during the vertical retrace interval (the time it takes to reset the electron guns from the lower right corner to the upper left corner of the screen) in order to display both the television signal and the accompanying (digital) text. Later, the delivery system may be expanded to use one or more complete video channels, however, all of the systems and field trials described below send data only during the retrace interval.

The protocol used to transmit the data uses only five percent of the total transmission time [TANT79]. Digital data is encoded and sent on a television "data" line (a horizontal line of a TV picture), one Ceefax display row per line. There is a fixed correspondence between a byte position on the data line and a character position on the display. This means that a transmission error affects only the character in error and not the rest of the line or page.

A page header is sent immediately in front of the first row of each page, containing the page number, eleven bits of control codes, and a "time code" (explained later) [TANT79]. Each subseqent row of the page has a row header which holds the "magazine" number of which is a member (see the section on Database characteristics), and the number of the row inside the page. As a consequence of having every row of the page labelled, only those rows containing information need to be sent, and pages from different magazines may be inter-

mixed in any convenient order. All headers are Hamming-encoded.

In Britain, Ceefax data is sent during the vertical retrace at a signalling rate of 6.9375 Mbps. However, only 81% of each scan line can be used for data. Furthermore, only two lines during the vertical retrace interval contain data; if more lines are used the data may appear visible on the screen. Therefore, approximately 100 lines carrying data are transmitted per second. Since each Ceefax page contains enough data to fill 24 scan lines, only four Ceefax pages may be transmitted per second in the vertical retrace interval.

In the CBS field trial in St. Louis [OCON79], various transmission rates for Ceefax (and Antiope) were tested, the first reported test rate being 5.5 Mbps. For the trial, the Ceefax page was reduced to 20 rows with 32 characters per row. CBS was authorized to use four lines in the vertical blanking interval, but they used only two lines due to technical problems. With sixty fields broadcast per second and two lines per field, this will give slightly more capacity than in Europe as long as the same quantity of information is provided per line.

Full Channel Ceefax Delivery

It is possible to use an entire video channel solely for the transmission of Ceefax pages. Recall that each scan line carries data signalled at 6.9375 Mbps. However, since

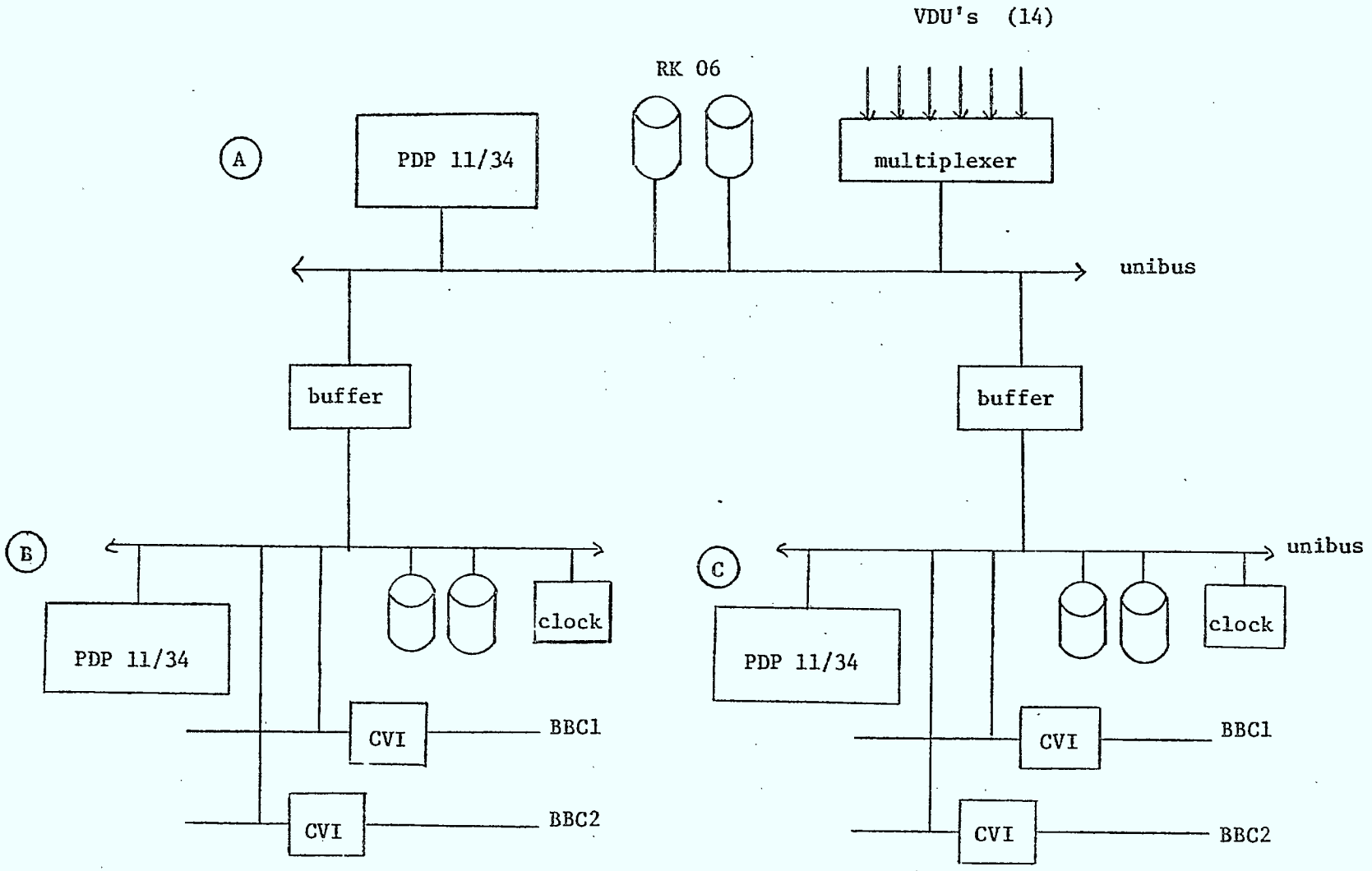
only 81% of the scan line can be used for data, the effective data rate is reduced to 5.62 Mbps. Although there are 625 scan lines in the video frame, only 575 of these are used for data. Thus, the effective data rate is further reduced to 5.2 Mbps. This allows the transmission of approximately 600 Ceefax pages per second on the video channel. Of course, the channel is incapable of transmitting video when it is used to transmit Ceefax pages in this fashion.

Ceefax Hardware Configuration

The hardware chosen for Ceefax [MORG80] consists of three PDP 11/34 machines (see figure 3.2). System A stores two replications of the database (for reliability) and handles input from up to fourteen information providers. Systems B and C are parallel systems which transmit data to the video channels, again for reliability.

Ceefax Database Characteristics

The data is organized into magazines of up to 100 pages each, with up to eight independent magazines per television channel. The number of pages per magazine has been extended by the use of minute-by-minute time slots and time-oriented pages which are automatically changed (hence the "time code" in the page header). Each page of the magazine can have up to 3200 sub-pages in this manner. There are currently about 10,000 pages of information stored in the database.



CVI= computer - video interface

Figure 3.2: Ceefax Hardware

The current Ceefax system is being broadcast on BBC1 and BBC2 with approximately 250 pages being broadcast on each channel. The average response time to retrieve a page is 12 seconds.

III.2.1.2 Antiope

Antiope, the French Teletext system, was developed shortly after Prestel was introduced, and was first demonstrated in 1976 [MART80].

The system is logically divided into two parts. The videotex services provided are called Antiope (Acquisition Numerique et Televisualisation d'Images Organizées en Pages d'Ecriture), while the data broadcast network is called Didon (Diffusion de Donnees).

Antiope Network Structure

Antiope Delivery System

The Didon system consists mainly of a packet switching protocol. Each packet consists of a header block plus a data block, and is sent along one line of the television signal (see figure 3.3 and [MART79]). The header consists of 8 bytes, and identifies the "channel" being used, the size of the data block, and a sequence number, all hamming-encoded. The channel identification defines what appears to be a virtual channel with a maximum of three channels. The size of the data block depends on the "bit signalling

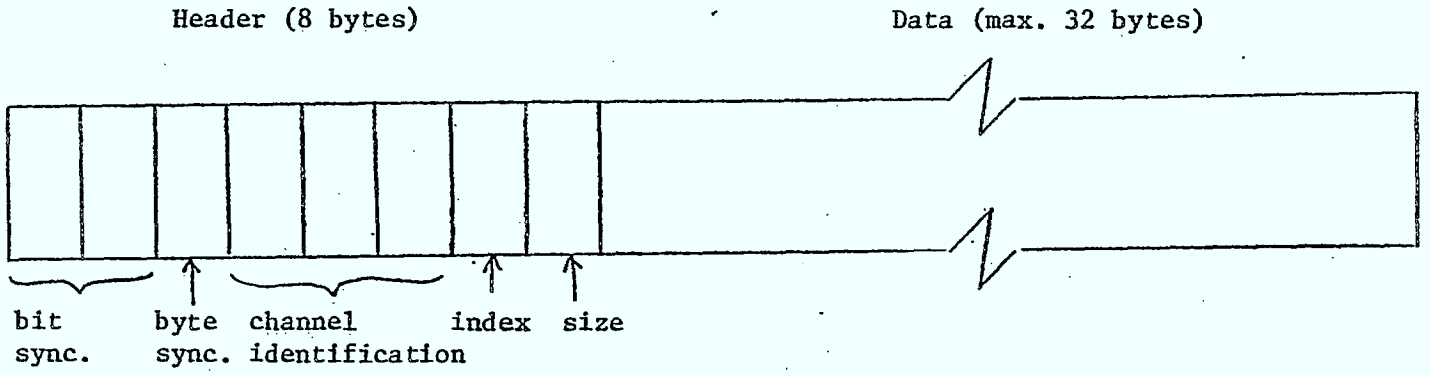


Figure 3.3: Antiope Data Packet

frequency" which is being used: in France, 32 bytes per packet; in Switzerland, 24 bytes per packet; and in North America, 20 bytes per packet.

The useful capacity of the delivery system can easily be calculated, by multiplying the number of data lines per second, by the number of bits per line. In Europe, the useful bit transmission rate is 4Mbps; in North America, the rate is 2.52 Mbps. When two lines of the vertical blanking interval are being used, the capacity of Didon is: 12,800 bps in Europe; and 9,600 bps in North America.

Experiments are now being carried out on "full-field" transmission [BERG81]. Currently, a rate of 60 lines per field is being used in a test to broadcast a cycle of approximately 600 pages.

Antiope Hardware Configuration

None given.

Antiope Database Considerations

The Antiope display has 25 rows by 40 characters. A total of 120 characters are needed to accommodate accents in French, and the ISO 2022 standard extension technique is used (character backspace overstrike).

Similar to Ceefax, the data is organized into magazines of between 50 and 300 pages apiece. At 4 Mbps, about 600 pages can be broadcast per second, or 9000 pages with a maximum delay of 15 seconds.

III.2.2 Two-way Distribution Networks

Five different schemes to implement two-way communications have been identified and four are described in this section. Section III.2.2.1 describes four systems designed for the telephone network: Prestel in Britain, Vista in Canada, Bildschirmtext in Germany, and Captain in Japan; section III.2.2.2 describes the Broadcast Out/Telephone In systems considered by the Antiope designers and TV station KSL in Salt Lake City; section III.2.2.3 describes characteristics of the Cable Out/Telephone In design; section III.2.2.4 describes the two-way cable system called QUBE in Columbus, Ohio. The fifth class of distribution networks, Integrated Services Networks (ISNs) will be describe separately in section III.2.3.

III.2.2.1 Telephone Network

The use of the telephone network for videotex came about largely due to the British Post Office, which wanted to increase telephone traffic during off-peak hours. To this end, they developed Prestel, and since then, most major videotex systems in the world have also used the telephone network.

The telephone system has inherent problems when it comes to carrying digital data. First, modems must be used at each end of the telephone system, which increases the cost. Second, telephone channels have been band-limited, limiting

digital data rates (1200 bps is the highest rate commonly found on switched lines today). Finally, telephone lines tend to be noisy, causing errors in the data.

The major advantage of the telephone system is that the network structure is already in place, with telephones in almost every home. It provides instant universal public access to the system, which is of great importance if the system is to be commercially viable.

III.2.2.1.1 Prestel

Prestel became the first commercially available videotex system in March, 1979. Since then, the Prestel technology has been sold to various countries, such as Germany, the Netherlands, Finland and Switzerland.

Prestel Network Structure

The network is composed of one or more "Update Centers" (UDCs) which serve information providing terminals (IPs), and many Information Retrieval Computers (IRCs) which house the database and communicate with the customers ([TROU80] and figure 3.4). All of the hardware is made up of machines from the GEC4082 line, a 16-bit minicomputer with moderate main memory capabilities (up to one megabyte).

The Update Centers provide facilities to edit videotex pages or to receive bulk data from intelligent terminals or remote computers. Communication with the IP terminals is mainly through 1200/75 bits per second telephone lines, with

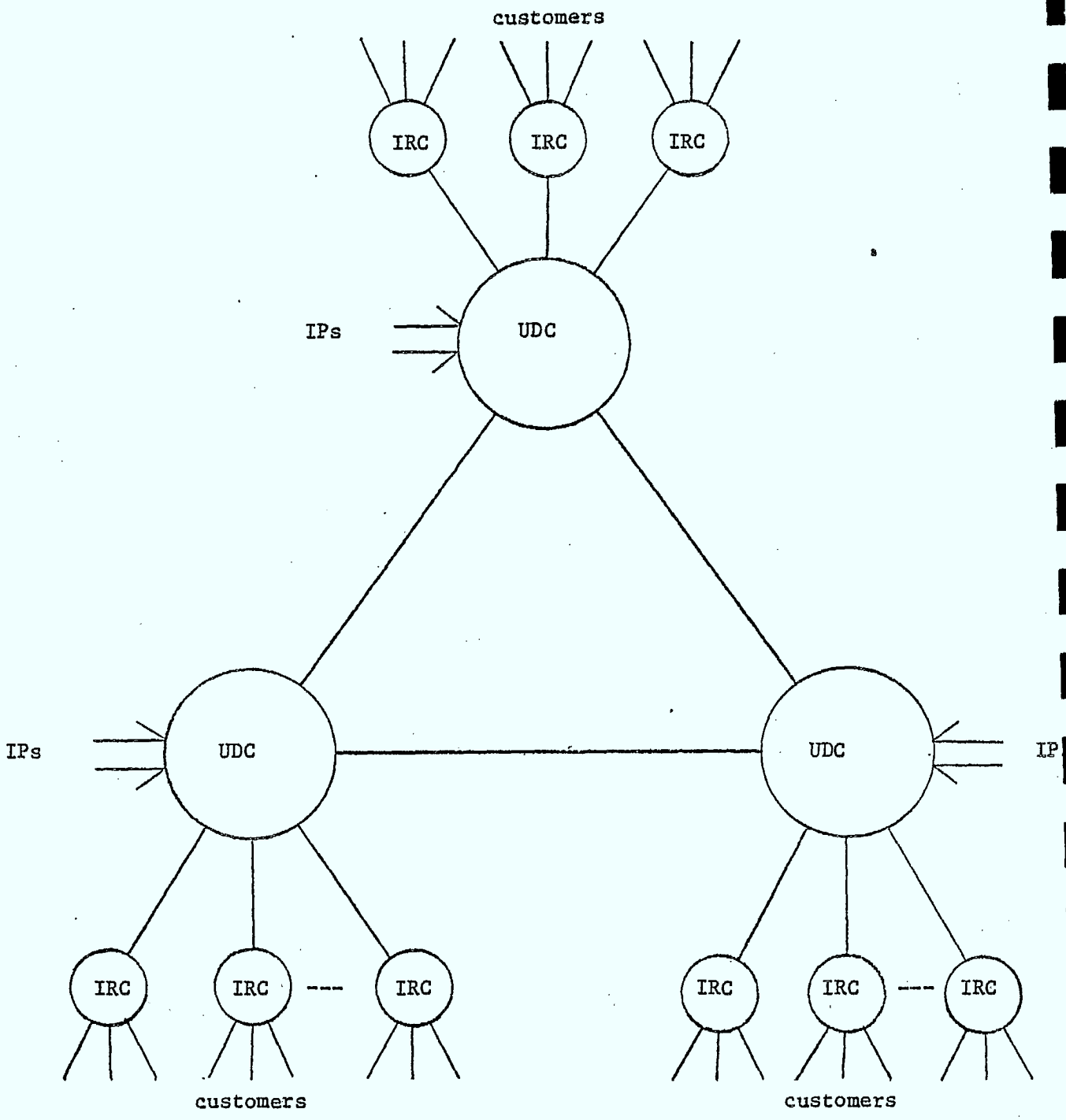


Figure 3.4: Current Prestel Network

some 300/300 or 1200/1200 bits per second lines. New or updated pages are sent to all IRCs attached to the UDC, and to all other UDCs (which update their IRCs). Communication with each IRC is via dual 4800 bits per second dedicated lines. When available, these lines will be replaced with a 2400 bits per second packet-switched communications service.

Reliability in the Update Centers is achieved by duplicating the CPU, memory, and disk drives. Recovery from machine failure is not automatic, however, since manual intervention is required to switch to the backup unit.

Each Information Retrieval Computer stores the complete database on disk. The IRC uses 1200/75 bits per second lines to communicate with user terminals, and each IRC can currently support up to 200 simultaneous users (and "several hundred" more if front-end processors are used [TROU80]).

Reliability for IRCs is achieved by routing the user to a secondary IRC if the first one it attempts to use (its primary IRC) is not operational. Also, the load from a single breakdown may be evenly distributed over a number of other IRCs, which avoids overloading any one computer.

As of February 1981, there was one UDC (in London), and a total of 18 IRCs, enabling 60 percent of the telephones in Britain to have local dial access [HOOP81]. In March 1981, there were 10,000 Prestel sets in operation, with 500 to 600 new subscribers being added per month (a subscriber may have more than one set).

The Prestel network will evolve in the future to a new network called PANDA (the Prestel Advanced Network and Database Architecture) [CLAR81]. This network will contain one or more Prestel Administrative Centers (PACs), which perform billing and other administrative functions; one or more Prestel Information Centers (PICs), which store a master copy of the database; and a number of Prestel User Centers (PUCs), which interface with customers and Information Providers. The PUCs request pages from the PIC in response to customer requests, and then store the page in a cache for some period of time. Updates are also sent from a PUC (initiated by an Information Provider) to the PIC, and only notification of the change is sent to all other PUCs. This new network structure was designed with a number of objectives, including a reduction in transmission costs, centralized billing and administration, and modularity for future expansion.

Prestel Database Characteristics

The Prestel display consists of 24 rows of 40 characters each, giving a maximum of 960 characters per page of information. The database itself currently consists of 180,000 pages of information, and it has been slowly shrinking recently, rather than expanding. The average size of a page has been found to be between 500 and 600 bytes, giving a total size of about 99 million bytes of information in the current database.

Prestel Simulation Study

When the British Post Office selected a computer for the IRCs, they performed simulation studies to determine potential bottlenecks [FEDI78b]. They wanted a maximum response time of two seconds, given twelve seconds of customer think time, 1K byte memory per user, and one millisecond processing time per request. The study found that the crucial features of the machine were disk access time and main memory size. They thought that disk speed and the disk access algorithm were significant factors.

III.2.2.1.2 Vista

Vista is a videotex system which is being developed by Bell Canada. The pilot trial started in February, 1979 and involved 25 Prestel and Telidon terminals accessing about 2000 pages of information. The current Vista field trial started in May, 1981 with nearly 500 Telidon terminals located in Toronto and Cap Rouge (near Quebec City) [TELI81].

Network Structure

The Vista designers [COST79] envision three major stages in the system's evolution:

1. A centralized system based on existing facilities.
2. A system with a distributed database, and "intelligent terminal interfaces" (see later).
3. An extension to the previous stage. Various "in-

tegrated services" such as digital telephony, messaging, and a packet-switching service will be added.

The field trial is a first stage, centralized design. It is based around a "Vista Exchange" (see figure 3.5a), a PDP 11/70, responsible for storing the database, providing terminal support, and performing various administrative services. The user terminals communicate with the computer via 1200/150 bits per second lines.

The next trial will be the second stage design (figure 3.5b). This design, called "iNET," for "intelligent network" allows a customer to connect with any of a number of third-party databases. The exchange presents a directory of databases to the user, and then routes the requests to the appropriate external machine. From the user's viewpoint, iNET disappears, and the external database performs the user interface. The iNET exchange also performs administrative functions, like billing.

In the long range plan (figure 3.5c), the Vista Exchange is logically broken down into a "node" and one or more "logical modules". The node provides a videotex meta-service: terminal support, the intelligent interface (indexing and routing), and administrative services (statistics collecting, billing, etc). The logical modules provide such services as switching, storage (of messages), and a small database. The bus structure at the bottom of the diagram

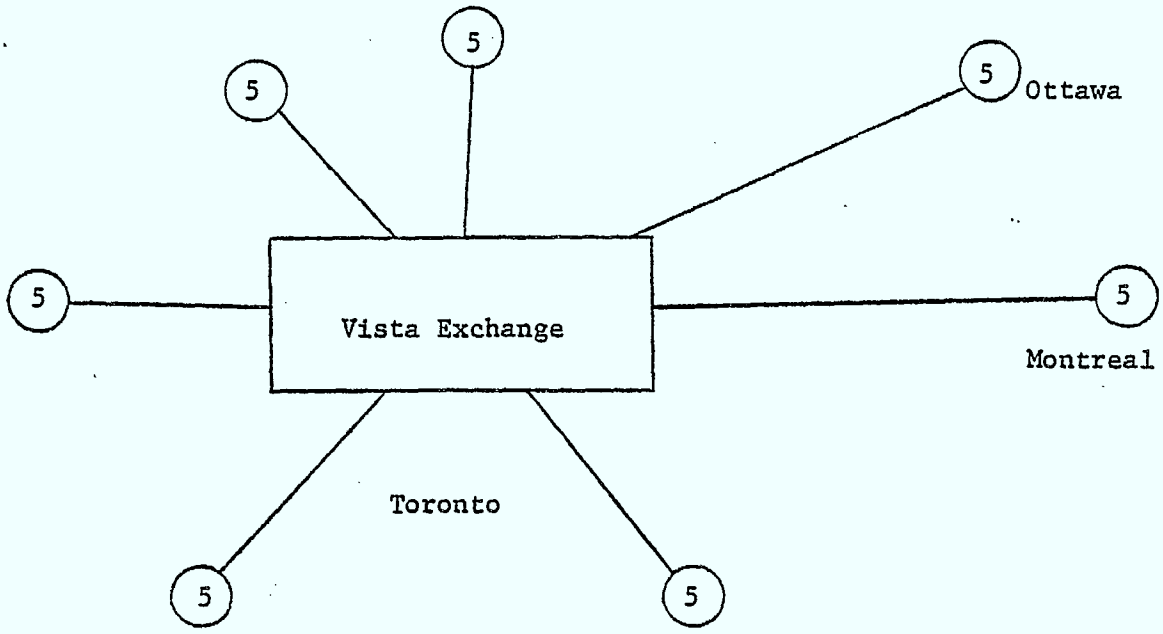


Figure 3.5a: Current VISTA Field Trial

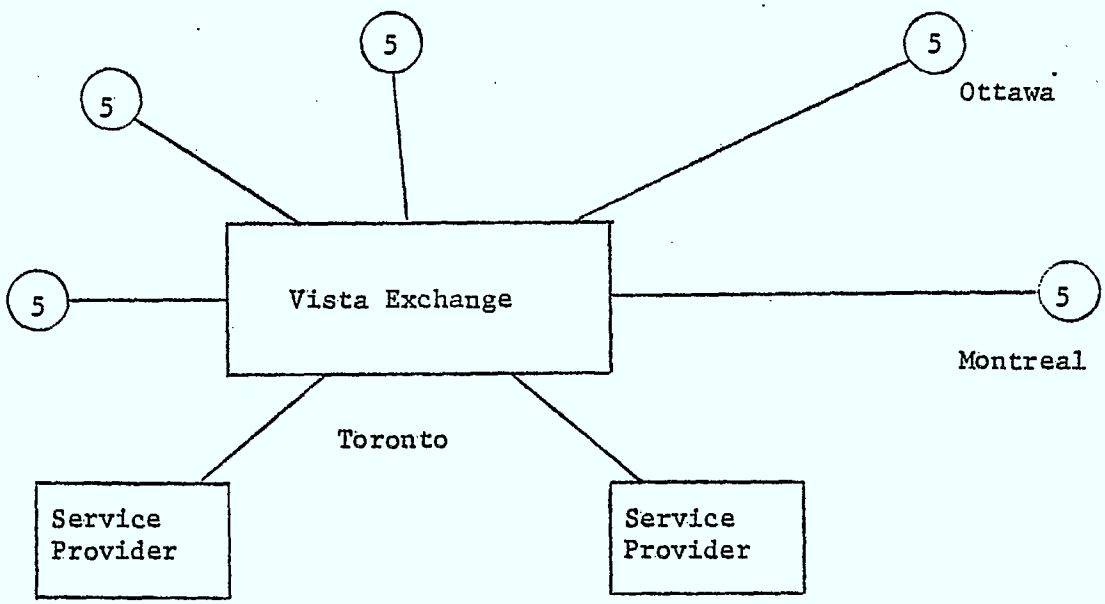


Figure 3.5b: VISTA Stage Two

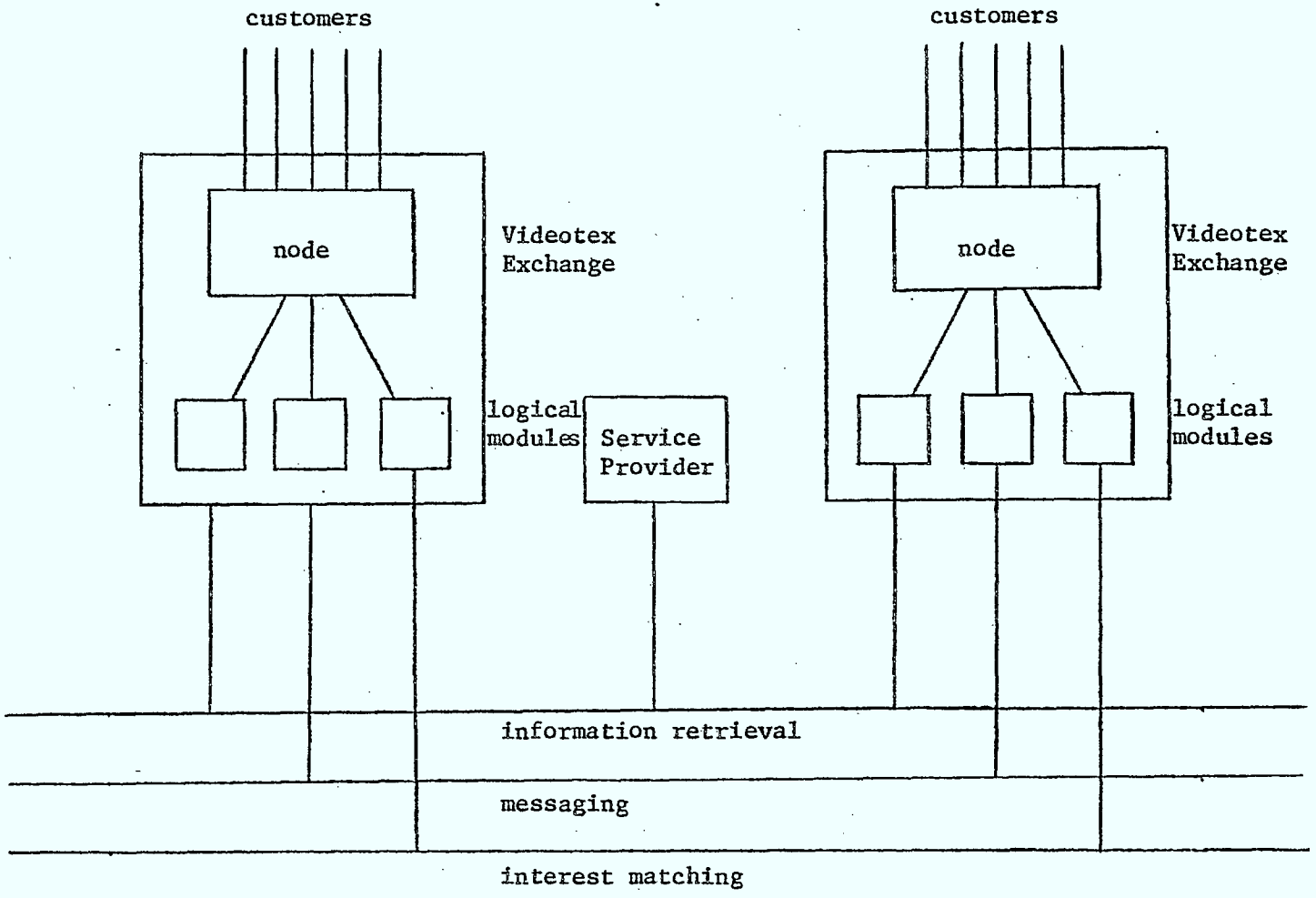


Figure 3.5c: Long Range VISTA Plan

connects similar logical modules in each exchange. This would allow network-wide services such as electronic mail, and possibly a distributed database.

Vista Database Characteristics

Telidon (or alpha-geometric) terminals use a fundamentally different method to display graphics than the Prestel or Antiope (alpha-mosaic) systems. The graphics are produced by a series of Picture Description Instructions (PDIs), such as "line x y" or "arc x y". A microprocessor in the terminal translates these commands and draws the appropriate object on the screen. The resulting graphics are high quality, limited only by the resolution of the TV monitor, and the amount of memory in the terminal. Please see CRC79 for a complete description of the PDI encoding scheme.

The cost of the high quality graphics is that more information may be needed for a page than in an alpha-mosaic system. Pages for Telidon displays range from 100 to 5000 (or more) bytes per page, but the average size of a page in the CCNG/Telidon demonstration database at the University of Waterloo is 716 bytes.

When the Vista trial opened, there were 15,000 pages in the database, with an intention of increasing that number to 75,000 by the end of the trial.

III.2.2.1.3 Captain

The Japanese videotex system, called Captain (Character And Pattern Telephone Access Information Network), was required to use over 3,000 complex Kanji characters, as well as Katakana, Hiragana and western characters, which presented some different constraints on the system [KUMA80, HARA81]. The solution to this problem was the "alpha-photographic" transmission scheme, where the picture is described pixel-by-pixel. This allows very precise graphics, at the expense of transmission speed.

Captain has already been through its first major field trial, which was held in Tokyo from December 1979 through to March 1981. A total of 1000 terminals were distributed, and close to 87,000 pages of information were created by the end of the trial. As a result of the data recovered from the trial, modifications are being made to the equipment and information retrieval methods (with the addition of keyword access), and a new trial was scheduled to start in August, 1981.

Network Structure

The Videotex Server for Captain stores and retrieves the information, and also generates the characters for the terminal. This method has the disadvantage of sending large volumes of data over the telephone lines, which results in a long delay in constructing the data on the customer's ter-

minal. As well, it tends to increase the probability of errors in the page as it is being sent. However, the method has the advantage of drawing a very precise image on the customer's terminal, which is required for Kanji, Katakana and Hiragana characters, and it allows a simpler and cheaper terminal design. (This of course is exactly the wrong tradeoff to make in an age when hardware costs are falling much faster than communications costs.)

Pages are transmitted from the central server to the customer's terminal in the form of packets at a speed of 3200 bits per second, and from the customer's terminal back to the server at 75 bits per second. To reduce the amount of transmitted data, the page is run-length encoded, and another compression technique is used called "redundancy compression". In this method, pixels for empty lines, or for the space between lines are not sent. As well, other minor techniques are used to save certain packets (see KUMA80). The end result of the compression techniques is that an average page on the system takes about 10 seconds to be displayed [KUMA80].

Database Characteristics

The Captain page consists of background graphics, and a set of characters, which can be Kanji, Kana, or alphanumeric. The screen can hold a total of 120 Kanji characters (15 characters per line, and 8 lines to a screen), or a total of 480 Kana or alphanumeric characters.

Typical pages hold around 180 characters, a mixture of Kanji and smaller characters.

The pages are stored essentially pixel-by-pixel, with the exception of the various kinds of characters, which are generated as the picture is being displayed. The pages are therefore very large, when compared with Telidon, Prestel, or Antiope pages. The exact storage format and average size of the pages have not been given in the literature.

III.2.2.1.4 Bildschirmtext

The German videotex system, called Bildschirmtext, is based on Prestel terminal technology, but has a completely different network structure.

Bildschirmtext Network Structure

In Germany, the videotex network is viewed not only as an information retrieval system, but also as a general data communications network [ZIMM80]. This has resulted in a network of videotex servers (called Btx Centers), interconnected with third-party computers (these are called "external" computers (or ECs) in Bildschirmtext) [MANT81, OTTO81]. See figure 3.5d for a network diagram. Also, since they view the data collection function of such a system to be very important, they have developed a protocol specifically for it.

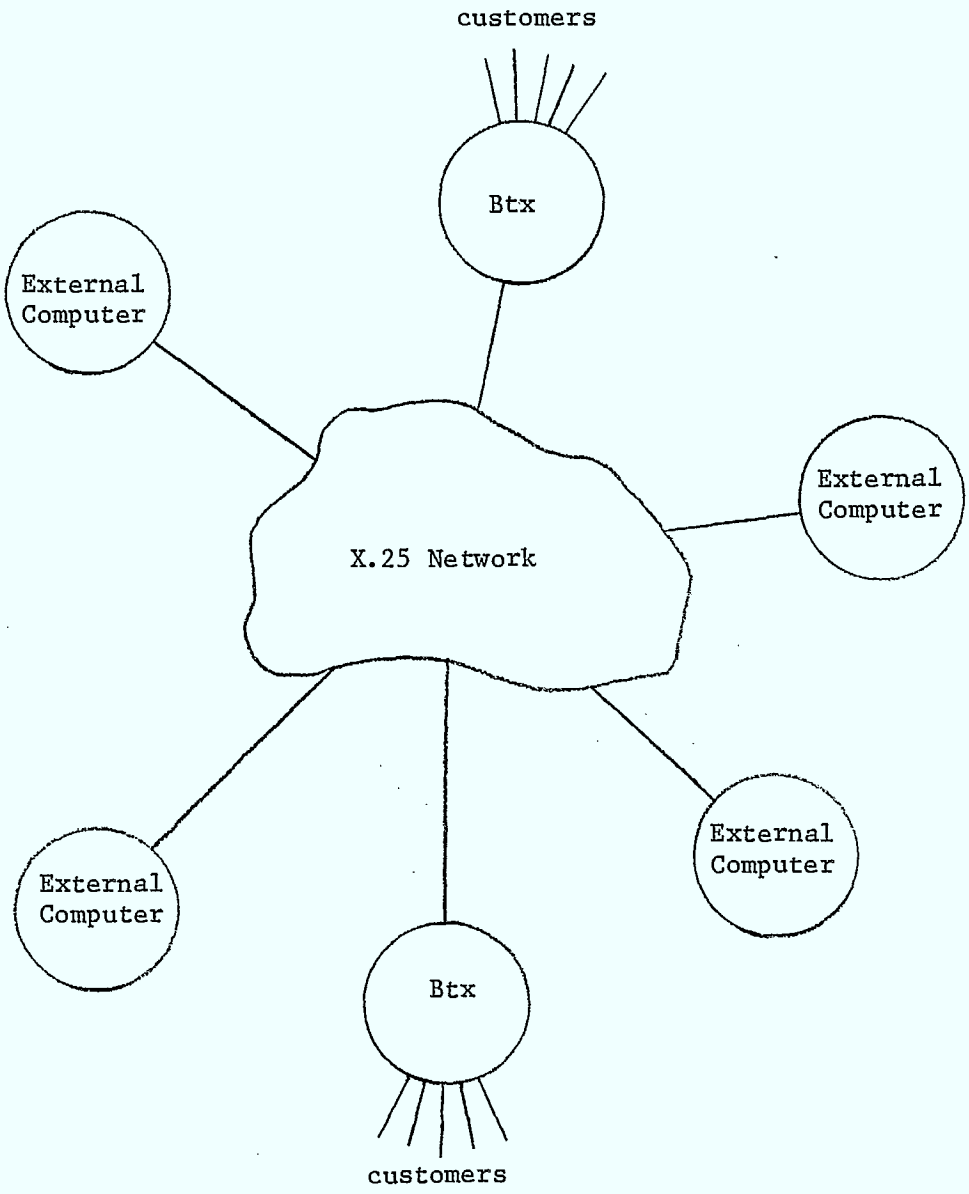


Figure 3.5d: the Bildschirmtext network

Videotex terminals connect with the Btx centers using the telephone network, or a circuit switched data system, and use either 1200/75 or 2400/2400 bits per second lines. Both full and half duplex lines are supported.

The Btx centers are interconnected with an X.25 packet switched network. These centers are specially designed microprocessor-based machines, which are responsible for terminal support and information retrieval. External computers are connected with the Btx centers via the packet switched network as well. These machines provide special functions such as reservations, order entry, computer aided instruction, and so on. The interface with the external computers has been standardized by using a form-filling protocol.

Bildschirmtext Database Considerations

The current field trial in Berlin and Dusseldorf supports between 4,000 and 5,000 customers, and a database of 75,000 pages. (See the Prestel section for more information on the page characteristics). There are two Btx centers holding the database, and about 10 external computers offering service [OTTO81].

III.2.2.2 Broadcast Out/Telephone In

This architecture is an extension of the teletext design presented earlier. In teletext systems, a set of pages is broadcast over the air cyclically (see section III.2.1), giving an average response time of one-half the time to broadcast the entire cycle. As a result, the number of pages in a cycle is kept low, typically a few hundred pages. The system's visible database can be expanded with a direct link from the customer to the videotex server, which allows an interactive videotex service to be offered. This design has been considered by the designers of Antiope [GUIL80], and field tested by TV station KSL in Salt Lake City [ROBI79].

III.2.2.2.1 KSL Field Test

The field test in Salt Lake City by TV station KSL was implemented by reserving a small set of pages in the normal cycle for interactive use via the telephone. The customer will request one of these pages, and that page will instruct him to dial a particular telephone number, which will link him with the KSL computer and enable him to request any page in the database. Once the requested page is retreived, it is broadcast in the normal cycle in place of the special page which originally carried the telephone number. Thus, anybody who displays the special interactive broadcast page may view the information displayed there, but only one per-

son may interact with the system through that page. In this field trial, the customer is limited to a short period of connect time before being automatically cut off.

III.2.2.2.2 Antiope Design

The Antiope designers have also considered this method. Their system is called DIODE (Diffusion d'Informations Obtenues par Demande), and is based on the Didon technology discussed in section III.2.1.2 [BERG81]. This system will broadcast pages from the videotex server to the subscriber, and will utilize a request channel (either through the telephone network, or cable TV) to field requests from subscribers. There will be three modes of operation of this service. First, there will be a normal teletext service. Next, there will be a "request broadcast" mode, where the terminal requests a batch of pages, and just waits for the request to be acknowledged before releasing the request channel. Finally, there will be a "controlled broadcast" mode, which is the same as the "request broadcast" mode, except that the request channel is kept until the information has been received. This design will utilize a full TV channel to broadcast the information [BERG81], and will use a terminal capable of storing a large number of pages [GUIL80].

III.2.2.3. Cable Out/Telephone In

This hybrid system has been considered mainly because videotex systems currently need high data rates from the database to the consumer (cable) and very low data rates from the consumer back to the database (telephone). This configuration has been considered by Antiope but we know of no field trials which have been implemented.

III.2.2.3.1 Cable Transmission Characteristics

Current computer communications systems on cable use either baseband transmission or some sort of frequency modulation and multiplexing, such as video signals to send their information. Currently, baseband transmission seems to be limited to around 10 Mbps, depending on the quality of the cable. This figure will probably climb as the state of technology advances. The capacity of cable transmission is considerably higher when frequency modulation (so called broadband transmission) is used. For example, the Sytek broadband local area network offers 120 channels, each of which uses a data signalling rate of 128 kbps. Thus, a total data rate of approximately 15 Mbps is achieved.

The numbers given above represent raw data rates. The effective data rate depends on the transmission protocol being used and in all cases it is smaller than the raw rate. For cable transmission, several transmission techniques have been suggested, for example: T1 streams (like the

telephone system), the Didon system, the Ceefax system, and the Mitrenet local area network system.

The T1 stream has 24 slots, each with a 64 Kbps data rate, which are time division multiplexed. This gives 1.544 Mbps per T1 stream and allows up to 4 T1 streams (a T2 stream) to be transmitted on one video channel.

The Didon system has been described in detail earlier. Its raw data rate is 5.0 Mbps in Europe and 3.528 Mbps in North America. Discounting header information and inactive broadcast lines, the effective data rate is 3.68 Mbps in Europe, and 2.33 Mbps in North America. The Ceefax system claims a raw data rate of 6.9375 Mbps and an effective rate of 5.2 Mbps (in Europe).

The Mitrenet local area network [WILL74] was designed for use on common CATV cable. The first version used a time division multiple access protocol with a raw data rate of 819.2 Kbps per channel. Packets are 256 bits long with 192 data bits, representing an effective rate of 614.4 Kbps.

III.2.2.3.2 Telephone Transmission Characteristics

The current telephone system can support a maximum of about 9600 bits per second for digital transmission on a switched connection, although a speed of 1200 bits per second is the highest speed modem which is reasonably affordable. Most current telephone-based videotex systems use 1200/75 or 1200/150 bits per second lines. Messages from

the terminal to the database are typically only a few bytes long (hence the very low speed) with one byte identifying the start of the message, one byte for the end of the message and the data between them. In the Telidon demonstration system, there are also two bytes for terminal identification.

III.2.2.3.3 Conclusions

Since no system has yet been designed specifically for a cable-out/telephone-in system, it is rather difficult to determine its attributes. The information in this section is an attempt to characterize the current transmission media and available protocols.

III.2.2.4 Two-way Cable

Two-way cable systems present perhaps the greatest potential for videotex systems, providing high data rates, potentially low noise, and good flexibility for the transmission method (ie., video, digital). There are currently three different kinds of systems proposed for two-way cable - an integrated services approach typified by Omnitel in Manitoba; a more limited television-oriented approach as in the QUBE system in Columbus, Ohio, and the "encryption-switching" approach which we have devised. The integrated services approach will be described in section III.2.3, encryption-switching is described section III.4 and the QUBE system will be described in the following section.

III.2.2.4.1 QUBE

The QUBE system was developed and introduced by Warner Cable Corporation in Columbus, Ohio. The services that they offer are a combination of normal cable TV an advanced form of Pay-TV; interactive TV, where the customer can "talk back" to the television and various polled services, such as a burglar alarm, smoke detector, medical emergency button, and a distress button. QUBE is not a videotex system , but it is included here because of its interactive nature based on the cable system. A similar system with the same range of services (except for Pay TV) has been introduced on a trial basis by London Cable-TV, the cable television operating company in London, Ontario.

Qube Delivery System

The system is based on a central computer at the cable head, which polls each customer every six seconds to detect any change in status. The customer owns a keypad with five response buttons, attached to a microprocessor, which is in turn attached to the television. The other services (burglar alarm, smoke detector, etc.) are also attached to the microprocessor. When this box is polled by the central computer, it is able to detect the status of each device attached - for example, whether the television is on, what channel is it is set to, the last response button pushed, and the status of the other services (usually an on/off setting). The central computer can identify the source of

the information (the name and address of the customer) which may be used for billing, usage statistics, and for more advanced features such as teleshopping, and electronic funds transfer. The keypad is also used for channel selection, and enables a more sophisticated form of Pay-TV. The central computer can detect when a customer tunes a Pay-TV channel, and it bills him accordingly. (Normally, Pay-TV systems charge a monthly rate).

This polling procedure essentially limits the kind of services which can be provided. Currently, the interaction is limited to multiple-choice type queries. The information retrieval service usually provided by videotex systems is not provided in QUBE, and because of the polling method used, it might be difficult to implement.

III.2.3 Integrated Services Networks.

One of the likely distribution networks that may be used for videotex systems in the future is the "Integrated Services Network", or ISN. This class of network is being designed to carry a wide range of services over a single transmission medium [DORR81, TSUK79]. The services that will be carried generally fall into one of the following classifications: voice, data, facsimile, video, and residential services (such as alarm services, and meter reading) [McDON81]. These services vary widely in their characteristics and the requirements placed on the transmission

medium. In addition, ISNs must also be designed to handle as yet undefined services, whose characteristics are not yet known.

The most prominant service on such systems now, and for some time in the future, is voice [DORR81, SKRZ81]. Voice is still being carried by analog signals in the present telephone system, although digital transmission is becoming more common. Digitally, one telephone call requires 64K bits per second, for an average duration of three minutes.

Residential services generally require a much lower data rate, on the order of tens or hundreds of bits per second [SKRZ81]. Call durations may also be short, but the calls may be more frequent than telephone calls (see section III.2.2.4.1 on QUBE).

"Data" services are varied in nature, but may include interactive terminal sessions, as well as bulk file transfer. This implies a range of data rates, call durations and frequencies. This kind of service, as well as residential service, is efficiently handled by packet switching networks, as opposed to the circuit switching network currently being used for the telephone system.

Facsimile and video require the highest data rates, with full-motion video requiring 6Mbps (or lower if compression techniques are being used) [SKRZ81]. As well, these services will require long call durations.

Thus, ISNs must support widely different service requirements, as well as different switching requirements (at least until packetized voice is universally adopted). Many people and organizations view the ISN as an evolving network, starting with the current (analog) telephone network [DORR81, McDON81, SKRZ81]. This evolution is envisioned because voice needs are dominant at the present time, and because the telephone network has been evolving towards a digital network for some time. The network will evolve to become and end-to-end digital network, with an integrated circuit and packet switching system [DORR81, TSUK79]. This evolution is currently being planned by TCTS and AT$T for the telephone systems in North America.

As an alternative, some ISNs are being developed for different transmission media. The Omnitel network [COYNE80] is an ISN being developed using coaxial cable; it will be fully described in the following section.

III.2.3.1 Omnitel

Omnitel is an integrated multiservice broadband distribution system being implemented by Coyne Associates and Interdiscom for Project Ida in Manitoba [COYNE80]. As an ISN, it is designed to support the wide range of digital services described earlier, as well as a variety of video services. The system was also designed to support twisted wire pairs, coaxial cable, or fibre, however the system

being implemented currently uses only cable. The current field trial will provide service to 100 homes in a suburb of Winnipeg.

Delivery System

The delivery system is an hierarchical structure in which the data is multiplexed (or demultiplexed) at each level from the subscriber to the switching hardware. The individual signals are then routed to the service providers. The data are carried as a video signal along a cable, similar to a normal cable TV network. See figure 3.6 for a schematic diagram of the system.

The network is controlled by a Central Computer Complex (CCC). This machine does not provide services (other than alarm reporting, administration and maintenance reporting), but rather performs switching and channel assignment for the various services. The video channels are shared between subscribers on a demand basis similar to inter-office telephony with some fixed assignment for certain services (for example, 24 channels for CATV, and 5 channels for Pay TV). The CCC communicates with various Distribution Control Terminals (DCTs) using a 9600 bits per second link. Hardware maintenance polling is distributed through the network, with the CCC receiving only reports of malfunctions.

The Distribution Control Terminal converts digital signals from the telephone switch or X.25 port to DS-1 streams (similar to T-1 streams, [DAVIES73]), and vice versa. One

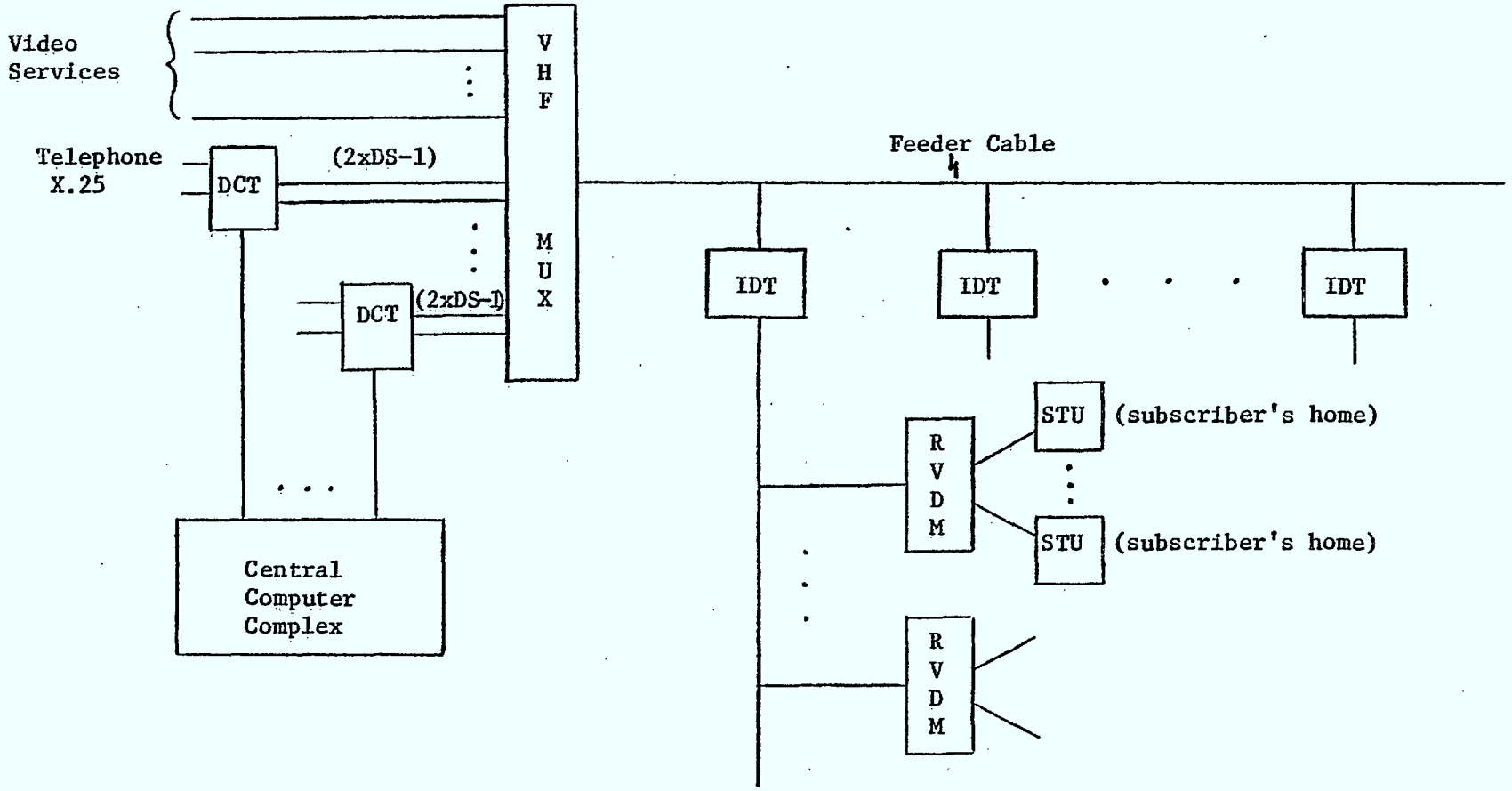


Figure 3.6: Omnitel Hardware

DCT serves 256 homes on two DS-1 streams with a capacity of 1.544 Mbps each (i.e., there are 2 X 24 time slots shared between 256 homes). Since a feeder line serves 2048 homes, then 8 DCTs are required per feeder line.

The DCT sends its signal to the VHF Mux, a multiplexor/demultiplexor for video signals. The input is taken from the DCTs, cable TV and Pay TV signals, other video services, and an information switch (for circuit switched data). The output is a set of video channels, with four DS-1 streams per video channel assigned to digital data (6 Mbps per video channel). At present there are four video channels in each direction for digital data or 16 DS-1 streams for the 2048 subscribers on the feeder line. The rest of the channels are reserved for video signals travelling downstream (from DCT to subscribers); there is no provision for upstream video.

The Intermediate Distribution Terminal (IDT) separates two DS-1 streams (1.544 Mbps each) from the feeder cable, passes them along with all video signals to the subscriber, and sends two DS-1 streams back to the DCT. The IDT also serves as a noise filter and signal regenerator. There can be up to eight IDTs per feeder cable, serving about 256 subscribers apiece.

The Remote Video and Digital Multiplexer (RVDM) performs most of the low level services for the subscriber. It controls cable and Pay TV for all its subscribers (i.e., it

does the tuning for the subscriber's TV), it does security checking, and administration functions. The RVDM also contains a mini-exchange for telephones. Communications to and from each subscriber are with 1.544 Mbps channels, one in each direction. There can be up to 32 RVDMs attached to each IDT, and each RVDM can serve up to 21 subscribers.

The Subscriber Terminal Unit (STU) is a microprocessor located in the subscriber's home. It controls cable for the TV, the phone jack system for up to four different telephone numbers per home, a videotex keypad or keyboard, and it polls alarm detectors, keypads, and meters.

III.2.3.1.2 Hardware

Currently, the CCC is an IBM Series/1 with 256K bytes main memory, a floppy disk drive of 512K bytes, and a 64M byte disk drive. This will be replaced in the future with a multiprocessor system design. The other units will probably be made up of one or more microprocessors each, depending on the complexity of their job.

III.3 Performance Study of Omnitel

This section describes a detailed study of the performance of Omnitel, described in the previous section. The term "performance" in this context, refers mainly to the end-to-end response time experienced by a customer. In addition to simple response time, however, other aspects of

Omnitel will be discussed, for example, the hardware units which act as bottlenecks will be identified.

The first step in the analysis is a general queueing model for the network, and this model is described in the first section. Both queueing theory and simulation techniques were used to determine the performance of the network, and these methods and equations are described in the next two sections. The fourth section describes the complete results from various experiments that have been performed, and the conclusions that can be drawn are discussed in the final section.

III.3.1. The Queueing Model of Omnitel

In the previous section, the hardware components which contribute to a a message's delay were described. This section will analyze the delay experienced by a digital messaged using a model consisting of a series of servers (where the message is being processed), and queues (where the message waits for processing). In later sections, this model will be "solved" using analytic and simulation techniques in order to characterize the network's behaviour. Figure 3.7 contains a diagram of the model which has been constructed. The various parameters required for the model will be discussed in the results section (section III.3.4), where

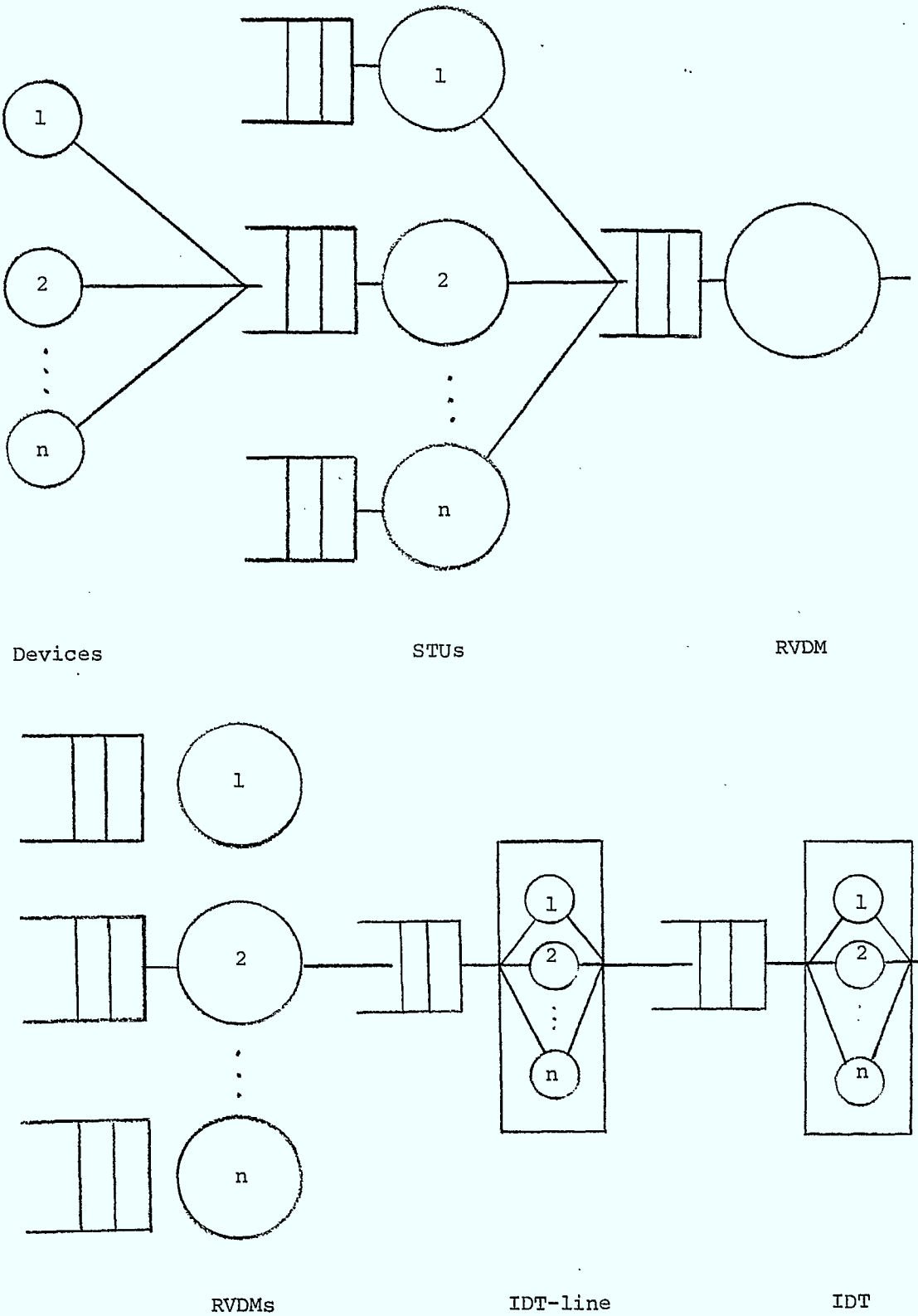


Figure 3.7: Omnitel Queuing Model

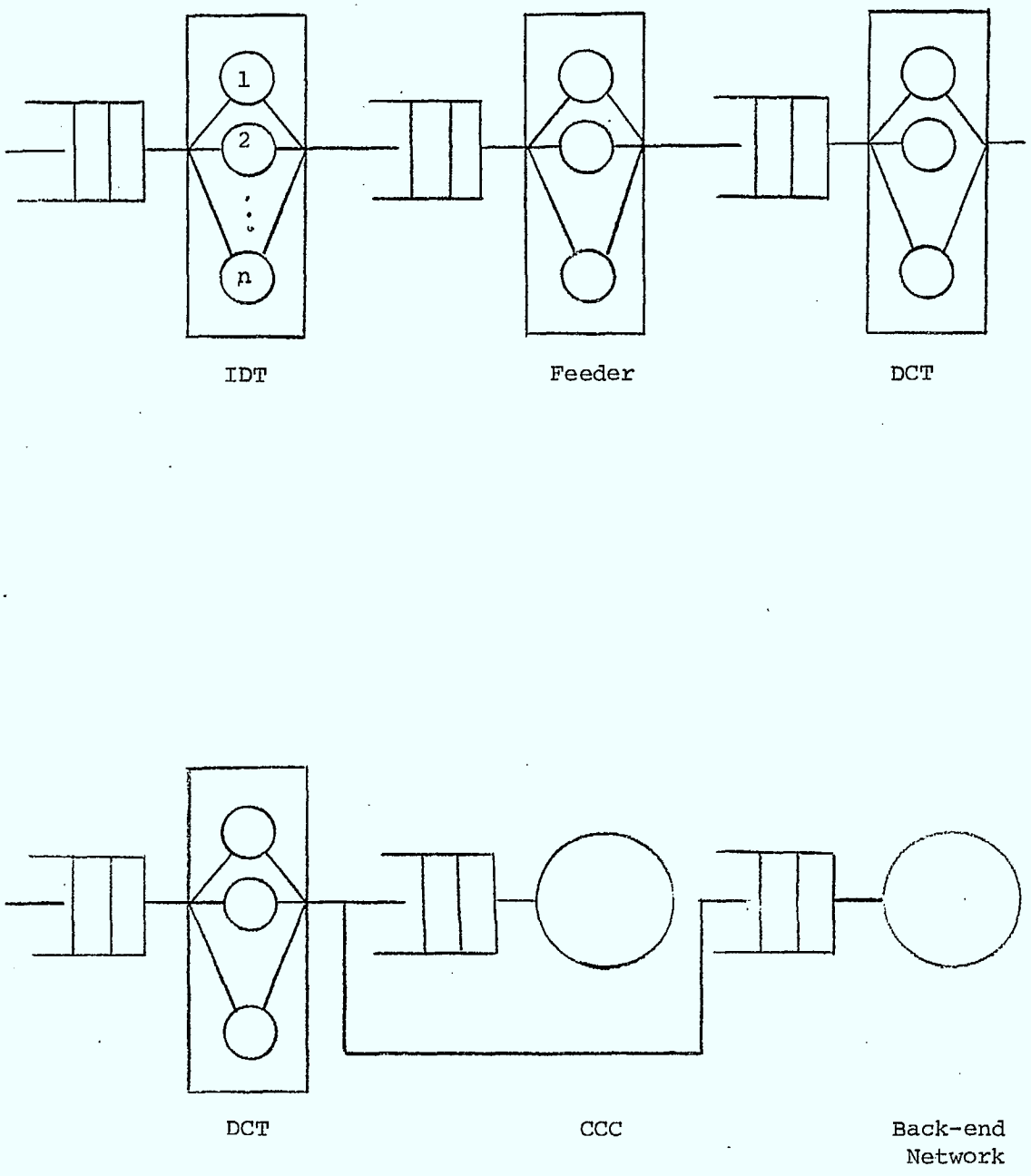


Figure 3.7: Omnitel Queuing Model

approximate values for them will be given.

To understand the performance of the network, we have separated the downstream operation (from the center to the subscriber) from the upstream operation (from the subscriber to the center). In most cases, this separation is a physical separation in the hardware (for example, separate frequency slots in the coaxial cable are used for upstream and downstream digital data). Video signals also travel through the network, but they use a dedicated, fixed portion of the bandwidth completely separated from that used by the digital messages, and so they will not interfere with digital traffic.

As discussed in the hardware section, the Subscriber Terminal Unit (STU) controls the various devices in the subscriber's home. For the purposes of modelling, we will assume that the STU receives messages from any one of its devices in a block, rather than byte-by-byte. The STU can be represented as a single server with an input queue coming from the device, and output going to the RVDM assigned to this particular STU (see figure 3.7). The input queue is not strictly first in-first out (FIFO) due to the polling regimen, but in order to simplify the implementation of the model, a FIFO queue can be assumed. This assumption is reasonably safe, especially since traffic levels for individual homes will probably be small. (Traffic from videotex terminals in a commercial environment is still

largely unknown, but most estimates are around one request every 10 seconds [FEDI78b, POWE80]; meter reading and other remote alarm services will require lower bandwidth than for videotex [SAKA80].) If only one device is active in an interval of time, then the input will be FIFO for that interval, and the exact queueing discipline is unimportant. Once the STU has the message, there will be a short delay for the message to be processed and put on the output line.

For downstream digital messages, the action of the STU is similar, although the assumption of its handling blocks or complete messages is more important because the videotex messages will be larger. This server has been modelled as a separate single server with a FIFO input queue and output lines to each device. The service time will be similar to that of the upstream STU, although a little larger if it is dealing with larger messages.

To model the upstream RVDM, a simple FIFO queue feeding a single server should be sufficient. The input will be taken from the various STUs that the RVDM controls, and the output will go to the communications line connecting the RVDM to the IDT. As with the STU, the RVDM polls the units (STUs) that are attached to it; this has been abstracted in the model to a FIFO input queue. This was done to simplify the analysis of the network later on. The delay introduced by the RVDM results from parsing the message type and reformatting the contents of the message into packets for trans-

mission.

For downstream messages, the RVDM must reassemble the message from packets in addition to parsing the message type, translating the command or message into a form readable by the display device, and sending the output to the proper output line. Again, this can be modelled as a single server with a FIFO input queue coming from the communications line, and a number of outputs to each STU that the RVDM serves. The service time will be similar to the upstream RVDM, with an additional delay caused by reassembly of the packets. Therefore, the message cannot be fully processed until all of the packets comprising it have been received. The RVDM must wait until the entire message has been received before passing it on to the STU.

The next source of delay is the communications line, in the form of a T1 Carrier (or DS-1 stream) connecting the RVDM with the IDT (the IDT-line). From the modelling viewpoint, the important characteristics of the T1 Carrier are that the system multiplexes 24 separate channels, and that each channel has a capacity of 64000 bits per second. The communications line can thus be modelled as a set of 48 parallel servers (since two DS-1 streams or T1 Carriers are used) with a single queue taking input from the RVDM's, and a single output to the IDT. The service time of each server can be calculated using the following equation:

service time = length of message (bits) / 64000 bits per second

This is the length of time that the server is occupied, but the delay that the message experiences in the communications line is only equal to the transmission time for one time slot (8 bits), or Ø.ØØØ125 seconds. After the first time-slot has been transmitted, the next unit in the hierarchy, the IDT, can begin processing. This queueing structure and service discipline are exactly duplicated for information flowing in the downstream direction.

Both upstream and downstream IDT units can be modelled as 48 parallel servers with one FIFO input queue. This is similar to the IDT-cable, since the IDT must handle two DS-1 streams (a total of 48 time-slots) simultaneously in each direction. The unit will have to operate at the same speed as the input and output lines, so the service time calculation is the same as for the IDT-line. However, the delay given to the message is equal to the processing delay associated with transferring one time-slot of data from the input line to the output line, which would be about one millisecond.

The main feeder cable connecting the IDT's with the DCT's behaves very similarly to the IDT-line. Each IDT appears to have bandwidth allocated specifically for the customers under its control (except for very light traffic conditions; see COYNE8Ø), so that the feeder cable will simply act as several independent sets of parallel servers, one set per IDT (see figure 3.7). Each set of servers is

independent of the others and so can be considered separately. These servers will have exactly the same set of characteristics as the RVDM line discussed earlier.

The Distribution Control Terminal (DCT) also must control two DS-1 streams, and as such, will look very similar to the IDT. The DCT is therefore modelled as 48 parallel servers with one FIFO input queue, and an output line to the CCC (this will be explained later). The service time is computed as with the IDT-line, and the delay time given to the message will be typically the same as for the IDT. The downstream DCT performs exactly the same function in the opposite direction, and so it is modelled in the same way as the upstream DCT.

The Central Control Computer (CCC) acts as the destination for some of the messages which are passed through the system, namely those messages for the functions that the CCC controls, and various control and maintenance messages. It is therefore inserted into the queueing model after the DCT, and it is represented as a single server, with a FIFO queue. The messages which do not use the CCC (like videotex messages) will experience no delay here but are passed directly to the next stage. Other messages, such as those for meter reading, stop at the CCC, which performs some sort of function related to the type of message it received.

Once the message has been passed through the upstream network, it enters a "back-end" network which is dependent

on the type of message (and type of service required). Telephone messages will be routed to a telephone switch, which will send the message downstream through Omnitel to a new destination, or possibly outside this network to some other network. Meter reading and alarm messages will be absorbed by the CCC, with no further action (at least, from the model's viewpoint). Videotex messages, which are the main item of interest here, will be sent through the rest of the Distribution Network, and through the Server Network to the appropriate Videotex Server. That server retrieves the requested page and sends it back through the Server Network and the Distribution Network, to Omnitel. This procedure has been grossly modelled as an infinite set of servers with no input queue, with output being sent to the downstream DCT queue. The service time is uncertain at present, but may be on the order of one-half second if this "back-end" network consists simply of a host videotex server, like most videotex networks in existence today. For videotex, the destination unit is the same STU which originated the request, and the length of the returned message will be changed to reflect the length of the videotex information page.

III.3.2 Analytical Solution

The Omnitel queueing model can be analyzed using standard queueing theory, providing that some simplifying as-

sumptions are made. This section will describe an analytic solution to the network in terms of end-to-end response time, mean waiting time per server, mean number of messages in each server (including those in the queue) and mean utilization of each server. As well, the servers will be analyzed to determine which servers are bottlenecks, and under what conditions the bottleneck changes from one server to another.

III.3.2.1 Delay Analysis

The first goal in the analysis will be to determine the end-to-end response time under a variety of parameter values for mean service times and input rates. In this context, "end-to-end response time" refers to the time it takes for a message to travel from the STU to the DCT (upstream), spend time in a back-end network, and travel back from the DCT to the STU (downstream). As well, other calculations will be made to help describe the network's behaviour (such as mean utilization and mean delay at each server).

The analytic solution is based on two simplifying assumptions concerning the arrival of messages at each server, and the amount of service required by the message at each server. First, we will assume that messages arrive at each server in a "Poisson process" -- ie. that arrivals are governed by the Poisson distribution function (see KLEIN75a, page 60 for a description of this function). Second, we will assume that a Poisson process governs when messages

leave a server. These assumptions mean that the time between the arrival of messages at a server will be taken from the exponential distribution, as will the service times of these messages (KLEIN75a, page 65 has the mathematical derivation of this property).

The Poisson process assumption has been made for two major reasons. First, the Poisson process has been shown to be a useful model of many natural processes. The classical example of a Poisson process originated in 1928, when it was shown to properly model the number of army soldiers killed by being kicked in the head by horses. As well, it has been shown to model such processes as the sequence of gamma rays emitted by a radioactive particle, and the sequence of telephone calls in a network. As well, the Poisson process often correctly models the sum of a large number of independent processes, each with a different, arbitrary statistical distribution. This property explains why this distribution models a large number of processes (eg. people making telephone calls) acting together. For these reasons, the Poisson processes will be a suitable model for messages passing through the Omnitel network. In this network, there will be a large number of subscribers attached to a single feeder cable (up to 2048), making independent decisions on when to make a request to the network. This is the situation described above, with a large number of independent processes, acting in aggregate.

The second major reason for using a Poisson process is that the mathematics involved simplifies tremendously without losing much accuracy. The exponential distribution has a number of good mathematical properties, which makes mathematics easier [KLEIN75a, page 65]. One of these properties (Burke's theorm), is used later in this section, to subdivide the network, and solve each section independently.

The model of the network described in the previous section is made up of a series of servers with queues attached acting one after another in tandem. With the assumptions described above, the queues and servers can be characterized

as M/M/1 and M/M/m queues* (where "M" means "Markovian", or the exponential distribution). The STU and RVDM may be represented as M/M/1 servers (with different mean service times) and the IDT line, IDT, feeder cable and DCT may be represented as M/M/m servers, where "m" refers to the maximum number of messages that the communications lines can transmit at one time (this will be 48 under the configuration being studied; see section III.3.4.1). The CCC will be left out of this analysis because it has no service time in the model, and the back-end network delay will be approximated by a realistic constant.

The problem of solving a network of these queues is normally quite complex, but the assumption made earlier of exponential (or Markovian) service and interarrival times makes this problem considerably easier. As discussed in KLEIN75a (page 149), Burke's theorem allows the analysis of each node in this network independently of the other nodes, providing that we have Markovian input and service time processes, and that the system is stable. Thus, each M/M/1 or M/M/m node in the network may be solved individually and the results accumulated.

The solution to an M/M/1 queue is quite well known, and may be found in KLEIN75a. The results of interest are as follows:

---

*This notation for queues is: A/B/m where "A" refers to the interarrival time distribution, "B" refers to the service time distribution, and "m" refers to the number of parallel servers being served by one queue.

utilization:	$\rho = \lambda/\mu$	[1]
mean number in system:	$\bar{N} = \rho/(1-\rho)$	[2]
mean waiting time:	$w = (\rho/\mu)/(1-\rho)$	[3]
mean response time:	$T = (1/\mu)/(1-\rho)$	[4]

where lambda represents the arrival rate (in messages per second) and mu represents mean service rate (also in messages per second). These equations will apply to the STU and RVDM, although the values for lambda and mu will differ. In particular, since the RVDM services a number of STU's (say n STU's per RVDM), then the RVDM's input rate will be n times STU's input rate.

The equations for the solution to an M/M/m queue are less well known than the M/M/1 queue, but the equations may be derived from the equilibrium set of probabilities (the equilibrium probability for a server is the probability that the server has a given number of messages in it when it is in an equilibrium state).

These probabilities are as follows (see KLEIN75a):

$$P_k = \begin{cases} p_0 \frac{(m\rho)^k}{k!} & \text{if } k \leq m \\ p_0 \frac{(\rho)^k m^m}{m!} & \text{if } k > m \end{cases}$$

$$p_0 = \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right) \right]^{-1}$$

From these equations, the following results may be derived:

$$\rho = \lambda/(\mu m) \qquad [5]$$

$$\bar{N} = \frac{p_0 m^m}{m!} \left[\frac{(m+1)\rho^{m+1}}{1-\rho} + \frac{\rho^{m+2}}{(1-\rho)^2} \right] + p_0 \left[\sum_{k=0}^{m} \frac{k(m\rho)^k}{k!} \right] \qquad [6]$$

$$W = \begin{cases} = \frac{1}{\lambda} \left[\bar{N} - \sum_{k=0}^{m} \frac{k p_0 (m\rho)^k}{k!} \right] & \text{if } k \geq m \\ = 0 & \text{if } k < m \end{cases} \qquad [7]$$

$$T = w + d \qquad [8]$$

where d is the delay that the message experiences at the server. The response time equation is slightly different than usual because the delay that a message experiences for the M/M/m servers in this network is not equal to the service time at the server (the length of time the server is

occupied), as discussed in the previous section (the model description). These equations will apply to the IDT-line, IDT, feeder cable, and DCT.

The values for utilization, mean number in the queue, mean waiting time, and response time give a good indication of the behaviour of individual servers in the network, and they should indicate whether or not any server is causing problems under a given set of parameter values. By virtue of Burke's Theorem, these values are also applicable to the network as a whole. Thus, to get the value for end-to-end response time, the following equation may be used:

$$T = \sum_i T_i + \text{back-end network delay} \qquad [9]$$

where "i" ranges over each server type, for both upstream and downstream servers.

III.3.2.2 Bottleneck Analysis

The delay analysis will show whether and where bottlenecks will occur under a given set of parameters; the network can also be analyzed to determine the parameter values required to produce bottlenecks in each server. This analysis will more graphically indicate the most vulnerable servers, and at which parameter levels they become critical.

This analysis involves the equations for mean utilization in the M/M/1 and M/M/m queues (equations 1 and 5 respectively). If a set of parameter values (service times for the various servers) results in the mean utilization of

some server in the system being 1.0 or over, then that server is working at capacity and will act as the system's bottleneck. We must determine then, which server or servers reach capacity "first."

For a given set of parameter values, with the system configured as described earlier, the utilization equations for each server are as follows:

STU: $\rho = \lambda x_{stu}$ [10]

RVDM: $\rho = 8\lambda x_{rvdm}$ [11]

IDT line: $\rho = 256\lambda (ml/64000)/48$ [12]

IDT: $\rho = 256\lambda (ml/64000 + S_{idt})/48$ [13]

Feeder: $\rho = 256\lambda (ml/64000)/48$ [14]

DCT: $\rho = 256\lambda (ml/64000 + S_{dct})/48$ [15]

where lambda represents average input rate to each STU in messages per second; x represents the average service time for the STU or RVDM; ml represents the mean message length (in bits); and S(xyz) represents the extra amount of delay that the hardware in unit xyz causes. This S-value is due to the nature of the IDT and DCT. The servers in these units are occupied by a message for a time equal to the delay it takes to process one time slot (in the T1 carrier), plus the time it takes for the complete message to pass

through, at 64,000 bits per second. The first delay is represented by "S(xyz)", and the second time is computed as "ml/64000".

These equations will be plotted (where utilization equals one) for a small set of different parameter values, and the results discussed, in section III.3.4.

Another useful analysis is to determine the set of parameters for which each server acts as the bottleneck. The above analysis tells us, for a given set of parameter values, what the safe levels for input rate (lambda) and message length (ml) will be. Since exact levels for the parameters are not known, it will be useful to know the range of parameters for which one server or another will be dominant. This is equivalent to plotting the cross-over point from one line to another in the graphs produced by the first analysis.

It can be seen from equations 10 through 15 that the utilization for the IDT and DCT will always be greater than the IDT-line and feeder cable. As well, the utilization for the RVDM will always be greater than the STU (as long as x(rvdm) > x(stu)). Thus if we compare the utilization of the RVDM to the IDT or DCT, we may determine the parameter values which cause one or the other to be dominant. By comparing equations 11 and 15, the RVDM will be dominant when the following equation is valid:

$$8\, x_{rvdm} > \frac{ml + 64000\, S_{dct}}{12000}$$

Plotting S(dct) (or S(idt)) versus x(rvdm) for various values of mean message rate, will indicate the sets of parameters which cause one of these units to dominate. This is done in section III.3.4, and the results are discussed there.

III.3.3 The Simulation of Omnitel

The analytic solution to the model provides an approximate solution, but one which is very easy and flexible to use. To get a more exact solution to the problem, a simulation program must be written. The simulation still relies on some assumptions, but the action of the servers can be more accurately modelled, and so the results should be more reliable. Simulation results, however, are very expensive (in terms of machine time) to produce, and thus, only a limited number of results have been generated. Consequently, the analytical solution will be used to explore the behaviour of the network, and the simulation results will be used to verify the analytical results. The simulation program is listed in Appendix A at the end of this report. It is written in the programming language C with the aid of the Unix operating system.

There are two kinds of messages available: videotex messages, and "external" messages. Videotex messages originate at an STU, and are very small to reflect the small keypad or keyboard requests (ranging from 2 to 16 bytes).

At the back-end network, these messages become very large, to represent the size of the returned page, and are returned to the STU which generated them. (The actual size of the returned message is a parameter in the simulation which can be varied. For the values chosen for the experiments, see section III.3.4.2.) External messages simply travel one way through the network, either from DCT to STU, or from STU to DCT. They were originally meant to simulate background traffic in the system, but by varying the size of these messages as a parameter, they can also be used to investigate the network's behaviour under a number of different circumstances.

The assumptions made in the simulation program deal mainly with the statistical distributions chosen for the various uses. These distributions are used to generate a sequence of "random" values used for service times, message interarrival times, and message lengths. These "assumptions" are really parameters chosen for a hopefully realistic pattern of messages and service times. The important distributions involved in the simulation are as follows:

Videotex Message Size:	Uniform distribution (downstream) Erlang distribution (upstream)
Videotex Interarrival Time:	Exponential distribution
"External" Message Size:	Erlang distribution
"External" Message Interarrival Time:	Exponential distribution

Service Times: generally Uniform distribution, unless the value can be exactly calculated

Back-End Network Delay: Normal distribution

These distributions were chosen for a number of reasons. The Uniform distribution was used for the downstream videotex message size, and for the service times that could not be calculated otherwise. This distribution gives each value in a given range the exact same probability of being chosen. In both cases where it was used, the range of possible values was quite small, and it was felt that this distribution provided a good enough estimate, without costing too much machine time. (The uniform distribution is the easiest to calculate.) The Erlang distribution was used to generate values for videotex page sizes (upstream messages) and "external" messages. This distribution rises to a peak (near the mean value), and then decends slowly to give a long tail. This is very similar to the distribution of videotex pages. It has been observed that the majority of Telidon pages are between 100 and 1000 bytes long, with a small number of larger pages, and a few very large pages. Finally, the Exponential distribution was used for generating the arrival of messages to the network. This was chosen for essentially the same reason as for the analytical solution -- Poisson processes model these situations quite well. All of these distributions, however, are parameters that can be changed if more realistic estimates are

available. For this evaluation, though, these distributions were not changed.

Both the analytical and simulation assumptions have been summarized in Table 3.6 at the end of section III.3.5. The major reason why the simulation model can be considered to be more accurate is that each individual server in the network is operated as closely as possible to the way it operates in the real network. Even though the exponential distribution is used to generate messages to the network, the message is processed through the network according to the exact queuing discipline of the various servers. The analytical model considers the average behaviour of all messages which pass through the network and it must make assumptions about how messages pass from server to server (exponential interarrival times). Since the simulation does not make this assumption, its results should be more realistic.

The exact details and results of the simulation runs will be discussed in the following section.

III.3.4 Presentation of Results

This section will describe the analytical calculations and simulation experiments which have been performed. First, the default parameters to both the analytic calculations and the simulation program are given. These represent approximate values for the various service times in the

network, and are used as a base for all of the calculations presented later. The second section will describe the experiments which have been done, and the third section will present the results. Finally, the results from the simulation program and the analytic calculations will be discussed and compared.

III.3.4.1 Model Parameters.

In order to perform the experiments, numeric values must be given for service times in the various queues, and the configuration of the system should be described. In Table 3.1, estimates of "service time" and "delay time" for each type of queue is detailed. The "service time" is the length of time that the server is occupied by a message, and the "delay time" is the amount of delay that the message is given. These values have not been detailed in the literature [COYNE8Ø], so the values given in Table 3.1 are best realistic estimates of the parameters, except for the delay time and service time for the communications channels (the IDT-line and Feeder), which can be exactly calculated.

For the single server queues (the STU and RVDM), the service time and delay time are the same because in the model, the whole message is received and processed before being passed on. For the multiple server queues (the IDT line, IDT, feeder, and DCT), the service and delay times are different due to a different queueing discipline. For these

queues, the progress of the message is delayed only by the transmission or processing delay of the first portion of the message (ie. the first time-slot in the time division multiplexing scheme). As soon as that time slot is processed, the next server in the network has access to the start of the message, and begins processing. The original server however, is still processing (or transmitting) the message, and it will continue to do so for the duration of the service time.

The configuration used for most of the experiments consists of a single DCT, servicing two DS-1 streams on the main cable (two DS-1 streams represent 48 time-slots, or 48 parallel servers in the model). These streams connect to one IDT which is loaded with 32 RVDM's, each supporting 9 STU's. On each STU (ie. in each subscriber's home), there is one videotex terminal, and possibly other devices. This configuration represents a heavily loaded system according to COYNE80, but does not load the system as fully as the addressability of the network allows.

Table 3.1: Mean Service and Delay Times (default)

Queue Type	Service Time	Delay Time
STU (upstream)	1.0 milliseconds	1.0 milliseconds
STU (downstream)	10.0 milliseconds	10.0 milliseconds
RVDM (upstream)	10.0 milliseconds	10.0 milliseconds
RVDM (downstream)	10.0 milliseconds	10.0 milliseconds
IDT line	msg length (bits)/64000 bps	0.125 milliseconds
IDT	msg length (bits)/64000 bps	1.0 milliseconds
Feeder	msg length (bits)/64000 bps	0.125 milliseconds
DCT	msg length (bits)/64000 bps	1.0 milliseconds
CCC	zero	zero
Back end network (videotex)	0.5 seconds	---

III.3.4.2 Description of the Experiments

Most of the analysis of the network has been done using the analytical approach, due to the length of time that it takes to run the simulation program. The analytical equations are a much more flexible tool to use in exploring the behaviour of the network. The simulation however, is more accurate, and so it must be run to ensure the validity of the conclusions. The bulk of the experiments then, are done only with the analytic equations and the simulation program is used to duplicate a selected number of the analytical results.

The experiments are conducted by varying one or more parameters in the model, and calculating the end-to-end delay of a message travelling through the network. These parameters consist of those described in the previous section (service and delay times, and the configuration of the network), as well as the mean message length in the network, and the traffic rate (measured as the number of messages generated every second by each STU).

In the first experiment, the delay times for all servers in the model (except the communications channels) were set to the same value, and their values simultaneously increased. The communications channels are not included here because their service and delay times are exactly known, and so they are not parameters which can be changed (unless the

configuration of the network is changed). This experiment was designed to quickly find out the server most likely to present congestion problems, and it uses only videotex messages. The mean size of videotex messages upstream from the STU to the back-end network was 90 bits, and the returned size of the videotex messages was 13000 bits. Both values represent high estimates of the actual values, so that if any congestion is likely to happen under normal videotex operation, it should show up here. The upstream videotex messages are generated at a rate of one message every ten seconds from each videotex terminal. Both simulation and analytical results were calculated, and the results can be found in figures 3.8a and 3.8b in the following section.

The next experiment shows the effect of running a constant flow of videotex messages (at a rate of one request every ten seconds per videotex terminal), and an increasing flow of "external" messages. The mean length of "external" messages here is 5000 bits, and the rate of messages varies from zero to as high as necessary to reach congestion in the network. This experiment was run both analytically and by simulation, and the results may be found in figures 3.9a and 3.9b (also in the next section).

The third experiment was designed to investigate the performance characteristics of each server in the network. For each unit in turn (except for the communications channels), the delay time was varied, keeping all other service

times at their default values. The length of the end-to-end delay was plotted for a series of curves, each with a different mean message length. For the communications channels, the number of parallel servers in the network was varied from 24 to 48 to 96, in order to investigate the effect of the channel capacity on end-to-end delay. The end-to-end delay values for these parameters were also plotted for a series of mean message lengths.

Finally, the fourth experiment alters the configuration of the network to see how the RVDM behaves when it is loaded with a different number of STUs. The total number of STUs on the network is kept roughly constant, and the number of RVDMs and STUs connected to each RVDM is changed. In an additional experiment, the total number of STUs is increased to test the network at a heavier load.

III.3.4.3 Presentation and Discussion of Results

III.3.4.3.1 Delay Analysis

The first experiment effectively increased the delay times for all servers (except the communications channels) to the point where one (or more) of them became congested, and thus limited the throughput of the network. Figures 3.8a and 3.8b are the simulation and analytic results from this experiment, respectively. From these figures, it can be seen that the network does not become congested (under a

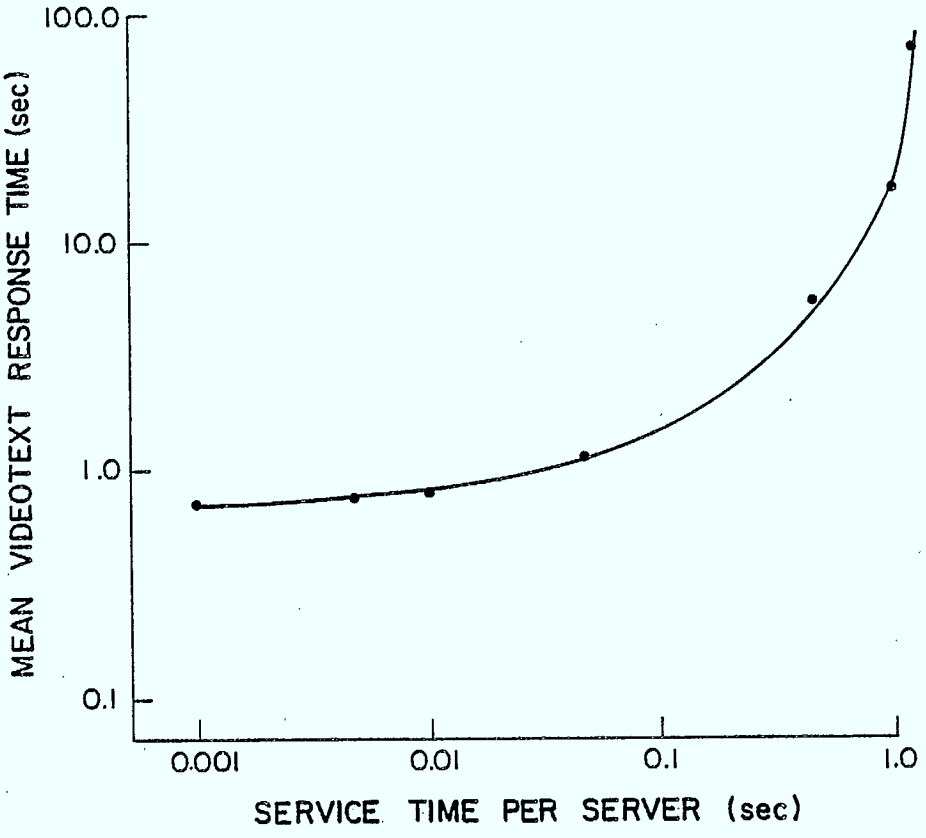


Figure 3.8a: Experiment 1, Simulation results

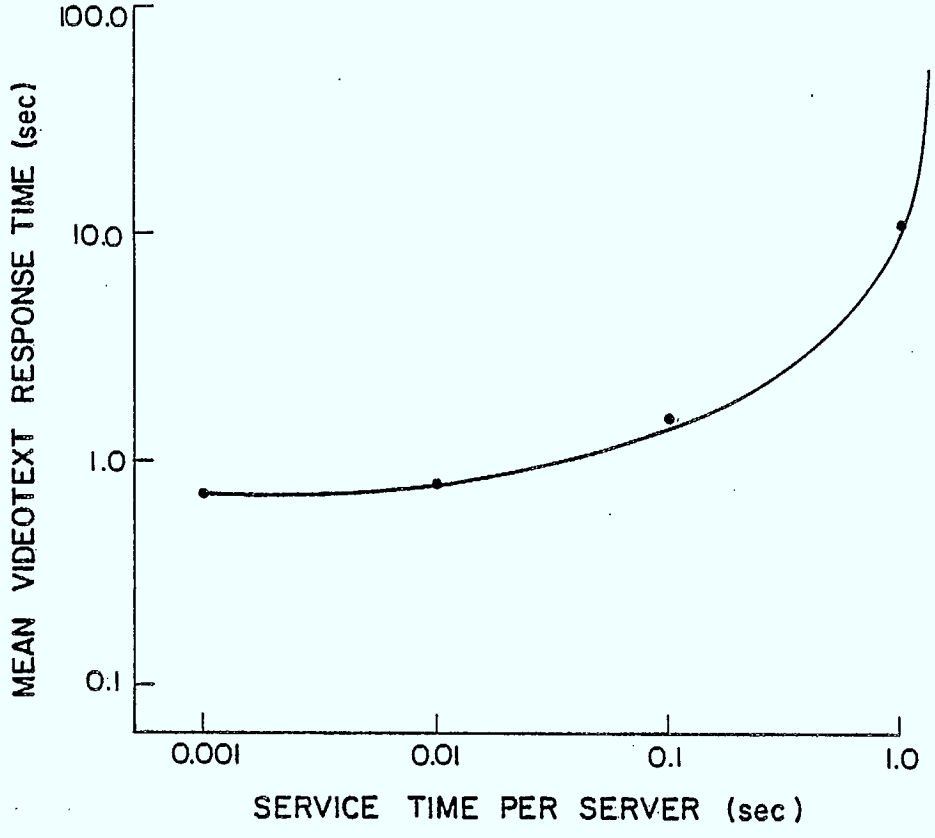


Figure 3.8b: Experiment 1, Analytical results

normal videotex load) until the delay time for each server has reached an unreasonably high value (1.25 seconds). This fact alone indicates that for reasonable service and delay times (like the default values in Table 3.1) the network will not be congested under normal videotex load conditions (one message every ten seconds per videotex terminal).

To see which server or servers actually caused the bottleneck, a table of utilization values for each server at the saturation point (1.25 seconds delay time) has been compiled for the analytical results:

Table 3.2: Utilization at the Saturation Point in Experiment 1

	Server	Utilization
a) Upstream	DCT	0.6674
	Feeder	0.0008
	IDT	0.6674
	IDT-line	0.0008
	RVDM	1.0000
	STU	0.1250
b) Downstream	DCT	0.7750
	Feeder	0.1083
	IDT	0.7750
	IDT-line	0.1083
	RVDM	1.0000
	STU	0.1250

This table shows that, at the saturation point, it is the RVDM, both upstream and downstream which is fully utilized, and thus is causing the congestion of the network. Thus, the first indication is that the RVDM is the most critical piece of hardware on the system.

The second experiment runs a constant flow of videotex requests, and increases the flow of background or "external" messages to simulate an increasing traffic level; the analytic and simulation results may be found in figures 3.9a and 3.9b respectively. Both graphs show a decrease in the end-to-end response time between a mean input rate of Ø.1 and about 1.Ø messages per second per STU. This is due to the fact that small messages are transmitted in less time than large messages, and that the mean message size decreases when external messages are added. At Ø.1 messages per second per STU, there are only videotex messages in the network, with a mean size of 13ØØØ bits. As external messages are introduced with a mean size of 5ØØØ bits, the mean size of messages in the network drops, as does the mean end-to-end delay. After a period of time, the traffic in the network causes queuing delays, and the mean delay starts to increase again.

In this experiment, the end-to-end response time of videotex messages can been seen from the first plotted point on figures 3.9a and 3.9b, when there are no external messages in the network. The actual value for the delay is

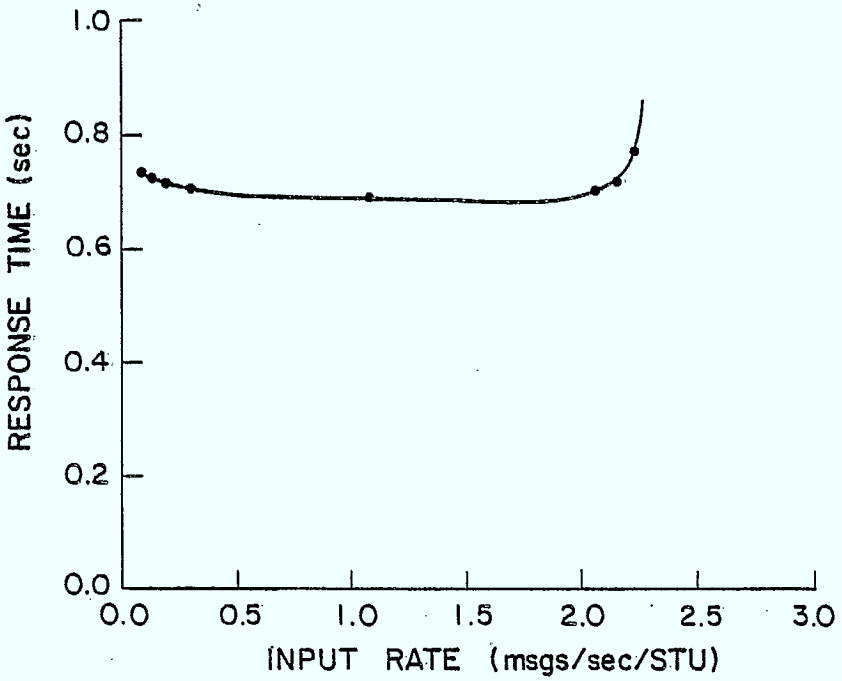


Figure 3.9a: Experiment 2, Simulation results

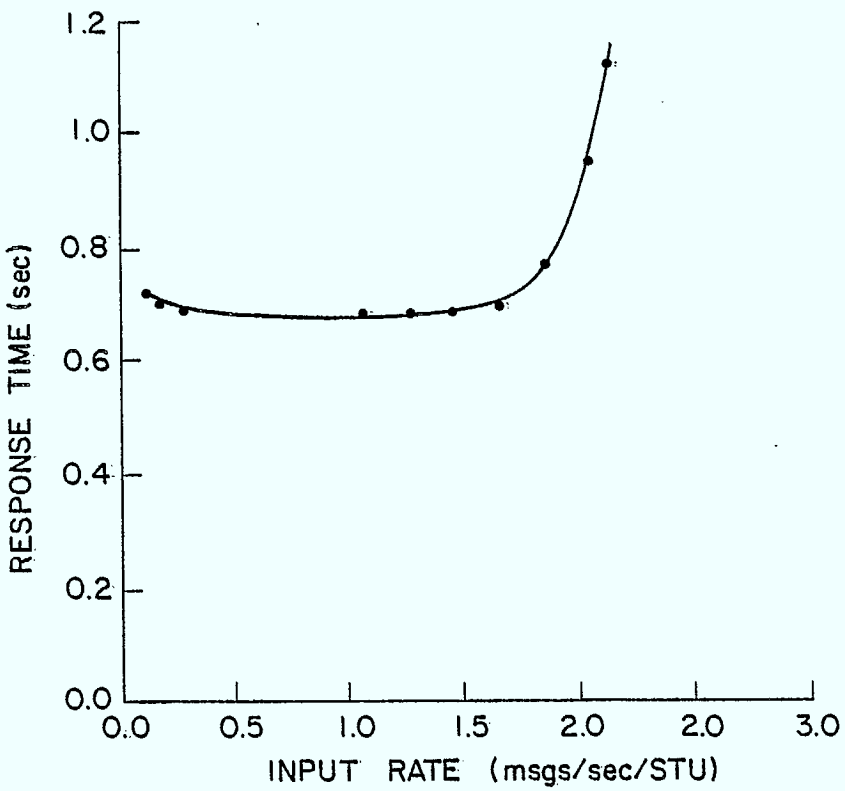


Figure 3.9b: Experiment 2, Analytical results

Ø.7398 seconds in the simulation experiment and Ø.7317 seconds in the analytic calculations. These figures consist of Ø.5 seconds delay in the back-end network, about Ø.Ø15 seconds upstream delay, and about Ø.22Ø seconds downstream delay. The difference between upstream and downstream delay is again due to the difference in mean message sizes travelling in the two directions.

This experiment also shows different behaviour at the saturation point (about 2.21 messages per second per STU) than in the first experiment, as Table 3.3 illustrates:

Table 3.3: Utilization at the Saturation Point in Experiment 2

	Server	Utilization
a) Upstream	DCT	Ø.8915
	Feeder	Ø.8797
	IDT	Ø.8915
	IDT-line	Ø.8797
	RVDM	Ø.1768
	STU	Ø.ØØ22
b) Downstream	DCT	Ø.999Ø
	Feeder	Ø.9873
	IDT	Ø.999Ø
	IDT-line	Ø.9873
	RVDM	Ø.1768
	STU	Ø.ØØ22

In this case, the DCT and IDT are causing the bottleneck, rather than the RVDM as in the first experiment. The bottleneck analysis discussed later will indicate the conditions under which the bottleneck changes.

The results from the main series of experiments, where the service times of the various hardware units was increased, and the capacity of the communications channels was altered, may be found in figures 3.10 through 3.14. The analytical results from the RVDM experiment are shown in figure 3.10. A series of simulation runs has been performed with the same parameters as shown in figure 3.10b (the default parameters as shown in Table 3.1), and results from these experiments may be found in figure 3.11. Figure 3.12 contains plots from the IDT and DCT experiment, figure 3.13 shows the STU results, and figure 3.14 plots the results from the communications channels experiment.

In figure 3.10, the effect of increasing the RVDM service time can be readily seen. Figure 3.10a shows the RVDM with a service time of 0.005. At this level, the RVDM serves as a bottleneck only when the mean message length in the network is very small (less than 480 bits per message); at larger message sizes, the DCT or IDT becomes the bottleneck. As the RVDM service time increases (in figures 3.10b, c, and d), the RVDM increasing becomes a more prominant bottleneck, and the maximum message rate that the network can handle (measured in messages per second per STU)

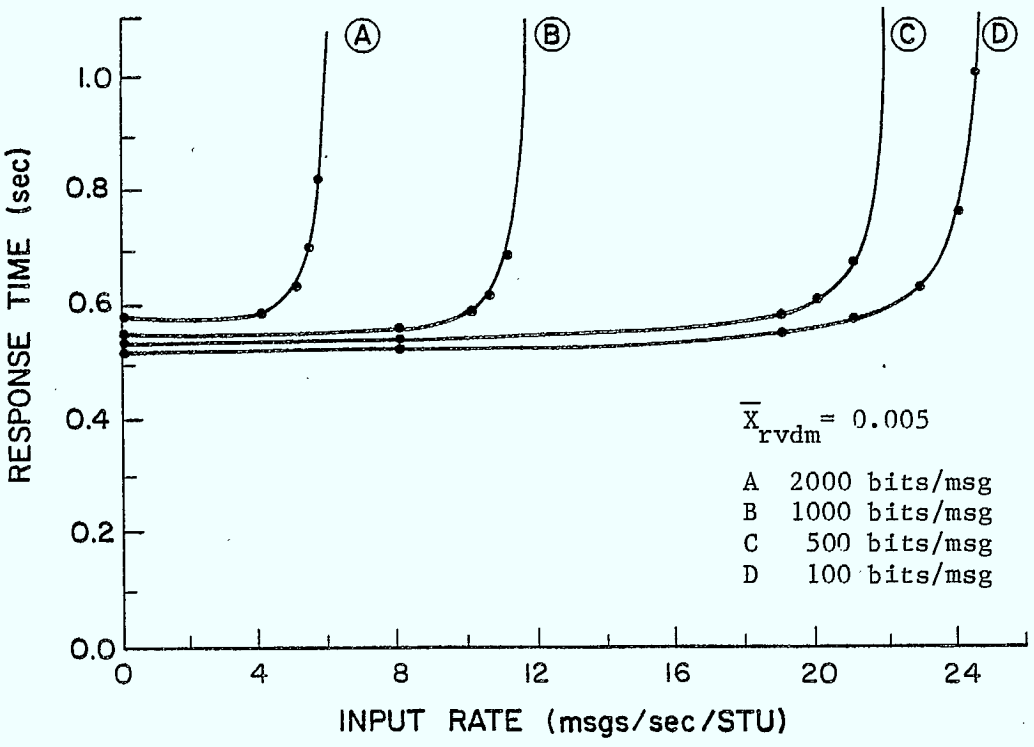


Figure 3.10a: Experiment 3, RVDM results

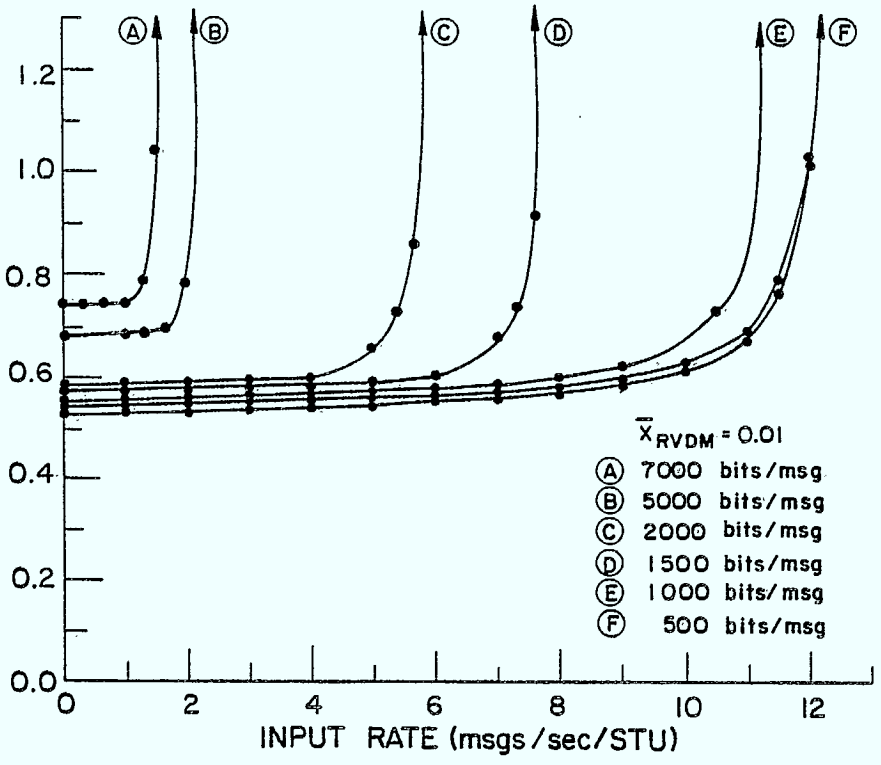


fig. 3.10b RVDM, DEFAULT PARAMETERS

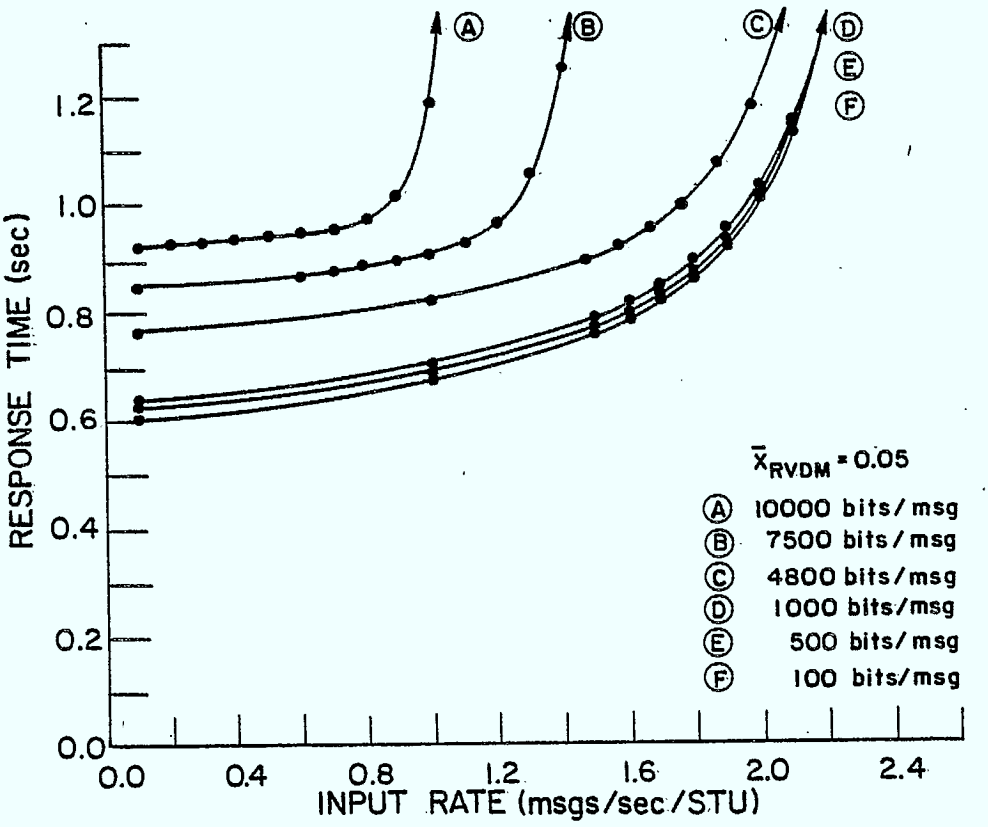


fig. 3.10c RVDM RESULTS

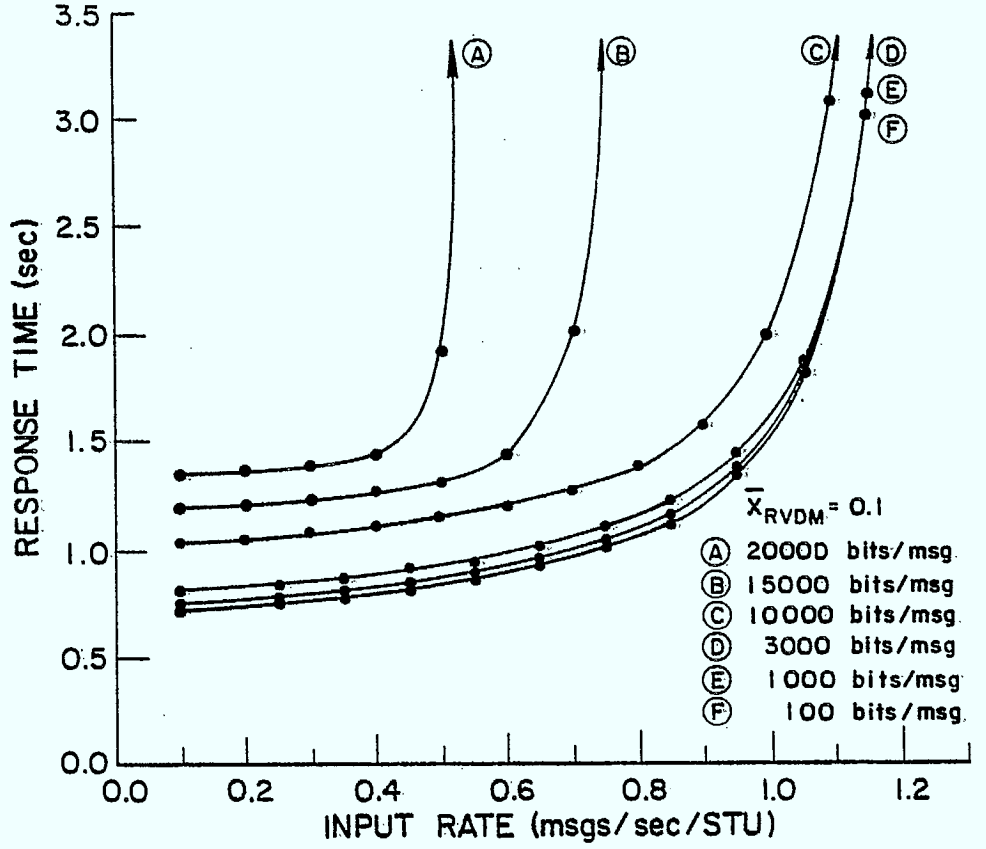


fig. 3.10d RVDM RESULTS

drops in proportion to the increase in service time. In fact, if the RVDM service time is as high as Ø.1 seconds per message (figure 3.1Ød), then the network can only accept a maximum of 1.25 messages per second from each STU, which may constrain the system if high bandwidth digital services are offered.

Figure 3.11 shows results from the computer simulation, run for the same parameters as for figure 3.1Øb (the default parameters as in Table 3.1). This figure shows essentially the same behaviour of the network as shown in figure 3.1Øb, the analytical results. The values given at low traffic levels in both figures are nearly identical, and both figures show the network becoming saturated at the same levels. There is a difference in how the response time behaves when traffic levels approach the saturation point, however. These differences are discussed more fully in section III.3.4.4.

The results from the IDT and DCT experiments are shown in figure 3.12. These two hardware units have identical forms in the network model (see section III.3.2.1), and so they have been analyzed in the same experiment. The four sections of the figure (3.12a to 3.12d) show the IDT/DCT unit with a delay time (the processing time for one time-slot of data) of Ø.ØØØ5 seconds, Ø.ØØ5 seconds, Ø.Ø1 seconds, and Ø.1 seconds respectively. Figure 3.1Øb shows results with the default parameters, where the IDT/DCT delay

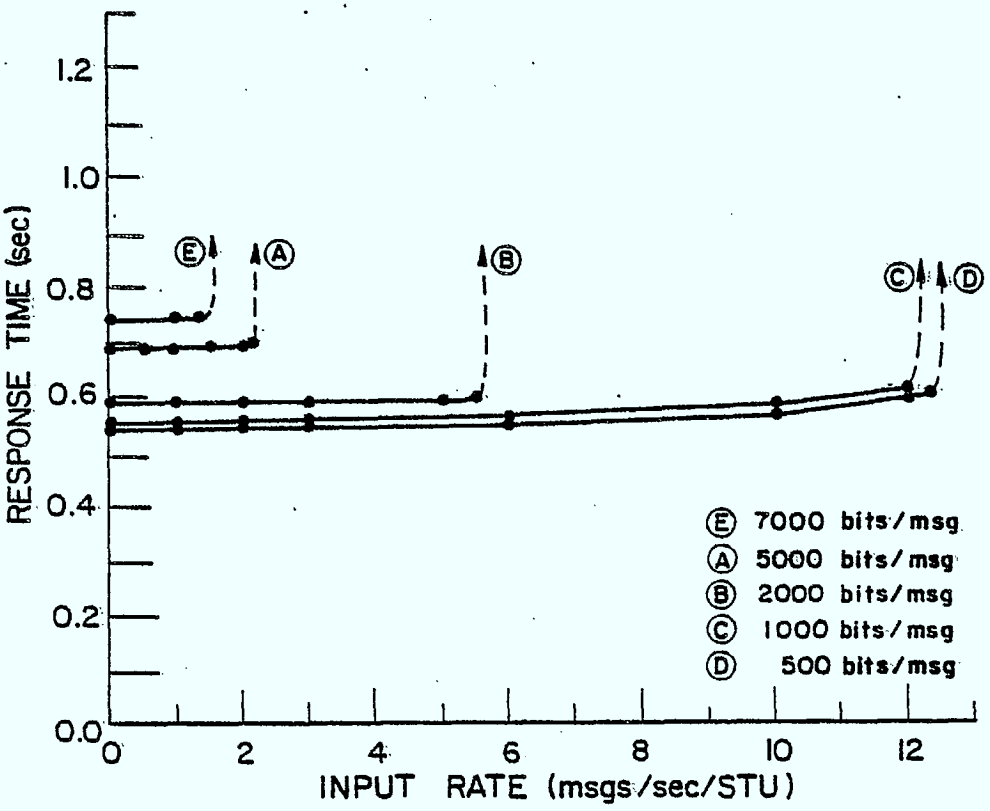


fig. 3.11 SIMULATION RESULTS, DEFAULT PARAMETERS

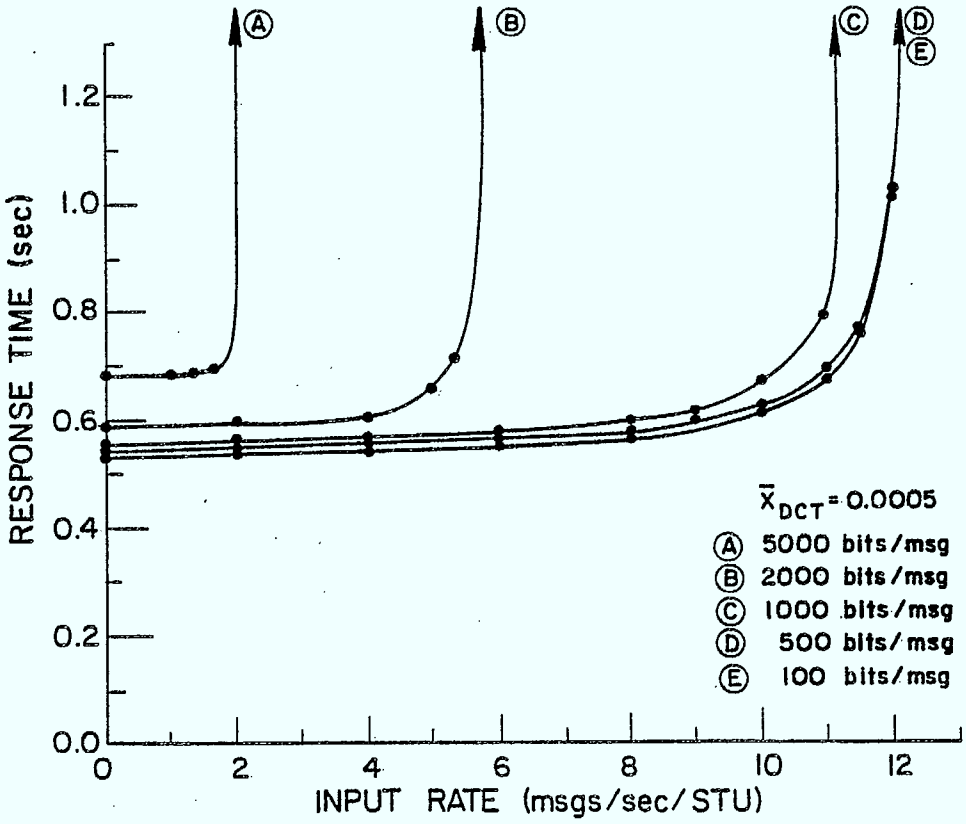


fig. 3.12 a DCT RESULTS

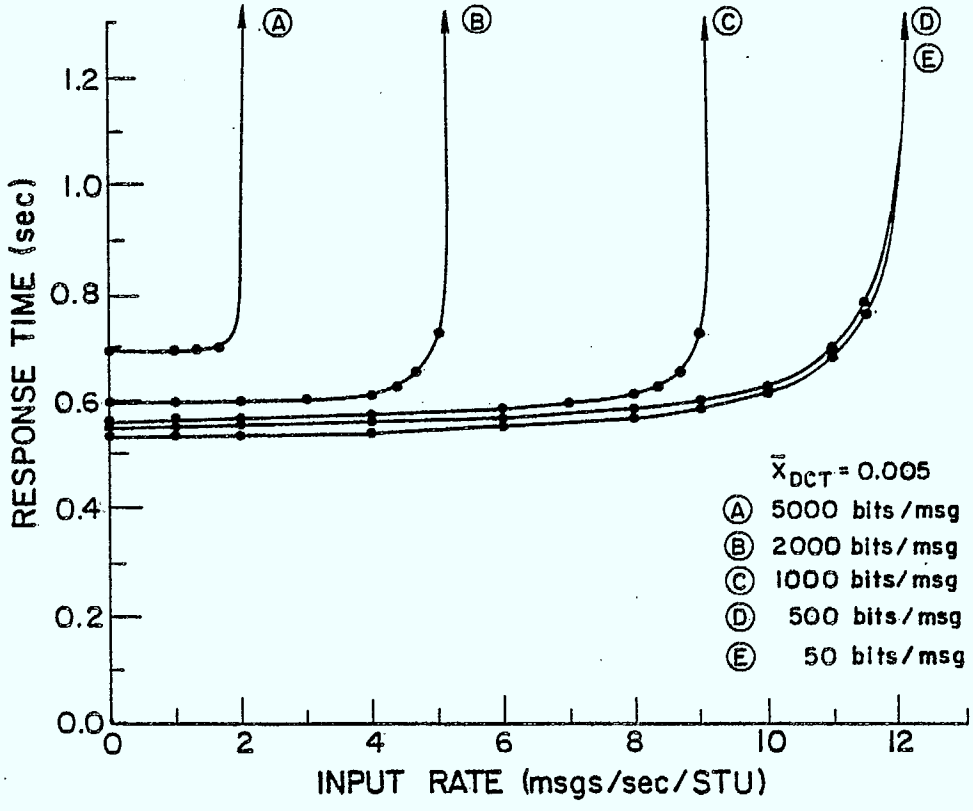


fig. 3.12 b DCT RESULTS

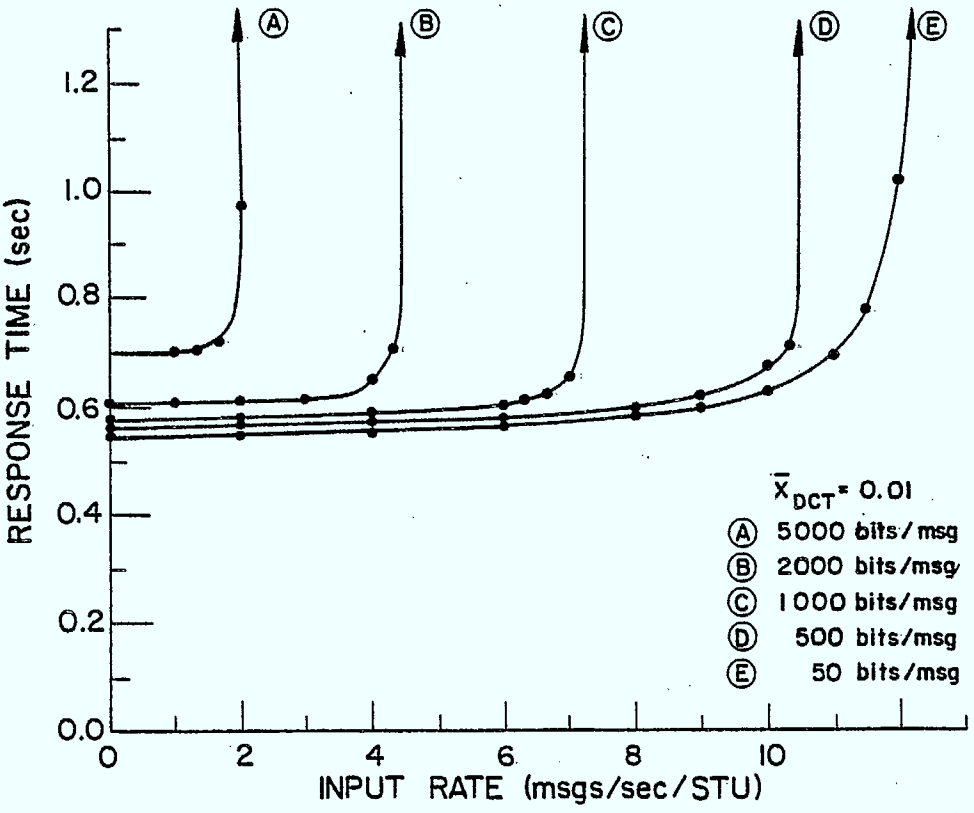


fig. 3.12 c DCT RESULTS

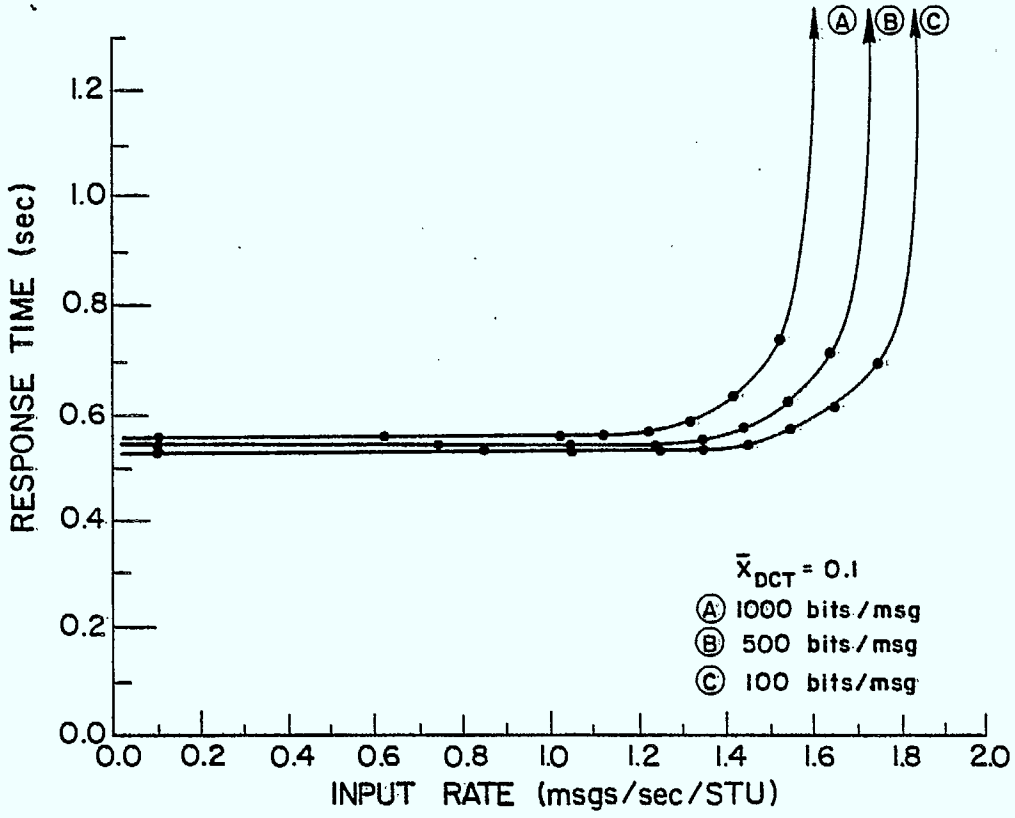


fig. 3.12 d DCT RESULTS

time is Ø.ØØ1 seconds. In all graphs, the overall response time at low input traffic rates is roughly the same. Unless the IDT/DCT delay time is very large, its value does not add a significant amount to the response time. Near the saturation point, queueing delays are much more important. As the IDT/DCT delay time increases, the saturation point for most of the plotted curves moves to the left (ie. the saturation point occurs at a lower rate of traffic). This indicates that the IDT and/or the DCT unit is acting as the network's bottleneck for these parameter values. This does not take place when the average size of messages in the system is very small -- around 5Ø to 1ØØØ bits per message (depending on the other parameter values). For these curves, the limiting factor is the RVDM, as in figure 3.1Ø. When the delay time of the IDT/DCT is very large (figure 3.12d), the IDT or DCT unit becomes a very serious bottleneck, limiting throughput to less than 2 messages per second per STU.

The STU experimental results are shown in figure 3.13. Figures 3.13a through 3.13c show the results when the STU has a service time of Ø.Ø1 seconds, Ø.Ø8 seconds, and Ø.1 seconds respectively. Again, figure 3.1Øb shows the default parameter results, where the STU service time is Ø.ØØ1 seconds (upstream) and Ø.Ø1 seconds (downstream). Since the volume of traffic being transmitted through the STU is eight times less than through the RVDM (in this configuration), the STU must have a very large service time before it

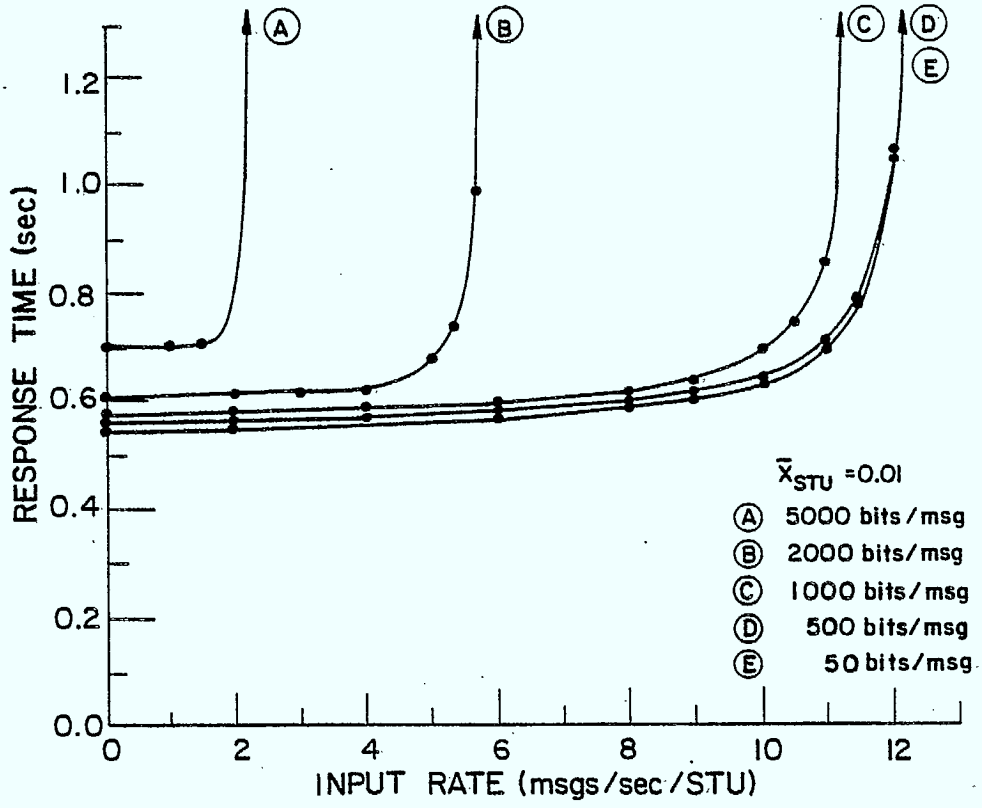


fig. 3.13a STU RESULTS

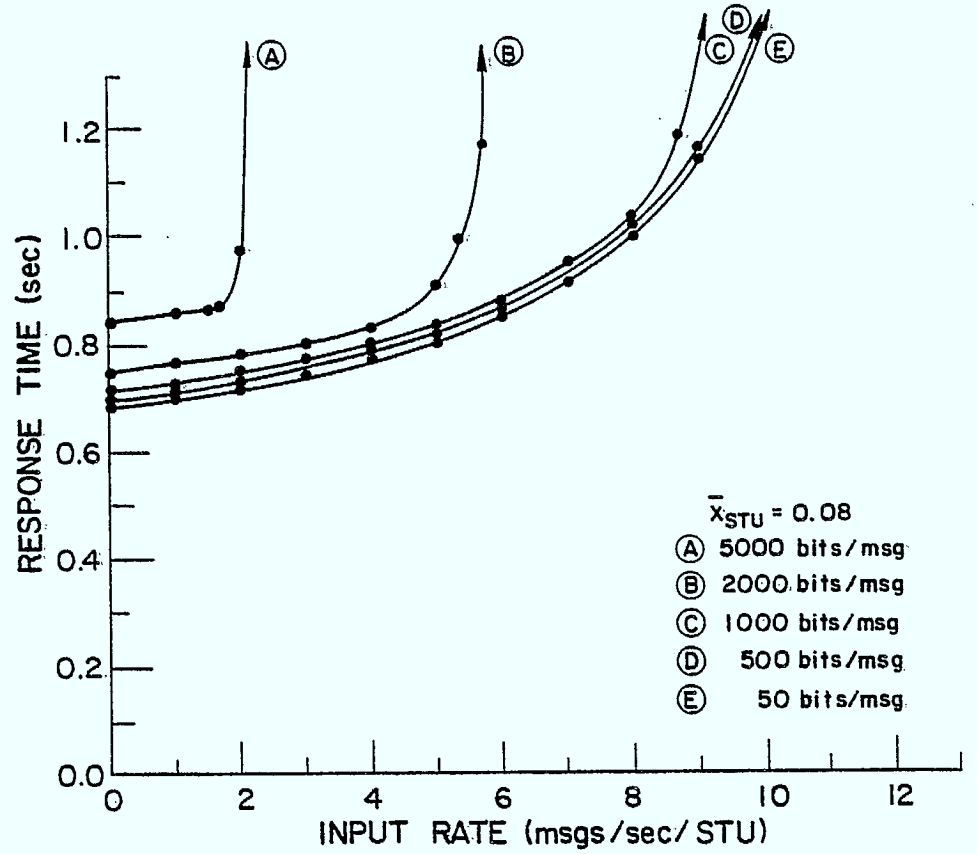


fig. 3.13b STU RESULTS

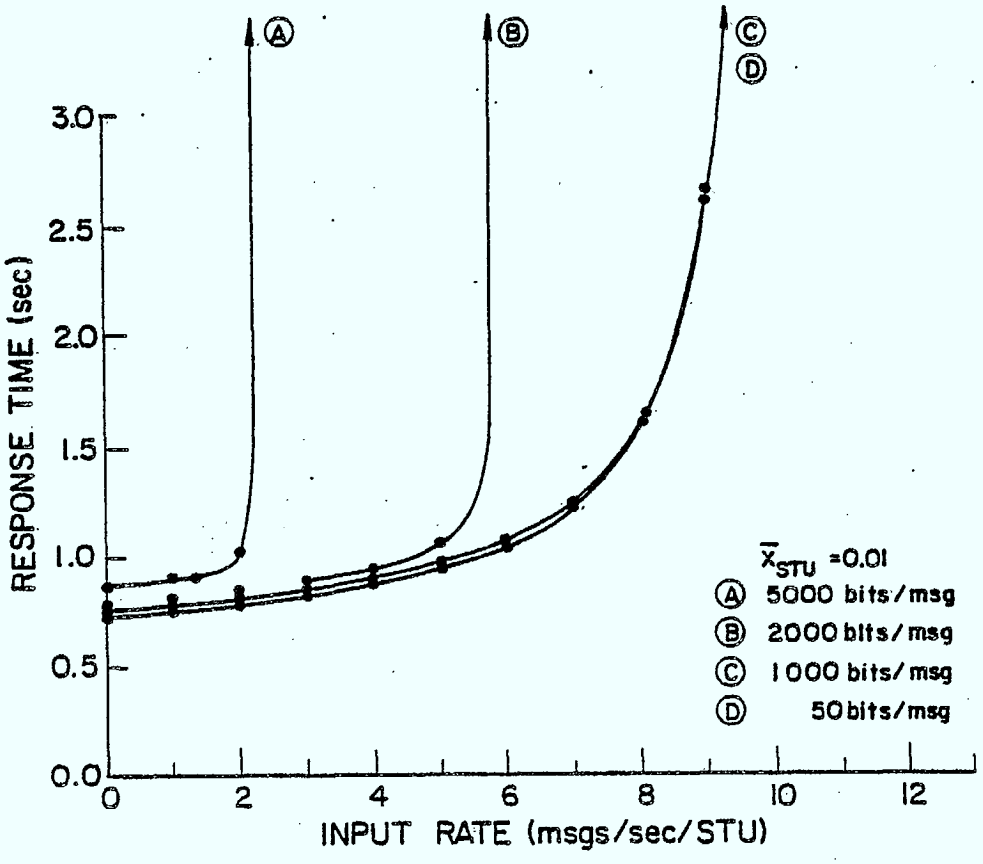


fig. 3.13c STU RESULTS

becomes a more serious problem than the RVDM. When the STU service time is the same as the RVDM (figure 3.13a), there is no significant effect of response time or saturation point. However, when the STU service time is eight or more times the RVDM, the STU replaces the RVDM as a bottleneck in the curves with small mean message sizes (50 and 1000 bits per message). As in figure 3.10, the IDT or DCT units are still bottlenecks for large mean message sizes. With very large service times, the curves also change shape, starting at higher initial values, and increasing more quickly due to a more serious queueing problem at the STU.

Figure 3.14 contains experimental results from varying the capacity of the communications channels. The capacity is measured by the number of 64K bit channels which are available for both upstream and downstream transmission. Figure 3.14a shows 24 channels (one DS-1 stream), figure 3.14b shows the default parameters, 48 channels (two DS-1 streams), and figure 3.14c has 96 channels (four DS-1 streams). When the capacity of the communications channel is altered, the capacity of the IDT and DCT must also be altered in the model, since these units process each channel in parallel. This means that altering the channel capacity has a dramatic effect on the system performance. In figure 3.14a, with 24 channels, the network is severly limited, except for very small messages. If the mean message size is under 500 bits per message, then the network can handle

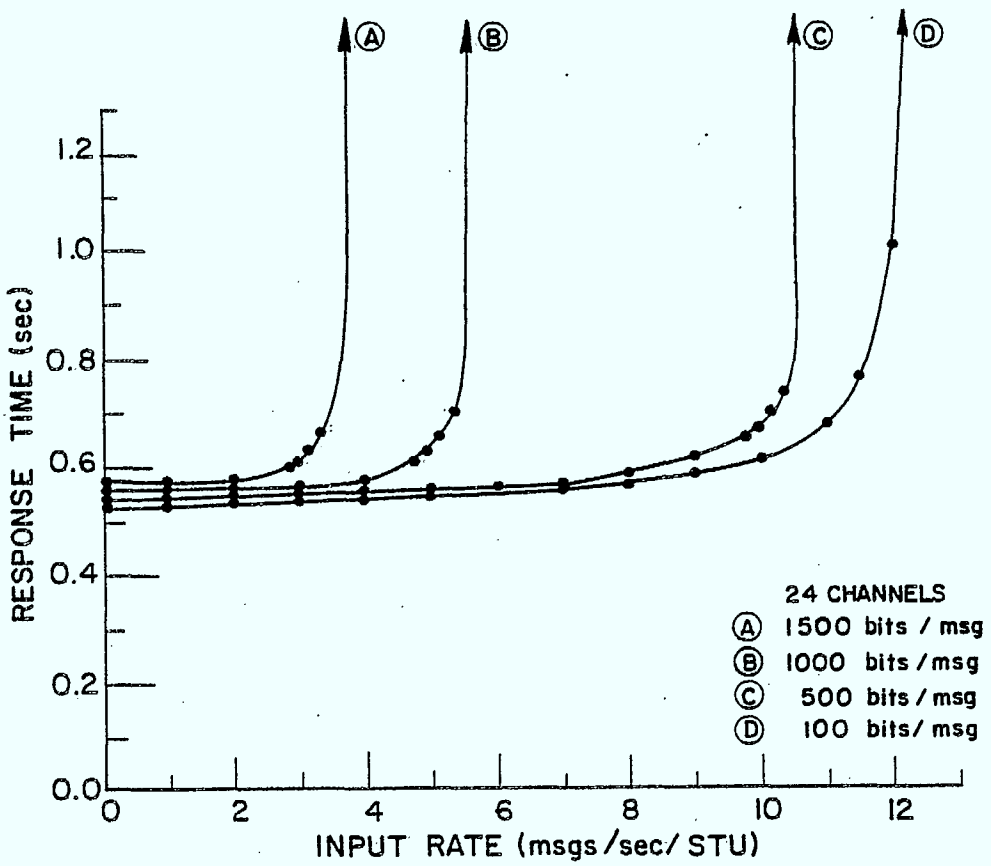


fig. 3.14a COMMUNICATIONS CHANNELS

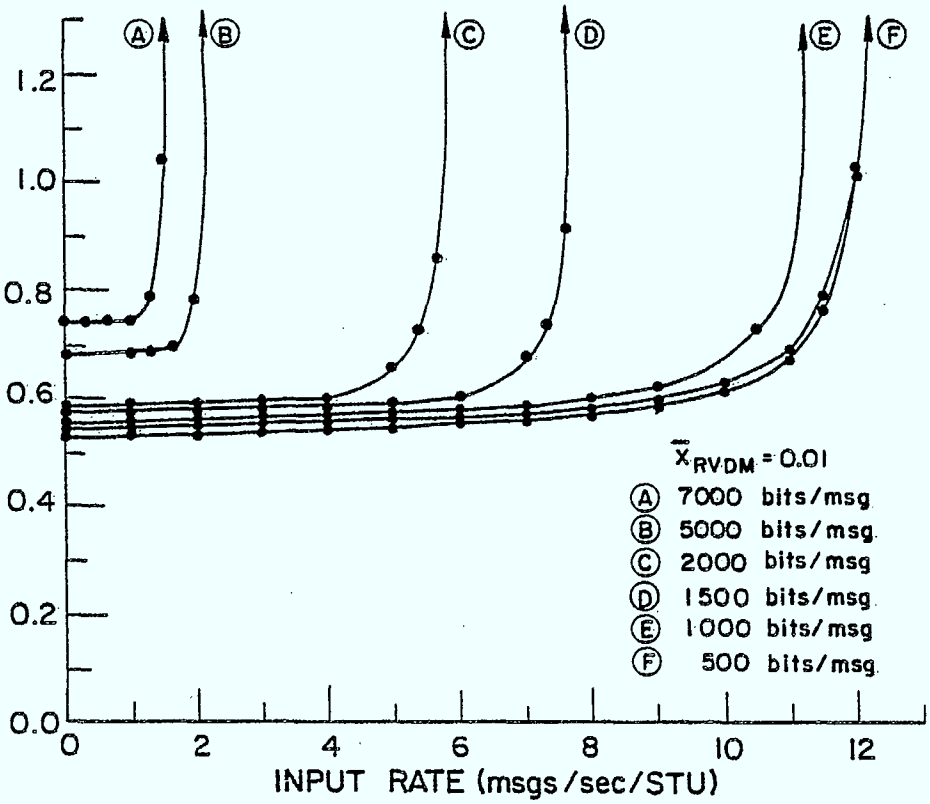


fig. 3.14b RVDM, DEFAULT PARAMETERS

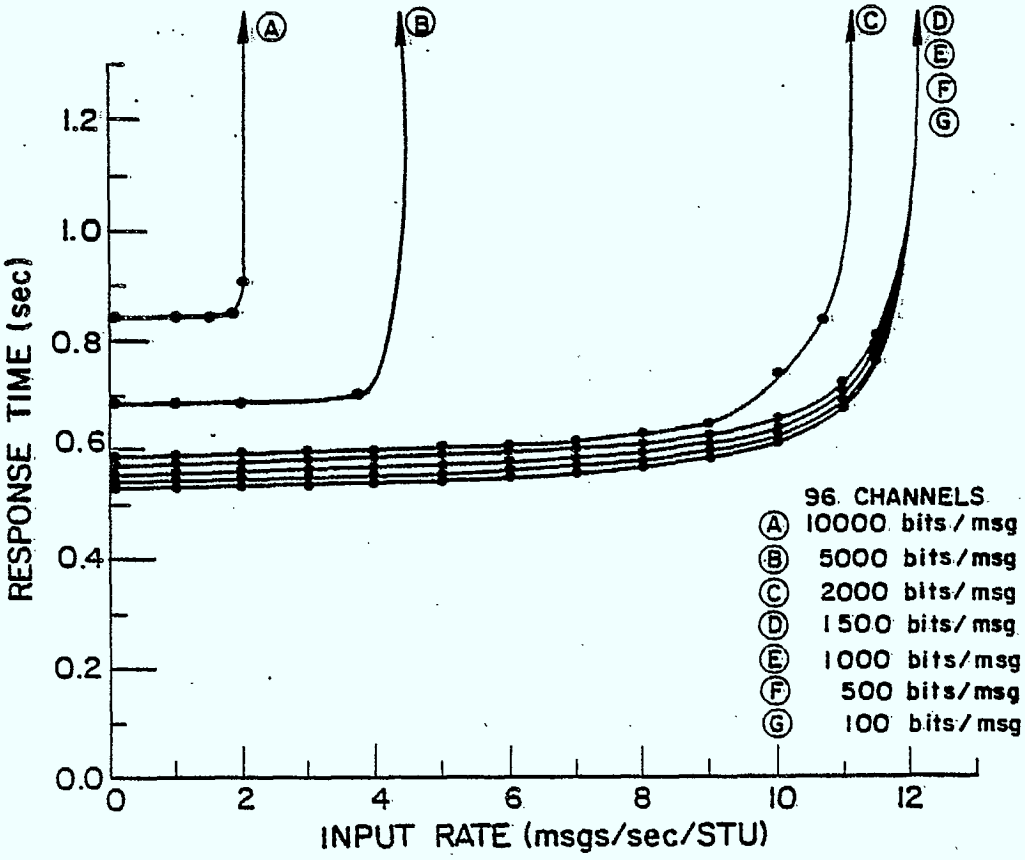


fig. 3.14c COMMUNICATIONS CHANNELS

about the same amount as for 48 channels. For these messages, the RVDM acts as the bottleneck. For larger mean message sizes, though, the network can handle about one-half the traffic rate as for 48 channels. This means that for videotex messages (7000 bits per downstream message), the network can handle just less than one message per second per STU. In figure 3.14c, the network has 96 channels, and can handle approximately twice the traffic level for larger messages. For messages of less than 2000 bits per message, the RVDM is still the bottleneck, and limits the traffic rate to 12.5 messages per second per STU.

The final experiment alters the configuration of the network, to test the RVDM when it has to handle more STUs. In figures 3.15a to 3.15c, the number of STUs on the network is kept roughly constant to minimize the effect of the rest of the network on the results. In these figures, the number of RVDMs is decreased from 21 (with 12 STUs per RVDM) in 3.15a, to 16 (with 16 STUs per RVDM) in 3.15b, to 8 (with 32 STUs per RVDM) in 3.15c. The default configuration, with 32 RVDMs and 8 STUs per RVDM may be found in figure 3.10b. In these figures, the RVDM becomes a more serious bottleneck, allowing a maximum of 8.3 messages per second per STU in figure 3.15a; 6.3 messages in figure 3.15b; and only 3.2 messages in figure 3.15c. The IDT/DCT units still act as the bottleneck for large messages, but when the RVDM is loaded as in figure 3.15c, the mean message size must be

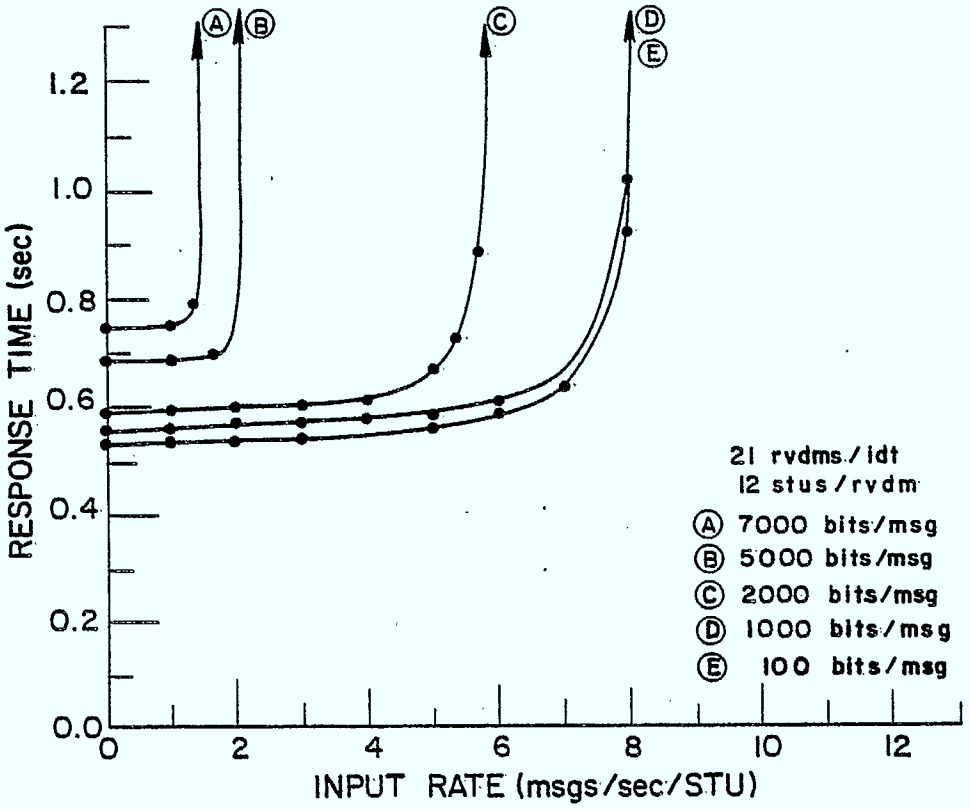


fig. 3.15a EXPERIMENT 4

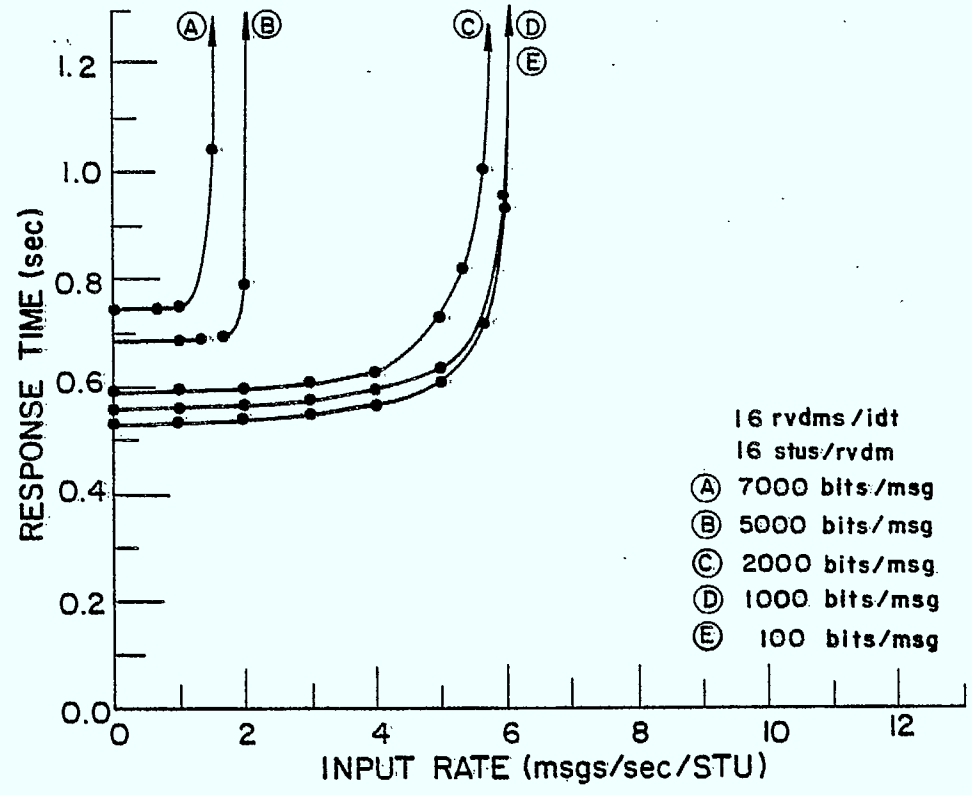


fig. 3.15b EXPERIMENT 4

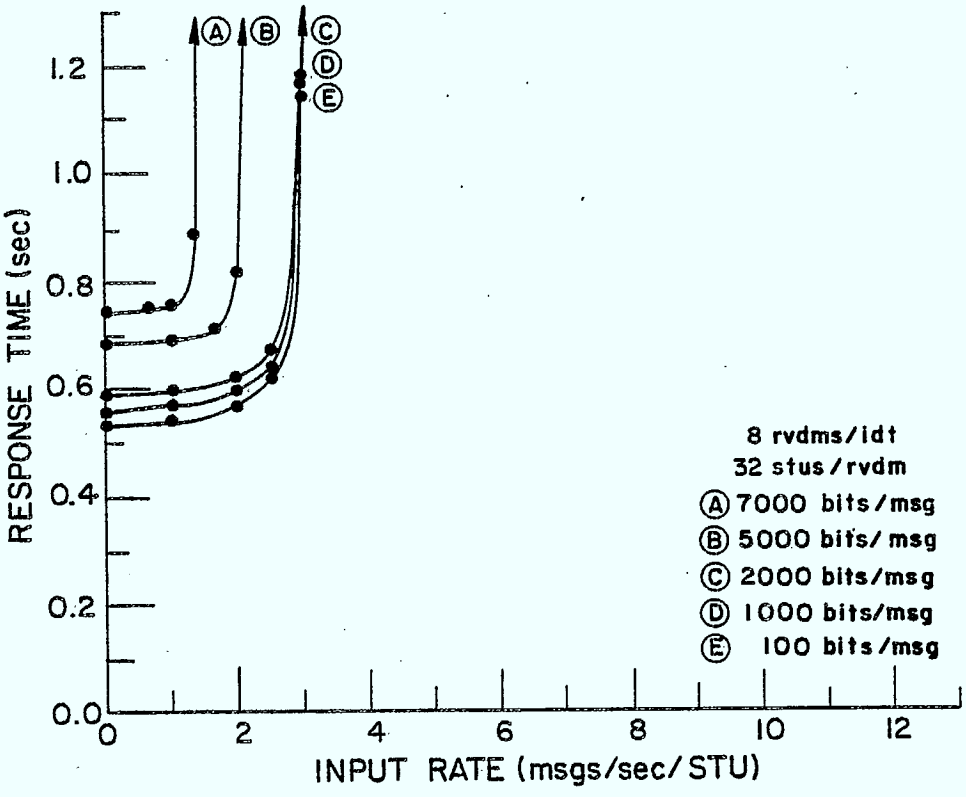


fig. 3.15c EXPERIMENT 4

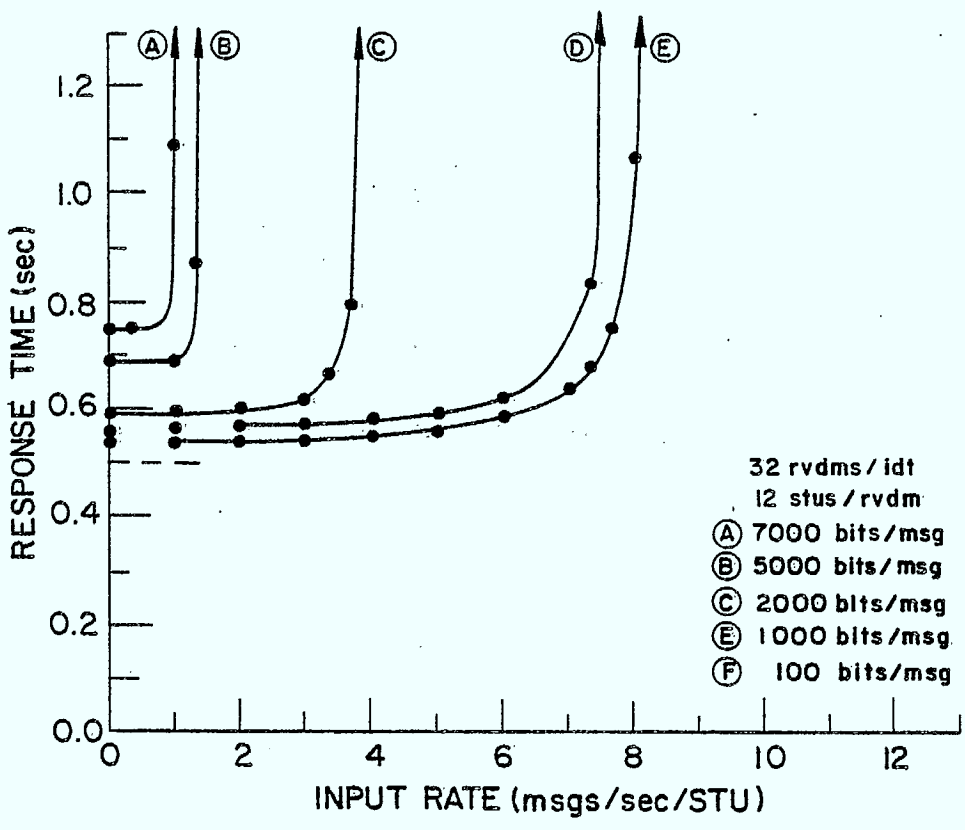


fig. 3.15d EXPERIMENT 4

around five thousand bits per message or more for this to happen.

In figure 3.15d, the total number of STUs on the network was increased from 256 to 384, with each of the 32 RVDMs now handling 12 STUs. Here, the values for response time are slightly higer than in the default case, due to the higher rate of messages and increased amount of queueing which arises from having more STUs. As well, each RVDM is handling more messages, and so the RVDM now restricts the traffic rate from each STU to 8.3 messages per second. Again, the IDT/DCT units restrict the flow of large messages in the network (mean message sizes of greater than 1000 bits per message). For videotex messages, this configuration allows only one message per second from each STU.

III.3.4.3.2 Bottleneck Analysis

The results from the bottleneck analysis are presented in figure 3.16. The first two graphs in this figure (3.16a and 3.16b) use the first set of equations as presented in section III.3.2.2. They show the maximum rate of messages that each server can tolerate, plotted against a range of mean message rates in the network. In the model, the RVDM has a constant service time, independent of the size of message, and as such, the RVDM line appears as a horizontal line. The other servers shown in these graphs have a service time dependent on the length of message, so that the

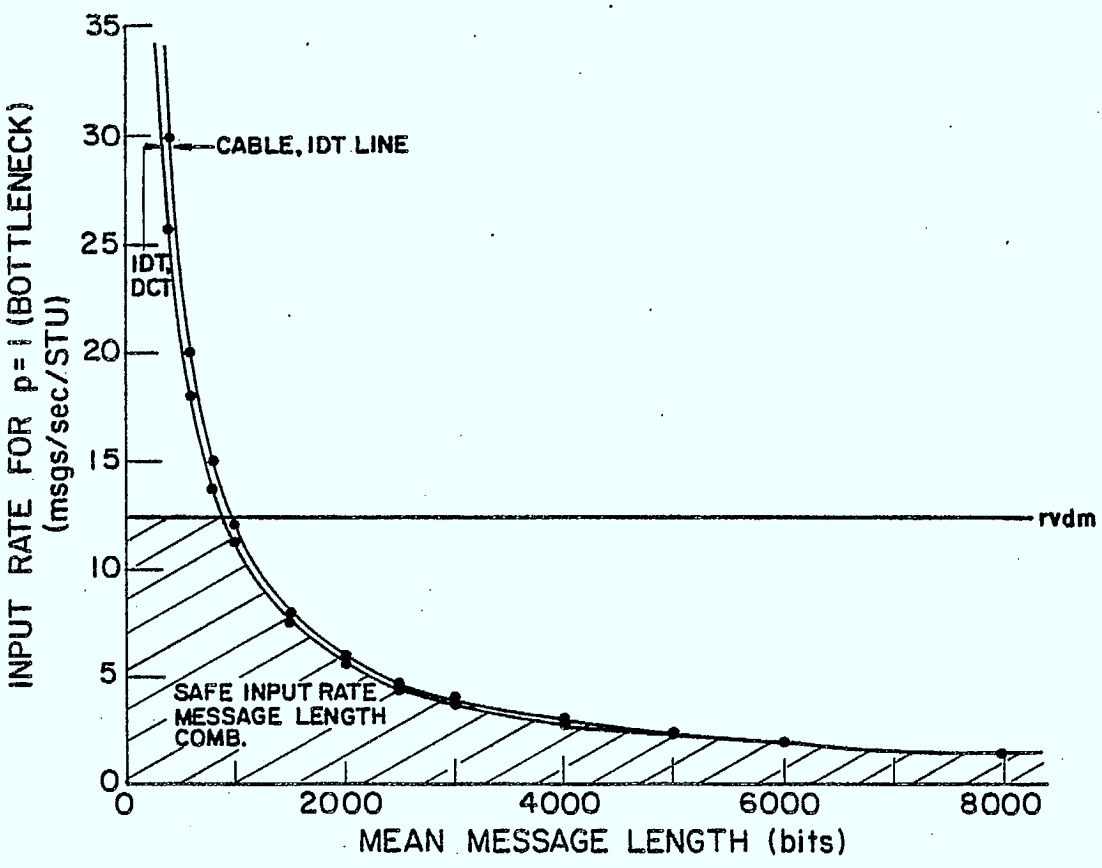


fig. 3.16 a BOTTLENECK ANALYSIS

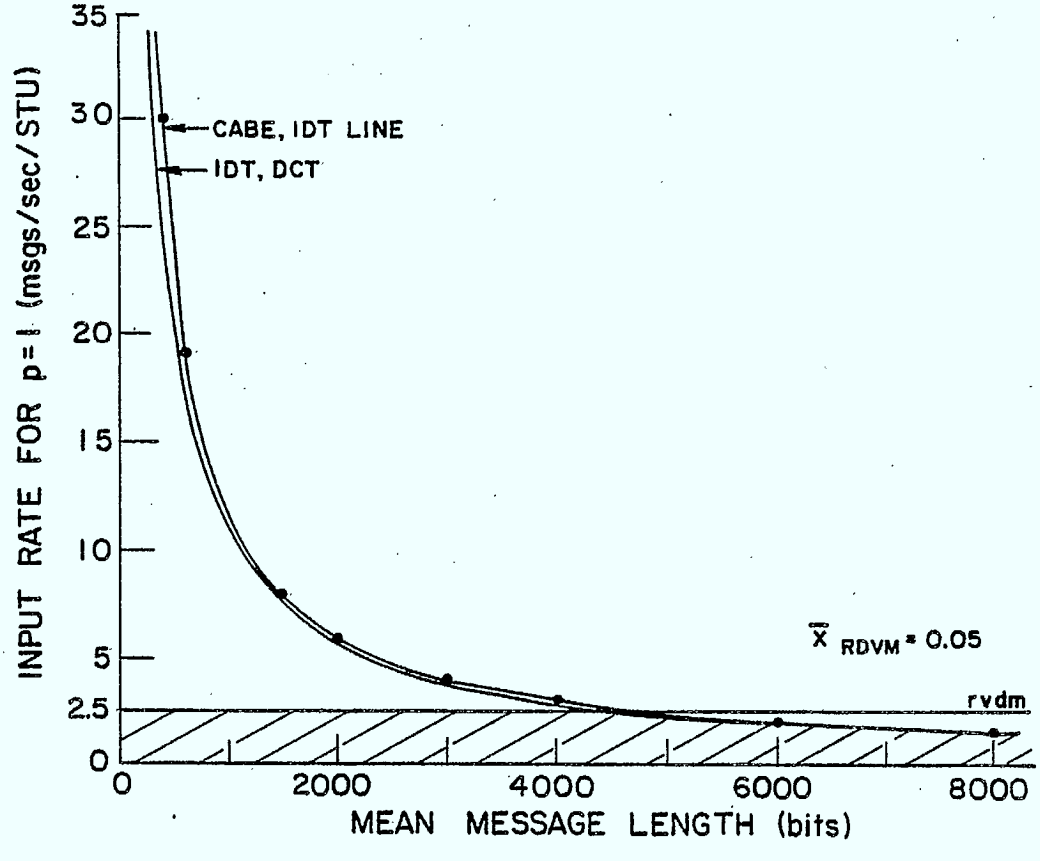


fig. 3.16 b BOTTLENECK ANALYSIS

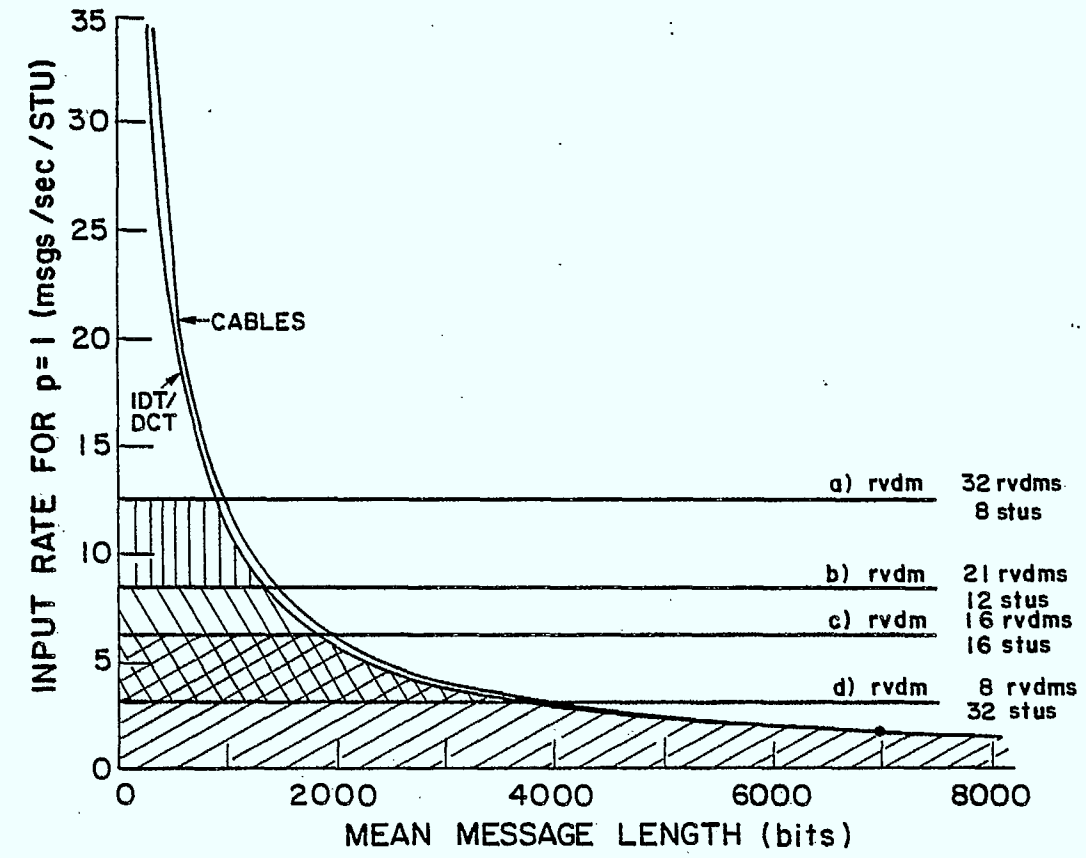


fig. 3.16c BOTTLENECK ANALYSIS

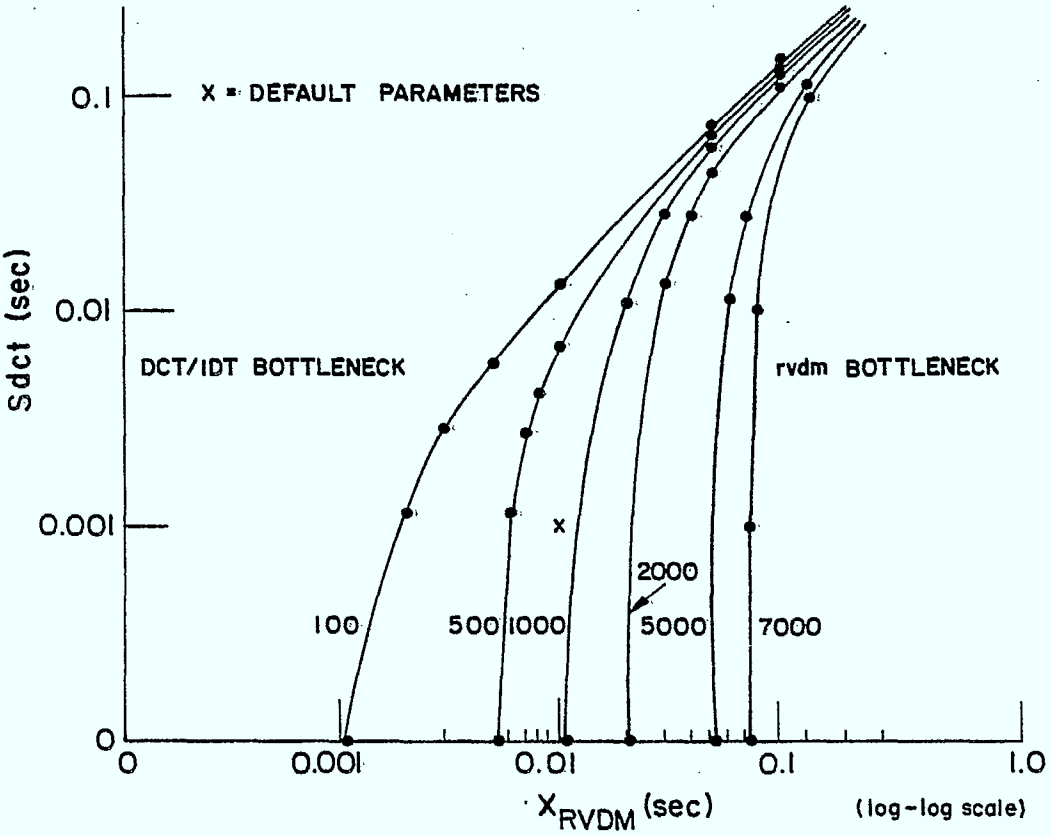


fig. 3.16d BOTTLENECK TRANSITION

maximum rate of messages which they can tolerate drops as the network's mean message size increases.

Figure 3.16a shows the results for the default parameters (see Table 3.1), and it can be seen that the RVDM limits throughput while the network contains messages whose mean size is less than 896 bits. As the mean size grows, the DCT and IDT units limit the throughput. The shaded area in the graph shows combinations of message rates and sizes which the network will handle without being congested. This graph shows that for videotex messages (which are actually about 7000 bits long in the current demonstration database at the University of Waterloo), the network can handle slightly over 1.5 messges per second per STU. It can handle then, the normal videotex load of 0.1 messages per second per STU.

Figure 3.16b shows a similar graph, but with the RVDM service time increased to 0.05 seconds (from 0.01 seconds in figure 3.16a). In this situation, the RVDM is the limiting factor for mean message lengths of less than 4736 bits, and the input rate is limited to 2.5 messages per second per STU. The network can still handle normal videotex loads under these conditions, however.

Figure 3.16c represents the same kind of bottleneck analysis, but it shows results from a set of different network configurations. These configurations change the number of RVDMs and STUs serviced by each IDT, as discussed

for figure 3.15 in the previous section. In figure 3.16c, there are four RVDM lines, each representing the RVDM saturation point for a different configuration. The line labelled "a" is the default configuration, which is discussed earlier. The "b", "c", and "d" lines represent the following configurations: "b", 21 RVDMs and 12 STUs per RVDM; "c", 16 RVDMs and 16 STUs; and "d", 8 RVDMs and 32 STUs. Each successive line represents an increased traffic load on the RVDM (which has more STUs to service), and thus each configuration can support a lower traffic rate from each STU.

These graphs also illustrate how the bottleneck changes from the RVDM to the IDT/DCT, and how the point of this change shifts from one configuration to the next. As the RVDMs become more busy, the point of intersection of the RVDM saturation line and the IDT/DCT line moves to the right. For the "a" line, this point is at about 900 bits per message, while for the "d" line, the point has shifted to about 3800 bits per message. The capacity to handle videotex messages (7000 bits on average), is not altered in any of these configurations, however. Each configuration can handle about 1.5 videotex messages per second from each STU.

The fourth graph (figure 3.16d) represents the results from the last equation in the bottleneck analysis. It indicates which server will act as a bottleneck (given an ap-

propriately large input rate) for all values of delay time for the DCT and RVDM. For any particular value for mean message length in the network, the parameter values which lie on the as a bottleneck, and the parameter values on the right of the line will cause the RVDM to act as a bottleneck. The default parameters (marked by an "x" on figure 3.16d) lie on the right of the 100 and 500 bits per message lines, and so the RVDM is the limiting server for these small message sizes. For larger messages, such as 1000 bits per message and higher (for example, videotex messages), the DCT and IDT units will limit the network.

III.3.4.4 Comparison of Simulation and Analytic Results

For the first three experiments discussed in the previous section, both analytical results and simulation results were presented (see figures 3.8, 3.9, 3.10b, and 3.11). If all of the assumptions are valid, these two methods should give very similar results, although some variation can be expected due to the random factors in the simulation. Each plotted point in the simulation graphs is the mean value taken from five repetitions of the run, so that this random variation will be very small.

Figure 3.8a and 3.8b are the analytical and simulation results (respectively) for the first experiment. These two graphs are almost exactly identical, except for values near the saturation point (1.25 seconds service time). Figures

3.9a and 3.9b are from the second experiment, and they are also very similar except near the saturation point. The divergence starts at about 75% utilization of the DCT and IDT, and it seems more drastic than in the first experiment. As well, there is a small divergence in the low-input rate section of the graph (less than 0.5 messages per second per STU). This second divergence does not seem too important because it is quite small.

Figures 3.10b and 3.11 show the analytic and simulation results respectively for the default set of parameters (described in table 3.1). These two graphs show virtually identical values for low traffic levels, and they also show the same saturation points. As with figures 3.9a and 3.9b, the analytic solution (3.10b) diverges from the simulation results, starting at about 70-75% utilization of the bottleneck server. The response time in the analytic solution increases fairly quickly as the traffic level approaches the saturation point. The response time in the simulation, however, remains at virtually the same level, until about 90% utilization of the bottleneck server, and then increases quickly to infinity.

The reason for the divergence at high traffic levels is most likely a failure in one or more of the assumptions discussed earlier. More information about the behaviour of individual queues and servers can be obtained from values for the mean number of messages in each queue (see section

III.3.2 for the analytic equations). Table 3.4 shows the mean queue lengths at the 90% utilization traffic level for the 5000 bits per message graph found in table 3.11 (the simulation run of the default parameters):

Table 3.4 Mean Queue Lengths

	Upstream	Downstream
DCT	44.2658	46.8458
Feeder	43.6542	43.5377
IDT	44.8195	44.1390
IDT-line	46.1064	43.5338
RVDM	0.1990	0.1990
STU	0.0022	0.0194

These numbers show that for the simulation, the DCT queue is longer than the rest for downstream messages. In the upstream direction, the IDT-line queue is longer than the rest. This phenomenon is shown more clearly in the table of mean waiting times for the same graph (values are shown in 1/10000ths of a second):

Table 3.5 Mean Waiting Time

	Upstream	Downstream
DCT	1.1980	49.4920
Feeder	0.0000	0.0000
IDT	11.8940	1.7876
IDT-line	43.0371	0.0000
RVDM	12.6727	12.1014
STU	0.0024	0.8222

The bulk of the waiting time is spend at the DCT downstream, and at the IDT-line upstream. In other words, the traffic seems to build up at the first multiple server queue that it reaches, whichever direction the message is flowing. The analytic solution does not show this behaviour.

Despite this divergence, it is important to note that not only are the two results similar for light traffic loads, but that they both reach saturation at approximately the same point. This is illustrated best with figures 3.9a and 3.9b. In the simulation experiment, the DCT and IDT downstream servers reached a 99% utilization level at 2.209 messages per second per STU (saturation is when one or more servers reach 100% utilization). The analytic solution reached saturation at 2.21 messages per second per STU. A simulation run, in practice, cannot reach the 100% utilization level, because at this level, queues may grow arbitrarily long and thus require more than all available

memory on the machine.

With these examples then, the analytic solution predicts the simulation results well, when the traffic load is light. At high traffic levels, near the saturation point for a particular set of parameters, the two results diverge (the analytic result giving higher values), but reach the saturation point at approximately the same traffic level.

III.3.5 Conclusions

A summary of the analytical and simulation experiments may be found in Table 3.6 at the end of this section.

From the results discussed in the previous section, it seems clear that the Omnitel network configuration studied here performs very well under videotex load conditions. Videotex presents a light load (about 0.1 messages per second per terminal) in comparison to the capacity of coaxial cable, and the network hardware does not seem to present any major problems. Since the exact values for service times of the hardware are not known, the system has been studied under a wide range of values. Even with "unreasonable" service time values (as in figure 3.10d), the network can handle over an average of one videotex message per second per STU, and so the network will not be congested with videotex traffic. These figures are averages per terminal connected to the network. This means that even if all of the videotex terminals are signed on at once, the network

can handle one message per second from each terminal, which is about ten times the estimated frequency.

However, Omnitel is an Integrated Services Network (ISN) and as such will carry a range of digital services in addition to videotex. For services like meter reading and alarm services, the load will be much less than videotex, as low as 2.6 bits per second per monitored item [WONG80]. The network can easily handle this load. For high bandwidth services like digital telephony, however, the situation is not as clear. Digital telephony puts a high load on the system, although research is being done to decrease the load (see ULUG77). It is possible that this kind of service may run into congestion problems under peak load conditions.

When bottlenecks occur due to very high traffic, they occur at either the RVDM or the DCT downstream and IDT-line upstream. The RVDM acts as a bottleneck in systems with small messages on average. With large messages, the bottleneck shifts to the DCT downstream, and IDT-line upstream -- the first multi-server unit in the direction of flow.

Table 3.6: Summary of Analytic and Simulation Experiments

a) Analytic Model Assumptions:

1. Exponential interarrival times at each server
2. Exponential service times for each server
3. One class of messages

b) Simulation Model Assumptions:

1. The Exponential distribution was used for the interarrival time of messages to the network as a whole
2. The Uniform distribution was used for service times, except for the communications channels, where service times are calculated exactly
3. The Erlang distribution was used for generating message lengths
4. Arbitrary classes of messages are allowed.

c) The Series of Experiments:

1. The amount of delay that a message receives at each server is increased until the system is overloaded. The network contains only videotex messages.
2. A constant flow of videotex messages, with an increasing flow of "external" messages are examined.
3. A series of experiments was conducted to examine the behaviour of each server in the

network. The mean size of messages in the system was varied in each experiment to determine the network's behaviour under a number of different conditions.

4. The configuration of RVDMs and STUs was changed to investigate the level of load that the RVDM could handle.

d) A Summary of Results:

1. In the configuration studied, Omnitel performs well under videotex load conditions
2. The RVDM, and DCT (downstream) and IDT (upstream) units are the most critical hardware units. The RVDM acts as a bottleneck when the network has small messages, and the IDT and DCT are the bottleneck when the network has large messages. (For more precise results, see section III.1.5).
3. It is uncertain whether or not Omnitel can handle very large load services such as digital telephony at their peak traffic levels.

III.4 Encryption-Switching for Delivery of Telidon on the CATV Network

As a network for the distribution of Telidon service to subscribers, the CATV network is an attractive alternative to the telephone network or even to public packet-switched networks. The cable network, like communication satellites, has the broadcast property which is essential in supporting teletext and distributed databases. The cable network can be used easily for the delivery of broadcast Telidon (teletext) since all subscribers receive the same cycle of broadcast pages and the page selection function is performed at the receiver. However, for interactive Telidon, the selection of pages and subscriber addressing are performed at the data base host. The conventional approach to the provision of this service (or other data services) on the CATV network requires the use of switching nodes throughout the network to switch circuits or route packets at each branch in the network's topology [COYNE80, WONG80]. In the Omnitel system, for example, the IDTs, RVDMs and STUs are basically switching nodes located at various levels in the hierarchy of the network. (Please see section III.2.3.1 for a review of the Omnitel system components). This type of network architecture arises from a need to conserve and share the available bandwidth on the lines between nodes.

However, on current CATV systems, network bandwidth is not a system bottleneck and will be even less so as optical

fibres become commonplace. Therefore, packets transmitted by the Telidon service host in response to user inquiries could be simply broadcast to all subscribers, with a great saving in equipment complexity. Each subscriber would attach to the network using an intelligent interface "box" which would check the address field of each packet and capture those packets which are destined for that particular interface and subscriber. (This is similar to broadcast videotex systems, but operating on the level of PDI-encoded pages rather than video frames.) In this approach, the portion of the CATV network used for the delivery of Telidon data behaves very much like a local area computer network. Local area network transmission protocols (e.g. CSMA-CD or IEEE 802 token passing) could be optimized* for the length of the cable system and implemented in the subscriber interface boxes, particularly so if we want to allow more than one subscriber to transmit pages.

Although they were originally designed to be used on bus - type local area computer networks consisting of a

*Since most CATV networks are somewhat longer than a typical local area network, the propagation delay time between the two most distant subscibers, which is sometimes called the network diameter, is longer for the CATV network. This delay time is significant in most local area network transmission protocols because it represents the time required to determine whether data collision has occurred. The typical length for a local area network is 1 km and 10 km is typical for a CATV net. Therefore, the transmission protocol used on the CATV net must be adjusted to accommodate collision detection times which are ten times greater than those for which the protocol was initially designed. This can be done, for example by using longer packets and lower speed channels than is usual.

single length of coaxial cable with no branches, local area net transmission protocols, such as CSMA-CD, can be used on a tree structured CATV system. Of course, the data cannot be transmitted at baseband. Two frequencies are required: one for "upstream" data and the other for "downstream" data. The subscriber interface boxes use the upstream channel to transmit data. This data is retransmitted, by a repeater at the cable head, on the downstream channel. Data from a central server, such as a Telidon database host, may either be injected on the upstream channel, and then repeated, or may be introduced on the downstream channel at the head end.

In summary, the cost and complexity of introducing switching nodes throughout a CATV system, in order to support data services for Telidon using circuit switching or packet switching, may be avoided if local area network techniques are employed on the cable. This requires the adoption of a transmission protocol and the use of microprocessor-based intelligent interfaces at the subscribers' premises.

However, this local area net system may not be sufficiently secure for some sensitive potential Telidon applications such as EFTS, teleshopping and electronic mail. For example, a subscriber could gain access to all the data or pages being sent to every other subscriber on the system just by altering his address or by changing address recognition logic in his subscriber interface box. Furthermore,

the subscriber must take deliberate, verifiable action to request some classes of information, so that he can be billed for such accesses and so it can be proved that he requested such access (the Accountability Requirement). In order to satisfy reasonable security requirements for applications such as EFTS and teleshopping and still be consistent with the local area computer networking approach, an encryption system based on classes of traffic may be used. (Also, we will see below that such schemes satisfy the Accountability Requirement.) We have coined the name encryption-switching for this approach; it will be discussed in detail in the following subsections.

III.4.1 Classes of Traffic

There are four types of traffic which will be used on a Telidon "information utility" service. These are:

1. Frequently-Used Pages

   These most frequently used pages form the basic information nucleus around which Telidon service for home use is built. Topics such as news, weather, sports, local events and perhaps business applications such as money market information would be included. Gordon Thompson refers to this class of pages as bubble-pack information in analogy to the common, self-service, high volume and non-specialty items packaged in bubble packs in, for example, hardware stores. We assume that such items of

information will not be charged for individually; rather, a flat monthly subscription fee will be used so that the Accountability Requirement does not apply.

2. Less Frequently Used Information

   This includes the specialized information pages which are not included in the first category, as well as services which require interaction such as teleshopping, EFTS and electronic mail. Such information is charged for on an item-by-item basis, so the Accountability Requirement applies.

3. Closed User Group Information

   This class may contain secondary information retrieval services which are publicly available for an additional (flat-rate) subscription cost. (Some pay television systems incorporate secondary channels for an additional fee which offer program material of a specialized type such as sports or violence). Another use for this traffic class would be as a delivery vehicle for private (e.g. corporate) information on the public system to an appropriate subset of subscribers. If such information is charged for on an item by item basis, then the Accountability Requirement applies. However, the system operator may wish to make this type of secondary service available for an additional,

fixed, monthly fee.

4. Subscriber-to-Subscriber Information

   This is the class of traffic which is carried by a point-to-point network service among subscribers. Accountability is important since the network administration will require that the technical means is available for the introduction of usage sensitive tarrifs for this service. Security is also very important.

III.4.2 Delivery Schemes

The following encryption, transmission and decoding schemes describe the manner in which we envisage the delivery of each of the traffic classes described above. These techniques, again, are based on an underlying local area computer network implemented on CATV plant.

1. High Use Information

   The "bubble-pack information" represents the basic information service available to subscribers. Therefore, this data is broadcast in a cycle to all subscribers. The desired pages are selected at the subscribers' decoders. In order to protect the revenue of the service provider, it may be desirable to encrypt this information at the lowest security level to ensure that people cannot easily receive the information without subscribing. However, it is

not worth protecting the service from sophisticated attempts at reception since the subscription fee for this service is relatively small. That is, it would be less expensive to pay the subscription fee than to cheat. A flat subscription fee entitles the subscriber to all of the bubble-pack information he can consume.

2. Less Frequently Used Information

This class of traffic carries Telidon pages which are selected interactively (videotex) as well as advanced services. These pages are charged for on a per use basis; therefore, encryption should be used between the host computer and each individual subscriber. (The process of encryption is described in section III.4.3.) Encryption provides improved protection against theft, and the act of requesting a decryption key is the deliberate action needed to satisfy the Accountability Requirement. The sensitive nature of electronic mail, EFTS and teleshopping suggests that the encryption technique must be secure from both passive and active attacks. Also, these same services will require secure and convenient identification and authorization methods.

3. Closed User Groups

This traffic is transmitted between the host com-

puter and the subset of subscribers who are members of a group. This traffic should be encrypted to the level of security which is appropriate for the particular value of the information. For example, a public secondary retrieval system specializing in sports pages may only warrant a relatively low security level compared to an in-house corporate data base carried on the public system. The system operator may choose to charge for this information on a per-item basis or a flat rate basis. For the former case, the Accountability Requirement is satisfied as in (2). In the latter case, the subscriber would be issued a long term key at subscription time which provides the deliberate action required to satisfy the Accountability Requirement.

4. Subscriber-to-Subscriber Information

This traffic class requires encryption for reasons of privacy. Although many users, particulary home subscribers, may be willing to waive this requirement, no point-to-point data services should be installed without the capability for encryption to provide privacy. The need for encryption is increased by the use of local area network techniques on the CATV system since all subscriber interfaces have access to all the data packets transmitted on

the cable. The Accountability Requirement for this class of traffic will be discussed in section III.4.10.

III.4.3 Data Encryption

Data encryption involves the transformation of plaintext to ciphertext (see figure 3.17). This transformation is performed at the data source and the ciphertext is transmitted over the network (or other communications service) to the destination, where the transformation from ciphertext to plaintext is performed. A good encryption technique should be immune to both passive (listening or tapping) and active (introduction on illicit data onto the network) attacks by intruders. Also, it must be assumed that a passive intruder can compare recorded ciphertext to strings of plaintext which he assumes are being transmitted. For example, each user session for a given service may begin with a "login" message or a standard set of codes. The intruder can compare this plaintext to the recorded ciphertext in an attempt to deduce the encryption technique. This is a type of passive attack and is referred to as a plaintext attack.

Encryption techniques are based on combinations of transpositions and substitutions of characters. The combinations are described by an algorithm and a key. The encryption algorithm specifies, in general, the types of substitutions and transpositions to be applied to the plaintext. The key is a word or sequence of digits which is

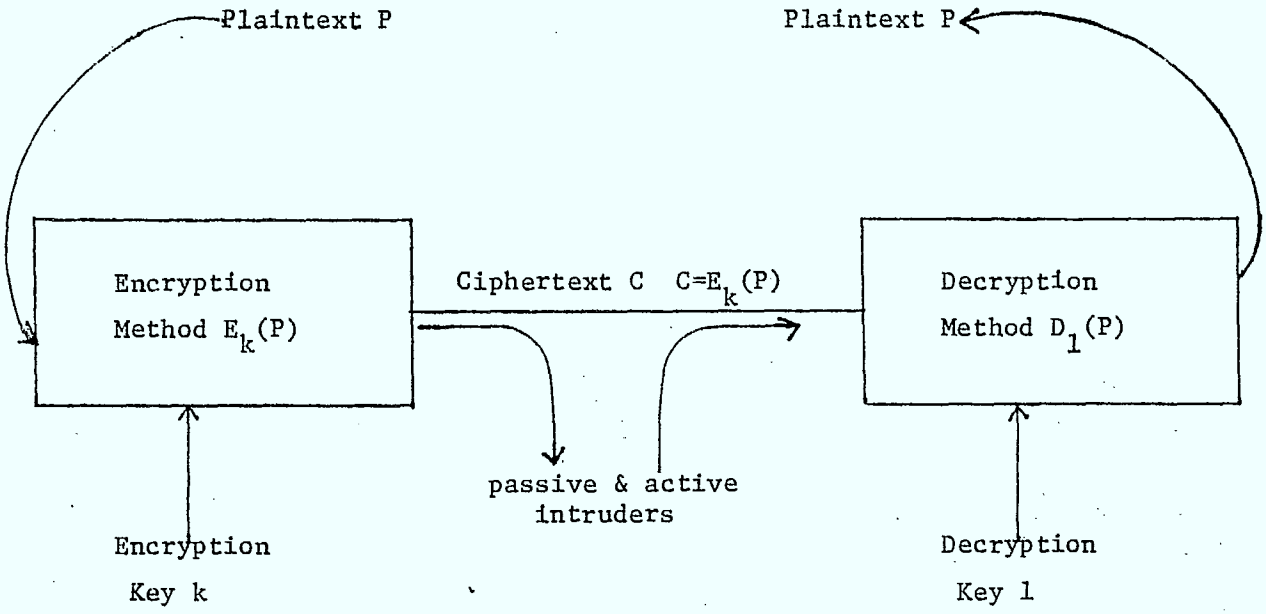


Figure 3.17: A Block Diagram of Basic Encryption

interpreted by the algorithm and specifies the precise substitutions and transpostions to be used. In the days of manual encryption, simple algorithms were used (to reduce computations) in conjunction with long keys which were changed frequently. Typically, the new keys were carried to both ends of a communication channel by a courier. Both the algorithm and the key were secret.

Current computerized encryption techniques use relatively complex algorithms (since the computation is automatic) in conjunction with a fairly short key. The algorithms are published and available but the key is kept secret.

III.4.4 The Data Encryption Standard

The Data Encryption Standard (DES) is a relatively popular encryption algorithm which was developed by IBM and adopted by the U.S. National Bureau of Standards in 1977 [TANE81]. It uses a 19 stage algorithm, which is available as a chip implementation, for both encryption and decryption. A 56 bit key is used - the same value of key is used for both encryption and decryption. These chips can encrypt blocks of 64 bits at a time, or they can perform stream encoding which is even more secure. Details on the Western Digital DES chip are included in Appendix B.

The DES technique is secure against plaintext attacks which can be mounted using computers, unless years of computer time are devoted to the attack or unless the intruder

knows some information about the key. It is also relatively secure against active intruder attack.

In order to use DES between a data source and a data sink it is necessary to transmit an identical key to both of these users in a secure fashion, by a central key server over individually encrypted channels or by use of Merkle's method [MERK78]. (An intruder who manages to learn the key has of course broken the security of the system.) Merkle's method requires the data source to send a number of puzzles to the data sink who, in solving a puzzle, recovers a DES encryption key which can be used for the duration of the connection. More detail on this technique is given in section III.4.6. Since Merkle's method involves significant computation by the data source and data sink to determine the key when the connection is established, we stated in the last progress report that this method is probably impractical for use with the subscriber interface boxes described above. The Project Officer asked us to examine this topic more thoroughly and, happily, we are able to report that a fairly simple form of Merkle's method which does not present a large drain on the subscriber box's CPU is feasible. The details are presented in section III.4.8.

DES is not an actual public key encryption system (see below for a description of public key cryptography) because of the key distribution difficulties. (True public key encryption systems allow secure point-to-point communications

to be established immediately between pairs of subscribers who have not previously communicated). However, DES is clearly useful for the first three classes of traffic since they are based on a central host computer (some possibilities are discussed below). DES can be used for privacy in a point-to-point system if either a central key server is implemented or Merkle's method is used. The central key server is discussed in more detail in section III.4.9.

Several alternatives exist for the use of DES to provide security and accountability for the transmission of the first three types of traffic. For example, one fixed key may be used by all subscribers to receive bubble pack information; this key would be provided at subscription time. The second class of information, interactive pages and advanced services (e.g. Teleshopping, EFTS) would use individual keys for each subscriber, set at subscription time. The computer which provides the service would retain each of these keys. If plaintext attacks are a danger, then the subscriber key could be used just long enough for the central server to send a session key to the subscriber interface box. This drastically reduces the amount of ciphertext which has been encrypted using the subscriber key and which is available to the intruder attempting the plaintext attack.

The third type of traffic, closed user groups and secondary services, could be encrypted using one fixed key on a system-wide basis for each instance of this type of service, or individual subscriber keys could be used. It may also be possible to combine these two approaches in a two step procedure.

III.4.5 Public Key Cryptography

In a true public key cryptography system, the encryption algorithm and encryption key to be used for each subscriber and/or service are made public. The security of the ciphertext arises from the choice of pairs of encryption and decryption keys so that the decryption key cannot easily be deduced from a complete knowledge of the encryption key. Therefore, anyone can obtain the means to send ciphertext to a particular subscriber but no one except the intended recipient can transform the ciphertext back to plaintext. There are serious difficulties concerning the immunity of this technique to active intruder attack.

Only two pairs of algorithms which satisfy the above requirements have been developed. Both of these rely on computational complexity for security. The MIT algorithm, devised by Rivest [RIVE78], is based on pairs of large prime numbers. The Knapsack algorithm, discovered by Merkle, involves multiplication of the plaintext by a weight vector [TANE81]. There is reason to believe that a third pair of algorithms may have been recently developed by researchers at MIT.

Since the area of public key cryptography is in its infancy, no hardware implementations of these algorithms are currently available. This limits their usefulness for Telidon delivery since it would probably be prohibitively expensive to dedicate CPU cycles in the subscriber interface box to the encryption and decryption of data. Furthermore, the subscriber interface box would either have to contain all the encryption keys for all the subscribers on the network, which is problematic in a dynamic situation or in a very large network, or, when the box needs a particular subscriber's encryption key it must obtain it from a central server (and there is no advantage over the use of DES with a central key server). Therefore, we are motivated to examine the use of Merkle's method or a central key server to provide public key cryptography using the DES chip for point-to-point data services. We also recall that DES is entirely sufficient for the first three classes of traffic.

It has been suggested [BLAK82] that an encrypted point-to-point data service could be based on a combination of public key cryptography algorithms and DES in the following way. When subscriber A wishes to send to subscriber B, he encrypts a DES session key using B's public key and the public key encryption algorithm. B then decrypts the session key, and both A and B switch to DES for the session. However, this method is not viable because of the same problems of storing or decrypting with public keys cited

above.

III.4.6 An Explanation of Merkle's Method

If the subscriber interface boxes incorporate DES chips for encryption, then in order to provide point-to-point data service, as is required for the class of subscriber-to-subscriber information described in section III.4.2, it is necessary for each pair of subscribers who wish to communicate to agree on a DES encryption key at the beginning of each session or connection. In section III.4.4, two techniques for establishing the key were introduced: Merkle's method and the central key server. Merkle's method is described in detail in this section; the central key server is discussed in section III.4.9.

Merkle's method involves sending a large number of puzzles, each of which is encrypted using an individual simplified key, from the data source to the data sink with whom it wishes to communicate. Each puzzle contains a large number of zero bits followed by a unique number and a 56 bit DES encryption key. The "simplified keys" used to encrypt the puzzles are DES encryption keys of a previously agreed format, for example, 34 random bits followed by 22 zeros.

The data sink selects one of the large number of puzzles at random and solves it by decrypting it with all possible keys consisting of 34 random bits followed by 22 zeros ($2^{34} = 1.7 \times 10^{10}$ possible keys). He knows he has decrypted the puzzle correctly when it yields plaintext

beginning with the large number of zero bits. The data sink then sends to the source the unique number from the puzzle which it solved in unencrypted form, followed by data which has been encrypted using the key obtained from the puzzle. When the data source receives the number of the puzzle it can look up the corresponding encryption key which the source then uses to decrypt the data from the data sink. This key is used to encrypt the data in both directions for the remainder of the session.

The security of this method of key distribution derives from the large number of puzzles which are sent by the data source. If an intruder hears all the transmissions, including the puzzles, he cannot know which puzzle the data sink will decide to solve to obtain the key. To find the key, the intruder must solve, on average, half the puzzles and compare the unique number from each puzzle to the number sent from the sink to the source. When the numbers match, the intruder has found the session key. The puzzles may be made sufficiently numerous to set the difficulty for the intruder at any desired level.

III.4.7 A Numerical Example of Merkle's Method

It is instructive to examine a typical application of Merkle's method to a hypothetical pair of communicating computers to investigate the timing and complexity involved for both the computers and an intruder.

Assume that the data source sends 10,000 puzzles to the data sink, each of which is encrypted with a key containing 28 random bits foollowed by 28 zeros.

After the data sink has chosen a puzzle at random, he must solve it by exhaustively trying each possible key of 28 bits followed by 28 zeros.

In total this is 228 or 2.68 x 108 keys. If the Western Digital DES chip described in Appendix A is used, then approximately 60 microseconds are required to load the key and first 64 bits of puzzle ciphertext and to retrieve the first 64 bits of plaintext. Although this does not yield the plaintext for the entire puzzle, if the first 64 bits are not all zeros then the puzzle has not been solved and another key should be generated and tried. Thus, the decryption of one 64 bit block will be sufficient to eliminate most keys.

In the worst case, the data sink must try all of the 2.68 x 108 keys. If we assume no delay for the program to evaluate the results of a decryption and generate the next key then we may use the decryption time (60 microseconds) as the time required to test each key.* The data sink will require a maximum of 4.5 hours and an average of 2.3 hours to solve the puzzle.

---

* The decryption time can be reduced by approximately 20% if the DES chip could be configured so that the ciphertext does not have to be reloaded each time a new key is loaded and tried.

An intruder who has copied all the puzzles can also solve a puzzle in 2.3 hours on average. However, the intruder must solve each puzzle until he finds the one chosen by the data sink. On the average, the intruder must solve 5,000 puzzles which will require about 1.3 years; the worst case is about 5 years.

Obviously a session establishment time of two to four hours is unacceptable for point-to-point data services on the encryption-switching Telidon delivery net. Probably the only users who could tolerate this establishment time are those who will use the same key over a long period or those with a very high security requirement.

III.4.8 Merkle's Method for the Encryption-Switching Net

The number and difficulty of the puzzles can be adjusted to provide a version of Merkle's method which is more suitable for use on the encryption-switching network. Consider the following scenario.

The data source sends 2000 puzzles to the data sink, each of which is encrypted by an individual key consisting of 17 random bits and 39 zeros. The data sink can solve a puzzle in approximately 4 seconds on average or 8 seconds for the worst case.

To break the code, the intruder must on average solve 1000 puzzles requiring 4 seconds each which yields approximately 1.1 hours. The worst case time is 4.5 hours.

Since each puzzle is about 195 bits in length, 1.3 minutes are required for the data source to send all the puzzles to the data sink if we assume the effective data rate between them is 4800 bps. (4800 bps is typical of the speed at which microprocessors can accept data while performing moderate amounts of processing on that data as it arrives. Also, 4800 bps is probably a good target for the network throughput as seen by each subscriber. If the subscriber interface box and the network can both perform at a higher speed, the session establishment time can be reduced.)

If this version of Merkle's method requires further adjustment for use on the encryption-switching net then the following observations will be useful:

1) The length of time required for the data sink to solve the puzzle is controlled by the number of random bits in the keys used to encrypt the puzzles.

2) The length of time for the intruder to break the code depends on both the number of puzzles and the number of random bits in the puzzle keys.

3) As the puzzles are made simpler to reduce the session establishment time their number must be increased to maintain the same level of security. This acts to increase the session establishment time since the data sink must receive all the puzzles

before choosing one for solution. Therefore, the complexity and number of the puzzles must be traded-off.

III.4.9 A Central Key Server for DES

To provide point-to-point services on the Telidon delivery net using DES encryption, a central key server may be used instead of the Merkle's method approach discussed above. The server will hold individual encryption keys for each individual subscriber; these are set at subscription time. (This is not unduly onerous, since the Telidon host system requires individual encryption keys for each subscriber for the interactive Telidon services anyway.) Of course, each subscriber interface box will be able to decrypt any information sent to it by the central server, since it knows its own key.

If subscriber A wishes to communicate with subscriber B, the following steps must be undertaken:

1. A requests a session encryption key from the central key server for an A-to-B connection.
2. The central key server sends two things to A: the session key encrypted by A's "own" (used for server-to-A communication) key; and the session key encrypted by B's "own" key.
3. A can decode the former to get the session key.
4. A sends the latter to B without further encryption as part of the session establishment procedure.

(Alternatively, the central key server could have sent the session key directly to B if B knew that A was establishing a connection with him and, therefore, was anticipating a key.)

5. A and B can now communicate using the session key.

III.4.10 Comparison of Merkle's Method and the Central Key Server

Both Merkle's method and the central key server method are feasible for the establishment of session keys for subscriber-to-subscriber data services on the encryption-switching network. However, they do have advantages and disadvantages relative to one another when compared.

The most significant issue concerns the Accountability Requirement in connection with the use of Merkle's method: since Merkle's method allows the key to be determined by the subscribers without involving a central server, there is no mechanism inherent in the encryption-switching technique to inform the network operator that network resources are being used. Therefore, the use of Merkle's method for subscriber-to-subscriber traffic does not satisfy the Accountability Requirement.

The central key server technique does satisfy the Accountability Requirement on two levels. Initially, each subscriber must register for subscriber-to-subscriber service to receive an encryption key for the central key server. Also, each time a subscriber-to-subscriber session

is initiated the subscribers must obtain a session key from the central key server.

Merkle's method requires more software in the subscriber interface box for the generation and solution of puzzles than the central key server. Unless the central key server has an unusually long response time, it will provide session establishment more quickly than will Merkle's method.

III.4.11 Network Capacity Considerations

The traffic classes discussed in section III.4.1 can be classified in terms of their communication paths as follows:

type 1 - broadcast traffic (cycles of pages)

type 2 - subscriber to central host

type 3 - subscriber to subscriber

Network capacity considerations for these types are discussed below.

a) Broadcast Traffic

This traffic may be transmitted as teletext in the vertical blanking interval (VBI) of a television signal or it may occupy the entire bandwidth of a television channel. Because there are limits to the delay or response time which the user will tolerate, the number of pages included in the broadcast cycle must be controlled. Typically, no more than 300 pages in he VBI and 5,000 pages in an entire television channel are considered practical.

A third alternative is to use one or more channels of a broadband local area computer network (LACN) to carry broadcast data. (A broadband LACN typically has a number of data channels; the data for each channel modulates an RF carrier so that the data channels can be transmitted on the same network in different frequency ranges. Broadband LACNs are quite compatible with CATV plant; however, A baseband LACN like Ethernet cannot be used on the CATV network because of the CATV amplifiers, cut-off below 5 MHz.)

Most local area networks support a broadcast mode all subscribers receive the data being broadcast. This could be exploited for the "bubble-pack" information. Since a broadband LACN on the CATV network will be required to support both the traffic between subscribers and the central host and subscriber-to-subscriber traffic, then if the same LACN subscriber box is also used for broadcast bubble-pack information it will save the subscriber the cost of an additional teletext decoder. Of course, this savings is only realized if the subscriber wants more than just the bubble-pack service.

The Sytek broadband LACN provides 120 channels of approximately 300 KHz bandwidth each with a data transmission rate of 128 kbps per channel [SYTE81]. If we assume the average length of a Telidon page is 1000 bytes* and that a user can wait no longer than a maximum of 45 seconds for a

*This is approximately correct for the UW Telidon database.

page to appear, then 720 pages may be transmitted in a cycle on a single Sytek LACN channel. This figure should be reduced somewhat (approximately 10%) to allow for the protocol overhead on the network.

b) Subscriber to Central Host Traffic

Several channels of a broadband LACN can be used to carry this traffic on the CATV network. Of course, the subscriber boxes would include the encryption switching techniques described earlier in this section. Undoubtedly, the traffic downstream from the server to the subscribers will be much greater than the upstream traffic (e.g. a user need only send a few characters to trigger the reception of an entire Telidon page). Therefore, to approximately gauge the number of subscribers each channel can support, we may assume the upstream traffic is negligible and consider only the downstream traffic. Furthermore, let's examine only interactive videotex applications, since transaction-oriented services probably involve less traffic. Assume the following:

1. 1000 bytes/page
2. 1 page request every 20 sec from each active subscriber
3. 10% of the subscribers are active at any time
4. an effective transmission rate/channel of 116 kbps, to account for the protocol overhead.

The delivery network can support:

$$\frac{116 \text{ (Kbits/sec)}}{1 \text{ (Kbyte/page)} \times 8\text{(bits/byte)} \times .05\text{(pages/subscriber second)}}$$

which is 290 active simultaneous subscribers per channel of the broadcast LACN described above. Since only 10% of the subscribers are active at any time, a total of 2900 subscribers can be supported on each channel.

If the system has 120 channels, as the Sytek does, then a total of 348,000 subscribers can be supported.

c) Subscriber-to-Subscriber Traffic

The traffic in this category may involve terminals communicating with other terminals or with computers. Computer to computer traffic between microcomputers is probably the most demanding application in terms of data rate which is likely to occur. Since these applications may not involve Telidon it is difficult to evaluate this traffic in terms of a number of Teldion pages. However, it is well known that most conventional microcomputers and many minicomputers cannot accept data faster than 4800 to 9600 bps for more than a short burst if they perform a moderate amount of processing on the data as it arrives. Therefore, approximately 12 to 24 sessions between subscribers may be simultaneously actively transmitting data on each Sytek broadband LACN channel. If we assume a 10% utilization of the communications services for each session then approximately 120 to 240 ses-

sions may be accommodated on each channel.

d) DES chip Throughput

The DES chip which is described in Appendix B of this report can support a data transfer rate of approximately 1.3 Mbps when used in block encryption mode using 64 bit blocks. This is sufficiently fast to permit this chip to be used in the encryption switching delivery network.

III.4.12 Summary

It appears that "encryption-switching" in conjunction with local area network techniques may provide a very practical alternative to conventional circuit- or packet-switching for the delivery of Telidon service on CATV networks. It minimizes the impact on existing CATV equipment while providing excellent incremental expansion capabilities.

We feel the error rate on the CATV network (mostly due to RF ingress noise) may be a very significant problem in using local area network techniques. This opinion is based largely on the work of Cableshare Engineering in establishing their two way cable system in London, Ontario [ALL079]. This problem requires experimental investigation.

Finally, we wish to thank Gordon B. Thompson for a comment which led us to the idea of encryption-switching.

IV. Database Architectures

Before we begin to design a database architecture, it is important to identify the criteria by which it will be judged. Only with those criteria in mind, will it be possible to determine the relative merits of alternative designs and to choose one which will be suitable for information retrieval from a Telidon-based information provider.

The system must be designed for user convenience; especially, naive users must find it easy to locate information. Whether for the home or for a business, a computer-supplied information system will only be acceptable if it is at least as convenient as conventional systems. Users will be reluctant to learn machine-imposed data organizations and will be resentful of machine-imposed restrictions on access. It is important to realize that many of the users will access the database(s) in a casual manner and only occasionally; for these users, in particular, the system must be natural, in order that access strategies are not quickly forgotten. At the same time, the access commands required of users must be concise so that relatively sophisticated users are not hampered by unnecessary verbosity.

Response time is a second characteristic which is observable by Telidon subscribers. Interactive users find delays of longer than five seconds very frustrating, and even shorter delays seem intolerable if responses are

sometimes much quicker. Thus subscribers will expect immediate responses to requests which they perceive to be simple, but, above all, they will demand consistently-timed responses to all requests. Service providers may be more tolerant of system delays, because they perceive their system resource demands to be larger. As well, they expect eventual financial (or other) gain, so they may accept some hardships in preparing their material. However, their tolerance of delays will certainly not be unlimited, and, once again, the consistency of the response time will be crucial.

Flexibility is certainly very important for the Telidon system, since this is the period of its infancy. The user community is likely to change quite dramatically over the next decade, and, as a result, the system services must evolve to provide more diverse information and facilities in more sophisticated environments. The diversity, together with an unpredictable future, requires a very flexible and adjustable database architecture.

Another concern, especially in a business environment, is reliability. If office automation is to be widely accepted, computerized record-keeping must be as safe as paper-based books. This means that the database must be reliable in spite of hardware, software, and user failures. Furthermore the data and the data manipulation routines must be perceived to be secure against unauthorized access and

improper modification. Without such integrity, users will not be prepared to trust the database system.

Finally, financial cost is another important criterion. The cost to the subscriber is directly related to the costs incurred by the service providers and the system operators. As well as the costs involved in providing a flexible, reliable, and fast database system, several other aspects are involved. Thus the costing of a database architecture must include consideration of storage costs (both primary and secondary memory requirements), processor and other hardware costs, software development and maintenance costs, and database provider costs [TOMPA81b].

In Section IV.1, we will examine alternatives to the currently used page access techniques. In particular, we will extend the notion of menu and demonstrate how a menu-based interface can be designed so that the users need not be aware either of numeric page identifiers or of the underlying tree structure of pages.

In Section IV.2, we will examine various representations for maintaining Telidon databases. A general model for a Telidon file structure is presented and this is followed by the results of some preliminary investigation into storage space and response time requirements of alternative file structures.

IV.1 Design of a Menu-Based Interface to a Telidon Database

IV.1.1 Introduction

Fox has claimed that information consists of unrelated units that must be structured to yield knowledge [FOX78]. Such structuring is apparent to users in the form of directories through which the information can be retrieved.

The current Telidon data structuring (as well as that of all other current videotex systems) takes the form of a hierarchical directory which corresponds very closely to a book's table of contents. Such hierarchies are common in classification systems, for example those found in the biological sciences, as well as in course catalogs, department store directories, and newspapers' classified sections. The Propaedia of Encyclopaedia Brittanica 3 and the body of a thesaurus are also each organized by "fields of knowledge" which may be viewed hierarchically. Unfortunately most bodies of information can be structured into several incompatible hierarchies, and it is often difficult for a user to locate information in another individual's classification scheme. Thus, even if the system response time is suitably quick for each probe into the hierarchy, the location of a particular piece of information may require many probes (including searches along false paths) and therefore have an intolerably long effective response time [PHIL81].

By examining the index structures that are now utilized in manual directories, some ideas for database directories may emerge. For example, most manual retrieval systems are alphabetically ordered or rely on an alphabetically ordered index: telephone directories, dictionaries, and encyclopaedias are alphabetically ordered, whereas library catalogs, road maps, and thesauri have alphabetical indices. Such indices have been incorporated into both Germany's and Austria's Bildschirmtext systems, with minor variations. News items, on the other hand, are perhaps best organized chronologically. As indicated by Parkhill, it is desirable that a videotex directory be, in fact, composed of many sub-directories [PARK79], but, in addition, each sub-directory may be organized as best fits the subject.

A major problem to overcome is the restricted amount of text that can appear simultaneously on a Telidon screen. Most manual classification schemes, and certainly most sequentially ordered lists, rely on a great number of choices at each level of a directory. For example, a list of special consultants for The American College Dictionary [RAND64] consists of 38 topics (such as anthropology, astronomy, chemistry, heraldry, law, and wines, spirits, and beers) arranged in alphabetical order, with each topic having between 1 and 34 entries, again arranged alphabetically. The videotex text restrictions of 20 lines with 40 characters per line does not lend itself to provide

this same structure. As a result, further levels of directory are introduced, and thus the user is required to search a deeper hierarchy than would be needed in a manual system.

Appendix C contains a copy of a paper written under the support of this contract [TOMPA81a]. In that paper, there are two main claims regarding the structuring of Telidon pages: the underlying tree should be eliminated, as its usefulness is outweighed by its apparent restrictions and resulting confusion to users; and page identifiers should be eliminated from all users' commands. The first of these points needs no further elaboration here. However, the second has been greatly refined since the writing of that paper.

In this section of the report, we first summarize the options available in providing menu-like indexes. Subsequently we demonstrate how a suitably rich menu structure can be used in lieu of numeric page identifiers, and we therefore argue that such identifiers should be made unavailable to all Telidon users (subscribers and service providers). Finally we summarize further work that should be undertaken to improve user access to Telidon's data.

IV.1.2 Options for menu design

The Telidon database, like those for all of today's videotex systems, is based on an underlying tree structure of pages navigated by means of menus that provide access paths [TOMPA81a]. In this section, we highlight some of the ideas presented in a recent report [BALL81] and augment them to yield a convenient framework for classifying generalized menus. In particular, a menu can be categorized by five aspects: breadth, labeling, scope, community, range, and mutability.

The breadth of a menu is the number of choices available. For some applications, the breadth can be as few as one or two ("press > to continue" or "enter yes or no"). At the other extreme the breadth may be arbitrarily large; for example, the set of all Telidon page identifiers constitutes an extremely broad menu. Most of today's videotex systems are limited to providing single digit menus (exceptions being Teletel/Star and Waterloo Telidon), thus restricting the breadth for most pages to ten. Whereas the Prestel and derivative systems also seem to be able to provide broader menus, they do so by interrupting the display of intermediate menus as soon as another digit is entered (typically before the display has even started). It is important for menu designers to note that the broader the menus, the less page accesses will be required to reach a particular page; but the display of valid menu selections (if required) will restrict the breadth in practice.

The labeling for a menu is the vocabulary of valid selections. Today's systems typically restrict all labels to be numeric. With the provision of more sophisticated input devices, this can be replaced by increasingly more expensive options, from alphanumeric labels, to pointing (e.g., by using a "mouse"), to perhaps voice. The role of alphanumeric labels will be elaborated in the next section.

The scope of a menu is the set of pages (and related videotex services) from which the menu selections are valid, i.e., from which a user directive could be interpreted as a selection for this menu (cf. "scope of names" in computer programming languages). For example, the scope of one of Telidon's single digit menus is exactly that one page with which that menu is associated, whereas the scope of the menu of all page identifiers is the set of all Telidon pages (since a page identifier can be entered meaningfully everywhere in the database). Because these two extremes are common in videotex they will be called "page-specific" and "page-independent" menus throughout the rest of this report; other scopes within the spectrum will be described as needed.

The community for a menu is the set of subscribers who may use the menu to retrieve pages. The menus in most videotex systems are required to have user-independent (i.e., universal) community; the role of private menus will be discussed in the next section.

The range for a menu is the set of potential target pages. In the first release of the Telidon database system, cross-links were not constructable, thus restricting the range of each page-specific menu to the immediate descendants in the tree. Although the Telidon system now includes all pages in almost every menu's range, today's Telidon does still contain a form of limited-range menu: the set of labels starting with a period (e.g., ".12") constitute a menu whose scope and range consist of the pages within one document only. As security and integrity become more relevant to Telidon systems, as they certainly will, the ability to restrict the ranges of menus judiciously will become increasingly important.

Finally, a menu's mutability is its mode of creation, alteration, and termination. Of concern here is who (system operator, service provider, or subscriber) is empowered to create or modify a menu, as well as when (e.g., at database creation time, at page update time, or at query time) the menu is created or updated. In the next section, we will show the value of having highly dynamic menus that are under the complete control of subscribers, in addition to menus under the control of the system operators and service providers.

IV.1.3 Data access without numeric page identifiers

There are several objections to the widespread use of numeric page identifiers to designate pages. Perhaps the most important one is that their non-mnemonic nature makes them highly error-prone. It is difficult for users to remember random sequences of digits accurately, especially in light of people's present burdens of telephone numbers, personal identification numbers, house numbers, etc. In addition, the accurate entry of such sequences is also a problem. As a result, unless some redundancy is built into the numbering scheme, thus lengthening the sequences, there will be a likelihood of input errors resulting from mistaken page identifiers.

A closely related problem is that of providing directories of page identifiers. In the same way that telephone numbers are recorded in printed directories as well as in private personal listings, page identifiers will have to be made available to users in some form. It is easily forseen that a user would rely on a published catalog to find a page number for a specific service provider or refer to a hand-written list to find the number for a personal favourite (e.g., games) page; such trends are already evident in Canadian field trials as well as in other nations. The cost of maintaining such directories and their rapid obsolence are certainly significant. Furthermore, the inconvenience (and absurdity) of requiring access to a printed medium

before being able to access an electronic one is certain to be detrimental to videotex.

A third problem arises if the numeric identifiers are in fact page addresses, as they are in Telidon's database (the identifier designates the location of the page with respect to the underlying tree). For example, a copy of the Department of Communication's demonstration database has been incorporated in a subtree rooted at page 9 of Waterloo's Telidon database; as a result the page numbers displayed on pages (for identification or cross-linking) are all invalid. Experience in other areas of computer science has shown that exposing internal addresses to users has a disastrous effect on system maintenance and improvement. Consider, for example, the use of absolute addresses in programming languages, for example: their use for control flow has been replaced by the provision of symbolic statement labels, and their use for data linkage has been replaced by references via variables. Similarly database researchers have long advocated the separation of internal from external "views" of the data, this being evidenced recently by the removal of numeric record "keys" from end-user languages. Finally, instances of a Telidon user's pages "disappearing" just before an important demo, because the database was reorganized, are all too well-known.

In light of these problems, there is little reason to restrict subscribers to numeric labels in the form of page

numbers. It has been claimed that many users would be intimidated by larger keyboards, but the widespread use of hand-held calculators with many buttons and of television channel convertors with many switches may indicate that subscribers prefer functionality and flexibility to over-simplicity.

If numeric page identifiers are to be removed from videotex systems, one must first identify the roles which they play in current systems. The principal use is for advertising: service providers want their root pages to be readily accessible by subscribers, and they also want to provide special pages (e.g., a game or a contest) which will attract new users to their services. A second role is for users, to recall a page containing potentially updated information on a topic of particular interest (e.g., gold prices), the page number being noted in some previous videotex session or earlier in the current session. Finally, identifiers are used by service providers to place pages within the underlying tree structure and to create the cross-link maps which provide alternative access paths. This third point will be addressed in the next section.

A naive solution to the problems of numerical identifiers is merely to replace them throughout the database by alphanumeric ones. Several shortcomings quickly become apparent. A principal concern is that the extra length needed to make identifiers mnemonic makes them more cumbersome and

error-prone, particularly if they were to be used in place of single digit menu identifiers. Furthermore, the requirement of uniqueness in page identifiers together with the vast number of identifiers required, would encourage identifiers that were not necessarily indicative of page contents; labels such as "gldprcs2" are barely better than "3Ø56112". Finally the particular choices, of identifiers would not necessarily be easier for subscribers to recall, thus necessitating large directories.

The first step to solving these problems results from the realization that most pages are not designed to be accessed directly when first used by a subscriber. In fact, each service provider has relatively few "entry" pages suitable for first-time users, and only these pages need to have labels in a universal page-independent menu. A subscriber would therefore typically encounter a set of pages from one of several designated roots and then proceed to access other pages by more localized menu selection. The resulting reduction in size for the page-independent menu greatly relieves the problems caused by over-labeling, particularly non-mnemonic labels, unnecessarily long labels, and conflicting requirements for the use of identical labels.

As mentioned in the previous section, mnemonic labels are most suitable when a menu is large. Whereas the set of all entry pages constitutes such a large menu and therefore

benefits from alphanumeric labels, most page-specific menus are relatively short. It is absurd to disallow alphanumeric page-specific menus if subscribers have full keyboards; but it is equally absurd to insist that all menus have mnemonic labels. In most cases, the advantages of short labels will dictate single digit or character labels for page-specific menus. Occasionally page-specific menus may be large enough to warrant the use of longer mnemonic labels, especially if the choices need not be displayed (e.g., "type another country's name for related information").

The menu structures as described so far satisfy the needs of service providers to advertise certain pages so as to attract new subscribers. However, a menu structure that aids users in subsequently recalling pages of interest is still required. If many pages do not have entries in a page-independent menu, most subscribers will become frustrated at repeatedly traversing certain access paths to retrieve favoured pages. For example, a subscriber's tastes may dictate daily access to a sports summary, a particular team's personnel summary, the grain futures, the joke of the day and the list of flights from Halifax to Quebec; many of these pages may not be popular enough to be included in a universal page-independent menu. Thus, the Telidon database system should provide a mechanism for each subscriber to maintain a private page-independent menu (cf. "labels" in the Hypertext Editing System [VAND71]). The subscriber must

have complete control over the entries in the menu and must be able to enter and delete labels online. In the Waterloo Telidon, for example, the command "+flights" would enter the label "flights" in the subscriber's personal menu, designating the currently viewed page as the target.* Thereafter, the menu selection "flights" from any page, when issued by that user only, would recall that target. Finally, the command "-flights" deletes the entry corresponding to that label from the subscriber's menu. At no time does the subscriber need to know the numeric page identifier.

Such private menus have many advantages. First, the universal menu can be kept small, since each subscriber can essentially augment the page-independent menu to customize it without affecting other menus. Of equal importance is the fact that each subscriber can use labels that are personally meaningful and therefore much easier to remember. (Of course, a subscriber could also choose short, cryptic labels if that were personally preferable.) Finally, the system could provide a mechanism to traverse the pages labelled in the private menu, thus further simplifying retrieval for repeatedly posed patterns. (The Waterloo Telidon system, for example, uses ">>" and "<<" to traverse the private menu linearly.)

As a slight extension to this idea, it may also be useful to provide a private menu of more transient labels.

---

* Actually a question mark is used rather than a plus, to accommodate Electrohome's primitive keyboard.

Every videotex system includes the capability to retreat along an access path (by using the "oops" key), but this is inadequate for recalling a page noted much earlier in the session. Pages that are deemed temporarily notable could be entered into the subscriber's private menu, but they would have to be culled periodically by the user. A simpler mechanism would be to allow a subscriber to enter a label that will be deleted automatically when the session terminates; for example, the command "+flights temp" could be used. This has an advantage over a simple stacking mechanism, in that a "retreat" is not required to access every noted page in reverse linear order.

When multiple menus exist simultaneously, a policy must be adopted to resolve ambiguity when the menus' scopes intersect. In the DOC Telidon database system, this is accomplished by syntactically distinguishing labels in each of the three menus accessible from a page: single digits and special codes (e.g., '>') refer to the page-specific menu, a period followed by a digit sequence is a label in the document menu, and all other labels refer to the page-independent menu. Alternatively, in Waterloo Telidon, where there are very few syntactic restrictions to labels, the ambiguity is resolved by resorting to a fixed strategy for searching menus: the page-specific menu is searched first, then the subscriber's private menu, then a subtree's menu (if such exists), and finally the page-independent menu. A

third alternative would be to search all menus having the current page within their scope, resolving an ambiguity by requesting further discriminating input from the user.

These three approaches are increasingly liberal in interpreting a user's command, and the third may therefore seem to be the most forgiving method. On the other hand, a misspelled label in either of the last two options may accidently match an entry in an unexpectedly applicable menu, thus confusing naive users. In spite of this, however, we currently prefer a method that avoids giving users error messages; we look forward to the results of the necessary behavioural experiments.

Finally, for more rapid page traversal by sophisticated users, a concatenated sequence of menu selections should be interpreted as if they were entered one at a time (as in Prestel and Bildschirmtext). Thus, for example, "AC.schedule.Montreal" identifies the same page as would be retrieved by entering the three individual labels consecutively.

IV.1.4 Page frame maintenance without numeric identifiers

In the previous section we described a menu structure which provides all the necessary facilities for page access without the explicit use of numeric page identifiers. Service providers can also benefit from alphanumeric page labelling when updating the database.

In this section we concentrate on the aspect of maintenance that updates the linkage between pages rather than the (PDI) display codes that constitute a page's contents. We therefore refer to a page frame as that part of a page which does not deal with display codes, but rather includes all the relevant interpage linking.

The first observation is that a primitive menu maintenance tool has already been described for manipulating a subscriber's private menu. Those facilities need to be extended to include the extra capabilities required by information providers, such as creating new pages and specifying which of the accessible menus is to be updated. Throughout this section we will adopt standard list-processing concepts and facilities, as they would appear in a high-level programming language.

When creating a new page, page frame manipulation should be separated from page contents manipulation. In particular, a "blank" page frame can be allocated from a pool of available pages and linked into the service provider's access paths dependently of filling it with display codes. We therefore assume a facility for copying prepared display codes into a "current" page frame and omit the detailed specification of such a command.

In all programming languages, when a new piece of storage is allocated, its address must be noted by the system and saved in a variable accessible by the user.

Similarly the identifier for a newly-allocated page frame must be stored in a menu, for example the service provider's private or temporary menu, some page's page-specific menu, or the universal page-independent menu. For consistency with the commands adopted by Waterloo Telidon, we suggest as an example the syntax summarized in the following table (in all cases the phrase "clean-page" is used to indicate that the label "flight" should subsequently refer to a newly allocated page):

Command	Menu Affected
"+flight clean-page"	private
"+flight private clean-page"	private (alternative syntax)
"+flight temp clean-page"	private, temporary
"+flight public clean-page"	page-independent
"+flight to clean-page"	page-specific (current page)

Naturally pages will need to be accessible via multiple access paths. Therefore a facility is also needed to enter labels having existing pages as targets (in all cases, the label "flight" will subsequently retrieve the page designated by the label "schedule" in the current page's context):

Command	Menu Affected
"+flights private schedule"	private
"+flights temp schedule"	temporary
"+flights public schedule"	page-independent
"+flights to schedule"	page-specific for current page

For symmetry with the last option, the command "+flights from schedule" should denote a change to the page-specific menu for the page labelled "schedule" such that the label "flight" therein refers to the current page; that is, a back link is to be created.

Finally two commands are required to tear down structures. The command "-flights" has already been introduced to delete the label "flight" from a private menu; this is merely extended to delete the label from that menu in which it is found (such that the menu's scope includes the current page, of course). In addition, page frames should be returnable to free storage for subsequent re-allocation; to this end, the command "destroy page" can be included to remove the current page.

The complete label manipulation language is summarized in Figure 4.1, using an extended form of BNF [GRIES71]. Because the facilities have been commonly used in programming languages and they are relatively simple, they are likely to be successful in videotex as well.

```
command          := retrieval
                  | update
retrieval        := path-descriptor
path-descriptor  := name {'.' name}*
name             := {letter | digit}+
update           := '+' name [[affected-menu] target]
                  | '+' name 'from' path-descriptor
                  | '-' name
                  | 'destroy page'
affected-menu    := 'private' | 'temp' | 'public' | 'to'
target           := path-descriptor
                  | 'clean-page'
```

Figure 4.1 - Label manipulation commands

IV.1.5 Further related work

As is true for all such studies, there are as many interesting new prospects raised as there are problems resolved. In this section, we briefly outline some investigative work that should be carried out as extensions to the research described here.

Dynamic menu maintenance and concurrency control

The menus described in this section may all be envisaged as being created and maintained as the update commands are given. However a menu's mutability, as described earlier, may involve alternative times. These alternatives should be investigated to determine their applicability to each form of videotex menu.

Most Telidon pages are currently edited off-line, and the database is updated in a subsequent batched run. This mode of operation lends itself to a batched update for menus as follows. The update commands are encoded and embedded in (or attached to) the pages to which they apply (typically the current page at the time the update command is entered). When the batch update is run, these commands are extracted, and the menus are altered as indicated. This mode of operation has the advantage that the menus are always synchronized with the pages they reference [GREI81]; therefore users are not presented with "dangling references" and their associated error messages.

A second alternative is to create a displayable menu only when it is requested, as is now done for the "list" command in Waterloo Telidon. In this mode of operation, a menu is stored in a compact, internal form during most of the time; a display page of (PDI) codes is created at the time that it must be presented to a user. For example, a menu can be dynamically created whenever an ambiguous label is entered so that the user can choose which alternative is desired. The advantage of this mode of operation is to save space for menu storage, at the expense of occasional menu creation time to respond to users' queries. Also, this technique saves time for input people during the updating of menus, since they only update the compact, internal form of the menu from which the system generates the menu page.

A problem that arises when updating is permitted online is the synchronization of concurrent updates. For example, if two service providers request that the universal, page-independent menu use the label "flights" for two distinct pages simultaneously, the system must ensure that at most one of them is granted. One alternative is to restrict service providers to choose labels from distinct vocabularies (e.g., Air Canada must prefix all labels with "AC"), but this solution is impractical and still does not solve the problem of synchronization among employees of a single service provider. A better solution is to employ a standard concurrency control technique (see, for example, [HABE76]).

In particular, the synchronization monitor deserves further investigation for videotex.

Keyword processing and forms utilization

The research reported here has concentrated on providing user facilities for retrieving pages in a navigational manner, i.e., one page requested at a time. Other videotex work has concentrated on so-called "keyword" access to pages, where a user request is satisfied by a set of pages to be displayed in some manner [BALL81]. The work described here should be extended to be considered in the light of multiple-page retrieval.

In fact, the marriage of menus and more traditional keyword systems is perhaps not very difficult to achieve. When describing ambiguous labels, it was proposed that a menu be created dynamically to allow a user to choose which alternative is desired; this facility can be extended to encompass situations where many pages are potential responses to a request. Some research must be pursued to determine how best to display a very large number of choices conveniently.

An aspect that has not been pursued at all here is that of combinations of labels in a single request. Such a user language is typified by Boolean expressions of keywords, but this is not the only mode of multi-label request. For example, a "form" may be used as an inquiry tool in which a user may specify several constraints on the information or

service desired; the system response to such a form may be one page of videotex data or a set of pages containing related information [TOMPA81a]. Research into form creation, maintenance, and use may be vital to videotex's acceptance for non-trivial applications.

Facilities for maintaining clusters of pages

As a further extension for data organization and querying, research should be conducted to determine whether sets of pages containing related information can be managed as a unit. The concept of "abstract data type" [LISK72], for example, lends itself to defining a set of pages and the valid operations that can be applied to that set. Adapting such an approach to videotex would allow structures such as rectangular grids, wheels of wheels, and arbitrarily many others to be incorporated into Telidon as conveniently as the tree structure is now an integral part. Such structures can then be used to store sets of pages in the database or to present sets of pages to users as they are dynamically collected in response to (purposefully) ambiguous requests.

Interfaces to externally controlled databases and services

There is much work to be done to integrate a videotex database with other databases in order to expand the range of information and services. Modes of distributed videotex can be classified as follows:

o compatible videotex systems with a shared name space

(A shared name space means that if multiple systems each contain an identical page, it will have the identical name on all systems.): Each physically realized database contains a subset of a conceptual universal database of videotex pages. Thus a request for a page can easily be satisfied by searching for that page in every database, although a more disciplined search strategy is certain to be cost-effective.

- compatible videotex systems with distinct name spaces (Distinct name spaces means that if multiple systems each contain an identical page, it can have a different name on each system.): Each physically realized database contains a subset of the pages from a conceptual universal database of videotex pages. A request for a page may be satisfiable at a remote site, but that request may need to be transliterated to use names particular to each database accessed.

- videotex systems with incompatible naming (Incompatible naming means that each system uses not just a distinct name but also a different naming technique. For example, one system may use labels which uniquely identify pages while another system may allow the user to retrieve multiple pages using key-

words for selection.): Each physically realized database contains a subset of the contents of a conceptual universal database of videotex pages. Unlike the previous category, however, a request for a page at one site may need to be completely reworked into a set of requests at a remote site; thus requests must be processed by an arbitrarily sophisticated translation mechanism to convert names from one site to be meaningful at another.

- o heterogeneous database systems and services: The contents of each physically realized database are not necessarily drawn from a conceptual universal database. This is the most general form of distributed database, in which videotex may be combined with record-oriented, bibliographic, statistical, and other database facilities as well as other data processing services (such as conventional programming languages). Sophisticated protocols and interfaces need to be developed to present a homogeneous query and manipulation language to users [BOCH81].

Behavioural studies

The techniques presented here have all been shown to be useful in various contexts, but they are novel for videotex. Before incorporating them into production videotex systems,

it is important that they be evaluated with respect to user convenience.

For example, the language described for manipulating labels in menus is a fairly conventional control language for computer systems. It would be surprising if it were the ideal syntax to be adopted for users who have had no experience with such systems. Only by careful experimentation can it be determined whether the concepts are reasonable for naive users and what sort of language would be most appropriate.

IV.2 File structure alternatives

The representation of pages and their retrieval access strategies are critical to the performance of videotex. In this section, we discuss our research into alternatives to the current Telidon database system implementation. This section is organized as follows: a description of a generic model for videotex file structures and its application to the current DOC implementation; an overview of appropriate search strategies to find the page associated with a given numeric identifier; an overview of page cache strategies; a description of benchmark studies and results; and conclusions and recommendations for videotex file structure design. Appendix D contains a copy of the Pascal code used to implement the simulation program whose results are explained in Section IV.2.4.

IV.2.1 A videotex file structure model

For the purpose of this study, we consider the use of videotex for information retrieval only (as distinct from transaction invocation, for example). Furthermore, we restrict ourselves to the user interface provided by the current DOC implementation; pages can be requested by numeric page identifier, by a numeric menu selection having up to ten choices, or by a tree-oriented directive (e.g., parent of page, next sibling, child, etc.). The incorpora-

tion of labels, as they are included in the Waterloo Telidon server, for example, is not included in this study.

Consider the model diagrammed in Figure 4.2. A user can enter a page identifier, menu choice, or tree-oriented directive. Thereafter a page retrieval strategy must be invoked in order to make the appropriate page available to the user. Associated request handlers scan for the requested page in the cache (via the cache manager), and if it is not there, they invoke a page retrieval from secondary store (assumed to be disks) by passing the page's (disk) address to the secondary store manager. In case of a page request by numeric identifier, a search strategy typically needs to be invoked first to find the disk address of a page.

The overall page retrieval strategy is as follows. In case of a menu-based or tree-oriented selection, the disk address is assumed to be stored with the page previously retrieved. Because each user can have at most one active page, this involves the retention of ten to twenty or so addresses per user (one for each menu choice and one for each neighbour of the active page in the tree), a storage overhead well worthwhile in view of the processing saved when subsequently retrieving those pages without invoking the search strategy. Thus first the cache can be searched for the desired page, and, in case it is not found there, the disk address can be passed to the secondary store manager for retrieval. If instead a numeric page identifier is used

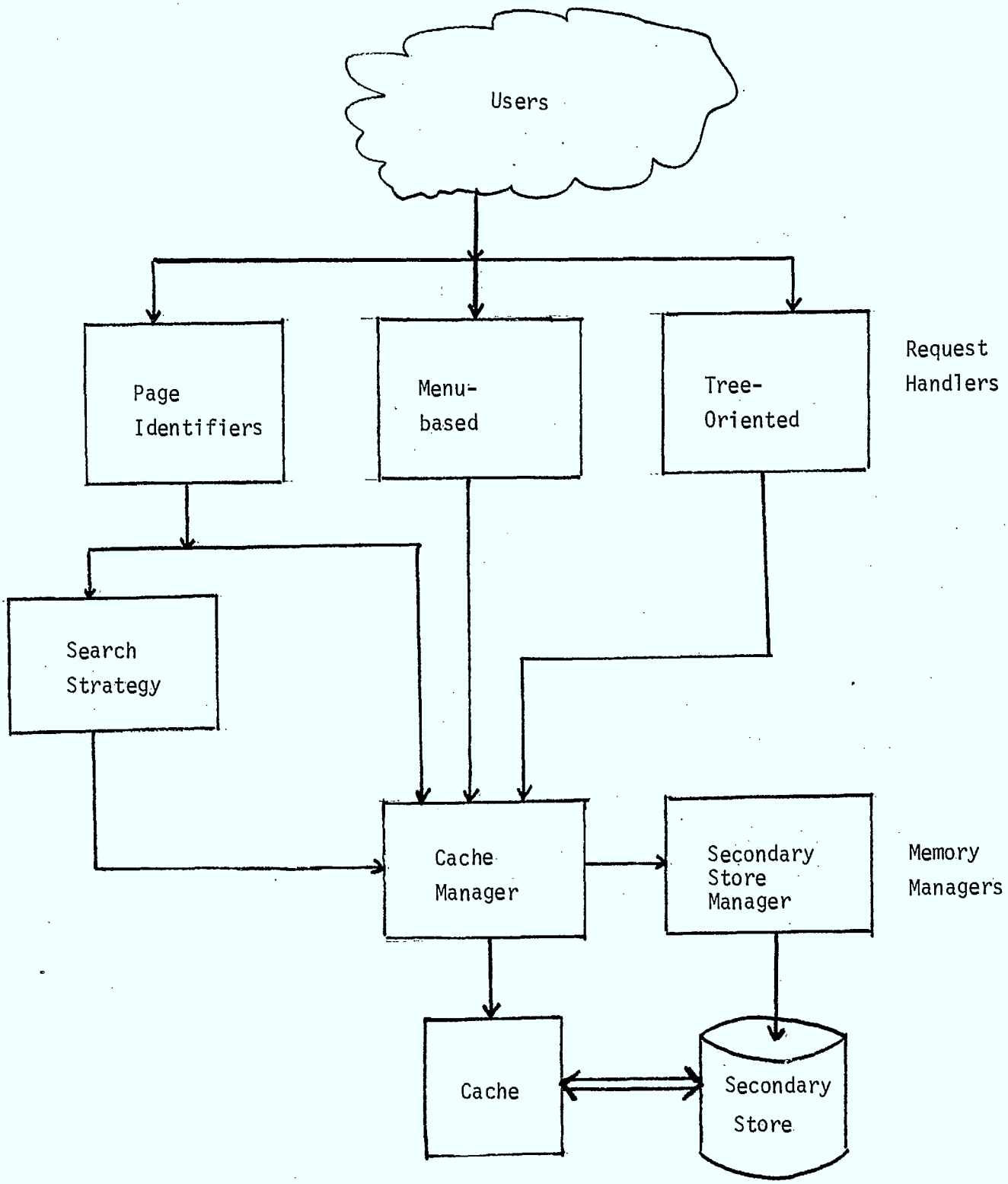


Figure 4.2: Videotex File Access Model

to select a page, the cache can again be searched first for the page. Subsequently the cache can be searched for a neighbour of the page (the disk address of the requested page would then be available as one of the retained links in that page). Failing that, the search strategy would be invoked to find the disk address for the requested page in a large dictionary-like table. Because of the size of this identifier-to-address dictionary, the search strategy itself may involve the retrieval of "meta-pages" from the cache and from secondary store.

This model can be applied to the current DOC implementation. Because there is no cache, the system behaves as if the cache manager always reports that the page is not in the cache. Thus, for example, menu-based and tree-oriented requests always pass the disk addresses of required pages to the secondary store manager. Similarly, page requests using numeric page identifiers are not passed to the cache manager first, but instead the search strategy is immediately invoked. The DOC search strategy uses four B-trees (one for low-numbered pages, two for news-of-the-day pages, and one for all other pages) implemented as four indexed files under Digital Equipment's RMS. In general, the search strategy retrieves several meta-pages corresponding to nodes in the B-tree, and finally it retrieves the requested videotex page, always from secondary store.

IV.2.2 Alternative search strategies

There has been a lot of activity in the design of algorithm and data structures for searching [KNUT73,GONN81]. In our context, we will limit ourselves to study search algorithms with the following properties:

(a) the dictionary (albeit for pages stored in the cache) resides on secondary store, and hence retrieving its pages is the most important cost of the search;

(b) searches will be much more frequent than update operations (additions, deletions, changes);

(c) search operations are restricted to be equality-searches (as opposed to range-searches, etc.);

(d) most of the searches are expected to be successful.

With these constraints, we rapidly reduce the set of interesting search strategies to three families:

- o B-tree type indices,
- o digital trees
- o hashing schemes using buckets.

The best candidate to succeed in the first group is the most general form of the B-trees [COME79]. The search time is short and uniform for all elements, and the node size can be easily adjusted to conform to convenient I/O blocks.

In the second group the TRIE [FRED60,KNUT73] is a prime candidate. Since the key on which we will search is most likely a Telidon page number, we have a natural "alphabet" to consider for branching at each level.

Hashing into buckets generally shows better performance than trees. There are some drawbacks in hashing (bad worst-cases, difficulty in expanding the table, etc.) which have been overcome in some recent new algorithms [LARS78,FAGI79].

IV.2.3 Cache strategies

The primary goal of the cache is to reduce the number of accesses to the secondary store. A cache mechanism is effective when the requests follow some routine or bias, and it is of little use for a sequence of random requests. Requests for pages in videotex are far from random (root and search pages will be requested repeatedly, some sequences of pages will typically be scanned sequentially, etc.). However the exact distribution of the requests for pages is not known, and it may change with time. The best approach for this study is therefore to simulate the cache strategies with recorded sequences of requests and according to this, select the best parameters.

Some possible variations in the cache strategies are to alter

- o the replacement algorithm
- o the size of the cache
- o the locality of transfers (as described below)

When a page is requested by a user and it is not in the cache, it is typically necessary to dispose of some other pages in the cache. The method of selecting the pages to be

replaced is called the "replacement algorithm". Such algorithms have been studied extensively in operating systems [SHAW74,COFF73], for which the strongest candidate is the "least recently used" algorithm. Some other candidates include "first in first out" and weighted algorithms (in which, for example, pages are replaced according to their number of accesses and length of time in the system).

We know that the larger the cache, the fewer accesses to secondary store will be required. In this case we are interested in estimating the best trade-off between the cost of the memory used by the cache and the speed-up obtained as a result.

The third strategy is related to the organization of the pages as well as the operation of the cache. The motivation comes from the fact that an input operation may read more information (a physical block) than was requested by a user. If related pages (e.g. next, descendants) are stored contiguously in the file, we may decide to keep all the information read in the cache, so that the user's next request may find its information without requiring an access to the secondary store.

These options should be studied together rather than independently, since it is very likely that the trade-offs are interrelated.

IV.2.4 Simulation of a Telidon server with cache memory

The most important measure of complexity in a Telidon server is the input from external devices, in this case the number of data-pages, the number of index-pages and the total number of characters read. The choices for the internal data structures of the server are selected in the simulation by setting the following parameters:

- o the size of the cache, or internal memory, used to store frequently used pages
- o the algorithm for replacing pages in the cache
- o the index method used to retrieve pages given solely by their number (absolute requests)
- o the characteristics of the index records (most notably size and load factor)
- o the size of the Telidon database (number of data-pages)
- o the distribution of the page sizes
- o the statistical distribution of types of requests
- o the correlation between types of requests

The first group of parameters are choices that the implementor of a Telidon server has available to tune the system to best performance. The second set of parameters describes potential environments and provides tools for judging the stability of the system under different conditions. Values for this second set of parameters may be obtained from a trace of experimental or real data.

The interesting combinations of parameters are endless. We selected a few combinations, differing from each other in one parameter at a time. This allows us to assess the effect of each choice.

Runs were made against the trace data provided by DOC and against a random trace of 1000 requests.

The "benchmark" server is as follows:

- The Telidon database consists of 50,000 pages.
- The pages have size normally distributed with mean 800 characters and standard deviation 250 characters.
- The index is a B*-tree, each node with maximum size 50 entries, 400 characters. It is assumed that the B*-tree has been optimized so as to guarantee that it has 80% occupation of its nodes. This gives an average branch factor of 40 and a B*-tree of 3 levels.
- The internal cache is 20,000 characters in size. The cache will store either data-pages or index-pages.
- The replacement is done on the page for which the rate of utilization per character is lowest.

The output of each simulation is grouped into three sections: (a) the model description which summarizes the characteristics of the database and the implementation decisions, (b) the input statistics that describe the charac-

teristics of the user population, and finally (c) the performance statistics which give the simulation result of the given input on the given model. The following comments apply to all simulation results:

- The "Disk requests per page requested" value probably is the most interesting measure of all. It will certainly regulate execution time for I/O bound systems.
- The "Percentage of requests satisfied within the cache" measures the cache effectiveness, individually for data-pages and index-pages.
- The "total data-characters read from disk" gives another important measure of the time required for I/O.
- The "average number of entries in cache at replacement", "number of replacements in the cache", and the total of pages (index or date) requested to the cache measure the heaviest of the central processor activities which depend on the parameters.

The results of the simulation on the benchmark server are shown in Table 4.1. The results for the random data are accurate within 1%.

The random sample, in some sense, is the worst data to test the cache, almost certainly giving lower bounds on cache effectiveness. This is due to two facts: (a) the pages requested will not be random: there will be some pages

```
      out1.nob


Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 12
Total number of page requests received  1119
Percentage of ancestors requested   2.50
Percentage of "next page" requested  57.73
Percentage of "previous page" requested   0.98
Percentage of direct page requests  27.08
Percentage of menu-type requests by menu-option
(descending tree)   2.77  2.14  1.61  0.80  0.54  0.36  1.16  0.00  2.32  0.00
(arbitrary link)   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Total time elapsed   4.0200


Performance Statistics --------
Average number of requests per unit time  278.3582
Number of pages requested to the cache   1119
Number of index-pages requested to cache    795
Number of pages requested to disk    874
Number of index-pages requested to disk    188
Disk requests per page requested 0.9491
Percentage of requests satisfied within the cache
  pages= 21.89    index-pages= 76.35
Number of data-characters read from disk    704846
Average number of entries in cache at replacement  32.67
Number of replacements in the cache   1031
```

Table 4.1 part (a)

```
Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 10
Total number of page requests received 10000
Percentage of ancestors requested  20.06
Percentage of "next page" requested  19.28
Percentage of "previous page" requested   5.19
Percentage of direct page requests   9.93
Percentage of menu-type requests by menu-option
(descending tree)   9.51  4.05  3.17  2.42  2.39  1.92  1.84  1.72  1.55  1.50
(arbitrary link)   1.87  1.23  1.67  1.47  1.70  1.47  1.57  1.33  1.52  1.64
Total time elapsed  9908.0319


Performance Statistics --------
Average number of requests per unit time   1.0093
Number of pages requested to the cache  10000
Number of index-pages requested to cache   2775
Number of pages requested to disk   8946
Number of index-pages requested to disk    808
Disk requests per page requested 0.9754
Percentage of requests satisfied within the cache
  pages= 10.54    index-pages= 70.88
Number of data-characters read from disk   7279253
Average number of entries in cache at replacement  30.86
Number of replacements in the cache   9722
```

Table 4.1 part (b)

that will be universal favourites, thus increasing the chances of finding them in the cache; and (b) the distribution is even more skewed if time is also considered (e.g. everybody might access news pages at 6:00 p.m.). These considerations improve the chances of finding pages in the cache.

From the "normal" case against the DOC data or the random data, we extract the following conclusions:

- The performance is better for the DOC trace data, as expected. It is quite significant, however, that the effectiveness of the cache for data-pages doubled!
- The effectiveness of the cache for index-pages is surprisingly high (3 out of 4 pages of the index are found in the cache) and not very different between sets of data. This points to one of the most significant aspects of the cache. Once the system is in steady state, the root and the next level of the B*-tree stay in the cache. They are frequently requested, small records; consequently they will tend to stay longer than larger data-pages.
- In both cases a request is satisfied with slightly less than one disk access on average.

Table 4.2 summarizes the simulations using a hashing table instead of a B*-tree. The hashing table for 50,000

pages is quite large (1,250 buckets). Consequently the use of the cache for index-pages will be almost useless, because of the random nature of hashing. When hashing is used then, the cache will not be used for index-pages; it will seldom pay. In this case we note:

- o The cache is slightly more effective for data-pages (since index-pages are not in the cache, it is effectively larger).
- o Fewer index-pages are required, roughly 1/3 of the requests as compared to the B*-tree, although now each index-page request requires one disk access.
- o The performance is slightly worse than for B*-trees, although it is still less than one access to disk per request.

Next we will analyze the effect of the size of the cache. The general tendency is that the larger the cache the better performance. What we want to evaluate is for how long this increase in size is cost effective. In Table 4.3 note that

- o The effectiveness of the cache for index pages remains almost invariant. This results from the fact that even in the small cache there is room for the root and most of the first level. However, if the size were to be decreased further, the effectiveness would drop sharply.
- o The effectiveness for data-pages changes

```
      outl.nohx


Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a hashing table
Number of buckets of hashing table is 1250


Input Statistics ---------
Total number of different users 12
Total number of page requests received  1119
Percentage of ancestors requested   2.50
Percentage of "next page" requested  57.73
Percentage of "previous page" requested   0.98
Percentage of direct page requests  27.08
Percentage of menu-type requests by menu-option
(descending tree)   2.77  2.14  1.61  0.80  0.54  0.36  1.16  0.00  2.32  0.0(
(arbitrary link)   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Total time elapsed   4.0200


Performance Statistics --------
Average number of requests per unit time  278.3582
Number of pages requested to the cache   1119
Number of index-pages requested to cache      0
Number of pages requested to disk    840
Number of index-pages requested to disk    263
Disk requests per page requested 0.9857
Percentage of requests satisfied within the cache
Number of data-characters read from disk    677589
Average number of entries in cache at replacement  27.97
Number of replacements in the cache    813
```

Table 4.2 part (a)

```
Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a hashing table
Number of buckets of hashing table is 1250


Input Statistics ---------
Total number of different users 10
Total number of page requests received 10000
Percentage of ancestors requested  20.06
Percentage of "next page" requested  19.28
Percentage of "previous page" requested   5.19
Percentage of direct page requests   9.93
Percentage of menu-type requests by menu-option
(descending tree)   9.51  4.05  3.17  2.42  2.39  1.92  1.84  1.72  1.55  1.50
(arbitrary link)   1.87  1.23  1.67  1.47  1.70  1.47  1.57  1.33  1.52  1.6
Total time elapsed  9908.0319


Performance Statistics --------
Average number of requests per unit time   1.0093
Number of pages requested to the cache  10000
Number of index-pages requested to cache      0
Number of pages requested to disk   8827
Number of index-pages requested to disk    962
Disk requests per page requested 0.9789
Percentage of requests satisfied within the cache
Number of data-characters read from disk   7187908
Average number of entries in cache at replacement  28.08
Number of replacements in the cache   8800
```

Table 4. 2 part (b)

```
      outl.sm


Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 10000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 12
Total number of page requests received  1119
Percentage of ancestors requested   2.50
Percentage of "next page" requested  57.73
Percentage of "previous page" requested   0.98
Percentage of direct page requests  27.08
Percentage of menu-type requests by menu-option
(descending tree)   2.77  2.14  1.61  0.80  0.54  0.36  1.16  0.00  2.32  0.00
(arbitrary link)   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Total time elapsed   4.0200


Performance Statistics --------
Average number of requests per unit time  278.3582
Number of pages requested to the cache   1119
Number of index-pages requested to cache    843
Number of pages requested to disk    945
Number of index-pages requested to disk    235
Disk requests per page requested 1.0545
Percentage of requests satisfied within the cache
  pages= 15.55    index-pages= 72.12
Number of data-characters read from disk    751629
Average number of entries in cache at replacement  16.70
Number of replacements in the cache   1164
```

Table 4.3 part (a)

```
Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 10000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 10
Total number of page requests received 10000
Percentage of ancestors requested  20.06
Percentage of "next page" requested  19.28
Percentage of "previous page" requested   5.19
Percentage of direct page requests   9.93
Percentage of menu-type requests by menu-option
(descending tree)   9.51  4.05  3.17  2.42  2.39  1.92  1.84  1.72  1.55  1.50
(arbitrary link)   1.87  1.23  1.67  1.47  1.70  1.47  1.57  1.33  1.52  1.64
Total time elapsed  9908.0319


Performance Statistics --------
Average number of requests per unit time   1.0093
Number of pages requested to the cache  10000
Number of index-pages requested to cache   2868
Number of pages requested to disk   9478
Number of index-pages requested to disk   1137
Disk requests per page requested 1.0615
Percentage of requests satisfied within the cache
  pages= 5.22    index-pages= 60.36
Number of data-characters read from disk   7682968
Average number of entries in cache at replacement  15.43
Number of replacements in the cache  10597
```

Table 4.3 part (b)

out1.lg

```
Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 30000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 12
Total number of page requests received  1119
Percentage of ancestors requested   2.50
Percentage of "next page" requested  57.73
Percentage of "previous page" requested   0.98
Percentage of direct page requests  27.08
Percentage of menu-type requests by menu-option
(descending tree)   2.77  2.14  1.61  0.80  0.54  0.36  1.16  0.00  2.32  0.00
(arbitrary link)   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Total time elapsed   4.0200


Performance Statistics --------
Average number of requests per unit time  278.3582
Number of pages requested to the cache   1119
Number of index-pages requested to cache    750
Number of pages requested to disk    787
Number of index-pages requested to disk    172
Disk requests per page requested 0.8570
Percentage of requests satisfied within the cache
  pages= 29.67    index-pages= 77.07
Number of data-characters read from disk    633775
Average number of entries in cache at replacement  48.82
Number of replacements in the cache    913
```

Table 4.3 part (c)

```
cat out2.lg


Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 30000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 10
Total number of page requests received 10000
Percentage of ancestors requested  20.06
Percentage of "next page" requested  19.28
Percentage of "previous page" requested   5.19
Percentage of direct page requests   9.93
Percentage of menu-type requests by menu-option
(descending tree)   9.51  4.05  3.17  2.42  2.39  1.92  1.84  1.72  1.55  1.
(arbitrary link)   1.87  1.23  1.67  1.47  1.70  1.47  1.57  1.33  1.52  1.6
Total time elapsed  9908.0319


Performance Statistics --------
Average number of requests per unit time   1.0093
Number of pages requested to the cache  10000
Number of index-pages requested to cache   2703
Number of pages requested to disk   8559
Number of index-pages requested to disk    752
Disk requests per page requested 0.9311
Percentage of requests satisfied within the cache
  pages= 14.41    index-pages= 72.18
Number of data-characters read from disk   6976480
Average number of entries in cache at replacement  46.03
Number of replacements in the cache   9267
```

Table 4.3 part (d)

drastically. For the DOC trace, the large cache has a very high efficiency (30%) even in absolute terms.

Finally, the variance of the data-pages is drastically reduced, from 250 characters to 100, representing a much more regular file (Table 4.4). In this case there is no noticeable effect. We may safely conclude that within some reasonable range of variability, it is the average size of the page that matters and not its distribution.

In summary, the main conclusions of this simulation are:

- o The "cache" concept is very appropriate to reduce the I/O traffic and even a small cache proves to be effective.
- o With a cache, B*-trees and hashing tables are equivalent. Without the cache the hashing scheme will be superior to B*-trees.

IV.2.5 Further aspects for investigation

Additional simulations

The simulation studies presented here constitute the beginning of an investigation into alternative file organizations. The model on which the simulation program is based is very general, and therefore it is appropriate for testing many hypotheses.

```
      out1.dv


Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 12
Total number of page requests received  1119
Percentage of ancestors requested   2.50
Percentage of "next page" requested  57.73
Percentage of "previous page" requested   0.98
Percentage of direct page requests  27.08
Percentage of menu-type requests by menu-option
(descending tree)   2.77  2.14  1.61  0.80  0.54  0.36  1.16  0.00  2.32  0.
(arbitrary link)   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.0
Total time elapsed   4.0200


Performance Statistics --------
Average number of requests per unit time  278.3582
Number of pages requested to the cache   1119
Number of index-pages requested to cache    807
Number of pages requested to disk    897
Number of index-pages requested to disk    192
Disk requests per page requested 0.9732
Percentage of requests satisfied within the cache
  pages= 19.84    index-pages= 76.21
Number of data-characters read from disk    715958
Average number of entries in cache at replacement  30.38
Number of replacements in the cache   1060
```

Table 4.4 part (a)

out2.dv

```
Model Description --------
Total size of data-pages in database  50000
Size of internal cache memory (chars) 20000
Size of index-page (chars) is  400
The index occupation factor is 0.80
Average number of entries per index-page is 40
The index is organized as a B*-tree
Height of B*-tree is 3


Input Statistics ---------
Total number of different users 10
Total number of page requests received 10000
Percentage of ancestors requested  20.06
Percentage of "next page" requested  19.28
Percentage of "previous page" requested   5.19
Percentage of direct page requests   9.93
Percentage of menu-type requests by menu-option
(descending tree)   9.51  4.05  3.17  2.42  2.39  1.92  1.84  1.72  1.55  1.50
(arbitrary link)   1.87  1.23  1.67  1.47  1.70  1.47  1.57  1.33  1.52  1.64
Total time elapsed  9908.0319


Performance Statistics --------
Average number of requests per unit time   1.0093
Number of pages requested to the cache  10000
Number of index-pages requested to cache   2808
Number of pages requested to disk   9010
Number of index-pages requested to disk    803
Disk requests per page requested 0.9813
Percentage of requests satisfied within the cache
  pages= 9.90    index-pages= 71.40
Number of data-characters read from disk   7228360
Average number of entries in cache at replacement  28.11
Number of replacements in the cache   9783
```

Table 4.4 part (b)

Analysis for critical points

It is likely that different alternatives are appropriate under various conditions. It is therefore important to continue the investigations of these alternatives by analyzing them in more detail at critical parametric points. For example, having determined that a cache is appropriate for fast retrieval as long as it is "large enough", it must be determined how large it should be in practice (perhaps in terms of multiples of average page size). This further work is likely not possible through simulation, but rather it must be carried out by a careful mathematical modelling and analysis.

Validation of results

Regardless of the method of determining the effect of the parameters on file performance, it must be remembered that the results are based on a model of a videotex system, which by definition is an approximation. Therefore before incorporating the results into real production systems, the model and the results should be validated by comparing predicted performance to actual behaviour.

Extensions to other query languages

The user interface assumed for this study is the standard Telidon language originally included in the DOC database server. In Section IV.1, however, alternative language constructs were proposed, and elsewhere other

researchers are also suggesting alternative interfaces. Several of the additional language constructs could be included in an extension of the model used for this simulation study, and the performance impact of these constructs on various database architectures could be examined.

Extensions to distributed videotex systems

Similarly, the model represents a single, centralized database system only. The model should be restructured to analyze the effect of data distribution on each videotex file component. Furthermore, simulation studies may be needed to determine how a conceptual universal database of pages should be distributed among several sites in order to reduce costs.

Bibliography

ALLO79 "Transmission System Evaluation for Two-Way Cable," by G. Allora-Abbondi, IEEE Transactions on CATV, Volume CATV-4, No. 3, pp 111-118, 1979.

BALL8Ø "Videotex Networks," by A. J. S. Ball, G. B. Bochmann, and J. Gecsei, Computer, December, 1980; pp 8-14.

BALL81 "User Interfaces for Future Videotex Systems," by A. J. S. Ball and J. Gecsei, report to the Dept. of Communications, #2ØSU.36100-Ø-95Ø6, June 1981.

BERG81 "DIODE: A New Tool for Data Retreival," by M. Berger and Y. Noirel; Videotex '81 Proceedings, pp 51-56; May, 1981.

BLAK82 Personal communication with Professor I. F. Blake, Chairman of Electrical Engineering, University of Waterloo, January 1982.

BOCH81 "Overview of Protocols in Distributed Videotex Systems," by G. V. Bochmann, report to the Department of Communications, #2ØSU.36100-Ø-95Ø6, June, 1981.

BOWE8Ø "Telidon and Education in Canada," by P. G. Bowers and M. Cioni, OECA; Viewdata '8Ø Proceedings, pp. 7-17, March 1980.

BRIG8Ø "The Telematique Program in France," by R. D. Bright, Sopritel; Viewdata '8Ø Proceedings, pp. 19-24.

CICI79 "An Introduction to Teletext and Viewdata with Comments on Compatibility," by W. Ciciora, G. Sgrignoli, and W. Thomas, IEEE Transactions on Consumer Electronics, Volume CE-25, no. 3, July 1979.

CLAR81 "The UK Prestel Service -- Technical Developments Between March 1980 and March 1981," by K. E. Clarke and B. Fenn; Videotex '81 Proceedings, pp 147-162; May, 1981.

COFF73 Operating Systems Theory, by E. G. Coffman and P. J. Denning, Prentice-Hall, Englewood Cliffs, 1973.

COME79 "The Ubiquitous B-tree," by D. Comer, ACM Computing Surveys, Volume 11, No. 2, June 1979.

COST79 "Videotex System Planning," by Jose M. Costa and Anthony H. Marsh, BNR, NEC '79, Chicago, 1979.

COYNE80 "Omnitel (TM) - An Integrated Broadband Distribution System for the Eighties,", by J. J. Coyne, First Montreal Workshop on Videotex Technology, Working Document #112, Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal, June 9-18, 1980.

CRC79 "Picture Description Instructions PDI for the Telidon Videotex System,", by H. G. Bown, C. D. O'Brien, W. Sawchuck, and J. R. Storey, CRC Technical Note No. 699-E, Communications Research Center, epartment of Communications, Ottawa, 1979.

DAVI79 "Computer Networks and Their Protocols," by D. W. Davies, D. L. A. Barber, W. L. Price, and C. M. Solomonides; John Wiley and Sons, New York. 1979.

DORRO81 "The ISDN -- A Challenge and Opportunity for the '80s," by I. Dorros, ICC 1, June, 1981.

FAGI79 "Extendible Hashing -- A Fast Access Method for Dynamic Files," by R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong, ACM Transactions on Database Systems, Volume 4, No. 3, September 1979.

FEDI77 "Viewdata," by S. Fedida, Wireless World, Volume 83, no. 1494-1497, February-May, 1977.

FEDI78a "The Viewdata Computer," by S. Fedida, Wireless World, Volume 84, No. 1508-1509, April-May, 1978.

FEDI78b "Viewdata Optimization," by S. Fedida, Wireless World, Volume 84, No. 1510, June 1978.

FRED60 "Trie Memory," by E. Fredkin, Communications of the ACM, Volume 3, No. 9, September 1960.

FOX78 "Knowledge stucturing: an overview," by M. Fox, Proc. of the Second Conference of the Canadian Society for Computational Studies of Intelligence, pp 146-155; 1978.

GONN81 A Handbook of Algorithms and Data Structures, by G. H. Gonnet, Computer Science Research Report CS-80-23, University of Waterloo; May, 1981.

GREI81 "Alphabetisches Suchen in Oesterreichischen Bildschirmtext," by G. Greiner, Institut fuer Informationsverarbeitung, Tech. Univ. Graz, Austria.

GRIES71 Compiler Construction for Digital Computers, by D. Gries, John Wiley and Sons, Inc., Toronto, 1971, 493 pp.

GUIL80 "Development and Applications of the Antiope-Didon Technology," by J. Guillermin, Viewdata '80 Proceedings, pp. 29-38, March 1980.

HABE76 Introduction to Operating Systems Design, by N. Habermann, SRA, Inc., Toronto, 1976, 372 pp.

HARA81 "Current Status of the CAPTAIN System," by S. Harashima and T. Kitamura, Videotex '81 Proceedings, pp. 113-121; May, 1981.

HOOP81 "The UK Scene -- Teletext and Videotex," by R. Hooper, Videotex '81 Proceedings, pp 131-135; May, 1981.

KLEIN75a Queueing Systems, Volume 1, by L. Kleinrock; John Wiley and Sons, New York, 1975.

KLEIN75b Queueing Systems, Volume 2, by L. Kleinrock; John Wiley and Sons, New York, 1975.

KNUT73 The Art of Computer Programming: Sorting and Searching, Volume 2, by D. E. Knuth, Addison Wesley, Don Mills, 1973.

LARS78 "Dynamic Hashing," by P. A. Larson, BIT, Volume 18, No. 2, 1978.

LISK72 "Specification techniques for data abstractions," by B. H. Liskov and S. N. Zilles, IEEE Trans. on Software Eng. SE-1, 1 (1975) pp.7-19.

KUMA80 "CAPTAIN System Features -- Capability and Transmission Method," by T. Kumamoto and S. Ohkoshi, Viewdata '80 Proceedings, pp. 93-106; March, 1980.

MANT81 "The Seltext Center for the Bildschirmtext Network," by H. Mantel, Proceedings of Videotex '81, pp 179-187; May, 1981.

MART79 "The Antiope Videotex System," by B. Marti, A. Poignet, C. Schwartz, and V. Michon, IEEE Transactions on Consumer Electronics, Volume CE-25, no. 3, July 1979.

MART80 "Broadcast Text Information in France," by B. Marti, Viewdate '80 Proceedings, pp. 359-370.

McDON81 "ISDN-81: Views by Manufacturers," by John C. McDonald, ICC '81, June, 1981.

MERK78 "Secure Communications Over Insecure Channels," by R. C. Merkle, Communications of the ACM, Vol. 21, No. 4, April 1978.

MORG80 "Britain's Teletext Services Are A Commercial Success," by G. Mortan, BBC, Viewdata '80 Proceedings, pp. 341-357, March 1980.

OCON79 "Teletext Field Tests," by R. A. O'Connor, IEEE Transactions on Consumer Electronics, Volume CE-25, No. 3, July, 1979.

OTTO81 Presentation at The Second Canadian Workshop on Videotex Technology, by H. Otto; May, 1981.

PARK79 "The necessary structure," Gutenburg 2, by Godfrey and Parkhill (eds.), Press Procepic, Toronto, 1979.

PHIL81 "The design of videotex tree indexes," by D. A. Phillips (ed.), Telidon Behavioural Research II, Department of Communications, Ottawa; May 1981.

RAND64 "Special Consultant," The American College Dictionary, Random House, New York; 1964.

RIVE78 "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," by R. L. Rivest et al., Communications of the ACM, Vol. 21, No. 2, February 1978.

ROBI79 "'Touch-Tone' Teletext: A Combined Teletext-Viewdata System," by G. Robinson and W. Loveless, IEEE Transactions On Consumer Electronics, Volume CE-25, No. 3, July 1979.

SAKA80 "An ISDN System Architecture," by I. Sakakibara and H. Yumiba, NTC 1980.

SIGE80 Videotex - The Coming Revolution in Home/Office Information Retreival, by E. Sigel (editor); Knowledge Industry Publications, Inc., White Plains, N. Y., 1980.

SHAW74 The Logical Design of Operating Systems, by A. C. Shaw, Prentice-Hall, Englewood Cliffs, 1974.

SKRZ81 "Bell System Planning of ISDN," by C. S. Skrzypczak, J. H. Weber, and W. E. Falconer, ICC '81, June, 1981.

SYTE81 LocalNet System 20 Model 200 - Packet Communications Unit Reference Manual, by SYTEK Incorporated, August 1981.

TANE81 Computer Networks, by A. S. Tanebaum; Prentice-Hall, Inc. Englewood Cliffs, N.J.; 1981.

TANT79 "UK Teletext - Evolution and Potential," by N. E. Tanton, IEEE Transactions on Consumer Electronics, Volume CE-25, no. 3, July 1979.

TELE79 "Vista gives television a new face," Telesis, Volume 6, No. 1, pp. 29-31, February, 1979.

TELI81 Telidon Reports, No. 7, August 1981.

TOMPA81a "Data Structuring Facilities in Interactive Videotex Systems," by F. W. Tompa, J. Gecsei, and G. V. Bochmann, Computer, 14, 8 (August 1981), pp 72-81.

TOMPA81b "The Application of Current Database Technology to Videotex," by F. W. Tompa, J. Gecsei, and G. V. Bochmann, report to the Department of Communications, #20SU.360100-0-9506, June 1981.

TROU80 "Prestel Operational Strategy," by Dr. P. Troughton, British Post Office Communications, Viewdata '80 Proceedings, pp. 51-57, March 1980.

TSUK79 "Integrated Services Data Switching Network," by K. Tsukada, Y. Yoshida, K. Yukimatsu, and H. Ohnishi, Sixth Data Communications Symposium, November, 1979.

ULUG77 "Statistical Multiplexing of Data and Encoded Voice in a Transparent Intelligent Network," by Uluq and Gruber, Proceedings of the 5th Data Communications Symposium, September 1977.

WICK79 "Wired City U.S.A., The Charms and Dangers of Two-Way TV," by John Wicklein, The Atlantic, 1979.

WILL74 "A Time Division Multiple Access System for Digital Communication," by D. G. Willard, Computer Design, July, 1974.

WILS80 "Bell Canada's VISTA Project," by L. Wilson, Proceedings of Inside Videotex, Infomart, March, 1980.

WONG80 "Switched Data Communications Services on the Cable Television lant, Phase I - Problem Definition," by J. W. Wong, J. W. Mark, J. A. Field, Eric Manning, and V. F. DiCiccio; A report by CCNG for the Cable Telecommunications Research Institute, October 1980.

ZIMM8Ø "Future Utilization of Interactive and Broadcast Videotex in Germany and its Effects on Standardization," by R. Zimmermann, Proceedings of Viewdata '8Ø, pp 263-269; March 198Ø.

APPENDIX A: SOFTWARE FOR THE SIMULATION OF A MODEL OF THE OMNITEL NETWORK

```
#define BITS            10              /* bits per byte: 7 plus parity + start + stop */
#define DS1_CAPACITY    24              /* number of tdm streams per DS1 */
#define RVDM_IDT        64000L          /* speed of idt_line, one channel */
#define CABLE_SPEED     64000L          /* speed of feeder cable, one channel */
#define D_MEAN          750             /* mean of downstream videotex msg length (in bytes) */
#define U_MEAN          9               /* mean of upstream videotex msg length (in bytes) */
```

```
#define DEVICE          0       /* names for various servers */
#define STU             1
#define RVDM            2
#define IDT_LINE        3
#define IDT             4
#define FEEDER          5
#define DCT             6
#define CCC             7
#define BE_NET          8
#define D_STU           9
#define D_RVDM          10
#define D_IDT_LINE      11
#define D_IDT           12
#define D_FEEDER        13
#define D_DCT           14
#define D_CCC           15

#define VIDEOTEX        1  /* for device sub_type */
#define TELEPHONE       2
#define OTHER           3

#define UPSTREAM        0
#define DOWNSTREAM      1

#define DEV_GEN         1  /* for event_type */
#define SNAPSHOT        2
#define BE_NET_PROC     3
#define DEV_COMP        4
#define END_SIM         5
#define RESET_STATS     6
#define STATISTICS      7
#define START_SERVICE   8
#define DEPARTURE       9
#define STU_SERVER      10
#define RVDM_SERVER     11
#define IDTLINE_SERVER  12
#define IDT_SERVER      13
#define FEEDER_SERVER   14
#define DCT_SERVER      15
#define D_STU_SERVER    16
#define D_RVDM_SERVER   17
#define D_IDTLINE_SERVER        18
#define D_IDT_SERVER    19
#define D_CABLE_SERVER  20
#define D_DCT_SERVER    21
#define MESSAGE         22
#define EXTERNAL        23
#define DEBUG           24
#define CCC_SERVER      25
#define D_CCC_SERVER    26
#define TELE_INIT       27
#define TELE_TALK       28
#define TELE_PAUSE      29

#define TRUE   1
#define FALSE  0
```

```
#define MAX_FEEDER      2       /* maximum number of feeders */
#define MAX_IDT         8       /* maximum number of idt's per feeder */
#define MAX_RVDM        32      /* maximum number of rvdm's per idt */
#define MAX_STU         21      /* maximum number of stu's per rvdm */
#define MAX_SERVERS     8       /* maximum number of server types in the network */

struct unit_spec{       /* structure to specify orig/dest device */
        int unit_type;   /* device, STU, RVDM, etc.                    */
        int feeder_line;        /* number of feeder that unit is on */
        int idt;        /* number of idt */
        int rvdm;       /* number of rvdm */
        int stu;        /* number of stu */
        int dev_type;   /* type of device */
        int device;     /* number of device */
};

struct message{
        struct unit_spec orig_unit;
        struct unit_spec dest_unit;
        int msg_type;   /* for possible future use */
        int direction;  /* UPSTREAM or DOWNSTREAM  */
        int msg_len;
        long orig_time;
        long req_start; /* start time of videotex request */
};

struct event_list{
        long param1;                    /* some integer parameter to be passed to the event routine */
        long param2;                    /* a second int parameter to be passed to the event routine */
        int event_type;
        long start_time;                /* time that event is to take place */
        struct message *msg;            /* a pointer to the message that is to be passed */
        struct event_list *fwd, *back;  /* forward and backward pointers */
};

struct queue_entry {
        long time;                      /* time that message was put in queue */
        int next_event;                 /* the manifest for the next server's event routine */
        struct message *q_msg;          /* pointer to message in queue */
        struct queue_entry *forward;    /* forward pointer for linked list */
};

/*    GLOBAL VARIABLES    */

struct event_list *P_ev, *Last_ev;
struct queue_entry **Q_ptr;     /* pointer to current (first) entry in queue */
struct queue_entry **Last_ptr;  /* pointer to last entry in queue */
int *Idle;      /* TRUE iff indicated server is idle */
int N_queues;   /* the total number of queues needed by servers */
                /* STU, RVDM, IDT_LINE, IDT, and DCT */
int N_stat_q;   /* the number of queues for which we will keep statistics */

long Sim_time;  /* the simulator clock */
char *Filename; /* filename read in for checkpoint file */
int Input;      /* boolean: true iff normal terminal input will be taken from a file */
```

```
char *Inputfile;        /* the filename for input if not taken from the terminal */
int Restart;       /* logical vbl - true iff -R option on command line */
char *Configfile;       /* filename for configuration file */
int Debug;      /* logical vbl: true iff -d option on command line */
int Periodic;   /* logical vbl: true iff -p option on command line */
long P_int;     /* for Periodic option: schedule SNAPSHOT every P_int time units */
long P_start;   /* for Periodic option: start at P_start time units */
int Reset;      /* logical vbl: true iff -r option set on command line */
                /* reset stats at specified time */
long Reset_time;        /* time at which to reset stats */
int Message;    /* logical vbl: true iff status messages are to be sent to user */
int Msg_freq;   /* number (or frequency) of messages to be sent */
int Seed;       /* random number seed */
int Service;    /* logical vbl: true iff user is inputting service time for all servers (except channels) */
float Service_time;     /* the service time for all servers except channels */
struct message Null_msg;        /* null message for use with schedule proc */
long End_time;  /* when to stop simulation */
long Max_msgs;  /* maximum number of messages for sim */
int Calc_stats; /* whether or not (T/F) we are to collect stats for individual queues */
int Fraction[MAX_SERVERS+1];    /* inverse of fraction of queues we are to individually sample */
int Exact_stats;        /* if we want exact statistics */
long Stats_mean;        /* mean interval between stats collection events */
int Total_videotex;     /* total number of videotex devices in configuration */
int Total_telephone;    /* total number of telephone devices in configuration */
int Total_DSl_q;        /* total number of dsl queues for all feeders in each direction */
int Dilation;           /* the time dilation - normally 1000 */
float V_freq;           /* videotex terminal message generation frequency (seconds per message) */
float E_freq;           /* external traffic message generation frequency (messages per second) */
int External;           /* logical vbl: true iff external messages are to be present */
int Telephone;          /* logical vbl: true iff telephone messages are to be present */
int Conv_mean;          /* mean conversation length in seconds */
int Tele_util;          /* mean number of active telephones */
long Ev_length;         /* length of event list */
long Num_msgs;          /* total number of messages in the system */
long Num_in_q;          /* total number of queue entry items */
int E_mean;             /* mean length of external messages */

/*      CONFIGURATION TABLE     */

int Num_dct;                                            /* number of DCT's */
int Num_feeder;                                         /* number of feeders */
int Num_idt[MAX_FEEDER+1];                              /* number of IDT's per feeder */
int Num_rvdm[MAX_FEEDER+1][MAX_IDT+1];                  /* number of rvdm's */
int Num_stu[MAX_FEEDER+1][MAX_IDT+1][MAX_RVDM+1];       /* number of stu's */
int Num_telephone[MAX_FEEDER+1][MAX_IDT+1][MAX_RVDM+1]; /* number of telephones */
int Num_videotex[MAX_FEEDER+1][MAX_IDT+1][MAX_RVDM+1];  /* number of videotex displays */
int Idt_dsl_q[MAX_FEEDER+1][MAX_IDT+1][2];              /* ds-1 queue (group) number for each idt (upstream and downstream) */
int Dsl_capacity[MAX_FEEDER+1][MAX_IDT+1];              /* number of available time slots per dsl group in feeder */
int *S_capacity;                                        /* stores the current capacity for each server */

int Num_dsl_q[MAX_FEEDER+1][2];                         /* number of dsl stream groupings (queues) in feeder(s) */
int Dct_idt[MAX_FEEDER+1][MAX_IDT+1];                   /* stores the dct number assigned to handle each idt */

int N_rvdm[MAX_FEEDER+1];                               /* number of rvdms on each feeder */
int N_stu[MAX_FEEDER+1];                                /* number of stus on each feeder */
```

```
/*      STATISTICS      */

int Msgs_received;      /* number of messages sent and received by device */
float Total_delay;      /* sum of end_to_end delays for all messages received */
double Delay_sq;        /* sum of squared delays */
long Total_len;         /* sum of all received message lengths */
long Be_service;        /* sum of back-end network service times */
int Be_msgs;            /* number of messages received at back-end network */

int *N_system;          /* number of messages in various server queues */
long *N_wait;           /* the number of waiting times collected */
long *S_wait;           /* sum of waiting times */
float *Sq_wait;         /* sum of squared waiting times */
long *S_delay;          /* sum of server delays (wait time + service time) */
long *Busy;             /* sum of busy times for each queue */
long *S_nsys;           /* sum for mean number in system for each queue */
long N_sample;          /* number of samples for statistics collection */
int *Stats_num;         /* vector of size N_queues: indicates if stats are being kept for that queue,   */
                        /* and stores the index for it in the above stat vectors                         */
long Num_up;            /* the number of upstream messages received */
long Num_down;          /* the number of downstream messages received */
float Serv_up;          /* the delay time of all upstream messages */
float Serv_down;        /* the delay time of all downstream messages */
```

```
#include <stdio.h>
#include "struct.h"
#include "externl.h"

#define V_BE_NET_MEAN   0.5     /* the mean service time for the videotex back end network */
#define SD      (V_BE_NET_MEAN/12.0)    /* standard deviation for the mean */
#define R               4       /* the number of stages in the erlang distn for message length */
                                /* the peak of the erlang distn is at D_MEAN * (R-1)/R */

be_net(next_server)
int next_server;        /* manifest for the next event after the back_end net */
/*
 * this procedure simulates the videotex back_end network.
 */
{
        long new_time;
        struct unit_spec temp_unit;
        float normal();

        if( Debug )
                printf("Sim_time %ld: Back end network.\n", Sim_time);

        ++Num_up;
        Serv_up += Sim_time - P_ev->msg->orig_time;

        if(P_ev->msg->msg_type == OTHER) {
                free( P_ev->msg );
                --Num_msgs;
                return;
        }

        /* change the size of the message: 100 - 3000 long */

        P_ev->msg->msg_len = erlang(R, D_MEAN);

        /* switch orig and dest units */

        copy_unit(&P_ev->msg->orig_unit, &temp_unit);
        copy_unit(&P_ev->msg->dest_unit, &P_ev->msg->orig_unit);
        copy_unit(&temp_unit, &P_ev->msg->dest_unit);

        /* send it back thru the network */

        P_ev->msg->direction = DOWNSTREAM;
        new_time = ((float) V_BE_NET_MEAN + (normal()*SD)) * (float) Dilation + 0.5;
        Be_service += new_time;
        ++Be_msgs;
        new_time += Sim_time;
        P_ev->msg->orig_time = new_time;
        if( Debug )
                printf("      schedule DOWNSTREAM message at %ld;\n", new_time);
        schedule(next_server, P_ev->msg, new_time, 0, 0);
}

copy_unit(unit1, unit2)
struct unit_spec *unit1, *unit2;
```

```
/* this routine copies unit1 into unit2 */
{
        unit2->unit_type = unit1->unit_type;
        unit2->feeder_line = unit1->feeder_line;
        unit2->idt = unit1->idt;
        unit2->rvdm = unit1->rvdm;
        unit2->stu = unit1->stu;
        unit2->dev_type = unit1->dev_type;
        unit2->device = unit1->device;
}
```

```
#include <stdio.h>
#include "struct.h"

clean_up()
/* finish off the simulation */
{
        printf("\nEND OF SIMULATION at time %ld.\n",Sim_time);
        print_stats();
        if( Calc_stats )
                print_individual();
        printf("EOS\n");
        if( Message )
                message();
}
```

```
#include <stdio.h>
#include "struct.h"

depart_unit(unit, next_server)
/* this routine processes the departure event for the designated type of server */
int unit;               /* the number of the queue which message is in */
int next_server;        /* the next server that the message will be passed to */
                        /* (actually, the manifest for the event routine to handle it) */
{

        if( Debug )
                printf("Sim_time %ld: Departure event.\n", Sim_time);

        --N_system[unit];       /* decrement number in system */
        ++S_capacity[unit];     /* increment capacity of the system */
        if( Debug )
                printf("Sim_time %ld: capacity of queue %d = %d\n", Sim_time, unit, S_capacity[unit]);
        if(Q_ptr[unit] != NULL) {
                --S_capacity[unit];
                start_service(unit);    /* keep server busy if non-zero queue */
        }
        else                                    /* Nobody is in queue */
                if( N_system[unit] == 0 )       /* and nobody is in the system */
                        Idle[unit] = TRUE;      /* therefore, server is idle */
}
```

```
#include <stdio.h>
#include "struct.h"

dev_comp()
/* received message from network for device */
{
        float temp;

        if( Debug )
                printf("Sim_time %ld: Message received at device.\n",Sim_time);

        ++Num_down;
        Serv_down += Sim_time - P_ev->msg->orig_time;

        if(P_ev->msg->msg_type == OTHER) {
                free(P_ev->msg);
                --Num_msgs;
                return;
        }

        Msgs_received++;
        Total_len += P_ev->msg->msg_len;
        Total_delay += temp = Sim_time - P_ev->msg->req_start;
        if( Debug )
                printf("         Orig_time = %ld.\n",P_ev->msg->orig_time);
        Delay_sq += temp * temp;
        free(P_ev->msg);        /* de-allocate the message which was passed thru the net */
        --Num_msgs;
}
```

```
#include <math.h>
#include <stdio.h>
#include "struct.h"

device_msg_gen()
/* this procedure generates messages from the devices */
{
        double log();
        double temp;
        struct message *msg, *alloc_msg();
        long new_time;
        int feeder, idt, rvdm, stu, dev;
        float time;
        float frand();

        if( Debug )
                printf("Sim_time %ld: Device_msg_gen procedure.\n",Sim_time);

        /* allocate space for the message to be sent thru the network */

        msg = alloc_msg();
        ++Num_msgs;
        if(msg == NULL)
                printf("\n** ERROR **  device_msg_gen procedure: ran out of memory at time %ld.\n",
                        Sim_time);
                else {

                /* fill message */

                get_rand_video(&feeder, &idt, &rvdm, &stu, &dev);

                msg->orig_unit.unit_type = DEVICE;
                msg->orig_unit.feeder_line = feeder;
                msg->orig_unit.idt = idt;
                msg->orig_unit.rvdm = rvdm;
                msg->orig_unit.stu = stu;
                msg->orig_unit.dev_type = VIDEOTEX;
                msg->orig_unit.device = dev;

                msg->dest_unit.unit_type = BE_NET;
                msg->dest_unit.feeder_line = NULL;
                msg->dest_unit.idt = NULL;
                msg->dest_unit.rvdm = NULL;
                msg->dest_unit.stu = NULL;
                msg->dest_unit.dev_type = NULL;
                msg->dest_unit.device = NULL;

                msg->direction = UPSTREAM;
                msg->msg_type = NULL;
                msg->msg_len = frand()*14.0 + 2.0;      /* mean is 9: to change, must also change "externl.h" */
                msg->orig_time = Sim_time;
                msg->req_start = Sim_time;

                /* send msg to next unit, the STU */

                schedule(STU_SERVER, msg, Sim_time, 0, 0);
```

```
        }
        /* schedule next message generation event */

        temp = 1.0 - frand();
        time = -(V_freq / Total_videotex) * log(temp) * Dilation + 0.5;
        if( time < 1 )
                time = 1;
        new_time = Sim_time + time;
        schedule(DEV_GEN, &Null_msg, new_time, 0, 0);
}

struct message *alloc_msg()
/* allocate space for a message */
{
        char *malloc();

        return((struct message *) malloc(sizeof(struct message)));
}

get_rand_video(feeder, idt, rvdm, stu, dev)
int *feeder, *idt, *rvdm, *stu, *dev;
/* this procedure generates a random videotex device number */
{
        int i, j, k, l;
        int device_num, number;
        float frand();

        device_num = frand() * (float) Total_videotex + 1.0;
        number = 0;

        for(i=1; i<=Num_feeder; i++)
                for(j=1; j<=Num_idt[i]; j++)
                        for(k=1; k<=Num_rvdm[i][j]; k++)
                                for(l=1; l<=Num_stu[i][j][k]; l++) {
                                        number += Num_videotex[i][j][k];
                                        if( number >= device_num ) {
                                                *feeder = i;
                                                *idt = j;
                                                *rvdm = k;
                                                *stu = l;
                                                *dev = number - device_num  + 1;
                                                if( Debug )
                                                        printf("device_num %d; feeder %d; idt %d; rvdm %d; stu %d; dev %d\n",
                                                                device_num, *feeder, *idt, *rvdm, *stu, *dev);
                                                return;
                                        }
                                }
        printf("\n** FATAL ** get_rand_video: couldn'd identify device number %d\n",
                device_num);
}
```

```
#include <stdio.h>
#include "struct.h"

enter_q(type, next_server)
int type;               /* type of unit that message is queueing for */
int next_server;        /* the next server that the message will queue for */
                        /* (actually, the name of the event to process it) */
/* this routine processes the queue entry event for a variety of units */
{
        int queue;      /* number of queue in which to store message */
        int feeder, idt, direction;     /* if queue is a feeder, these are the feeder, idt and direction numbers */
        int index;

        if( Debug )
                printf("Sim_time %ld: Enter_q routine for server type %d.\n", Sim_time, type);

        if( P_ev->msg->direction == UPSTREAM )
                queue = get_q_num(&P_ev->msg->orig_unit, UPSTREAM, type);
        else
                queue = get_q_num(&P_ev->msg->dest_unit, DOWNSTREAM, type);

        ++N_system[queue];
        insert_q(queue, next_server);
        if( S_capacity[queue] > 0 ) {
                --S_capacity[queue];
                if( Debug )
                        printf("Sim_time %ld: capacity of queue %d = %d\n", Sim_time, queue, S_capacity[queue]);
                start_service(queue);
        }
}

insert_q(unit, next_server)
int unit;       /* the number of the queue to put message into */
int next_server;        /* manifest for the next server's event routine */
/* this routine inserts the current message (P_ev->msg) into a queue
   associated with the indicated unit */

{
        struct queue_entry *new_entry, *alloc_q(), *ptr;

        if( Debug )
                printf("Sim_time %ld: Insert_q routine. Queue number %d\n", Sim_time, unit);

        new_entry = alloc_q();
        ++Num_in_q;
        if(new_entry == NULL) {
                printf("\n** ERROR **  enter_q procedure: ran out of memory at time %ld.\n",
                        Sim_time);
                return;
        }
        new_entry->time = Sim_time;
        new_entry->next_event = next_server;
        new_entry->q_msg = P_ev->msg;
        new_entry->forward = NULL;
```

```
        if( Q_ptr[unit] == NULL ) {
                Q_ptr[unit] = new_entry;
                Last_ptr[unit] = new_entry;
        }
        else {
                Last_ptr[unit]->forward = new_entry;
                Last_ptr[unit] = new_entry;
        }
        if( Debug ) {
                ptr = Q_ptr[unit];
                while(ptr != NULL) {
                        pr_entry(ptr, unit);
                        ptr = ptr->forward;
                }
        }
}

struct queue_entry *alloc_q()
/* allocates space for one entry in the queue */
{
        char *malloc();

        return((struct queue_entry *) malloc(sizeof(struct queue_entry)));
}

pr_entry(ptr, unit)
struct queue_entry *ptr;
int unit;
{
        printf("Queue entry for unit %d: Origination time %ld;\n", unit, ptr->time);
        printf("Next server: %d\n", ptr->next_event);
        print_message(ptr->q_msg);
}
```

```
#include <stdio.h>
#include <math.h>
#include "struct.h"

erlang(stages, mean)
int stages;     /* number of stages in the erlang distribution */
int mean;       /* the mean value of the distribution          */
/*
 * this function returns an erlang random variable
 */
{
        int i, rv;
        int j;
        double m_rand;
        double log_val;
        float frand();
        float frand_val;

        m_rand = 1.0;
        j = 0;
        for(i=0; i<stages; i++) {
                j++;
                frand_val = frand();
                m_rand *= frand_val;
        }
        log_val = log(m_rand);
        rv = -1.0 * (float) mean / (float) stages * log_val;

        if( Debug )
                printf("Sim_time %ld: Erlang function returning %d\n", Sim_time, rv);

        return( rv );
}
```

```
#include <stdio.h>
#include <math.h>
#include "struct.h"
#include "externl.h"

#define R               10      /* number of stages for erlang distn */

external_msg()
/*
 * this procedure generates an external message (size is erlang, mean value is
 * E_mean), and then bootstraps itself
 */
{
        int feeder, idt, rvdm, stu;
        float temp;
        float frand();
        long time, new_time;
        struct message *msg, *alloc_msg();

        if( Debug )
                printf("Sim_time %ld: External_msg procedure.\n", Sim_time);

        msg = alloc_msg();
        ++Num_msgs;
        if(msg == NULL) {
                printf("\n** ERROR **  external msg generation: ran out of memory at time %ld.\n", Sim_time);
        }
        else {
                get_stu_rand(&feeder, &idt, &rvdm, &stu);
                msg->msg_type = OTHER;
                msg->msg_len = erlang(R, E_mean);
                msg->orig_time = Sim_time;
                if(frand() > 0.5) {
                        msg->direction = UPSTREAM;
                        msg->orig_unit.unit_type = DEVICE;
                        msg->orig_unit.feeder_line = feeder;
                        msg->orig_unit.idt = idt;
                        msg->orig_unit.rvdm = rvdm;
                        msg->orig_unit.stu = stu;
                        msg->orig_unit.dev_type = OTHER;
                        msg->orig_unit.device = NULL;
                        msg->dest_unit.unit_type = BE_NET;
                        msg->dest_unit.feeder_line = NULL;
                        msg->dest_unit.idt = NULL;
                        msg->dest_unit.rvdm = NULL;
                        msg->dest_unit.stu = NULL;
                        msg->dest_unit.dev_type = NULL;
                        msg->dest_unit.device = NULL;

                        schedule(STU_SERVER, msg, Sim_time, 0, 0);
                }
                else {
                        msg->direction = DOWNSTREAM;
                        msg->orig_unit.unit_type = BE_NET;
                        msg->orig_unit.feeder_line = NULL;
                        msg->orig_unit.idt = NULL;
```

```
                        msg->orig_unit.rvdm = NULL;
                        msg->orig_unit.stu = NULL;
                        msg->orig_unit.dev_type = NULL;
                        msg->orig_unit.device = NULL;
                        msg->dest_unit.unit_type = DEVICE;
                        msg->dest_unit.feeder_line = feeder;
                        msg->dest_unit.idt = idt;
                        msg->dest_unit.rvdm = rvdm;
                        msg->dest_unit.stu = stu;
                        msg->dest_unit.dev_type = OTHER;
                        msg->dest_unit.device = NULL;

                        schedule(D_DCT_SERVER, msg, Sim_time, 0, 0);
                }
        }

        temp = 1.0 - frand();
        time = -(1.0/E_freq) * log(temp) * Dilation + 0.5;
        if(time < 1)
                time = 1;
        new_time = time + Sim_time;
        schedule(EXTERNAL, &Null_msg, new_time, 0, 0);
        if( Debug )
                print_message(msg);
}

get_stu_rand(feeder, idt, rvdm, stu)
int *feeder, *idt, *rvdm, *stu;
/*
 * this procedure generates a random unit number, based on the total number
 * of stu's on the system.
 */
{
        int i, j, k;
        int device_num, number, total_stu;
        float frand();

        total_stu = 0;
        for(i=1; i<=Num_feeder; i++)
                total_stu += N_stu[i];

        device_num = frand() * (float) total_stu + 1.0;

        number = 0;
        for(i=1; i<=Num_feeder; i++)
                for(j=1; j<=Num_idt[i]; j++)
                        for(k=1; k<=Num_rvdm[i][j]; k++) {
                                number += Num_stu[i][j][k];
                                if( number >= device_num ) {
                                        *feeder = i;
                                        *idt = j;
                                        *rvdm = k;
                                        *stu = number - device_num + 1;
                                        return;
                                }
                        }
```

```
        printf("\n** FATAL **  get_stu_rand: couldn't identify unit number %d\n", device_num);
        exit(1);
}
```

```
#include "struct.h"
float
frand()
/*
 * This procedure returns a floating-point, uniformly distributed pseudo-
 * random number on the range [0,1).
 */
{
        float number;
        long random();
        int rand_val;

/*
        number = (float) (random()>>16) / 32768.0;
*/
        rand_val = rand();
        number = (float) rand() / 2147483649.0;
        return( number );
}
```

```
#include <stdio.h>
#include <ctype.h>
#include "struct.h"
#include "extern1.h"

int error;

get_parms()

/*      This routine reads in the configuration file and prompts the user for parameters */

{
        FILE *fopen(), *stream, *istream;
        char *type = "r";               /* file will be read only */
        char word[80];
        int i, j, k, temp, temp2, temp3;
        int n1, n2, i1, i2, d1, d2;
        int dsl_q_num;
        int dct_num;
        long l, atol();
        float f, atof();

        End_time = 0;   /* default end of simulation */
        Max_msgs = 0;   /* default no max on number of messages */

        printf("Configuration file: %s\n", Configfile);
        error = FALSE;
        stream = fopen(Configfile, type);       /* open config file */
        if(stream == NULL) {
                printf("Error - cannot open %s\n", Configfile);
                return( FALSE );
        }
        /* prompt user for max time, max messages etc. in this sim */

        Calc_stats = FALSE;
        for(i=1; i<=MAX_SERVERS; i++)
                Fraction[i] = 1;
        Exact_stats = FALSE;
        Stats_mean = 10000;
        if( !Input ) {
                printf("\nMaximum time on simulation?: ");
                l = -1;
                while(l < 0) {
                        scanf("%s", word);
                        if( (l=atol(word)) >= 0) End_time = l;
                        else printf("Must be a non-negative integer: ");
                };
                printf("Maximum number of messages?: ");
                l = -1;
                while(l < 0) {
                        scanf("%s", word);
                        if( (l=atol(word)) >= 0) Max_msgs = l;
                        else printf("Must be a non-negative integer: ");
                };
                if(End_time == 0 && Max_msgs == 0) {
                        printf("Error: at least one of the above two parameters must be a non-");
```

```
                printf("zero number to end the simulation.\n");
                return( FALSE );
        };
        printf("Collect individual queue statistics? ");
        word[0] = 'z';
        while( word[0] != 'y' && word[0] != 'n' ) {
                printf("yes/no: ");
                scanf("%s", word);
        }
        if( word[0] == 'y' ) {
                Calc_stats = TRUE;
                printf("Fraction of DCT queues to sample?: 1/");
                Fraction[DCT] = get_int();
                printf("Fraction of DS-1 queues to sample?: 1/");
                Fraction[FEEDER] = get_int();
                printf("Fraction of IDT queues to sample?: 1/");
                Fraction[IDT] = get_int();
                Fraction[IDT_LINE] = Fraction[IDT];
                printf("Fraction of RVDM queues to sample?: 1/");
                Fraction[RVDM] = get_int();
                printf("Fraction of STU queues to sample?: 1/");
                Fraction[STU] = get_int();
                printf("Collect exact statistics? ");
                word[0] = 'z';
                while( word[0] != 'y' && word[0] != 'n' ) {
                        printf("yes/no: ");
                        scanf("%s", word);
                }
                if( word[0] == 'y' )
                        Exact_stats = TRUE;
                else {
                        Exact_stats = FALSE;
                        printf("Mean time between collection events?: ");
                        l = -1;
                        while( l <= 0 ) {
                                scanf("%s", word);
                                if( (l=atol(word)) > 0 )
                                        Stats_mean = l;
                                else
                                        printf("Must be a positive integer: ");
                        }
                }
        }
        else
                Calc_stats = FALSE;
        printf("Videotex message generation frequency (secs per message)?: ");
        f = -1;
        while( f < 0 ) {
                scanf("%s", word);
                if( (f=atof(word)) > -0.001 )
                        V_freq = f;
                else
                        printf("Must be a positive value: ");
        }

        printf("Do you want external traffic? ");
```

```
	word[0] = 'z';
	while(word[0] != 'y' && word[0] != 'n') {
		printf("yes/no: ");
		scanf("%s", word);
	}
	if(word[0] == 'y') {
		External = TRUE;
		printf("External message frequency (messages per second)?: ");
		f = -1;
		while(f < 0) {
			scanf("%s", word);
			if( (f=atof(word)) > 0)
				E_freq = f;
			else
				printf("Must be a positive value: ");
		}
		printf("Mean length of external messages (in bytes)?: ");
		i = -1;
		while(i < 0) {
			scanf("%s", word);
			if( (i=atoi(word)) > 0)
				E_mean = i;
			else
				printf("Must be a positive value: ");
		}
	}
	else
		External = FALSE;

	printf("Do you want telephone traffic? ");
	word[0] = 'z';
	while(word[0] != 'y' && word[0] != 'n') {
		printf("yes/no: ");
		scanf("%s", word);
	}
	if(word[0] == 'n') {
		Telephone = FALSE;
	}
	else {
		Telephone = TRUE;
		printf("Mean length of telephone calls (in seconds)? ");
		i = -1;
		while(i <= 0) {
			scanf("%s", word);
			if( (i=atoi(word)) > 0 )
				Conv_mean = i;
			else
				printf("Must be a positive integer: ");
		}
		printf("Mean percentage of active telephones?: ");
		i = -1;
		while(i <= 0) {
			scanf("%s", word);
			if( (i=atoi(word)) > 0 )
				Tele_util = i;
			else
```

```
                                        printf("Must be a positive integer: ");
                        }
                }
        }
        else {
                /*
                 * must read the input from a file and report the values on std output
                 */
                istream = fopen(Inputfile, type);
                if( istream==NULL ) {
                        printf("Error: cannot open %s\n", Inputfile);
                        return( FALSE );
                }
                fscanf(istream, "%ld", &End_time);
                fscanf(istream, "%ld", &Max_msgs);
                fscanf(istream, "%s", word);
                if( word[0]=='n' )
                        Calc_stats = FALSE;
                else if( word[0] != 'y' ) {
                        printf("Error in %s: illegal value for Calc_stats, %s\n", Inputfile, word);
                        return( FALSE );
                }
                else {
                        Calc_stats = TRUE;
                        fscanf(istream, "%d", &Fraction[DCT]);
                        fscanf(istream, "%d", &Fraction[FEEDER]);
                        fscanf(istream, "%d", &Fraction[IDT]);
                        Fraction[IDT_LINE] = Fraction[IDT];
                        fscanf(istream, "%d", &Fraction[RVDM]);
                        fscanf(istream, "%d", &Fraction[STU]);
                        fscanf(istream, "%s", word);
                        if(word[0]=='y')
                                Exact_stats = TRUE;
                        else {
                                Exact_stats = FALSE;
                                fscanf(istream, "%ld", &Stats_mean);
                        }
                }
                fscanf(istream, "%f", &V_freq);
                fscanf(istream, "%s", word);
                if(word[0] == 'y') {
                        External = TRUE;
                        fscanf(istream, "%f", &E_freq);
                        fscanf(istream, "%d", &E_mean);
                }
                else
                        External = FALSE;
                fscanf(istream, "%s", word);
                if(word[0] == 'y') {
                        Telephone = TRUE;
                        fscanf(istream, "%d", &Conv_mean);
                        fscanf(istream, "%d", &Tele_util);
                }
                else
                        Telephone = FALSE;
```

```
                printf("Maximum time on simulation = %ld\n", End_time);
                printf("Maximum number of messages = %ld\n", Max_msgs);
                printf("Collect individual queue statistics = ");
                if( Calc_stats == FALSE )
                        printf("FALSE\n");
                else {
                        printf("TRUE\n");
                        printf("Fraction of DCT queues to sample = %d\n", Fraction[DCT]);
                        printf("Fraction of DS-1 queues to sample = %d\n", Fraction[FEEDER]);
                        printf("Fraction of IDT queues to sample = %d\n", Fraction[IDT]);
                        printf("Fraction of RVDM queues to sample = %d\n", Fraction[RVDM]);
                        printf("Fraction of STU queues to sample = %d\n", Fraction[STU]);
                        printf("Collect exact statistics = ");
                        if(Exact_stats == TRUE)
                                printf("TRUE\n");
                        else {
                                printf("FALSE\n");
                                printf("Mean time between collection events = %ld\n", Stats_mean);
                        }
                }
                printf("Videotex message generation frequency = %f seconds per message\n", V_freq);
                if(External == TRUE) {
                        printf("External message frequency = %f messages per second\n", E_freq);
                        printf("Mean external message length = %d bytes\n", E_mean);
                }
                else
                        printf("No External messages\n");
                if(Telephone == TRUE) {
                        printf("Mean length of telephone calls = %d\n", Conv_mean);
                        printf("Mean percentage of active telephones = %d\%\n", Tele_util);
                }
                else
                        printf("No Telephone messages.\n");
        }
        if( Debug )
                printf("End_time = %ld, Max_msgs = %ld\n", End_time, Max_msgs);


        /* NOW READ THE CONFIGURATION FILE */

        getword(word, stream);
        if(strcmp(word, "dct") != 0)
                config1_error(word, "dct");
        else {
                getword(word, stream);
                if((Num_dct=atoi(word)) <= 0) config2_error(Num_dct,"dct");
        };

        /* read number of feeders */
                if( error )
                        return( FALSE );
        getword(word, stream);
        if( strcmp(word, "feeder" ) != 0)
                config1_error(word, "feeder");
        else {
                getword(word, stream);
```

```
        if( (Num_feeder=atoi(word)) <= 0 || Num_feeder > MAX_FEEDER )
                config2_error(Num_feeder,"feeder");
}

/* read in info for each feeder */

ds1_q_num = -1;

if( error )
        return( FALSE );
for( i=1; i <= Num_feeder; i++) {
        getword(word, stream);  /* get number of idt's */
        if(strcmp(word, "idt") != 0) {
                config1_error(word, "idt");
                return( FALSE );
        }
        getword(word, stream);
        if( (Num_idt[i]=atoi(word)) <= 0 || Num_idt[i] > MAX_IDT)
                config2_error(Num_idt[i],"idt");

        /* read in dct-idt assignments */

        getword(word, stream);  /* should be "dct" */
        while( strcmp(word, "dct") == 0 && !error ) {
                getword(word, stream);
                dct_num = atoi(word);
                if( dct_num <= 0 || dct_num > Num_dct ) {
                        printf("\nError in configuration file: illegal dct number: %d\n", dct_num);
                        return( FALSE );
                }
                getword(word, stream);  /* should be "idt" */
                if( strcmp(word, "idt") != 0 ) {
                        config1_error(word, "idt");
                        return( FALSE );
                }
                getword(word, stream);
                parse_range(word, &i1, &i2);
                if( i1 <= 0 || i2 > Num_idt[i] ) {
                        printf("\nError in configuration file: illegal range for idt: %d, %d\n", i1, i2);
                        return( FALSE );
                }
                for(j=i1; j<=i2; j++)
                        Dct_idt[i][j] = dct_num;
                getword(word, stream);
        }
        for(j=1; j<=Num_idt[i]; j++) {
                if( Debug )
                        printf("idt %d assigned to dct %d\n", j, Dct_idt[i][j]);
                if( Dct_idt[i][j] == 0 ) {
                        printf("Error in configuration file: not all idt assignments given in feeder %d\n", i);
                        return( FALSE );
                }
        }


        /* read in idt to ds1 stream assignments */
```

```
while( strcmp(word, "idt") == 0 && !error ) {
        getword(word, stream);
        parse_range(word, &i1, &i2);    /* get idt range */
        if( i2>Num_idt[i] || i1<=0 ) {
                printf("\nError in configuration file:  illegal idt range: %d, %d.\n",i1,i2);
                return( FALSE );
        }
        if( error )
                return( FALSE );
        getword(word, stream);  /* should be "dsl" */
        if(strcmp(word, "dsl") != 0)
                config1_error(word, "dsl");
        else {
                getword(word, stream);
                parse_range(word, &d1, &d2);    /* get dsl range */
                if( !error ) {
                        dsl_q_num += 2;
                        for(j=i1; j<=i2; j++) {
                                Idt_dsl_q[i][j][UPSTREAM] = dsl_q_num;
                                Idt_dsl_q[i][j][DOWNSTREAM] = dsl_q_num + 1;
                                Dsl_capacity[i][j] = (d2 - d1 + 1)*DSL_CAPACITY;
                        }
                        ++Num_dsl_q[i][UPSTREAM];
                        ++Num_dsl_q[i][DOWNSTREAM];
                        getword(word, stream);  /* get next idt info, if there */
                }
        }
}
Total_DSl_q = (++dsl_q_num)/2;
if( dsl_q_num == 0 && !error ) {        /* no idt-dsl assignments read in */
        printf("\nError in configuration file: no idt-dsl assignments.\n");
        error = TRUE;
}

if( error )
        return( FALSE );

/* read in number of rvdm's per idt */
/* note: the word with number of rvdm's has already been read in */

if(strcmp(word, "rvdm") != 0)
        config1_error(word, "rvdm");
else for( j=1; j <= Num_idt[i]; j++) {
        getword(word, stream);
        if((Num_rvdm[i][j]=atoi(word)) <= 0 ||
                Num_rvdm[i][j] > MAX_RVDM)
                config2_error(Num_rvdm[i][j],"rvdm");
};

/* read in info on stu's attached to rvdm's */

for( j=1; j<=Num_idt[i] && !error; j++) {
        n2 = 0;
        while( n2 < Num_rvdm[i][j] ) {
```

```
                                getword(word, stream);
                                parse_range(word, &n1, &n2);
                                if( error )
                                        return( FALSE );
                                getword(word, stream);  /* get "stu" from file */
                                if( strcmp(word, "stu") != 0 ) {
                                        config1_error(word, "stu");
                                        return( FALSE );
                                }
                                getword(word, stream);
                                if((temp = atoi(word)) <= 0 || temp > MAX_STU)
                                        config2_error(temp,"stu");
                                getword(word, stream);   /* get number of telephones */
                                if (strcmp(word, "telephone") != 0) {
                                        config1_error(word, "telephone");
                                        return( FALSE );
                                }
                                getword(word, stream);
                                if((temp2=atoi(word)) < 0) config2_error(temp2,"telephone");
                                getword(word, stream);  /* get number of videotex units */
                                if (strcmp(word, "videotex") != 0) {
                                        config1_error(word, "videotex");
                                        return( FALSE );
                                }
                                getword(word, stream);
                                if((temp3=atoi(word)) < 0) config2_error(temp3,"videotex");
                                for( k=n1; k<=n2; k++) {
                                        Num_stu[i][j][k] = temp;
                                        Num_telephone[i][j][k] = temp2;
                                        Num_videotex[i][j][k] = temp3;
                                }
                        }
                }
        };
        fclose( stream );
        if( Input )
                fclose( istream );
        return( !error );
}

get_int()
/* this procedure will read a positive integer from the terminal */
{
        int i;
        char word[80];

        i = -1;
        while(i <= 0) {
                scanf("%s", word);
                if( (i=atoi(word)) <= 0)
                        printf("Must be a positive integer: ");
        }
        return( i );
}

getword(w, stream)
```

```
FILE *stream;
char *w;
{
        char c;
        int i;

        i = 0;
        while( isspace(c = getc(stream)) && c != EOF);
        if(c == EOF) {
                printf("Unexpected EOF on configuration file.\n");
                error = TRUE;
                return(NULL);
        }
        else {
                *w++ = c;
                while( !isspace(c = *w++ = getc(stream)) && c != EOF) i++;
                *--w = '\0';            /* replace space at end with EOS */
                return(i);
        };
}

config1_error(s1, s2)
char *s1, *s2;
{
        printf("\nError in configuration file: '%s'; expecting '%s'\n",s1,s2);
        error = TRUE;
}

config2_error(n,s)
int n;
char *s;
{
        printf("\nError in configuration file: illegal value '%d' near '%s'.\n", n,s);
        error = TRUE;
}

parse_range(w, n1, n2)
char *w;        /* word (string) to parse */
int *n1, *n2;   /* start and end of range */
{
        char s[10], *ps;

        *n1 = *n2 = 0;
        ps = s;

        while(isspace(*w)) w++; /* get rid of leading spaces */
        if( *w == '\0' ) {
                printf("\nError in configuration file: no range given.\n");
                error = TRUE;
                return(NULL);
        };
        while( !ispunct(*w) && !isspace(*w) && *w != '\0') *ps++ = *w++;
        *ps = '\0';
        *n1 = atoi(s);

        ps = s;
```

```
        if( *w == '\0' ) *n2 = *n1;
        else {  /*read second half of range */
                while( ispunct(*w) || isspace(*w)) w++;
                while( (*ps++ = *w++) != '\0' );
                *n2 = atoi(s);
        };
        if( *n1 > *n2 || n2 <= 0 ) {
                printf("\nError in configuration file:  illegal range %d,%d\n",*n1,*n2);
                error = TRUE;
        };
        return(NULL);
}
```

```
#include <stdio.h>
#include "struct.h"

get_q_num(unit, direction, type)
struct unit_spec *unit;
int direction;
int type;
/*
 * This procedure returns a queue number for the server specified by 'unit'
 * 'direction' and 'type'.  The queues are kept in an array as follows:
 *
 *            | <-------------------- feeder 1 ----------------------> |
 * ------------------------------------------------------------------------------
 * | C |  all  |  DS-1  |  idt  |  idt-  |    rvdm    |      stu      |
 * | C |  DCT  | stream | queues|  line  |   queues   |     queues    | . . . other
 * | C | queues| queues |       | queues |            |               |       feeders
 * ------------------------------------------------------------------------------
 *
 * A server's UPSTREAM queue is followed immediately by its DOWNSTREAM queue.
 */
{
        int base, number;
        int i, j;

        if( Debug )
                printf("Sim_time %ld: get_q_num procedure; ", Sim_time);

        base = 2 + Num_dct*2;           /* constant two for the CCC */
        for(i=1; i<Num_feeder; i++)
                base += Num_dsl_q[i][UPSTREAM] + Num_dsl_q[i][DOWNSTREAM] +
                        Num_idt[i]*2*2 + N_rvdm[i]*2 + N_stu[i]*2;

        switch( type ) {
        case CCC:
        case D_CCC:
                number = 2;
                break;
        case DCT:
        case D_DCT:
                number = 2 +  Dct_idt[unit->feeder_line][unit->idt]*2;
                break;
        case FEEDER:
        case D_FEEDER:
                number = base + Idt_dsl_q[unit->feeder_line][unit->idt][DOWNSTREAM];
                break;
        case IDT:
        case D_IDT:
                number = base + Num_dsl_q[unit->feeder_line][UPSTREAM]*2 +
                        unit->idt*2;
                break;
        case IDT_LINE:
        case D_IDT_LINE:
                number = base + Num_dsl_q[unit->feeder_line][UPSTREAM]*2 +
                        Num_idt[unit->feeder_line]*2 + unit->idt*2;
                break;
        case RVDM:
```

```
        case D_RVDM:
                number = base + Num_dsl_q[unit->feeder_line][UPSTREAM]*2 +
                        Num_idt[unit->feeder_line]*4 + unit->rvdm*2;
                for(i=1; i<unit->idt; i++)
                        number += Num_rvdm[unit->feeder_line][i]*2;
                break;
        case STU:
        case D_STU:
                number = base + Num_dsl_q[unit->feeder_line][UPSTREAM]*2 +
                        Num_idt[unit->feeder_line]*4 + N_rvdm[unit->feeder_line]*2;
                for(i=1; i<unit->idt; i++)
                        for(j=1; j<=Num_rvdm[unit->feeder_line][i]; j++)
                                number += Num_stu[unit->feeder_line][i][j]*2;
                for(j=1; j<unit->rvdm; j++)
                        number += Num_stu[unit->feeder_line][unit->idt][j]*2;
                number += unit->stu*2;
                break;
        default:
                printf("*** FATAL *** get_q_num: unidentified unit_type = %d\n", type);
                exit(1);
                break;
        }

        if( direction == UPSTREAM )
                number -= 1;

        if( Debug )
                printf(" queue number = %d.\n", number);

        return( number );
}


get_unit_type( queue, number )
int queue;              /* global number of the queue */
int *number;            /* number of queue within its queue type (returned) */
/*
 * this procedure returns the type of queue, given its global queue number.
 * It also returns the number of the queue within the queue type.
 */
{
        int i;

        if( Debug )
                printf("Sim_time %ld: get_unit_type procedure, queue num %d\n", Sim_time, queue);

        *number = queue;
        queue -= 2;
        if( queue <= 0 ) {
                if( *number%2 == 0 )
                        return( D_CCC );
                else
                        return( CCC );
        }
        *number = queue;
        queue -= Num_dct*2;
```

```
        if( queue <= 0 ) {
                if( *number%2 == 0 )
                        return(D_DCT);
                else
                        return(DCT);
        }
        for(i=1; i<=Num_feeder; i++) {
                *number = queue;
                queue -= Num_dsl_q[i][UPSTREAM]*2;
                if( queue <= 0 ) {
                        if( *number%2 == 0 )
                                return(D_FEEDER);
                        else
                                return(FEEDER);
                }
                *number = queue;
                queue -= Num_idt[i]*2;
                if( queue <= 0 ) {
                        if( *number%2 == 0 )
                                return(D_IDT);
                        else
                                return(IDT);
                }
                *number = queue;
                queue -= Num_idt[i]*2;
                if( queue <= 0 ) {
                        if( *number%2 == 0 )
                                return(D_IDT_LINE);
                        else
                                return(IDT_LINE);
                }
                *number = queue;
                queue -= N_rvdm[i]*2;
                if( queue <= 0 ) {
                        if( *number%2 == 0 )
                                return(D_RVDM);
                        else
                                return(RVDM);
                }
                *number = queue;
                queue -= N_stu[i]*2;
                if( queue <= 0 ) {
                        if( *number%2 == 0 )
                                return(D_STU);
                        else
                                return(STU);
                }
        }
        printf("*** FATAL *** get_unit_type: unidentified queue number.\n");
        exit(1);
}
```

```
#include <stdio.h>
#include "struct.h"
#include "extern1.h"

int index;      /* for use in set_num procedure: index into Stats_num */
int num;        /* for use in set_num procedure: counter for stats queue number */

init_sim()
{
        int i, j, k, number;
        int ind;
        int capacity;

        if( Debug )
                printf("Sim_time %ld: Init_sim procedure.\n", Sim_time);

        Msgs_received = 0;      /* initialize statistics */
        Total_delay = 0;
        Delay_sq = 0;

        /* calculate the number of server queues, and allocate storage for them */

        Total_videotex = 0;
        Total_telephone = 0;

        N_queues = 1;           /* for the CCC */
        N_queues += Num_dct + Total_DS1_q;
        for(i=1; i<=MAX_FEEDER; i++) {
                N_queues += 2*Num_idt[i]; /* one set of idt's, one set of lines to idt's */
                N_rvdm[i] = N_stu[i] = 0;
                for(j=1; j<=MAX_IDT; j++) {
                        N_queues += Num_rvdm[i][j];
                        N_rvdm[i] += Num_rvdm[i][j];
                        for(k=1; k<=MAX_RVDM; k++) {
                                N_queues += Num_stu[i][j][k];
                                N_stu[i] += Num_stu[i][j][k];
                                Total_videotex += Num_videotex[i][j][k] * Num_stu[i][j][k];
                                Total_telephone += Num_telephone[i][j][k] * Num_stu[i][j][k];
                        }
                }
        }
        N_queues *= 2;  /* account for both upstream and downstream queues */

        N_stat_q = 1;   /* Number of queues to calc stats for */
        if( Calc_stats ) {
                N_stat_q += (Num_dct-1)/Fraction[DCT] + 1
                         + (Total_DS1_q - 1)/Fraction[FEEDER] + 1;
                for(i=1; i<=Num_feeder; i++)
                        N_stat_q += (Num_idt[i] - 1)/Fraction[IDT] + 1
                                 + (Num_idt[i] - 1)/Fraction[IDT_LINE] + 1
                                 + (N_rvdm[i] - 1)/Fraction[RVDM] + 1
                                 + (N_stu[i] - 1)/Fraction[STU] + 1;
                N_stat_q *= 2;  /* account for both upstream and downstream queues */
        }
        if( Debug )
                printf("N_stat_q = %d\n", N_stat_q);
```

```
if( Debug )
        printf("Total_videotex = %d; Total_telephone = %d\n",Total_videotex, Total_telephone);
++N_queues;     /* this is just to let me access arrays from 1 to N_queues */
Q_ptr = (struct queue_entry **) calloc(N_queues, sizeof(struct queue_entry *));
Last_ptr = (struct queue_entry **) calloc(N_queues, sizeof(struct queue_entry *));
Idle = (int *) calloc(N_queues, sizeof(int));
N_system = (int *) calloc(N_queues, sizeof(int));
Stats_num = (int *) calloc(N_queues, sizeof(int));
S_capacity = (int *) calloc(N_queues, sizeof(int));
if(N_system==NULL || Idle==NULL || Last_ptr==NULL || Q_ptr==NULL || Stats_num==NULL || S_capacity==NULL) {
        printf("\n** FATAL **  not enough memory available for this configuration. N=queues = %d\n",
                N_queues);
        return( TRUE );
}
if( Calc_stats ) {
        number = N_stat_q + 1;
        N_wait = (long *) calloc(number, sizeof(long));
        S_wait = (long *) calloc(number, sizeof(long));
        Sq_wait = (float *) calloc(number, sizeof(float));
        S_delay = (long *) calloc(number, sizeof(long));
        Busy = (long *) calloc(number, sizeof(long));
        S_nsys = (long *) calloc(number, sizeof(long));
        if( N_wait==NULL || S_wait==NULL || Sq_wait==NULL || S_delay==NULL || Busy==NULL || S_nsys==NULL ) {
                printf("\n** FATAL **  not enough memory available for this configuration. N_queues = %d\n", N_queues);
                return( TRUE );
        }
}
--N_queues;     /* to undo what I did above */
for(i=1; i<=N_queues; i++) {
        Idle[i] = TRUE;
        Last_ptr[i] = NULL;
        Q_ptr[i] = NULL;
}

/*
 * the following section of code depends on the queue numbering
 * system, documented in get_q_num.c
 */

index = 0;
num = 0;
ind = 0;
set_num(1, CCC);
S_capacity[++ind] = 1;
S_capacity[++ind] = 1;
for(i=1; i<=Num_dct; i++) {
        set_num(i, DCT);
        capacity = 0;
        for(j=1; j<=Num_feeder; j++)
                for(k=1; k<=Num_idt[j]; k++)
                        if(Dct_idt[j][k] == i)
                                capacity += Ds1_capacity[j][k];
        S_capacity[++ind] = capacity;
        S_capacity[++ind] = capacity;
}
```

```
        for(i=1; i<=Num_feeder; i++) {
                for(j=1; j<=Num_ds1_q[i][UPSTREAM]; j++) {
                        set_num(j, FEEDER);
                        S_capacity[++ind] = Ds1_capacity[i][j];
                        S_capacity[++ind] = Ds1_capacity[i][j];
                }
                for(j=1; j<=Num_idt[i]; j++) {
                        set_num(j, IDT);
                        S_capacity[++ind] = Ds1_capacity[i][j];
                        S_capacity[++ind] = Ds1_capacity[i][j];
                }
                for(j=1; j<=Num_idt[i]; j++) {
                        set_num(j, IDT_LINE);
                        S_capacity[++ind] = Ds1_capacity[i][j];
                        S_capacity[++ind] = Ds1_capacity[i][j];
                }
                for(j=1; j<=N_rvdm[i]; j++) {
                        set_num(j, RVDM);
                        S_capacity[++ind] = 1;
                        S_capacity[++ind] = 1;
                }
                for(j=1; j<=N_stu[i]; j++) {
                        set_num(j, STU);
                        S_capacity[++ind] = 1;
                        S_capacity[++ind] = 1;
                }
        }

        if( Debug ) {
                printf("Sim_time %ld: N_queues = %d;\n", Sim_time, N_queues);
                for(i=1; i<=Num_feeder; i++)
                        printf("                feeder %d: N_rvdm = %d, N_stu = %d\n", i, N_rvdm[i],
                                N_stu[i]);
                printf("Stats_num:  ");
                for(i=1; i<=N_queues; i++)
                        printf("%d ", Stats_num[i]);
                printf("\n");
                printf("S_capacity: ");
                for(i=1; i<=N_queues; i++)
                        printf("%d ", S_capacity[i]);
                printf("\n");
        }
        return( FALSE );
}

set_num(number, type)
int number;     /* number of queue within its queue type */
int type;       /* the type of the queue                 */
/*
 * this procedure stuffs the Stats_num vector with the index of the
 * queue in the stats collection vectors.
 */
{
        ++index;
        if( (number%Fraction[type]) == 1 || Fraction[type] == 1) {
                Stats_num[index] = ++num;
```

```
                Stats_num[++index] = ++num;
        }
        else {
                Stats_num[index] = 0;
                Stats_num[++index] = 0;
        }
}
```

```
#include <stdio.h>
#include <signal.h>

kill()
/*
 * this procedure causes the program to die gracefully after a break.
 */
{
        clean_up();
        fflush(stdout);
        exit(1);
}

flush()
/*
 * this procedure flushes the output buffer after a kill -16.
 */
{
        signal(16, flush);
        fflush(stdout);
        message();
}

fpe(signal)
int signal;
/*
 * this procedure catches all sorts of bad (software) errors.
 */
{
        printf("\n*** Software bug!!  Value of signal is %d ***\n\n", signal);
        clean_up();
        fflush(stdout);
        exit(1);
}
```

```
/*
 * (c) Robert R. Williams, 1981.
 *
 *                                                                    *
 * This program simulates a network similar in form to Omnitel II (TM),
 * a heirarchical delivery network designed for an integrated services
 * network, implemented using coaxial cable.  The network is described
 * in "Omnitel (TM) - An Integrated Broadband Distribution System for the
 * Eighties" by J. J. Coyne, published in the proceedings of the First
 * Montreal Workshop on Videotex Technology, working document #112,
 * University of Montreal, June, 1980.
 *
 * [*: Omnitel II is a trademark of Interdiscom Systems Ltd.]
 *
 */
#include <stdio.h>
#include <signal.h>
#include "struct.h"

main(argc, argv)
int argc;       /* number of command line args */
char *argv[]; /* array of pointers to command line args */
{

        struct event_list *old_p_ev;
        int error, i, dont_stop, temp;
        long msg_ind;
        int kill(), flush(), fpe();
        long init_time, j;

        printf("CoyneSim 5.3a\n");

        if( signal(SIGTERM, SIG_IGN) != SIG_IGN )
                signal(SIGTERM, kill);
        if( signal(SIGINT, SIG_IGN) != SIG_IGN)
                signal(SIGINT, kill);
        signal(16, flush);      /* catch kill -16, and flush the buffer */
        signal(SIGBUS, fpe);
        signal(SIGILL, fpe);
        signal(SIGTRAP, fpe);
        signal(SIGIOT, fpe);
        signal(SIGEMT, fpe);
/*      signal(SIGFPE, fpe);*/
        signal(SIGSEGV, fpe);
        signal(SIGSYS, fpe);

        error = FALSE;
        init_time = 1;
        stats_seed(244);        /* random number seed for the statistics routine */

        /* set up the null message */

        Null_msg.orig_unit.unit_type = NULL;
        Null_msg.orig_unit.feeder_line = NULL;
        Null_msg.orig_unit.idt = NULL;
        Null_msg.orig_unit.rvdm = NULL;
```

```
        Null_msg.orig_unit.stu = NULL;
        Null_msg.orig_unit.dev_type = NULL;
        Null_msg.orig_unit.device = NULL;
        Null_msg.dest_unit.unit_type = NULL;
        Null_msg.dest_unit.feeder_line = NULL;
        Null_msg.dest_unit.idt = NULL;
        Null_msg.dest_unit.rvdm = NULL;
        Null_msg.dest_unit.stu = NULL;
        Null_msg.dest_unit.dev_type = NULL;
        Null_msg.dest_unit.device = NULL;
        Null_msg.direction = NULL;
        Null_msg.msg_type = NULL;
        Null_msg.msg_len = NULL;
        Null_msg.orig_time = NULL;

/* Set up all parameters for the simulation */

        if( (i=parse_command(argc, argv)) == -1) error = TRUE; /* parse cmd line */
        else {
                if (Restart) restart_chkpt();
                else {
                        Set_seed( Seed );       /* set random number seed */
                        printf("Random Seed = %d.\n", Seed);
                        if( get_parms() ) {     /* read in parameters */
                                if( Message )
                                        if( End_time > 0 )
                                                msg_ind = End_time/Msg_freq;
                                        else
                                                msg_ind = Max_msgs/Msg_freq;
                                if( Debug )
                                        print_config();
                                error = init_sim();                             /* initialize global vbls */
                                if( Periodic )
                                        schedule(SNAPSHOT, &Null_msg, P_start, 0, 0);
                                if( Calc_stats && !Exact_stats )
                                        schedule(STATISTICS, &Null_msg, Stats_mean, 0, 0);
                                if( Reset )
                                        schedule(RESET_STATS, &Null_msg, Reset_time, 0, 0);
                                if( External )
                                        schedule(EXTERNAL, &Null_msg, init_time, 0, 0);
                                if( Telephone ) {
                                        temp = (float) Total_telephone * (float) Tele_util / 100.0;
                                        for(i=0; i<temp; i++)
                                                telephone(1L);
                                }
                                if( V_freq > 0 )
                                        schedule(DEV_GEN, &Null_msg, init_time, 0, 0); /* start simulation */
                                if( End_time > 0)
                                        schedule(END_SIM, &Null_msg, End_time, 0, 0);   /* schedule end of the simulation */
                        }
                        else error = TRUE;
                }
        }

/* main loop:  Get current (next) event from the event list, and call
                        the proper procedure.  On an empty event list, stop the
```

simulation. P_ev points to the current event.

```
        dont_stop = TRUE;
        if( Reset )
                printf("Statistics reset at time %ld.\n\n", Reset_time);
        while (!error && P_ev != NULL && dont_stop) {

                Sim_time = P_ev->start_time;    /* advance simulation clock */
                if( Calc_stats && Exact_stats ) /* collect stats immediately before change in state */
                        statistics();

                switch( P_ev->event_type){
                case SNAPSHOT:
                        snapshot();
                        break;
                case RESET_STATS:
                        reset_stats();
                        break;
                case STATISTICS:
                        statistics();
                        break;
                case DEBUG:
                        Debug = TRUE;
                        break;
                case DEV_GEN:
                        device_msg_gen();
                        break;
                case EXTERNAL:
                        external_msg();
                        break;
                case TELE_INIT:
                        telephone(P_ev->param1);
                        break;
                case TELE_TALK:
                        tele_talk(P_ev->param1, P_ev->param2);
                        break;
                case TELE_PAUSE:
                        tele_pause(P_ev->param1);
                        break;
                case DEPARTURE:
                        depart_unit((int) P_ev->param1, (int) P_ev->param2);
                        break;
                case STU_SERVER:
                        enter_q(STU, RVDM_SERVER);
                        break;
                case RVDM_SERVER:
                        enter_q(RVDM, IDTLINE_SERVER);
                        break;
                case IDTLINE_SERVER:
                        enter_q(IDT_LINE, IDT_SERVER);
                        break;
                case IDT_SERVER:
                        enter_q(IDT, FEEDER_SERVER);
                        break;
                case FEEDER_SERVER:
```

```
                enter_q(FEEDER, DCT_SERVER);
                break;
        case DCT_SERVER:
                enter_q(DCT, CCC_SERVER);
                break;
        case CCC_SERVER:
                enter_q(CCC, BE_NET_PROC);
                break;
        case BE_NET_PROC:
                be_net(D_CCC_SERVER);
                break;
        case D_CCC_SERVER:
                enter_q(D_CCC, D_DCT_SERVER);
                break;
        case D_DCT_SERVER:
                enter_q(D_DCT, D_CABLE_SERVER);
                break;
        case D_CABLE_SERVER:
                enter_q(D_FEEDER, D_IDT_SERVER);
                break;
        case D_IDT_SERVER:
                enter_q(D_IDT, D_IDTLINE_SERVER);
                break;
        case D_IDTLINE_SERVER:
                enter_q(D_IDT_LINE, D_RVDM_SERVER);
                break;
        case D_RVDM_SERVER:
                enter_q(D_RVDM, D_STU_SERVER);
                break;
        case D_STU_SERVER:
                enter_q(D_STU, DEV_COMP);
                break;
        case DEV_COMP:
                dev_comp();
                break;
        case END_SIM:
                clean_up();
                exit(0);
                break;
        default: printf("\n** FATAL **  Bad event type: %d\n", P_ev->event_type);
                error = TRUE;
                clean_up();
                break;
        };

        old_p_ev = P_ev;
        P_ev = P_ev->fwd;       /* get next event list item */
        free(old_p_ev);         /* return item to system        */
        --Ev_length;

        /* check to see if sim should be stopped on max # of messages */

        if(Max_msgs > 0 && Msgs_received >= Max_msgs) {
                clean_up();
                dont_stop = FALSE;
        }
```

```
                /* check for status reporting time */

                if( Message && dont_stop )
                        if( End_time > 0) {
                                if( Sim_time >= msg_ind ) {
                                        message();
                                        msg_ind += End_time/Msg_freq;
                                }
                        }
                        else {
                                if( Msgs_received >= msg_ind ) {
                                        message();
                                        msg_ind += Max_msgs/Msg_freq;
                                }
                        }
        };      /* end_while */

        if (!error && dont_stop ) {
                if (P_ev == NULL) {
                printf("AAAARG!  Empty event list at simulation time %ld.\n",
                                Sim_time);
                clean_up();
                }

        };
}

restart_chkpt()
{
        printf("Restart with checkpoint file: %s\n", Filename);
}
```

```
#include <stdio.h>
#include "struct.h"

message()
/*
 * This procedure sends a MSG to user rrwilliams, giving a short status report.
 */

{
        FILE *send;

        if((send = popen("/usr/bin/msg rrwilliams", "w")) == NULL)
                return;
        fprintf(send, "CoyneSim is at time %f, with %d messages received.\n",
                ((float) Sim_time / (float) Dilation), Msgs_received);
        pclose(send);
}
```

```
#include <stdio.h>
#include "struct.h"

float
normal()
/*
 * this procedure returns a pseudo-normal pseudo-random variable on the range
 * [-6.0, 6.0], mean 0.  Reference "Statistical Distributions" by Hastings and
 * Peacock, page 100.
 */
{
        float sum;
        float frand();
        int i;

        sum = 0;
        for(i=0; i<12; i++)
                sum += frand();
        sum = sum - 6.0;
        if( Debug )
                printf("Sim_time %ld: Normal procedure returns %f.0\n", Sim_time, sum);
        return( sum );
}
```

```
#include <stdio.h>
#include <ctype.h>
#include "struct.h"

parse_command(argc, argv)
  int argc;
  char *argv[];

/* this function parses the command line, and fills any pertinent
   global variables.  It then returns a code as follows:
           code = 0:  no command line args were found.
           code = 1:  one or more valid args were found, and no other errors.
           code = -1: an error was found in the command line.
*/
{
        int i, code;
        char *s;
        long atol();
        long debug_time;
        float atof();

        Configfile = "/u/rrwilliams/sim/config";
        Message = FALSE;
        Msg_freq = 4;
        Debug = FALSE;
        Periodic = FALSE;
        P_int = 100;
        P_start = 1;
        Reset = FALSE;
        Reset_time = 0;
        Restart = FALSE;
        Seed = 1;
        Dilation = 10000;
        code = 0;

        while(--argc > 0) {
          if((*++argv)[0] != '-') {
                printf("Error:  illegal item on command line: %s. Expecting '-'.\n",
                                argv[0]);
                code = -1;
          }
          else {
                for(s=argv[0]+1; *s != '\0'; s++)
                switch(*s) {
                case 'R':
                        if(--argc > 0 && (*++argv)[0] != '-') {
                                Restart = TRUE;
                                Filename = argv[0];
                                if(code == 0) code = 1;
                        }
                        else {
                                printf("Error:  no filename for -R option.\n");
                                code = -1;
                                ++argc;
                                --argv;
                        }
```

```
                break;
        case 'c':
                if(--argc > 0 && (*++argv)[0] != '-') {
                        Configfile = argv[0];
                        if(code == 0) code = 1;
                }
                else {
                        printf("Error:  no filename for -c option.\n");
                        code = -1;
                        ++argc;
                        --argv;
                };
                break;
        case 'd':
                if(code == 0) code = 1;
                if(--argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0])) {
                        debug_time = atol(argv[0]);
                        schedule(DEBUG, &Null_msg, debug_time, 0, 0);
                }
                else {
                        Debug = TRUE;
                        ++argc;
                        --argv;
                }
                break;
        case 'i':
                if(--argc > 0 && (*++argv)[0] != '-') {
                        Inputfile = argv[0];
                        if(code == 0) code = 1;
                        Input = TRUE;
                }
                else {
                        printf("Error:  no filename for -i option.\n");
                        code = -1;
                        ++argc;
                        --argv;
                }
                break;
        case 'm':
                Message = TRUE;
                if(code == 0) code = 1;
                if( --argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0]))
                        Msg_freq = atoi(argv[0]);
                else {
                        ++argc;
                        --argv;
                }
                break;
        case 'p':
                Periodic = TRUE;
                if(code == 0) code = 1;
                if(--argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0])) {
                        P_start = (float) Dilation * atof(argv[0]);
                        if(--argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0]))
                                P_int = (float) Dilation * atof(argv[0]);
                        else {
```

```
                        ++argc;
                        --argv;
                    };
                }
                else {
                    ++argc;
                    --argv;
                }
                if( P_int <= 0 || P_start < 0) {
                    printf("Error: illegal values for -p option: %ld, %ld.\n",
                        P_start, P_int);
                    code = -1;
                }
                break;
        case 'r':
                Reset = TRUE;
                if(code == 0) code = 1;
                if(--argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0])) {
                    Reset_time = (float) Dilation * atof(argv[0]);
                    if(Reset_time < 0) {
                        printf("Error: illegal time for -r option: %ld\n", Reset_time);
                        code = -1;
                    }
                }
                break;
        case 's':
                if(code == 0) code = 1;
                if(--argc > 0 && (*++argv)[0] != '\0' && isdigit(*argv[0])) {
                    Seed = atoi(argv[0]);
                    if( Seed <= 0 ) {
                        printf("Error: illegal value for -s option: %d\n", Seed);
                        code = -1;
                    }
                }
                else {
                    ++argc;
                    --argv;
                    printf("Error: no value given for -s option\n");
                    code = -1;
                }
                break;
        case 'S':
                if(code == 0) code =1;
                if(--argc > 0 && (*++argv)[0] != '\0' ) {
                    Service = TRUE;
                    Service_time = atof(argv[0]);
                    if(Service_time < 0) {
                        printf("Error:  illegal value for -S option: %f\n", Service_time);
                        code = -1;
                    }
                }
                else {
                    ++argc;
                    --argv;
                    printf("Error:  no value given for -S option\n");
                    code = -1;
```

```
                        }
                        break;
                case 't':
                        if(code == 0) code = 1;
                        if(--argc>0 && (*++argv)[0]!='\0' && isdigit(*argv[0])) {
                                Dilation = atoi(argv[0]);
                                if( Dilation <= 0 ) {
                                        printf("Error: illegal value for -t option: %d\n", Dilation);
                                        code = -1;
                                }
                        }
                        else {
                                ++argc;
                                --argv;
                                printf("Error: no value given for -t option.\n");
                                code = -1;
                        }
                        break;
                default:
                                printf("Error:  illegal option on command line: -%c\n",
                                *s);
                                code = -1;
                                break;
                };  /* end of switch */
          };    /* end of else clause */
        };        /* end of while */
        if( Debug ) {
                printf("Periodic = %d; P_start = %ld; P_int = %ld.\n", Periodic, P_start, P_int);
                printf("Reset = %d; Reset_time = %ld.\n", Reset, Reset_time);
                printf("Restart = %d;", Restart);
                if( Restart )
                        printf(" Filename = %s\n", Filename);
                else
                        printf("\n");
                printf("Time Dilation = %d.\n", Dilation);
                printf("Random Number Seed = %d.\n", Seed);
        }
        return(code);
}
```

```
#include <stdio.h>
#include "struct.h"

print_ev_list()
{
/* this procedure dumps the event list onto the printer */

        int n;
        struct event_list *ptr;

        n = 0;
        ptr = P_ev;
        if (ptr == NULL) printf("Empty event list\n");

        while(ptr != NULL) {
                n++;
                printf("\nItem %d;  Event type %d;  Start time %ld\n",n,
                        ptr->event_type, ptr->start_time);
                print_message(ptr->msg);
                ptr = ptr->fwd;
        }
}

print_message(p_msg)
struct message *p_msg;
{
        printf("Message:\n");
        printf("Originating unit: feeder %d; Destination unit: feeder %d\n",
                p_msg->orig_unit.feeder_line, p_msg->dest_unit.feeder_line);
        printf("                      type %d;                      type %d\n",
                p_msg->orig_unit.unit_type, p_msg->dest_unit.unit_type);
        printf("                       idt %d;                       idt %d\n",
                p_msg->orig_unit.idt, p_msg->dest_unit.idt);
        printf("                      rvdm %d;                      rvdm %d\n",
                p_msg->orig_unit.rvdm, p_msg->dest_unit.rvdm);
        printf("                       stu %d;                       stu %d\n",
                p_msg->orig_unit.stu, p_msg->dest_unit.stu);
        printf("                  dev_type %d;                  dev_type %d\n",
                p_msg->orig_unit.dev_type, p_msg->dest_unit.dev_type);
        printf("                    device %d;                    device %d\n",
                p_msg->orig_unit.device, p_msg->dest_unit.device);
        printf("Direction = ");
        if(p_msg->direction == UPSTREAM)
                printf("upstream; ");
        else
                printf("downstream; ");
        printf("Message length: %d; Origination time: %ld\n; Request start time = %ld", p_msg->msg_len,
                p_msg->orig_time, p_msg->req_start);
}
```

```
#include <stdio.h>
#include "struct.h"

print_individual()
/* this routine prints the statistics gathered for individual queues */
{
        int i, k, queue;
        int index;

        printf("Storage statistics:  Number of messages in system = %ld\n", Num_msgs);
        printf("                     Number of entries in queues = %ld\n", Num_in_q);
        printf("                     Length of event list = %ld\n\n", Ev_length);
        for( index = 0; index <= 1; index++) {
                queue = index - 1;
                printf("\nIndividual queue statistics ");
                if( index == 0 )
                        printf("UPSTREAM: \n");
                else
                        printf("DOWNSTREAM:\n");
                printf("              number of    mean       mean                          mean number\n");
                printf("     Queue    messages    delay      wait    variance       util     in system\n\n");
                queue += 2;
                print_q_stats("CCC", 1, queue);
                for(i=1; i<=Num_dct; i+=Fraction[DCT]) {
                        queue += 2;
                        print_q_stats("Dct", i, queue);
                }
                for(i=1; i<=Num_feeder; i++) {
                        printf("\nFeeder %d:\n",i);
                        for(k=1; k<=Num_dsl_q[i][UPSTREAM]; k+=Fraction[FEEDER]) {
                                queue += 2;
                                print_q_stats("DS-1", k, queue);
                        }
                        for(k=1; k<=Num_idt[i]; k+=Fraction[IDT]) {
                                queue += 2;
                                print_q_stats("Idt", k, queue);
                        }
                        for(k=1; k<=Num_idt[i]; k+=Fraction[IDT_LINE]) {
                                queue += 2;
                                print_q_stats("Idt_line", k, queue);
                        }
                        for(k=1; k<=N_rvdm[i]; k+=Fraction[RVDM]) {
                                queue += 2;
                                print_q_stats("Rvdm", k, queue);
                        }
                        for(k=1; k<=N_stu[i]; k+=Fraction[STU]) {
                                queue += 2;
                                print_q_stats("Stu", k, queue);
                        }
                }
        }
}

float variance(n, sum, sum_sq)
long n; /* the number of items */
long sum;       /* the sum of n items */
```

```
float sum_sq;   /* the sum of squared items */
{
        float var;

        if(n > 1) {
                var = sum_sq - (float) sum * (float) sum/ (float) n;
                var = var / ((float) n - 1.0);
        }
        else var = 99999;
        return( var );
}

print_q_stats(name, number, queue)
char *name;     /* the name of the type of queue */
int number;     /* the number of the queue within queue type */
int queue;      /* the actual number of the queue */
{
        float denom, var, variance();
        long temp;

        printf("%8s %-4d: ", name, number);

        var = variance(N_wait[queue], S_wait[queue], Sq_wait[queue]);
        if(N_wait[queue] <= 1)
                temp = 1;
        else
                temp = N_wait[queue];

        printf("%5ld", N_wait[queue]);
        printf("  %9.4f", (float) S_delay[queue]/(float) temp);
        printf("  %9.4f", (float) S_wait[queue]/(float) temp);
        if( var != 99999.0 )
                printf("  %9.4f", var);
        else
                printf("  undefined");
        if( Reset_time >= Sim_time )
                denom = Sim_time;
        else
                denom = Sim_time - Reset_time;
        if( Sim_time > 0 ) {
                printf("  %9.4f", (float) Busy[queue] / denom);
                printf("  %9.4f\n", (float) S_nsys[queue]/denom);
        }
        else
                printf("       0.0000        0.0000\n");
}
```

```
#include <stdio.h>
#include "struct.h"

print_stats()
/* prints out current statistics */
{
        float var;
        int temp;

        if( Msgs_received > 1 ) {
                var = Delay_sq - Total_delay*Total_delay/Msgs_received;
                var = var / ((float) Msgs_received - 1.0);
        }
        else var = 99999.0;

        printf("\nTime Dilation = %d\n", Dilation);
        printf("\nTotal number of messages received = %d\n", Msgs_received);
        temp = Msgs_received;
        if( Msgs_received <= 1) temp = 1;
        printf("Total delay = %f; Total squared delay = %f\n",Total_delay, Delay_sq);
        printf("Average delay per videotex message = %f\n", Total_delay/temp);
        printf("Variance of delay = %f\n", var);
        printf("Mean (downstream) videotex message length = %f\n", (float) Total_len/(float) temp);
        printf("Mean back-end network delay = ");
        if( Be_msgs > 0 )
                printf("%f\n", (float) Be_service / (float) Be_msgs);
        else
                printf("0.0\n");
        printf("\nNumber of upstream messages received = %ld\n", Num_up);
        printf("Mean delay of upstream messages = ");
        if( Num_up > 0 )
                printf("%f\n", (float) Serv_up / (float) Num_up);
        else
                printf("0.0\n");
        printf("Number of downstream messages received = %ld\n", Num_down);
        printf("Mean delay of downstream messages = ");
        if( Num_down > 0 )
                printf("%f\n", (float) Serv_down / (float) Num_down);
        else
                printf("0.0\n");
}
```

```
#define SCRAMBLE 861328125L     /* 9 * 49 * 5**9, for what that's worth */
#define BITS_LONG       (sizeof(long)*8)


long Seed;

/*
 * random - generates a 31-bit pseudo-random integer based on the previous
 * value of Seed.  It uses a linear congruential random number generation
 * method as specified by Knuth in Seminumerical Algorithms.  The high bit
 * in the long random value is always zero.
 */

long
random()
{
        Seed *= SCRAMBLE;
        Seed &= ~(1L << (BITS_LONG-1));  /* i.e. mod 2**31 */
        return(Seed);
}

/*
 * Set_seed - sets Seed to the given long value
 *
 * this value should be odd to ensure a maximum period for the generator
 */
Set_seed(i)
long i;
{
        int j;

        Seed = i;
        j = i;
        srand(j);
}
```

```
#include <stdio.h>
#include "struct.h"

reset_stats()
/* reset all statistics to initial values */
{
        int i;

        if( Debug )
                printf("Sim_time %ld: Reset stats procedure.\n", Sim_time);

        Msgs_received = 0;
        Total_delay = 0;
        Delay_sq = 0;
        N_sample = 0;
        Be_service = 0;
        Be_msgs = 0;
        Total_len = 0;
        Num_up = 0;
        Num_down = 0;
        Serv_up = 0;
        Serv_down = 0;
        for(i=1; i<=N_stat_q; i++) {
                N_wait[i] = 0;
                S_wait[i] = 0;
                Sq_wait[i] = 0;
                S_delay[i] = 0;
                Busy[i] = 0;
                S_nsys[i] = 0;
                N_sample = 0;
        }
}
```

```
#include <stdio.h>
#include "struct.h"

schedule(event_kind, msg_p, time, param_one, param_two)
long param_one, param_two;      /* parameters to be passed to some event routines */
int event_kind;         /* type of event */
struct message *msg_p;  /* message to be passed (if any) */
long time;              /* simulation time of event */
{
        struct event_list *alloc_event(), *ptr, *new_ev;
        /* search event list for spot to put this event */

        ++Ev_length;
        if( Debug )
                printf("Sim_time %ld: Schedule procedure, event type %d, for time %ld.\n",
                        Sim_time, event_kind, time);

        ptr = P_ev;

        while(ptr != NULL && ptr->start_time <= time)
                ptr = ptr->fwd;

        new_ev = alloc_event();
        if(new_ev == NULL) {
                printf("\n** ERROR **  schedule procedure: ran out of memory at time %ld.\n",
                        Sim_time);
                return;
        }
        new_ev->param1 = param_one;
        new_ev->param2 = param_two;
        new_ev->event_type = event_kind;
        new_ev->start_time = time;
        new_ev->msg = msg_p;

        if(P_ev == NULL) {
                P_ev = new_ev;
                P_ev->fwd = NULL;
                P_ev->back = NULL;
                Last_ev = P_ev;
        }
        else if (ptr == NULL) {     /* insert new item at end of list */
                Last_ev->fwd = new_ev;
                new_ev->fwd = NULL;
                new_ev->back = Last_ev;
                Last_ev = new_ev;
        }
        else if(ptr == P_ev) {  /* insert at beginning of list */
                new_ev->fwd = ptr;
                new_ev->back = NULL;
                P_ev->back = new_ev;
                P_ev = new_ev;
                if( Debug )
                        printf("Sim_time %ld: Adding to front of event list.\n", Sim_time);
        }
        else {  /* insert in the middle of the list */
                new_ev->fwd = ptr;
```

```
                new_ev->back = ptr->back;
                ptr->back->fwd = new_ev;
                ptr->back = new_ev;
        };
}

struct event_list *alloc_event()
{
/* allocate an element in the event list */

        char *malloc();

        return((struct event_list *) malloc(sizeof(struct event_list)));
}
```

```
#include <stdio.h>
#include "struct.h"
#include "extern1.h"

#define E_MEAN_MSG      (E_mean*BITS)                   /* mean message length of external msgs, in bits */
#define U_MEAN_MSG      (U_MEAN*BITS)                   /* mean message length upstream, in bits */
#define D_MEAN_MSG      (D_MEAN*BITS)                   /* mean message length downstream, in bits */
#define TS_DELAY(S)     (8.0/(float)(S))                /* delay for one time_slot; S = line speed  */

#define MIN(A,B)        (((A) < (B)) ? (A) : (B))       /* minimum function */

long delay_time(unit_type, msg_len, msg_type)
/* this procedure gives a random delay time for a server of type unit_type, based on
   message length */
int unit_type;  /* the type of server */
int msg_len;    /* the length of the message */
int msg_type;   /* the type of the message */
{
        long new_time;
        float temp;
        float frand();
        long u_size, d_size;

        if( !Service ) {
                if(msg_type == OTHER) {
                        u_size = d_size = E_MEAN_MSG;
                }
                else {
                        u_size = U_MEAN_MSG;
                        d_size = D_MEAN_MSG;
                }
                temp = (frand() + 0.5) * (float) msg_len * BITS;
                switch( unit_type ) {
                case STU:               /* service time o(millisecs) */
                        temp = (temp / 1000.0) / u_size;
                        break;
                case D_STU:
                        temp = (temp / 1000.0) / d_size;
                        break;
                case RVDM:              /* service time o(10 millisecs) */
                        temp = (temp / 100.0) / u_size;
                        break;
                case D_RVDM:
                        temp = (temp / 100.0) / d_size;
                        break;
                case IDT_LINE:          /* service time depends on msg len */
                case D_IDT_LINE:
                        temp = TS_DELAY(RVDM_IDT);
                        break;
                case IDT:               /* service time ~1 millisec */
                        temp = (temp / 1000.0) / u_size;
                        break;
                case D_IDT:
                        temp = (temp / 1000.0) / d_size;
                        break;
                case FEEDER:            /* service time depends on msg len */
```

```
                case D_FEEDER:
                        temp = TS_DELAY(CABLE_SPEED);
                        break;
                case DCT:               /* service time o(millisecs) */
                        temp = (temp / 1000.0) / u_size;
                        break;
                case D_DCT:
                        temp = (temp / 1000.0) / d_size;
                        break;
                case CCC:
                case D_CCC:
                        temp = 0;
                        break;
                default:
                        printf("\n** FATAL **  service_time: illegal unit_type; value %d.\n", unit_type);
                        break;
                }
                new_time = (float) temp * (float) Dilation + 0.5;
        }
        else {
                /*
                 * service time has been input on command line - same speed
                 * will be used for all servers except channels.
                 */
                switch( unit_type ) {
                case IDT_LINE:
                case D_IDT_LINE:
                        new_time = TS_DELAY(RVDM_IDT) * (float) Dilation + 0.5;
                        break;
                case FEEDER:
                case D_FEEDER:
                        new_time = TS_DELAY(CABLE_SPEED) * (float) Dilation + 0.5;
                        break;
                case CCC:
                case D_CCC:
                        new_time = 0;
                        break;
                default:
                        new_time = Service_time * (float) Dilation + 0.5;
                        break;
                }
        }
        if( Debug ) {
                printf("Random_time: msg len = %d; new_time = %ld\n", msg_len, new_time);
                printf("Unit_type = %d, temp = %f\n", unit_type, temp);
        }
        return(new_time);
}

long
service_time(unit_type, msg_len)
/*
 * this procedure returns actual service time for the unit, as an increment
 * to the delay time given above.  This will determine the time increment to
 * the departure event.
 */
```

```
int unit_type;          /* the type of the unit */
int msg_len;            /* the length of the message */
{
        float temp;
        long new_time;

        switch(unit_type) {
        case IDT_LINE:
        case D_IDT_LINE:
                temp = ((float) msg_len * BITS - 8.0) / RVDM_IDT;
                break;
        case IDT:
        case D_IDT:
                temp = ((float) msg_len * BITS - 8.0) / MIN(RVDM_IDT, CABLE_SPEED);
                break;
        case FEEDER:
        case D_FEEDER:
                temp = ((float) msg_len * BITS - 8.0) / CABLE_SPEED;
                break;
        case DCT:
        case D_DCT:
                temp = ((float) msg_len * BITS - 8.0) / CABLE_SPEED;
                break;
        default:
                temp = 0;
                break;
        }

        new_time = temp * (float) Dilation + 0.5;
        if( Debug )
                printf("Sim_time %ld: service time returned = %ld\n", Sim_time, new_time);
        return( new_time );
}
```

```
#include <stdio.h>
#include "struct.h"

snapshot()
/* prints out the current status of the program. */
{
        long new_time;

        printf("\nSnapshot at simulation time %ld.\n",Sim_time);
        if( Debug ) {
                print_config();
                printf("\nEvent list:");
                print_ev_list();
        }
        print_stats();
        if( Calc_stats )
                print_individual();
        new_time = Sim_time + P_int;
        schedule(SNAPSHOT, &Null_msg, new_time, 0, 0);
}

print_config()
/* prints out the configuration table */
{
        int i, j, k, index;

        printf("Configuration file: %s\n", Configfile);
        printf("Number of dct's = %d; Number of feeders = %d.\n",
                Num_dct, Num_feeder);
        printf("Total number of ds-1 queues = %d\n", Total_DS1_q);
        index = 0;
        for( i=1; i<=Num_feeder; i++) {
                printf("\nFeeder %d:\n", i);
                printf("Number of idt's = %d\n", Num_idt[i]);
                printf("Number of DS-1 queues = %d\n", Num_dsl_q[i][UPSTREAM]);
                printf("Idt - DS-1 assignments:\n");
                for(j=1; j<=Num_idt[i]; j++) {
                        ++index;
                        printf("Idt %d, DS-1 queue %d\n", j, Idt_dsl_q[i][j][UPSTREAM]);
                }
                printf("Number of rvdm's:");
                for( j=1; j<=Num_idt[i]; j++)
                        printf(" %d", Num_rvdm[i][j]);
                for( j=1; j<=Num_idt[i]; j++) {
                        printf("\n%d. Stu's:", j);
                        for( k=1; k<=Num_rvdm[i][j]; k++) printf(" %d",Num_stu[i][j][k]);
                        printf("\n   Phones:");
                        for( k=1; k<=Num_rvdm[i][j]; k++) printf(" %d",Num_telephone[i][j][k]);
                        printf("\n   Videotex:");
                        for( k=1; k<=Num_rvdm[i][j]; k++) printf(" %d",Num_videotex[i][j][k]);
                }
                printf("\n");
        }
}
```

```
#include <stdio.h>
#include "struct.h"

start_service(unit)
/* this routine processes the start service event for a variety of servers */
int unit;       /* the number of the queue to retreive message from */
{
        long new_time, orig_time, service_time(), delay_time();
        long serv_time;
        struct message *msg;
        int next_server;        /* manifest for the next server's event routine */
        int sample;
        int unit_type;  /* the type of unit we are dealing with. returned from proc. */
        int number;     /* returned from get_unit_type: not used here. */
        long temp;

        if( Debug )
                printf("Sim_time %ld: Start service procedure.\n",Sim_time);

        Idle[unit] = FALSE;     /* mark server busy */
        remove_q(unit, &orig_time, &msg, &next_server);

        if( Debug )
                print_message(msg);
        unit_type = get_unit_type(unit, &number);
        serv_time = delay_time(unit_type, msg->msg_len, msg->msg_type);
        new_time = Sim_time + serv_time;
        if( unit_type != D_IDT_LINE && unit_type != DCT )
                schedule(next_server, msg, new_time, 0, 0);
        new_time += service_time(unit_type, msg->msg_len);
        schedule(DEPARTURE, msg, new_time, (long) unit, (long) next_server);
        if( unit_type == D_IDT_LINE || unit_type == DCT )
                schedule(next_server, msg, new_time, 0, 0);

        /* accumulate waiting time for this queue */

        if( Calc_stats && Stats_num[unit] != 0 ) {
                sample = Stats_num[unit];
                ++N_wait[sample];
                S_wait[sample] += temp = Sim_time - orig_time;
                Sq_wait[sample] += (float) temp * (float) temp;
                S_delay[sample] += temp + serv_time;
        }
}

remove_q(unit, time, msg_p, next_server)
/* this routine extracts one element from unit's queue, and returns
   the time and message (msg_p) stored there */
int unit;       /* the number of the queue */
long *time;     /* time at which message was stored */
struct message **msg_p; /* pointer to the message stored in the queue */
int *next_server;       /* manifest for the next server's event routine */
{
        struct queue_entry *temp;

        if( Q_ptr[unit] != NULL ) {
```

```
                *time = Q_ptr[unit]->time;
                *next_server = Q_ptr[unit]->next_event;
                *msg_p = Q_ptr[unit]->q_msg;
        }
        else {  /* error - empty queue */
                printf("\n** FATAL ** start_serv: empty queue for server %d. Time %ld.\n\n",
                        unit, Sim_time);
                *time = 0;
                *msg_p = 0;
                *next_server = END_SIM;
        }

        temp = Q_ptr[unit];
        Q_ptr[unit] = Q_ptr[unit]->forward;
        free(temp);
        --Num_in_q;
}
```

```
#include <stdio.h>
#include "struct.h"

statistics()
/* this routine collects mean number in system and utilization for some */
/* Fraction of the queues */
{
        static long last_clock;
        int i, queue, max, init;
        int type, num;
        long new_time;

        if( Debug )
                printf("Sim_time %ld: Statistics collection routine.\n", Sim_time);

        ++N_sample;

        for(i=1; i<=N_queues; i++)
                if(Stats_num[i] != 0) {
                        type = get_unit_type(i, &num);
                        if(type==RVDM || type==STU || type==CCC ||
                           type==D_RVDM || type==D_STU || type==D_CCC) {
                                if( !Idle[i] )
                                        Busy[Stats_num[i]] += Sim_time - last_clock;
                        }
                        else {
                                if( N_system[i] < Dsl_capacity[1][1] ) {
                                        Busy[Stats_num[i]] += (float) N_system[i] / (float) Dsl_capacity[1][1]
                                                * (float) (Sim_time - last_clock);
                                } else
                                        Busy[Stats_num[i]] += Sim_time - last_clock;
                        }
                        S_nsys[Stats_num[i]] += N_system[i] * (Sim_time - last_clock);
                }

        last_clock = Sim_time;
        if( !Exact_stats ) {
                new_time = Stats_mean + ((float) stats_rand()/32767.0 - 0.5)*Stats_mean + Sim_time;
                schedule(STATISTICS, &Null_msg, new_time, 0, 0);
        }
}
```

```
#define C 1     /* increment */
#define A       32767   /* multiplier = 2**15 + 1 */
#define M       65537L  /* modulus = 2**16 + 1 */

static long Last_x;

stats_rand()
{
        long x;
        int i;

        x = (A*Last_x + C)%M;
        i = x*(32767.0/(float) M);
        Last_x = x;
        return( i );
}

stats_seed(seed)
int seed;       /* the random number seed */
{
        Last_x = seed;
}
```

```
#include <stdio.h>
#include <math.h>
#include "struct.h"

#define MAX     2.3026          /* ln(10) */

tele_pause(end_conversation)
long end_conversation;          /* interval of time until the end of the conversation */

/*      this routine simulates pauses in a telephone conversation       */
/*      The pauses are distributed as log(uniform), max 10.0, min 0.1   */

{
        long new_time;
        double temp;

        if( Debug )
                printf("Sim_time %ld:  telephone pause routine; end conversation = %ld.\n", Sim_time, end_conversation);

        temp = MAX * (2.0*frand() - 1.0);
        new_time = exp(temp) * (float) Dilation + 0.5;
        end_conversation -= new_time;
        new_time += Sim_time;
        if(end_conversation >= 0)
                schedule(TELE_TALK, P_ev->msg, new_time, end_conversation, 0);
}
```

```
#include <stdio.h>
#include <math.h>
#include "struct.h"
#include "extern1.h"

#define MEAN    -0.1176 /* log 1.311 (base 0.1); 1.311 = mean of talkspurt */
#define SD      0.20    /* standard deviation of distn */
#define LEN     100     /* length of digital telephone packets, in bytes */
#define BASE    0.1     /* base for logarithms used for distribution */

tele_talk(end_conversation, pause_time)
long end_conversation;  /* interval of time until the end of the conversation */
long pause_time;        /* interval of time until talkspurt ends and pause begins */

/*
        This routine simulates talkspurts in a conversation.  The length
        of the talkspurt will be log(Normal).
*/
{
        struct message *msg, *alloc_msg();
        long time, new_time;
        double temp;
        float normal();

        if( Debug )
                printf("Sim_time %ld: telephone talkspurt routine; end conversation = %ld.\n", Sim_time, end_conversation);

        /* first, generate a pause time if necessary */

        if(pause_time <= 0) {
                temp = normal()*SD + MEAN;
                pause_time = pow((double) BASE, temp) * (float) Dilation + 0.5;
                new_time = Sim_time + pause_time;
                schedule(TELE_PAUSE, P_ev->msg, new_time, end_conversation, 0);
        }
        msg = alloc_msg();
        ++Num_msgs;
        if(msg == NULL) {
                printf("\n** ERROR ** telephone talkspurt: ran out of memory at time %ld.\n", Sim_time);
        }
        else {
                copy_unit(&P_ev->msg->orig_unit, &msg->orig_unit);
                copy_unit(&P_ev->msg->dest_unit, &msg->orig_unit);
                msg->direction = P_ev->msg->direction;
                msg->msg_type = TELEPHONE;
                msg->orig_time = Sim_time;
                msg->msg_len = LEN;
                if(msg->direction == UPSTREAM) {
                        schedule(STU_SERVER, msg, Sim_time, 0, 0);
                }
                else {
                        schedule(D_DCT_SERVER, msg, Sim_time, 0, 0);
                }

                time = (float) LEN * (float) BITS / (float) CABLE_SPEED * (float) Dilation + 0.5;
                new_time = Sim_time + time;
```

```
                end_conversation -= time;
                pause_time -= time;
                if(end_conversation > 0 && pause_time > 0)
                        schedule(TELE_TALK, P_ev->msg, new_time, end_conversation, pause_time);
        }
}
```

```
#include "struct.h"
#include <stdio.h>
#include <math.h>

#define NORMAL          0L
#define APPROX_MAX      300

telephone(mode)
long mode;      /* 0 = NORMAL: exponential distn for conversation length */
                /* 1 = init:   uniform conversation length (max 300 secs) */

/*
    This routine initializes a telephone conversation and bootstraps itself
    to keep a set proportion of the telephones busy.
 */
{
        struct message *msg, *msg2, *alloc_msg();
        int feeder, idt, rvdm, stu, number;
        long end_conversation;  /* increment of time until end of conversation */
        long new_time;
        float mean;
        float frand();

        if( Debug )
         v$ pr  telephone.c
```

```
#include "struct.h"
#include <stdio.h>
#include <math.h>

#define NORMAL          0L
#define APPROX_MAX      300

telephone(mode)
long mode;      /* 0 = NORMAL: exponential distn for conversation length */
                /* 1 = init:   uniform conversation length (max 300 secs) */

/*
    This routine initializes a telephone conversation and bootstraps itself
    to keep a set proportion of the telephones busy.
 */
{
        struct message *msg, *msg2, *alloc_msg();
        int feeder, idt, rvdm, stu, number;
        long end_conversation;  /* increment of time until end of conversation */
        long new_time;
        float mean;
        float frand();

        if( Debug )
                printf("Sim_time %ld: telephone conversation initialization.\n", Sim_time);

        msg = alloc_msg();
        ++Num_msgs;
        if(msg == NULL) {
                printf("\n** ERROR **  telephone conversation initialization: ran out of memory at time %ld.\n", Sim_time);
        }
        else {
                random_tele(&feeder, &idt, &rvdm, &stu, &number);

                /* generate random conversation length */

                if( mode == NORMAL )
                        end_conversation = - (float) Conv_mean * log(frand()) * (float) Dilation + 0.5;
                else
                        end_conversation = frand() * (float) APPROX_MAX * (float) Dilation + 0.5;

                /* stuff message for both ends of the conversation */

                msg->direction = UPSTREAM;
                msg->orig_unit.unit_type = DEVICE;
                msg->orig_unit.feeder_line = feeder;
                msg->orig_unit.idt = idt;
                msg->orig_unit.rvdm = rvdm;
                msg->orig_unit.stu = stu;
                msg->orig_unit.dev_type = TELEPHONE;
                msg->orig_unit.device = number;
                msg->dest_unit.unit_type = BE_NET;
                msg->dest_unit.feeder_line = NULL;
                msg->dest_unit.idt = NULL;
                msg->dest_unit.rvdm = NULL;
```

```
                msg->dest_unit.dev_type = NULL;
                msg->dest_unit.device = NULL;

                schedule(TELE_TALK, msg, Sim_time, end_conversation, 0L);

                msg2 = alloc_msg();
                ++Num_msgs;
                if(msg == NULL)
                        printf("\n** ERROR **  telephone conversation initialization: ran out of memeory at time %ld.\n", Sim_time);
                else {
                        msg2->direction = DOWNSTREAM;
                        msg2->orig_unit.unit_type = BE_NET;
                        msg2->orig_unit.feeder_line = NULL;
                        msg2->orig_unit.idt = NULL;
                        msg2->orig_unit.rvdm = NULL;
                        msg2->orig_unit.stu = NULL;
                        msg2->orig_unit.dev_type = NULL;
                        msg2->orig_unit.device = NULL;
                        msg2->dest_unit.unit_type = DEVICE;
                        msg2->dest_unit.feeder_line = feeder;
                        msg2->dest_unit.idt = idt;
                        msg2->dest_unit.rvdm = rvdm;
                        msg2->dest_unit.stu = stu;
                        msg2->dest_unit.dev_type = TELEPHONE;
                        msg2->dest_unit.device = number;

                        schedule(TELE_PAUSE, msg, Sim_time, end_conversation, 0);
                }
        }
        mean = (float) Conv_mean / ( (float) Total_telephone * (float) Tele_util / 100.0);
        new_time = Sim_time - mean * log(frand()) * Dilation + 0.5;
        schedule(TELE_INIT, &Null_msg, new_time, NORMAL, 0);
}

random_tele(feeder, idt, rvdm, stu, dev)
int *feeder, *idt, *rvdm, *stu, *dev;
/* this procedure generates a random telephone device number */
{
        int i, j, k, l;
        int device_num, number;
        float frand();

        device_num = frand() * (float) Total_telephone + 1.0;
        number = 0;

        for(i=1; i<=Num_feeder; i++)
                for(j=1; j<=Num_idt[i]; j++)
                        for(k=1; k<=Num_rvdm[i][j]; k++)
                                for(l=1; l<=Num_stu[i][j][k]; l++) {
                                        number += Num_telephone[i][j][k];
                                        if( number >= device_num ) {
                                                *feeder = i;
                                                *idt = j;
                                                *rvdm = k;
                                                *stu = l;
                                                *dev = number - device_num + 1;
```

```
                                        if( Debug )
                                                printf("device_num %d; feeder %d; idt %d; rvdm %d; stu %d; dev %d\n",
                                                        device_num, *feeder, *idt, *rvdm, *stu, *dev);
                                        return;
                                }
                        }
        printf("\n** FATAL ** random_tele: couldn'd identify device number %d\n",
                device_num);
}
```

```
program GenerateSimulationTrace(input,output);

const
        maxusers = 10;  { total number of users }
        IntArrTime = 1.0;       { Mean interarrival time }
        Prancest = 0.2; { Probability of selecting ancestor }
        Prnext = 0.2;   { Probability of selecting next page }
        Prprev = 0.05;  { Probability of selecting previous page }
        Prabsol = 0.1;  { Probability of selecting absolute page }
        Prdesc = 0.3;   {       "       "       "  menu (1-10) }
        Prjump = 0.15;  {       "       "       "  other menu (11-20) }
        { The above probabilities better add up to 1 }

var
        command, size : integer;
        CurrTime, r : real;
        samplesize, seed : integer;

        { F  U  N  C  T  I  O  N  S  }
        { ------------------------- }

        function random ( var seed : integer ) : real;
        { Uniform Random number generator   U(0,1).
        This version should update the seed.    }
        begin
        seed := seed*65539+1;
        if seed<0 then seed := -seed;
        random := seed/2147483648.0;
        end;


begin
readln(samplesize,seed);
for size:=1 to samplesize do begin
        write(trunc(random(seed)*maxusers+1):4);
        r := random(seed);
        if r<Prancest then command := -1
        else begin
                r := r-Prancest;
                if r<Prnext then command := -2
        else begin
                r := r-Prnext;
                if r<Prprev then command := -3
        else begin
                r := r-Prprev;
                if r<Prabsol then command := 0
        else begin
                r := r-Prabsol;
                if r<Prdesc then command := trunc(sqr(random(seed))*10+1)
        else    command := trunc(random(seed)*10+11)
                end end end end;
        write(command:3,0:2,0:2);
        CurrTime := CurrTime - ln(random(seed))*IntArrTime;
        writeln(CurrTime:8:4)
        end
end.
```

APPENDIX B: A BRIEF DESCRIPTION OF THE WESTERN DIGITAL WD2001/WD2002 DATA ENCRYPTION DEVICES

GENERAL DESCRIPTION

The Western Digital WD2001/2 are N-channel MOS TTL compatible devices which use the Federal Information Processing Data Encryption Standard algorithm to encrypt a 64 bit plaintext word using a 56 bit user specified key to produce a 64 bit ciphertext word. They also decrypt using the same user-specified key.

The WD2001/2 may be operated in block mode, or the WD2001 can be used to encrypt synchronous data in a stream using the "N"-bit cipher feedback technique.

In block encryption mode, the DES chip is programmed for encryption by loading the key into the key register. It is loaded in as eight successive 8-bit bytes (one bit of each byte is reserved for parity). The data register is then loaded with eight successive 8-bit bytes of plaintext data. Typically, 12 microseconds is required to load the key and 12 microseconds is required to load the plaintext data. The encryption time is 25 microseconds and an additional 12 microseconds are required to unload the encrypted ciphertext data in eight successive bytes. The resulting throughput is about 1.05 Mbps to 1.3 Mbps.

In cipher feedback cryptography a pseudorandom bit stream is produced as a function of the previous ciphertext. This bit stream is added modulo 2 to the plaintext to produce the next ciphertext. This is accomplished by adding a shift register and exclusive-OR logic to the DES chip. This technique may be used for any feedback size from 1 to

64 bits. If N bits are used, the resulting throughput is (N/64)* 1.Ø5 Mbps typically, or (N/64)* 1.3Ø4 Mbps in the best case.

APPENDIX C: DATA STRUCTURING FACILITIES FOR INTERACTIVE VIDEOTEX SYSTEMS

Data Structuring Facilities for Interactive Videotex Systems

Frank Wm. Tompa
Jan Gecsei
Gregor V. Bochmann

CS-80-50

November, 1980

Data Structuring Facilities for Interactive Videotex Systems

Frank Wm. Tompa †
Jan Gecsei ‡
Gregor V. Bochmann ‡

ABSTRACT

Interactive videotex systems will soon emerge as a principal information, entertainment, and communications medium. Until now there has been much written about the image presentation and data transfer facilities of the several current systems, but little attention has been devoted to the provisions for accessing pages of videotex data.

In this paper, existing data structuring facilities for interrelating videotex pages are described, and alternative structures are proposed. Throughout, emphasis is placed on the aspects of data organization that are visible from the perspective of videotex users as well as on those that affect the ability of information providers to present their data.

Key phrases: videotex, database, data structure, page organization, viewdata, Captain, Prestel, Teletel, Telidon.

CR *categories:* 4.33, 3.72, 4.32, 8.2.

November 13, 1980

† Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.
‡ Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, P.Q. H3C 3J7, Canada.

Data Structuring Facilities for Interactive Videotex Systems

Frank Wm. Tompa
Jan Gecsei
Gregor V. Bochmann

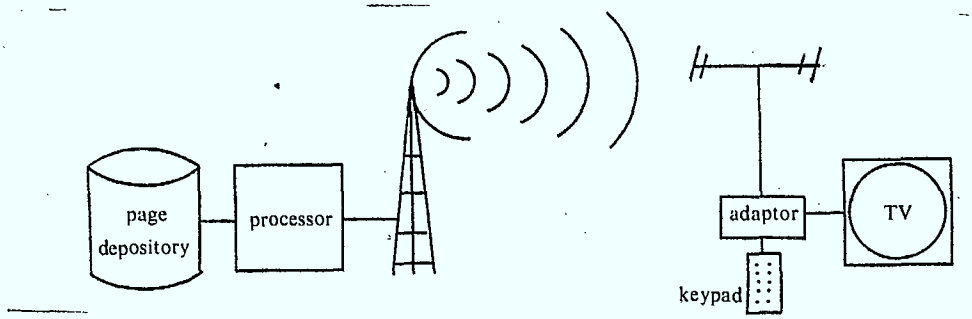
1. INTRODUCTION

The ability to access large data banks of information through the use of a television set in the home or office has long been proposed as a capability of the future. In fact, it has become a reality of the present, at least in a limited way. Around 1970 the British Broadcasting Corporation developed Ceefax, originally to provide captioning for the deaf, and by 1974 the British Post Office developed Prestel, originally called Viewdata and resulting from research towards a picture-phone.[11] Since that time, much research throughout the world has been devoted to the development of competitive and complementary systems (e.g., Captain,[16] Teletel,[3] and Telidon[12]).

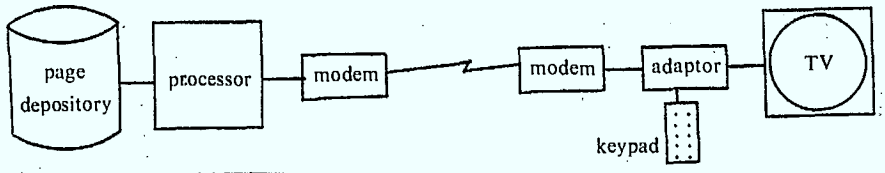
Two modes of operation have developed to date: strictly broadcast, or one-way, *teletext* systems and interactive, or two-way, *videotex* systems (Figure 1). The first of these follows the pattern established by Ceefax, that is, transmitting pages of information in a cyclic pattern and using a local adaptor on each television receiver to capture the page selected by a user when it next appears in the cycle. As an immediate consequence of this mode of operation, there is a trade-off between the number of pages of information available on one channel (and therefore the length of the cycle) and the delay in servicing one user's request to see a page (which is proportional to the number of unrequested pages that occur in the cycle between the time a user posts a request and the appropriate page is next transmitted†). Teletext service is commonly, although not exclusively, provided by over-the-air broadcasting of the signals.

Interactive videotex systems, on the other hand, disseminate pages of information solely in response to user demand. A user's request for a page is transmitted from a videotex terminal to a host processor, which in response sends the appropriate information back to the originating user. Unlike teletext, there is (virtually) no trade-off between the number of pages available and the response time; however there is a dependence of the response time on the number of users requesting service simultaneously. Videotex systems typically connect users to the central system via telephone lines, packet subnetworks, or cable television.

† In fact, whole pages need not be transmitted contiguously, but instead subpages may be interleaved; the trade-off still applies.



a) Teletext using unidirectional broadcast over the air



b) Videotex using bidirectional point-to-point (e.g., telephone) connections

Figure 1: Schematics for teletext and interactive videotex reception

The provisions of two-way communication in videotex systems and the independence of response time on database size has encouraged the development of further interactive services, including teleshopping, telebanking, telemonitoring, and interactive access to other centralized facilities from the home or office. In fact, the integrated access to multiple interactive services has been promoted as the greatest potential for videotex systems and is now beginning to be realized by Omnitel[1,5] and Mitrenet.[8]

Overviews and further information about teletext and videotex systems can be found in several books[6,14] as well as in articles about individual systems.

Data organization facilities are a major part of videotex system design. In this paper, we first summarize the facilities for inter-relating pages of information within the major current videotex systems, i.e., Captain, Prestel (and its direct derivatives including Germany's Bildschirmtext), Teletel, and Telidon. † We then describe the data structure requirements imposed by additional videotex facilities, including response pages, gateway pages to other databases, and accounting. In Section 3 we describe the experimental Teletel/Star system developed by the French Centre Commun d'Etudes de Télévision et Télécommunications (CCETT).

† Henceforth the use of the term "Captain" will refer to the system operated by the Japanese Captain System Research and Development Center, "Prestel" will refer to the system operated by the British Post Office, "Telidon" to the system operated by the Canadian Department of Communications, and "Teletel" to the system operated by the French Direction Générale des Télécommunications.

Finally we propose some alternatives to existing structures and evaluate their implications on the providers and consumers of videotex-based information.

Before discussing videotex systems further, we introduce here some vocabulary that will enable us to describe the systems using a uniform terminology. Data is primarily divided into discrete *pages* identifiable as logical units. In some cases, for example if the contents of a page is more than can be displayed at one time on a videotex screen, the page may be sub-divided into *frames* each of which forms one display image. There need not be any relation between videotex pages or frames and the units of storage ("sectors", "pages" or "blocks") used to maintain the data. The computer system on which the videotex pages are stored is managed by a *videotex system operator* (for example, a post office, telephone company, or television cable company). The pages themselves are supplied and editorially controlled by an *information provider* (commonly referred to as an IP). In fact, an information provider may also serve as the videotex system operator, and in that role is often referred to as a "value-added system operator." The consumer of the information, that is, the individual accessing the videotex system for the purpose of information retrieval or to invoke a transaction, will simply be termed a *user*.

Because the research and development of videotex systems are progressing at a very fast rate on a worldwide scale, often in a proprietary manner, we have been forced to limit our discussion of existing facilities to publicized systems only. As a result, the details included in this paper are accurate as of mid-1980, and they occasionally ignore developments contained solely within research laboratories.

2. EXISTING DATA STRUCTURING FACILITIES

2.1. The basic tree structure

The underlying data structure for all current interactive videotex systems is a (positional, labelled, rooted) tree of page nodes (Figure 2), each node serving as the root for a number of subtrees, depending on the particular system. The content of any page can include text and/or graphics and can be used to convey application information and/or index or routing prompts. Pages are numerically identified by their position in the tree; for example the root is page 0, its descendents have identifiers from 1 to 9, the descendents of page 1 have identifiers 11, 12,..., 19, the descendents of page 12 have identifiers 121, 122,...,129, etc. A user can request a page directly at any time by entering its numeric identifier via a keypad or keyboard attached to the videotex terminal.

The provision of several frames of information for one page has taken two distinct forms. In Prestel, any node (page) in the tree is a sequence of one to twenty-six frames, distinguished by appending an appropriate letter of the alphabet to the page identifier. Direct access to a page results in the display of the *primary* frame (labelled *a*), the *secondary* frames being reachable by subsequent sequential access only. Similarly, in Captain each page has one to ten frames (distinguished by an appended digit). Alternatively, Telidon allows multiple frames at the leaves of the tree only, thus distinguishing in the system implementation between so-called index and document pages (the content of either is still under complete control of the information provider). A document page can consist of up to 1000

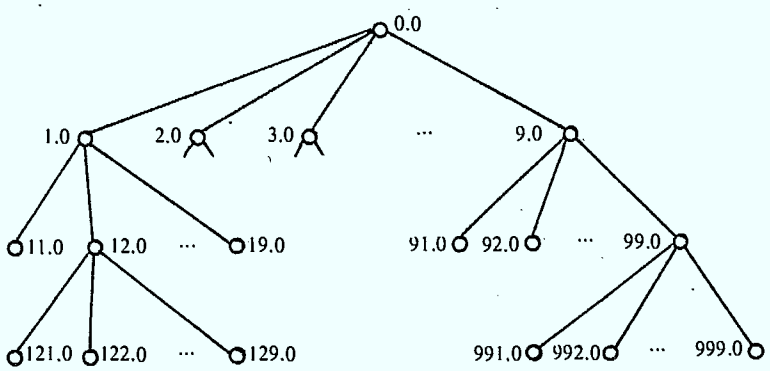


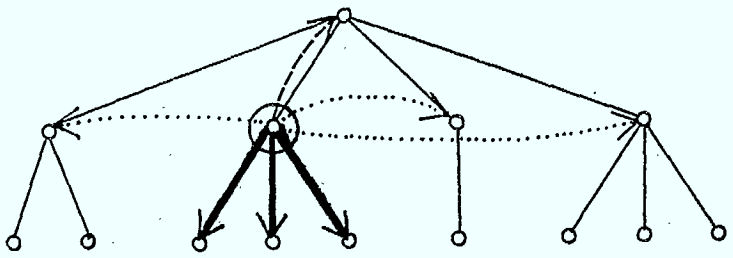
Figure 2: A tree of page nodes, using Telidon identifiers

frames identified by one, two, or three digits following the decimal point in the page identifier (e.g., 122.0, 122.1,..., 122.999), and, unlike Captain, Prestel, and Teletel, any frame can be directly accessed by entering its numeric identifier. (For consistency, a Telidon index page has a decimal point followed by the digit 0 at the end of its identifier, e.g., 12.0.)

The tree structure serves as a framework for allocating pages to information providers. For example, in Prestel and Captain each information provider is assigned a page number identifying the root of the subtree in which pages, frames, and their contents may be constructed under that information provider's editorial control. Both systems also reserve subtrees to contain general-purpose and index information controlled by the videotex system operator. Such localization of the information providers' pages simplifies the system operator's tasks of preventing interference among information providers and providing accounting (statistical and financial) to each information provider (see Section 2.3.3). The visibility of the tree structure through page identifiers also reinforces an information provider's identity to the user by explicitly containing the corresponding root page's identifier.

The tree structure may also serve as the framework in which data is presented to the user in cases where a logical hierarchy is inherent in the information. Telidon alone provides facilities for traversing the tree structure under direct user control (without any provisions by the information provider). From any index page, a user can request its immediate ancestor, an immediate descendent, or any sibling pages (i.e., those sharing the same parent), without using the target page's numeric identifier (Figure 3). Unlike the use of numeric identifiers as *absolute* page labels,† the interpretation of a user's directive is dependent on the page currently being accessed; this is therefore a form of relative page addressing, henceforth called access by *relative page label*. Such an ability to traverse the tree is extremely useful for browsing through hierarchically structured data.

† Throughout this paper, the term "identifier" will be used to refer to a unique page designator, whereas "label" will be used to refer to a name available to users for accessing a page.



From the circled node, relative access is provided to the immediate ancestor (- - →), immediate descendants (➡), and sibling pages (····→) without reference to page identifiers.

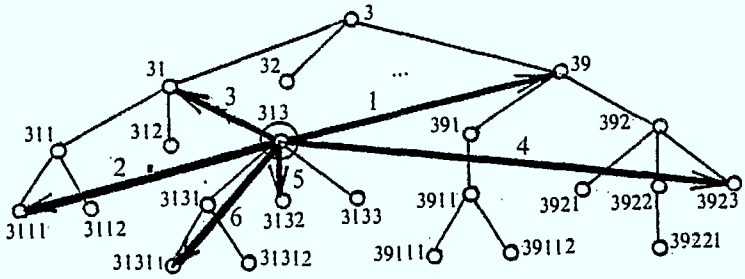
Figure 3: Tree traversals in Telidon

2.2. User-oriented access paths

Information rarely fits comfortably in a unique hierarchy. For example, it seems unreasonable to require information about Swedish restaurants to be listed exclusively under the Swedish subheading in the restaurant subtree or under the restaurant subheading in the Swedish subtree, thus forcing all users to adopt the same hierarchical ordering for retrieving the information. Furthermore, if users are given only one tree structure through which to access data, they will often be unable to locate required information without first traversing many irrelevant sections of the tree.[9] As a result, the effective response time of the system and the effective financial cost to the user (resulting from charges for connect time, page accesses, and possibly communications) will be large despite individual page retrievals being relatively efficient. Secondly, information providers will have only limited opportunity to entice users to request related information elsewhere in the tree.

It is extremely wasteful to allow multiple listings solely by physically duplicating the data. Captain, Prestel, Teletel, and Telidon therefore have facilities for allowing information providers to superimpose arbitrary directed graph structures on the underlying tree. Rather than interpreting relative page labels in the context of the underlying tree structure, an information provider can build an arbitrary access structure by establishing a transparent cross-link map to selected pages. In building the cross-link map for a page, the information provider associates with a (typically one-digit) numeric label the absolute page identifier of the page to be retrieved when a user enters that label. The page itself will usually contain a multiple-choice display that indicates the valid numeric labels together with content descriptors for the corresponding pages (Figure 4).

Although the access structures can be arbitrarily constructed (as long as the number of edges leaving a node does not exceed some bound), it has been found useful to exercise some control over the access paths so that users maintain a sense of consistency. Thus, as part of their editorial policy, information providers typically select one or several presentation styles for relating their pages of data. (Some commonly used structures have been identified and defended elsewhere.[15]) In addition, certain digits may be reserved for common functions such as "return to table of contents" or "proceed to next".



transparent cross-link map for page 313:

1 39
2 3111
3 31
4 3923
5 3132
6 31311

display for page 313:

Page 313 Great Lakes

The Great Lakes are located between the United States and Canada, covering an area of approximately 94,690 square miles.

Key 1-6 for more information:

1 Overview
2 Lake Superior (31,810 sq. mi.)
3 Lake Huron (23,000 sq.mi.)
4 Lake Michigan (22,400 sq. mi.)
5 Lake Erie (9,940 sq. mi.)
6 Lake Ontario (7,540 sq. mi.)

Figure 4: Cross-link map for user-oriented access structures

In addition to allowing a user to enter a relative numeric directive to traverse access paths, Teletel also has a primitive keyword facility. The Teletel architecture is such that there are two levels of access: the first to select a *service* (possibly corresponding to all the pages of one information provider) and the second to traverse the pages within a service. The keyword structure matches this architecture exactly: one set of keywords is used to give mnemonic names as absolute labels to individual services and each service may use keywords to give mnemonic names as absolute labels to individual pages. For example, "travel" may be used to select a particular service within which "Toronto", "hotels", or "parliament" can be used at any time to access the corresponding page as determined by the information provider.

In summary users may select pages at random (using absolute page identifiers), may traverse the directed graphs designed by the information providers

(using relative page labels), or may traverse the underlying tree structure supplied by the system operator (again using relative page labels). Finally all major systems give the user the opportunity to backtrack through retrieved pages in the reverse order of access (up to at most three pages in Prestel and ten pages in Telidon), and Telidon provides a simple command to allow a user to retrace the path in the forward direction again.

2.3. Support for other videotex facilities

To be useful, videotex systems must provide services other than mere page retrieval. Because it is important that such services are not overlooked in system designs and evaluations, some are briefly described here.

2.3.1. User response pages

Although at first videotex systems seem to be primarily information retrieval facilities, their functionality should not be limited to retrieval. The purpose of many retrievals is to obtain information in order to begin a transaction. For example a request for Swedish restaurants is not likely to be for mere academic interest, but rather so that a reservation can be made (and the food eventually consumed). Similarly, a look at stock market quotations is to determine whether to buy, to sell, or to stay pat.

Business practice dictates that a customer is more likely to respond to information if such a response can be made immediately. Thus, to be practical, videotex systems must be truly interactive and therefore distinguishable from teletext services, by (at the least) providing facilities for users to respond to pages produced by information providers.

In Prestel, an information provider can create a so-called *response page*, as distinguished from an *information page*. Such a page contains a form to be filled in by the user. The form's layout and its blend of prompting information, system generated response data (such as automatic fill-in of user's name and address), and user-generated response data is under the control of the information providers. When completed, the form is appended to a queue of users' responses, which can subsequently be examined by the appropriate information provider only.

For example, if a restaurant manger is an information provider (or uses the services of a videotex-based booking agency), he or she will likely create a videotex information page on which conventional data may be displayed (e.g. logo, name, address, telephone number, and menu extracts). As well as giving routing information for related information pages, the manager can give a prompt such as "key 9 to make a reservation." The page's cross-link map will have been set to designate a suitable response page, also created by the manager, as the corresponding target. That page will be formatted to ask the user to key in the date, time, and number in the party (all of which can be entered numerically) and to fill in the user's name and address automatically. When completed, the reservation form will be forwarded to the information provider, who can then process the information as if it were relayed by telephone. Similar transactions can be invoked for other applications.

The service provided by a response page facility is comparable to a limited electronic mail service. In fact, because response pages can be created to allow

free-form input, arbitrary messages can be sent if full character set keyboards are used. However, since only recognized information providers are eligible recipients, only a limited number of "mailboxes" are provided, and mail traffic is not as heavy as in a more general mail system.

2.3.2. Gateway pages

Even as a pure information retrieval facility, videotex systems will not realize their potential if they can only be used in isolation. There are many databases that have been built up independently of videotex – for business, libraries, and entertainment. Many of these databases will continue to be maintained independently, and therefore access must be provided to such external, or *third-party*, databases from within videotex systems.

The data in some third-party databases may already be in a format that is compatible with the videotex 40-character line standard. For other databases, the data can be reformatted into videotex pages, either when requested or *en masse* in anticipation of request (possibly by splitting 80-character lines in half, but more typically through the use of specialized reformatting routines). Finally, for some third-party databases, the videotex terminal could be made to function as a conventional alphanumeric terminal having 80-character lines.† In any of these situations, a natural mechanism for a user to request access to a third-party database is to request the appropriate *gateway* page which will have been created by an information provider.

The role of a gateway page is to integrate access to non-videotex services with videotex's page-oriented system. When a user requests a gateway page for a third-party database, the log-in protocol for that database system is initiated on the user's behalf, possibly requiring additional input from the user. Thereafter, the videotex system becomes transparent to the user, merely serving as an intermediary node in the access network by relaying the interactions between the user and the database. When the user disconnects from the third-party database system, control returns to the videotex system, and the preceding user context is re-established in preparation for further videotex commands. Such a facility is incorporated into Germany's Bildschirmtext system.

The data structures required to support the simplest form of gateway are not complex: linking information, including the (network) address of the gateway's target and interface protocols, must be stored with the page. If the contents of the third-party database are to be converted to a page-oriented format, however, the interface protocols may require extremely sophisticated techniques to control the reshaping of tables, text, and graphics.

A closely-related facility is represented by Telidon's *action pages*, which allow access to executable programs. When the program is loaded into and executed in the user's terminal (instead of being executed by the remote processor), such a facility is known as *telesoftware*. Once again integrated access

† This last alternative currently requires that the terminal be provided with suitable hardware to provide adequate resolution for the small characters that result from double-length lines. Such an option is available, for example, with the new Telidon Integrated Videotex Terminal manufactured by Electrohome in Kitchener, Ontario, Canada.

can be achieved through the use of a gateway page. When an action page is requested, videotex's page orientation can become transparent in order that the user may interact directly with the running program. Again, upon termination, the control and context can be returned to the gateway for further page-oriented activity.

2.3.3. Accounting provisions

Accounting is a vital function in any system. In videotex systems, where services are provided from many independent sources (including the system operator, the communications providers, and the information providers), accounting is extremely critical. The information required for accounting will be briefly considered here from only two viewpoints: the data collected per information provider and the data collected per user. In neither case is the data page-oriented, but rather it may be handled in a conventional record-oriented manner (and, in fact, maintained in a separate database from that containing videotex pages).

There should be an accounting entry for each information provider. Such an entry must contain the information provider's name and address (for billing) as well as a description of the set of page identifiers that are under the information provider's control (this may be simply the page identifier of the root of the appropriate subtree). In addition there should be fields for the amount of storage consumed by the information provider's pages, the resources used for page updating (e.g., for creating pages, altering the contents, altering the routing), and the total of the page charges resulting from users' accesses (for crediting the information provider). In addition, more detailed accounting information (such as the number of accesses to each page, the number of traversals of each cross-link, and a collection of statistics on *complete access paths* to pages), would provide useful feedback on the effectiveness of page layouts and organization.

Similarly, a user's accounting information must be maintained. Again the user's name and address must be included, as well as accumulated charges for system usage (processor service, input/output operations, etc.), page access charges levied on behalf of information providers, and communications charges. If gateways to independent services are available through the videotex system, then the third-party charges should also be accumulated in the user's accounting entry in order that the user is billed from only one source.

3. TELETEL/STAR

The French Centre Commun d'Etudes de Télévision et Télécommunications (CCETT) has developed a videotex system which is expected to be publicly available (at least on a trial basis) in late 1981. The Star experimental system (Source Teletel Accès Réseau) is operational in a prototype form in Rennes, based on a multiprocessor-multiminicomputer host architecture and the French Transpac public communication network.[10] It is the Star system that forms the base of the "electronic telephone directory" trials taking place in Saint-Malo since 1980. Throughout this section, the term "Teletel/Star" will refer to the videotex system based on the Star system.

Similarly to the systems described in Section 2, Teletel/Star is based on pages consisting of a (non-empty) sequence of frames and organized into a tree upon which an information provider superimposes access paths. As in the conventional Teletel system, there are two distinct levels of access, the first to choose a service and the second to traverse a structure within one service. Unlike the other systems, there is no limit to the number of frames per page nor to the number of subtrees per node (the information provider can specify the number of digits that are necessary to identify a descendent; for example, the immediate descendents of page 25 may be labelled 250, 251,..., 259; or 2500, 2501,..., 2599; or 25000, 25001,..., 25999; etc.). The most striking aspects of the Teletel/Star design, however, lie in the sensitivity of the system to a user's access path when interpreting user directives and in the innovative provisions for numeric and keyword directives.

3.1. The basic node control programming language

Associated with each page in a service's tree is a program that is written by the information provider at the time of page creation and executed whenever a user accesses the page.[7] The function of the program is to control the display of information on the videotex terminal and to interpret the commands input by the user. To govern program execution, there is associated with each user a state vector that includes the page identifier of the current page whose program is being executed, the identifier of some other "base" page, a condition code array consisting of sixteen flags, and two stacks containing identifiers of (other) pages.

Flow of control within a page's program is governed in a manner similar to a decision table. The information provider specifies a sequence of conditional expressions involving the sixteen flags. The expressions are sequentially tested, and as soon as one is found to be true, a corresponding sequence of instructions is executed (either once only or until the expression becomes false).

In all instructions having a page label as an operand (e.g., for branching or calling other pages' programs), a label *expression* can be used in place of a numeric page identifier. To implement a standard tree structure, the page retrieved when a user enters an integer is the one having a label equal to the current page's label concatenated with the numeric input. In the language for Teletel/Star, this would be indicated as "$,CCT,@" where $ is a numeric register containing the current page label, CCT indicates concatenation, and @ is a register containing the user's input; for example, on page 25 when the user inputs a 3, "$,CCT,@" assumes the value 253. To produce non-tree structures, operators for addition and subtraction are also provided; on page 25 when the user inputs a 3, "$,ADD,@" assumes the value 28 and "$,SUB,@" assumes the value 22. By using (absolute) numeric page identifiers as well as register names in combination with these three operators, arbitrary user-oriented access paths can be constructed by the information provider.

3.2. Servicing keyword directives

In keeping with the Teletel keyword facility as described in Section 2.2, Teletel/Star allows users to traverse the pages within a service by using keywords as well as numeric directives. Unlike the conventional Teletel facility, however,

the keyword is interpreted as a relative page label, i.e., the designated page depends on the page being accessed by the user at the time the directive is entered. Thus, in addition to user-oriented access structures based on directed graphs with numeric labels on the edges (as in Figure 4), an information provider can define a directed graph with keyword labels on the edges.

As an example, consider a hypothetical metals service provided via such a system (Figure 5). If a user accesses a page containing information about steel production in Canada, the keyword "copper" may lead to a page on Canadian *copper* production, "consumption" may lead to a page on Canadian steel *consumption*, and "France" may lead to a page on *French* steel production. If "iron" is entered after having entered "France", the user would be lead to *French iron* production. Thus the system is cognizant of the user's context within the database, and it interprets keyword directives accordingly.

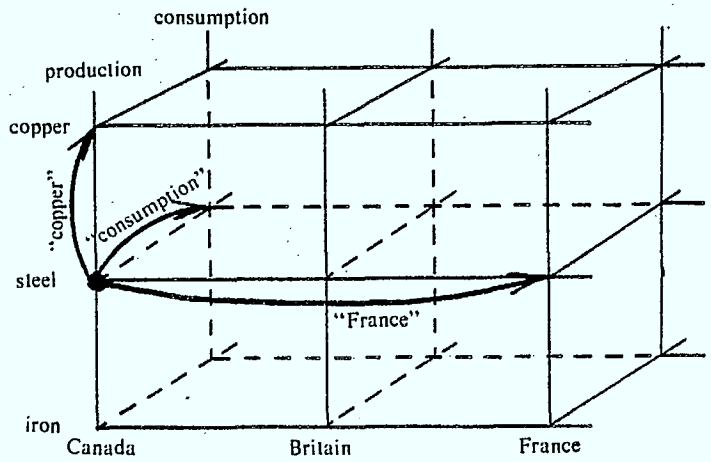


Figure 5: Grid structure using keywords

Rather than storing largely overlapping keyword directories at each node, the keyword facility is based on a service-wide set of indices defined by the information provider (Figure 6). As part of a page's program, the information provider can designate arbitrarily many keyword/index pairs, each of which causes that keyword within that index to refer to that page, henceforth known as the *target page*. Thus any page can have several keyword entries in arbitrarily many indices for which it is the target. In Figure 6, page 34 is the target page for "Britain" and "UK" in index IRON-PROD and for "iron" in index BRITISH-PROD.

Associated with each page is a sequence of up to three index names that together designate the dictionary to be used to interpret keyword directives issued when the user is accessing the page. These dictionaries can be configured to provide arbitrary user-oriented access structures. When a user enters a keyword, the first designated index is searched; if there is no match, the second and third indices are searched successively, stopping when/if a match is found. If a matching keyword is contained in one of the indices, the corresponding target page

IRON-PROD

Canada	31
Britain	34
UK	34
France	38
...	

STEEL-PROD

Canada	51
Britain	54
UK	54
France	58
...	

CANADA-PROD

iron	31
steel	51
copper	81
...	

BRITISH-PROD

iron	34
steel	54
copper	84
...	

contents of sample indices

page #	dictionary indices
31	IRON-PROD, CANADA-PROD, CANADA-IRON
34	IRON-PROD, BRITISH-PROD, BRITISH-IRON
51	STEEL-PROD, CANADA-PROD, CANADA-STEEL
81	COPPER-PROD, CANADA-PROD, CANADA-COPPER
...	...

Figure 6: Index structure for keyword processing

is accessed (i.e., its display is generated and its program executed); that is, the user proceeds along a directed labelled edge. Again referring to Figure 6, if a user enters the keyword "copper" while on page 31, the indices searched would be IRON-PROD, CANADA-PROD, and CANADA-IRON; because "copper" is not in index IRON-PROD, it would match the entry in CANADA-PROD, and thus page 81 would be the target page selected. If the keyword does not appear in any of the indices constituting the dictionary for the current page, an error flag is set and the current page's program may take corrective action if so-designed by the information provider.

3.3. Access path sensitivity

As described so far, the directed graph of pages is defined by the information provider and remains static until the information provider explicitly amends it. The Teletel/Star system, however, also provides facilities for causing the active set of labelled edges to be dependent not only on the current page, but also on the user's access path that was traced to reach that page.† For example, if a tour organizer wishes to include the same information about Toronto in the descriptions of several packaged tours, a videotex page for Toronto data can be constructed so that the keywords "schedule", "cost", and "hotel" are interpreted within the context of the tour being examined by the user. As a result, pages need

† It should be noted that the directed graphs in Teletel/Star are still static, predesigned access structures; they merely give the illusion of adapting to the user's access activity.

not be duplicated merely to provide different page-exit linkages.

The simplest form of access path dependent addressing is the ability to backtrack as provided by all videotex systems (see Section 2.2). As an extension of this idea, Teletel/Star provides an explicit stack of pages (controlled by push and pop) and a subroutine-like capability (controlled by call and return), both of which allow an information provider to cause the user's state to be saved and subsequently restored.

A third form of access path sensitivity results from the notion of a base page. As part of any page's program, the information provider can cause a register in the user's state vector (the *base*, denoted by #) to be assigned the identifier of an arbitrary page (e.g., selected by the label 25, $, or "$,CCT,@"). This base register may be used subsequently in any label expression (e.g., "#,ADD,@"), thus allowing the information provider to interpret a user's input with respect to some base that depends on the pages previously accessed.

To extend this capability to the processing of keyword as well as numeric directives, a label expression may involve the use of a dictionary. Normally a keyword is searched in the context of the current page's dictionary, but the use of the operator IDX in a label expression indicates that the dictionary of some (other) specified page is to be consulted. Thus, whereas normal keyword processing is defined by the program statement "$,IDX,@" (search for the input keyword in the current page's dictionary), the information provider can instead specify "#,IDX,@" (search in the dictionary associated with the base page) or use any other label expression involving IDX. For example, the tour organizer mentioned above could cause the base to be set at each tour's entry page and certain keywords to be interpreted with respect to that base.

4. ALTERNATIVE DATA STRUCTURING FACILITIES

The data structuring facilities of today's videotex systems are very similar. Only Teletel/Star has deviated from the others by introducing a sophisticated keyword facility and the concept of an executable program related to each page.

We will now re-examine some of the basic notions common to all systems and propose some alternatives. In Section 4.1 we discuss the role of the underlying tree structure, in Section 4.2 we propose the expunction of visible page identifiers, and in Section 4.3 we examine the applicability of conventional database system technology to videotex systems.

4.1. Eliminating the underlying tree structure

The basic tree structure was described in Section 2.1. As discussed subsequently, it is rare that the tree forms a suitable structure for presenting information (i.e., a suitable external schema); rather other structures are superimposed on the tree to provide better user-oriented access paths. In addition, the tree is not a convenient structure for data representation on the physical storage media (internal schema); rather the page identifiers are used independently of the tree structure for retrieval by a storage-oriented access method based on hashing, indexed sequential store, or some other keyed retrieval method. As a result, the primary use of the tree structure is as a storage management device for the convenience of the videotex system operator.

Instead of depending on the tree structure, which has on occasion been confusing to computer-naive users,[13] the underlying structure of the database may be better suited when viewed as a one-dimensional array of pages. The logical structure is then merely an arbitrarily large set of numbered pages having identifiers 1, 2,..., 9, 10, 11,..., 99, 100, 101,.... Such a structure is, in fact, the one most commonly used for other address spaces, such as for labelling the cells of a primary memory. In this section we will show that the facilities now provided by a tree would not be lost nor significantly impaired.

The tree structure provides a framework for storage management in terms of subtrees. Similarly the array structure can be managed in terms of logically contiguous blocks. Using an array, pages could be allocated to information providers in blocks of a suitable size, for example 100 or 1000 pages per block. Cross-links can still be established independently of the underlying structure (since there is no requirement that cross-links be contained within a subtree, there need not be a restriction that they remain within one block allocation), and thus there is no problem with the page identifier spaces for several information providers being interleaved, as long as the information providers are restricted to editing in their own subspaces. Such a structure is commonly used by operating systems: programs are allocated one or more regions in memory in which they can store arbitrary data, often including references to data within the system's and other users' regions. The extensive operating systems experience of managing primary memory and the allocation of blocks of tracks on secondary storage devices can be fruitfully applied to videotex page management. In fact, in keeping with this experience, it may be better if each information provider has a separate "virtual array of pages," some of which are designated to be "entry pages" to which cross-links may be made; this would likely simplify the task of maintaining consistency among the pages supplied by different information providers.

With the absence of the tree structure, the inclination to distinguish secondary frames from primary page frames also disappears. In fact, the distinction need not be made until an information provider constructs access paths. Whenever a new page/frame is needed, an unused slot is suballocated from one of the information provider's blocks and linked to the existing pages by cross-links or by insertion into a frame sequence. Thus, unlike in Prestel, Captain, and Telidon, frame sequences can be made arbitrarily long and the difference between primary and secondary frames is merely in their placement as designed by the information provider and in their access paths as interpreted by the users.

The remaining use of the tree structure is to serve as a focal point for an information provider; that is, the root of a subtree is identified as being an entry point and the set of page identifiers in the subtree is recognizable as belonging to the information provider. Designating an arbitrary page as the information provider's entry point causes no real problems, since it has already been found that commercial videotex systems require printed directories to catalog the information providers' entry labels. In Prestel, the information provider's name is placed next to the page identifier at the top of each frame; thus using the page identifier itself to indicate the information provider is unnecessary.

It may be argued that it is desirable for a user to return to the appropriate information provider's entry page by merely entering the first few digits of any

current page's identifier, as is the case for Captain and Prestel. Because there is no obvious root for an array of pages, the alternative is to provide an implicit "link to information provider" from each page; for example, the entry "#0" or "↑" may be reserved to designate a user's request to access the appropriate information provider's entry page. (The videotex system can retrieve the appropriate page identifier from the information provider's accounting entry as described in Section 2.3.3.)

4.2. Eliminating users' awareness of page identifiers

Having found that the tree structure is of little value, it may be quickly realized that the page identifiers themselves are virtually meaningless. It is ironic that the videotex information medium requires the extensive use of printed material to inform users of page identifiers (see, for example, the *Prestel Business Guide* published periodically by The Financial Times Ltd.).

It has long been realized by programming language, operating system, and data base system designers that users' exposure to absolute memory addresses or address-dependent data base keys is highly undesirable. Such exposure inhibits the system's flexibility to reorganize the data (for example, to increase efficiency) because at any time any user may request a piece of data by an address that may no longer be valid. Similar criticisms can be levelled at page identifiers: they are non-mnemonic labels that inhibit the reassignment of pages to information providers or their re-use by an information provider within a service. † Again it needs to be shown that the useful roles of page identifiers as page labels can be better served by alternative means. (Naturally some form of page identifier must still be used by the system itself; the criticism here is directed at their visibility to users.)

Page identifiers serve as a means to traverse the set of pages in a videotex system. One role of the information provider is to develop access paths that anticipate users' needs; these are incorporated by means of multiple-choice displays and cross-links, for which absolute page identifiers are not required by users. This is, of course, enhanced when keywords are permitted, since the cross-linking becomes mnemonic.

Often users do not wish to traverse long access paths to retrieve often-requested pages, and absolute page identifiers allow direct access. The problem is easily circumvented by the information provider constructing an arbitrarily large set of quick-access absolute page labels (unrelated to the *system's* page identifiers) that are stored in an information provider's label map, similar to a page's cross-link map, and are usable from any page. For example, whereas entering "3" uses the current page's cross-link map to determine the target page, entering "#3" would use the information provider's map. The system could reserve "#0" to access the entry page (as mentioned in the previous section) or to serve as a complete system reset in order to insure that a user is not unwillingly trapped within an information provider's access structure. (In fact, a user *could* always

† The British Post Office is contemplating increasing the number of recognized information providers by using four-digit instead of three-digit identifiers for their roots; this may well cause problems for current information providers and users.

disconnect from the system by unplugging the receiver or hanging up the telephone, but this is inelegant.)

It is impossible for information providers to supply absolute labels for all their users' needs, and there is therefore still some demand for each user to access other arbitrary pages conveniently. With the use of numeric page identifiers, each user can keep a private, written list of commonly used identifiers for reference when using the system. A more satisfactory method, and a necessity if there is no direct access to page identifiers, is to have the system maintain for each user a private, relatively small absolute label map similar to that of the information provider. For example, the command "*3" could access the user's label map to determine the target page to be retrieved, and "**3" could be used to indicate that the current page (typically retrieved by following some access path) should be entered opposite label 3 in the user's label map.

Although absolute labels have been described in terms of numeric directives, their use is even more appealing with keyword labels. The availability of keywords would, of course, allow the absolute page labels to be mnemonic and, thus, easier to recall. Furthermore, the information providers' maps could contain arbitrarily many entries and each user's map may contain up to some fixed number of entries. † When a user issues a keyword directive, either the desired map could be indicated explicitly (e.g., "INDEX" *vs.* "#INDEX" *vs.* "*INDEX") or, similarly to Teletel/Star's indices, the maps could be searched in some prespecified order (e.g., *first* the current page *then* the user's map *then* the information provider's map). Such a system may eventually reduce the need for printed catalogs of information providers' pages.

Having removed page identifiers from the users' realm, it remains to remove them from the information providers' as well. In fact, information providers access pages in a manner similar to their users; thus the absolute labels together with relative cross-links should suffice. The only remaining use for page identifiers is to build the maps in the first place. Again the experience from programming language design could be applied: for example, similarly to conventional linked list manipulation, when a page is created, it becomes the current page and it must be linked into some access structure before proceeding to another page. Using the proposed facilities, it must either be entered into a label map as explained for users above, or it must be entered into some other page's cross-link map by the use of a suitable notational convention, e.g., "5@#3" could mean to assign it as the fifth cross-link on the page denoted by absolute label 3 and "7@2@INDEX" could indicate that the new page is to be linked as the seventh cross-link on the page accessed from the second cross-link on the page labelled INDEX.

† Depending on the system's cost to maintain label maps, the information providers and users could be charged for their maps according to the number of entries, as they are now charged for page storage and page accesses, respectively.

4.3. Providing conventional record-oriented facilities

It seems strange that although data base technology has advanced significantly over the years, videotex systems are still based on facilities that closely resemble early file systems. Surely the research and development devoted to conventional record-oriented structures can be applied to videotex facilities.

Consider, for example, a real estate application for which pages have been constructed to contain photographs, floor plan diagrams, and textual descriptions of houses available from some agent acting as an information provider. Current systems are not designed to service typical user requests from such a service. All the current videotex systems provide page access through unique identifiers only; that is, each user directive identifies at most one target page. However, the typical search for houses is specified by a boolean expression over a set of secondary keys that are not unique identifiers, e.g., "three bedrooms and a formal dining room and priced between $50000 and $70000." In fact, this is exactly the sort of request best handled by conventional data base systems.

The application of conventional data base technology to videotex systems can be recognized as soon as it is realized that "videotex page" may be made to be a valid domain in a record-oriented system. In particular, using the relational model[4] as an example videotex pages could be used for one or more attributes in a relation. For the real estate example, a relation could be defined in which the attributes are *number of bedrooms, price, dining room type, address,..., layout, picture, description* where the first two attributes' domains are integers, the next two are character strings, and the last three are videotex pages (which would not normally be part of a query expression). Thus conventional relational processing could retrieve the set of addresses, layouts, and pictures for which *number of bedrooms* = 3, *dining room type* = "formal" and $50000 \leqslant price \leqslant 70000$.

Rather than completely rewriting the videotex page management systems and the conventional record processing systems in order to merge the two, a distributed architecture similar to that depicted in Figure 7 may be beneficial. In the real estate example, a user would request a house inquiry page from the videotex system, such a page having been constructed as a response page by the information provider. This page would then be filled in by the user in some convenient notation and format (the use of Query-by-Example[17] or some other forms-oriented language may well lend itself to such specification), after which it is passed to the conventional data base system for processing. The response is then passed back to the videotex access machine, which either requests the target pages from the videotex page depository, requests more information from the user, or displays a dynamically constructed index page for further selection by the user. † Further discussion of this architecture can be found elsewhere.[2]

† This sort of capability already exists in many bibliographic retrieval systems as well as in Teletel/Star's electronic directory, for which the responses are displayed directly if they are few enough in number or else further information is requested from the user.

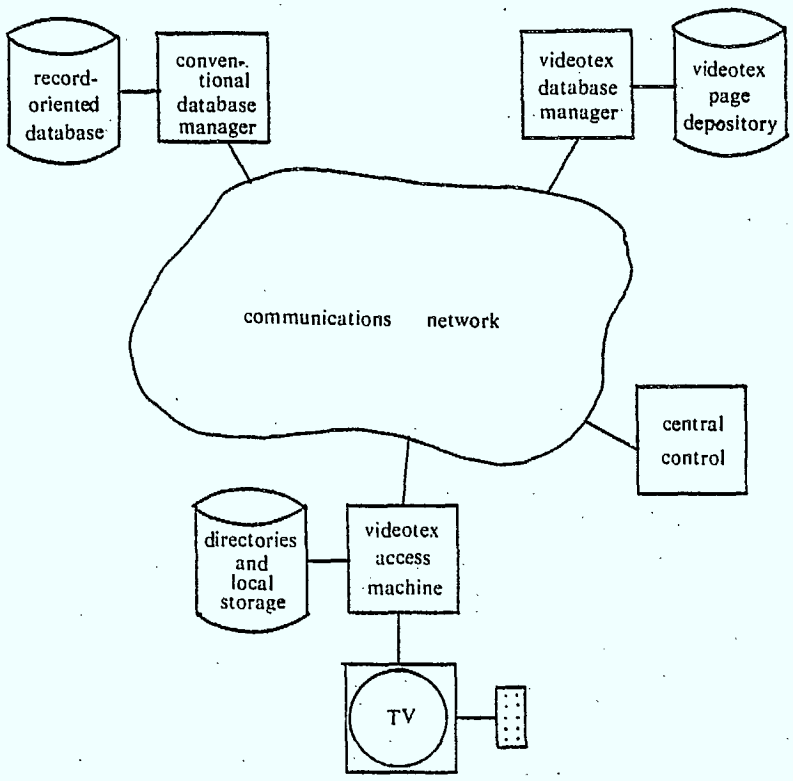


Figure 7: Schematic for an architecture combining videotex page management and conventional record processing.

5. CONCLUSIONS

In this paper we have shown that the data structuring facilities available in current videotex systems are still relatively primitive and need more attention. The Teletel/Star experimental system has shown that simple concepts, the use of keywords and the association of an executable program with each page, can lead to far more sophisticated and user-oriented facilities. Whereas the provision of keywords does not expand the functionality of videotex systems, the program facility provides an opportunity for user interaction that goes beyond the simple selection of pages. Much more research and development is still warranted.

We feel that the experience gained by those who have dealt with conventional programming languages, graphics, operating systems, and data base systems should be applied to the future design of videotex systems. For example, we have proposed replacing the underlying tree structure by an array of pages and removing the visibility of page identifiers. In addition the marriage of page-oriented videotex technology with conventional record-oriented retrieval systems seems very promising.

Acknowledgments

Most of the material in this paper was collected or developed during the First Montréal Workshop on Videotex Technology sponsored by the Université de Montréal and the Canadian Natural Sciences and Engineering Research Council under grant G0363. We wish to acknowledge the contributions of the other attendees, especially A. Ball, B. Botten, D. Leahy, D. Le Moign, and A. Turpin, as well as the contributions of our colleagues who could not attend. We are particularly indepted to Daniel Le Moign for his careful explanation of Teletel/Star. The financial support of the Canadian Natural Sciences and Engineering Research Council under grants A9292 and G0363 and the Canadian Department of Communications under project 908-01 are appreciatively acknowledged.

References

1. M. Aysan, "Project IDA: home of the future," *Inside Videotex*, Infomart, Toronto, Canada, 1980, pp. 60-75.
2. A. S. Ball, G. V. Bochmann, and J. Gecsei, "Videotex networks," *IEEE Computer*, Vol. 13, No. 12, December 1980.
3. R. D. Bright, "The télématique program in France," *Viewdata 80*, Online Conf. Ltd., London, UK, 1980, pp. 19-24.
4. D. D. Chamberlin, "Relational data-base management systems," *ACM Computer Surveys*, Vol. 8, No. 1, March 1976, pp. 43-66.
5. J. J. Coyne, "OMNITEL^{TM} – an integrated broadband distribution system for the eighties," Coyne Associates Systems Consultants, Ltd., Winnepeg, Canada, 1980, 15 pp.
6. D. Godfrey and D. F. Parkhill (eds.), *Gutenberg 2*, Press Porcepic, Toronto, Canada, 1979, 231 pp.
7. A. Henriot and J. Yclon, "Langage de programmation des bases de données Star," CCETT Note Technique RSI/41/443/79, Rennes, France, December 1979, 39 pp.
8. G. T. Hopkins, "Multinode communications on the Mitrenet," *Proc. of the Local Area Comm. Networks Symp.*, Mitre Corp. and NBS, May 1979, pp. 169-177.
9. E. Lee and S. Latrémouille, "Evaluation of tree-structured organization of information on Telidon," Telidon Behavioural Research I, Dept. of Communications, Ottawa, February 1980.
10. D. Le Moign, "Presentation de Star," CCETT Note Technique RSI/NT/23/17/80, Rennes, France, March 1980, 13 pp.
11. P. McFarland, "Videotex 1980: state of the art in Britain," *Inside Videotex*, Infomart, Toronto, Canada, 1980, pp. 20-24.
12. D. F. Parkhill, "Videotex 1980: state of the art in Canada," *Inside Videotex*, Infomart, Toronto, Canada, 1980, pp. 12-18.

13. D. A. Phillips, "Telidon and the human factors of videotex data bases," *Proc. of the Sixth Inter. Conf. on Very Large Data Bases*, IEEE, ACM, and CIPS, October 1980, pp. 330-331.

14. E. Sigel (ed.), *Videotex: The Coming Revolution in Home/Office Information Retrieval*, Knowledge Industries Publ., White Plains, N.Y., 1980, 154 pp.

15. K. H. Taylor, "On-line business databases and viewdata," *Proc. Online 79*, Online Conf. Ltd., London, UK, 1979, pp. 273-282.

16. K. Yasuda, "Conception of Captain system — background, experiment, and future plans," *Viewdata 80*, Online Conf. Ltd., London, UK, 1980, pp. 107-111.

17. M. M. Zloof, "Query-by-Example: a data base language," *IBM Systems J.*, Vol. 16, No. 4, 1977, pp. 324-343.

APPENDIX D: SOFTWARE FOR THE SIMULATION OF PAGE CACHE STRATEGIES AND VIDEOTEX FILE STRUCTURES

```
program TelidonCacheSimulation(input,output);

const
        scache = 20000; { capacity of the cache in characters }
        ecache = 100;   { maximum number of entries in cache }
        maxusers = 100; { maximum number of users }
        NoPages = 50000;        { Total number of Pages of the Data Base }
        IndexSize = 400;        { Size of an index page (chars). }
        IndOccup = 0.80;        { Occupation factor of the index pages }
        MaxTreeLevel = 6;       { Maximum level of Telidon Tree }
        IntArrTime = 1.0;       { Mean interarrival time (when not supplied }
        Pmean = 800.0;          { Mean data-page size (when not supplied) }
        Pstddev = 250.0;        { Standard deviation of data-page size }


type
        link = record
                pageid, absaddr : integer;
                end;

        links = record
                parent, next, previous : link;  { tree links }
                menu : array [1..20] of link;   { menu selection links }
                end;

        page = record
                Plinks : links;
                pdi : array[1..10] of char      { contents of a telidon page }
                end;

        mapping = record
                { Here we will assume it is a B-tree or Hash table }
                { The B-tree/Hashing page contents are omitted }
                end;

        Rtype = ( Tpage, Tmapping );


var

        { Cache variables }
        icache : 0..ecache;     { number of entries in cache }
        storused : integer;     { total storage in use }
        cache : array [1..ecache] of record
                                pageid : integer; { page or mapping number }
                                size : integer;
                                TimesUsed : integer;
                                TimeInCore : real;
                                LastTimeUsed : real;
                                case CacheType : Rtype of
                                Tpage : ( Ppage : ^ page );
                                Tmapping : ( Pmapping : ^ mapping )
                                end;
```

```
{ User variables }
iuser : 0..maxusers;    { number of users }
user : array [1..maxusers] of record
                        Unumber : integer;      { user number }
                        Upageid : integer;      { current page }
                        Ulinks : links; { User relative location }
                        end;

{ MAIN PROGRAM VARIABLES }
i, genseed : integer;
CurrUser, CurrComm, CurrPageid, CurrSize : integer;
iu : integer;
Time, CurrTime : real;
SearchMethod : ( Hashing, BTree );
{ These should be constants, but in Pascal we have to compute them }
EntrInd, BtreeHeight, HashNoBuckets, MaxBtree : integer;
Prob2Acc : real;

{ STATISTIC GATHERING VARIABLES }
Srequests : integer;    { total number of requests }
SreqType : array [-3..20] of integer; { total requests per type }
Stime : real;   { total time elapsed }
SindDisk : integer;     { number of index requests to disk }
SdatDisk : integer;     { number of data-page requests to disk }
SdiskChar : integer;    { number of chars (data-page) read from disk }
SdatCache : integer;    { number of data-pages requests to cache }
SindCache : integer;    { total index requests to cache }
SsizeRepl : integer;    { total entries in cache at replacement time }
SreplCache : integer;   { total number of pages deleted from cache }


{ F U N C T I O N S }
{ ------------------------- }

function random ( var seed : integer ) : real;
{ Uniform Random number generator   U(0,1).
This version should update the seed.    }
begin
seed := seed*65539+1;
if seed<0 then begin seed:=seed+2147483647; seed:=seed+1 end;
random := seed/2147483648.0
end;

function normal ( seed : integer ) : real;
{ Generate a normal random number with mean 0 and variance 1
N(0,1). }
var     tot : real;
        i : integer;
begin
{ small seeds are very bad behaved, so we arbitrarily increase it }
seed := abs(22222*seed+1);
tot := -6.0;
for i := 1 to 12 do tot := tot+random(seed);
normal := tot
end;
```

```
function InCache ( id : integer ) : boolean;
{ Search in cache for page id, either data page or index page
        and perform corresponding updates }
var     i : integer;
begin
if id<0 then SindCache := SindCache+1 else SdatCache := SdatCache+1;    {Stat}
i := 1;
while (i<icache) and (cache[i].pageid<>id) do i := i+1;
if cache[i].pageid = id then begin
        writeln('-->in InCache, id=',id:6,' found');    {debug line}
        cache[i].TimesUsed := cache[i].TimesUsed+1;
        cache[i].LastTimeUsed := CurrTime;
        InCache := true
        end
else InCache := false;
end;

function replacement : integer;
{ Find one entry to be deleted from the cache }
var
        i : 1..ecache;
        index, ind : real;
begin
index := 1.0e20;        { very large value }
SsizeRepl := SsizeRepl+icache;  {Stat}
SreplCache := SreplCache+1;     {Stat}
i := 1;
while i<=icache do begin
        ind := cache[i].TimesUsed/
                (cache[i].size*(CurrTime-cache[i].TimeInCore+0.0001));
        if index>ind then begin
                index := ind;
                writeln('-->in replacement, ind=',ind,' id=',cache[i].pageid:6);        {debug line}
                replacement := i
                end;
        i := i+1
        end
end;

procedure ReadPage ( id, s : integer );
{ This procedure simulates reading in a new page.
        It follows these steps:
        Release pages as necessary to make space
        Update information of new page.
if id>0 this is a data page, if id<0 it is an index page. }
var     i : integer;
begin
writeln('-->in ReadPage, will read id=',id:6,' of size',s:5);   {debug line}
while (storused>0) and (storused+s > scache) do begin
        i := replacement;
        storused := storused - cache[i].size;
        cache[i] := cache[icache];
        icache := icache-1
        end;
if storused+s > scache then writeln('ERROR - cache too small');
icache := icache+1;
```

```
        with cache[icache] do begin
                size := s;
                TimesUsed := 1;
                TimeInCore := CurrTime;
                LastTimeUsed := CurrTime;
                pageid := id;
                if id < 0 then begin
                        CacheType := Tmapping;
                        SindDisk := SindDisk+1  end     {Stat}
                else begin
                        CacheType := Tpage;
                        SdatDisk := SdatDisk+1; {Stat}
                        SdiskChar := SdiskChar+s        end
                end;
        storused := storused+s
        end;

        procedure SearchPage ( id : integer );
        { This procedure simulates the search strategy, searching an
                index, to find the disk address of a page.
        Two strategies are implemented now: B-trees and Hashing }
        var     i, ip, base : integer;
                rel : real;
        begin
        writeln('-->in SearchPage, id=',id:6);  {debug line}
        case SearchMethod of
        Hashing : begin
                { A small seed makes is bad, so enlarge it }
                id := abs(2222*id+1);
                { Decide probe position }
                ip := trunc(random(id)*HashNoBuckets+1);
                {if not InCache(-ip) then ReadPage(-ip,IndexSize);}
                SindDisk:=SindDisk+1;
                { Decide if extra probe is needed }
                if random(id)<Prob2Acc then
                        {if not InCache(-ip-1) then ReadPage(-ip-1,IndexSize);}
                        SindDisk:=SindDisk+1;
                end;
        BTree : begin
                { Find relative location in Btree }
                rel := id/1000000*NoPages/MaxBtree;
                base := 1;
                for i := 1 to BtreeHeight do begin
                        ip := -trunc(rel)-base;
                        if not InCache(ip) then ReadPage(ip,IndexSize);
                        rel := rel*EntrInd;
                        base := base*(EntrInd+1)
                        end
                end
        end
        end;

        function gensize ( pageid : integer ) : integer;
        { Generate (or find) size for a page identified by pageid.
                It would be very nice if the same pageid always
```

```
        { Currently is generated as a normal variable with
                parametric mean and standard deviation }
        var i : integer;
        begin
        i := 1;
        while (i<icache) and (cache[i].pageid<>pageid) do i := i+1;
        writeln('-->in gensize, id=',pageid:6,' i=',i:3,' icache=',icache:2);   {debug line}
        if cache[i].pageid = pageid then gensize := cache[i].size
                else gensize := trunc(normal(pageid)*Pstddev+Pmean)
        end;

        function genpageid : integer;
        { Generate a random number that can be used as page id
                1,2,3.., or 6 digit number      }
        var     t : real;
        begin
        t := random(genseed)*9+1;
        for i:=1 to trunc( random(genseed)*6 ) do t := t*10;
        genpageid := trunc(t)
        end;

        procedure ZeroLinks ( var x : links );
        {       Clear a "links" record  }
        var i : integer;
        begin
        x.parent.pageid := 0;   x.parent.absaddr := 0;
        x.next.pageid := 0;     x.next.absaddr := 0;
        x.previous.pageid := 0; x.previous.absaddr := 0;
        for i:=1 to 20 do begin
                x.menu[i].pageid := 0;  x.menu[i].absaddr := 0  end
        end;


begin
{ Initialize variables }
icache := 0;
storused := 0;
iuser := 0;
genseed := 7777777;
Time := -1;
SearchMethod := BTree;
{ Average number of entries per index-page, assume 8 chars per entry }
EntrInd := trunc( IndexSize*IndOccup/8 );
{ Height of B-tree }
BtreeHeight := trunc( ln(NoPages)/ln(EntrInd) + 0.999 );
HashNoBuckets := trunc( NoPages/EntrInd + 0.999 );
MaxBtree := EntrInd;
for i:=2 to BtreeHeight do MaxBtree := MaxBtree*EntrInd;
{ Probability of two accesses for Hashing table }
Prob2Acc := IndOccup/(1-IndOccup)/2/(IndexSize/8);
Srequests := 0; Stime := 0;
for i:=-3 to 20 do SreqType[i] := 0;
SindDisk := 0;  SdatDisk := 0;  SdiskChar := 0; SdatCache := 0; SindCache := 0;
SsizeRepl := 0; SreplCache := 0;
```

```
{ The input to this simulation program is in the standard input file;
        it contains one record per request with the following information:
        (a) User identification (integer)
        (b) Type of request
                1..20 is menu selection,
                -1 is ancestor,
                -2 is next,
                -3 is previous
                0 is absolute request by page id.
        (c) Requested page id number (0 if it is not known)
        (d) Size of page requested (0 if not known)
        (e) Time (decimal) of request.
                                                }

{--------------------}
{ M A I N    L O O P  }
{--------------------}
while not eof(input) do begin
        readln(CurrUser,CurrComm,CurrPageid,CurrSize,CurrTime);
        Srequests := Srequests+1;       {Stat}
        { Adjust time }
        if Time=-1 then Time := CurrTime;
        if (CurrTime=0) or (CurrTime<Time) then
                Time := CurrTime + IntArrTime*ln(random(genseed));
        Stime := Stime + CurrTime-Time; {Stat}
        Time := CurrTime;
        if (CurrComm<-3) or (CurrComm>20) then writeln('error in input file');

        {       Locate user in User table       }
        iu := 1;
        while (iu<iuser) and (user[iu].Unumber<>CurrUser) do iu := iu+1;
        if user[iu].Unumber <> CurrUser then begin
                {       New user entry  }
                iuser := iuser+1;       iu := iuser;
                user[iu].Unumber := CurrUser;
                user[iu].Upageid := 0;
                ZeroLinks(user[iu].Ulinks);
                { For new user, force request of absolute page 1 }
                CurrComm := 0;  CurrPageid := 1;
                end;


        SreqType[CurrComm] := SreqType[CurrComm]+1;     {Stat}
        { If page id is not specified, generate according to command
                and previous data }
        if CurrPageid = 0 then
                if CurrComm = -1 then
                        CurrPageid := trunc(user[iu].Upageid/10)
                else if CurrComm = -2 then CurrPageid := user[iu].Upageid+1
                else if CurrComm = -3 then CurrPageid := user[iu].Upageid-1
                else if (CurrComm<=10) and (CurrComm>0) then
                        CurrPageid := user[iu].Upageid*10+CurrComm-1
                else CurrPageid := genpageid;
        { If out of bounds, make it random }
        if (CurrPageid<1) or (CurrPageid>999999) then CurrPageid := genpageid;
        user[iu].Upageid := CurrPageid;
```

```
        if CurrSize = 0 then CurrSize := gensize(CurrPageid);

        { Get requested page into the cache }
        if not InCache(CurrPageid) then begin
                if CurrComm = 0 then SearchPage(CurrPageid);
                ReadPage(CurrPageid,CurrSize)
                end;


        {       write read and generated information    }
        writeln('User=',CurrUser:2,'    Command=',CurrComm:3,   {debug line}
                '    Page Req=',CurrPageid:6, '    Page Size=', {debug line}
                CurrSize:6, '    Time=', CurrTime:7:4); {debug line}
        end;

{ STATISTIC PRINTING }
writeln;        writeln;
writeln('Model Description --------');
writeln('Total size of data-pages in database',NoPages:7);
writeln('Size of internal cache memory (chars)',scache:6);
writeln('Size of index-page (chars) is',IndexSize:5);
writeln('The index occupation factor is',IndOccup:5:2);
writeln('Average number of entries per index-page is',EntrInd:3);
write('The index is organized as a ');
case SearchMethod of
        Hashing : begin writeln('hashing table');
        writeln('Number of buckets of hashing table is',HashNoBuckets:5) end;
        BTree: begin writeln('B*-tree');
                writeln('Height of B*-tree is',BtreeHeight:2)   end;
        end;
writeln;        writeln;
writeln('Input Statistics ---------');
writeln('Total number of different users',iuser:3);
writeln('Total number of page requests received',Srequests:6);
writeln('Percentage of ancestors requested',SreqType[-1]/Srequests*100:7:2);
writeln('Percentage of "next page" requested', SreqType[-2]/Srequests*100:7:2);
writeln('Percentage of "previous page" requested', SreqType[-3]/Srequests*100:7:2);
writeln('Percentage of direct page requests', SreqType[0]/Srequests*100:7:2);
writeln('Percentage of menu-type requests by menu-option');
write('(descending tree) ');
for i:=1 to 10 do write(SreqType[i]/Srequests*100:6:2);
writeln;
write('(arbitrary link) ');
for i:=11 to 20 do write(SreqType[i]/Srequests*100:6:2);
writeln;
writeln('Total time elapsed ',Stime:8:4);
writeln;        writeln;
writeln('Performance Statistics --------');
if Stime>0 then
writeln('Average number of requests per unit time ', Srequests/Stime:8:4);
writeln('Number of pages requested to the cache ',SdatCache:6);
writeln('Number of index-pages requested to cache ',SindCache:6);
writeln('Number of pages requested to disk ',SdatDisk:6);
writeln('Number of index-pages requested to disk ',SindDisk:6);
if Srequests>0 then
writeln('Disk requests per page requested',(SindDisk+SdatDisk)/Srequests:7:4);
```

```
writeln('Percentage of requests satisfied within the cache');
if (SindCache>0) and (SdatCache>0) then
        writeln('  pages=',100*(SdatCache-SdatDisk)/SdatCache:5:2,
                '    index-pages=',100*(SindCache-SindDisk)/SindCache:5:2);
writeln('Number of data-characters read from disk ',SdiskChar:9);
if SreplCache>0 then
        writeln('Average number of entries in cache at replacement ',
                SsizeRepl/SreplCache:5:2);
writeln('Number of replacements in the cache ', SreplCache:6);
end.
```