NTIA Report 82-112

User-Oriented Performance Measurements on the ARPANET: The Testing of a Proposed Federal Standard

D. R. Wortendyke
N. B. Seitz
K. P. Spies
E. L. Crow
D. S. Grubb

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

Bernard J. Wunder, Jr., Assistant Secretary
for Communications and Information

November 1982

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF FIGURES

LIST OF FIGURES (cont'd)

LIST OF TABLES

USER-ORIENTED PERFORMANCE MEASUREMENTS ON THE ARPANET: THE TESTING OF A PROPOSED FEDERAL STANDARD

D. R. Wortendyke, N. B. Seitz, K. P. Spies, E. L. Crow, and D. S. Grubb*

This report presents the results of a trial implementation of a newly developed data communication performance measurement methodology which has been proposed as Federal Standard 1043. In this experiment, a prototype data communication performance measurement system was developed in accordance with specifications defined in the standard. The system was used to assess the data communication service provided to a typical pair of ARPANET end users (host computer application programs). These user-oriented measurements differed from earlier ARPANET measurements in that the host computer operating systems and network control programs were regarded as providers of an end-to-end data communication service, rather than as users of the subnetwork.

Results of the experiment will be useful in three ways. First, the prototype performance measurement system developed in this experiment will facilitate future implementations of the measurement standard. Second, the experience of implementing the measurement standard identified a number of ways in which that standard could be improved. These improvements will be incorporated in a future revision. Finally, the user-oriented performance values measured in this experiment will assist communication system planners in relating end-to-end performance objectives to the performance of subsystems.

Key words: American national standard; ARPANET; computer networks; data communications; end user; federal standard; performance measurement

1. INTRODUCTION

New applications for data communication services and the trend towards competition and deregulation in the telecommunications industry have created a need for uniform methods of specifying and measuring the performance of data communication services as seen by the end user. Over the past 5 years, standards groups in the Federal government and industry have been working together to meet that need through the development of user-oriented, system-independent performance parameters and measurement methods. Results are

---

*D. R. Wortendyke, N. B. Seitz, K. P. Spies, and E. L. Crow are with the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, Boulder, CO 80303.

D. S. Grubb is with the Institute for Computer Sciences and Technology at the National Bureau of Standards, Washington, DC 20234.

expected to be promulgated within the Federal government in the form of Federal Telecommunication/Federal Information Processing Standards, and in industry in the form of American National Standards.

Within the Federal Government, the major responsibility for developing data communication performance standards is executed by staff from the Commerce Department's National Telecommunications and Information Administration/Institute for Telecommunication Sciences (NTIA/ITS) and the Institute for Computer Sciences and Technology at the National Bureau of Standards (NBS/ICST). The ICST has overall responsibility for developing Federal Information Processing Standards (FIPS) to promote the economic and efficient use of computer technology in the Federal Government. The National Communications System (NCS) Office of the Director promulgates Federal Telecommunication Standards (FTS).

Industry work on data communication performance standards is centered in the American National Standards Institute (ANSI), Task Group X3S35. That group includes representatives from the common carrier, data communications, and data processing industries as well as representatives of the Federal government. Its objective is to develop a set of industry/government standards for specifying and measuring data communication performance from an end user perspective.

Two related data communication performance standards have been developed. The first specifies a set of user-oriented performance descriptors or parameters. That standard was initially approved in 1979, as Interim Federal Standard 1033 (GSA, 1979; Seitz, 1980a). It has since been applied in a number of trial Federal procurements (EPA, 1980; USDA, 1980); and a similar proposed industry standard has been developed by ANSI Task Group X3S35 (ANSI, 1982). The latter standard, designated X3.102, is currently undergoing review and formal balloting within ANSI. It is planned to ultimately replace Interim Federal Standard 1033, possibly as a joint Federal Telecommunications/Federal Information Processing Standard (NBS, 1981).

Federal Standard (FED STD) 1033 and its ANSI counterpart are unique in providing a set of performance descriptors that can be applied to any digital communication service, irrespective of system design features such as topology and control protocol. This property of system independence makes the parameters useful as a "common language" for relating the performance needs of data communications users with the capabilities of offered systems and services. Serious deficiencies in Federal communications management and

increasing emphasis on competitive procurement demonstrate that such a capability is needed in the Federal government (GAO, 1977; GSA, 1980). Similar needs exist in many non-Federal user organizations as well.

The second standard, proposed by the National Communications System's Federal Telecommunications Standards Committee (FTSC) as Federal Standard 1043, is the focus of this report. That standard defines uniform measurement methods to enable users to obtain representative numerical values for the Interim FED STD 1033 performance parameters (Seitz et al., 1981a, 1981b). It was developed by the NTIA/ITS in 1980, and is currently being reviewed in government and industry as a first step towards approval as an Interim Federal Standard.

The proposed measurement standard will promote innovation and fair competition in the data communications industry by providing users with a practical method of measuring delivered performance. Improved ability to measure performance will enable users to make more intelligent choices among service and equipment alternatives, and will lead, in many cases, to more realistic communication requirements. An independent National Research Council committee has estimated that a 20% reduction in total Federal data communication costs could be realized through the promulgation of an efficient method of selecting the right system or service for a given need - a potential user saving in excess of $400 million per year by the mid-1980's (NRC, 1977). This work is intended to contribute to the realization of these savings.

1.1 Purpose and Scope of Report

Because of the scope and potential impact of the measurement standard, a realistic trial implementation of the standard at the draft stage was considered essential. In October of 1979, NTIA/ITS and NBS/ICST began such a trial implementation, using the Defense Communications Agency's ARPA computer network as a test bed. In this experiment, a prototype data communication performance measurement system was developed in accordance with specifications defined in the standard. The prototype system was then used to assess the data communication performance provided to a typical pair of ARPANET end users (host computer application programs).

The purpose of this report is to summarize the results of that experiment. It is anticipated that these results will be useful in three ways. First, the prototype performance measurement system developed in this experiment will serve as a model for future implementations of the measurement

standard. Two of the four performance measurement subsystems developed in this experiment were implemented in machine-independent software which should be executable in virtually any modern computer system. The other two measurement subsystems are inherently application dependent, but the hardware, software, procedures, and techniques used in the prototype implementation are typical of those which would be used in other applications. The existence of these prototype subsystems will substantially reduce the time and effort required to develop future measurement systems.

Results of the measurement standard prototype implementation will be useful, secondly, in improving the measurement standard itself. The experiment of implementing that standard identified a number of ways in which it could be clarified, and also produced an abundance of practical examples which will be helpful in illustrating its use. These improvements will be incorporated in a future revision.

The results of the measurement standard prototype implementation will also be useful in a third way - as an aid to system planners in the difficult process of relating end-to-end performance objectives to the performance of subsystems. As discussed later, most previously reported data communication performance measurements actually address the performance of a subsystem (e.g., the ARPA IMP/TIP subnetwork) rather than the performance of the end-to-end data communication system (e.g., ARPA subnetwork plus host computer network access and control software). The difference between subsystem and end-to-end performance can be substantial, and often profoundly influences end user perceptions of a subnetwork service. The end-to-end measurement results presented in this report will assist data communication system planners in assessing such differences.

The experimental, user-oriented nature of these measurements necessarily imposed certain limitations on the applicability of the measured values. First, these values are usage dependent. They characterize the information path between a particular pair of host computer application programs utilizing the ARPA network during the period September 17 to November 24, 1981. The values are probably similar to those provided to many other network users, but they do depend on the application program usage; and users with different usage patterns might observe substantially different performance. The "ancillary" performance parameters, discussed later in this section, provide a method of "factoring out" much of this usage dependence. A more comprehensive

performance characterization, not contemplated at this time, would involve classification of user applications, random selection of user pairs, and a much more extensive instrumentation and data collection effort.

A second limitation on the reported performance values is the relatively small measurement sample on which they were based - approximately 3,000 data communication sessions and 10^7 transmitted bits observed over 2 calendar months. The collected data provide reasonable confidence in the values for most parameters, but are not sufficient to show trends in service performance or similar second-order effects. In a few cases (e.g., Bit Error Probability), the measured sample sizes were clearly insufficient to accurately characterize the long-term system performance. Most of these sample inadequacies were anticipated, and they are not a major concern in view of the experimental nature of the measurement project. They could, of course, be reduced by more extensive data collection.

A third limitation on the reported performance values, from the point of view of a system planner, is their restriction to a single service interface. The measurements would have been more useful in performance allocation if they had been made both at the end user interfaces and at the host/TIP interfaces. The latter measurements were not undertaken in the present experiment because of resource limitations, but there is no restriction in either of the standards that would prevent them. Considerable insight into the differences between end-to-end and subnetwork performance can nevertheless be gained by comparing the results presented here with those of earlier subnetwork measurements (e.g., Kleinrock, 1976).

1.2 Report Organization

This report is divided into seven major sections. Section 2 summarizes the ARPA network and each of the three data communication performance standards which were used in its measurement. Section 3 defines the overall objectives of the ARPA network measurements project in the form of a series of questions to be answered by the project results. Section 4 describes the detailed technical approach used in implementing the prototype performance measurement system. Section 5 summarizes the overall conduct of the measurements and describes, step by step, the operator procedures that were followed in each measurement phase. Section 6 presents the measurement results in the form of answers to the questions posed in Section 3. Section 7

summarizes the major conclusions and recommendations developed during the experiment.

The report also includes a series of appendices. These contain detailed information which is of interest to specialists in performance measurement, but is not essential to an overall understanding of the experiment. Appendix A presents narrative/symbolic definitions for the Interim Federal Standard 1033 and ANSI X3.102 performance parameters. Appendix B presents flowcharts and listings of the computer programs which controlled on-line data extraction. Appendix C presents examples of each type of measurement data file created during the experiment. Appendix D describes the time correction procedure which was used to compensate for differences between actual event times and the associated clock readings. Appendix E provides directories of the operating system "shell files" and application programs used in the ARPANET experiment. Appendix F describes the "log files" created during on-line data extraction, and Appendix G summarizes and lists the command files used in extracting and processing performance data.

2. CONTEXT OF THE EXPERIMENT

This section defines the overall context of the ARPANET experiment by describing first, the general design of that network, and second, the technical content of the three standards which were used in its measurement. The section is divided into four subsections. The first provides a brief overview of the ARPANET from a user perspective. The second, third, and fourth subsections summarize the overall approach and technical content of Interim Federal Standard 1033, ANSI X3.102, and proposed Federal Standard 1043, respectively. The entire section is tutorial in nature, and may be omitted by readers already familiar with the subjects discussed.

2.1 Overview of the ARPANET

This subsection provides a brief introductory description of the ARPANET for those not familiar with it. For further information, refer to Kleinrock (1976) and the references cited therein. A few additional references of particular significance to performance assessment are cited below.

About 15 years ago, Dr. L. G. Roberts of the Advanced Research Projects Agency (ARPA) proposed the development of a new experimental computer network (Roberts, 1967). The purpose of the network was to interconnect the computer resources at various research centers in such a way that persons and programs

at one research center would have ready access to the data, programs, and computing power at any other center. It was anticipated that the resulting resource sharing would improve the productivity of all users, and provide a substantial reduction in data transmission costs in comparison with systems then available. Key network performance objectives were (1) low transit delays (less than 1/2 second for short interactive messages); (2) high throughput capacity for long file transfers; and (3) reliable, essentially error-free transmission of user data.

These objectives and benefits were largely achieved in the design of the Advanced Research Projects Agency Network. The ARPANET has grown from an experimental, four-node regional network in late 1969 (Roberts and Wessler, 1970) to a fully operational 80-node network which today interconnects over 200 host computers in the continental United States, Hawaii, England, and Norway (DCA, 1980). Total ARPANET traffic is in excess of 1 billion bits per day. A number of similar public resource-sharing networks have been developed in the United States and other countries based on the ARPANET technology (e.g., see Drukarch et al., 1980).

The basic concept which underlies the design of the ARPANET is packet switching. Packet switching is a communication technology in which user messages are fragmented, at a source network node, into smaller message segments called packets. Each packet is relayed through the network independently, in a store-and-forward fashion, with very minimal delays at intermediate nodes (e.g., less than 10 ms). Individual packets within a message may follow different physical routes through the network. At the destination node, the various packets comprising a message are reassembled in the proper order for delivery to the destination user.

The ARPANET is conceptually and physically divided into two parts: the subnetwork, which consists of small, dedicated communication processing computers interconnected by leased digital communication lines; and the hosts, which are larger, user-owned, application-oriented computers attached to the subnetwork communication processors. Two basic types of communication processors exist in the subnet: Interface Message Processors (IMP's), which provide subnetwork access only to host computers; and Terminal Interface Processors (TIP's), which also provide direct subnetwork access to user data terminals. Virtually all leased lines interconnecting the ARPA network IMP's and TIP's operate at 50 kilobits per second (kb/s). Control of the subnetwork is distributed equally among the various subnetwork processors.

Each host and user terminal is connected to the subnetwork via a single IMP or TIP. Each IMP can accommodate up to four hosts; each TIP can accommodate up to three hosts, plus up to 63 terminals. So-called "Pluribus" IMP's and TIP's provide substantially larger host and terminal handling capacities. Hosts are connected to their associated IMP or TIP via one of three types of interface, depending on their physical proximity: Local Host (less than 30 feet between host and IMP or TIP); Distant Host (30 to 2,000 feet); and Very Distant Host (greater than 2,000 feet). Terminals are typically connected to the TIP's via low-speed dial-up or leased telephone lines.

The ARPANET host computers vary widely in hardware and software configuration, but each is provided with a Network Control Program (NCP)[1] which establishes and breaks network connections and transfers information on behalf of user programs, or processes, executing in that computer. A number of high-level communication protocols are also supported in most ARPANET hosts (e.g., TELNET, a terminal operator interface protocol, and FTP, a File Transfer Protocol).

The outer boundaries of the ARPANET are somewhat unclear in the published literature, but "the network" certainly includes the host computer Network Control Programs as well as other host protocols whose only purpose is to facilitate ARPANET communication. A central view of this report is that the host computer operating systems are also part of the end-to-end network from the end user viewpoint, since a typical user can only access network resources via operating system calls. The NCP is, in fact, a part of the operating system in most ARPANET hosts.

Figure 1 shows a topological view of the ARPA subnetwork as it existed in December 1980 (DCA, 1980). The Department of Commerce, Boulder (DOCB) and National Bureau of Standards (NBS) TIP's, which served as the subnetwork entry and exit nodes in this experiment, are highlighted. Also highlighted is the shortest (8 hop) path between these nodes, over which most of the test traffic probably passed. A logical map of the ARPANET, showing the connected host computers, is provided in Figure 2. The DOCB and NBS TIP's are again

---

[1] Some hosts use an alternative program, the Transmission Control Program (TCP).

ARPANET GEOGRAPHIC MAP, DECEMBER 1980

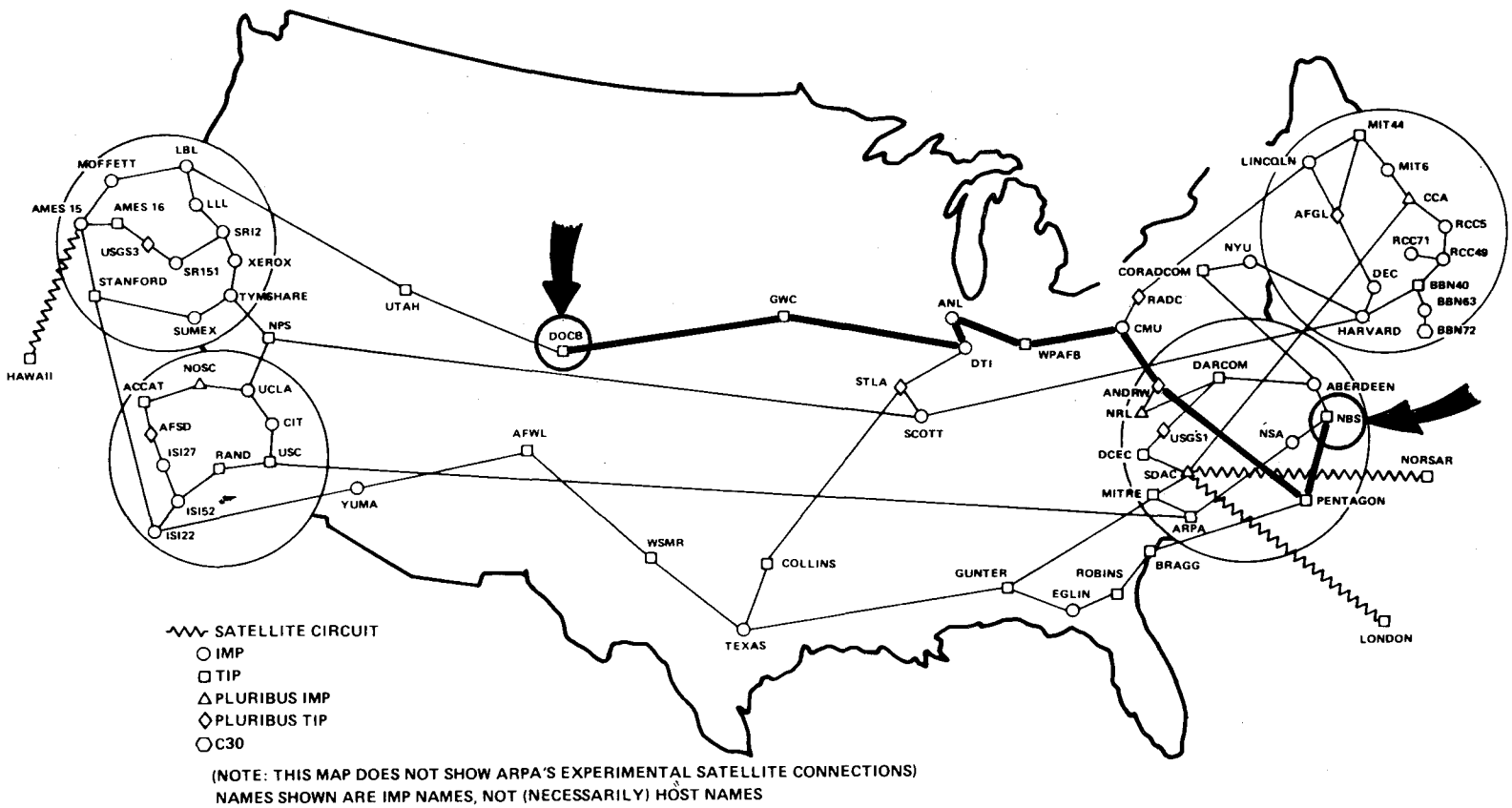


Figure 1. ARPANET Geographic Map, December 1980.

ARPANET LOGICAL MAP, DECEMBER 1980

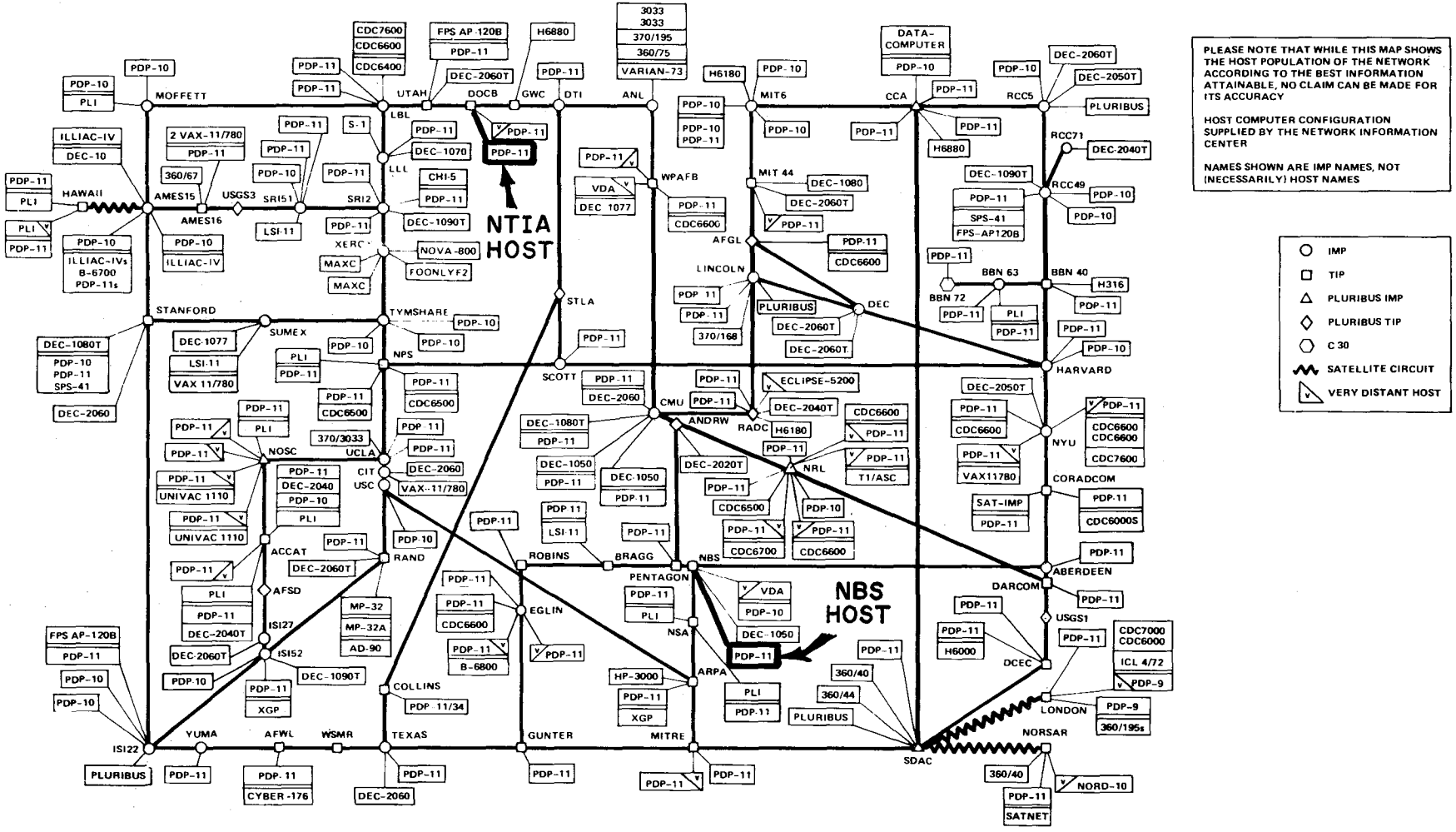


Figure 2. ARPANET Logical Map, December 1980.

highlighted. Both computers used in this experiment were PDP-11's[2] connected via Local Host interfaces.

The ARPANET was chosen as a test bed for trial implementation of the proposed measurement standard for two primary reasons. First, the ARPANET is typical of existing modern data communication networks, many of which employ packet switching. Some experts believe packet switching will be dominant in future networks as well, with gradual enhancements encompassing voice as well as data transmission. Packet switching is being considered very seriously as a principal switching technology for the future Integrated Services Digital Network (Dorros, 1981).

A second reason for the choice of the ARPANET as a measurement test bed was its experimental history. As noted earlier, the ARPANET was conceived and developed as a test of the then-novel concept of packet switching. A number of very powerful network instrumentation features (e.g., trace, snapshot) were built into the original IMP/TIP designs to support this research; and the network's operation and performance were measured, analyzed, and simulated in literally hundreds of technical studies spanning at least a 5-year period.[3] Although many of these studies focus on the performance of the subnetwork, and thus do not consider all the effects of high-level protocols, they nevertheless represent an extremely valuable source of background information for the present study. As noted earlier, a comparison of the end-to-end performance values measured in this experiment with corresponding subnetwork values should provide some good insight into the problem of performance allocation.

2.2 Overview of Interim Federal Standard 1033

This subsection provides a brief summary of the overall approach and technical content of Interim Federal Standard 1033. As noted earlier, this interim standard specifies a set of user-oriented, system-independent data communication performance parameters. It was developed by FTSC Technical

---

[2]Certain commercial equipment and software products are identified in this report to adequately describe the design and conduct of the ARPANET experiment. In no case does such identification imply recommendation or endorsement by the National Telecommunications and Information Administration, nor does it imply that the material or equipment identified is necessarily the best available for the purpose.

[3]For example, see Cole, 1971; Kleinrock and Naylor, 1974; Kleinrock et al., 1976; and the many references cited therein.

Subcommittee 2, under the leadership of NTIA's Institute for Telecommunication Sciences, during the period 1975-1978; was promulgated by FTSC for optional use by Federal agencies in 1979; served as the basis for a corresponding ANSI standard, X3.102; and will ultimately be replaced by the X3.102 standard when the latter is formally approved by ANSI.

Figure 3 summarizes the overall approach used in developing performance parameters for Interim FED STD 1033. The parameter development process consisted of four major steps (Seitz and Bodson, 1980):

1. Model Development. Existing and proposed data communication services were surveyed and certain universal performance characteristics shared by all were identified. These characteristics were consolidated in a simple, user-oriented model which provided a system-independent basis for the performance parameter definitions.

2. Function Definition. Five primary communication functions were selected and defined in terms of model reference events. These functions (access, bit transfer, block transfer, message transfer, and disengagement) provided a specific focus for the parameter development effort.

3. Failure Analysis. Each primary function was analyzed to determine the possible outcomes an individual trial performance might encounter. Possible outcomes were grouped into three general outcome categories: successful performance, incorrect performance, and nonperformance. These categories correspond to the three general performance concerns (or criteria) most frequently expressed by end users: efficiency (or speed), accuracy, and reliability.

4. Parameter Selection. Each primary function was considered relative to each performance criterion in matrix fashion; and one or more specific parameters were selected to represent performance relative to each function/criterion pair.

The specified parameters are of three general types: primary parameters, which provide a relatively specific, short-term view of performance, including both user and system contributions; secondary parameters, which provide the more macroscopic, long-term view of performance traditionally associated with the concept of availability; and ancillary parameters, which describe the influence of user delays on the primary "speed" parameter values.

The performance model used in defining the Interim FED STD 1033 parameters differs from those used in earlier standards and specifications in two major respects. The first is the definition of user/system interfaces. The model defines the end user of a data communication system or service as a human terminal operator, unattended device medium (e.g., punched cards), or

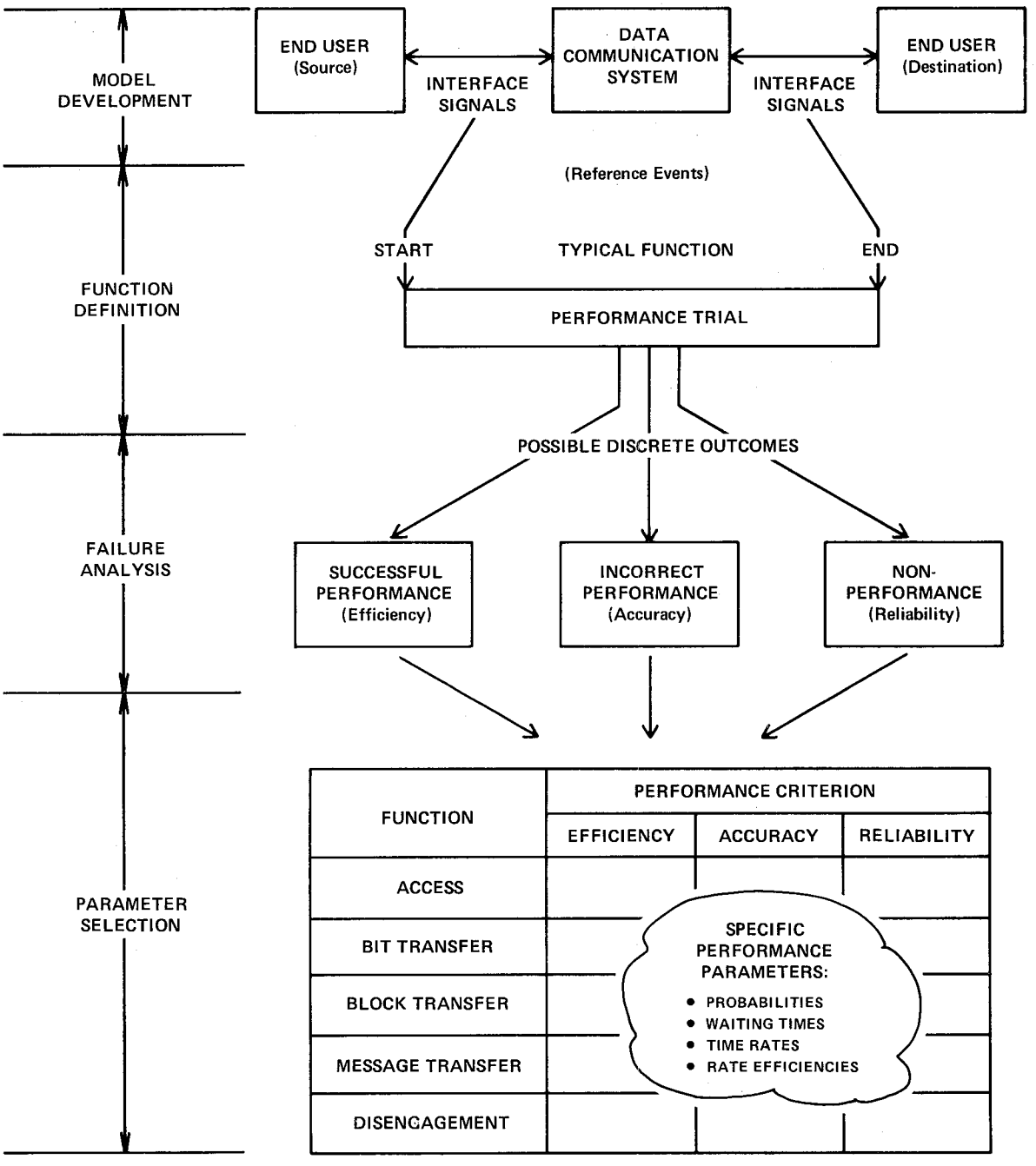


Figure 3. Parameter development overview.

computer application program. Thus, the end-to-end data communication system extends to the operator or medium side of the data terminal, or to the application program side of the host computer operating system. The end-to-end system includes communication control protocols such as the ANSI Advanced Data Communication Control Procedure (ADCCP) and International Telegraph and Telephone Consultative Committee (CCITT) Recommendation X.25.

This viewpoint is essential in a user-oriented standard, since modern terminals and protocols perform functions (such as error control, flow control, and virtual circuit establishment) which have a profound effect on end-to-end performance. One modern data communication network whose end user interfaces are defined in this way is IBM's Systems Network Architecture (McFadyen, 1976). The International Organization for Standardization (ISO) is developing a Reference Model for Open Systems Interconnection (OSI) which follows a similar interface definition approach (ISO, 1980).

The second major difference between the Interim FED STD 1033 model and earlier performance models is the selection of parameter-defining events. In any description of data communication performance, certain information transfers or device state changes are identified as events to be counted, timed, or compared in calculating performance parameter values. As noted earlier, most existing standards and specifications identify such events by reference to particular system-dependent interface signals (e.g., clear to send). The FED STD 1033 model departs from this approach by defining the performance parameters in terms of more general, system-independent reference events. Each FED STD 1033 reference event is a generic event which subsumes many system-specific interface signals having a common performance significance; and each is defined in such a way that it necessarily occurs, at some point, in any end-to-end data communication session.[4]

System-specific interface signals are mapped into FED STD 1033 reference events on the basis of the user interface involved, the type of information transferred (e.g., user information or overhead), and the nature of the state change the transfer produces. One example of a system-independent reference event defined in FED STD 1033 is the start of block transfer.

Any description of performance ultimately refers to some particular function. The second step in developing the Interim FED STD 1033 performance

---

[4]ISO uses the term "service primitive" to express the "reference event" concept in the OSI Reference Model.

parameters was to define the particular functions to be evaluated. The standard defines five primary communication functions, in terms of particular reference events, as follows:

- The access function begins on issuance of an access request signal at the originating user interface, and ends (successfully) on the first subsequent transfer of a user information bit or block from a source user to the system. It encompasses all activities traditionally associated with physical circuit establishment (e.g., dialing, switching, ringing, modem handshaking) as well as any establishment activities performed at higher protocol layers (e.g., layer 3 of X.25). Making the end of access coincident with the start of user information transfer reflects the user view that no data communication service has actually been provided until user information begins to flow.

- The bit, block,[5] and message transfer functions describe the flow of information between end users at three distinct levels of detail. Each function begins on the start of output of the associated information unit from the source user and ends (successfully) on completion of delivery of that unit to the intended destination. Each function encompasses all formatting, transmission, storage, error control, and media conversion activities performed between start of output and completion of delivery, including retransmission if required. All three functions must normally be considered in a comprehensive performance specification, as discussed more fully in Seitz and McManamon (1978).

- The disengagement function begins on issuance of a disengagement request signal at either user interface, and ends (successfully) on return of a corresponding disengagement confirmation signal.

As noted earlier, the terms "access request", "disengagement request", and "disengagement confirmation" are general descriptors of purpose (reference events) rather than specific names of interface signals. They denote, respectively, any event whose purpose is to initiate, terminate, or confirm termination of an entity's participation in a data communication session.

---

[5]As used in the standard, the term "block" denotes a contiguous group of user information bits delimited at the source user/system interface for transfer to a destination user as a unit. Thus, for instance, a block may be a single ASCII character, a card image, a computer word, or the information field of a frame, depending on the equipment and protocol characteristics at the user/system interface. The "message" information unit defined in Interim FED STD 1033 is a performance measurement sample size which does not necessarily correspond to the user information transferred during a single data communication session.

The third major step in developing the Interim FED STD 1033 performance parameters was the failure analysis. In that analysis, the performance of each primary function was regarded as an experiment, or trial, in the statistical sense; and a set of possible outcomes of each performance trial was defined in a "pie" diagram of outcome possibilities called a sample space. Figure 4 shows how this step was accomplished in the case of the access function. The standard defines four possible access outcomes: Successful Access, Incorrect Access, Access Denial, and User Blocking. Successful Access is the case where user information transfer is initiated as intended within a specified maximum access time. Incorrect Access is the case where transfer is initiated within the maximum time, but communication is established with a user other than the one intended by the originator (i.e., a "wrong number"). Access Denial is the case where an access attempt fails as a result of either issuance of a blocking signal or excessive delay by the system. User Blocking is the case where an access attempt fails as a result of either issuance of a blocking signal or excessive delay by a user. Familiar examples of system and user blocking signals are the "circuit busy" and "user busy" signals in the public switched network. User Blocking outcomes are excluded in defining the access performance parameters.

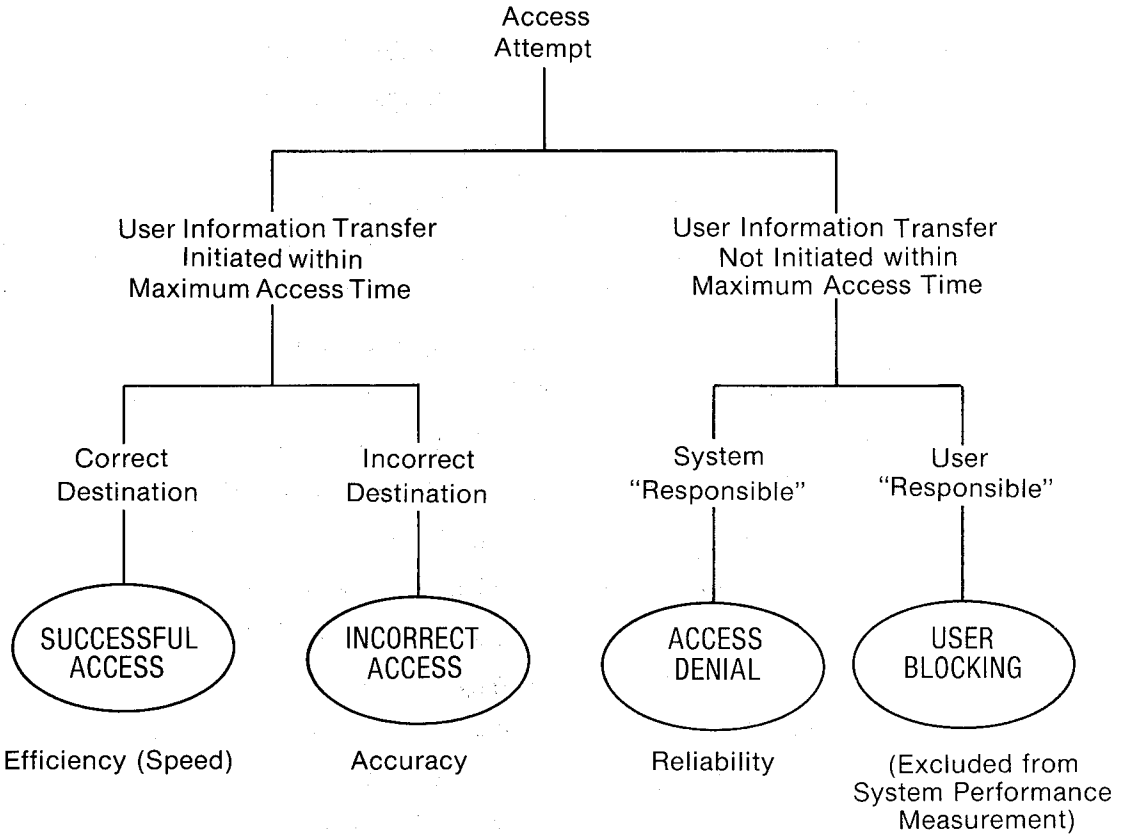
A similar approach was used in defining possible outcomes for the other primary communication functions. In each case, the pertinent outcomes were grouped in three general categories, reflecting the three general user performance concerns noted earlier: efficiency (or speed), accuracy, and reliability.

The final step in developing the Interim FED STD 1033 parameters was to select and define a minimum set of parameters to describe system performance relative to each function and outcome. Figure 5 illustrates how this was accomplished in the case of the access function. Access performance was described in terms of three specific parameters, one associated with each of the three general performance criteria noted earlier. The standard defines the selected access parameters essentially as follows.[6]

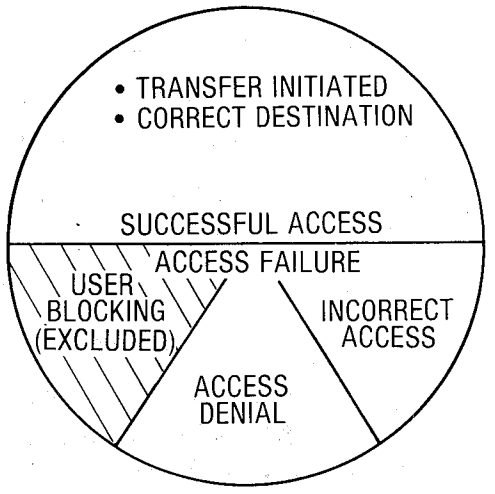
1. Access Time - Average value of elapsed time between the start of an access attempt and Successful Access. Elapsed time values are calculated only on access attempts that result in Successful Access.

---

[6]Terminology and notation differ slightly from that used in the standard.



a. Possible Outcomes of an Access Attempt.



b. Sample Space Representation.

Figure 4. Access outcome definition.

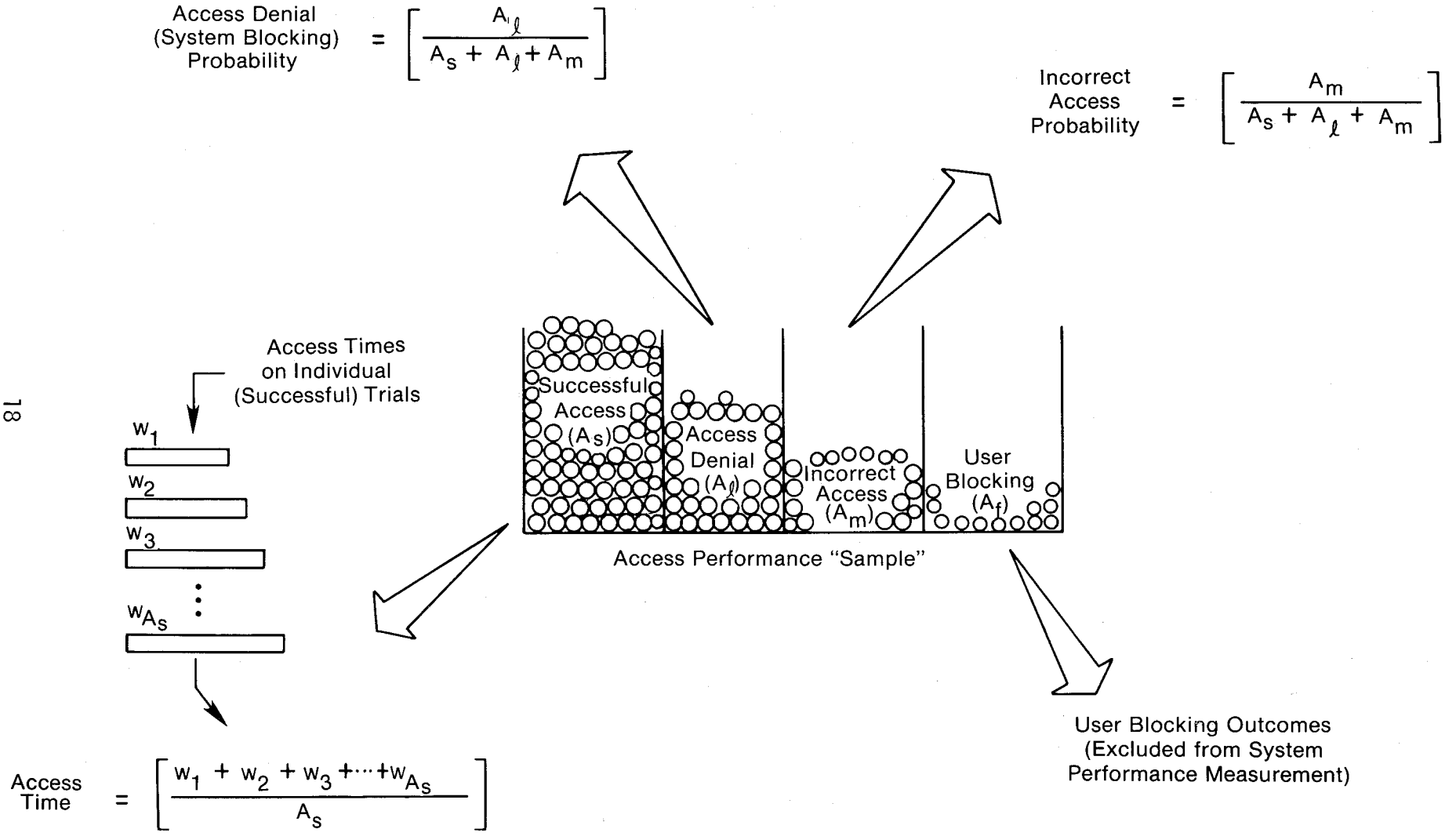


Figure 5. Access parameters.

2. Incorrect Access Probability - Ratio of total access attempts that result in Incorrect Access (i.e., connection to an unintended destination) to total access attempts included in an access performance sample (excluding User Blocking outcomes).

3. Access Denial Probability - Ratio of total access attempts that result in Access Denial (e.g., system blocking) to total access attempts included in an access performance sample (excluding User Blocking outcomes).

A key aspect of the Interim FED STD 1033 parameter definitions is their expression in mathematical form. As noted earlier, this approach eliminates the ambiguity often associated with purely narrative definitions, and also provides a standard procedure for calculating performance parameter values. The mathematical parameter definitions are based, in each case, on the concept of an access performance "sample" - i.e., a large number of successive access trials distributed, like apples, in appropriate outcome "bins". Each Successful Access outcome has an associated elapsed time value (the total time required to complete that particular attempt).

Values for the access parameters may be calculated directly from the data in an access performance sample. The value of the efficiency parameter Access Time is calculated by adding the individual elapsed times (w_i) for all A_s Successful Access outcomes, and then dividing by A_s. The value of the accuracy parameter Incorrect Access Probability is calculated by dividing the total number of Incorrect Access outcomes (A_m) by the total number of outcomes in the access sample, excluding the User Blocking outcomes - i.e., dividing A_m by $(A_s+A_\ell+A_m)$. Similarly, the value of the reliability parameter Access Denial Probability is calculated by dividing the number of Access Denial outcomes A_ℓ by $(A_s+A_\ell+A_m)$. User Blocking outcomes are excluded in calculating the Access Failure probabilities to ensure the comparability of values measured under different usage conditions.

To make access measurement practical (and to make the measured values comparable), the standard defines a "maximum access time" beyond which an access attempt is declared a failure for performance assessment purposes. This "timeout" point is fixed at three times the average Access Time specified for the service; i.e., three times the delay the user "expects to see" on any given access attempt. Note that the timeout point has significance only in the assessment of performance; access attempts that extend beyond the timeout point need not be abandoned. Additional characteristics of the Access Time

distribution (e.g., variance or 95% points) will also be of interest in many applications.

The same general approach used in the access case was followed in selecting and defining performance parameters for the user information transfer and disengagement functions. A separate probability parameter was defined to express the likelihood of each possible failure outcome; and an "average elapsed time" parameter was defined, in each case, to quantify the delay associated with successful performance. Bit and Block Transfer Rate and Rate Efficiency parameters were also defined, to express system performance from the standpoint of "throughput" and resource utilization. A complete list of the primary performance parameters specified in Interim FED STD 1033 is provided at the conclusion of this section.

Although the primary parameters described above provide a relatively detailed description of data communication performance, they fall short of completeness in two respects:

1. They do not provide the kind of macroscopic, long-term view of performance users traditionally associate with the concept of availability.

2. They are user dependent, and thus cannot be applied directly in situations where it is necessary to describe the unilateral performance of the system.

A small set of additional "secondary" and "ancillary" performance parameters was included in the standard to meet these needs.

Figure 6 illustrates the approach used in defining the secondary (availability) parameters. Very briefly, the sequence of transmissions between a specified pair of users is divided into a series of consecutive performance measurement periods or samples, each corresponding to the "message" information unit discussed earlier. Values for each of five "supported" primary performance parameters are calculated over each successive message transfer function. The calculated values are compared with corresponding outage thresholds to define the "secondary outcome" of that message transfer trial as either Operational Service state or Outage state. Finally, appropriate time and probability parameters are defined to describe the resulting sequence of availability state transitions.

In assessing availability performance, the service connecting a user pair is observed only during the User Information Transfer (UIT) phase: i.e., the time, during each transaction, between Successful Access and disengagement of

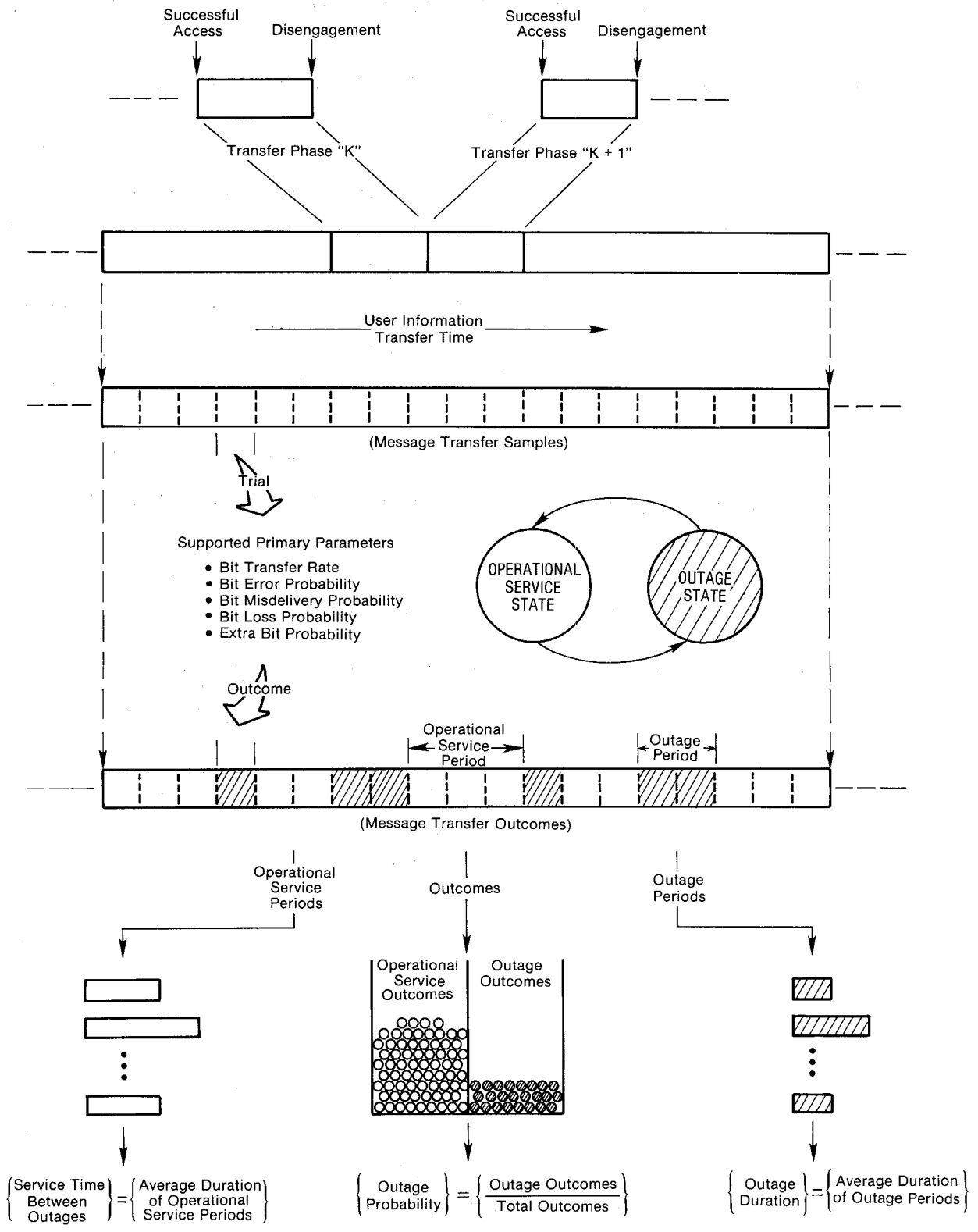


Figure 6. Secondary parameter development.

the last committed user (Seitz, 1980b). As noted earlier, there is no necessary correspondence between the amount of information transferred during an individual data communication session and the length of a "message". The "message" information unit is essentially a performance measurement sample size - a fixed number of transferred bits.

Five primary user information transfer parameters are defined as supported performance parameters: the four bit transfer failure probabilities (Bit Error Probability, Bit Misdelivery Probability, Bit Loss Probability, and Extra Bit Probability) and Bit Transfer Rate. Outage thresholds for the supported performance parameters are defined as function of the corresponding values specified for the service as follows:

1. The outage threshold for Bit Transfer Rate is defined as one-third (1/3) of the specified Bit Transfer Rate.

2. The outage thresholds for the four bit transfer failure probabilities are defined as the square root of the corresponding specified probability values. For example, a specified probability value of 10^{-6} corresponds to an outage threshold of 10^{-3}.

A service is defined to have been in the Operational Service state (during the preceding performance measurement period) whenever the measured values for all supported parameters are better than their associated outage thresholds. A service is defined to have been in the Outage state whenever the measured values for one or more supported parameters are worse than their associated outage thresholds. This classification produces, in the measurement record, a sequence of alternating Operational Service and Outage periods each having a known duration in the User Information Transfer (UIT) time. Each period comprises an integer number of messages or samples.

The secondary performance parameters provide a statistical description of this two-state random process. They are defined in the standard essentially as follows:

Service Time Between Outages - Average value of elapsed user Information Transfer time between entering and next leaving the Operational Service state.

Outage Duration - Average value of elapsed User Information Transfer time between entering and next leaving the Outage state.

Outage Probability - Ratio of total message transfer attempts resulting in the Outage state to total message transfer attempts included in a secondary parameter measurement.

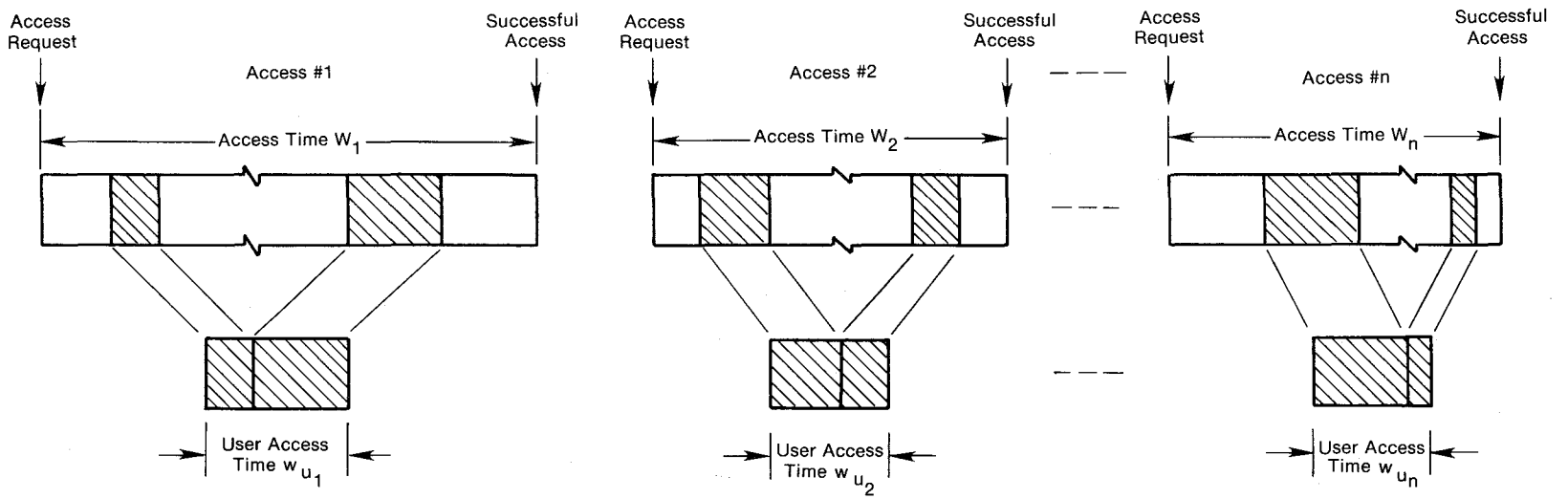
These parameters are termed "secondary" to emphasize the fact that they are defined on the basis of measured primary parameter values, rather than on the basis of direct observations of interface events.

The final FED STD 1033 parameters to be described are the ancillary parameters. The primary communication functions defined in that standard are, in general, user dependent; and there is a need, then, for a quantitative means of expressing the influence of user delay on the primary parameter values. The ancillary parameters fulfill that need.

Very briefly, the ancillary parameters are developed by dividing the total performance time for an associated primary function into alternating periods of system and user "responsibility"; and then calculating the average proportion of total performance time for which the users are "responsible". As a simple illustration, consider the access (or connection establishment) phase in a conventional voice telephone call. The total performance time for the access function is the time between the calling user's off-hook action and the called party's answer. This total performance time can be divided into alternating periods of system and user responsibility by noting, at any time, which entity must produce the next interface event. During the period between off-hook and dial tone, the system is responsible; during the period between dial tone and positioning of the first dialed digit, the user is responsible; and so on. The ancillary parameter User Access Time fraction expresses the average proportion of total Access Time that is attributable to the user activities.

Figure 7 illustrates the approach used in defining the ancillary parameters in more detail, again using the primary function of access as an example. The figure depicts a series of successful access attempts, each having a total access time w and a total <u>user</u> access time w_u. The latter quantity represents the total access time attributable to the users on each particular trial. The ancillary parameter User Access Time Fraction is calculated by adding the user access time values over a suitable number of successful access attempts, and then dividing by the corresponding sum of the total access times. Only Successful Access outcomes are considered in estimating User Access Time Fraction in order to avoid biasing the average with unrepresentative values.

A similar approach is used in defining ancillary performance parameters for the block transfer, message transfer, and disengagement functions. No separate ancillary parameter is defined for the bit transfer function, since



$$\left\{\begin{array}{c}\text{User Access}\\ \text{Time Fraction}\end{array}\right\} = \left\{\frac{\text{Average User Access Time}}{\text{Average Total Access Time}}\right\}$$

$$= \left\{\frac{w_{u_1} + w_{u_2} + \ldots + w_{u_n}}{W_1 + W_2 + \ldots + W_n}\right\}$$

Figure 7. Ancillary parameter development - access example.

its values can be inferred from the corresponding block transfer parameter. The standard thus defines a total of four ancillary performance parameters: User Access Time Fraction, User Block Transfer Time Fraction, User Message Transfer Time Fraction, and User Disengagement Time Fraction.

The ancillary parameters have two specific uses:

1. They enable calculation of "user-independent" values for the associated efficiency parameters - i.e., the values that would be observed if all user delays were zero.

2. They provide a basis for identifying the entity "responsible" for timeout failures - the user or the system.

Each of these uses is described more fully in Seitz (1980a).

Figure 8 summarizes the performance parameters which were selected for inclusion in Interim Federal Standard 1033. Twenty-six parameters were specified, including 19 primary parameters, 3 secondary parameters, and 4 ancillary parameters. A narrative/symbolic definition for each parameter is provided in Appendix A.

2.3 Summary of Proposed American National Standard X3.102

Proposed American National Standard X3.102 is similar, in both overall approach and technical content, to the interim 1033 standard. From a measurement standpoint, the most significant differences between the two standards are in parameter selection and definition. This subsection briefly summarizes these differences, proceeding by function and category in the manner outlined above: i.e., access parameters, user information transfer parameters, disengagement parameters, secondary or availability parameters, and ancillary parameters.

The primary access parameters defined in X3.102 differ from their Interim FED STD 1033 counterparts in that the Access Denial outcome is separated into two distinct cases of system nonperformance: Access Denial and Access Outage. As defined in X3.102, Access Denial occurs when the system responds to the user's access request in some manner, but either (a) issues a blocking (e.g., circuit busy) signal during the access period, thereby preventing Successful Access, or (b) delays excessively in responding to user inputs during the access period, with the result that user information transfer is not initiated before access timeout. Access Outage occurs when the system fails to issue any active interface signal during the access attempt: i.e., the system appears to the user to be "dead".

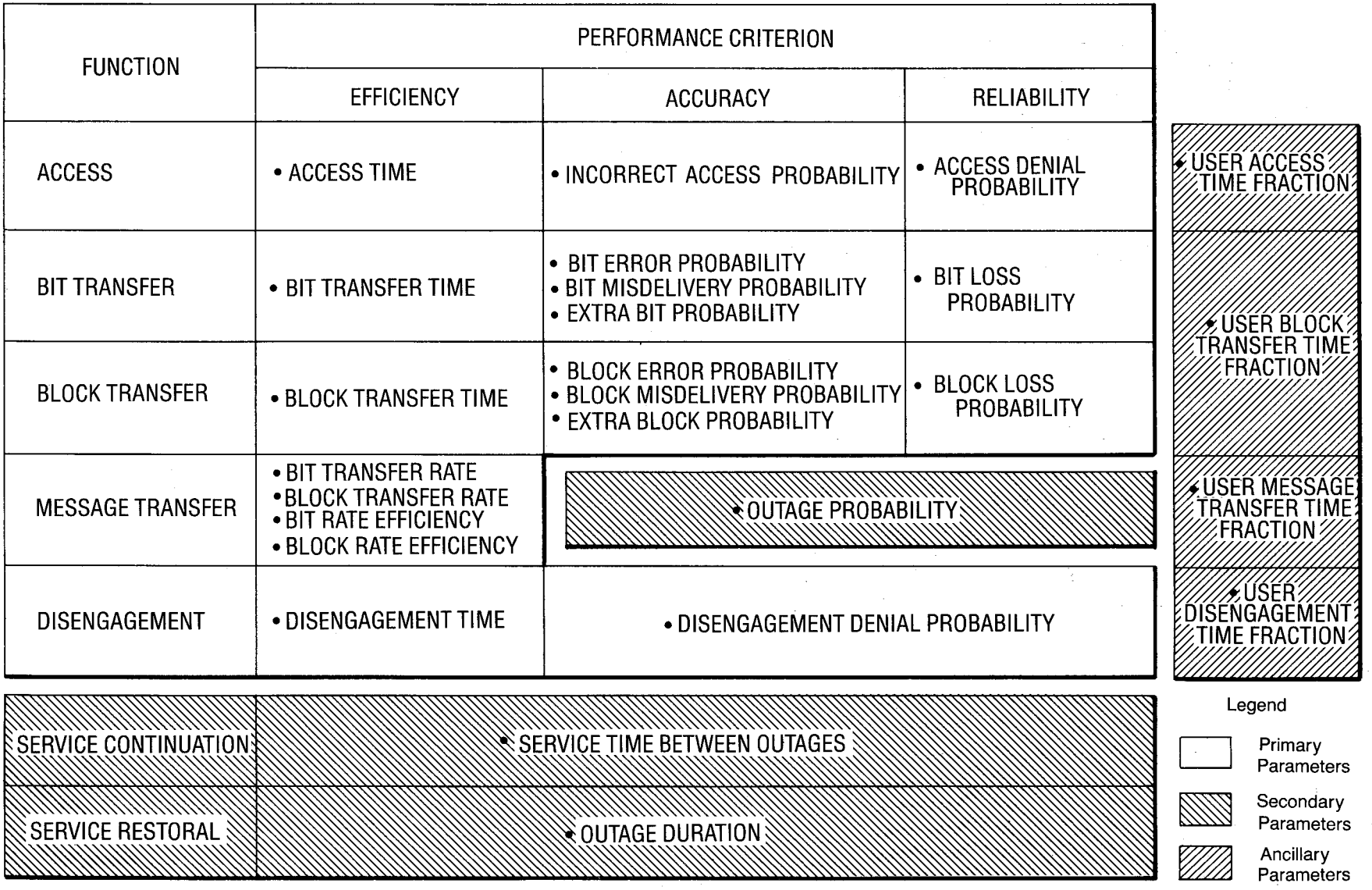


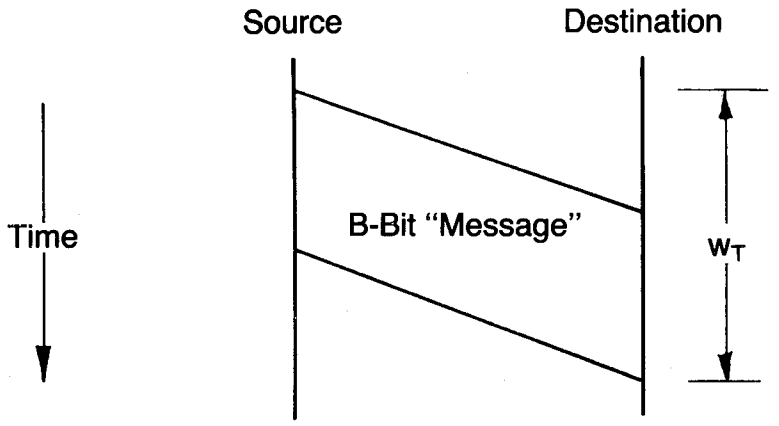
Figure 8. Interim FED STD 1033 parameters.

The rationale for distinguishing these two failures is that the appropriate user responses are often different. In the case of Access Denial (blocking), the user will normally re-try the call. In the case of Access Outage, his best response may be to seek maintenance action. The X3.102 standard defines a separate probability parameter for each of these outcomes.

The ANSI review and adaptation of Interim FED STD 1033 resulted in the omission of four user information transfer parameters: Bit Transfer Time, Block Transfer Rate, and the Bit and Block Rate Efficiencies. Bit Transfer Time was omitted on the basis that (1) it is often difficult to measure, and (2) it is identical to Block Transfer Time in many buffered systems. Block Transfer Rate was omitted on the basis that it is somewhat redundant with, and less useful in performance comparison than, Bit Transfer Rate. (The X3.102 standard renames the latter parameter "User Information Bit Transfer Rate" to emphasize the fact that overhead bits are excluded from its numerator). The Bit and Block Rate Efficiencies were omitted on the basis that they are of primary concern to communication suppliers rather than users. Bit Rate Efficiency can be calculated from User Information Bit Transfer Rate by simply dividing the latter by the channel signalling rate.

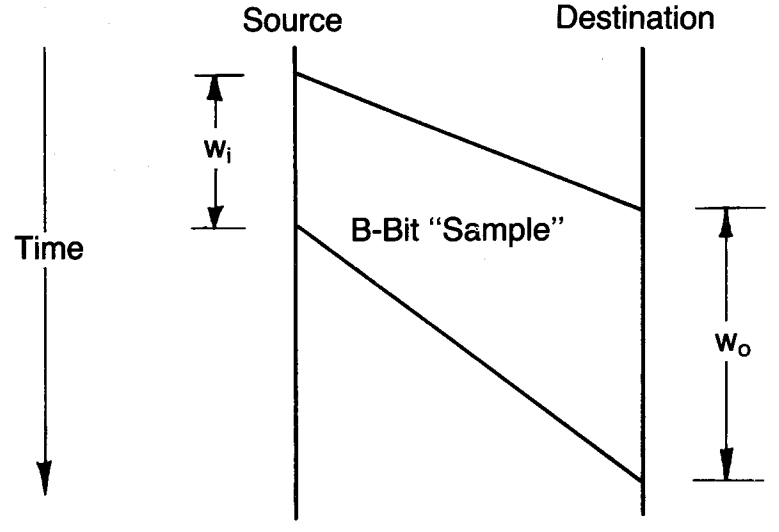
The ANSI X3S35 committee also changed the rate parameter definition with respect to the time interval used in the denominator. Figure 9 illustrates that change. In Interim FED STD 1033, the time in the denominator of a rate parameter includes all time between the start of input of a "message" at the source and the end of output of that message at the destination (W_T). In X3.102, the time used is the <u>larger</u> of the input time (W_i) or the output time (W_o) for a corresponding information unit (called a "sample"). The latter definition has the advantage that it excludes propagation time, and thereby provides a more realistic measure of system throughput capacity.

The disengagement parameter definitions in Interim FED STD 1033 were not significantly changed in X3.102, but their application was generalized in one important respect. As defined in Interim FED STD 1033, the disengagement parameters represent the average performance provided to <u>all</u> users participating in a data communication session. In X3.102, these parameters may also be defined and evaluated separately for <u>each</u> participating user. It was believed that the latter option would provide more realistic values in cases where the disengagement functions differ significantly at the separate user interfaces.



$$\text{Bit Transfer Rate} = \left[\frac{B}{w_T} \right]$$

a. FED STD 1033 Rate Parameter Definition



$$\left\{ \begin{array}{c} \text{User Information} \\ \text{Bit Transfer} \\ \text{Rate} \end{array} \right\} = \frac{B}{\text{Max}\,[w_i \text{ or } w_o]}$$

b. ANSI X3.102 Rate Parameter Definition

Figure 9. Comparison of FED STD 1033 and ANSI X3.102 rate parameter definitions.

The ANSI Task Group also made several changes in the Interim FED STD 1033 secondary parameter definitions. The group concluded, first, that Service Time Between Outages and Outage Duration should be omitted; and second, that two specific changes should be made in the definition of Outage Probability (renamed "Transfer Denial Probability"). These changes were (1) omission of Bit Misdelivery Probability as a supported performance parameter; and (2) use of the fourth root instead of the square root in calculating the probability parameter outage thresholds. These changes were intended to simplify the measurement of availability, particularly with respect to the observation time required to obtain representative parameter values. The concept of a separate "secondary" performance parameter category was dropped in X3.102, since that category would have contained only one parameter.

The ancillary performance parameters defined in Interim FED STD 1033 and X3.102 are the same, with the exception of (1) renaming, and (2) a change from "message transfer time" to "input/output time" as the interval to be observed in assessing user influence on throughput (Figure 9).

Figure 10 summarizes the performance parameters selected for inclusion in ANSI X3.102. A total of 21 parameters were defined, including 17 primary parameters and 4 ancillary parameters. A narrative definition for each parameter is provided in Appendix A.

Of the 21 parameters specified in ANSI X3.102, 17 are identical to their Interim FED STD 1033 counterparts or nearly so; 3 (Access Denial Probability, User Information Bit Transfer Rate, and Transfer Denial Probability) are significantly redefined; and 1 (Access Outage Probability) is new. Six parameters originally specified in Interim FED STD 1033 were omitted from the ANSI version. One objective of the ARPA network measurement project was to assess the merit of these changes in parameter choice and definition from a measurement perspective. Results of this assessment are presented in Section 6.

2.4 Summary of Proposed Federal Standard 1043

This section summarizes the overall approach and technical content of proposed Federal Standard 1043. As noted earlier, that standard defines uniform methods of obtaining values for the standard performance parameters specified in Interim FED STD 1033. A major purpose of the ARPA network measurements project was to validate and demonstrate proposed Federal Standard 1043 by implementing it in a practical measurement situation.

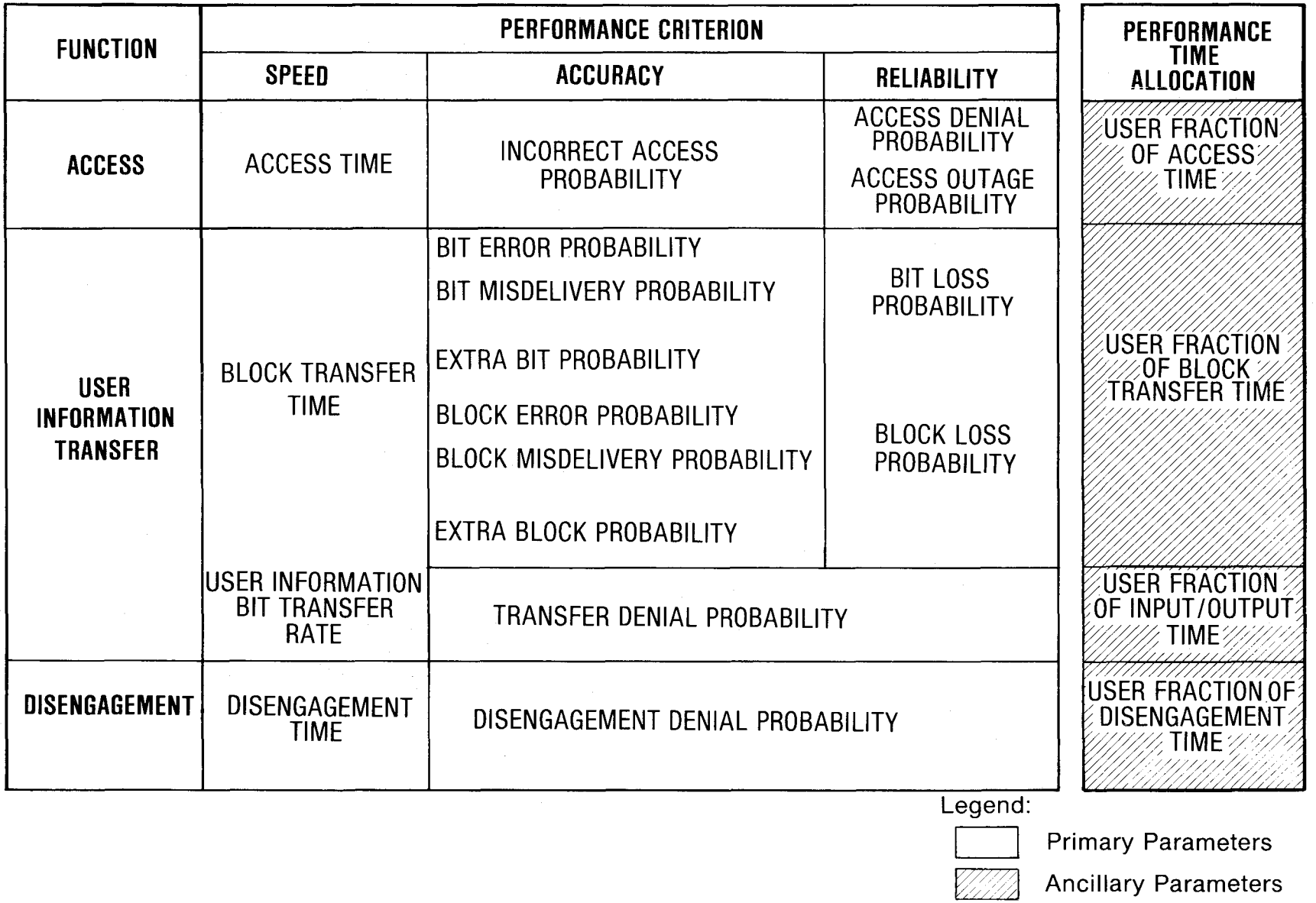


Figure 10. Summary of ANSI X3.102 performance parameters.

Figure 11 summarizes the overall approach used in defining the proposed FED STD 1043 performance measurement methods. The standard is written as a specification for a conceptual data communication performance measurement system consisting of four major elements:

1. Data Extraction. This system element observes overhead and user information signals transferred across the user/system interfaces in real time; determines the performance significance and time of occurrence of each interface signal; and outputs this performance information in the form of a chronological reference event history. The events observed at each user/system interface provide the basis for performance parameter calculations.

2. Data Files. This system element consists of a set of standard ASCII character files whose format is independent of the system under test. These files record the real-time event histories produced by the data extraction function in a standard form for later (off-line) reduction and analysis. Separate files are maintained for user and overhead information, and for each direction of user information flow.

3. Performance Assessment. This system element consists of a standard FORTRAN program which merges, correlates, and analyzes the ASCII files described above to produce a set of measured FED STD 1033 parameter values. Primary elements of the Performance Assessment program are the access, transfer, and disengagement subroutines, which calculate values for the 19 "primary" parameters defined in FED STD 1033. These subroutines are supported by programs which calculate "secondary" and "ancillary" parameter values and perform various utility functions. The entire Performance Assessment program is written in ANSI (1966) standard FORTRAN to ensure its transportability.

4. Statistical Design and Analysis. This system element is a set of procedures which enable users to relate measurement precision objectives with associated sample sizes. Those statistical criteria are used to control the data extraction and performance assessment elements of the FED STD 1043 performance measurement system.

These four measurement system elements differ substantially in the degree to which their implementation can be standardized, and their specification in the standard reflects those differences. The extraction and statistical elements (1 and 4) are interface and application dependent, and they are therefore specified in relatively general functional terms. These functional specifications are intended to guide standards users in designing data extraction subsystems and developing statistical design/analysis criteria appropriate to their particular needs.

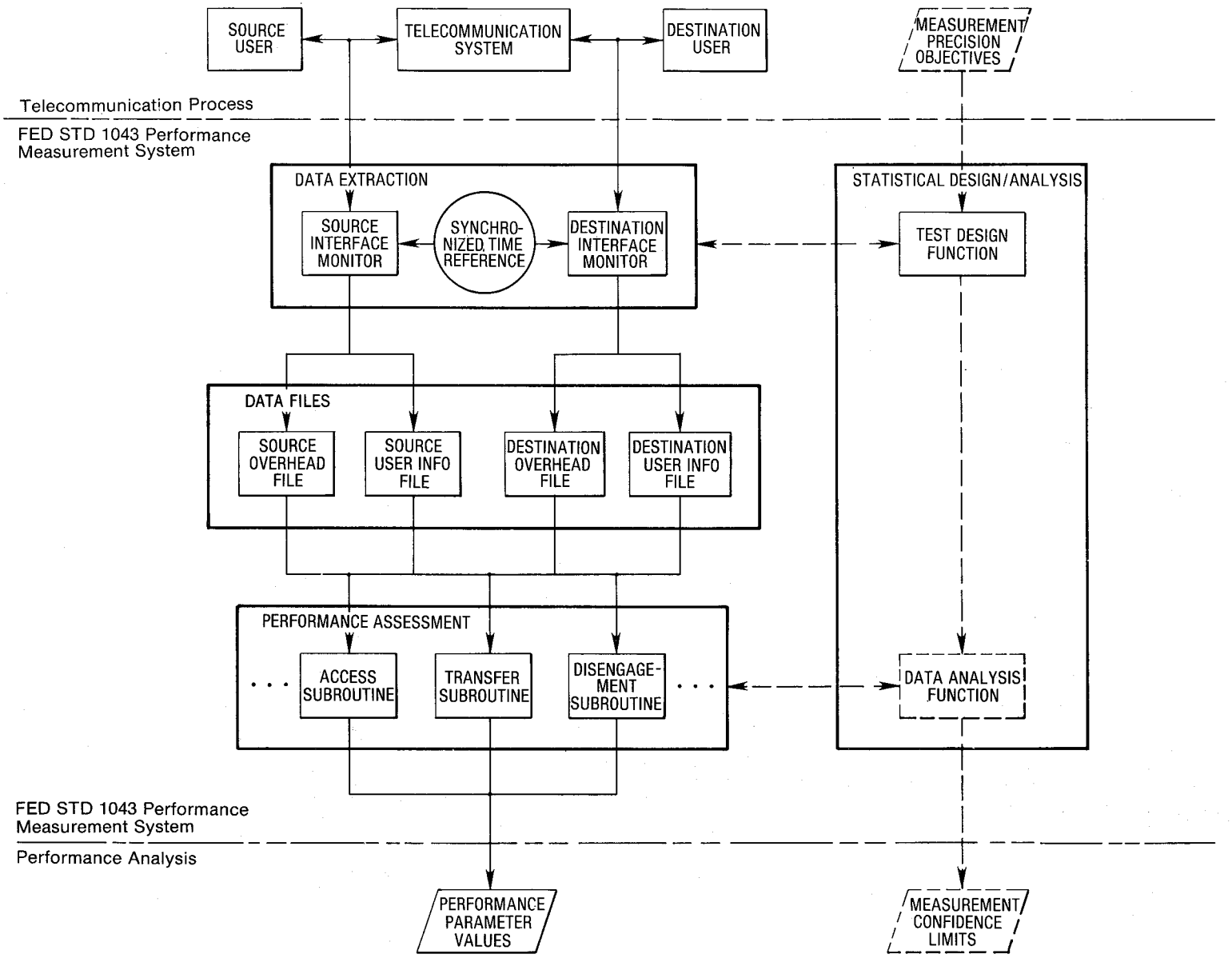


Figure 11. Performance measurement overview.

The user-oriented approach implemented in Interim FED STD 1033 has made the data recording and performance assessment elements (2 and 3) both interface and application independent, and the standard exploits that property by completely standardizing their implementation in a set of detailed subsystem designs. Availability of these standard designs (and the associated software products) will relieve standards users of the requirement to develop data file structures and data reduction software for each measurement application, and will ensure the comparability of measurement results obtained by different user organizations.

2.4.1 Data Extraction

The data extraction section of proposed Federal Standard 1043 defines the general functional requirements for a typical FED STD 1043 Interface Monitor. Any such monitor must perform three major functions:

1. Input. Detect and properly interpret all signals transferred across the monitored user/system interface. Record and time-tag all performance significant events for interface monitor processing.

2. Processing. Determine the FED STD 1033 performance significance of each observed interface event. Associate each performance-significant interface event with a corresponding system-independent reference event.

3. Output. Create a machine-independent ASCII character record defining the nature and time of occurrence of each reference event. Store the reference event records in the appropriate (overhead or user) information files.

The basic input to a FED STD 1043 Interface Monitor is a sequence of interface events occurring at the physical or functional boundary between an end user and a data communication system. As discussed in Section 2.2, Interim FED STD 1033 defines the end user of a data communication system or service to be either a human terminal operator, an unattended device medium (such as punched cards), or a computer application program. In some cases, more than one type of entity supports the overall user function: for example, a terminal operator providing control inputs and a punched paper tape providing information storage at the same data communication station. In such cases, each relevant interface must normally be monitored.

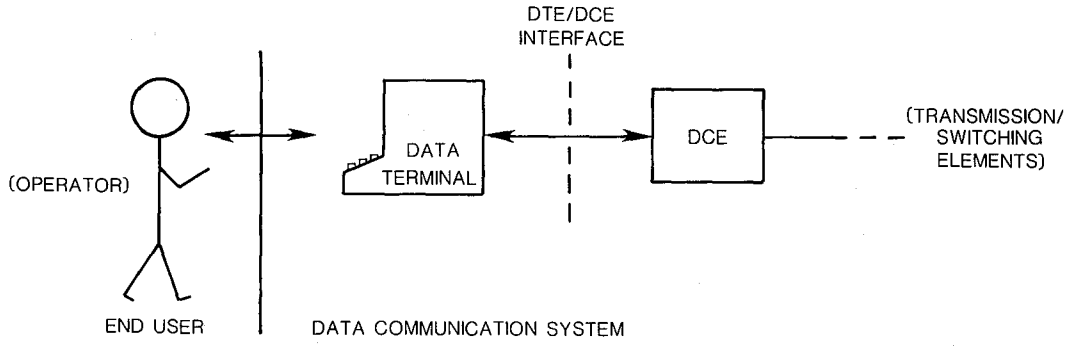
The FED STD 1043 draft defines three general types of user/system interfaces, each corresponding to a particular type of end user. When the end user is a human terminal operator, the user/system interface is defined to be

the physical interface between the operator and the data terminal (Figure 12a). When the end user is a device medium, the user/system interface is defined to be the physical interface between the medium and the data terminal (Figure 12b). When the end user is an application program, the user/system interface is defined to be the functional interface between that program and the local host computer operating system, telecommunication access method, or equivalent. One such interface is illustrated in Figure 12c, where the Application Layer represents the local telecommunication access method.

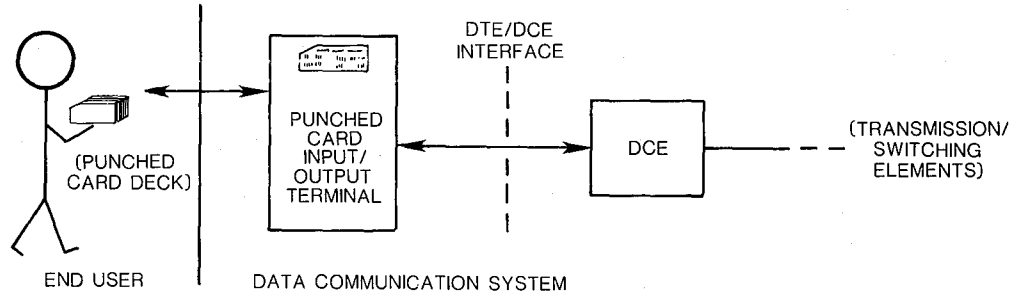
Any discrete transfer of digital information across a user/system interface is called an interface event. Such events can occur in a variety of ways. Typical events at the operator/terminal interface are manual keystrokes and the printing or displaying of received characters. Typical events at the medium/terminal interface are the reading and punching of punched cards. Typical events at the application program/access method interface are the issuance of operating system calls, co-routine calls, and interprocess communications, and the setting and clearing of flags.

The end user/communication system interfaces defined above will often not coincide with the data communication carrier/customer interfaces. In such cases, it may be desirable to observe the interface event sequence, and determine certain parameter values, at the latter (subsystem) interfaces as well. Figure 13 illustrates two possible subsystem applications. In the first, Figure 13a, the subsystem interface is placed at the DTE/DCE (Data Terminal Equipment/Data Circuit-Terminating Equipment) physical interface; and the operator and terminal are regarded as an "aggregate user" of the information transfer channel. In the second, Figure 13b, the subsystem interface is placed between the Session and Transport layers of a typical Open Systems Interconnection protocol hierarchy; and the Application Process and Application, Presentation, and Session layers are regarded as an "aggregated user" of the Transport Subsystem. Subsystem applications can be useful in allocating end-to-end performance objectives to purchasable components and services, and conversely, in determining the impact of subsystem choices on end-to-end performance.

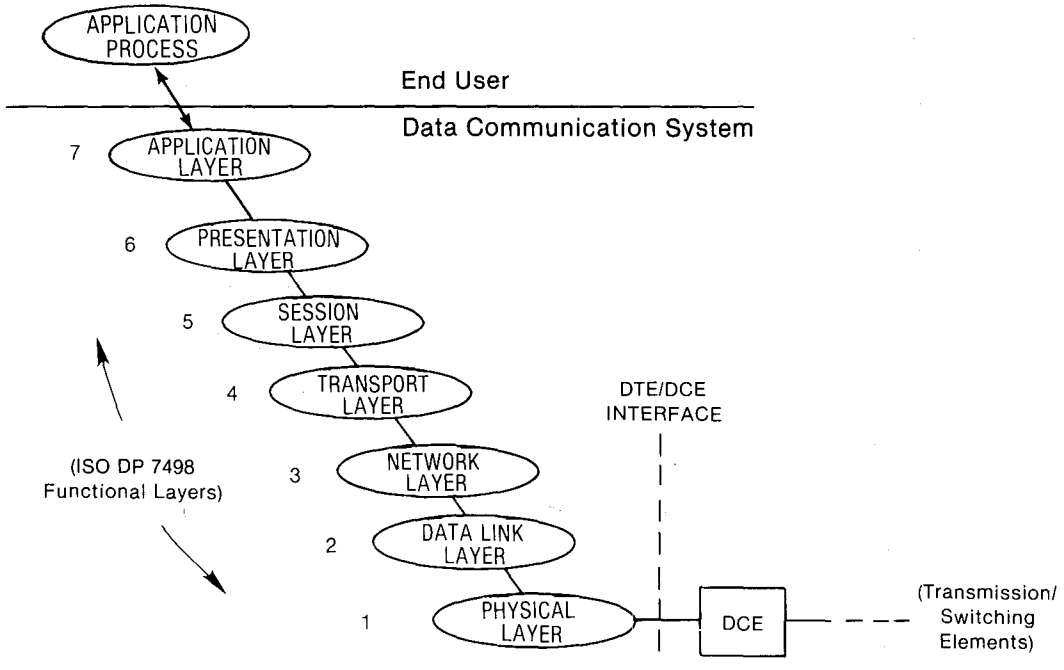
Interface events are generated by the communicating user and system entities. To be in accordance with proposed FED STD 1043, an Interface Monitor must detect all interface events generated at the monitored interface during a performance measurement period, and must interpret each interface



a. Operator/Terminal Interface

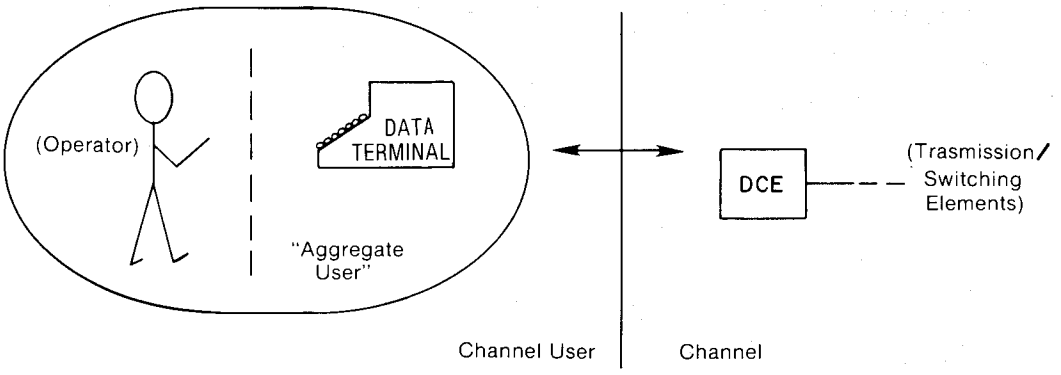


b. Medium/Terminal Interface

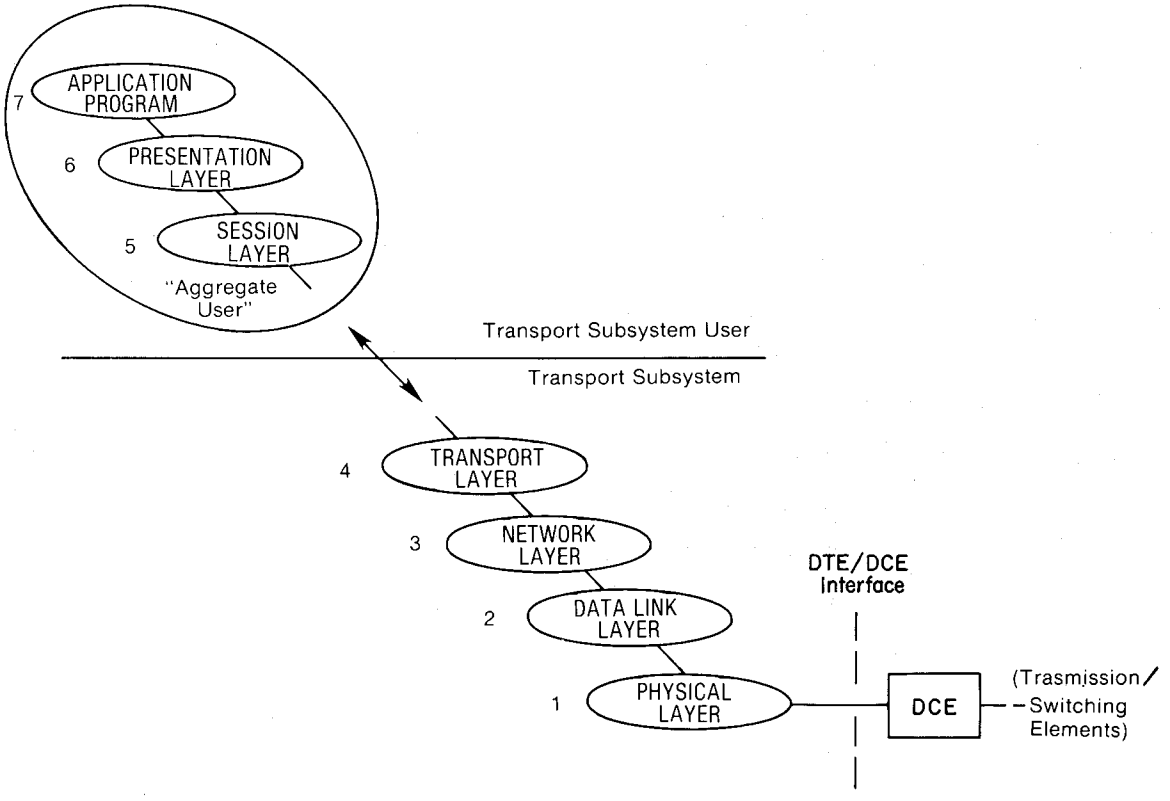


c. Application Program Interface

Figure 12. End user/data communication system interfaces.



a. Operator and Terminal as an "Aggregate User" of the Information Transfer Channel.



b. Application, Presentation and Session Layers as an "Aggregate User" of the Transport Subsystem.

Figure 13. Aggregate user concept.

event in accordance with its significance to the communicating entities. The Interface Monitor must accomplish these functions without significantly influencing the timing or quality of the communicated signals.

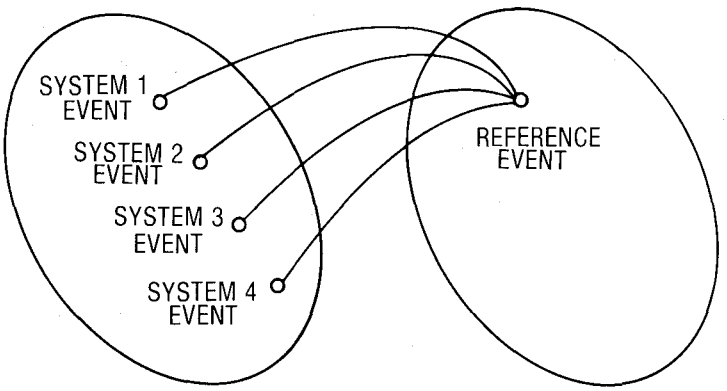
The events observed at a particular user/system interface cannot be used directly in defining user-oriented performance parameters because they are system dependent - i.e., they vary from one data communication system to another. As noted earlier, these user-oriented standards overcome the problem of system dependence by defining the parameters not in terms of particular system-dependent interface events, but in terms of more general, system-independent reference events.

An example will help to clarify the relationship between system-dependent interface events and the associated reference events. A user's action in lifting a telephone handset off-hook transfers one bit of information (the new hookswitch position) from the user to the system. That transfer is a system-dependent interface event which notifies the system of a user need for service in circuit-switched systems (e.g., the public switched network). Completely different interface events may convey the same information in message- or packet-switched systems (e.g., issuance of the Connect system call in the ARPA network).

All interface events which communicate a user's need for data communication service (or otherwise initiate an information transfer transaction) are represented in proposed FED STD 1043 by the single reference event "access request". This reference event is used to define the beginning of the access function, and to start the counting of access time. Defining the access parameters in terms of this and similar reference events makes the parameters system independent, and makes their values comparable between services with different user interface protocols.

As a second example, consider the reference event "start of block transfer" (Figure 14). The nature of the user information unit called a "block", and the physical events associated with its movement across a source user/data communication system interface, will vary from one system to another. Nevertheless, some specific information unit can be identified as a FED STD 1043 block, and its transfer across the user/system interface can be identified, in any system; and the corresponding block transfer parameters can therefore always be defined.

Sixteen reference events are used in measuring the 26 Interim FED STD 1033 performance parameters: 10 "primary" reference events and 6 "ancillary"



Example — Reference Event "Start of Block Transfer":

System	User Interface Device	System-Specific Event
1	Character-asynchronous TTY	Typing a letter or figure
2	Buffered CRT terminal	Typing CR (end of line)
3	Host computer access method	Issuing WRITE system call
4	Punched card reader	Reading of card image

Figure 14. Example-reference event "Start of Block Transfer."

reference events. The primary reference events are listed in Figure 15. These events record the basic performance information needed to calculate values for the "primary" and "secondary" Interim FED STD 1033 parameters. Two such events are the "access request" and "start of block transfer" events just discussed. The ancillary reference events record changes in the state of affairs at a monitored interface with respect to which communicating entity, the user or the system, is "responsible" for creating the next interface event. Three distinct responsibility states may exist at an interface during a data communication session: User Responsible, System Responsible, and Responsibility Undefined.[7] The ancillary reference events are the six possible transitions between these three states.

The ancillary reference events are used in defining the ancillary performance parameters, which describe the influence of user delays on the primary efficiency (or speed) parameters. A record of these events is also needed to identify the entity responsible for "timeout" performance failures. Ancillary reference events occur asynchronously at the two user interfaces, and are recorded independently by the associated Interface Monitors.

Interim Federal Standard 1033 defines a user-oriented "transaction state model" which consolidates the various overhead transfer and ancillary reference events and serves as a conceptual aid in understanding their relationships. In a measurement context, that model provides a simple, concise method of recording individual occurrences of those same events, and underlies the overhead information file structure defined in FED STD 1043. The following paragraphs briefly describe the transaction state model and define the relationship between overhead and ancillary reference events and the corresponding model events.

The transaction state model represents the digital communication process at a given user/system interface as a sequence of discrete interactions between two participating entities: the end user receiving service and a conceptual "half-system" which represents the portion of the end-to-end system that interacts with that user. Each communicating entity is represented as simple finite-state machine that can be characterized, at any given time, by a distinct Transaction state. An entity's Transaction state fully describes the

---

[7]The latter state may exist, for example, when the user and the system elements communicating across a particular interface are both waiting for an event at the other interface. Such a state exists at the calling interface during ringing in the public switched telephone network.

FUNCTION	REFERENCE EVENT	SYSTEM IMPACT	PERFORMANCE SIGNIFICANCE	EXAMPLES
ACCESS	1. ACCESS REQUEST	REQUESTS INITIATION OF AN INFORMATION TRANSFER TRANSACTION AND COMMITS THE ORIGINATING USER TO PARTICIPATE.	BEGINS ACCESS FUNCTION. STARTS THE COUNTING OF ACCESS TIME.	ORIGINATING USER CONNECT REQUEST IN THE ARPANET.
	2. NONORIGINATING USER COMMITMENT	IN A CIRCUIT-ORIENTED TRANSACTION INDICATES NONORIGINATING (CALLED) USER WILLINGNESS TO PARTICIPATE	ELIMINATES INCORRECT ACCESS AS A POSSIBLE ACCESS OUTCOME.	NONORIGINATING USER CONNECT REQUEST IN THE ARPANET.
	3. SYSTEM BLOCKING SIGNAL	NOTIFIES ORIGINATING USER THAT THE SYSTEM CANNOT SUPPORT A REQUESTED INFORMATION TRANSFER TRANSACTION.	IDENTIFIES ACCESS ATTEMPT OUTCOME AS ACCESS DENIAL.	DESTINATION HOST DEAD (DESTDEAD) CONTROL MESSAGE IN THE ARPANET.
	4. USER BLOCKING SIGNAL	NOTIFIES SYSTEM THAT THE ISSUING USER CANNOT SUPPORT A REQUESTED INFORMATION TRANSFER TRANSACTION	IDENTIFIES ACCESS ATTEMPT OUTCOME AS USER BLOCKING (EXCLUDED FROM SYSTEM PERFORMANCE MEASUREMENT).	NONORIGINATING USER CLOSE REQUEST (DURING ACCESS) IN THE ARPANET.
USER INFORMATION TRANSFER	5. START OF FIRST BLOCK INPUT TO SYSTEM	MOVES ONE OR MORE USER INFORMATION BITS FROM SOURCE USER TO SYSTEM.	COMPLETES ACCESS FUNCTION AND BEGINS USER INFORMATION TRANSFER STOPS THE COUNTING OF ACCESS TIME.	TYPING OF FIRST USER INFORMATION CHARACTER AT A BUFFERED CRT TERMINAL.
	6. START OF BLOCK TRANSFER	AUTHORIZES THE SYSTEM TO TRANSMIT A GIVEN USER INFORMATION BLOCK.	BEGINS BLOCK TRANSFER FUNCTION. STARTS THE COUNTING OF BLOCK TRANSFER TIME.	TYPING OF ANY USER INFORMATION CHARACTER AT AN UNBUFFERED SOURCE TERMINAL. TYPING OF CARRIAGE RETURN AT A BUFFERED SOURCE TERMINAL.
	7. START OF BLOCK OUTPUT TO DESTINATION USER	MOVES ONE OR MORE USER INFORMATION BITS FROM SYSTEM TO DESTINATION USER.	STOPS COUNTING OF BIT TRANSFER TIME FOR THE FIRST USER INFORMATION BIT IN A BLOCK.	PRINTING OR DISPLAYING OF THE FIRST CHARACTER IN A SOURCE USER INFORMATION BLOCK AT A DESTINATION TERMINAL.
	8. END OF BLOCK TRANSFER	TRANSFERS A GIVEN USER INFORMATION BLOCK TO THE DESTINATION USER, WITH APPROPRIATE NOTIFICATION TO THAT USER WHERE REQUIRED.	ENDS BLOCK TRANSFER FUNCTION. STOPS THE COUNTING OF BLOCK TRANSFER TIME.	PRINTING OR DISPLAY OF A COMPLETE SOURCE USER INFORMATION BLOCK AT THE DESTINATION TERMINAL.
DISENGAGEMENT	9. DISENGAGEMENT REQUEST	REQUESTS TERMINATION OF A USER'S PARTICIPATION IN AN INFORMATION TRANSFER TRANSACTION.	BEGINS DISENGAGEMENT FUNCTION. STARTS THE COUNTING OF DISENGAGEMENT TIME.	CLOSE REQUEST (AFTER ACCESS) IN THE ARPANET.
	10. DISENGAGEMENT CONFIRMATION	CONFIRMS TERMINATION OF A USER'S PARTICIPATION IN AN INFORMATION TRANSFER TRANSACTION.	COMPLETES DISENGAGEMENT FUNCTION. STOPS THE COUNTING OF DISENGAGEMENT TIME.	CLOSE COMPLETION SIGNAL IN THE ARPANET.

Figure 15. Primary reference events.

entity's current situation with respect to its participation in a data communication session. The overhead transfer and ancillary reference events are represented as discrete changes in the Transaction states of one or both monitored entities.

Figure 16a is a state diagram which represents the possible Transaction states of each of the two model entities (the end user and the associated half-system). Relative to any given data communication session, each entity is in one of three primary Transaction states at any time. These three states are defined as follows.

1. Idle state. The entity is not involved in the given session. The entity can be involved in another session, or can be uninvolved in any session.

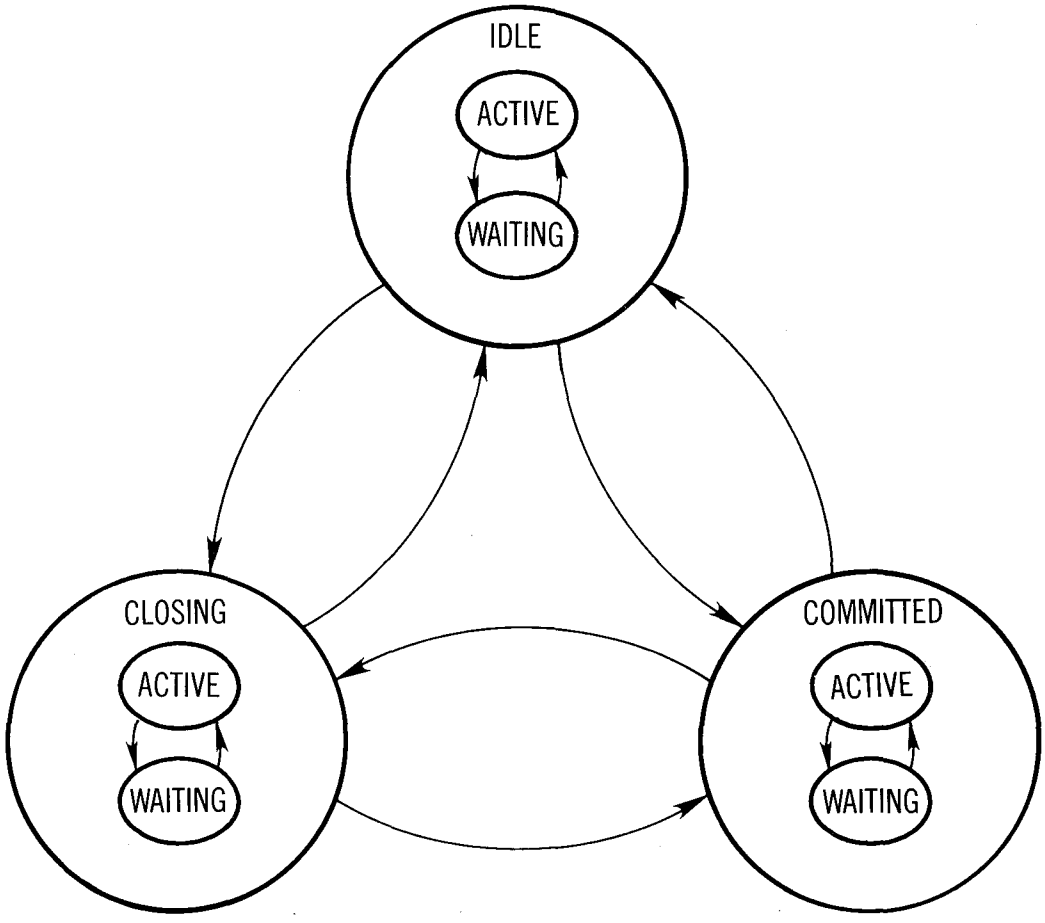
2. Committed state. The entity is involved in the given session, with intent to transfer further user information.

3. Closing state. The entity is involved in the given session, with intent to terminate involvement without transferring further user information.

Within each primary state are two "ancillary states": the Active state and the Waiting state. In general, an entity can enter either ancillary state on entry to the corresponding primary state; and can change ancillary states any number of times within a given primary state. The ancillary substates have slightly different meanings depending on the associated primary states.

Within the Committed and Closing states, the terms Active and Waiting describe an entity's "responsibility" relative to the ongoing sequence of interactions at the monitored user interface. When an entity is responsible for producing the next interface event at the monitored interface, the entity is defined to be in the Active state; otherwise, the entity is defined to be in the Waiting state. The User Responsible state exists at an interface when the user is Active and the associated half-system is Waiting. The System Responsible state exists when the user is Waiting and the associated half-system is Active; the Responsibility Undefined state exists when both entities at an interface are Waiting. All substate transitions within the Committed and Closing states occur as a result of overhead or user information transfers, but not all such transfers produce responsibility state changes.

Many telecommunication services are available to the user only during certain designated time periods, termed "service time intervals." Users may restrict their participation in telecommunication activities in a similar way.



a. Entity Transaction State Diagram

COMPOSITE STATE	CLOSING STATE	COMMITTED STATE	RESPONSIBILITY STATE
IDLE/ACTIVE	0	0	1
IDLE/WAITING	0	0	0
COMMITTED/ACTIVE	0	1	1
COMMITTED/WAITING	0	1	0
CLOSING/ACTIVE	1	0	1
CLOSING/WAITING	1	0	0

b. Binary Representation of Transaction States

Figure 16. Transaction state diagram and binary representation.

The Active and Waiting substates within the Idle state represent the status of an entity relative to such service restrictions. When an entity is within a designated service time interval, but is not involved in a data communication session, the entity is defined to be in the Idle/Active state. When an entity is not within an established service time interval, the entity is defined to be in the Idle/Waiting state. Transitions involving these states indicate the beginning and end of planned service time intervals. Note that it is possible for two half-systems within the same telecommunication system to be in different Idle substates; an example is a world-wide message switching system that provides service to subscribers only during local business hours.

Figure 16b represents the six Transaction states defined above in three-bit binary form. The octal equivalents of these binary state variables are used in the ASCII character records in the overhead information file, as described below.

Successful measurement of the Interim FED STD 1033 parameters depends on a proper translation of the system-specific events observed at a monitored user/system interface into corresponding system-independent reference events. The Interface Monitor processing function performs that translation. Details of the translation process are dependent on the user/system interface being monitored, but the standard provides detailed guidelines to assist users in accomplishing that task.

Measurement of the Interim FED STD 1033 speed parameters also requires comparison of event times at the two distant user interfaces. The two Interface Monitors must therefore have access either to a single common time source or to separate, but synchronized, time sources. Federal Standard 1043 describes three practical ways of obtaining the necessary time synchronization: HF WWV broadcasts, NBS telephone time-of-day service, and the Direct Satellite Time Service of the National Oceanic and Atmospheric Administration's Geostationary Operational Environmental Satellite (Kamas and Howe, 1979; NBS, 1978).

The output of a FED STD 1043 Interface Monitor is a set of formatted ASCII character records defining the sequence of reference events observed at the monitored interface during a performance measurement period. The nature of these ASCII character records is described in the following subsection.

2.4.2 Data Files

The Data Files section of proposed Federal Standard 1043 specifies standard formats for recording output from the Data Extraction element of the FED STD 1043 performance measurement system. These formats are independent of measurement application.

A given performance measurement batch consists of four data files: a source overhead information file and source user information file generated by the source Interface Monitor, and a destination overhead information file and destination user information file generated by the destination Interface Monitor. Each overhead information file contains a chronological record of overhead and ancillary reference events corresponding to the sequence of interface events observed at the monitored interface. Each user information file contains an ASCII character representation of the transferred user information.

Both the overhead and user information files are sequential files of formatted ASCII character records. Sequential files are used because that is the only file structure supported by the 1966 ANSI standard FORTRAN. (Conversion to the more recent 1977 ANSI standard FORTRAN will be accomplished when use of that standard becomes widespread.) The choice of ASCII character records instead of binary (unformatted) records permits file transportability for a larger class of computers.

Each record in an overhead information file corresponds to a unique reference event. The record contains the time of the event, and the octal representation of the three-bit transaction state code of each model entity (source user, source half-system, destination half-system, destination user) immediately after the event, as indicated in Figure 16b.

Figure 17 illustrates how binary transmitted and received data are represented in a corresponding ASCII character file. The sequence of user information bits corresponding to a block is divided into a sequence of strings, each composed of 15 bits. The last string in a block is completed, if necessary, with binary zero fill following the final user information bit. The 15 bits that form a string are treated as the binary representation of a decimal integer, where the bit of lowest index within the string is most significant. Note that the resulting decimal integer lies in the (inclusive) range 0-32,767. The user information in a block is thus mapped into a sequence of decimal integers. Each such integer is stored in a user information file as an ASCII character string of not more than five digits.

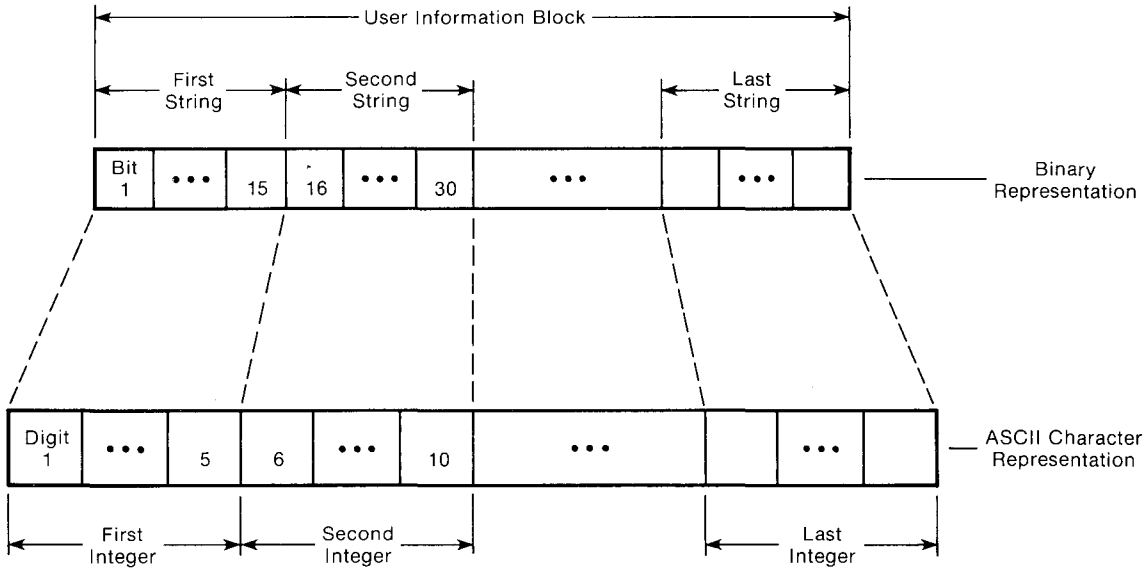


Figure 17. Binary and ASCII character representations of user information blocks.

These are right-justified in a five-character field with blank or zero (character) fill on the left. The data in each field are read and converted by the performance assessment program using an I5 FORTRAN format specification. After conversion, the low order (least significant) 15 bits of the corresponding storage location in memory form an exact copy of the original user information bit string.

A somewhat different technique is used in representing overhead information transfer events. As discussed earlier, each user and system entity is modeled as a finite state machine whose participation in an information transfer transaction can be represented, at any time, by a discrete transaction state. Each possible transaction state for an entity is represented by a corresponding octal digit, which is encoded in a single ASCII character. All overhead transfer events are then recorded as changes in the transaction states of one or more entities.

2.4.3 Performance Assessment

The performance assessment section of proposed Federal Standard 1043 specifies a computer program that processes 4 input ASCII character files to produce a set of 26 Interim FED STD 1033 performance parameter values. The Performance Assessment program and its input/output files are independent of measurement application, which allows them to be completely standardized. To enhance its transportability, the program was written in 1966 ANSI standard FORTRAN.

As documented in proposed FED STD 1043, the Performance Assessment program consists of a main driver (ASSESS)[8] and a set of subroutines that perform the various data processing procedures required to produce calculated parameter values. These program elements, and the input/output files they reference, are shown in Figure 18. In this diagram, each subroutine at a given level is connected by a heavy line to a program element at the next higher level. The intended relation is that the higher level program element calls the subroutine. The left-to-right arrangement of subroutines at a given level indicates the order in which they are called by a particular program element. A thin line from an I/O file to a subroutine implies that the

---

[8]To assist the reader in identifying program and file names appearing in the body of the text, such names are written with upper case characters regardless of the form used in the actual software.

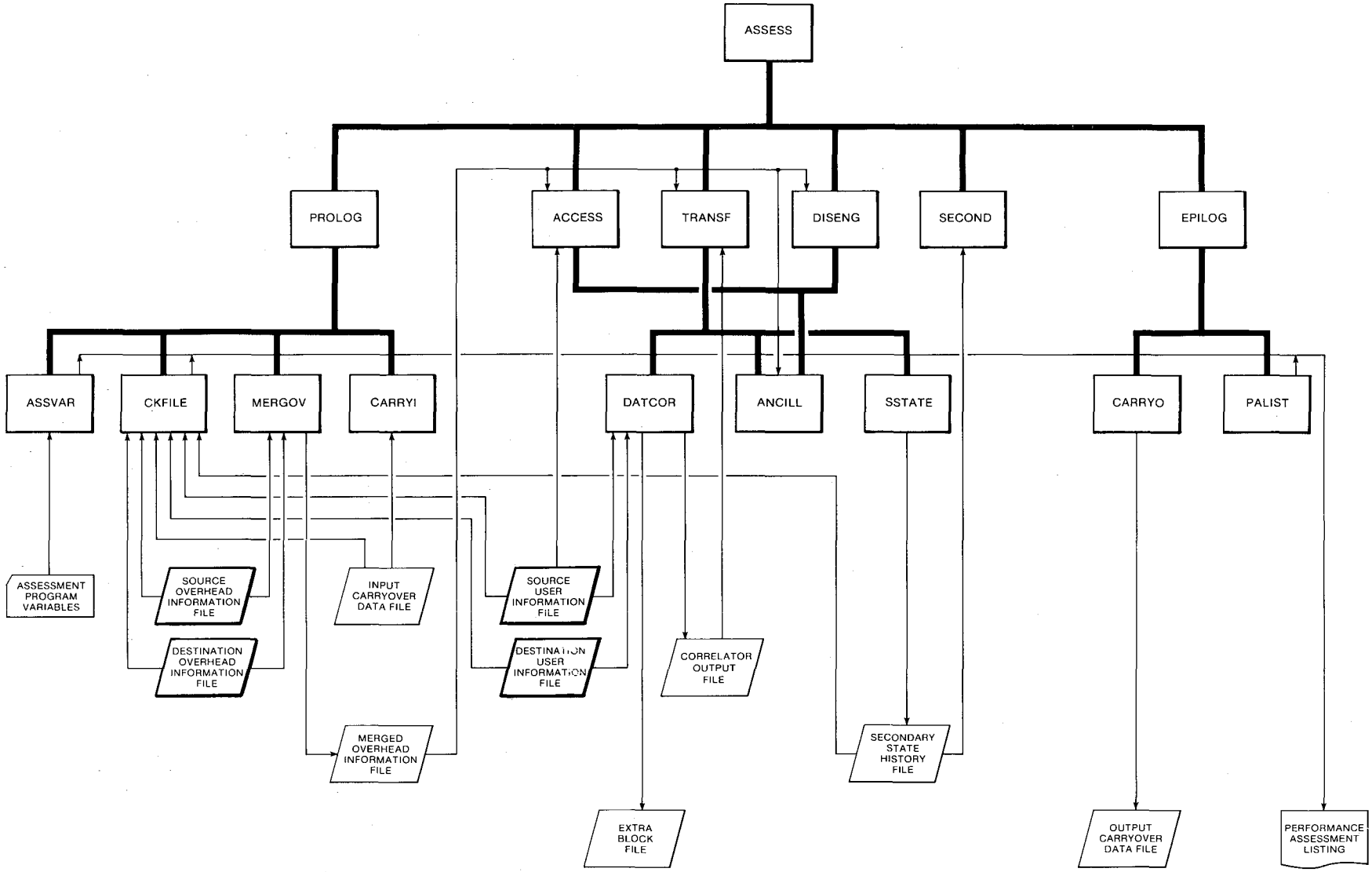


Figure 18. Performance assessment program elements and I/O files.

subroutine reads from that file; a thin line in the opposite direction implies that the subroutine writes to the file.

A detailed description of each ASSESS subroutine is provided in the measurement standard. Briefly, subroutines ACCESS, TRANSF, DISENG, and SECOND calculate values for the access, user information transfer, disengagement, and secondary performance parameters, respectively. Subroutine ANCILL supports the former three programs in calculating values for the ancillary performance parameters. Subroutine DATCOR matches transmitted and received bits and identifies undelivered and extra bits to assist TRANSF in evaluating the user information transfer parameters. Subroutine SSTATE determines (a posteriori) the secondary (availability) state of the monitored service during a particular measurement period. Subroutine PROLOG and the subroutines under it prepare extracted performance data batches for processing. Subroutine EPILOG and the subroutines under it print and store the program outputs.

Figure 19 illustrates one of five pages of ASSESS program output. Each performance parameter is listed, followed by the numerical value calculated for the parameter during the current ASSESS program run. (Note that the values shown are for format illustration only.) Other program outputs summarize batch descriptors, user inputs, and raw parameter data.

Each batch of measurement data recorded by the Interface Monitor applies to a specific source/destination/originating user triplet. In order to use proposed FED STD 1043 to characterize the performance of duplex transactions, multipoint transactions, and multiuser networks, it is necessary to aggregate performance data from two or more batches to produce composite performance values.[9] The Performance Assessment program facilitates such aggregation by the use of a pair of carryover data files. At the end of each batch assessment run, key outcome counts and cumulative time durations used in parameter evaluation are recorded in an output carryover data file. On the succeeding batch assessment run, this same file is used as an input carryover data file to initialize the relevant counts and times. Carryover data management is controlled by a set of user-specified carryover data codes. This feature enables users to develop composite parameters representing several directions of flow in duplex or multipoint transactions; or to combine

---

[9]The batches may be collected either concurrently or at different times. In the case of duplex transactions, a single Interface Monitor may record the performance data for both directions of flow.

(PERFORMANCE ASSESSMENT IDENTIFIER PRINTED HERE)
RUN 0

PERFORMANCE MEASUREMENT SUMMARY

PART A - PRIMARY PARAMETERS

1.	ACCESS TIME	37.5	SECONDS
2.	INCORRECT ACCESS PROBABILITY	1.1X10(-04)	*
3.	ACCESS DENIAL PROBABILITY	1.1X10(-02)	*
4.	BIT TRANSFER TIME	0.068	SECONDS
5.	BIT ERROR PROBABILITY	1.5X10(-05)	*
6.	BIT MISDELIVERY PROBABILITY		*
7.	BIT LOSS PROBABILITY	6.8X10(-04)	*
8.	EXTRA BIT PROBABILITY	0	*
9.	BLOCK TRANSFER TIME	0.068	SECONDS
10.	BLOCK ERROR PROBABILITY	1.5X10(-04)	*
11.	BLOCK MISDELIVERY PROBABILITY		*
12.	BLOCK LOSS PROBABILITY	6.8X10(-04)	*
13.	EXTRA BLOCK PROBABILITY	0	*
14.	BIT TRANSFER RATE	102.4	BITS/SECOND
15.	BLOCK TRANSFER RATE	14.6	BLOCKS/SECOND
16.	BIT RATE EFFICIENCY	68	PERCENT
17.	BLOCK RATE EFFICIENCY	68	PERCENT
18.	DISENGAGEMENT TIME	2.25	SECONDS
19.	DISENGAGEMENT DENIAL PROBABILITY	5.7X10(-05)	*

PART B - SECONDARY PARAMETERS

20.	SERVICE TIME BETWEEN OUTAGES	8.2	HOURS
21.	OUTAGE DURATION	38	MINUTES
22.	OUTAGE PROBABILITY	1.4X10(-03)	*

PART C - ANCILLARY PARAMETERS

23.	USER ACCESS TIME FRACTION	0.4 *
24.	USER BLOCK TRANSFER TIME FRACTION	0 *
25.	USER MESSAGE TRANSFER TIME FRACTION	0.02 *
26.	USER DISENGAGEMENT TIME FRACTION	0 *

*NOTE - THE PROBABILITIES AND USER PERFORMANCE TIME FRACTIONS ARE DIMENSIONLESS NUMBERS BETWEEN ZERO AND ONE.

Figure 19. Performance assessment listing--performance measurement summary.

performance data collected from many user pairs in characterizing a multiuser network.

The real value of the Performance Assessment program is its time savings to the user. In the absence of such a standardized, transportable program, users who wished to conduct performance measurements in accordance with proposed FED STD 1043 would be forced to develop an ad hoc data reduction capability for their particular measurement, at substantial cost; and there would be no assurance that the parameter values produced by different reduction methods would be comparable.

The Performance Assessment program was based on the Interim FED STD 1033 parameters rather than the X3.102 parameters because the latter were not firmly established at the time the program was developed. The parameter changes ultimately adopted by ANSI had relatively little impact on the post-test processing, and the conversion of Interim FED STD 1033 parameter values into ANSI X3.102 values was straightforward. The Performance Assessment program is now being revised to directly compute all 21 of the X3.102 parameters.

2.4.4 Statistical Design and Analysis

The Statistical Design and Analysis section of proposed Federal Standard 1043 defines detailed statistical methods for designing performance measurements and analyzing the observed results. It addresses three major topics. The first is qualitative design: i.e., how variables such as time of day, selected user pairs, and traffic loading should be accounted for in the arrangement of measurements. The second is sample size determination: i.e., the number of measurements of each type a user should conduct in order to ensure sufficiently precise results. The third is analysis of data: i.e., how to analyze actual measured data to check pre-test assumptions and refine measurement precision[10] estimates. The methods specified in this section are drawn mainly from three prior reports (Crow, 1974; Crow and Miles, 1977; Crow, 1979).

---

[10] In statistics, the term "precision" refers to the agreement of repeated experimental results. It is used in preference to the more stringent term "accuracy" here to emphasize that the statistical procedures in the standard quantify measurement uncertainties resulting from limited sample size, rather than possible bias or system nonstationarity. For further discussion, refer to Ku (1969).

The statistical guidelines of proposed FED STD 1043 may be illustrated by a step-by-step procedure the standard provides for determining the probability P of a performance failure (e.g., an error in transferring a bit) with a specified relative precision. The specification may be, for example, that 95% confidence limits should be within 50% of P. A sufficient number of observed failures can be calculated as follows:

1. Specify the relative (percentage) precision desired for P in terms of confidence limits and the confidence level (80%, 90%, 95%, or 99%) with which that precision will be attained.

2. Specify the upper bound P_{11max} on the conditional probability P_{11} of failure given a failure on the previous trial, such that $0 < P_{11} \leq P_{11max} < 1$.

3. Locate the desired relative precision on the vertical axis of Figure 20 and draw a horizontal line to the curve marked with the desired confidence level. Draw a vertical line from the point of intersection to the horizontal axis. Read from the horizontal axis the number, s_{ind}, of failures that would be required if the successive trials were independent.

4. Calculate the sufficient number of failures from

$$s_0 = s_{ind} \frac{1 + P_{11max}}{1 - P_{11max}}$$

(or the next largest integer).

5. Proceed with testing until s_0 failures have occurred. The sample size will thus not be predetermined and will vary somewhat randomly. However, if a prior estimate P_0 of P can be made, the sample size will be approximately $n_0 = s_0/P_0$.

The standard specifies a separate procedure for estimating P_{11max} if there is no prior information available. Statistical procedures like these provide users with a quantitative basis for relating measurement precision with measurement cost.

3. TEST OBJECTIVES

The ARPA network measurements project had two major objectives. The primary objective was to verify and demonstrate the proposed data communication performance measurement standard by actually implementing it in a representative test situation. A secondary objective was to obtain some typical performance values to characterize the data communication service provided by the ARPA network and its host computers to end users. It was

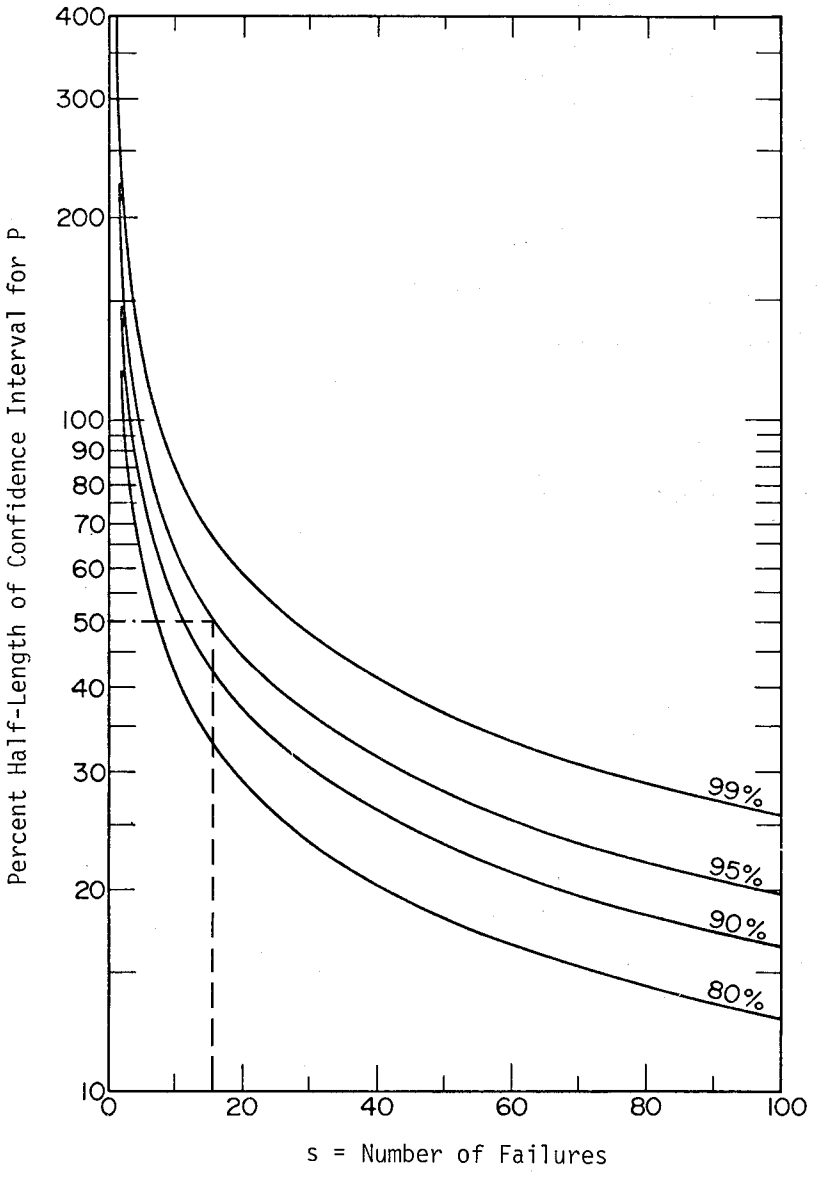


Figure 20. Relative precision in estimating P from large samples when number of failures is prescribed and successive observations are independent. Curve labels are confidence levels.

anticipated that these values would be useful first, in understanding differences between subnetwork and user-perceived performance; and second, in assessing proposed refinements to the parameter definitions themselves.

This section defines these overall objectives in more detail, in the form of a series of questions to be answered by the project results. The questions are clustered in two groups: those which relate to the adequacy of the proposed measurement standard, and those which relate to the actual measured data. They are discussed, in turn, in the following subsections. Their answers are developed in Sections 4 through 6, and summarized in Section 7, of this report.

3.1 Primary Objectives

This section poses a series of technical questions which naturally arise in assessing the adequacy of proposed FED STD 1043 as a specification for a performance measurement system. The questions are grouped and arranged in sequence in accordance with the four performance measurement system elements identified earlier (Figure 11): i.e., the Data Extraction element, the Data Files, the Performance Assessment program, and the Statistical Design and Analysis procedures.

3.1.1 Data Extraction Element

Measurement of data communication performance obviously requires the observation, interpretation, and recording of information exchanged between the data communication system and its users. These functions are performed by the Data Extraction element of the proposed FED STD 1043 performance measurement system. As discussed in Section 2, that element consists, in a typical implementation, of two Interface Monitors and a Synchronized Time Reference. FED STD 1043 specifies these Data Extraction components in relatively general functional terms to permit a variety of physical implementations.

The initial questions relative to implementation of the Data Extraction element deal with its inputs: i.e., the identification and instrumentation of the user/system interfaces. The essential questions are:

1. Can the end user/data communication system interfaces be identified unambiguously from the guidance provided in the standard? Where are those interfaces in the configuration tested?

2. What specific data communication-related events take place at the end user interfaces? Can these interface events be sequentially related in a "session profile" of the type illustrated in the standard? Can they be observed and recorded with a negligible (or at least measurable) effect on the process under examination?

3. What is the effect of system or user failures on the observed event sequence? Is it necessary (and practical) to represent possible failure events in a session profile?

4. How do the specified end user interfaces relate to the "computer/communications" interfaces traditionally used in performance measurement? Is the difference between the two significant from an instrumentation point of view?

The answers to these questions will determine whether it is, in fact, feasible to identify and instrument the end user/data communication system interfaces in accordance with proposed FED STD 1043 in a typical measurement situation.

A second group of questions arises in connection with the Interface Monitor processing of observed interface events. These are:

1. Can the various events observed at the user/system interfaces in the configuration tested be unambiguously associated with the universal "reference events" defined in the standard? What is the exact association between these events in the case studied? Are there reference events with no corresponding interface event, or vice versa? Are the "user information" and "overhead information" categories clearly distinguishable from the guidance in the standard?

2. Is it practical and economically feasible to provide a synchronized time reference to each of two geographically distant Interface Monitors? How can this be accomplished, and at what cost?

3. What accuracy can be achieved in the time-stamping of observed interface events? What mechanisms must be used to correct for differences between the actual event times and the associated clock readings?

The answers to these questions will determine whether all information needed to calculate values for the standard performance parameters can actually be obtained at the interfaces defined.

A third group of questions arise with regard to the generation and recording of Interface Monitor outputs. These are:

1. How can the separate event histories recorded at each end of a data communication system be brought together in one computer for processing without introducing errors not present in the original recorded data?

2. How much computer time is required to translate the "raw" event records into the specified standard output format? Should that translation be performed on-line (during actual data collection) or off-line (after the test)? How large, and how complicated, are the translation programs?

The answers to these questions will determine the feasibility of assembling remotely recorded performance data for centralized processing, and will strongly influence the design and operation of the Interface Monitor function.

3.1.2 Data Files

A key feature of proposed Federal Standard 1043 is the provision of a standard Performance Assessment program capable of transforming observed reference event histories into performance parameter values in an efficient, uniform way. The use of such a program requires that observed interface events be recorded in a set of common data files that can be directly written by the Interface Monitor, and directly read by the Performance Assessment program. The standard data files defined in proposed Federal Standard 1043 fulfill this need.

A number of significant questions arise with respect to the implementation of these data files. They are:

1. Can the specified data file formats be directly written and read by software developed using different programming languages?

2. What requirements will such standard files place on the memory capacity of the recording equipment? Are the memory requirements commensurate with the capacities of host/measurement systems which might reasonably be expected to implement the standard? How much is the memory requirement increased by the choice of a machine-independent data file format?

3. What is the impact of data aggregation (e.g., combining measurement results for many user pairs) on the data file structure? On overall memory requirements? How important is the "carryover data" capability to standards users?

The answers to these questions will have a critical impact on the transportability of the standard Performance Assessment program.

3.1.3 Performance Assessment Program

The preceding paragraphs pointed out the importance of the standard Performance Assessment program to the proposed FED STD 1043 measurement

approach. Significant questions associated with the implementation and use of that program include the following:

1. How large is the program, including both machine instructions and data? Will the program "fit" within the memory capacity of computer systems on which it might reasonably be implemented?

2. How long does the program take to process a "batch" of recorded measurement data?

3. Does the program correctly determine the outcomes of data communication attempts under all observed conditions of performance? Under what conditions does the program fail or give misleading results?

4. What flexibility must such a program provide to users with respect to (a) aggregation of measurement data from many batches, (b) calculation of parameter subsets, (c) adaptation of program variables to measured system characteristics?

The answers to these questions will strongly influence the utility of the standard Performance Assessment program, and consequently, the efficiency and economy with which the standard performance parameters can be measured.

3.1.4 Statistical Design and Analysis Procedures

Performance measurement requires the estimation of parameter values on the basis of a finite "sample", or set, of observations. The use of finite samples inevitably produces a certain imprecision in the measured parameter values. That imprecision can always be reduced through the use of a larger sample, but only at the cost of a longer (and more expensive) test. The statistical design and analysis procedures specified in proposed FED STD 1043 enable users to design performance measurements on the basis of specified precision and cost objectives; and to analyze actual measured data to determine the precision actually achieved during a test.

One important objective of the ARPA network measurements project was to assess the utility of the proposed FED STD 1043 Statistical Design and Analysis procedures in an actual measurement situation. Significant questions to be answered during the project included the following:

1. How should a data communications user's measurement precision objectives be determined? Can/should FED STD 1043 provide guidance to users in establishing such objectives?

2. How strongly do underlying measurement conditions (e.g., time of day, direction of transfer) influence the measured

parameter values? Does the current FED STD 1043 draft provide adequate guidance to users in identifying such conditions and assessing their impact?

3. Given a specified measurement precision objective, is it possible to determine the sample size required to achieve that objective in a straightforward way?

4. Can the post-test data analysis procedures and formulas provided in the current FED STD 1043 draft be applied to the ARPA network test data with reasonable effort? How much do the precision estimates calculated on the basis of the observed test results differ from the objectives established before the test? How realistic were the statistical assumptions on which the pretest calculations were based?

The answers to these questions will provide a basis for assessing the practicality and utility of the Statistical Design and Analysis procedures specified in the proposed measurement standard.

3.2 Secondary Objectives

The ARPA network has been analyzed, simulated, and measured in an large number of studies since its development in the late 1960's, and published data on its performance is abundant. However, much of the published data on ARPA "network" performance actually refers to the subnetwork - i.e., the IMP's, TIP's, and 50 kb/s leased lines interconnecting host computers. Because it excludes the effects of higher level protocols which are executed in the host computer, this data is often not directly usable in characterizing the performance of the end-to-end service ultimately delivered to ARPA network end users - e.g., application programs executing in the host computers. A secondary objective of the ARPA network measurements project was to provide such a user-oriented, end-to-end characterization of ARPA network/host computer performance. It was anticipated that the process of collecting that user-oriented performance data would also provide a basis for assessing proposed refinements to the parameter definitions.

This subsection poses a series of questions to be answered in accomplishing those secondary ARPA network measurements project objectives. The questions are grouped in three categories, in accordance with the three primary communication functions defined earlier: access, user information transfer, and disengagement. Within each category, questions related to ARPA network performance are presented first, followed (where appropriate) by

questions related to the parameter definitions. Most of the latter questions focus on differences between the Interim FED STD 1033 and X3.102 parameters.

3.2.1 Access Parameters

Interim Federal Standard 1033 defines four access performance parameters: three primary parameters (Access Time, Incorrect Access Probability, and Access Denial Probability), and one ancillary parameter (User Fraction of Access Time). ANSI X3.102 defines one additional primary access parameter (Access Outage Probability).

A comprehensive measurement of these parameters on the ARPA network should provide answers to the following questions:[11]

1. How long must an ARPA network end user wait, after requesting communication service, before his first block of information is actually input to the system for transmission? How variable is that delay? Is it significantly influenced by direction of transfer, application program priorities, or time of day? What proportion of the delay is attributable to subnetwork packet transmissions? What proportion is attributable to delays introduced by the users?

2. What is the likelihood that transmitted user information will be directed towards a destination other than the one intended as a result of a system connection establishment error?

3. What is the likelihood that the system will fail to give the user access to communication service (within a specified maximum time) on any given request? How frequently are such failures attributable to system outages? How sensitive is that likelihood to the host computer utilization?

The ARPA network access measurements were also intended to answer a number of questions regarding the detailed definition of the access performance parameters. These were:

1. Are the observed values for Access Time substantially influenced by the choice of "start of block transfer" rather than "connection confirmation" as the end of the access function? Does the ancillary parameter User Fraction of Access Time provide an adequate method of factoring out that influence where user-independent parameter values are desired?

---

[11] In each of the following questions, "the system" includes all functional and physical elements which participate in communicating user information between the host computer application program/operating system interfaces. "The system" thus includes the Network Control Program, the host computer operating system, and associated high-level protocols.

2. Is Incorrect Access Probability practical to measure as defined in the current standards? Is the distinction between Incorrect Access Probability and Block Misdelivery Probability significant from a measurement standpoint?

3. How strongly is the value for Access Denial Probability influenced by the choice of maximum access time? Would a different maximum access time value, or a different algorithm for determining that value, be more appropriate than the one selected?

4. ANSI X3.102 differs from Interim FED STD 1033 in that it distinguishes Access Outage (no system response) from Access Denial (negative system response or excessive system delay). Are these two outcomes readily distinguishable in the case studied? Is that distinction meaningful and important in the case studied?

The answers to these questions will be useful in characterizing the access performance of the ARPA network from a user perspective; and in assessing the practicality and utility of the access parameters themselves.

3.2.2 User Information Transfer Parameters

Interim Federal Standard 1033 defines 19 parameters which directly or indirectly describe the performance of the user information transfer functions. These include 10 parameters which focus on the outcomes of individual bit or block transfer attempts (Bit/Block Transfer Time, Bit/Block Error Probability, Bit/Block Misdelivery Probability, Extra Bit/Block Probability and Bit/Block Loss Probability); 7 parameters which focus on the transfer of multiblock "messages" (Bit/Block Transfer Rate, Bit/Block Rate Efficiency, and the three "secondary" parameters Outage Probability, Service Time Between Outages, and Outage Duration); and 2 ancillary parameters (User Block Transfer Time Fraction and User "Message" Transfer Time Fraction). The ANSI X3.102 standard specifies 13 UIT parameters which are a subset of these, excluding Bit Transfer Time, Block Transfer Rate, both Rate Efficiencies, Service Time Between Outages, and Outage Duration.[12]

Questions related to the measurement of these parameters on the ARPA network include the following:

1. What total delay does a bit or block of user information experience, on average, between a request for its transmission

---

[12]Other differences include renaming of the two ancillary parameters and Bit Transfer Rate, and a redefinition of the latter.

by the source application program and the completion of its delivery to the destination program? How is that delay influenced by block length? How variable is that delay, for a given block length? Does user performance contribute significantly to the observed end-to-end delay in the case studied? How do the measured Block Transfer Times compare with previously measured "average message delays" for the subnet?

2. The ARPANET error control algorithms have been designed to ensure that undetected bit (or block) errors will occur on the order of "years to centuries apart" (Kleinrock, 1976). Has this objective been attained, or is it, in fact, possible to observe undetected bit errors in a measurement spanning only a few million bits?

3. What is the likelihood that a unit of information delivered by the ARPANET to a given destination user will, in fact, have been intended for some other user?

4. Kleinrock (1976) has reported that "on the average, every hundredth message which enters the ARPANET will not reach its destination. The reason for this undesirable behavior is that many destination hosts are tardy in accepting messages." Is the loss of user information as frequent today, in the configuration tested, as it was in 1976? What proportion of the observed Block Loss outcomes is attributable to performance timeouts as opposed to actual absence of the data in the received user information record? Is Block Loss primarily attributable to "tardy" hosts, or does the subnetwork also contribute?

5. Data loss and data duplication are similar in that both can be caused by ARQ protocol failures and system "crashes". Does data duplication occur with measurable frequency in the ARPANET? Are nonduplicate extra bits delivered to ARPANET users with measurable frequency? What phenomena cause such events, assuming they occur?

6. What throughput is typically achieved between host application programs utilizing the ARPA network? How does this throughput compare with the theoretical maximum process-to-process bandwidth of the network (Kleinrock, 1976)? How does it compare to the values obtained in earlier, experimental measurements? How does it compare with the allocated channel signalling rate? To what extent are the measured throughput values influenced by source and destination user delays?

7. What is the overall availability of data communication service between application programs in the case studied? How is the downtime subdivided between the subnetwork and the hosts? How does the subnetwork (e.g., IMP/TIP) downtime observed in the case studied compare with earlier measured values?

The ARPA network user information transfer measurements were also intended to answer a number of questions regarding the detailed definition of the UIT parameters. These were:

1. Are Bit Transfer Time and Block Transfer Time distinguishable in the case studied? Is a comparison of these parameter values useful in expressing the influence of block length on transfer time?

2. Are Bit Error Probability and Block Error Probability independently useful in characterizing transfer accuracy in the case studied? Does a comparison of these values provide useful information on error clustering?

3. Are Bit and Block Misdelivery Probability, as defined in Interim FED STD 1033 and X3.102, measurable in the case studied? Could their measurement be made simpler if the parameters were defined differently?

4. How strongly are the values for Bit and Block Loss Probability influenced by the choice of maximum block transfer time? Would a different algorithm for determining maximum block transfer time be more appropriate in the case studied?

5. ANSI X3.102 differs from Interim FED STD 1033 in the definition of User Information Bit Transfer Rate. Are these differences significant in the case studied? Which definition is preferable from a measurement standpoint?

6. ANSI X3.102 differs from Interim FED STD 1033 in that the Block Transfer Rate and Bit/Block Rate Efficiency parameters are omitted. Does the omission of these parameters detract significantly from the completeness of the performance description?

7. ANSI X3.102 differs from Interim FED STD 1033 in the selection and definition of availability parameters. Which approach appears preferable from a measurement standpoint? Are Service Time Between Outages and Outage Duration measurable, with reasonable precision, in the case studied? Does the omission of these parameters detract significantly from the completeness of the performance description? How strongly are the Outage (or Service Denial) Probability values influenced by the choice of degraded performance (or outage) thresholds? How much do the required sample sizes differ under the FED STD 1033 and X3.102 threshold criteria? Is the difference significant from a measurement standpoint?

The answers to these questions will be useful in characterizing the transfer performance of the ARPA network from a user perspective; and in assessing the practicality and utility of the UIT parameters themselves.

3.2.3 Disengagement Parameters

Interim Federal Standard 1033 defines three disengagement performance parameters: two primary parameters (Disengagement Time and Disengagement Denial Probability) and one ancillary parameter (User Disengagement Time Fraction). The same three parameters are specified, with minor refinements, in ANSI X3.102. Questions related to the measurement of these parameters on the ARPA network include the following:

1. How long must a user wait, after requesting disengagement from an established data communication session, for the disengagement function to be successfully completed? Does that delay differ substantially for each user? How variable is the delay? What proportion of the delay is attributable to subnetwork packet transmissions? What proportion is attributable to delays introduced by the users themselves?

2. What is the likelihood that the system will fail to detach a user from an established data communication session (within a specified maximum time) on any given request? How sensitive is that likelihood to the specified maximum time?

The ARPA network disengagement measurements were also intended to answer a number of questions regarding the detailed definition of the disengagement performance parameters. These were:

1. The ANSI X3.102 definitions for Disengagement Time and Disengagement Denial Probability differ from their Interim FED STD 1033 counterparts in one important respect: they allow the specification of separate parameter values for each end user in cases where the disengagement functions at the user interfaces are dissimilar. Is this distinction important in the case studied? How meaningful are the "aggregate" parameter values representing both user interfaces?

2. How strongly are the values for Disengagement Denial Probability influenced by the choice of maximum disengagement time? Would a different algorithm for determining maximum disengagement time be more appropriate in the case studied?

The answers to these questions will be useful in characterizing the disengagement performance of the ARPA network from a user perspective; and in assessing the practicality and utility of the disengagement parameters themselves.

3.3 Summary

The preceding subsections have defined the objectives of the ARPA network measurements project in terms of a series of questions to be answered by the

project results. These questions were clustered into two groups. The key issue addressed by the first group of questions was the feasibility of implementing proposed Federal Standard 1043 in an actual performance measurement system. Detailed questions dealt with the time and effort required to develop each element of the measurement system; the size and complexity of the resulting system; and the performance of that measurement system in terms of speed, accuracy, and reliability. Clearly, these are questions which must be answered before the standard is promulgated in a permanent form.

The questions in the second group addressed two key issues: (1) the user-to-user performance of the ARPA network and its host computers, as contrasted with the performance of the subnetwork; and (2) the practicality and utility of the standard performance parameters in characterizing a typical data communication service. Results will clarify the relationship between end-to-end and subnetwork performance; and will provide a useful (and perhaps final) set of refinements to the 1033/X3.102 parameters.

4. MEASUREMENT APPROACH

This section describes the design of the prototype data communication performance measurement system NTIA/ITS and NBS/ICST developed as a trial implementation of proposed Federal Standard 1043. As noted earlier, that measurement system was designed to assess the data communication service provided to a typical pair of ARPA network end users - host computer application programs. The material in this section is divided into four subsections, corresponding to the four performance measurement system elements defined earlier (Figure 11).

4.1 Data Extraction Element

The general functional requirements for the Data Extraction element of the proposed FED STD 1043 performance measurement system were presented earlier in this report (Section 2.4.1). This subsection describes the design of the Data Extraction subsystem developed to perform these functions in the ARPANET experiment.

The major features of this design are summarized as follows. At each user/system interface, a single application program (1) performed all end user activities related to the access, user information transfer, and disengagement functions during monitored data communication sessions, and (2) executed the

input function of the associated Interface Monitor (i.e., detected and determined the time of all performance-significant events at the monitored interface). This design feature exemplifies the active approach to data communication performance measurement described in Appendix B of proposed FED STD 1043. The Interface Monitor processing and output functions were performed off-line by a set of computer programs designed to convert recorded raw data into the standard ASCII character files specified by proposed FED STD 1043. The synchronized Time Reference was obtained from the NBS satellite time dissemination service utilizing the Geostationary Operational Environmental Satellite (GOES).

The detailed description of the Data Extraction subsystem design is divided into four parts. The first part provides background for that design by: (1) describing the hardware and software configurations of the data communication system tested; (2) identifying and describing the user/system interfaces and all performance-significant events at those interfaces; and (3) relating the system-specific interface events to the corresponding FED STD 1043 reference events. The second part describes major hardware devices and some general features of the software used in the Data Extraction subsystem. The third part describes the design of the on-line phase of Data Extraction; i.e., the design of the user/monitor programs used to generate and record interface events. The fourth part describes subsequent off-line processing which converted extracted raw data into standard ASCII character files for input to the FORTRAN Performance Assessment program.

4.1.1 Design Background

Figure 21 illustrates the overall hardware configuration tested. The basic hardware consisted of two similar PDP-11 host computer systems interconnected by the ARPA subnetwork. The NTIA/ITS host, located in Boulder, Colorado, was a PDP-11/40 computer with 256K bytes of semiconductor random access memory (RAM) and peripherals including a cathode-ray tube (CRT) terminal, an LP-11 line printer, three RK-05 disk units, two TU-10 magnetic tape units, and an LA-36 teletyewriter (TTY), which served as the system control console. The computer was connected to the DOCB TIP via a standard ARPANET Local Host interface device. The NBS/ICST host, located in Gaithersburg, Maryland, was a PDP-11/45 computer with 256K bytes of semiconductor RAM, a 1K cache memory, and a set of peripherals similar to

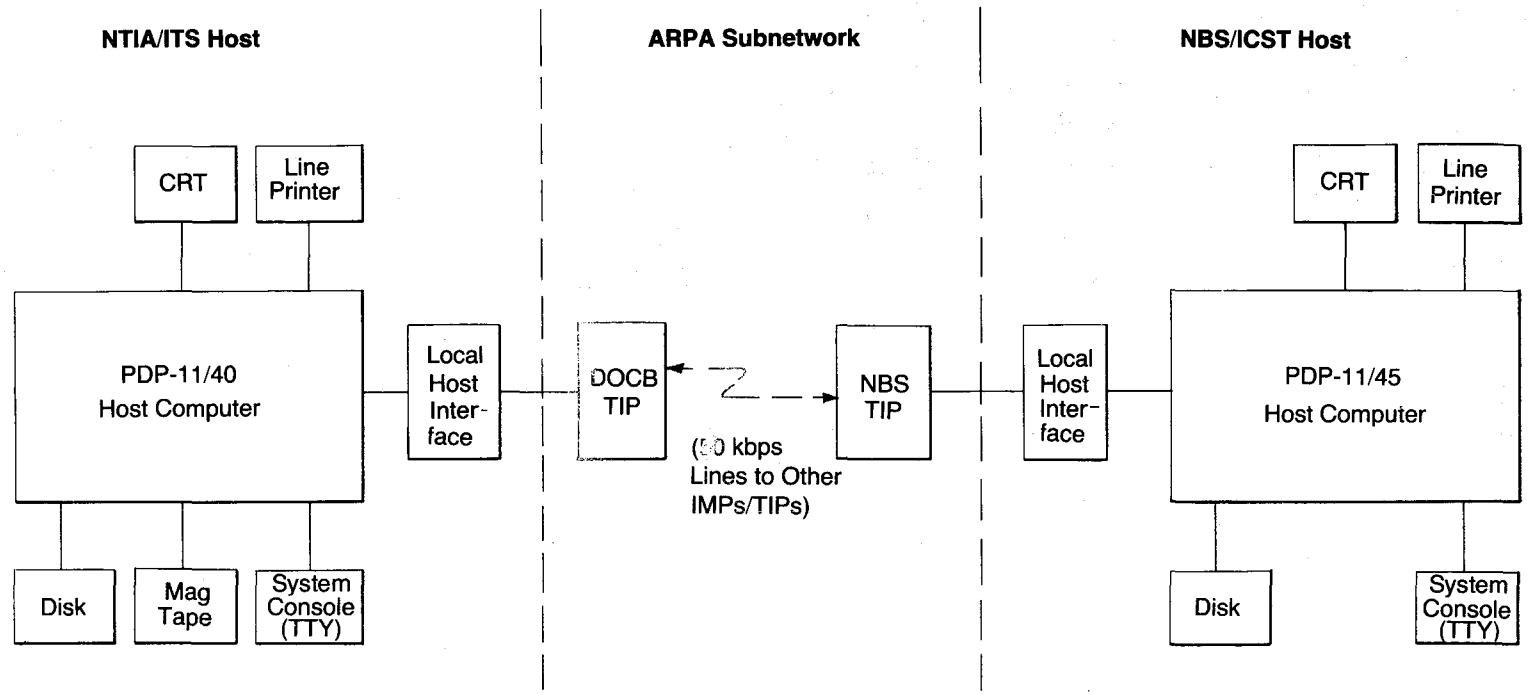


Figure 21. Hardware configuration tested.

those at NTIA/ITS. The NBS/ICST host computer also shared use of four 65 Megabyte capacity disk units.

During the time period of the tests described in this report there were, typically, one to four time-sharing users connected to the NTIA/ITS host and two to eight time-sharing users connected to the NBS/ICST host. In most cases, these users performed tasks that did not make heavy computing demands on the system. With rare exception, the other users were not using the ARPANET. Usage of the source and destination TIPs by other hosts and terminals was not monitored.

Figure 22 shows the host computer software configurations. Both the NTIA/ITS PDP-11/40 and the NBS/ICST PDP-11/45 ran a UNIX[13] Version 6 operating system (Thompson and Ritchie, 1975) which included input/output drivers for the Local Host interface and other local peripherals. Each computer ran a UNIX Network Control Program (NCP), which was originally developed for the ARPA network at the University of Illinois (Chesson, 1975). The NCP's ran as application-level programs, and implemented the ARPA network host-to-host protocol (Carr et al., 1970). User application programs communicated with each other and with peripheral equipment via the usual operating system calls.

Figure 23 illustrates the logical flow of user data within the host computers, and also indicates the end user/data communication system interfaces and associated interface events. The end user in each case was a host computer application program, and the end user/data communication system interface was the functional interface between that program and the host computer's operating system (OS). The Network Control Programs (NCP's) regulated the exchange of user information between hosts via the ARPA subnetwork.

Any discrete transfer of information across a user/system interface is called an interface event. The specific interface events which occur at the user/system interfaces in the configuration tested are listed in the lower portion of Figure 23. They are of two general types: system calls, which are issued by an application program to request the performance of a particular operating system function; and system responses, which are issued by the operating system to indicate completion of a previously requested function.

The four operating system calls which may be issued by an application program executing under UNIX are OPEN, WRITE, READ, and CLOSE. With respect

---

[13]Registered trademark of Bell Laboratories.

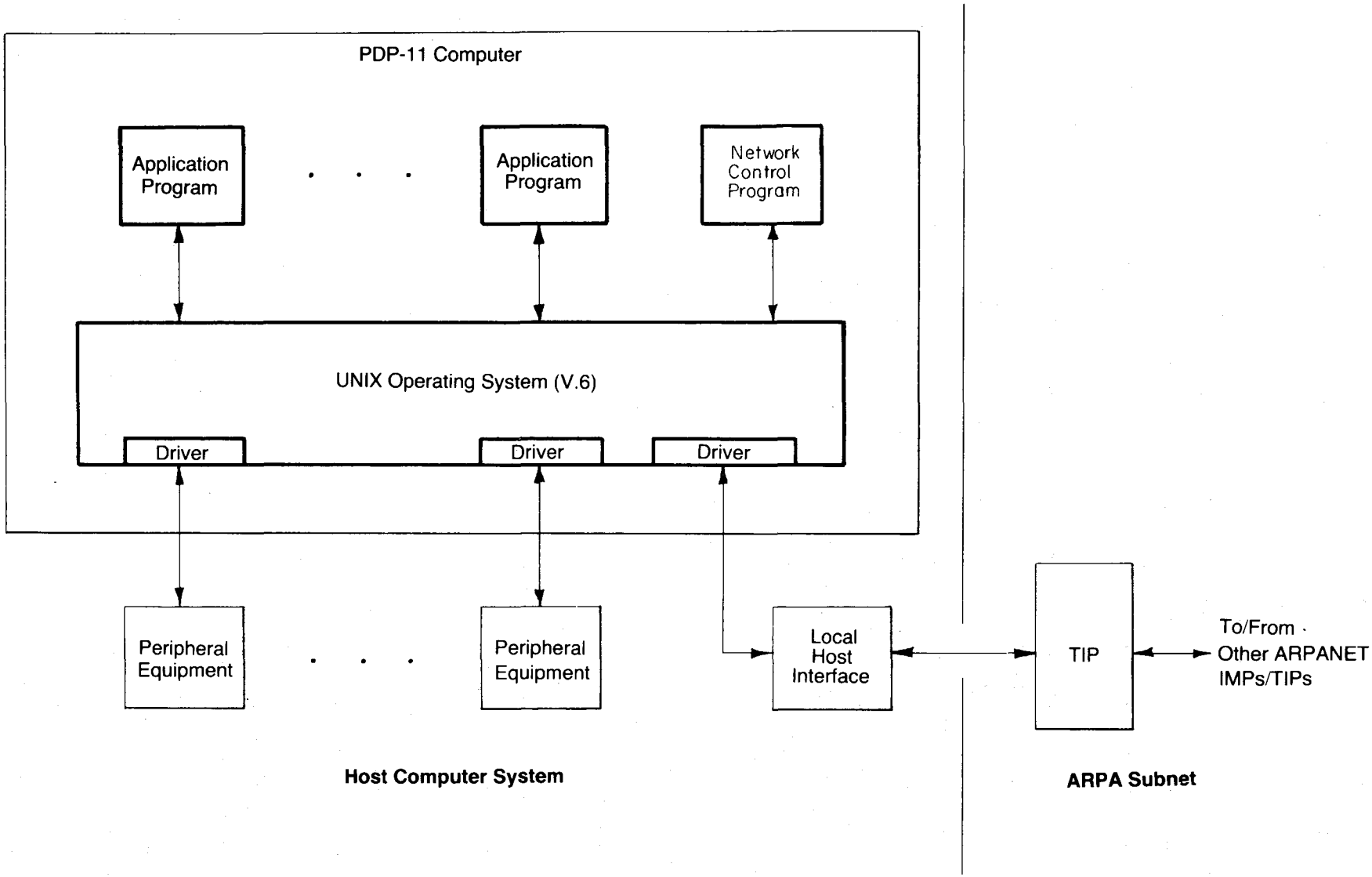


Figure 22. Host computer software configurations.

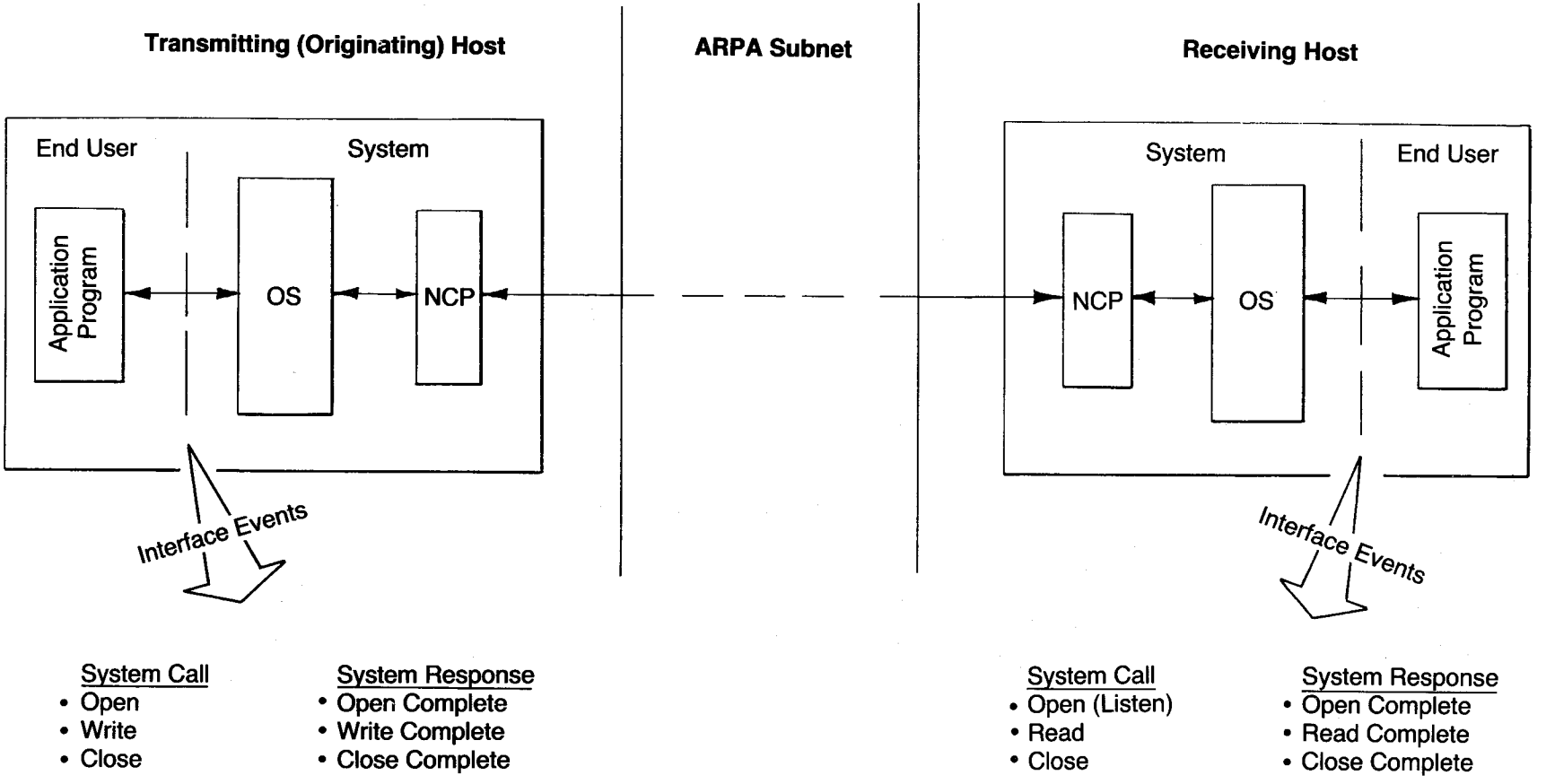


Figure 23. Logical data flow within the host computers.

to ARPANET communication, OPEN and CLOSE are used to establish and release logical connections, called "virtual circuits", between application programs. WRITE causes a specified block of user information to be passed from a source application program to its local NCP for transmission over an established connection. READ causes received user information to be passed from the receiving NCP to the local (requesting) application program. In the normal case, the system's "complete" response indicates that the requested function has been successfully accomplished. System failures are indicated by special exception codes.

Under UNIX, the operating system calls used in local data processing are the same as those used in ARPANET communication. Thus, each application program also issues OPEN, WRITE, READ, and CLOSE system calls to access its local data files; to exchange information with other application programs in the same computer; and to communicate with the local program operator via peripherals. To an application program, the ARPA NCP, subnetwork, and remote host are like any other external communicant.

As discussed in Section 2.4, successful measurement of the 1033/102 parameters depends on a proper translation of the system-specific events observed at the monitored user/system interfaces into the system-independent reference events on which the parameter definitions are based. The ten primary reference events defined in proposed FED STD 1043 were defined earlier, in Figure 15. The user/system interface events depicted in Figure 23 can be directly related to those reference events as follows.

Reference Event 1. The OPEN system call issued by the user originating a data communication session is an Access Request event.

Reference Event 2. The OPEN system call issued by the user not originating a data communication session is a Nonoriginating User Commitment event.

Reference Event 3. When the system is unable to establish a requested connection, it notifies the requesting application program by issuing an OPEN COMPLETE response with a negative (-1) response code. That negative OPEN COMPLETE response is a System Blocking Signal.

Reference Event 4. If an originating application program were to "change its mind" and issue a CLOSE system call while a previously issued OPEN was still pending, that CLOSE would be an Originating User Blocking signal. There is no system call at the non-originating application program interface that corresponds to a User Blocking signal. (The existence of a Nonoriginating User

Blocking signal is not required to measure the standard performance parameters).

Reference Events 5 and 6. The first WRITE system call issued by a source application program after connection establishment corresponds to both the Start of First Block Input to System (reference event 5) and the Start of Block Transfer for that block (reference event 6). The standard notes that these two reference events are coincident in many systems. All subsequent WRITE system calls issued by a source application program during a data communication session are also Start of Block Transfer events.

Reference Events 7 and 8. Operating system issuance of a READ COMPLETE response with a nonzero byte count to a destination application program constitutes both the Start of Block Output to Destination User (reference event 7) and the End of Block Transfer (reference event 8). The standard notes that these two events are coincident in many systems.

Reference Event 9. The first CLOSE system call issued by a user participating in an established data communication session is the Disengagement Request for both participating users. This is true (as noted in the standard) because a connection with one end has no meaning.

Reference Event 10. The system issues a CLOSE COMPLETE response to both participating application programs to confirm the termination of an established connection. Each of these separate responses is a Disengagement Confirmation signal.

The reference events defined in proposed FED STD 1043 also include a set of "ancillary" events, which mark transitions of performance "responsibility" from a user to the system or vice versa. The interpretation of those events in the case studied is straightforward: each operating system call transfers performance responsibility from the issuing user to the system, and each operating system response transfers responsibility from the system to the receiving user.

There are minor differences between the primary reference events defined in the 1033/1043 standards and those defined in ANSI X3.102. The first of these differences is the deletion of event 7 (Start of Block Output to Destination User). That event was needed only to define Bit Transfer Time, a parameter which is not specified in the ANSI standard. The second difference is the addition of four "derived" user information transfer events, which mark, respectively, the beginning and end of the input and output of a "transfer sample" - the randomly chosen user information sequence used to calculate the throughput and availability parameter values. The latter events are "derived" in the sense that they are defined by an a posteriori analysis

of the measurement data, rather than by the occurrence of a particular user/system interface signal. They correspond, in general, to the start and end of transfer of the first and last blocks in a sample.

The ARPA network Interface Monitor software was programmed to record each UNIX operating system call and response by a corresponding reference event as defined above.

4.1.2 Data Extraction Hardware and Software Tools

This section describes hardware devices and some general features of the software used in the Data Extraction subsystem for the ARPANET measurements. The hardware devices, indicated by boldface in Figure 24, consisted of a GOES satellite antenna/clock receiver system at each site, an operator terminal connected via a Multi Line Unit (MLU) to the DOCB TIP, and the associated device interfaces. No special designs were required, since all Data Extraction hardware was off-the-shelf.

The Synchronized Time Reference required to record event times at the two distant user interfaces in the ARPANET experiment was provided by the NBS time dissemination service utilizing the GOES satellite (Kamas and Howe, 1979). This service makes it possible to obtain a time signal accurate to within 1 millisecond anywhere in North America. Several vendors supply an antenna/receiver/interface package to obtain time from the satellite. The receiving systems used in this experiment consisted of a 1 ft square active antenna connected via coaxial cable to a small (9 in x 17 in x 1-3/4 in) clock receiver/interface unit. The antenna operates satisfactorily inside many buildings. Each clock receiver was equipped with a serial RS-232-C interface, which was connected to the PDP-11 host via standard serial communication cards. The cost of the package for each site was about $2,000 in 1980.

The additional terminal used in the experiment was a standard dot matrix printing terminal interfaced into the Multi Line Unit connected to the DOCB TIP. It was interfaced at 1200 bps, although any of the usual speeds from 300 to 2400 bps could have been used. As shown in Figure 24, this terminal was connected to the NBS UNIX-3 host computer in Gaithersburg without going through the ITS host in Boulder. Under normal conditions, this arrangement allowed a single operator at Boulder to conduct ARPANET measurements without involving personnel at Gaithersburg. To the Boulder operator, it appeared that the NBS host was in the next room rather than 1600 miles away. Software

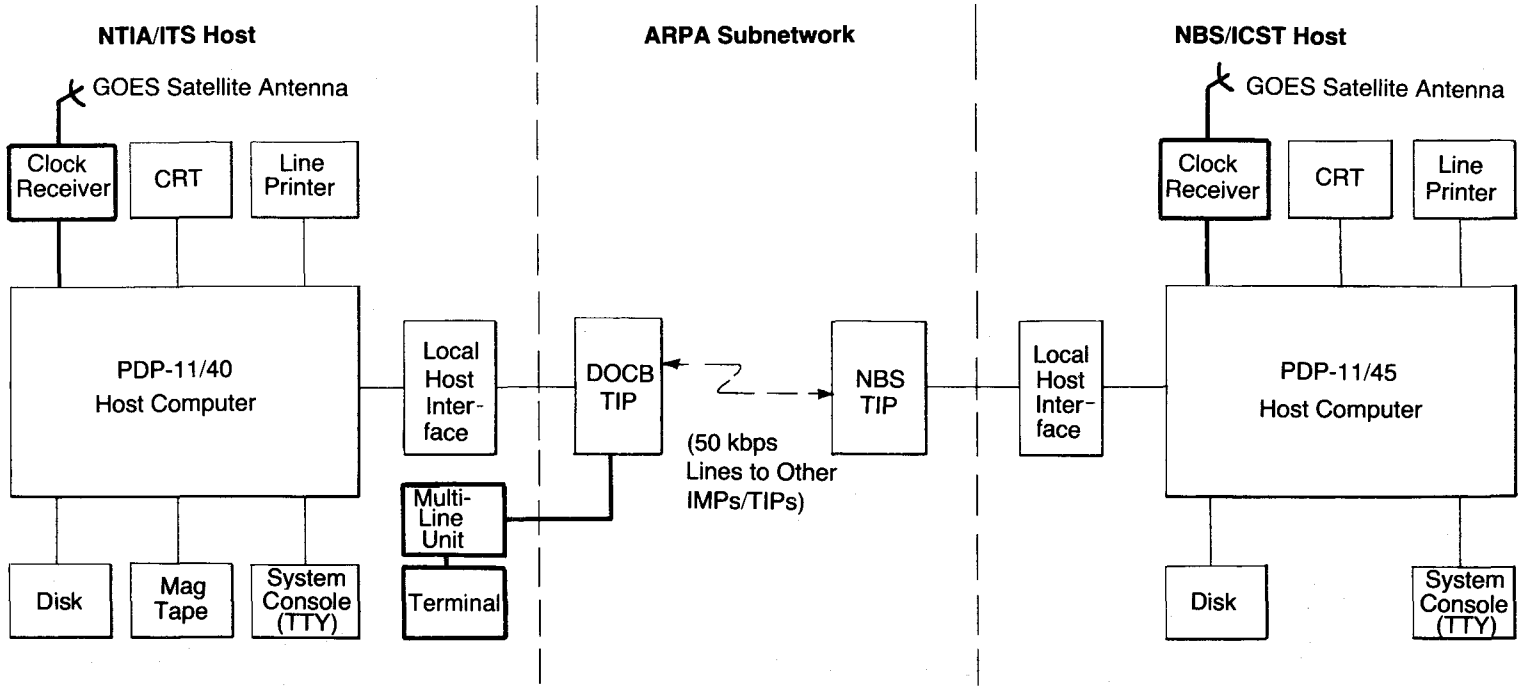


Figure 24. Additional hardware (in boldface) required for testing.

connection procedures are described fully elsewhere (BBN, 1977) and will not be discussed here.

All application programs and routines used the Data Extraction subsystem of the ARPANET experiment were written in the "C" programming language (Kernighan and Ritchie, 1978). This is a general purpose language which has been used for writing operating systems (e.g., UNIX), as well as programs for numerical, text-processing, and data-base management applications. It is also a relatively "low level" language, oriented toward the bits, bytes, and addresses dealt with by most computers. This latter feature, and the fact that "C" is a structured language, were exploited in designing efficient procedures for performing the tasks of the Data Extraction element. Software design and coding were done in a modular fashion. Main programs were kept fairly short (and intelligible) by using procedures or routines to perform specific tasks. Thus, additions and modifications to the Data Extraction software were straightforward.

The UNIX operating system includes several major features which were frequently utilized in developing and implementing the Data Extraction software:

1. I/O Redirection. This feature permits the redirection of a standard output (e.g., a program print statement) that would normally have appeared on a CRT screen, to some other device (including a disk file). Thus, without modifying any software, the program output may be preserved in a file for later use by means of a simple command like

   runprog >filename

   where the name of the program is RUNPROG and its output (except for errors) is totally redirected into the file called FILENAME.

2. Command Files. These provide a method of storing a sequence of commands in a text file, then executing each command just as though it were being typed by an operator. Command files may also include local and passed parameters, numerics with simple addition and subtraction computations, logical comparisons, and simple branching. All test and post-test processing procedures were executed under control of command files.

3. Hierarchial File Structure. This feature allows different directories to exist on the same disk for different purposes. Thus, no directory need be excessively long. For example, data for each month were stored under a directory whose name was the three letter abbreviation of the month (e.g., oct, nov, dec).

4. Built-in Utilities. These include a text editor, and enable the operator or user of the system to convert data or text from one form to another with a minimum of software development. For example, several important steps in preparing a text file of statements for a particular data run were pre-processed with a command file invoking utilities such as the editor and the sort routines.

5. Filters, Forks, and Pipes. A filter is a program which takes its input from one source or file, transforms the data, and outputs it to a different file. A fork permits starting a program then returning control to the initiator of the process. This allows concurrent processes to coexist in the machine. A pipe allows the directing or funnelling of data from one program to another program (i.e., the standard output of the first program becomes the standard input of the second program). The pipe, with the TEE program, causes the standard output of a program to appear on the standard output device (i.e., the CRT or TTY) for the operator to monitor and to be written to a disk file. For example, the command line

```
xmit.u  | tee log.x127
```

causes the XMIT.U program to direct all output both to the terminal and to a file called LOG.X127.

4.1.3 On-Line Data Extraction

As stated previously, the design of the ARPANET measurements followed an active approach; i.e., in each host computer, the end user activities and the input function of the associated Interface Monitor were merged in a single application program. At the source (transmitting) user/system interface, a program called XMIT performed all end user activities (i.e., issued all system calls) required during the monitored data communication sessions. This program also recorded the nature and time of occurrence of each performance-significant event (system call or system response) at the source interface. A companion program, called RECV, performed the corresponding functions at the destination (receiving) user/system interface.

In the context of the ARPANET experiment, a test consisted of a single run of companion XMIT and RECV programs. Off-line (post-test) processing of the raw data extracted during a test produced a set of four standard ASCII character files for input to the Performance Assessment program; these files are referred to in proposed FED STD 1043 as a performance data batch. The on-line Data Extraction software was designed to perform either of two types of tests, depending on whether the test was intended to obtain data for calculating access and disengagement parameters, on the one hand, or user

information transfer parameters, on the other. A normal access/disengagement test consisted of a sequence of 160 data communication sessions, with only one user information block of 64 bytes transferred in each session. A normal user information transfer test consisted of a single data communication session in which 20 blocks of user information, of 512 bytes each, were transferred.

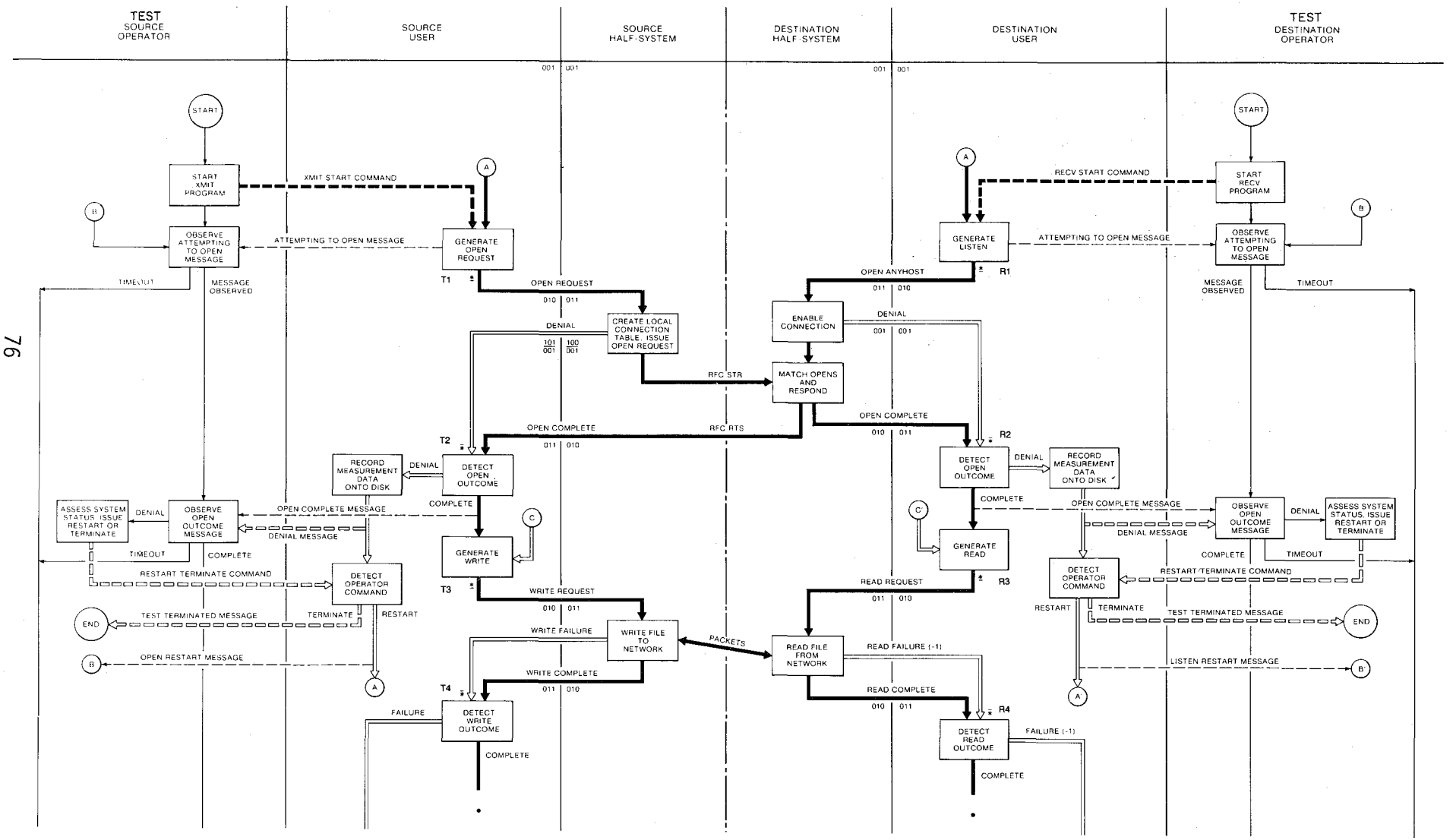
The ARPANET measurements were designed so that, in any given test, either the NBS/ICST host or the NTIA/ITS host could serve as the source host. This design feature was implemented by including appropriate versions of both XMIT and RECV in the Data Extraction software at each site. Except for minor variations due to machine and system software differences between the two hosts, the NBS and ITS versions of these programs were the same. The compiled forms of XMIT and RECV occupied about 7800 and 7400 bytes of storage, respectively. Including ARPANET variable arrays and extracted data buffers, the executable software for a test required about 20,000 bytes of memory in each host.

An overview of the on-line (real-time) Data Extraction scheme for the ARPANET experiment is provided by the session profile of Figure 25. This shows the nature and sequence of the basic functions performed by participating entities during a typical ARPANET test, as well as the performance-significant events resulting from those functions.

Participating entities are indicated at the top of the diagram. As described previously, the source and destination half-systems collectively consist of the ARPA subnet, the two host computer operating systems, the Network Control Programs, and the Local Host Interfaces. The source and destination users are, respectively, the XMIT and RECV programs. Complete listings of these programs are contained in Appendix B, and simplified flow charts are shown in Figure B-1. The source and destination test operators represent the human operator (located in Boulder) who initiated and monitored the test, and the command files which simulated an operator typing commands. In all tests, the source user was the originating user (i.e., the source user initiated all access requests).

Rectangular boxes indicate functions performed by participating entities. Large circles denote entry and exit points, and small circles denote interconnect points.

Lines indicate actual or potential movement of information or control from one function to another. Six types of lines are shown:


TEST SOURCE OPERATOR
SOURCE USER
SOURCE HALF-SYSTEM
DESTINATION HALF-SYSTEM
DESTINATION USER
TEST DESTINATION OPERATOR
START
START XMIT PROGRAM
XMIT START COMMAND
OBSERVE ATTEMPTING TO OPEN MESSAGE
ATTEMPTING TO OPEN MESSAGE
GENERATE OPEN REQUEST
TIMEOUT
MESSAGE OBSERVED
T1
OPEN REQUEST
CREATE LOCAL CONNECTION TABLE, ISSUE OPEN REQUEST
DENIAL
RFC STR
ENABLE CONNECTION
MATCH OPENS AND RESPOND
OPEN ANYHOST
GENERATE LISTEN
R1
RECV START COMMAND
START RECV PROGRAM
OPEN COMPLETE
RFC RTS
T2
DETECT OPEN OUTCOME
RECORD MEASUREMENT DATA ONTO DISK
COMPLETE
OBSERVE OPEN OUTCOME MESSAGE
OPEN COMPLETE MESSAGE
DENIAL MESSAGE
ASSESS SYSTEM STATUS, ISSUE RESTART OR TERMINATE
RESTART TERMINATE COMMAND
DETECT OPERATOR COMMAND
TERMINATE
RESTART
TEST TERMINATED MESSAGE
END
OPEN RESTART MESSAGE
GENERATE WRITE
T3
WRITE REQUEST
WRITE FILE TO NETWORK
WRITE FAILURE
WRITE COMPLETE
T4
DETECT WRITE OUTCOME
FAILURE
PACKETS
READ FILE FROM NETWORK
READ FAILURE (-1)
READ COMPLETE
R2
R3
R4
GENERATE READ
DETECT READ OUTCOME
FAILURE (-1)
LISTEN RESTART MESSAGE

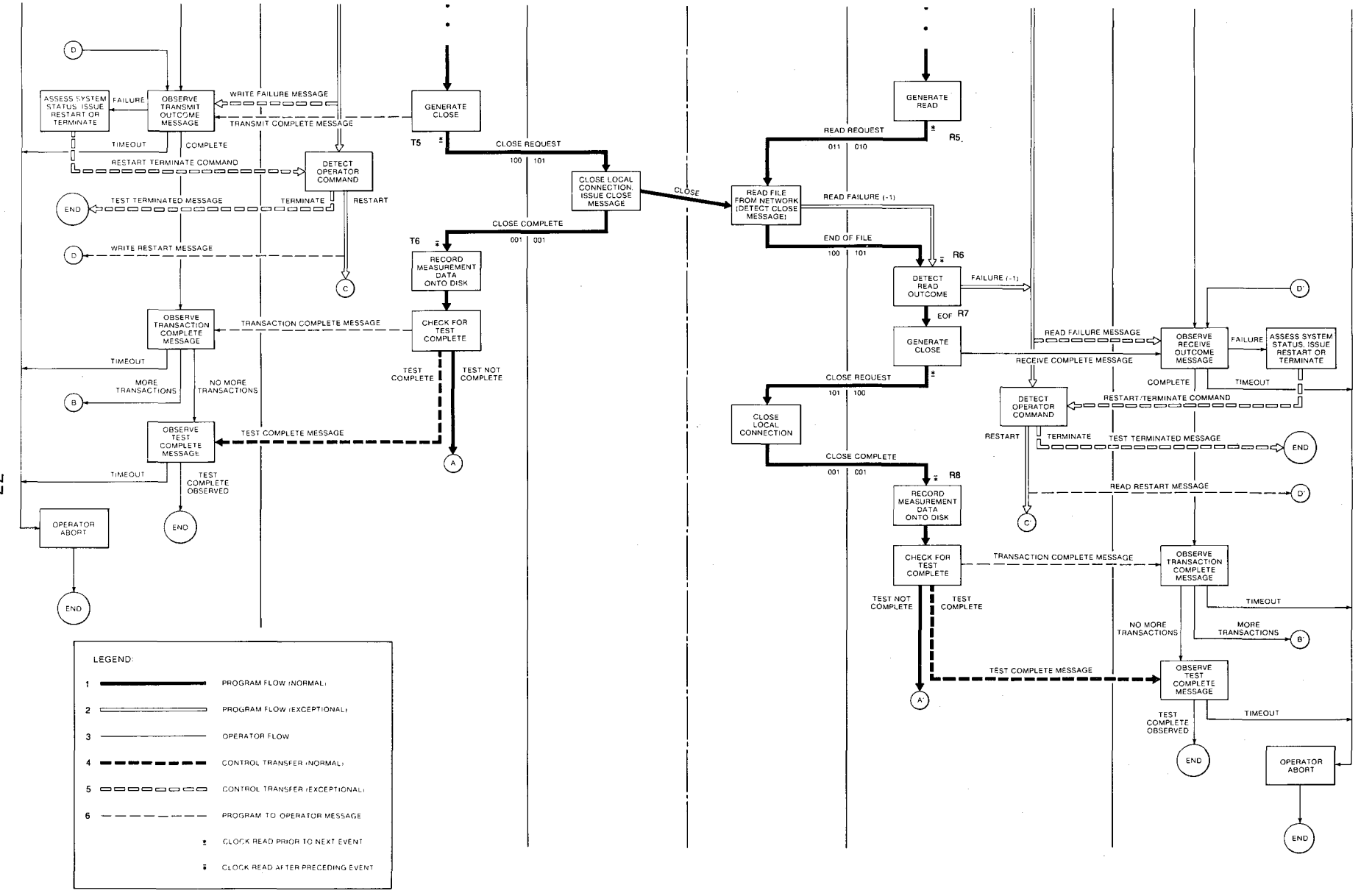


Figure 25. Session profile.

o Solid Line - Information Flow
  Heavy (Type 1) - Normal Program Flow
  Double (Type 2) - Exceptional Program Flow
  Light (Type 3) - Operator Flow

o Dashed Line - Control Flow
  Heavy (Type 4) - Normal Control Transfer
  Double (Type 5) - Exceptional Control Transfer
  Light (Type 6) - Program to Operator Message

Captions accompanying a line describe the nature of the information communicated. During a normal test all paths on the session profile are followed except for the double lines (Types 2 and 5). A line crossing a user/system interface in Figure 25 corresponds to a performance-significant event. The three-digit binary representation of the transaction state immediately following such an event is shown in the session profile for each relevant entity.

Clocks readings used to obtain event times are shown by a small symbol at either the beginning or ending of the lines where the event occurred. The times are labelled T1, T2, ..., T6 for the source or transmit site, and R1, R2, ..., R8 for the destination or receive site. The odd numbered clock readings, i.e., T1, T3, T5 and R1, R3, R5, R7, give times which precede an event, and the even numbered clock readings, i.e., T2, T4, T6 and R2, R4, R6, R8, give times which follow an event. In implementing the time-tagging process with the UNIX Operating System, the clock readings required subsequent correction to obtain the event times. The correction factor was different depending on whether the clock readings preceded or followed the associated event. Details of the correction procedures are described in Appendix D. This correction would not have been necessary had a clock routine been interfaced into the operating system; however, the clock was read from the user program. Including the clock routines in the user program makes the software less machine dependent and more transportable to other systems.

Both XMIT and RECV began by reading test descriptors from "preface" files created prior to the test, then wrote header information in the appropriate output files. In addition, XMIT generated and stored sequence of 10240 bytes of pseudorandom data, which served as the source user information transmitted during the test.

The RECV program then displayed an "attempting open" message to the operator, read the satellite clock, and generated a "listen" by issuing an OPEN command to any host on the network (with a particular set of parameters,

including a specific connection number). Issuance of the OPEN command corresponded to the FED STD 1043 reference event "nonoriginating user commitment." Control was then passed from RECV to the destination NCP, which issued an "enable connection" and waited for the source to act.

After a 10-second delay (following generation of the pseudorandom data), XMIT performed the analogous sequence of procedures at the source site, i.e., it displayed an "attempting open" message to the operator, read the satellite clock, issued an OPEN command to the network with an appropriate set of parameters, and passed control to the source NCP. This OPEN command corresponded to the FED STD 1043 reference event "access request." The 10-second delay normally assured that the destination was "listening" before XMIT passed control to the source NCP. When control was passed to the source NCP, it issued a "request for connection from sender to receiver" (RFC-STR) through the network. If the source request parameters matched the destination listen parameters, then a "socket" was established and the destination sent the "request for connection from the receiver to sender" (RFC-RTS) back through the network to the source. A parameter was passed from each NCP through the local host operating system to the user/monitor programs which detected the OPEN outcome. A successful OPEN produced an "opened" message to each operator. After issuance of this message, both XMIT and RECV again read the satellite clock.

After a successful OPEN, XMIT wrote a block of user information to the network from the pseudorandom data file and passed control to the NCP. The WRITE command issued by XMIT corresponded to the two FED STD 1043 reference events "start of first block input to system" and "start of block transfer." At the destination site, RECV issued a network READ with three times the expected number of bytes, and passed control to the NCP. Thus, if extra bits were inserted, or even if an entire block were duplicated, the extra data would have been detected. After a block was sent, the source NCP returned control to XMIT, which read the clock, and in the case of a user information transfer test, immediately wrote another user information block from the pseudorandom data buffer (with the appropriate incrementing of the pointer to the next part of the buffer). Upon receipt of a user information block from the NCP, RECV stored this data in a buffer, advanced the pointer by the number bytes actually received (not the number expected) and read the clock. The completion of the network READ, issued earlier by RECV, corresponded to the FED STD 1043 reference event "end of block transfer". The RECV program was

designed to read from the network up to four blocks more than the number expected in the type of test being conducted.

After the last user information block had been sent, XMIT generated a CLOSE command. The issuance of this command corresponded to the FED STD 1043 reference event "disengagement request" for the source user, and initiated the disengagement function for both users. The clock was read at the source site, and a message "xmit complete" issued to the operator. Control was briefly passed to the source NCP, which issued a network CLOSE message. A CLOSE COMPLETE response was then returned to XMIT. This event corresponded to the reference event "disengagement confirmation" for the source user. Event times and transaction state codes were recorded to disk files, and the operator was informed by the "transaction complete" message that the data were preserved.

Meanwhile, RECV continued to read data from the network, store the data in a buffer, and record the number of bytes received. When the CLOSE message reached the destination NCP, the local operating system relayed the information to RECV with an end of file (EOF) indicator. The clock was read, a "read complete" message issued to the operator, and a CLOSE generated for the network. Control was briefly passed to the NCP, and a CLOSE COMPLETE response was then returned to RECV. This event corresponded to "disengagement confirmation" for the destination user. Clock times were read before and after the CLOSE. Event times and transaction state codes were recorded to disk, and the message "transaction complete" was issued to the operator. This completed the test if it were a user information transfer test; however, for an access/disengagement test the entire process was repeated for a total of 160 times. A 10-second pause at the source occurred before each network OPEN. When the entire test was completed, the operator on each end was issued a "test completed" message.

Not all tests ran exactly as described above. Exceptional cases observed were:

1. Denial or failure to open a connection to the distant site,
2. A WRITE or READ failure on a block of data, and
3. A READ failure at the receive site for a network CLOSE from the transmit site.

In the cases of an outright failure or denial, the operator was prompted by the message:

"open denied, a(bort) or c(ontinue)" or
"write error, a(bort) or c(ontinue)" or
"read error, a(bort) or c(ontinue)" ,

whichever was appropriate. The operator could then choose to type "a" or "c" for abort or continuation of the test. Sometimes, however, there would not be an outright failure, and the program would "hang" or be suspended in the NCP. Since the timeout capability was not implemented in either the NBS or ITS NCP, the software would remain indefinitely in this state. The operator would then have to decide when to terminate the test.

Both types of tests, user information transfer and access/disengagement, followed the same sequence of events shown in the session profile and used the same XMIT and RECV programs. However, three constants were defined to cause the test to be executed differently for each test type. Table 1 shows the number of accesses per test, the number of blocks transmitted per access, and the number of bytes sent per block.

Table 1. Constants for the Two Types of Tests

Type of Test	Accesses Per Test	Blocks Per Access	Bytes Per Block
Block Transfer	1	20	512
Access/Disengagement	160	1	64

In each type of test, the total number of bytes sent was 10240. When the test was complete, the received data file should have contained exactly the same psuedorandom sequence of 10240 bytes as the transmitted file, provided there were no unusual events. After each test, a copy of the transmitted user information was generated at the destination site and compared with the received data.

In addition to the execution of companion XMIT and RECV programs, an ARPANET test included the recording of several items of information to document conditions surrounding the specific test. This information was written in a "log file" which is described in Appendix F. To simplify operator procedures and minimize the chances of error, each test was executed under the control of a pair of command files named RUNXMIT and RUNRECV. These files performed the following procedures:

1. Displayed the name of program run and the name of the log file containing the narrative.

2. Stored the system date, user names, and all operating process identifiers in the log file.

3. Compared and stored the system and satellite clock times in the log file.

4. Stored the name of the test program and test number in the log file.

5. Directed all standard program output from the test to the operator and to the log file, and ran the test.

6. Compared and stored in the log file the times from the system and satellite clocks after the test.

7. Compared the received data file with the transmitted data file, and stored the results in the log file (RUNRECV file only).

8. Incremented the test number in the preface files in readiness for the next test.

Figure 26 shows a bubble chart of the data flow for the entire experiment. Whenever the data are in a clearly identifiable form between processes, they are shown in a bubble. The lines between bubbles represent data transformation functions, which are identified on the right of the diagram. The flow of the experiment and data is from top to bottom, with initial or raw data nearer the top and processed or refined data nearer the bottom. The on-line operations occur above the short dashed line and correspond to the data extraction shown by the session profile in Figure 25.

The XMIT and RECV programs each read a short preface file (PREFACE.X and PREFACE.R, respectively) to obtain information about test number, source and destination, and type of test. The two programs each created three files in which all the extracted data from the test were stored. The three files created by XMIT were OVERHEAD.X, HISTORY.X, AND DATA.X. The files created by RECV had the same names, except the .X suffix was replaced by .R. The HISTORY and OVERHEAD files began with the preface header information for the test. The remainder of information in the OVERHEAD files consisted of the event number, the transaction state code, and the time the event took place. The balance of the information in the history files consisted of the block number, the number of bytes transmitted (or received) in a block, and the start and end time for that block transmission (or reception). A detailed description of all the files is presented in Appendix C.

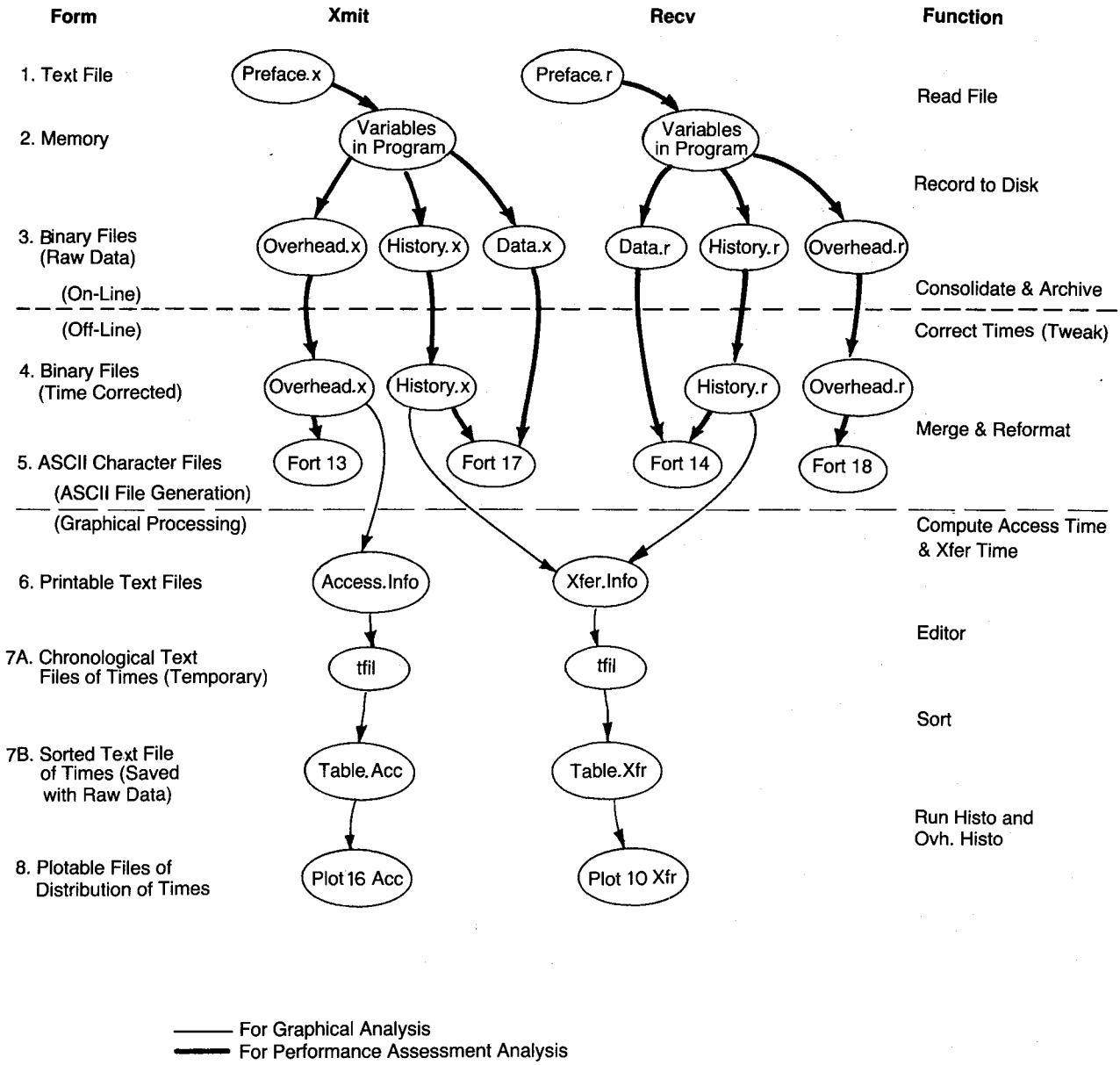


Figure 26. Data flow diagram.

The last step in the on-line phase of a test was to consolidate all the files at the Boulder site for the post-test processing and analysis. This was done using the network File Transfer Program (FTP). An elaborate system of bookkeeping and checking was established to guard against introducing errors or losing files in transfer. The files at ITS were moved into a data subdirectory with the test number appended to the file name. Then the files were transferred from NBS to the same ITS data directory. A command file called MOVEX or MOVER did the local moving, but all files were transferred on a one-by-one basis. Then a command file called GETDTG was run to obtain the date-time-group (DTG) from the beginning of the HISTORY.X file. Another command file called STOREM was used to move all the files from a single test to the archive directory. The archive directory was chosen to be the three-letter abbreviation of the month (i.e., dec for December). Each file was stored with the date-time-group as the beginning of its name. This made it easier to keep track of the data later. The files were not actually moved by UNIX once they were stored on the disk; only the names and pointers from the various directories were changed. For this reason the bubble chart does not show any of the moves and name changes. The system used to check against introduction of errors when the files were transferred across the network is discussed in detail in Section 5.1.2. The files shown in Figure 26 in line 3 are binary files; after consolidation into the monthly directory with the date-time-group, they were termed the "archive" or "raw data" files for the test.

4.1.4 Off-Line Data Extraction Processing

The processing necessary to generate the ASCII character files for the analysis programs was done automatically by a linked series of command files called DOIT.U or DOIT.O, depending on whether the user information transfer test or the access/disengagement test was being processed. All the command files are listed and explained more fully in Appendix G. The DOIT command files began by printing a comparison between the transmitted reference data file and the raw data received file. Then any temporary files that were on the disk from the previous processing were removed. The command file then called two more command files, named PROCESS_QIK and PROCESS FOR, each twice - once to process the receive files and once to process the transmit files. The PROCESS_QIK command file printed out the log files and provided the satellite clock time corrections for each of the sites as explained in detail in

Appendix D. The names of the programs that provided time corrections were TWEAK.01200, TWEAK.02400, TWEAK.H1200, and TWEAK.H2400. The programs read the raw OVERHEAD and HISTORY files from the month subdirectory, provided the time correction factor, and wrote them back to the data directory with the names OVERHEAD.X, OVERHEAD.R, HISTORY.X., and HISTORY.R. A byproduct of these four programs were files called O.INFO.X, O.INFO.R, H.INFO.X and H.INFO.R. These four text files contain a record of all the events or blocks and their associated times in an easy-to-read format. These files show the raw times obtained from the satellite clock prior to time-correction.

The two history information files (H.INFO.X and .R) were printed. After the times were corrected, the next command file (PROCESS_FOR) was called to run the MERGE and REFORM programs.

The MERGE program (written in "C") took the DATA and HISTORY file for a given site and interleaved the data and block transmission times into an ASCII readable file. The REFORM program reformatted the OVERHEAD files into ASCII character readable files also. The merged files were temporarily named FORT90, and the reformatted OVERHEAD files were temporarily named FORT80. Before leaving the command file, the FORT90 file was renamed FORT17 or FORT18 and the FORT80 file was renamed FORT14 or FORT15 for the transmit and receive cases, respectively.

The final step in the processing, which is not shown in the bubble chart, was to move the four FORT files to the /DISK2/ASSESS directory and add a date-time prefix to each file name. The four files are discussed in the next section.

4.2 Data Files

The second major element of the proposed FED STD 1043 performance measurement system is a set of four standard-formatted performance data files. These files serve to link the two system-dependent Interface Monitors with the single system-independent Performance Assessment program. Their individual formats and contents were summarized earlier, in Section 2.4.2. Each file is fully described in Section A.2 of the measurement standard.

During the course of the ARPA network measurements project, each of the four performance data files was implemented as specified in proposed FED STD 1043. The files were written by the "C" language programs MERGE and REFORM at the completion of each Data Extraction run, as discussed in Section 4.1. The files were read by the FORTRAN program PROLOG at the beginning of each

Performance Assessment run, as discussed in Section 4.3. With a minor exception, discussed later, the file formats used in each case were identical to those specified in the draft measurement standard. No significant problems were encountered in using the files as a medium for transferring the measurement data between the "C" language Data Extraction and FORTRAN Performance Assessment programs.

4.3 Performance Assessment Element

This subsection describes the implementation of the third major FED STD 1043 performance measurement element specified in Figure 11 - the Performance Assessment element. The subsection is divided into two parts. The first describes implementation of the standard FORTRAN Performance Assessment program, with a focus on differences between the original FED STD 1043 program design and that ultimately used to reduce the ARPA network data. The second part describes the design of various supplemental data reduction programs which were used to obtain additional performance statistics such as delay histograms.

4.3.1 Standard FORTRAN Performance Assessment Program

Section 2.4.3 described the design of the Performance Assessment program as originally specified in proposed Federal Standard 1043. In the process of implementing that program in this experiment, several changes in the program structure were adopted. The effect of these changes was to fragment the overall ASSESS program into three separate programs, which are executed sequentially: PROLOG, which checks the format and content of input files and prepares them for processing; ANALYZ, which reduces the extracted performance data to a set of performance parameter values; and EPILOG, which prints the calculated parameter values.

The purpose of fragmenting ASSESS into three stand-alone programs was to reduce the computer memory required to accomplish the performance assessment function. The original ASSESS program comprised over 50,000 bytes of executable code. A user program of that size would not be directly executable on some smaller host computer systems. The PROLOG, ANALYZ, and EPILOG programs are substantially shorter and will thus be more widely usable.

The major change this fragmentation required was the addition of utility subroutines which read and write FORTRAN files linking the three programs. These utility subroutines were not required when ASSESS was a single program,

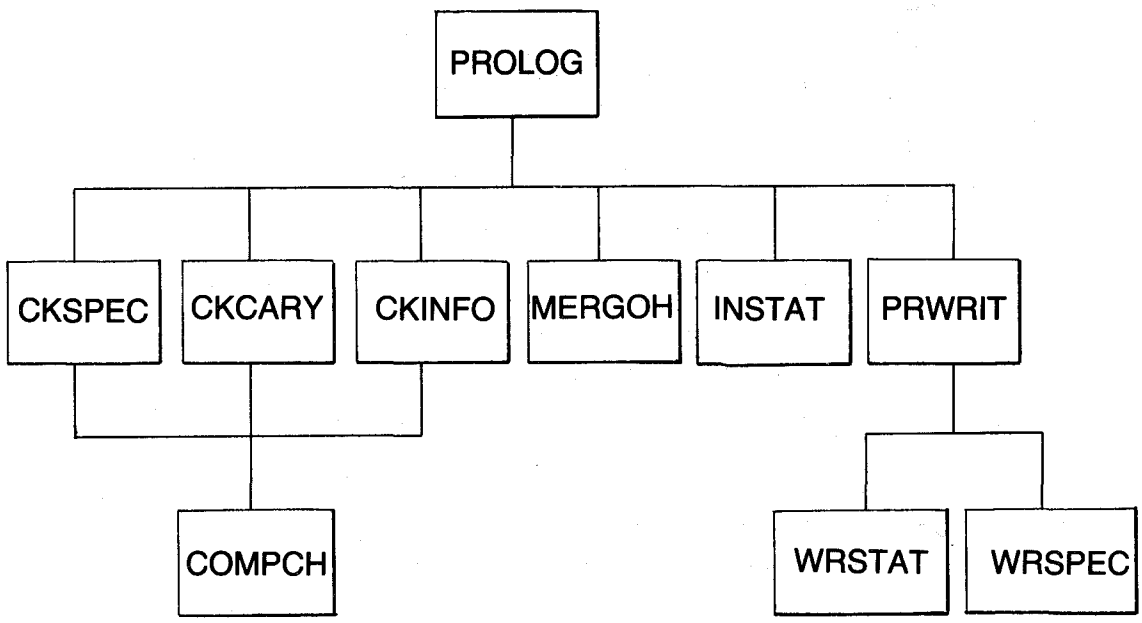
since the equivalent linkages were made via COMMON blocks. Four program linkage subroutines were added, consisting of two symmetrical pairs: one pair which write and read the FORTRAN files linking PROLOG and ANALYZ; and another pair which write and read the files linking ANALYZ and EPILOG.

Figures 27 and 28 illustrate the structure of the revised Performance Assessment programs. The major routines under the new PROLOG program are shown in Figure 27, and may be briefly summarized as follows: CKSPEC, CKCARY, CKINFO, COMPCH perform format and validity checks on the information input to PROLOG. This information includes run specifications defined by the test operator (e.g., timeout values, data correlator window size), carryover files in multi-run performance assessments, and performance measurement information in the preface records of the overhead and user information files (e.g., batch numbers). The MERGOH routine combines the data from the separate source and destination overhead information files to create a single chronological merged overhead information file. The INSTAT routine initializes the access, transfer, and disengagement sample statistics. The PRWRIT routine and its subroutines create the program linkage files in which data are passed from PROLOG to ANALYZ.

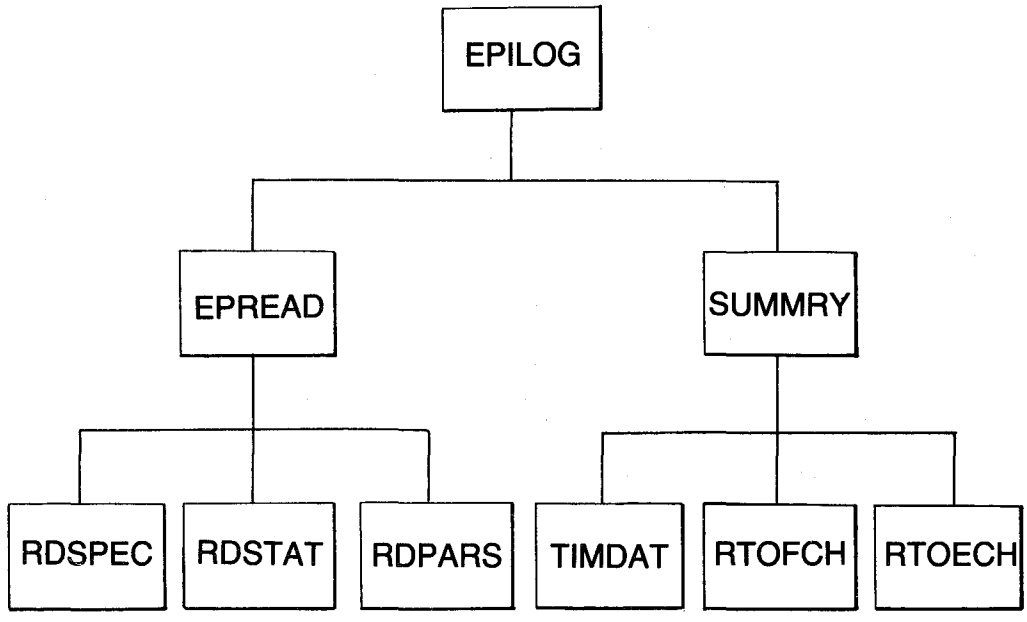
The major routines under the new ANALYZ program are shown in Figure 28. Briefly, ANREAD and its subroutines read the program linkage files in which data are passed from PROLOG to ANALYZ. The ACCESS, TRANSF, DISENG, and SECOND routines and their associated subroutines calculate values for the access, user information transfer, disengagement, and secondary performance parameters, respectively. The ANWRIT routine and its subroutines create the program linkage files in which data are passed from ANALYZ to EPILOG.

The two most important subroutines within the ANALYZ program are BITCOR and ANCILL. The BITCOR subroutine matches transmitted and received user information bits and identifies incorrect, undelivered, and extra bits. The ANCILL subroutine calculates the user fractions of access, user information transfer, and disengagement time intervals as requested by ACCESS, TRANSF, and DISENG. These fractions are used in calculating ancillary parameter values, and in determining "responsibility" for timeout failures.

The major subroutines under the new EPILOG program are shown in Figure 27. Briefly, EPREAD and its subroutines read the program linkage files in which data are passed from ANALYZ to EPILOG. The SUMMRY routine and its subroutines print and store the results of the performance assessment run.



a. PROLOG Structure



b. EPILOG Structure

Figure 27. PROLOG and EPILOG program structures.

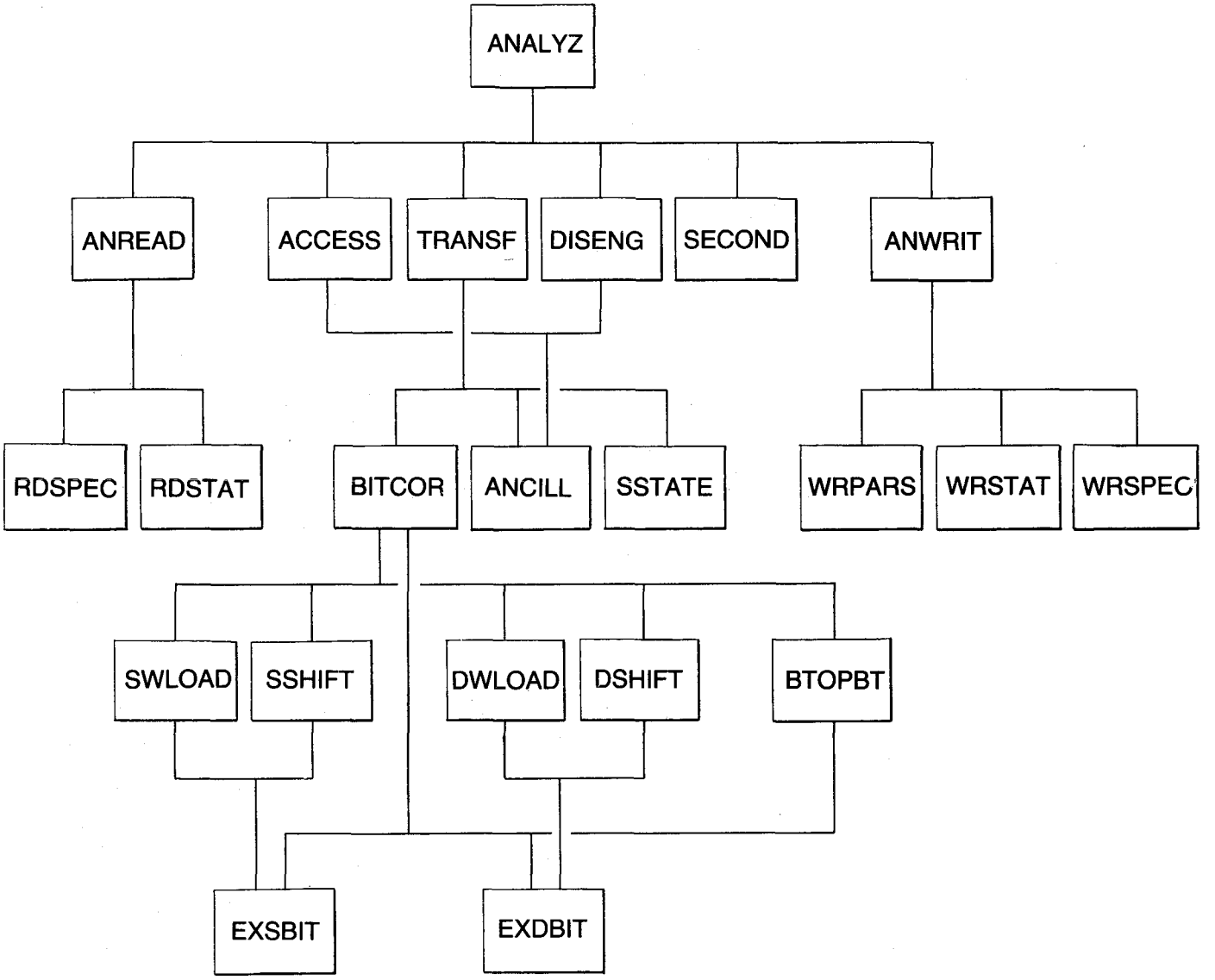


Figure 28. ANALYZ program structure.

The input/output files used by the revised Performance Assessment programs are similar to those used by ASSESS, except that two new files (the PROLOG-to-ANALYZ and ANALYZ-to-EPILOG program linkage files) have been added. Other, less significant changes to the Performance Assessment programs are not documented here. All program changes made during the trial implementation period will be incorporated in a revised version of the draft 1043 standard.

4.3.2 Supplemental Data Reduction

The ARPANET experiment was designed to provide a more complete description of access and block transfer time distributions than is required by proposed FED STD 1043. For an operator-specified set of tests (of the same type), supplemental data processing procedures produced the following output: (1) the minimum, maximum, mean, and standard deviation of the observed access (or block transfer) times; and (2) a histogram showing the distribution of access (or block transfer) times. Software for the supplemental data processing includes both "C" language and FORTRAN routines. The software was designed to deal specifically with the ARPANET measurement data, and is generally not "system independent."

Key preliminary procedures in the supplemental data processing were performed by the same DOIT command file that generated the four standard ASCII character output files specified by proposed FED STD 1043. For each test, the DOIT command file created a text file containing access or block transfer times sorted according to increasing performance time. The text files were used later as input to separate data processing procedures that generated statistics and histograms for aggregated tests. The DOIT command file also generated a file consisting of x-y coordinates of the vector trace for a histogram of performance times for each individual test.

The following discussion outlines, in general terms, the procedures executed by the DOIT command files during the supplemental data processing. Because the procedures are significantly different for the two types of tests, they will be treated separately.

Access Time Processing

The DOIT.O command file executed another command file called GRAPH.ACC, which ran a program called ACCESSTIME. This program read the time corrected OVERHEAD.X data file and produced an output text file named ACCESS.INFO, in the temporary directory. This text file, which was very large, contained the

same data as that in O.INFO.X, plus some very important additions. These additions were the differences in time between adjacent events and the differences between alternate events. The file name from whence the raw data originated and the session number (1-160) were also added.

A short command file named TAB.ACC edited the large ACCESS.INFO file and reduced it from about 976 lines to 160 lines, with each line containing the time difference corresponding to the FED STD 1033 definition of access time. This shortened temporary file, called TFIL, was immediately sorted by TAB.ACC into a table of ascending access times. This table was called TABLE.ACC and was stored with the raw data in the monthly directory.

The command file GRAPH.ACC then ran the program OVH.HISTO, which read the TABLE.ACC file and generated a text file consisting of x-y coordinates of the vector trace for a histogram of access times. This file of x-y coordinates was called PLOT16ACC, where the 16 denoted the number of class intervals in the histogram between the upper and lower limits. The command file also moved, sorted, and aggregated some of these data files, but it had no impact on the data flow shown in Figure 26 and need not be discussed further.

Block Transfer Time Processing

The DOIT.U command file executed another command file named GRAPH.XFR. GRAPH.XFR ran a program called XFERTIME which read the corrected times from both the HISTORY.X and HISTORY.R files. The output text file was called XFER.INFO, and contained the usual preface information (from both transmit and receive files), the start time of the block transfer at the source, the end time of the block transfer at the destination, and the difference between the two times. In addition, the number of bytes transmitted and block numbers were shown. At the end of the text file the block and bit transfer rates and transmission times were given. In a manner similar to that for access time processing, another short command file called TAB.XFR was executed to edit the unnecessary lines from the XFER.INFO file and produce a file (TFIL) of only those lines which contained a block transfer time. The short command file sorted TFIL into the TABLE.XFR file of ascending block transfer times and returned control to the GRAPH.XFR command file. Processing then paralleled the access time processing, with a HISTO program which read the TABLE.XFR file and produced a PLOT10XFR file of x-y coordinates. The TABLE.XFR was stored with the raw data and the temporary files were moved, aggregated, and sorted.

Plotting of the Histograms

The final step in obtaining the histograms for individual tests involved listing the plot files to an intelligent graphic CRT. The operator selected appropriate histogram format parameters from an autoplot menu displayed on the CRT. The terminal then read the vector trace of the x-y coordinates and plotted the histogram on its screen (soft copy). It was possible for the operator to clear the screen, modify the axes or plot parameters, and replot the histogram quickly (usually in less than 30 seconds) until the desired aesthetic result was produced. Labels and annotations concerning the test or tests were then added to this soft copy. Finally, a few CRT keystrokes transferred the soft copy graph to an associated dot matrix graphic printer. This produced the final hard copy graph and completed the supplemental data reduction.

4.4 Statistical Design and Analysis

As discussed in Section 2.4.4, the Statistical Design and Analysis procedures specified in proposed FED STD 1043 deal with three general subjects: qualitative test design, sample size determination, and test data analysis. The qualitative design and sample size determination procedures are normally accomplished during the preliminary test planning. Their application in the ARPANET experiment is summarized below, following a brief tutorial on statistical sampling fundamentals. The data analysis procedures are normally accomplished after a test has been completed. Selected examples of the post-test analysis performed on the ARPANET data are presented with the measurement results in Section 6.

4.4.1 Sampling Fundamentals

The following paragraphs define a number of statistical concepts which are important in data communication performance measurement. Each concept is illustrated using the measurement of block transfer performance as an example.

In statistics, it is basic to distinguish between a population and a sample from that population. The population is the complete set of items of interest: for example, all blocks transferred between a pair of end users. In a complex measurement, the items in a population may be multivalued (block transfer outcomes and transfer times, for example). Often, the items are not fixed in number, but continue to come into existence indefinitely in time; and thus, the size of a population may be theoretically infinite.

A population is normally characterized by a number of statistical descriptors called parameters. Each parameter is a specific function of the numerical values which characterize the population items. Parameters are chosen on the basis of experimental objectives. Block Error Probability and Block Transfer Time are examples.

A sample is a finite part (or subset) of a population, which is used in estimating the population parameters. The process of sampling makes the experimental determination of population parameters practical by restricting the number of items which must be observed. In the measurement of block transfer performance, a sample might consist, for example, of 10,000 consecutive transferred blocks.

In order to draw valid inferences about a population from a sample, the sample must be drawn with some element of randomness or chance. A sample whose items are chosen in a random manner is called a random sample. In the measurement of block transfer performance, the collection of a random sample might be accomplished by observing block transfer attempts at many different, arbitrarily chosen times.

The value of each population parameter is estimated by a mathematical function of the sample. Any such function is called an estimator; and the value it produces is called a statistic or (more simply) an estimate. For example, the value of the performance parameter Block Transfer Time can be estimated by the mean Block Transfer Time calculated over a sample. In general, an estimator need not be the same function of the sample as the corresponding parameter is of the population; for example, the sample median might be a better estimator of Block Transfer Time than the sample mean in some cases.

The standard performance parameters defined in Interim FED STD 1033 are of four basic types: probabilities (e.g., Block Error Probability); mean time intervals (e.g., Block Transfer Time); time rates (e.g., Bit Transfer Rate); and rate efficiencies (e.g., Bit Rate Efficiency). The probability parameters are estimated by the relative frequency, in the sample, of the particular (failure) outcome in question. The mean time parameters are estimated by sample means. The rate and rate efficiency parameters are estimated by the individual sample values, or by mean rate values calculated over several samples. The ancillary parameters, which are ratios of mean times, may be regarded as a fifth parameter type. They are estimated by ratios of the appropriate sample means.

The estimation of population parameters from a finite sample obviously involves some uncertainty. The goodness of a parameter estimate is measured by its precision, i.e., the length (or half-length) of the confidence interval for the estimate. A confidence interval is a range of values about a parameter estimate within which the "true" value of the parameter can, with a stated percent confidence, be expected to lie. The end points of a confidence interval are called confidence limits. The stated percent confidence is called the confidence level, and expresses the likelihood that the confidence interval calculated from a sample will include the "true" value of the parameter. For example, a confidence level of 95% implies that the confidence limits will include the "true" parameter value in about 95 out of 100 experiments.

The precision of an estimate is high when the confidence interval (for a given confidence level) is small, and vice versa. Specifically, the precision of an estimate is inversely proportional to the square root of the sample size. Thus, one can calculate either the required sample size or the precision of an estimate if the other is given. FED STD 1043 defines the details of these calculations and gives examples.

In some measurement applications, it is more practical to deal with the relative precision of an estimate than with its absolute precision. The relative precision of a parameter estimate is the half-length of the confidence interval for the estimate divided by the estimate itself. Relative precision is normally expressed in percent. As an illustration of the meaning of relative precision, assume 10 independent Block Errors are observed in a sample of 10,000 transferred blocks. Then the Block Error Probability estimate is 10^{-3}. The upper and lower 95% confidence limits about that estimate, from Crow (1974), are roughly 1.8×10^{-3} and 4.8×10^{-4}. Half the difference between these confidence limits, i.e., the half-length of the confidence interval, is $(1/2)(18 - 4.8)10^{-4} = 6.6 \times 10^{-4}$; and the relative precision of the Bit Error Probability estimate is $(6.6 \times 10^{-4})/(10^{-3})(100) = 66\%$. This value can be determined directly by inspection of Figure 19.

Classical statistical methods are based on the assumption that successive observations are independent, i.e., unrelated in value to each other. That assumption is unrealistic for observations taken in rapid succession on a data communication system. Dependence between observations is commonly measured by correlation coefficients, or, in the case of observations on the same variable, by autocorrelation coefficients. An autocorrelation coefficient p_k

is a number, in the range $-1 \leq p_k \leq 1$, which expresses the degree of dependence or linkage between two observed values of a variable. Positive autocorrelation indicates that the observations are more similar than pure chance would predict, and vice versa. The lag (k) of an autocorrelation coefficient is the separation, in a sequence of values, between the pairs of values whose autocorrelation is being expressed. If pairs of observations separated by lag k are independent, then $p_k = 0$.

The collection of autocorrelation coefficients for k = 1, 2, 3, . . . of a variable observed at discrete times is called its autocorrelation function. This function may be complicated, but as a first approximation it is often represented according to the "Markov model", in the form $p_k = p_1^k$. In that special case, we can characterize the dependence between observations by the one parameter p_1. This is done in FED STD 1043 and herein. If p_1 is positive, there is positive dependence between successive sample values, and the sample contains less information than if p_1 were zero. Thus, the sample size required to achieve a given measurement precision is larger in the case of positive dependence. Calculations presented in FED STD 1043 demonstrate, for example, that a 0.5 autocorrelation between successive access time values triples the sample size required to estimate the mean with any given precision.

4.4.2 Qualitative Test Design

We now briefly summarize the application of the proposed FED STD 1043 qualitative test design guidelines in the ARPANET experiment. Those guidelines require, first, that all test variables known to significantly influence the measured performance be clearly specified (and, where appropriate, specifically tested); and second, that the testing pattern be varied so as to "average out", and if possible detect, the effects of other unknown test variables. The intent of these guidelines is to make the measured values more informative, more reproducible, and more comparable between services.

In preliminary design of the ARPANET experiment, the following test variables were identified as clearly influencing the measured performance:

1. Identity and characteristics of the end users.
2. Sequence of interface events during a typical data communication session.

3. Time of test (within day and week).

4. Priority of the user programs relative to that of other computer programs executing in the hosts.

The first variable influences measured performance in two ways: first, by defining the location and nature of the end user interfaces, which are also the system boundaries and the performance measurement points; and second, by defining the frequency and duration of user delays during a data communication session. The end users were completely specified in the ARPANET experiment via the XMIT and RECV program designs. Those programs are listed in Appendix B.

The second test variable, the interface event sequence, influences observed performance primarily in its effect on user delays. As an example, the measured value for the parameter Access Time can be strongly influenced by the order in which the originating and nonoriginating user Open requests are issued: the former request starts the access function, but the latter is required to complete it. The expected interface event sequence was defined in the ARPANET experiment via the session profile diagram of Figure 25. The most likely alternative event sequences were also identified in that figure.

The third variable, the time of test, reflects the aggregate effect of three extremely important test conditions: the utilizations of the source and destination host computers, and the subnetwork traffic. None of these conditions could be completely controlled or directly measured during the ARPANET experiment, although the test log files gave rather good insight into the former two. The effects of time of test on performance were reflected in the test design in two ways: (1) by random selection of test start times; and (2) by the division of all tests into "peak hours" and "off hours" categories, as described in Section 5. The exact date and time of each test was, of course, recorded with the measurement data.

The final test variable identified in the preliminary ARPANET experiment design was the priority of the end user programs. This variable directly affects user performance time when a host computer is shared between several programs, since a user program may be interrupted by any concurrently executing program of higher priority. The effect of user program priority on measured performance was reflected in the ARPANET test design by conducting tests with both very high and normal program priorities. Since user program priority influences only the performance of the users, it can, of course, be "factored out" by use of the ancillary performance parameters.

In order to detect other possible test variables of significance, a large number of preliminary ARPANET measurements were conducted with selected changes in hardware and software configuration, operator procedures, and event timing. These preliminary measurements revealed one test variable that had not been considered in the preliminary design: the direction of data transfer. It was initially anticipated that the system's performance in transferring data from Boulder to Gaithersburg would be about the same as its performance in transferring data from Gaithersburg to Boulder, since the two paths appear symmetrical. Preliminary measurements suggested that performance on the two paths might be significantly different, perhaps because of the much heavier utilization of the NBS/ICST host. This variable was reflected in a revised test design via explicit "West to East" or "East to West" classification of all test runs.

Although the qualitative test design guidelines provided in FED STD 1043 proved to be generally adequate in the ARPANET application, the participants agreed that a more comprehensive itemization of network and user conditions which may affect data communication performance would be helpful to future users. Such an itemization will be added to the standard.

4.4.3 Sample Size Determination

The second step in applying the proposed FED STD 1043 Statistical Design and Analysis procedures in the ARPANET experiment was to select target sample sizes. The measurement standard provides explicit, step-by-step procedures for determining sample sizes for each type of performance parameter, and these procedures were successfully applied in the ARPANET experiment. However, the selection of sample sizes did involve a substantial amount of "trial and error" as a result of two factors: (1) a lack of prior information on the user-to-user performance of the ARPANET, and (2) a certain inexperience on the part of the project engineers in assessing the time and cost implications of measurement precision objectives. In the following, we summarize the major steps by which sample sizes were determined in the ARPANET experiment, and comment briefly on how the proposed FED STD 1043 procedures might be made a bit more tolerant of such user deficiencies in future applications.

A preliminary step in determining the measurement sample sizes was to establish the size of an individual test run or "batch." The batch size can be determined independently from the overall sample size, since individual batches can be aggregated to create a larger sample. The batch size does

affect the efficiency with which a given confidence level is achieved, however, for two reasons: (1) data from different batches are often much less correlated than data from the same batch; and (2) it takes much less time to process a few large batches containing a desired total number of trials than to process many small batches containing that same total number of trials. The access/disengagement and UIT batch sizes specified in Section 4.3.1 (160 accesses, 10,240 bytes) were chosen as a reasonable compromise between these goals. The effect of correlation within batches on the access/disengagement sample size requirement is discussed below.

As noted earlier, the determination of overall sample sizes for the various parameter measurements proved to be an iterative process. An initial feeling of the measurement engineers was that it would be "nice" to measure the probability parameters with at least 50% relative precision at a 90% confidence level, and the time parameters to within $\pm$ 5%. This formulation of the measurement precision objectives, while conceptually simple, proved to be unrealistic for two reasons:

1. Measuring different parameters with the same precision requires, in general, that each parameter be measured with a different sample size. This is true because the sample size is determined, in part, by the performance values observed. The collection and processing of separate samples for each parameter would have greatly complicated the measurements and would have represented inefficient use of time and data.

2. Measuring probabilities with 50% relative precision and a 90% confidence level is very costly and time consuming if the probability values are low (e.g., 10^{-6}), because the required sample sizes are extremely large.

As an example of the latter problem, Figure 20 shows that 50% relative precision at a 90% confidence level on a failure probability estimate requires the observation of 12 failure outcomes, assuming independence between successive trials. If the probability being measured is, say, 10^{-10}, the measurement will require, on average, 1.2×10^{11} trials--e.g., 120,000,000,000 transmitted bits. Even at a transmission rate of 5 kilobits per second, such a measurement would require continuous transmission of test data for more than 9 months!

The fundamental problem here, quite clearly, was that the initial precision objectives were selected too arbitrarily. A better approach is to derive the measurement precision objectives from a systematic study of <u>how the measured values are to be used</u>--specifically, the <u>relationship</u> between the

precision of a measured parameter value and the utility or worth of that value to the user. The engineers responsible for the ARPANET tests agreed that the measurement standard should be revised to include general guidelines for conducting such a study.

The "second pass" at determining measurement sample sizes was undertaken more carefully, and proved to be much more successful. The general approach used was the following:

1. Sample sizes were defined by function rather than by individual parameter. Thus, for example, all of the access parameters were estimated on the basis of a common access sample--a specified number of access attempts.

2. Within each function, the selection of sample size was based on the precision objectives for one or two target parameters for which the precision was judged to be particularly important or particularly difficult to achieve. The precisions for the other parameters were simply calculated, a posteriori, from the sample sizes already specified. In general, the measured precisions for these parameters were both better and less important than those of the target parameters.

3. Sample sizes for the target parameters were derived in four iterative steps: first, an examination of each parameter value's probable application in the "outside world"; second, the specification of a tentative precision objective based on that application; third, calculation of a sample size corresponding to the precision objective; and finally, an evaluation of that sample size from the point of view of measurement time feasibility.

The third step involved the conduct of preliminary measurements in most cases. The sample sizes ultimately selected for the ARPANET experiment were the following:

Access function - 3,000 attempts.

User information transfer function - 3×10^6 bits.

Disengagement function - 3,000 attempts per user.

The following paragraphs briefly summarize the process by which these measurement sample sizes were derived.

4.4.3.1 Access Function

The sample size for the access function was determined by the precision objective for the parameter Access Time. The basic application envisioned for

the ARPANET Access Time data was comparison with other measured data (e.g., Access Times for commercially available "value-added" services). An examination of specification practices for several such services indicated that any precision better than ± 5% of the specified value would be fully satisfactory; and the feasibility of achieving measurements of this precision was substantiated by the ± 5 millisecond accuracy of the event time-stamping (Appendix D). An Access Time precision objective of ± 0.1 second was tentatively selected.

Proposed FED STD 1043 defines six different procedures for determining sample sizes for the time parameters, each assuming different prior knowledge about the performance of the measured service. Neither an upper bound on the standard deviation nor the lag-1 autocorrelation of ARPANET access times was known prior to this experiment, and thus it was necessary to estimate these parameters via a set of preliminary samples in order to define an overall sample size. A number of early access/disengagement tests were examined, and four tests (numbers 71, 73, 75, and 76 in Table 3) were selected as the preliminary samples. The calculation procedure, following proposed FED STD 1043 Section A.4.2.2, case 5, was as follows:

1. A target precision (half-length of confidence interval, A) was specified as 0.1 second, with a confidence level of 95%.

2. Means $\bar{w}$, standard deviations s, and lag-1 autocorrelations r_1 were calculated for the four samples:

$\bar{w}$	1.770	1.491	2.017	1.819	Mean = 1.774 second
s	0.399	0.440	0.778	0.568	RMS = 0.566 second
r_1	0.157	0.149	0.145	0.038	Mean = 0.123

3. The half-length H_{cor} of the confidence interval was calculated for the 640 preliminary observations, ignoring at first the fact that they were obtained in four groups. For $n_1 = 640$, $t_{n_1-1,.05} = 1.96$, the normal percentage point. Hence

$$H_{cor} = 1.96 \frac{0.566}{\sqrt{640}} \left(\frac{1.123}{0.877}\right)^{1/2} = 0.0496 \text{ second}$$

The desired precision, A = 0.1 second, appeared to have been attained already, and no further observations appeared to be needed. However, it was necessary to examine two assumptions: first, that r_1 accounts for correlation among the

observations sufficiently (according to the Markov model $r_k = r_1^k$); and second, that the internal variances s^2 accounted for variation among the four samples sufficiently. The autocorrelations of lag-2 through -10 of the data, averaged over the four samples (groups), were:

0.129 0.121 0.116 0.105 0.066 0.076 0.061 0.027 0.036;

whereas the Markov model with $r_1 = .123$ would dictate $r_2 = r_1^2 = 0.015$, $r_3 = r_1^3 = 0.002$, $r_4 = r_1^4 = 0.0002$, ... Thus, the Markov model was not followed very well. The significance of the departure from the Markov model was examined as follows. It was determined first whether these autocorrelations differed significantly from 0, using the rough standard error $(n)^{-1/2} = 0.04$. Based on 640 observations, the autocorrelations r_1 through r_5 do differ significantly from 0, and from the model values, at the 5% significance level. Fortunately, none of them was large; the increase in number of observations required by $r_1 = 0.123$ according to the Markov model was 28%. To bring H_{cor} up to A = 0.1 using the Markov model would have required r_1 to be 0.651, and then r_2 through r_6 would have exceeded the values observed. Hence, the departure from the Markov model did not weaken the precision greatly in this case. However, it is generally desirable to check consistency with the model, and it is not difficult to do if a program for calculating the autocorrelations is at hand. The IMSL program was used here.

The second assumption was that observations in different groups differed from one another no more than ones in the same group (to phrase it a bit differently). This was tested by comparing the variance derived from the four means $\bar{w}_1$ with the mean square internal variance $s^2 = 0.320$. The variance of the four means was $(0.2169)^2 = 0.0471$, which must be multiplied by 160 (since $\sigma_{\bar{w}} = \sigma_w/\sqrt{n}$) to provide an independent estimate of the variance of individual access times. That yielded 7.530, which is 23.5 times the internal variance, so there was no question that there was additional variation between groups.

The confidence limits for Access Time were revised to take account of this added variation by using equations (A.4)-(A.10) of Crow (1979). The 95% confidence limits for Access Time, including the added variation between samples, were calculated to be

$$1.774 \pm 0.217 = \begin{cases} 1.991 \\ 1.557 \end{cases} \text{ seconds.}$$

Taking account of the variation between groups of 160 access times occurring some time apart showed that the mean Access Time was much more poorly

determined than a single sample of 160 taken in rapid succession (10 seconds apart) would have led one to believe. It was thus necessary to take more samples to achieve a precision of $\pm$ 0.1 second.

The total number of samples (of 160 trials each) required was calculated to be approximately

$$(0.217/0.1)^2 \times 4 = 18.8,$$

since a precision of 0.217 was achieved with four samples and the number of samples goes up as the square of the precision (as measured inversely by half-length of confidence interval). This intuitive result is in accord with (A.13) of Crow (1979). Rounding up to 19 samples gave an Access Time sample size objective of about (19)(160) = 3040 access attempts. Obtaining an overall access sample of that size was judged to be quite feasible, since the first 640 attempts were collected in less than 2 weeks.

The fact that so little of $s_{\bar{x}}^2$ was contributed by internal variance showed that it would have been unnecessary to include as many as 160 access attempts in each sample. If only 40 attempts had been included in each sample, the total number of samples would have increased only to

$$\frac{4}{(0.1)^2}\left(0.0451 + \frac{0.320}{40}\right) = 21.2$$

The total number of access times with 22 samples of 40 each would have been 880, compared with 3040 with 19 samples of 160 each. The test design was not changed to reflect this savings because a larger number of trials was needed to obtain the desired precision in estimating Disengagement Time, as discussed below.

4.4.3.2 User Information Transfer Functions

The sample size for the bit, block, and "message" transfer functions was determined by a precision objective for the parameter Bit Error Probability. It was assumed that this parameter would impose the most stringent sample size requirement among the UIT parameters, because a very effective (24-bit) end-to-end error control is used in the ARPANET. This assumption is reflected in Kleinrock's (1976) observation that ". . . as far as we know, no undetected errors have as yet passed through the network. These failures to detect errors have been designed to occur on the order of years to centuries

apart. . .". A preliminary ARPANET measurement substantiated this view, since no bit errors were observed in over 300,000 transferred bits.

In defining a sample size for Bit Error Probability, it was decided to make the "worst-case" assumption that no bit errors would be observed, even in a very extensive test of several month's duration. In such a case, the estimated Bit Error Probability is zero; and the critical parameter becomes the upper confidence limit on that estimate--the lowest value one can say is "almost certainly" greater than the true Bit Error Probability. On the basis of earlier measurement results and known constraints on the experiment duration, it was decided to seek a sample size which would place that limit near 10^{-6} at the 90% confidence level. Equations presented in the measurement standard demonstrate that this objective requires a sample size between 1 and 5 million bits, depending on the dependence assumed. A sample size of 3 million bits was selected for the ARPANET UIT measurements.

4.4.3.3 Disengagement Function

Sample size determination for the disengagement function was accomplished in the same general manner as described above for access. As a practical matter, the access and disengagement sample sizes are normally similar because the two functions are operationally a pair--"what goes up must come down."

The measurement precision objective for Disengagement Time was substantially less stringent than that for Access Time, for two reasons: first, because previously measured values with which the ARPANET Disengagement Times might be compared were very coarsely specified; and second, because the source user Disengagement Times observed in preliminary measurements were extremely short, and therefore, more subject to other types of measurement error (e.g., the $\pm$ 5 ms uncertainty in time-stamping interface events). It was concluded that even a measurement uncertainty in the neighborhood of $\pm$ 20% would be quite satisfactory.

Because the precision objective for Disengagement Time was less stringent than that for Access Time, it was expected that a disengagement sample about equal to the access sample would provide more than sufficient precision in the Disengagement Time estimates. However, a preliminary measurement of Disengagement times was conducted to determine whether it might be possible to base the Disengagement Time estimates on a smaller sample. The four preliminary samples of 160 access attempts made available about the same number of disengagement attempts. These were observed at both the source and

the destination. It was immediately apparent that the two sets should not be combined, because the largest source time was much less than the smallest destination time.

Since the access time samples had much larger between-sample variation than within-sample (internal) variation, it was hypothesized that the same would be true for the disengagement times, and that thus it might be sufficient to analyze just 40 times in each sample. The means and standard deviations for the last 40 trials in each sample, averaged over the four preliminary samples, are:

	$\bar{w}$	s	
Source	0.0117	0.0073	second
Destination	2.4868	1.1536	second

The internal standard deviations are much larger fractions of the means than in the case of access times, and thus, it was concluded that the full (access) sample size should be used in estimating the disengagement parameters.

5. CONDUCT OF THE TEST

The numerical measurement results described in Section 6 of this report are based primarily on 21 access/disengagement tests and 23 user information transfer tests. Each test was characterized by a set of "test variables" or measurement conditions expected to have a significant influence on measurement results, as discussed in Section 4.4.2. These conditions may be summarized as follows:

1. Direction of Transfer
   a. East-to-West (NBS/Gaithersburg to ITS/Boulder)
   b. West-to-East (ITS/Boulder to NBS/Gaithersburg)

2. User Program Priority
   a. High (enhanced execution priorities for both XMIT and RECV programs)
   b. Normal

3. Time of Test
   a. Peak Hours (weekdays, 8 AM - 6 PM Eastern Local Time (ELT), except 12 N - 1 PM)
   b. Off Hours (all hours except peak hours)

Table 2 shows the type/condition characteristics and date-time identifier of each of the 44 tests used in calculating the measured values. As indicated in the table, these tests spanned a 2-month measurement period from September 17 to November 24, 1981. Gaps in the chronological sequence of test numbers

Table 2. Summary of ARPANET Tests and Conditions

TEST NO.	TEST TYPE		TEST VARIABLE						TEST DATE-TIME
			DIRECTION		PROGRAM PRIORITY		TIME OF TEST		
	A/D	UIT	E-W	W-E	HIGH	NORMAL	PEAK HRS	OFF HRS	
71	X		X		X		X		9/17-1806
73	X		X		X			X	9/17-2214
74	X		X			X		X	9/22-2209
75	X		X		X		X		9/23-1846
76	X		X		X		X		9/24-1827
77	X		X		X		X		9/24-1946
80	X		X		X		X		9/25-1511
81	X		X			X		X	9/25-1600
82	X		X		X		X		9/25-1735
83	X		X		X		X		9/25-1919
84	X		X		X		X		9/25-2024
85	X		X			X	X		9/25-2140
86	X		X		X			X	9/26-1726
87	X		X		X			X	9/28-2238
89	X		X		X		X		9/29-1328
90	X		X		X		X		9/29-1824
92	X			X		X	X		10/16-0005
93	X			X		X	X		10/19-1727
95	X			X	X		X		10/21-1713
96	X			X	X		X		10/23-1738
97	X			X	X		X		10/23-1836
98		X	X			X	X		10/27-2108
99		X	X		X		X		10/27-2139
101		X		X	X			X	10/28-2213
102		X	X		X			X	10/28-2224
103		X		X	X		X		10/29-1522
104		X	X		X		X		10/29-1524
105		X	X		X		X		11/03-1432
106		X		X	X		X		11/03-1435
107		X	X		X		X		11/04-1625
108		X		X	X		X		11/04-1627
109		X		X	X		X		11/05-2135
110		X	X		X		X		11/05-2137
111		X	X			X	X		11/10-1418
112		X		X		X	X		11/10-1421
113		X	X		X		X		11/12-1945
114		X	X		X		X		11/13-1447
115		X		X	X		X		11/13-1449
118		X	X		X		X		11/18-1553
119		X		X	X		X		11/18-1557
122		X	X		X		X		11/20-1815
123		X		X	X		X		11/20-1818
126		X	X		X		X		11/24-2054
127		X		X	X		X		11/24-2057

indicate tests which were excluded in calculating the measured values. These excluded tests were of two types: (1) tests conducted prior to the formal measurement period to verify and fine-tune the measurement system; and (2) tests rejected during the formal measurement period as a result of measurement system (or operator) errors. No valid test data taken during the formal measurement period was excluded.

This section describes the detailed operator procedures used in conducting the ARPANET measurements. This discussion is organized into four subsections covering data extraction, performance assessment, statistical analysis, and data archiving.

5.1 Data Extraction

The description of the conduct of the Data Extraction phase of the ARPANET experiment is divided into three subsections. The first deals with the conduct of a typical ARPANET test, i.e., the on-line (real-time) extraction of a measurement data batch. The second describes the procedures followed in consolidating test data files for processing at the ITS site. The third outlines the conduct of the post-test processing of a measurement data batch. Where test procedures were sufficiently similar for access/disengagement and user information transfer tests, only one is described, with differences noted. Where the two test procedures were substantially different, each is described separately. While the majority of the tests proceeded as described, there were some unusual situations which necessitated a decision by the operator as to whether to abort, continue, or terminate a test.

Most of the test steps were incorporated into command files, and hence did not require individual operator actions. Exceptional conditions and error messages were directed to the test operator's input/output terminal. Although the two software programs which performed the data extraction, i.e., the XMIT and RECV programs, were written to accept control parameters at runtime, it was easier to have two versions of the program resident in the UNIX files: one for access/disengagement tests and one for user information transfer tests. The same files were created in both tests, but the file lengths were different.

The amount of time to conduct a test varied. An access/disengagement test took approximately 40 minutes for a full set of 160 sessions (with one block of 64 bytes being sent during each session). A user information test

took less than 3 minutes to send and record the same 10,240 bytes of data, clustered in 20 blocks of 512 bytes, during a single session.

5.1.1 On-Line Testing

Under normal conditions, tests were run from Boulder without involving personnel at Gaithersburg. The test operator first logged onto the Boulder UNIX system via a local CRT terminal, and onto the Gaithersburg UNIX system via a second terminal a few feet away. This second terminal was connected directly to the DOCB ARPA network TIP, and therefore did not have any connection with the NTIA/ITS host. Connection was made to the NBS UNIX-3 system by typing

@o 211 ,

where the @ symbol informs the TIP the next character is a command, the o is short for open, and the 211 is the address recognized in the ARPA network for the UNIX-3 system in Gaithersburg. After logging on at both sites and changing to the appropriate software directory at ITS with the command

cd /disk2 ,

the operator checked the preface files with the command

cat pref*

to ensure that the test numbers and other information were correct for the test to be run. If not, the text editor was used to correct the data.

The next step was to check the system clock and satellite receiver clock in each system. If their respective times differed by more than a preset amount (5-20 minutes), the program notified the operator and exited. (Note that the system clock was not used in time-tagging events.) Also, since the NBS satellite receiver clock could not be seen at ITS, that clock was remotely checked for synchronization with the GOES time satellite signal. This check was made with the command

qik_clock ,

which elicited a brief display of the satellite receiver and system clock times. In the normal case where the satellite receiver clock time and the received GOES satellite time were synchronized, that time was printed with a trailing blank space. If the printed time ended with any character other than a blank space, the receiver clock was out of synchronization; the printed

character indicated the approximate amount of error that could be expected as a worst case. If the clock was out of synchronization, the operator determined the cause and restored synchronization before proceeding. Such instances were rare.

The operator initiated a test by typing a pair of shell commands, one at each terminal. For example, the commands

```
sh/runrecv recv.o 126
```

at ITS, and

```
sh/runxmit xmit.o 126
```

at NBS caused an access/disengagement test (number 126) to be run, with the source user at NBS. If the test programs RECV.U and XMIT.U had been specified, a user information transfer test would have been run. (In general, similar user information transfer and access/disengagement command files are distinguished by a .U or .O suffix.) Once the program initiate commands were typed, the shell command files performed the test as described in Section 4.1.3.

As a general practice, RUNRECV was started first and RUNXMIT second. Coupled with the 10-second delay in XMIT described previously, this practice normally assured that the destination user was "listening" before XMIT issued an access request, thereby avoiding a user blocking condition.

While the test was running, the operator made a handwritten entry into his log book. This included the date, the test number, the local time, the direction of data transfer, the names of the programs used at each site, the number of accesses, the number of blocks per session, and the number of bytes per block. A sample of the form used to record this data is shown in Figure 29.

When each test was completed, a normal operating system prompt was returned to the operator. The operator then had the option to perform another test run in the reverse direction by typing (for example)

```
sh/runxmit xmit.o 127
```

at ITS, and

```
sh/runrecv recv.o 127
```

at NBS. If tests were run in both directions before consolidating the files to the ITS site for processing, there were seven files to transfer from the NBS site. These files were, for the example above:

RUN NO.	DATE MONTH-DAY	LOCAL TIME HR:MIN	SITES X(MIT) OR R(ECV) NBS	ITS	NBS PROG NAME	ITS PROG NAME	AMOUNT XMITTED # ACCESSES	# BLOCKS	# BYTES	DATE-TIME FOR ARCHIVING	DATE PROCESSED	OUTCOME OF TEST AND COMMENTS	ARCHIVED TO TAPE NO.
82	9/25	1130	X	R	xmit.o	recv.o	160	1	64	25-1735	10/22	OK	77
83	9/25	1315	X	R	xmit.o	recv.o	160	1	64	25-1919	10/20	OK	77
84	9/25	1415	X	R	xmit.o	recv.o	160	1	64	25-2024	10/16	OK	77
85	9/25	1530	X	R	xmit.oslo	recv.o	160	1	64	25-2140	10/15	OK	77
86	9/26	1130	X	R	xmit.o	recv.o	160	1	64	26-1726	10/14	OK	77
87	9/28	1170	X	R	xmit.o	recv.o	160	1	64	28-2238	10/13	OK	77
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
122	11/20	1115	X	R	xmit.u	recv.u	1	20	512	20-1815	11/20	OK	84
123	11/20	1118	R	X	recv.u	xmit.u	1	20	512	20-1818	11/20	OK	84

Figure 29. Example test log form.

xmit files (first test)	recv files (second test)
1. log.x126	1. log.r127
2. history.x	2. history.r
3. overhead.x	3. overhead.r
	4. data.r

When the operator completed the desired testing (either one run or a pair of runs in opposite directions), he created a directory listing of the remote (NBS) site files by typing

```
ls -l
```

The operating system returned the time the files were created and the length of the files in bytes. The operator noted the byte counts on the files to be transferred as an important initial check on loss or insertion of user information in the transfer process. The operator then logged out of the NBS Gaithersburg system and closed the TIP connection by typing

```
@c
```

5.1.2 Data Consolidation

After concluding a test (or a pair of tests in opposite directions), the next step in the measurement process was to move the files to appropriate directories in the ITS host computer. All the steps now described were performed at ITS in Boulder, and assume, for example, the test numbers specified above.

The operator first changed directories to the subdirectory for data by the command

```
cd data    ,
```

and moved the receive and transmit files from the /DISK2 directory to /DISK2/DATA by means of the two command files

```
sh/mover 126
sh/movex 127    .
```

The test number was used, in each case, after the appropriate command file name. Note that the command MOVER moved the receive files and the command MOVEX moved the transmit files. At this point, the DATA subdirectory contained the following files from ITS:

receive files (first test)	transmit files (second test)
1. log.r126	1. log.x127
2. history.r126	2. history.x127
3. overhead.r126	3. overhead.x127
4. data.r126	4. data.x

In addition to moving data files, the command files appended 126 or 127 to the end of the file name of the HISTORY, OVERHEAD, and receive DATA files. The DATA.X file was the same for all tests and was permanently left in the directory, and hence did not need to be moved or identified with a suffix. (The DATA.X file is the eighth file listed above and was not moved with the other seven.)

It should be recalled that when a file is "moved" from one directory on a disk to a subdirectory on the same disk, it is not physically moved at all by the UNIX operating system; only the pointer to its name is changed. Thus one need not be concerned about losing bytes or changing bits while "moving" files around on the same disk. This is not true if one moves data to a different disk.

The next step in the data consolidation was done by the File Transfer Program. That high-level ARPANET program copied the seven files at NBS to ITS. The program was invoked by the following operator command/computer response sequence (computer responses are underscored):

```
                        ftp
                        HOST: nbs-unix
                        Attempting connection to nbs-unix
(after a second or two) Connection open
                        user name
                        pass ***
                        User logged in    .
```

"Name" is the user name of the account in which the test was conducted; and of course, the appropriate password was used. The binary mode was then set by typing

```
bin
```

and the transfer of each file was done with

```
get
Local filename  history.x126
Foreign filename  history.x
```

and so on for all seven files. Notice that the HISTORY, OVERHEAD, and received DATA files must have the test number appended to them; and the creation of the new file in the ITS host was the opportune time to do this. As each file was transferred over the network, a total byte count was reported by the ftp program. This was compared with the byte count obtained from the ls -l directory obtained earlier. After all seven transfers were successfully completed the ftp was ended by the command

bye .

The operator was then returned to the UNIX operating system.

It was now necessary to compare the transferred DATA file, and the locally received file, with the stored reference. This was done by the command sequence

```
cmp   data.r126   data.x
cmp   data.r127   data.x   .
```

Normally, the files compared. If so, the system provided no messages to the operator. Any difference in the files produced an error response.

If there were differences, the operator noted this in the handwritten log sheet. If the received DATA file that was transferred over the network did not compare with the stored reference, the error could have occurred either during the file transfer process or during the actual measurements. This was easily determined by doing an ftp in the reverse direction (naming the file differently at the remote site), and then comparing the two remote files. If the remote files differed, the error occurred during the post-test file transfer process. In that case, the original file was transferred to ITS again to get the correct "image" of the file recorded at NBS. If the remote files were identical, then an error occurred in the network during the test.

After the files were properly consolidated into the DATA directory, the next step was to determine the archive date-time group to be used in storing the files. Each file associated with a single test was stored in a directory whose name had the following form. The first three characters were letters which indicated the month. The next seven characters were numbers, separated by a hyphen, which indicated the day-hour minute (in UT) at which the test was started. For example, a test that was started on December 5, 1981 at 5:32 p.m. would have the seven raw data files stored on the disk as

dec/05-1732name.x .

The identifiers NAME.X and NAME.R denote a transmit or receive file, respectively. The name identifies the file type as history, overhead, log, or data.

The command to determine the date-time group (dtg) was

```
sh/getdtg 126
```

where the lower test number in the set of the two tests was entered. In the example case, the SHOW program was automatically executed for both test 126 and 127. The test number was read from either the CRT screen or from a file in the DATA directory where all the run numbers and archive times were stored. For example, after issuing the above command, the last four lines of the DTG.SHORT file read

```
  Data from file history.x126
**File prefix (dd-hhmm) = 05-1732
  Data from file history.x127
**File prefix (dd-hhmm) = 05-1735
```

The test operator manually entered these numbers into the log sheet in the proper column.

The two sets of seven files for each test are then moved to their final subdirectory with the command

```
sh/storem dec 05-1732 126    ,
```

where the month, dtg, and test numbers were entered separately after the command file name STOREM. (Again, only the name pointer was changed - not the file.) In a similar manner, the raw data files for the other test were moved with the command

```
sh/storem dec 05-1735 127    .
```

This completed data consolidation and file archiving. The file directory for the month included all the other files, but to recall only the files run in this test the wild card * may be used when asking for the December directory. For example, the command

```
lsm dec/05-17*
```

would show on the screen

```
05-1732data.r     05-1732ovh.r      05-1735log.r
05-1732his.r      05-1732ovh.x      05-1735log.x
05-1732his.x      05-1735data.r     05-1735ovh.r
05-1732log.r      05-1735his.r      05-1735ovh.x
05-1732log.x      05-1735his.x    .
```

5.1.3 Post-Test Processing

The processing of the raw data files which were stored in the month directory was done completely by a command file. By merely executing, for example, the command

sh/doit.u dec 05-1732 its

from the DATA directory, all the time corrections, merging, and reformatting of binary files to ASCII character files, as well as the determination of access or block transfer times and creation of the plottable histogram files, were accomplished. The name of the command file was DOIT.O or DOIT.U depending on whether the file was for the access/disengagement or user information transfer test type. The parameters after the command file name were the month directory, the dtg of the archive file name, and the location of the destination site, i.e., "its" or "nbs".

The entire process took about 12 minutes for the information transfer data and about 45 minutes for the access/disengagement data. The latter data took longer to process because of the greater number of times in the files. The processing took much longer on the ITS computer system than it might have because the "C" compiler did not recognize the minicomputer's hardware floating point, and therefore had to perform all the time corrections with software floating point routines.

5.2 Performance Assessment

This section outlines the operator procedures followed during the Performance Assessment phase of the ARPANET experiment. The description, which parallels that for Performance Assessment design in Section 4.3, is organized into two subsections. The first deals with the FORTRAN processing of the four standard ASCII files in a performance measurement batch as specified by proposed FED STD 1043. The second describes the conduct of supplemental data processing procedures used to obtain additional statistical information about access and block transfer times.

5.2.1 Standard FORTRAN Performance Assessment

The performance measurement data batch corresponding to each ARPANET test (i.e., the four standard ASCII character files output by the interface monitors, as specified in proposed FED STD 1043) was processed in a single-batch performance assessment. In addition several multibatch performance

assessments were made by utilizing the carryover data features described in proposed FED STD 1043. Aggregate samples were formed by combining performance data batches derived from tests having uniform type, direction, and program priority.

Prior to each performance assessment run, the operator created the specifications input file appropriate for that run by using the UNIX text editor. The operator also assigned the name FORTXX to each input file required in the particular performance assessment run, and stored that file under the /DISK2/ASSESS directory. (The symbols XX in a file name represent the digits of the unit number associated with the file by a BLOCK DATA subprogram included in the standard performance assessment software.)

The standard FORTRAN performance assessment programs PROLOG, ANALYZ, and EPILOG were executed by invoking the appropriate version of a command file named PASSRUN. At the conclusion of each run, this command file also printed the assessment summary file, printed auxiliary files of chronologically arranged performance times for access/disengagement or for block transfer, sorted these auxiliary files in ascending numerical order, and printed the sorted files.

5.2.2 Supplemental Data Reduction

For each ARPANET test, the post-test command file DOIT generated a file containing the x-y coordinates of the vector trace of a histogram of access time (for access/disengagement type tests) or block transfer times (for user information transfer type tests). This file was named, for example, PLOT20ACC or PLOT10XFR, where the digits denote the maximum number of class intervals in the histogram. The plot files were translated into histograms as described below.

The plot file was listed from the /DISK2/DATA/TEMP directory to the intelligent graphics terminal by typing (for example):

```
cat plot16acc .
```

The CRT memory capacity was about 10K bytes, and the plot file normally was less than 1.5K bytes. At this point the PDP-11 computer was no longer needed and the operator logged out. Using the "scroll" keys and "delete" and "clear" keys of the CRT, the operator removed all lines of data from the display except the x-y data coordinates. The operator then selected the autoplot menu and entered appropriate maximum and minimum values for the x and y axes. The

histogram was plotted, and appropriate scales were placed on its axes, all within a few seconds after depression of the CRT "autoplot" and "autoplot axes" keys.

The graphics cursor was positioned for labels and the graphics text size and orientation keys were used to complete the CRT graphics screen. The total process took only 3 to 5 minutes, with most of the time being used for the composition, typing, and placement of text annotating the graph. An additional stroke of one of the user definable keys caused the graphics display to be printed by a dot matrix graphics printer; the printing required about 1 minute. An example of the graphics output from the printer is shown in Figure 30.

Supplementary access and block transfer time statistics and histograms for operator-defined test aggregates were obtained by utilizing the "TAB.ACC" or "TAB.XFR" files generated by DOIT and stored with raw data in the month subdirectory. These files, whose name also included a month/date-time prefix (e.g., SEP/24-1827TAB.ACC), consisted of a sequence of records sorted according to ascending performance times. The operator extracted performance times from the files for the desired aggregate and concatenated the results to a file which served as input for the appropriate statistical and histogram routines.

When the standard FORTRAN performance assessment and the supplemental data reduction for a performance data set were completed, the results of the two independent processing techniques were manually compared. In all normal tests which had run on the principal paths of the session profile of Figure 25, the two sets of data agreed. Only in exceptional cases did any differences appear. Depending on the nature of the discrepancy, the raw data files or time-corrected data files were listed to examine in detail all the event times and conditions surrounding the abnormal event. For example, one very long access time occurred in the middle of a test which ran unattended. Normally, the operator would have terminated the test, but after 12 minutes, the access attempt "succeeded" and the test continued. In spite of that one extremely long delay, the test ran to completion; however, 3 of the 160 transmitted data blocks were lost in this test.

5.3 Statistical Analysis

About midway in the planned sequence of access/disengagement tests, a statistical analysis of the collected Access Time data was conducted to check

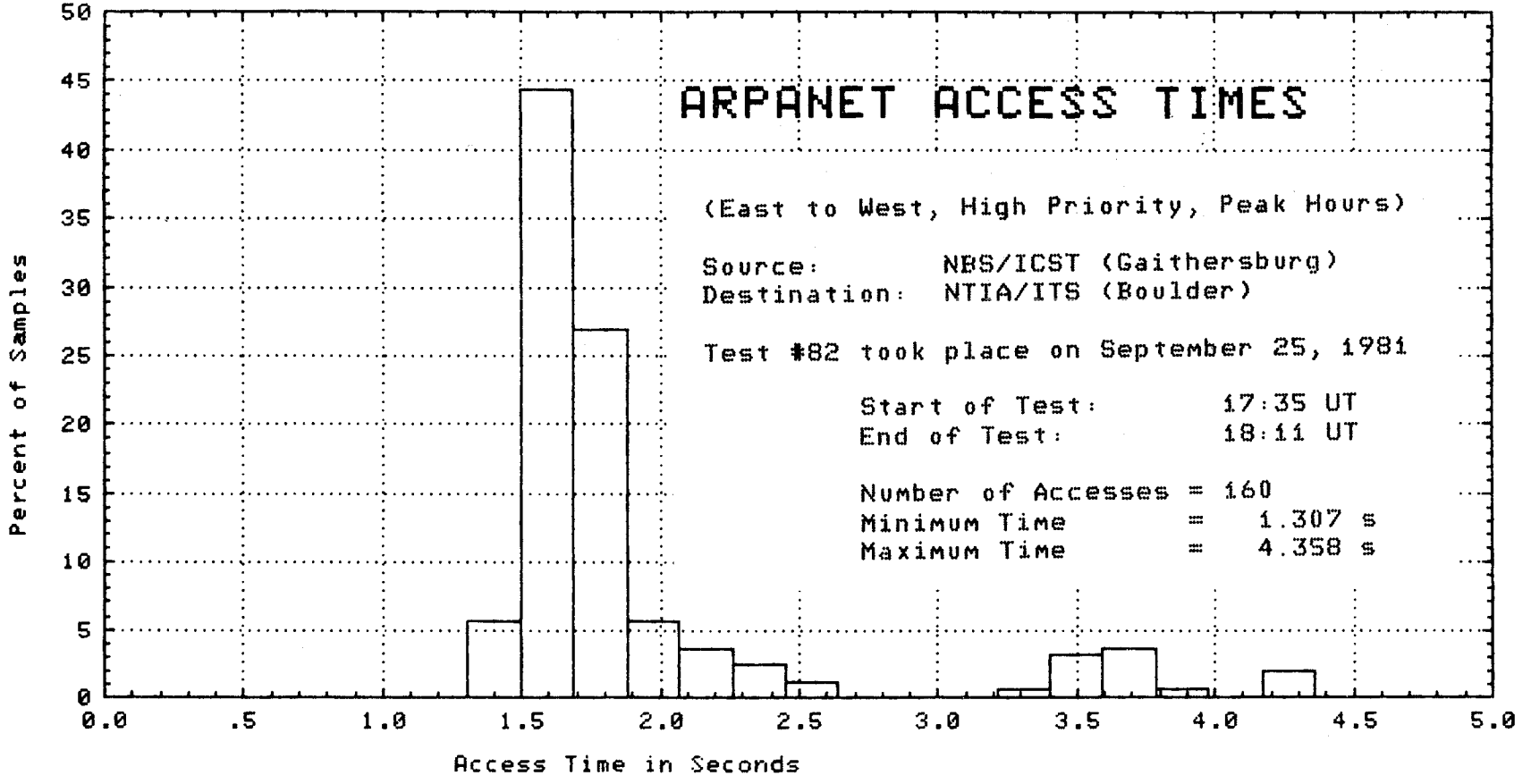


Figure 30. Graphical sample of histogram of access times for test 82.

earlier dependence assumptions and refine the preliminary sample size estimate. This analysis, which is summarized in Section 6.2.1, essentially confirmed the earlier estimate. At about the same point in the measurements, Disengagement Time data were subjected to a similar statistical analysis, as described in Section 6.2.3.

After all tests had been completed and the resulting data had been processed by the various Performance Assessment routines, final confidence limit calculations were carried out for the measured performance parameters. In addition, several statistical analyses were performed to assess the significance of "test variables" (i.e., direction of transfer, program priority, time of test) in measuring performance parameters. The results of these calculations and analyses are presented in Section 6.

5.4 Data Archiving to Removable Mass Storage

The "on-line" mass storage for the ARPANET measurements consisted of a single, dedicated 2.5-Megbyte RK05 hard disk platter. This storage was allocated among program and command files, raw data files, and processed data files. The (compiled) program and command files, which were absolutely essential, occupied about 20% of the disk (i.e., about 1000 512-byte blocks). Another 20% of the disk was required for temporary files during post-test and performance assessment processing. Each access/disengagement test generated about 140 blocks of raw data; the "TAB.ACC" and standard ASCII files required another 13 and 350 blocks, respectively. A user information transfer test required about 40 blocks for raw data, 2.5 blocks for the "TAB.XRF" files, and 13 blocks for the standard ASCII files. The ASCII files corresponding to a test were regarded as expendable after they had been processed by the standard FORTRAN performance assessment programs, and were removed from the /DISK2/ASSESS directory. Even with this storage-conserving tactic, there was not enough space on the disk for all raw data and "TAB" files generated during the experiment. Preservation of these files was essential for additional processing or re-examination at a later time, and for aggregation of selected tests. Consequently, raw data and "TAB" files were copied onto magnetic tape whenever it became necessary to obtain disk space for additional tests and processing.

All files on /DISK2 and its subdirectories were stored on tape drive 0 from /DISK2 by the command

```
tar c0  ,
```

then a directory of the archived tape was made with

```
tar tOr    .
```

If any of the files had to be recovered later from the tape, the command

```
tar xO ./data/nov/05-2135*
```

would, for example, recover all the files from the test that began on November 5 at 21:35.

6. PROJECT RESULTS

Section 3 defined the specific objectives of the ARPA network measurements project in terms of a series of questions to be answered by the project results. This section summarizes those results and answers each of the associated questions.

The organization of this section follows that of Section 3. Each question posed in that section is restated in its entirety here, followed by a detailed discussion of the pertinent project results. In all but a few cases, the project results directly answer the questions posed. Exceptions are highlighted as topics for possible future study. Overall conclusions and recommendations of the experiment are summarized separately, in Section 7.

6.1 Primary Results

As noted in Section 3, the primary objective of the ARPA network measurement project was to verify and demonstrate proposed Federal Standard 1043 by actually implementing it in a representative test situation. This subsection summarizes the ARPA network measurements project results that relate to that objective. The results summarized here focus on the measurement process, and its specification in proposed FED STD 1043, rather than on the performance values obtained. Implementation results are presented separately for each of the four performance measurement system elements identified earlier (Figure 11).

Because the questions in this subsection focus on the measurement process, most have been answered implicitly in the descriptions of that process in Sections 4 and 5. The purpose of this subsection is to recall each question and explicitly state its answer. For conciseness, material adequately presented in Sections 4 and 5 is not repeated here. Specific references to those sections are provided where appropriate.

6.1.1 Data Extraction Element

The prototype FED STD 1043 Data Extraction subsystem used in the ARPANET measurements is described in Sections 4.1 and 5.1 of this report. As noted in Section 3.1.1, the development of that subsystem was intended to answer a number of specific implementation questions. Those questions were divided into three groups, corresponding to the input, processing, and output functions of a typical Data Extraction subsystem. The four questions related to the Data Extraction input function are restated and answered below.

1. Can the end user/data communication system interfaces be identified unambiguously from the guidance provided in the standard? Where are those interfaces in the configuration tested?

Identification of the end user/data communication system interfaces proved to be straightforward in the configuration tested. As discussed in Section 2.4, proposed FED STD 1043 defines three general types of end user/data communication system interfaces, each corresponding to a particular type of end user: operator, medium, and application program. Both end users in this case were computer application programs. In all cases where the end user is an application program, proposed FED STD 1043 defines the user/system interface to be the functional interface between that program and the local host computer operating system, telecommunication access method, or equivalent. Thus, in the case studied, each user/system interface was defined to be the functional interface between the (applicable) XMIT or RECV application program and the local UNIX operating system.

2. What specific data communication - related events take place at the end user interfaces? Can these interface events be sequentially related in a "session profile" of the type illustrated in the standard? Can they be observed and recorded with a negligible (or at least measurable) effect on the process under examination?

Analysis of the user/system interfaces identified above revealed eight well-defined interface signals by which all application program/operating system communications are effected: namely, the OPEN, READ, WRITE, and CLOSE system calls and their associated "complete" responses. These signals, which were illustrated in Figure 23, are the interface events on which all assessment of performance was based.

The user/system interface events depicted in Figure 23 can occur in many different sequences. However, because the test approach used was an active

one, where the measurement entity and the end user were one and the same, it was possible to completely determine the normal interface event sequence via the application program designs. This was done, resulting in the session profile shown earlier (Figure 25). As discussed in Section 4, virtually all of the ARPA network tests followed the normal data flow path depicted in that figure.

Because an active measurement approach was used, all user/system interface events were directly available to the Interface Monitors, and no special "probes" were required to observe them. As discussed in Section 4, the process of recording the interface events was divided into on-line and off-line steps, with only the essential functions performed on-line. As a result, the only significant influence of the measurement process on the events being measured was the reading of the satellite clock. As discussed in Appendix D, that action took at most 132 milliseconds, and occurred once per interface event - just before the event in the case of system calls, and just after the event in the case of system responses.

The effect of the clock read times on the observed time parameter values was measurable in all cases, and negligible in most. The effect of each clock reading was to delay the occurrence of the next interface event by (at most) 132 milliseconds. In each case, these delays occurred at a point in the data communication session when a user was "responsible" for creating the next interface event. Thus, all of the clock reading delays were counted as user delays; contributed to the ancillary parameter values; and could be "factored out" of the corresponding primary parameter values. In no case was the total clock read time more than 15% of the average performance time being measured.

3. What is the effect of system or user failures on the observed event sequence? Is it necessary (and practical) to represent failure events in a session profile?

There were no "user failures" during the tests since both users were, in fact, carefully designed measurement programs. The effects of system failures on the observed event sequence depended on the type of failure. Two general types of system failures were postulated (and observed): failures which suspend (or "hang up") the test, and failures which omit expected events from the interface event sequence. An example of the first type is a failure of the system to respond to an Open request. That failure causes the measured system to "hang up" waiting for the response; and in the case studied, test operator action was required to break the impasse. An example of the second

type of failure is the loss of a block of data by the system. That failure does not cause the system to hang up, since the input of data by the source user is not dependent on the delivery of previously transmitted data.

Failures of the first type are readily handled by adding test operator functions to the basic session profile. The operator functions shown in Figure 25 were, in fact, added to an earlier session profile after the need for predictable, standardized operator responses to system failures became apparent.

Failures of the second type cannot be explicitly represented in the session profile because they are, in fact, absences of expected interface events. Such failures require no special processing in the Data Extraction subsystem in any case, since the Performance Assessment program detects missing events.

A third type of system failure could have occurred (but in fact, did not occur) during the testing: the generation of unexpected or superfluous events. An example would be the duplicate delivery of user information. As in the case of the absence of expected events, it is not practical or necessary to depict such failures in the session profile. Again, if such events had occurred, they would have been detected by the Performance Assessment program.

4. How do the specified end user interfaces relate to the "computer/communications" interfaces traditionally used in performance measurement? Is the difference between the two significant from an instrumentation point of view?

In the configuration tested, the end user functional interfaces are separated from the host computer physical interfaces by two distinct host programs: the UNIX operating system and the NCP (Figure 23). A third host program, the ARPANET I/O driver, actually handles transfer of data across the physical computer interface. This program is assembled as a part of the operating system.

Instrumentation of the computer/communications interfaces would be substantially different from instrumentation of the end user interfaces in the configuration studied. Principal reasons are first, the obvious differences in event significance and timing at the two interfaces; second, the fact that the former interface is physical and the latter functional; and third, the fact that the ARPANET NCP would be on the user side of the monitored interface in the former case. The latter fact would almost certainly dictate a

"passive" measurement approach (e.g., tapping into the user/system interfaces or placing traps in the TIP programs), since an "active" (surrogate user) approach would require extensive changes to the host NCP's.

The second group of three questions dealt with the Interface Monitor processing of observed interface events. These questions are restated and answered below.

1. Can the various events observed at the user/system interfaces in the configuration tested be unambiguously associated with the universal "reference events" defined in the standard? What is the exact association between these events in the case studied? Are there reference events with no corresponding interface event, or vice versa? Are the "user information" and "overhead information" categories clearly distinguishable from the guidance in the standards?

As discussed in Section 4.1.1, the process of associating system-specific interface events with system-independent reference events was straightforward in the configuration tested. The OPEN, WRITE, and CLOSE system calls and the OPEN COMPLETE and CLOSE COMPLETE system responses were each associated with a particular primary reference event. Each of the eight interface events was associated with one of two basic ancillary reference events - four of each type. There were no reference events without a corresponding interface event, and no interface events without performance significance.

The "user information" and "overhead information" categories were also clearly distinguishable in the case studied. The data files transferred in conjunction with the WRITE and READ COMPLETE events contained only user information, since all data in those files was intended to cross both user/system interfaces. All other transferred information was overhead, since it crossed either one or no user/system interfaces.

2. Is it practical and economically feasible to provide a synchronized time reference to each of two geographically distant Interface Monitors? How can this be accomplished, and at what cost?

The ARPA network measurements project clearly demonstrated that it is possible to accurately synchronize remote data communication Interface Monitors using the NBS/GOES satellite time dissemination service. As discussed in Section 4.1.2, this was accomplished by placing a commercially available satellite clock receiver at each test site, and connecting each receiver with the local host computer via a conventional terminal I/O (RS-232-C) interface. Installation and connection of the clock receivers was

accomplished with little difficulty, and the receivers proved to be effective and reliable over the entire test period. Times within the remote clock receivers are synchronized to within ± 10 microseconds.

The economic feasibility of synchronizing remote Interface Monitors via the satellite time service depends on the cost of the other measurement system components and, ultimately, the cost of the monitored system. The satellite clock receivers used in this experiment retail for less than $2K. That cost is a negligible fraction of the total cost of most large- and medium-scale computer systems on the market today, but could add substantially to the cost of very small systems. In cases where a less expensive synchronization scheme is needed, two options are available:

1. HF WWV Broadcasts. Time difference between a local clock signal and the WWV reference can be determined to within ± 1 millisecond using an HF receiver and oscilloscope. Time of day can also be read automatically from a BCD time code in the WWV broadcast, using commercially available WWV receiving equipment.

2. Telephone Time-of-Day Service. The same time-of-day signals broadcast over WWV are also available over commercial telephone lines, by calling (303) 499-7111. These reference signals can be compared with a local clock in the same way as the WWV signals. As a result of variability in the telephone line delay, the resulting synchronization is slightly less accurate than that obtained from HF transmission.

In the first case, the equipment cost could be less than $500. The necessary equipment would be readily available at many computer sites. The second approach would be still less expensive, since no HF receiver is required.

It appears that the unavailability of an accurate, inexpensive remote clock synchronization technique has substantially inhibited performance measurement in the past. This has been evidenced in two ways: first, a restriction of measurements to relatively few applications; and second, the selection of performance parameters which correspond poorly with user concerns in order to avoid the requirement for measurement system synchronization. The findings presented here should help to eliminate synchronization as an obstacle to performance measurement. For further information on time synchronization techniques, refer to Kamas and Howe (1979).

3. What accuracy can be achieved in the time-stamping of observed interface events? What mechanisms must be used to correct for

differences between the actual event times and the associated clock readings?

As discussed in Appendix D, serial transfer of the time from the clock receiver to the host computer reduces its accuracy somewhat, but the recorded event times are still accurate to within ± 5 milliseconds. That experimental error is much less than 1% of all but one of the performance times to be measured.[14]

Because the time information is transferred serially from the clock receiver to the host, there is a significant difference between the time at which an event actually occurs and the time at which a clock sample, or "time hack", marking that event is obtained. Fortunately, that difference is highly predictable. The mechanism used to correct for such differences is simply to add (or subtract) a constant to (or from) the sampled time, depending on whether the event follows or precedes the time hack. The correction times were (+120, -12 ms) at ITS and (+65, -7 ms) at NBS for events following and preceding the time hack, respectively. Refer to Appendix D for further information.

Two additional questions were asked with regard to the generation and recording of Interface Monitor outputs. These were:

1. How can the separate event histories recorded at each end of a data communication system be brought together in one computer for processing without introducing errors not present in the original recorded data?

The consolidation of remotely recorded event histories in one computer system for processing must be error-free if the processed measurement results are to be accurate. In the measurements reported here, this consolidation was accomplished routinely using the ARPANET File Transfer Protocol and the UNIX Compare utility command (Section 5.1.2). The procedure used in these experiments is equally applicable in many other test configurations. Briefly, a data file can be transferred from a remote site (say A) to a processing site (B) without error as follows:

1. FTP the file from A to B.
2. FTP the received file from B back to A.
3. Compare the two files at A. If the two compare exactly, the file was transferred correctly from A to B. If the two do not compare exactly, repeat.

[14]The one exception, source user Disengagement Time, is so small that its precise value is relatively unimportant.

This procedure fails only in the extremely unlikely event of exactly compensating errors during the two file transfers.

2. How much computer time is required to translate the "raw" event records into the specified standard output format? Should that translation be performed on-line (during actual data collection) or off-line (after the test)? How large, and how complicated, are the translation programs?

The translation of binary event records into ASCII output format was accomplished by four "C" language programs: a pair of time correction programs called TWEAK.H and TWEAK.O, which adjust the user information and overhead event times, respectively,[15] and the MERGE and REFORM programs. The time correction programs took about 30 minutes to process all the times recorded during a 160-session access/disengagement test, or about 2-3 minutes to process all the times recorded during a single user information transfer test. These times are about the same as the corresponding test execution times without processing. All the processing times could have been reduced by more than a factor of 10 if the ITS "C" complier had used hardware rather than software floating point instructions. The MERGE and REFORM execution times were negligible by comparison.

The translation of raw binary records into ASCII formatted records was accomplished off-line in this experiment for two reasons: (1) to minimize user influence on the measured values; and (2) to simplify the on-line test functions. Either on-line or off-line processing is feasible, but the latter probably results in a simpler overall test. None of the four Data Extraction processing programs used in this experiment is particularly complex. The TWEAK programs read successive event times from the raw data files, classify the times as leading or trailing, apply the appropriate correction factors, and write the corrected times into new files. The MERGE program appends corrected block transmission times to the corresponding recorded user information blocks. The REFORM program accomplishes the actual binary-to-ASCII file conversions. Each of the four programs is between 7 and 15K bytes in size. These program sizes could be reduced by 30%-40% through editing if necessary. That effort might well be justified if the processing were performed on-line.

---

[15]Separate TWEAK programs were actually used to process the ITS and ICST data, since the time correction factors were different in the two cases.

6.1.2 Data Files

The standard ASCII character files used to link the Data Extraction and Performance Assessment elements of the measurement system are described briefly in Section 2.4.2 of this report, and more fully in the measurement standard. As noted in Section 3.1.2 above, the implementation of these data files in the ARPA network measurements project was intended to answer three specific questions. These questions are restated and answered below.

1. Can the specified data file formats be directly written and read by software developed using different programming languages?

The problem of creating data files which are directly accessible to programs written in different languages is an important one. In the process of recording data in a logical record, a program typically delimits the data into several blocks, and appends various control words which define the record structure. Programs written in the same language as the recording program can read a record without difficulty, since they expect a record structure exactly like that created. However, this is often not the case when the recording program is written in a different language: even though the same computer and operating system are used, record structures typically differ between languages. The result is an inability to pass data directly between programs written in different languages.

This problem arose immediately in the ARPA network testing, since the Data Extraction programs were written in "C" and the Performance Assessment program was written in FORTRAN. An early version of proposed FED STD 1043 specified unformatted binary FORTRAN files as the input to the Performance Assessment program. Efforts to create such files in "C" proved difficult; and it was soon realized that even if this were successful in the present experiment, other users of the standard would face a similar problem. It was therefore concluded that the file formats used in passing performance data between the Data Extraction and Performance Assessment functions should be as nearly language and machine independent as possible. The approach chosen to achieve this objective was to transfer all performance data between these two measurement system elements in the form of ASCII character records.

As noted in Section 4.2, the process of transferring performance data from the "C" Data Extraction programs to the FORTRAN Performance Assessment program via ASCII character records proved to be very straightforward. The extent to which this common file structure is language and machine independent

was not determined in this experiment, but it seems likely that the chosen file structure will minimize translation problems in other applications as well. Virtually any computer system can create ASCII character records, and the file delimiters are specified as particular characters in the ASCII code (ANSI, 1967). Further tests of the transferability of the specified data files are planned.

2. What requirements will such standard files place on the memory capacity of the recording equipment? Are the memory requirements commensurate with the capacities of host/measurement systems which might reasonably be expected to implement the standard? How much is the memory requirement increased by the choice of a machine-independent data file format?

The memory capacity required to store the event histories recorded during a performance test depends, of course, on the number of trials observed. Four ASCII character files were generated during each test: an overhead information file and a user information file for each user/system interface. The ASCII overhead and user information files generated during a complete 160-session access/disengagement test were about 25,000 and 65,000 bytes in length, respectively. The corresponding overhead and user information file lengths for a single UIT test were 1,000 and 33,000 bytes. Files of this size will fit comfortably within the disk capacities of even relatively small computer systems, including portable microcomputer systems which might well be used as dedicated FED STD 1043 measurement instruments.

These data file sizes could be reduced substantially, if necessary, by including fewer sessions or less transmitted data in each test. This would not necessarily reduce the ultimate precision of the measured values, since the results of several tests can be aggregated off-line using the Performance Assessment program's carryover data files. Thus, it appears that data storage requirements will not significantly constrain the ability of users to conduct FED STD 1043 performance measurements.

As noted earlier, the problem of transferring data files between the Data Extraction and Performance Assessment functions was solved through the use of standard formatted ASCII character records. These formatted records store data files less efficiently than binary files, as was shown in Figure 16. The binary-to-ASCII translation process increases the user information file lengths by a factor of 2.67, since each group of fifteen UIT bits is mapped into five 8-bit ASCII characters. A similar expansion occurs in the case of

the overhead information files. This memory utilization cost appears to be well justified by the direct program-to-program transfer capability the ASCII files provide.

3. What is the impact of data aggregation (e.g., combining measurement results for many user pairs) on the data file structure? How important is the "carryover data" capability to users?

The provision of data aggregation in the Performance Assessment program has no effect whatsoever on the format of individual ASCII files. The data aggregation capability can substantially reduce the memory capacity required to record measurement data, since the same memory space can be used to store the results of many successive batches. To achieve this memory savings, the operator simply reduces the number of performance trials in each individual test and runs more tests, always processing the data from one test before initiating the next. Irrespective of test size, the ultimate test results are always reduced to a very small number of outcome values. These can be stored between batches and aggregated with subsequent results using very little memory. The carryover feature will be of particular significance to users with limited memory capacity.

6.1.3 Performance Assessment Program

The standard FORTRAN Performance Assessment program used to reduce the interface event histories to parameter values has been described in Section 2.4.3 of this report. As noted in Section 3.1.3, the application of that standard program in the ARPA network measurements project was intended to answer four specific questions. These questions are restated and answered below.

1. How large is the program, including both machine instructions and data? Will the program "fit" within the memory capacity of computer systems on which it might reasonably be implemented?

As discussed in Section 4.3, the Performance Assessment program used to reduce the ARPA network performance data actually consisted of three separately compiled programs linked by FORTRAN data files. The number of bytes in the compiled (object) version of each program is shown below.

Program	Object Code in Bytes
PROLOG	16,000
ASSESS	38,000
EPILOG	29,000

As noted earlier, these programs were executed sequentially in processing a data batch, and only one program needed to be in the computer memory at any time. Most modern computer systems can accommodate application programs up to at least 64K bytes in length, and thus the size of the Performance Assessment program should not be a deterrent to its widespread use.

2. How long does the program take to process a typical "batch" of recorded measurement data?

The Performance Assessment program took between 4 and 5 minutes to process a typical 160-session access/disengagement test, and about 7 minutes to process a typical 10,240-byte user information transfer test, on the PDP-11/40. The processing took about 10 minutes if both the access/disengagement and UIT parameters were calculated on the former type of test. This was rarely done in processing the ARPANET data because the access/disengagement and UIT parameters were evaluated in separate tests.

The printing of test results on the LP-11 added another 40 to 50 seconds in each case. Operator input of run parameters would add another 2 to 3 minutes if the parameters were different for each test, but this delay was avoided in most cases by storing fixed run parameters in a file. No operator involvement was required after test processing was initiated, since the PROLOG, ASSESS, and EPILOG programs were executed by a UNIX shell file.

3. Does the program correctly determine the outcomes of data communication attempts under all observed conditions of performance? Under what conditions does the program fail or give misleading results?

The ARPA network measurements data proved to be very useful in testing the Performance Assessment program logic. Essentially all of the extracted performance data was input to that program, and the correctness of the output was tested in two independent ways: first, by comparing that output with the results of an operator analysis of the raw performance data; and second, by comparing that output with the results of the supplementary data reduction. With two exceptions, discussed below, the program functioned exactly as expected.

The first Performance Assessment program failure observed during the ARPA network measurements occurred during the processing of a group of three very unusual data communication sessions recorded in test numbers 78 and 79. In each of these three sessions, the source user program (at ITS) opened a connection, transmitted a user information block, and closed the connection in the normal manner; but only one event occurred at the destination user interface - the destination program's original Open (any host) request. That request was never completed, and the destination user program therefore had no opportunity to participate in the session. From the point of view of that program, it was as if the session never took place. Naturally, the transmitted data was never received.

The difficulty in processing these sessions arose when the Performance Assessment program attempted to determine the destination user disengagement time. As defined in both Interim FED STD 1033 and X3.102, that time begins when the first Close request is issued (in this case, by the source user). It ends when the system issues a Close Complete signal to the destination user. The problem, of course, is that the latter event does not occur with the session being examined. It does ultimately occur, however - in the next normal session. The Performance Assessment program's original disengagement subroutine did not restrict its search for the disengagement confirmation signal to one session. It therefore erroneously calculated the destination user disengagement time for the faulty session on the basis of the Disengagement Confirmation signal in the next session! The problem was obvious once it occurred, since the erroneous disengagement time was more than five times longer than any other recorded. The solution was equally obvious - adding a check for session boundaries to the disengagement subroutine.

The original Performance Assessment program also proved to be deficient in identifying the precise beginning of a string of undelivered (or extra) bits. This function was originally accomplished in the program's DATCOR subroutine, which worked by comparing the number of matching transmitted and received bits in a correlator "window" with a threshold (Seitz et al., 1981). That subroutine correctly identified the beginning of an undelivered (or extra) bit string whenever the first bit in that string differed from the next subsequent received (or transmitted) bit; but displaced the beginning of such strings by one or more bits if the string bits and the subsequent bits fortuitously matched. The program computed the correct number of undelivered

or extra bits in every case, but sometimes transformed a single undelivered or extra block into two incorrect blocks as a result of the displacement.

This deficiency in the original DATCOR subroutine was resolved by replacing that subroutine with two operator-selectable subroutines, each appropriate in a particular type of measurement application. The first, termed BITCOR, matches transmitted and received bits without reference to block boundaries. That subroutine is used in applications where the source block boundaries need not be preserved in delivering information to the destination.[16] The block-oriented UIT parameters are far less important in such cases. The second subroutine, termed BLKCOR, is still under development. That subroutine will match transmitted and received bits on the basis of block delimiters provided by the source user and the system, and will be used in applications where the source block boundaries must be preserved. It will eliminate the outcome displacement problem by performing data comparison and outcome decision-making on a block-by-block basis.

Although the ARPA network application provided a reasonably comprehensive test of the Performance Assessment program, it was not possible to verify all of the program logic using the ARPANET data because certain possible failures were never observed. Where such deficiencies were identified, the logic in question was tested separately via simulated measurement data. All possible access and disengagement performance outcomes were simulated and correctly detected in this supplementary testing. The UIT processing subroutines were tested in a similar manner, but additional testing will be required to assess those subroutines' processing of tightly clustered UIT failures (e.g., Lost Bits and Incorrect Bits within two bytes of each other).

4. What flexibility must such a program provide to users with respect to (a) aggregation of measurement data from many batches, (b) calculation of parameter subsets, and (c) adaptation of program variables to measured system characteristics?

The benefit of data aggregation in memory-limited applications has been discussed earlier, in answer to question 3 in Section 6.1.2. The carryover feature also proved to be of value during the ARPA network measurements for another reason: the need to cluster performance data from different batches a posteriori, based on the observed test results. This need arose first,

---

[16] As noted earlier, the end-to-end ARPANET service measured in this experiment had that property.

because the influence of test conditions on the measured values could not be predicted in advance; and second, because there were cases in which it was necessary to exclude data batches from the test results due to measurement system failures. The ability to combine performance data from separate batches appears to be essential in all but the most rudimentary measurement applications.

The ability to calculate parameter subsets proved to be useful, but not essential, during the ARPA network measurements. As noted earlier, access/disengagement and UIT performance were evaluated separately, using different test procedures during those measurements; and the ability to consider only the parameters of interest in reducing the data saved both computer and operator time. That savings would be much more significant in processing very lengthy tests. The ability to enable or inhibit the calculation of parameters by primary function (access, user information transfer, disengagement) proved to be more useful than the ability to enable or inhibit calculation by parameter type (primary, secondary, ancillary).

The key program variables which must be input by the Performance Assessment program operator are (1) the specified or expected performance times for the access, block transfer, and disengagement functions; (2) the corresponding specified user fractions; (3) the data correlator window size, correspondence threshold, and maximum bit shift constants; and (4) the outage or Transfer Denial Thresholds. The ability to easily vary the specified performance times and user fractions proved to be essential in reducing the ARPA network performance data. The reason was that the appropriate values for these input variables were not known a priori; and yet those values must be specified with reasonable accuracy in order to properly reduce the performance data. The specified access, block transfer, and disengagement times are used in determining performance timeouts, as discussed in Section 2.2. The specified user fractions of these times are used in assigning "responsibility" for performance failures to the users or the system.

This apparent impasse was easily resolved by processing the first few performance data batches twice: once to determine the appropriate input variables, and once to calculate final performance values based on those variables. As an example, both Interim FED STD 1033 and ANSI X3.102 define an access attempt to be a failure if its total performance time exceeds three times a "specified" Access Time value. The specified value is defined as the mean of the Access Time distribution <u>with no truncation</u>. To determine this

mean, the first few access/disengagement tests were processed with a "specified" value so large that no timeouts occurred. The same data was then re-processed with the specified input value equal to the calculated mean. This two-step processing will not be required in most operational performance measurements, since appropriate "specified" performance values will be available either from previous measurements or from the system requirements.

6.1.4 Statistical Design and Analysis Procedures

The Statistical Design and Analysis procedures used in conducting the ARPA network measurements are described briefly in Section 2.4.4 of this report, and more fully in the measurement standard. As noted in Section 3.1.4 above, the application of these procedures in the ARPA network measurements project was intended to answer four specific questions. These questions are restated and answered below.

1. How should a data communications user's measurement precision objectives be determined? Can/should FED STD 1043 provide guidance to users in establishing such objectives?

The determination of measurement precision objectives is not a simple task, nor is it a task that can be reduced to any one "standard" procedure. However, the engineers and mathematicians who participated in the ARPA network measurements project agreed that at least some additional guidance on the determination of measurement precision objectives should be added to proposed FED STD 1043. That guidance should do two things. First, it should encourage standard users to base the determination of measurement precision objectives on a serious examination of the relationship between the precision of a measured parameter value and its usefulness in service selection or operation. Such an examination would lead, in many cases, to less stringent measurement precision objectives and more economical measurements. Second, it should point out the practical necessity of choosing a single sample size for measuring parameters that are necessarily measured together (e.g., the various access parameters). Guidelines for clustering parameter measurements based on sample size requirements would also be useful. Guidelines covering these principles will be incorporated in a future revision of the measurement standard.

2. How strongly do underlying measurement conditions (e.g., time of day, direction of transfer) influence the measured parameter values? Does the current FED STD 1043 draft provide

adequate guidance to users in identifying such conditions and assessing their impact?

As discussed in Section 5, the ARPA network measurements were designed to assess the network performance with respect to three primary measurement variables: direction of transfer (east-to-west or west-to-east), user program priority (high or normal), and time of day (peak hours or off hours). The test results indicated that each of these variables had a measurable effect on the observed end-to-end performance.[17] No other significant measurement variables were identified. The guidance on the identification of measurement conditions in proposed FED STD 1043 was judged to be generally adequate, but it was felt that a more comprehensive itemization of network and user conditions that may affect data communication performance would be helpful to users. Such an itemization will be added to the standard in a future revision.

3. Given a specified measurement precision objective, is it possible to determine the sample size required to achieve that objective in a straightforward way?

The ARPANET experiment indicated that calculation of sample size requirements from measurement precision objectives using proposed FED STD 1043 is very straightforward. What is less straightforward is the determination of realistic measurement precision objectives. As suggested in answer to question (1) above, the first step in determining such objectives is to establish the relationship between the precision of the measured values and the usefulness of those values in communication management decision making. The proposed FED STD 1043 sample size determination procedures provide the essential second step: relating measurement precision to measurement cost.

4. Can the post-test data analysis procedures and formulas provided in the current FED STD 1043 draft be applied to the ARPA network test data with reasonable effort? How much do the precision estimates calculated on the basis of the observed test results differ from the objectives established before the test? How realistic were the statistical assumptions on which the pretest calculations were based?

No particular problems were encountered in applying the proposed FED STD 1043 statistical analysis procedures to the ARPANET measurement results. As

---

[17] The effects on specific parameters (and the probable reasons for these effects) are discussed in Section 6.2.

discussed in answer to question (3) above, the measurement precision objectives established at the outset of the test proved to be much too stringent in the case of several parameters; and the desired and actual confidence limits differ correspondingly in those cases. As discussed in answer to question (1) above, similar difficulties could be avoided in future applications of the standard by a more careful development of the measurement precision objectives.

Perhaps the most significant statistical assumption underlying the pre-test calculations of confidence limits was the assumption of first-order Markov dependence between performance times. As discussed in Section 4.4, that assumption was checked in the case of the ARPANET access times. The results indicated that higher order dependence does exist, but has relatively little impact on the precision estimates. Thus, the Markov assumption appears to be reasonable in the case studied.

6.2 Secondary Results

As noted in Section 3, a secondary objective of the ARPA network measurements project was to obtain some typical performance values to characterize the data communication service provided by the ARPA network and its host computers to end users. It was anticipated that these values would be useful first, in understanding differences between subnetwork and user-perceived performance; and second, in assessing proposed refinements in the performance parameter definitions.

This subsection summarizes the ARPA network measurements project results that relate to that objective. The same question/answer format employed in the preceding subsection is used here. As in Section 3, the questions are grouped in three categories, addressing the access, user information transfer, and disengagement parameters successively. Within each category, questions related to network performance are addressed first, followed by questions related to the parameter definitions. The various questions are discussed more extensively here than in the preceding subsection, since the measured performance results have not been presented in earlier sections.

Table 3 summarizes the performance measurement results in qualitative terms. As would be expected in measuring a modern packet-switching network like the ARPANET, the majority of the tests ran faultlessly, with no deviation from the normal flow path of Figure 25. What is shown in Table 3 are the relatively few exceptions. The measured values for the 1033/102 accuracy and

IDENTIYING NUMBER	TEST NUMBER(S)	NATURE OF ANOMALY	CAUSE (IF KNOWN)	REMEDIAL ACTION (IF ANY)	PERFORMANCE EFFECT
1	77	DOCB TIP "crashed" during an A/D test sequence.	Routine TIP maintenance procedures.	TIP restart. See next item.	Access Denial (timeout).
2	78/79	Systematic disappearance of first 1 or 2 blocks transmitted during an A/D test sequence.	NTIA/ITS NCP altered by TIP "crash" during previous test.	Host software restart.	Block Loss.
3	94	Systematic disappearance of all transmitted blocks beginning at the middle of an A/D test sequence.	Probably a host software failure - NBS system "crashed" soon after test.	Host software restart.	Block Loss (all blocks transmitted between failure and test termination).
4	95	Destination host refused connection (and source program received a denial response) after 67 trials in an A/D test sequence.	Destination NCP "crashed."	Host software restart.	Access Denial (system blocking signal).
5	97	Random loss of blocks during an A/D test sequence.	Unknown. Destination host was under very heavy local use.	None. Problem did not persist.	Block Loss (three random blocks in a 160-block sequence).
6	97	12-minute access time observed on one trial in a 160-trial A/D sequence. Access ultimately established. Remaining access times normal.	Unknown. Destination host was under very heavy local use.	None. Problem did not persist.	Access Denial (timeout).
7	Not Assigned	Host down at initiation of test.	Various (e.g., hardware failures, software "crashes").	Host hardware maintenance or software restart.	Access Outage. This was a very common occurrence.
8	Not Assigned	TIP down at initiation of test.	TIP program failure.	TIP restart (via remote action).	Access Outage.
9	Not Assigned	Host and TIP disconnected at initiation of test.	TIP operator error.	Reconnect TIP/host cable.	Access Outage.
10	All UIT TESTS	First few blocks input by source are delivered to destination in two pieces (512 bytes →64 bytes + 448 bytes).	Unknown. May be a characteristic of the host flow control protocol.	None. Delivered bit sequence is correct; only block delimiters are changed.	Longer Block Transfer Times for first few blocks transmitted during UIT tests.

Table 3. Summary of Observed Anomalies

reliability parameters were largely determined by these anomalies. Each anomaly is discussed more fully below, in conjunction with the particular parameter value(s) it influenced.

6.2.1 Access Parameters

The three questions posed in Section 3.2.1 regarding ARPA network access performance are restated and answered below.

1. How long must an ARPA network end user wait, after requesting communication service, before his first block of information is actually delivered to the system for transmission? How variable is that delay? What proportion of the delay is attributable to delays introduced by the users? What proportion is attributable to subnetwork packet transmissions?

The first of these questions is answered by the measured value of the standard performance parameter Access Time. The measured value of that parameter, averaged over all 2993 Successful Access outcomes observed during the 5 weeks of ARPA network access/disengagement testing, was 1.8 seconds. The Access Time measurement precision objective specified in Section 4.4 was achieved, and thus the upper and lower 95% confidence limits on this estimate are 1.9 and 1.7 seconds, respectively.

The variability of the measured access times is further illustrated in the series of five histograms shown in Figures 31 through 35.[18] Figure 31 shows the distribution of measured access times for 3049 observed trials. The smallest access time observed was 602 milliseconds; the largest was 24.1 seconds. (An astounding 728.6-second [12.1 minute] access time was observed on one trial, but was discarded as an obvious anomaly.) A total of 14 access times fell between 6 and 25 seconds. The overall standard deviation of the measured access times was 1.0 second.

Figures 32 through 35 provide a basis for assessing the effects of direction of transfer, user program priority, and time of day on the measured access times. The observed effect of direction of transfer on access time is illustrated by Figures 32 and 33. There is a measured difference in mean Access Time of about 0.1 second between attempts initiated at NBS/ICST and

---

[18]These histograms include all measured values which are graphically representable. They include 17 trials which exceeded the "3 times nominal" timeout and 39 trials which were not included in the sample used in calculating the access parameters because a clock error occurred at the destination.

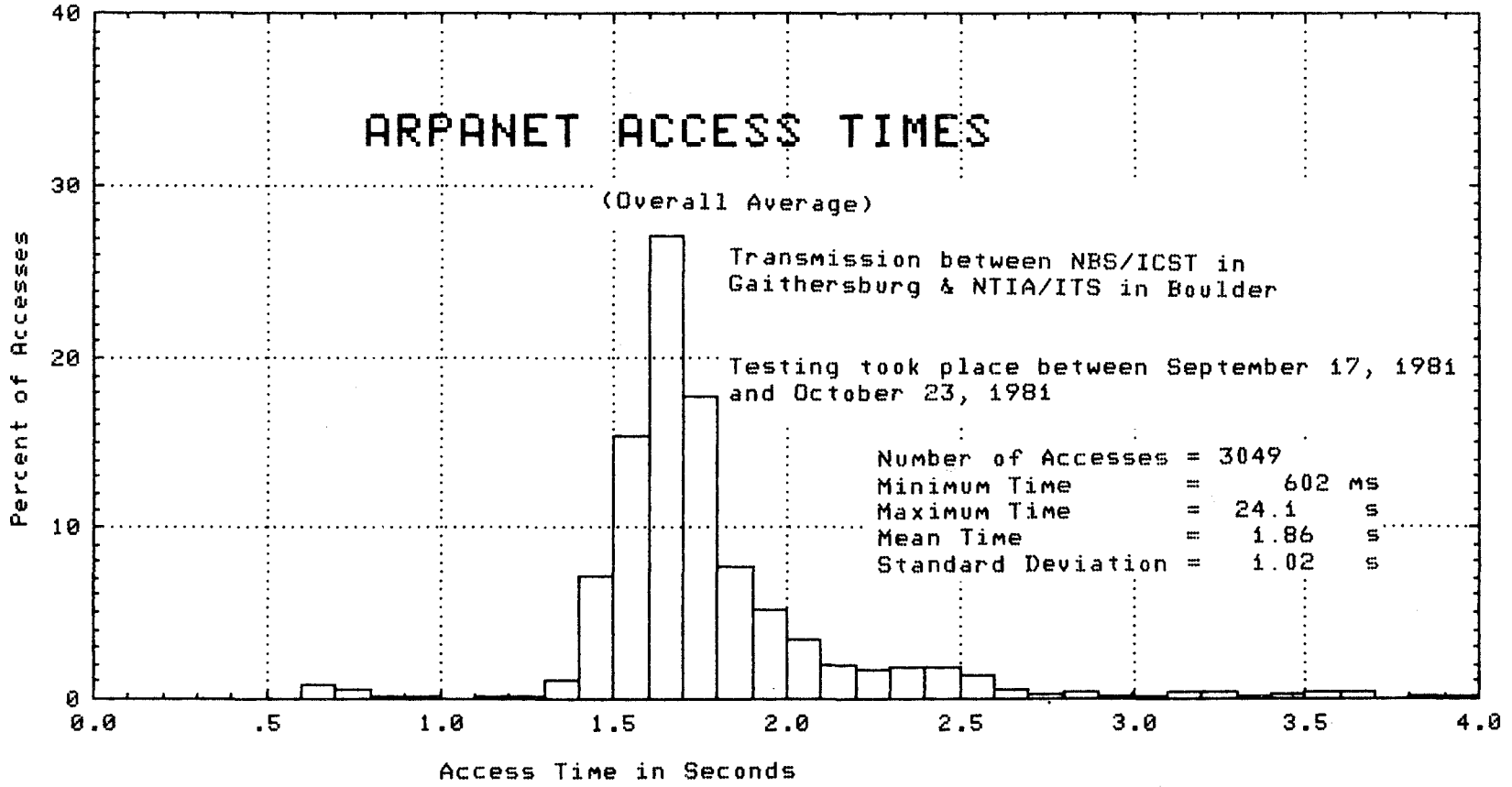


Figure 31. Histogram of access times - overall average.

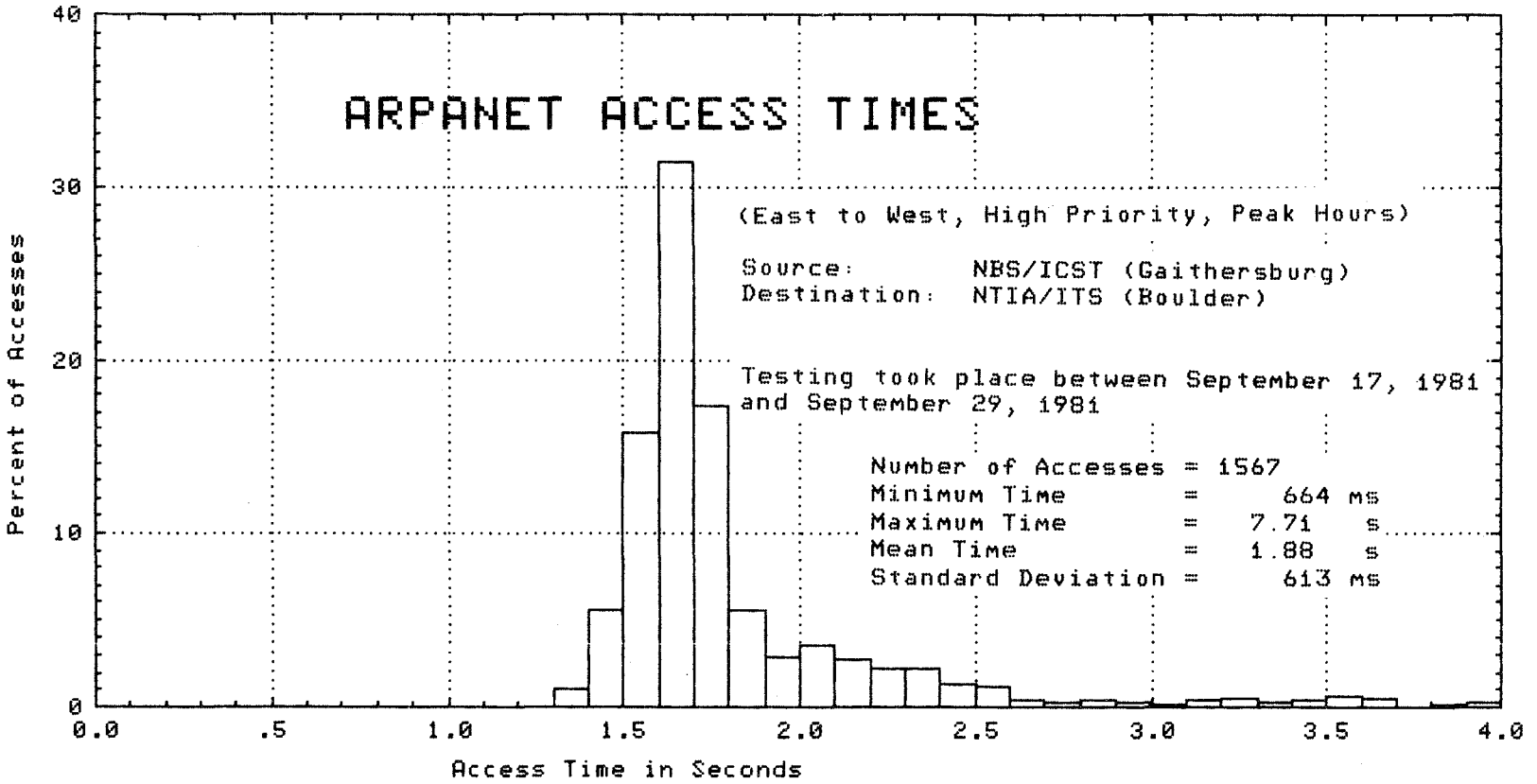


Figure 32. Histogram of access times - east to west, high priority, peak hours.

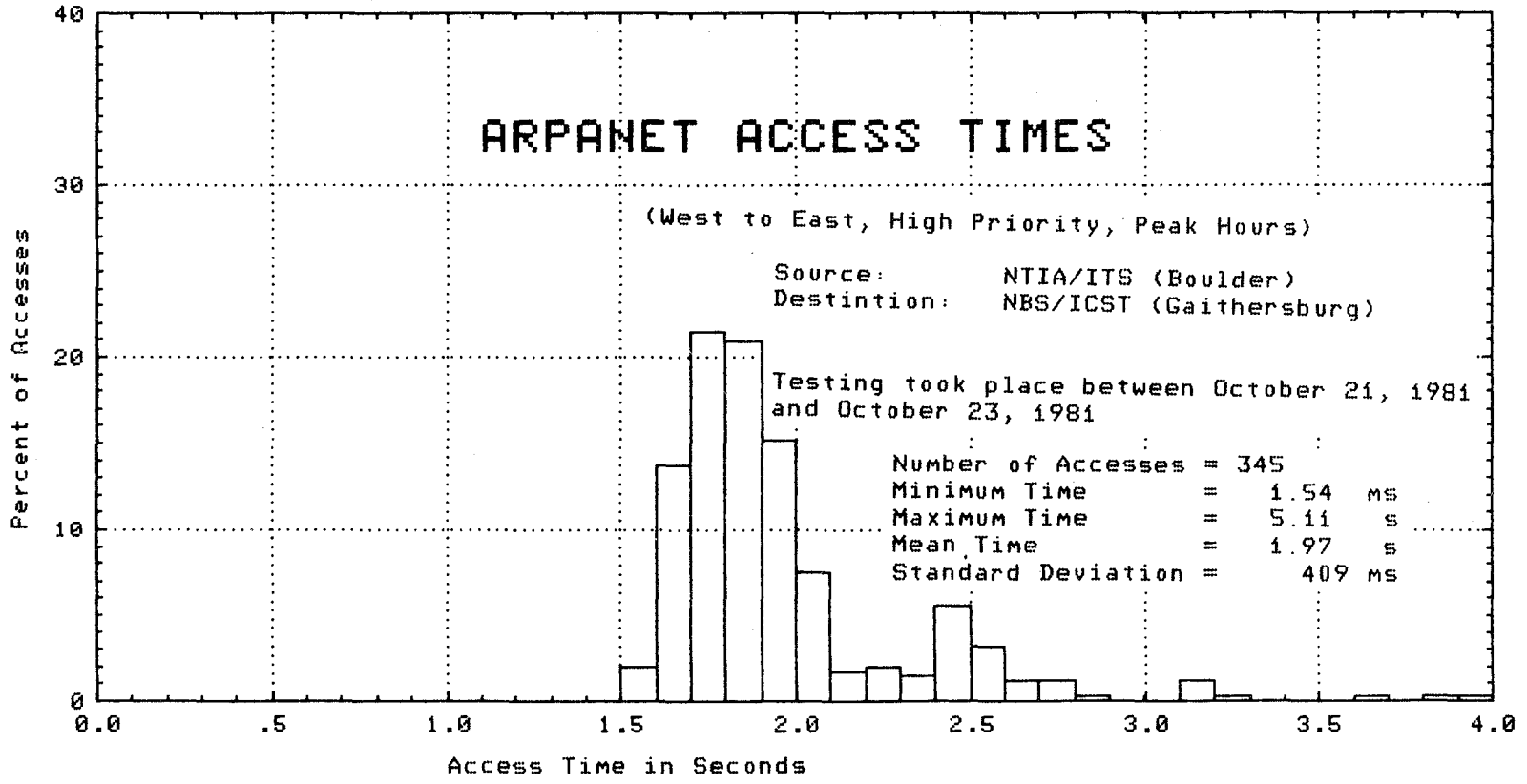


Figure 33. Histogram of access times - west to east, high priority, peak hours.

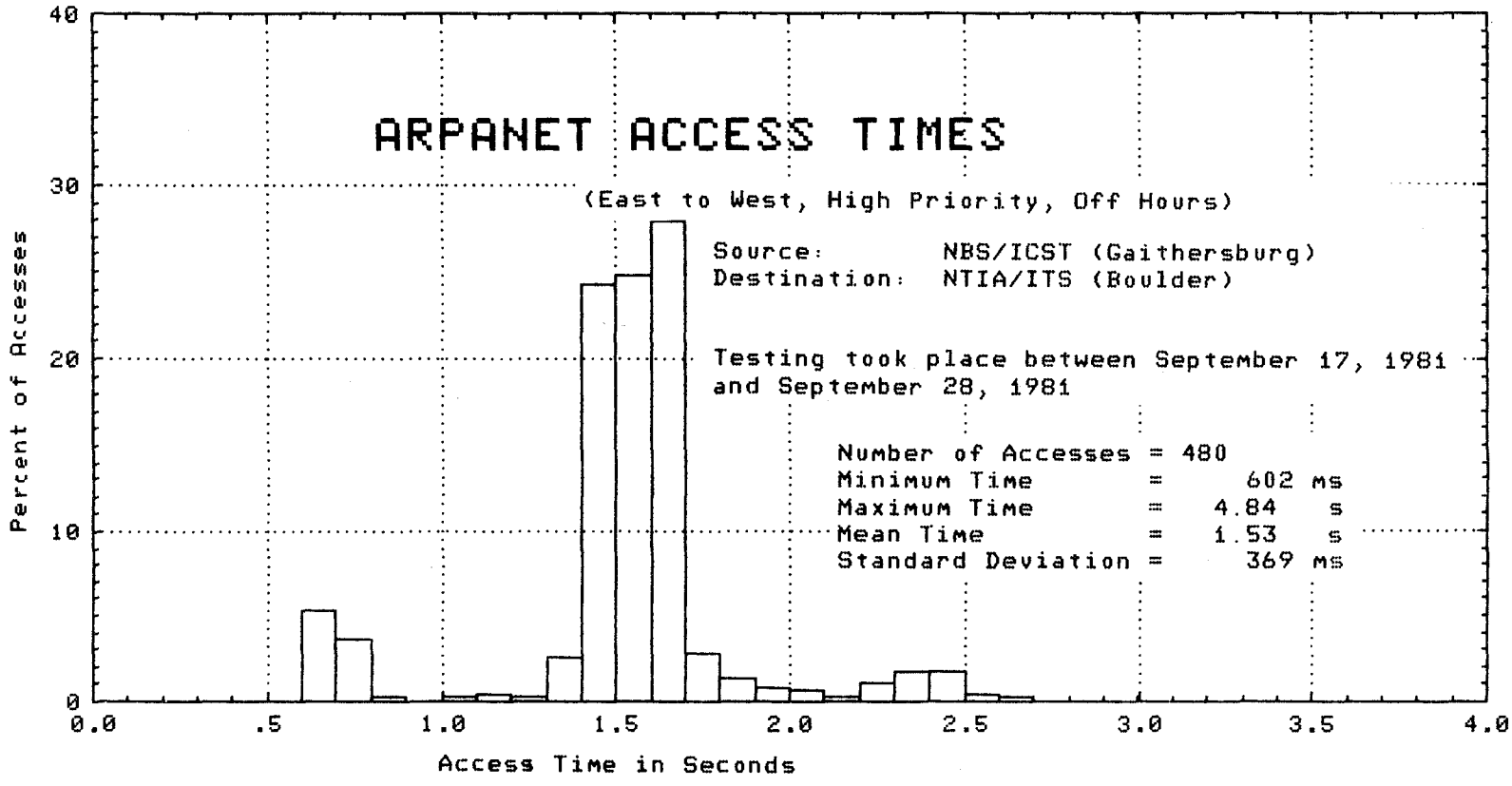


Figure 34. Histogram of access times - east to west, high priority, off hours.

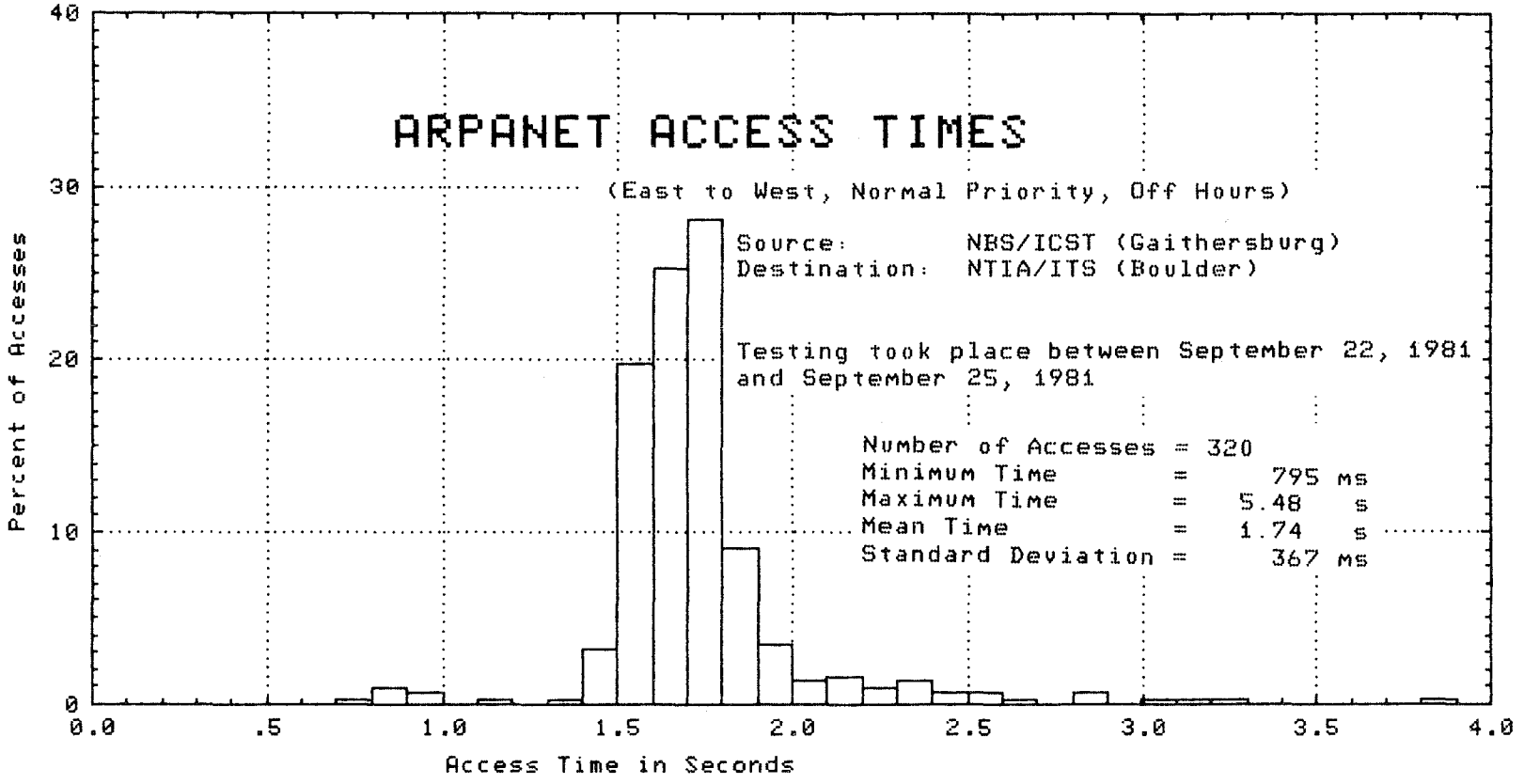


Figure 35. Histogram of access times - east to west, normal priority, off hours.

attempts initiated at NTIA/ITS, but this is attributable to the difference in clock read times at the two sites (Appendix D).

The observed effect of user program priority on Access Time is illustrated by Figures 34 and 35. As one would expect, the mean Access Time measured under high user program priority was shorter than that measured under normal priority. The observed difference is about 0.2 second. However, the access time variation among tests having the same measurement conditions was sufficiently large that this difference could not be declared statistically significant.

The effect of time of day on Access Time is illustrated by Figures 32 and 34. The mean Access Time during peak hours was about 0.4 second longer than the corresponding off hours value. This difference, which was statistically significant, undoubtedly reflects a substantial user component, although variations in subnetwork utilization between peak and off hours probably also contributed (Kleinrock et al., 1976).

The average proportion of Access Time that is attributable to delays introduced by the transmitting and receiving end users (host computer application programs) is indicated by the measured value of the ancillary parameter User Fraction of Access Time. That value, averaged over all 2993 Successful Access outcomes observed during the testing, was 0.15. Thus, the average user component of Access Time was 0.15 x 1.8 = 0.3 seconds, and the average system component of Access Time was 0.85 x 1.8 = 1.5 seconds. In the application tested, most of the user time was occupied by clock readings (144 and 264 milliseconds per access for NBS/ICST and NTIA/ITS originated attempts, respectively). The user programs would not be instrumented in actual operational data communication sessions, of course, but they would still introduce delays as a result of their processing of an access request. Such processing functions might include, for example, user resource allocation, user ID verification, and quality of service negotiation.

Decomposing the measured Access Time values into subnetwork and host components was not really an objective of this experiment, but a rough estimate of these components can be developed as follows. Beginning with the 1.8 second average Access Time, we first remove the end user (application program) component (0.3 second) using the ancillary parameter as noted above. The resulting 1.5 second user-independent Access Time is composed of two subnetwork packet transit delays (one for the connection request and one for

the response) and two host computer processing delays (one to generate the response and one to generate the first WRITE request).

Consider first the average contribution of the two subnetwork packet transit delays. The typical transit delay for a single-packet message across an 8-hop ARPA subnetwork path is less than 250 ms. One-half second is thus a reasonable, although rough, upper bound on the connection request and response subnetwork transit delays. This leaves 1 second as the probable contribution of the two host computer operating systems and NCP's. A substantial portion of that delay is probably a result of time-sharing interrupts.

About midway through the testing, a statistical analysis of the collected Access Time data was conducted to test earlier dependence assumptions and refine the sample size requirements. Major steps in this analysis are summarized below, as a demonstration of use of the FED STD 1043 Statistical Design and Analysis procedures.

The measured Access Times were analyzed in accordance with A.4.3.2 of proposed FED STD 1043 and Crow (1979), the second reference providing the formulas for taking account of the substantial intergroup differences between samples separated in time. As noted earlier, these differences led to determining that 19 samples of 160 trials each would be required to estimate the mean access time within $\pm$ 0.1 second with 95% confidence. At the mid-point in the planned test interval, nine such samples had been acquired and assessed; and it was therefore assumed that the specified precision had not, as yet, been attained.

To the four preliminary samples used in the sample size determination (Section 4.4) we adjoined the five further samples available:

$\bar{w}$	1.665	1.897	1.566	1.397	1.932	Mean = 1.691 seconds
s	0.239	0.742	0.147	0.333	0.455	RMS = 0.435 seconds
r_1	0.009	0.004	0.028	0.180	0.228	Mean = 0.090

The means and RMS internal variance for all nine samples were thus

1.728 seconds, 0.497 seconds, 0.104 .

(Very slight differences arose from rounding at different states of calculation.)

The variance of the nine means was $(0.2120)^2 = 0.0450$, little different from that found previously for the first four means. Following the previous calculations, we have all $n_i = 160$, $m = 9$, $s_e^2 = 0.247$,

$$s_a^2 = \frac{8}{1440-160}\,(160 \times 0.0450 - 0.247) = 0.0434 \quad ,$$

$$s_{\bar{x}}^2 = \frac{0.0434}{9} + \frac{0.247}{1440} = 0.00482 + 0.0017 = 0.00499 \quad ,$$

$$2s_{\bar{x}} = 0.141 \quad .$$

Since the internal variance contributed only 3% of $s_{\bar{x}}^2$, it mattered little whether the slight increase in it due to the autocorrelation was included. The factor $(1 + r_1)/(1 - r_1) - 1.232$ increased that term to 0.00021, and $2s_{\bar{x}}$ to 0.142. Hence, the 95% confidence limits for Access Time, based on the first nine 160-access samples, were

$$1.728 \pm 0.142 = \begin{matrix} 1.870 \\ 1.586 \end{matrix} \text{ seconds} \quad .$$

As predicted, the prescribed accuracy of 0.1 second was not attained with nine samples, and the necessity of about 19 (perhaps 18) samples to attain that accuracy was confirmed.

2. What is the likelihood that transmitted user information will be directed towards a destination other than the one intended as a result of a system connection establishment error?

This question is answered by the measured value for the standard performance parameter Incorrect Access Probability. No Incorrect Access outcomes occurred in the 3019 access attempts observed during this experiment, so that the Incorrect Access Probability is estimated as zero. Based on the assumption that successive Incorrect Access outcomes are independent, which seems to be as reasonable an assumption as any based on the data collected, the 95% upper confidence limit for Incorrect Access Probability is 1×10^{-3} (Crow, 1974).

3. What is the likelihood that the system will fail to give the user access to communication service (within a specified maximum time) on any given request? How frequently are such failures attributable to system outages? How sensitive is that likelihood to host computer utilization?

The first two questions are answered by the measured values for the standard performance parameters Access Denial Probability and Access Outage Probability. As discussed in Section 2.3, the definitions of these parameters in Interim FED STD 1033 and X3.102 differ in that the former standard treats Access Outages as Access Denials, whereas the latter treats Access Outages as separate outcomes. As noted earlier, 3019 total access attempts were observed

during the experiment. Of these, 17 failed as a result of access timeout; 1 failed as a result of issuance of a system blocking signal; and 8 failed as a result of Access Outage (e.g., network or host observably down at the outset of the test). The Interim FED STD 1033 Access Denial Probability value is thus $(17 + 1 + 8)/3019 = 8.6 \times 10^{-3}$; the X3.102 Access Denial Probability value is $(17 + 1)/3019 = 6.0 \times 10^{-3}$; and the X3.102 Access Outage Probability value is $8/3019 = 2.6 \times 10^{-3}$. Thus, a little less than one-third of the observed access failures were attributable to system outages.

The upper and lower 90% confidence limits for these estimates are summarized below:

Parameter	Upper 90% Confidence Limit	Lower 90% Confidence Limit
FED STD 1033 Access Denial Probability	1.2×10^{-2}	6.0×10^{-3}
X3.102 Access Denial Probability	8.8×10^{-3}	3.8×10^{-3}
X3.102 Access Outage Probability	4.8×10^{-3}	1.3×10^{-3}

Of the three types of access failures discussed above, the timeout failures are undoubtedly the most sensitive to host computer utilization. Of the 17 timeout failures observed during the access/disengagement tests, all but 1 occurred during peak hours. The seventeenth timeout failure was apparently the result of an unusually long subnetwork transit delay.

The ARPA network access measurements were also intended to answer four questions regarding the detailed definition of the access performance parameters. These questions are restated and answered below.

1. Are the observed values for Access Time substantially influenced by the choice of "start of block transfer" rather than "connection confirmation" as the end of the access function? Does the ancillary parameter User Fraction of Access Time provide an adequate method of factoring out that influence where user-independent parameter values are desired?

The measured difference between these two possible "end of access" events was about 175 milliseconds for access attempts initiated at NBS/ICST, and 300 milliseconds for access attempts initiated at NTIA/ITS. In each case, the majority of the time is occupied in reading the satellite clock.

The delay between these two access ending events is less significant given the existence of the ancillary parameter User Fraction of Access Time, since that parameter can be used to factor out such user-dependent delays. User Fraction of Access Time and the other ancillary parameters proved to be effective and useful in analyzing the ARPA network measurement results, particularly where comparisons between end-to-end and subnetwork performance were undertaken.

2. Is Incorrect Access Probability practical to measure as defined in the current standards? Is the distinction between Incorrect Access Probability and Block Misdelivery Probability significant from a measurement standpoint?

The answer to the first question depends on one's definition of "measure". If that definition implies that a probability estimate obtained by measurement must be greater than zero, then Incorrect Access Probability will be difficult to "measure" in data communication systems with effective end-to-end error control. The standards tested in this experiment are based on the opposite view, that a probability estimate of zero is a valid measurement result as long as the sample size, and the associated upper confidence limit, are stated. The ARPANET experience demonstrated that the instrumentation effort required to estimate Incorrect Access Probability is straightforward in any case.

The distinction between Incorrect Access Probability and Block Misdelivery Probability is significant from a measurement standpoint in that the latter measurement requires the provision of dual Interface Monitors, with similar instrumentation, synchronization, and storage capabilities, at one or both user sites. This subject is addressed more fully in the discussion of Block Misdelivery Probability in Section 6.2.2.

3. How strongly is the value for Access Denial Probability influenced by the choice of maximum access time? Would a different maximum access time value, or a different algorithm for determining that value, be more appropriate than the one selected?

The maximum access time (or "access timeout") value influences Access Denial Probability very directly, since it establishes a measured delay beyond which an access attempt is declared to be a failure for performance assessment purposes. In essence, the timeout value truncates the access time distribution at a value equal to three times the pre-truncation mean; and any access attempt whose performance time exceeds that truncation point (as a

result of system delays) is declared to be an Access Denial. Clearly, a larger timeout value would cause fewer access attempts in the "tail" of the Access Time distribution to be classified as Access Denials, and vice versa.[19]

The specific effect of the access timeout value on Access Denial Probability depends on two factors:

1. The frequency of Access Denial outcomes attributable to causes other than timeout. The possible other causes of Access Denial are System Blocking signals and, in the FED STD 1033 case, Access Outages. The more such non-timeout denials there are in a measurement sample, the less influence the timeout threshold has on Access Denial Probability; and vice versa.

2. The shape of the Access Time distribution. The more peaked that distribution is, the less influence the timeout threshold has on Access Denial Probability; and vice versa.

The relative numbers of timeout, system blocking, and Access Outage outcomes observed during the ARPA network measurements have been stated earlier: i.e., 17, 1, and 8 outcomes, respectively. Thus, timeouts account for about two-thirds of the observed Access Denials under the FED STD 1033 definition, and 94% of the observed Access Denials under the X3.102 definition.

The choice of a timeout threshold for defining performance failures is inevitably somewhat arbitrary. The constant 3 was chosen in the belief that it would (1) be generally consistent with user expectations about service performance, and (2) include most of the area under a typical delay time distribution. Crow (1979) analyzed the effect of truncating various delay time distributions at 3 times the mean, and related this data to typical communication delay distributions available at that time. He pointed out that some form of truncation is necessary in conducting practical measurements,[20] and concluded that the "3 times nominal" truncation rule appears to be as reasonable as any other based on the distributions studied.

---

[19]The truncation point also affects Access Time, since only Successful Access outcomes are included in that average. However, the effect on Access Time is normally much less significant than the effect on Access Denial Probability.

[20]The need for some truncation point is clearly illustrated by the 12.1-minute access time observed in these measurements. Inclusion of that one point in the Access Time distribution would increase its standard deviation from 1.0 to 13.2 seconds!

4. ANSI X3.102 differs from Interim FED STD 1033 in that it distinguishes Access Outage (no system response) from Access Denial (negative system response or excessive system delay). Are these two outcomes readily distinguishable in the case studied? Is that distinction meaningful and important in the case studied?

The relative numbers of outage and non-outage Access Failures observed during the ARPA network tests have been presented earlier. To summarize briefly, the exclusion of Access Outage outcomes in calculating Access Denial Probability reduces the value of that parameter by about 30%, from 8.6×10^{-3} to 6.0×10^{-3}. By itself, that difference would not justify the separation of these outcomes and the definition of a new performance parameter. However, a second reason for distinguishing Access Outage from Access Denial was that the appropriate user actions are often different in the two cases. In the former case, maintenance action is normally required; in the latter case, the best user response often is simply to re-attempt access. This distinction proved to be a very accurate and practical one in the ARPA network application; and on the basis of that experience, the ANSI approach is clearly preferable.

6.2.2 User Information Transfer (and Availability) Parameters

The seven questions posed in Section 3.2.2 regarding ARPA network user information transfer and availability performance are restated and answered below.

1. What total delay does a bit or block of user information experience, on average, between a request for its transmission by the source application program and the completion of its delivery to the destination program? How is that delay influenced by block length? How variable is that delay, for a given block length? Does user performance contribute significantly to the observed end-to-end delays in the case studied? How do the measured Block Transfer times compare with previously measured "average message delays" for the subnet?

The average delays experienced by user information bits and blocks in transit between ARPANET host application programs are expressed by the standard performance parameters Bit Transfer Time and Block Transfer Time. The values for these two parameters are virtually identical in the case studied, because each block is passed across the application program/operating

system interface as a unit rather than as a series of bits.[21] Only the latter parameter is discussed here. Values are presented for two block lengths: the 512-byte (4096-bit) block transferred during the user information transfer tests, and the 64-byte (512-bit) block transferred during the access/disengagement tests.

The measured value of Block Transfer Time for 512-byte blocks, averaged over all 428 512-byte blocks successfully transferred during the 4 weeks of ARPA network UIT testing, was 709 milliseconds. The 95% upper and lower confidence limits on this value, assuming independence between successive trials, are 739 and 679 milliseconds, respectively. The corresponding average value for the 2996 64-byte blocks successfully transferred during the access/disengagement testing was 262 milliseconds ($\pm$ 10 ms). The difference between these values is primarily a result of differences in the transmission and flow control protocols for single-packet and multi-packet messages. The 64-byte blocks fit easily into a single packet, whereas the 512-byte blocks were transmitted as five separate packets.

Figures 36 and 37 show the distributions of measured block transfer times for the 512-byte and 64-byte blocks, respectively. (Again, for completeness, these histograms are not truncated at three times the mean.) The 512-byte block transfer times range between 569 milliseconds and 4.0 seconds, with a mean of 757 milliseconds and a standard deviation of 386 milliseconds. The 64-byte block transfer times range between 216 milliseconds and 2.9 seconds, with a mean of 264 milliseconds and a standard deviation of 70 milliseconds.

The variations of Block Transfer Time with direction of transfer and system utilization were qualitatively similar to those observed in the case of Access Time. The choice of high or normal user program priority should have no effect on Block Transfer Time since there are no user delays during that time (Figure 25).

A rough comparison of end-to-end and subnetwork Block Transfer Times can be derived from the subnetwork data shown in Figure 38 (adapted from Kleinrock, 1976). Figure 38a shows average and minimum subnetwork "round-trip delay" as a function of hop distance. Figure 38b presents histograms of round-trip delay for 1-, 5-, and 9-hop paths. Kleinrock defines "round-trip delay" as follows:

---

[21] An occasional exception was observed, as identified in Table 3, anomaly No. 10.

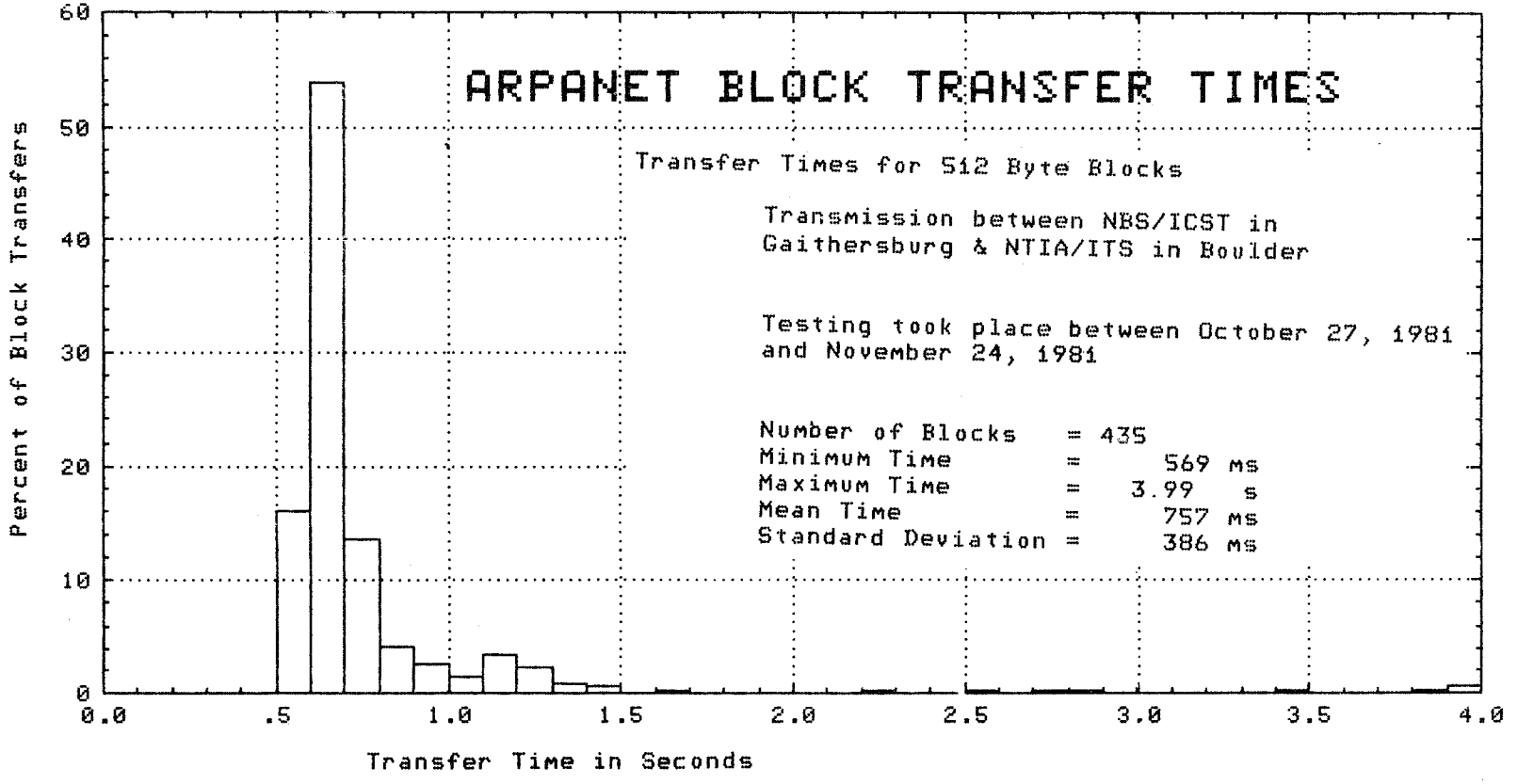


Figure 36. Histogram of block transfer times - overall average for 512-byte blocks.

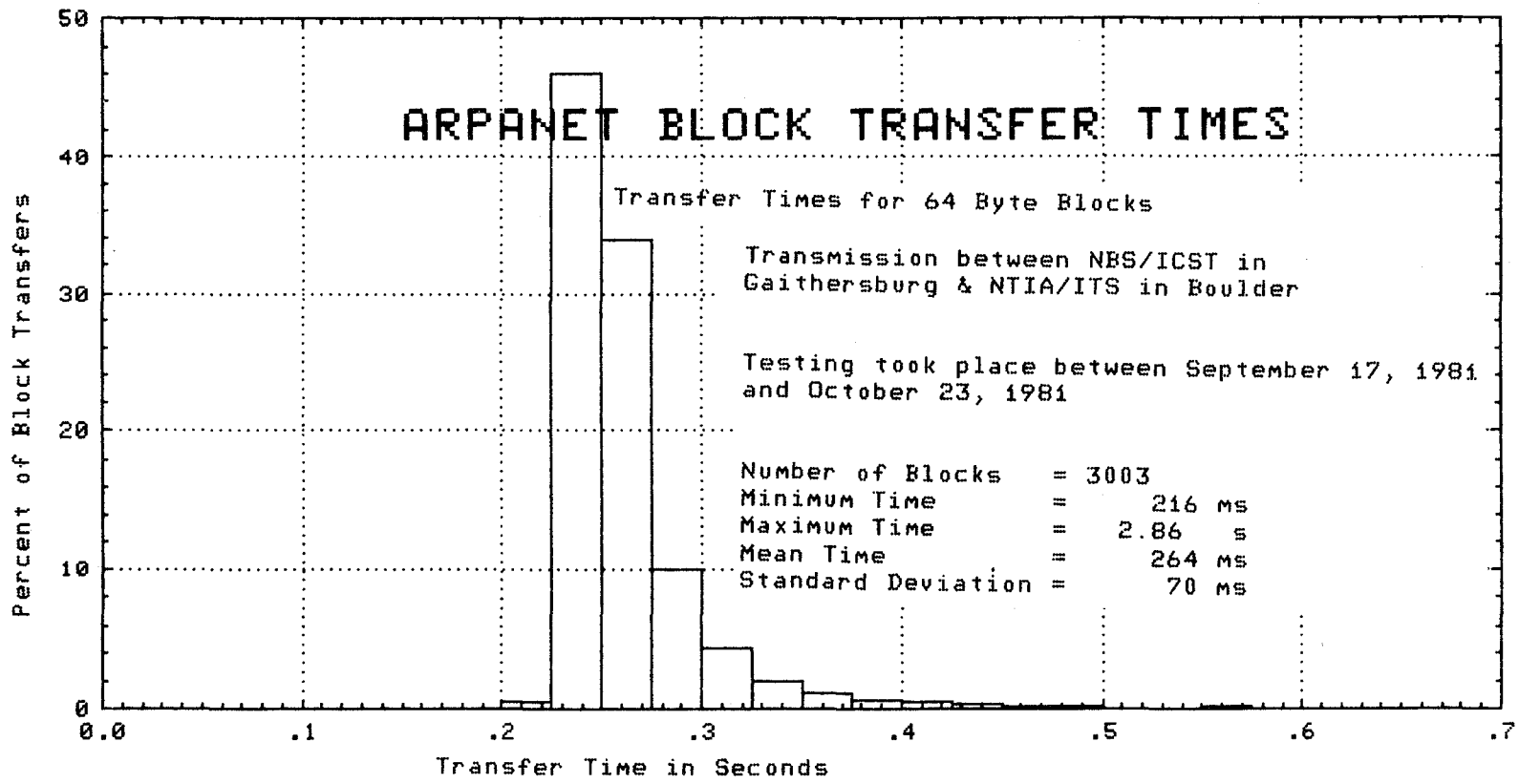
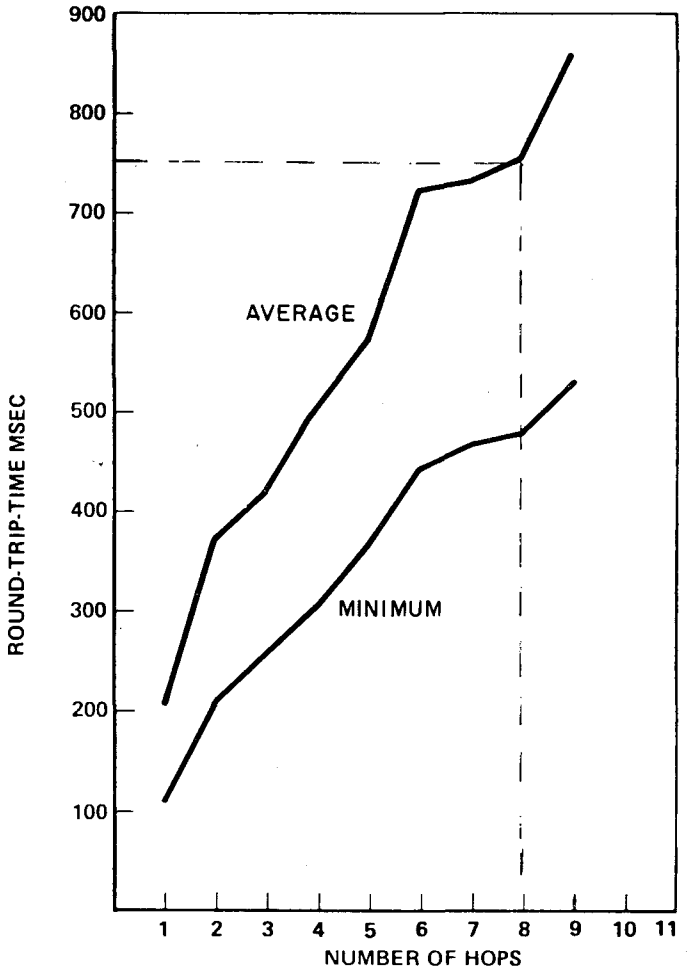
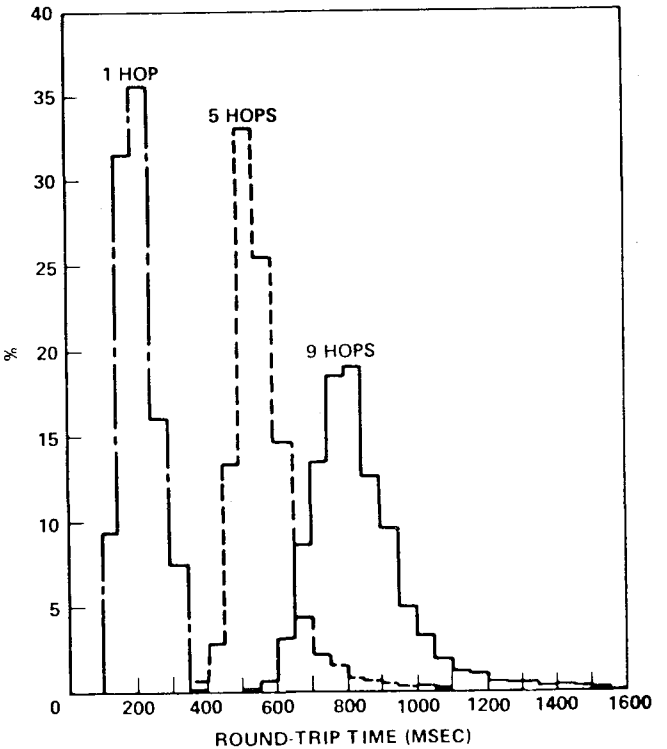


Figure 37. Histogram of block transfer times - overall average for 64-byte blocks.



a. Round-trip delay in the ARPANET (October 1974).



b. Histogram of round-trip delays (msec).

Figure 38. Published data on "round-trip delays" in the ARPANET.

"Round-trip delay is measured by the IMP's and is the time from when a message enters the network until the network's end-to-end acknowledgement in the form of a RFNM is returned."

Thus, "round-trip delay" includes two subnetwork Block Transfer Times: one for the source block and one for the Request for Next Message (RFNM). The source blocks transmitted in these "round-trip delay" measurements were 8063 bits (8 packets) in length (Kleinrock and Opderbeck, 1975). The RFNM's are short, one packet messages. Based on a factor of 3 as the approximate difference in transit delay between the source message and the RFNM, the average one-way subnetwork Block Transfer Time for a 8-packet message across the 8-hop DOCB/NBS path is estimated to be about (.75)(750 ms) = 563 milliseconds. The corresponding single-packet delay would be (.25)(750 ms) = 187 milliseconds. Thus, about 70% of the user-to-user Block Transfer Time is estimated to be subnetwork delay in each case.

2. The ARPANET error control algorithms have been designed to ensure that undetected bit (or block) errors will occur on the order of "years to centuries apart" (Kleinrock, 1976). Has this objective been attained, or is it, in fact, possible to observe undetected bit errors in a measurement spanning only a few months?

Not a single bit error was observed in the 3,287,040 bits transferred during the ARPANET access/disengagement and UIT tests. Thus, both the Bit Error Probability and Block Error Probability values are estimated as zero. The 95% upper confidence limit for the former parameter, (assuming the conditional probability of an error, given an error on the preceding bit, is $p_{11} = 0.7$) is 3×10^{-6}.

3. What is the likelihood that a unit of information delivered by the ARPANET to a given destination user will, in fact, have been intended for some other user?

The question is answered by the measured values for Bit Misdelivery Probability and Block Misdelivery Probability. As discussed more fully in answer to a later question, these values could have been measured with little additional effort during the ARPANET tests by simply creating a duplicate of the destination user program, with a different process address, at each site. Unfortunately, the simplicity of that approach was not realized until most of the ARPANET tests had been completed; and the limited data collected after its implementation were not sufficient to estimate those parameters in a statistically valid way. Based on the error probability results just

described, it is highly unlikely that Block Misdelivery would have been observed in any case.

4. Kleinrock (1976) has reported that "on the average, every hundredth message which enters the ARPANET will not reach its destination. The reason for this undesirable behavior is that many destination hosts are tardy in accepting messages." Is the loss of user information as frequent today, in the configuration tested, as it was in 1976? What proportion of the observed Block Loss outcomes is attributable to performance timeouts as opposed to the actual absence of the received user information? Is Block Loss primarily attributable to "tardy" hosts, or does the subnetwork also contribute?

These questions are answered by the measured values for the Bit Loss Probability and Block Loss Probability parameters. Unlike Block Error, Block Loss proved to be readily observable in the ARPANET experiment. Excluding timeouts, the value measured for that parameter over all blocks transmitted during the experiment was $7/3445 = 2 \times 10^{-3}$. That value is somewhat lower than the value measured by Kleinrock 5 years earlier.

An additional 14 transmitted blocks were classified as lost on the basis that their transfer times exceed the "3 times nominal" timeout value. Thus, the overall Block Loss Probability measured during the test was $(7 + 14)/3445 = 6 \times 10^{-3}$. The upper and lower 90% confidence limits on this value are 9×10^{-3} and 4×10^{-3}, respectively. Exactly two-thirds of the blocks classified as "lost" during the test were timeout failures.

The proportion of Lost Block outcomes caused by "tardy" hosts cannot be determined exactly from the data observed, but a survey of the anomaly causes listed in Table 3 suggests that a substantial proportion of the 7 "total" losses were caused by host NCP failures. The question is somewhat academic to the end users in any case, since the data is unavailable either way.

The calculation of Block Loss Probability required one rather difficult statistical decision. An additional 39 Block Loss outcomes were observed in tests 78, 79, and 94, as described in Table 3. These failures were not included in the above figures because they were a result of "hard" failures in the host software which would have caused Block Loss outcomes in all subsequent transmitted data. Adding the Block Loss outcomes actually observed after these failures to those observed on other tests would be misleading, since the former total is more a function of the test operator's alertness than the system's performance. Each of these observed software failures was counted in calculating the secondary (availability) parameters.

Bit Loss Probability was identical to Block Loss Probability in the case studied, since only entire blocks were lost. In the general case, bits may be lost within a delivered block, making the Bit Loss Probability higher than the Block Loss Probability. Note that FED STD 1033 and X3.102 classify a block as lost only if all bits in the block are lost. Delivered blocks from which some, but not all bits have been lost are classified as Incorrect Blocks.

5. Data loss and data duplication are similar in that both can be caused by ARQ protocol failures and system "crashes". Does data duplication occur with measurable frequency in the ARPANET? Are nonduplicate extra bits delivered to ARPANET users with measurable frequency? What phenomena cause such events, assuming they occur?

No instances of either data duplication or the delivery of nonduplicate "extra" bits were observed during the ARPANET testing. Thus, both the Extra Bit Probability and the Extra Block Probability values are estimated as zero. The 95% upper confidence limit for the former parameter, based on the same sample size and conditional failure probability defined in the case of Bit Error Probability, is 3×10^{-6}.

Although delivery of duplicate or extra data is rare or even nonexistent in the ARPA network, it is always possible whenever two "copies" of an information unit exist simultaneously at different physical locations in a network. For further discussion, see Sunshine (1975) and Seitz (1980a).

6. What throughput is typically achieved between host application programs utilizing the ARPA network? How does this throughput compare with the theoretical maximum process-to-process bandwidth of the network (Kleinrock, 1976)? How does it compare to the values obtained in earlier, experimental measurements? How does it compare with the allocated channel signalling rate? To what extent are the measured throughput values influenced by source and destination user delays?

The two standard performance parameters which most directly measure the throughput performance of a network are the Interim FED STD 1033 parameter Bit Transfer Rate and its ANSI equivalent, User Information Bit Transfer Rate. The values for these two parameters differed insignificantly in the ARPANET application because of its relatively short transit delays, so that the discussion of results can be focussed on Bit Transfer Rate without loss of generality.

Bit Transfer Rate values were measured only in the case of the 512-byte UIT tests, because only a single block was transmitted during each session in the access/disengagement tests. The average user-to-user Bit Transfer Rate

measured during the ARPA network UIT tests was 4872 bits per second (bps). That value is substantially lower than both the previously measured and the theoretical maximum ARPANET throughputs reported earlier. Kleinrock (1976) has calculated a maximum possible bandwidth for ARPANET process-to-process (user-to-user) communication of 39.6 kilobits per second (kbps); and has measured ARPANET throughput vs. hop distance with the results shown in Figure 39. Note that an average throughput of 32.5 kbps was achieved for an 8-hop path in that (1975) measurement. In a later study, Kleinrock et al. (1976) examined the throughput performance of the ARPANET under operational traffic conditions, and concluded that "if the overall traffic characteristics remain unchanged, not more than roughly 10 kbps of the 50 kbps will, on the average, be available for process-to-process communication." The Bit Transfer Rate values measured in this experiment are roughly half that, or about 10% of the 50 kbps ARPANET signalling rate.

The difference between the 10 kbps throughput predicted by Kleinrock and the roughly 5 kbps measured in this experiment is probably a result of three factors:

1. User Delays. The measured value of User Message Transfer Time Fraction for the overall UIT tests was 0.13. Dividing 5.3 kbps by the one's complement of that factor would produce a user-independent throughput value of 6.1 kbps.

2. Host Time-Sharing Delays. Both host NCP's are subject to interruption by other programs, and of course the operating system itself introduces some delay in transferring data between the NCP's and the user programs.

3. Occasional Long Block Transfer Times. Such delays will retard "throughput" under either the Interim FED STD 1033 or the X3.102 parameter definitions, since ARPANET flow control protocols allow only a certain amount of user data "in the pipe" at any time.

The relative influence of the latter two factors on Bit Transfer Rate is unknown.

The Interim FED STD 1033 parameter Block Transfer Rate differs only by a constant from the corresponding bit-oriented parameter, since the block length used in the UIT measurements was fixed. The measured 512-byte Block Transfer Rate was 4872/(512)(8) = 1.19 blocks per second. Thus, about one 512-byte block per second was transferred between the NTIA/ITS and NBS/ICST host computer application programs during a typical UIT test.

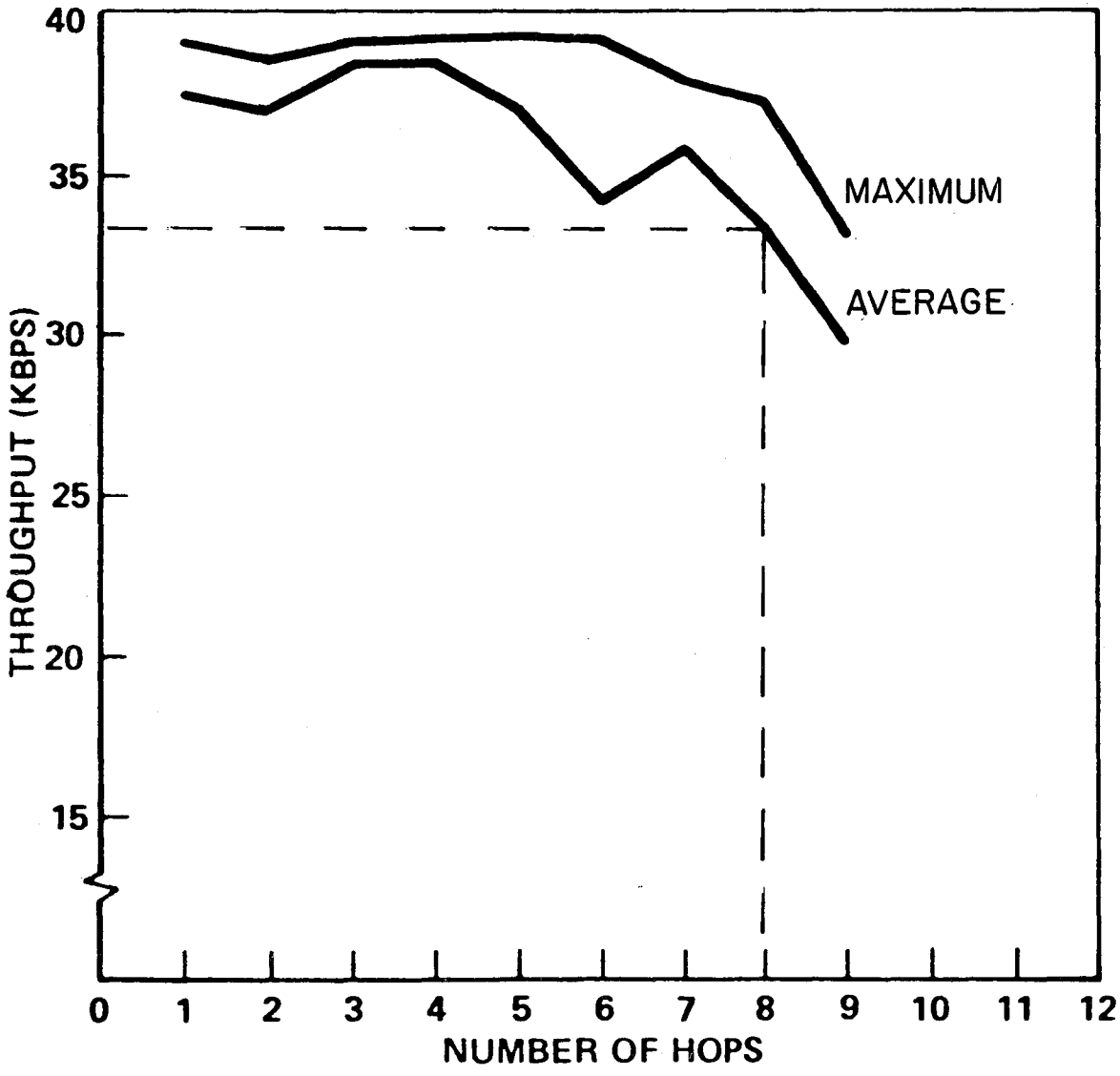


Figure 39. Throughput in the ARPANET (October 1974).

7. What is the overall availability of data communication service between application programs in the case studied? How is the downtime subdivided between the subnetwork and the hosts? How does the subnetwork (e.g., IMP/TIP) downtime observed in the case studied compare with earlier measured values?

As discussed in Section 2, these questions are addressed by the Interim FED STD 1033 parameter Outage Probability and its ANSI counterpart, Transfer Denial Probability; and by the Interim FED STD 1033 parameters Service Time Between Outages and Outage Duration. The ANSI standard parameter Access Outage Probability also bears on these questions, as discussed above.

Outage Probability and Transfer Denial Probability are calculated by dividing the period of time during which user information transfer is in progress (UIT time) into subintervals consisting of a specified number of bits; and then defining the service to be in either the operational service or the outage state during each subinterval on the basis of the outcomes encountered by those bits. The subinterval (called a "message" in Interim FED STD 1033 and a "sample" in X3.102) is chosen so as to obtain values of a given precision for a set of "supported" performance parameters; and the secondary (availability) state of the service is determined over any given sample by comparing the observed values for the supported parameters with established outage thresholds.

In the ARPANET application, a method was developed by which both these parameters could be measured by processing the same samples in slightly different ways. What was done was the following:

1. The "expected" performance of the end-to-end service was characterized in terms of specified values for the supported performance parameters.

2. Outage or Transfer Denial thresholds for these supported parameters were defined on the basis of the specified values, using both the Interim FED STD 1033 and X3.102 algorithms.

3. A "message" or "transfer sample" size was established on the basis of the threshold values and the measurement precision objectives defined in X3.102. (Interim FED STD 1033 does not require any particular measurement precision in the determination of outages.)

4. Secondary or availability outcomes were determined by comparing the "supported" parameter values observed during each sample with the corresponding thresholds. Evaluation of these availability parameters did not require separate measurements, since the same data used in evaluating the primary parameters could be used.

"Expected" or specified values for the supported performance parameters would normally be established on the basis of either user requirements or previous measurements. In the case of User Information Bit Transfer Rate, preliminary measurement results were available and indicated that a specified value of 5 kbps would be appropriate. Neither source of values was available in the case of the probability parameters, and it was therefore necessary to define the values somewhat arbitrarily. It was decided to adopt the user requirements specified in a recent network service solicitation issued by the Environmental Protection Agency (EPA, 1980) as the "expected" ARPANET values for three reasons: (1) those requirements are defined directly in terms of the Interim FED STD 1033 parameters; (2) the service desired was similar, in many respects, to that provided by the ARPANET; and (3) several major suppliers of public data communication services submitted quotations indicating a willingness and ability to comply with the specified values. The specified values (and the associated threshold values) for each of the four supported parameters are summarized below.

Supported Parameter	Specified Value[22]	FED STD 1033 Threshold Value	X3.102 Threshold Value
Bit Transfer Rate	5000 bps	1667 bps	1667 bps
Bit Error Probability	8×10^{-6}	2.8×10^{-3}	5.3×10^{-2}
Bit Loss Probability	8×10^{-6}	2.8×10^{-3}	5.3×10^{-2}
Extra Bit Probability	8×10^{-6}	2.8×10^{-3}	5.3×10^{-2}

A minimum transfer sample size, based on the X3.102 threshold values and Figure 19, was calculated as follows:

$$\frac{18 \text{ failures}}{(5.3 \times 10^{-2}) \text{ failures /bit}} = 340 \text{ bits}$$

The ANSI standard stipulates an additional sample size requirement - that a transfer sample must include at least one interblock gap. The smallest sample size fulfilling that requirement in the case of the UIT tests was 8192 bits, since the block size in those tests was 512 bytes. A "message" or sample size of 8192 bits was used in calculating both Outage Probability and Transfer Denial Probability values.

Values for Outage Probability and Transfer Denial Probability were calculated from the UIT test data as follows. The 435 blocks transmitted

---

[22]Based on specified character-oriented parameter values. Misdelivery probability values were not specified and are ignored here.

during the UIT tests were partitioned into 217 two-block "samples", each composed of 8192 consecutive bits. Performance during each sample was compared with the specified thresholds to identify any Outages or Transfer Denials. Six occurrences were identified, each on the basis of one or more Block Loss outcomes. Each failure qualified as both a Interim FED STD 1033 Outage and an X3.102 Transfer Denial. The values for both Outage Probability and Transfer Denial Probability were thus estimated as $5/217 = 2.8 \times 10^{-2}$.

For comparison, a similar calculation of Outage/Transfer Denial Probability was made on the basis of the 64-byte UIT results observed during the access/disengagement tests. The same (8192 bit) sample size was used; but in this case, the sample comprised 16 consecutive blocks, each transmitted in a separate data communication session. These samples do not include an inter-block gap, and thus they are, strictly speaking, not appropriate for estimating X3.102 User Information Bit Transfer Rate. However, this deficiency had no practical effect because the only transfer failures observed were Lost Blocks.

A total of 208 samples were available for calculating Outage/Transfer Denial Probability values from the access/disengagement test data: 186 samples from the 3010 64-byte blocks transmitted during the 21 access/disengagement tests listed in Table 2, and 22 samples from the 356 blocks transmitted in three excluded batches (78, 79, and 94 - see Table 3). Of the 208 samples, 14 contained one or more Lost Blocks and thus qualified as Outages/Transfer Denials. The values for Outage Probability and Transfer Denial Probability were thus estimated from the access/disengagement test data as $14/208 = 6.7 \times 10^{-2}$. A single, aggregate estimate of Outage/Transfer Denial Probability based on both the UIT and access/disengagement tests would be $(6 + 14)/425 = 4.7 \times 10^{-2}$. As noted earlier, that value can be regarded as a sampled measure of the system's unavailability during user information transfer. The corresponding availability value would be 95.3%

Based on the anomaly causes listed in Table 3, it appears that most of the end-to-end service unavailability observed during the tests was attributable to host failures. However, the Outage/Transfer Denial Probability values measured in this experiment are actually rather close to the 1.64% ARPANET IMP "down rate" reported earlier (Kleinrock, 1976).

It became apparent relatively early in the ARPA network measurements project that it would not be possible to obtain realistic measured values for the other two "secondary" performance parameters specified in Interim FED STD

1033: Service Time Between Outages and Outage Duration. The reason for this difficulty was the limited measurement time available. Unlike Outage Probability, which can be estimated on the basis of very few observed outages, the two time parameters require the averaging of many successive operational service and outage periods. Successful measurement of the latter parameters would require either much more continuous sampling of the service (e.g., tests in progress throughout the normal business day) or a much longer overall test period (e.g., 1 year). Neither option was feasible in this experiment, and thus no meaningful measurement results for these parameters can be reported.

The ARPA network user information transfer measurements were also intended to answer seven questions regarding the detailed definition of the UIT performance parameters. These questions are restated and answered below.

1. Are Bit Transfer Time and Block Transfer Time distinguishable in the case studied? Is a comparison of these parameter values useful in expressing the influence of block length on transfer time?

Bit Transfer Time and Block Transfer Time were virtually identical in the case studied because user information blocks were passed across both user/system interfaces as a unit (with the Write request or Read Complete response). The effect of block length on transfer time could not be determined by comparison of these values, but was measured directly in terms of the Block Transfer Times for two representative block lengths (as shown earlier in Figures 38 and 39).

2. Are Bit Error Probability and Block Error Probability independently useful in characterizing transfer accuracy in the case studied? Does a comparison of these values provide useful information on error clustering?

Both questions are moot, since no bit errors were observed during the experiment. The use of these two parameters in characterizing error clustering is described in Appendix B of X3.102 (ANSI, 1982).

3. Are Bit and Block Misdelivery Probability, as defined in Interim FED STD 1033 and X3.102, measurable in the case studied? Could their measurement be made simpler if the parameters were defined differently?

The answers to these questions require a bit of explanation. The Bit and Block Misdelivery Probabilities could have been measured rather easily in the ARPA network testing through the creation of "dummy" XMIT or RECV programs; but this approach was not conceived and implemented until most of the testing

had been completed. As a result, no useful values for these two parameters were obtained.

One relatively simple approach which could have been used to measure the misdelivery probabilities is illustrated in Figure 40. The possibility for misdelivery is created by "forking" (duplicating) the RECV program at each destination user site, and then changing the address of the forked program to one distinct from, but similar in binary representation to, that of the intended destination program. Both programs would be activated at the outset of the test, with the expectation that only the intended destination program would actually receive any traffic. Receipt of traffic by the unintended destination would, of course, indicate misdelivery.

While the above approach is conceptually and procedurally simple, it has about as much intuitive appeal to a practical measurement engineer as looking for a needle in a haystack. That is probably why it was overlooked in the ARPANET test design. Both Interim FED STD 1033 and X3.102 point out that separate consideration of the misdelivery probabilities is optional. The extremely high transfer accuracy measured during the ARPANET experiment argues that misdelivery is very unlikely in that network; and the measurement outlined in Figure 40 would almost certainly have confirmed that expectation.

4. How strongly are the values for Bit and Block Loss Probability influenced by the choice of maximum block transfer time? Would a different algorithm for determining maximum block transfer time be more appropriate in the case studied?

As discussed above, two-thirds of the Block Loss outcomes observed during the ARPANET tests were timeout failures. Eliminating all timeout failures from the Block Loss category would have reduced the reported Block Loss Probability by a factor of 3, from 6.1×10^{-3} to 2×10^{-3}. As noted in a similar discussion of access timeout, the "3 times nominal" rule for defining performance failures is somewhat arbitrary but appears reasonable in the case studied.

5. ANSI X3.102 differs from Interim FED STD 1033 in the definition of User Information Bit Transfer Rate. Are these differences significant in the case studied? Which definition is preferable from a measurement standpoint?

These differences, described in Section 2.3 and illustrated in Figure 9, proved to be unimportant in the ARPANET application. The throughput values measured under the two definitions typically differed by only a few percent. The differences in parameter definition would be much more significant in

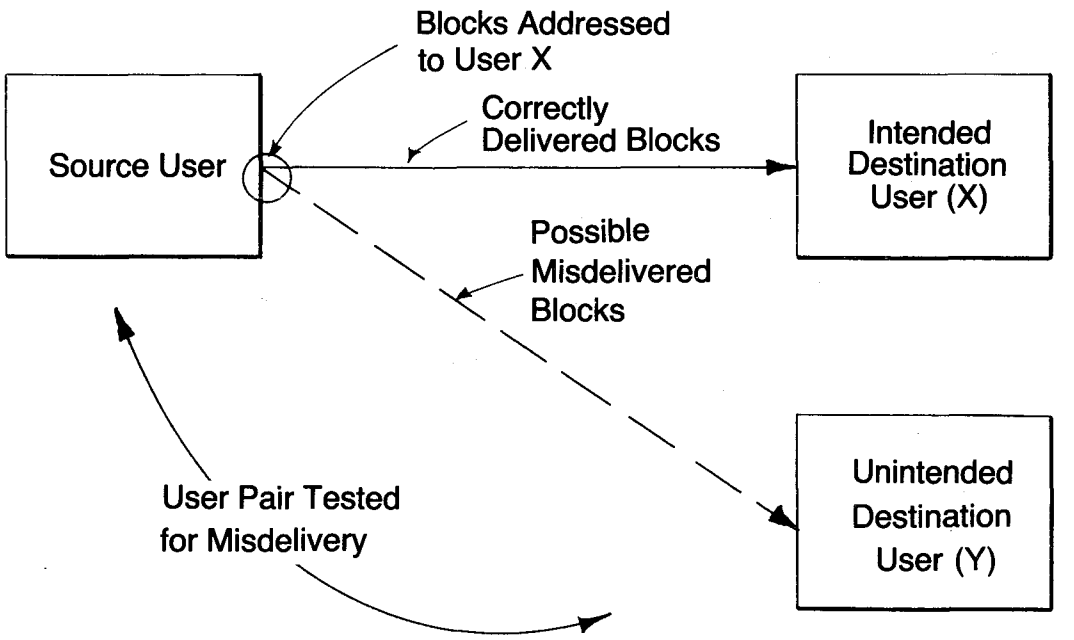


Figure 40. Misdelivery measurement approach.

applications with longer Block Transfer Times (e.g., electronic mail or traditional message switching systems).

6. ANSI X3.102 differs from Interim FED STD 1033 in that the Block Transfer Rate and Bit/Block Rate Efficiency parameter are omitted. Does the omission of these parameters detract significantly from the completeness of the performance description?

The custom of relating user-to-user transfer rate to the associated channel signalling rate is a very useful one, as evidenced by its widespread acceptance by communication system analysts and designers. However, omission of the Rate Efficiency parameters from X3.102 can be justified on the basis that their values are primarily of interest to service suppliers rather than end users. The decision has essentially no measurement implications in any case, since the Rate Efficiency parameter values differ only by a constant from the corresponding Bit Transfer Rates.

7. ANSI X3.102 differs from Interim FED STD 1033 in the selection and definition of availability parameters. Which approach appears preferable from a measurement standpoint? Does the omission of Service Time Between Outages and Outage Duration detract significantly from the completeness of the performance description? How strongly are the Outage (or Service Denial) Probability values influenced by the choice of degraded performance thresholds? How much do the required sample sizes differ under the two threshold criteria? Is the difference significant from a measurement standpoint?

As discussed earlier, it was not practical to obtain values for the Interim FED STD 1033 "secondary" parameters Service Time Between Outages and Outage Duration during the ARPA network measurements because of the very extensive observation time required. These MTBF/MTTR parameters are conceptually important to end users, and their omission definitely detracts from the completeness of a performance specification. Unfortunately, it appears certain that the same difficulty encountered in measuring these parameters in the ARPANET experiment would also be encountered in many operational measurements. Even if these parameters were successfully measured over a particular observation period, they would be less useful than other parameters in predicting future performance because their longer measurement time would increase the likelihood of nonstationarity in the measured service. With some reluctance we must conclude, therefore, that the omission of Service Time Between Outages and Outage Duration from the set of standard performance parameters appears to be a practical necessity.

The differences between the X3.102 and Interim FED STD 1033 outage threshold criteria had little effect on the assessment of transfer availability in the application tested, since the service performance within a sample varied between extremely good and extremely poor performance (Block Loss) in an essentially binary fashion. The choice of threshold criterion did significantly influence the transfer sample size, however: if the Interim FED STD 1033 "square root rule" rather than the ANSI X3.102 "fourth root rule" had been used in defining the probability parameter thresholds, and the same level of precision had been sought, the minimum transfer sample size would have been increased from $18/(5.3 \times 10^{-2}) = 340$ bits to $18/(2.8 \times 10^{-3}) = 6429$ bits. This difference was masked by the "inter-block gap" requirement in this experiment, but it would be significant in cases where a smaller block size was used. Based on the performance observed during this experiment, the ANSI approach appears preferable.

6.2.3 Disengagement Parameters

The two questions posed in Section 3.2.3 regarding ARPA network disengagement performance are restated and answered below.

1. How long must a user wait, after requesting disengagement from an established data communication session, for the disengagement function to be successfully completed? Does that delay differ substantially for each user? How variable is the delay? What proportion of the delay is attributable to subnetwork packet transmissions? What proportion is attributable to delays introduced by the users themselves?

These questions are addressed by the Interim FED STD 1033 and X3.102 Disengagement Time parameters. The ARPANET measurements quickly demonstrated a distinct asymmetry in the values for these parameters between the user originating disengagement (the source user in this experiment) and the other (nonoriginating) user. This asymmetry is illustrated in the bimodal Disengagement Time distribution shown in Figure 41. The figure summarizes the results of 3054 disengagement attempts, all observed during peak hours under high priority, with the source user (and disengagement originator) at NBS. The average originating user Disengagement Time was 12 milliseconds (within $\pm$ 1 ms); the average nonoriginating user Disengagement Time was 2.5 seconds (within $\pm$ 0.1 second).

The reason for the substantial difference between the originating and nonoriginating user disengagement times measured in the ARPANET experiment is

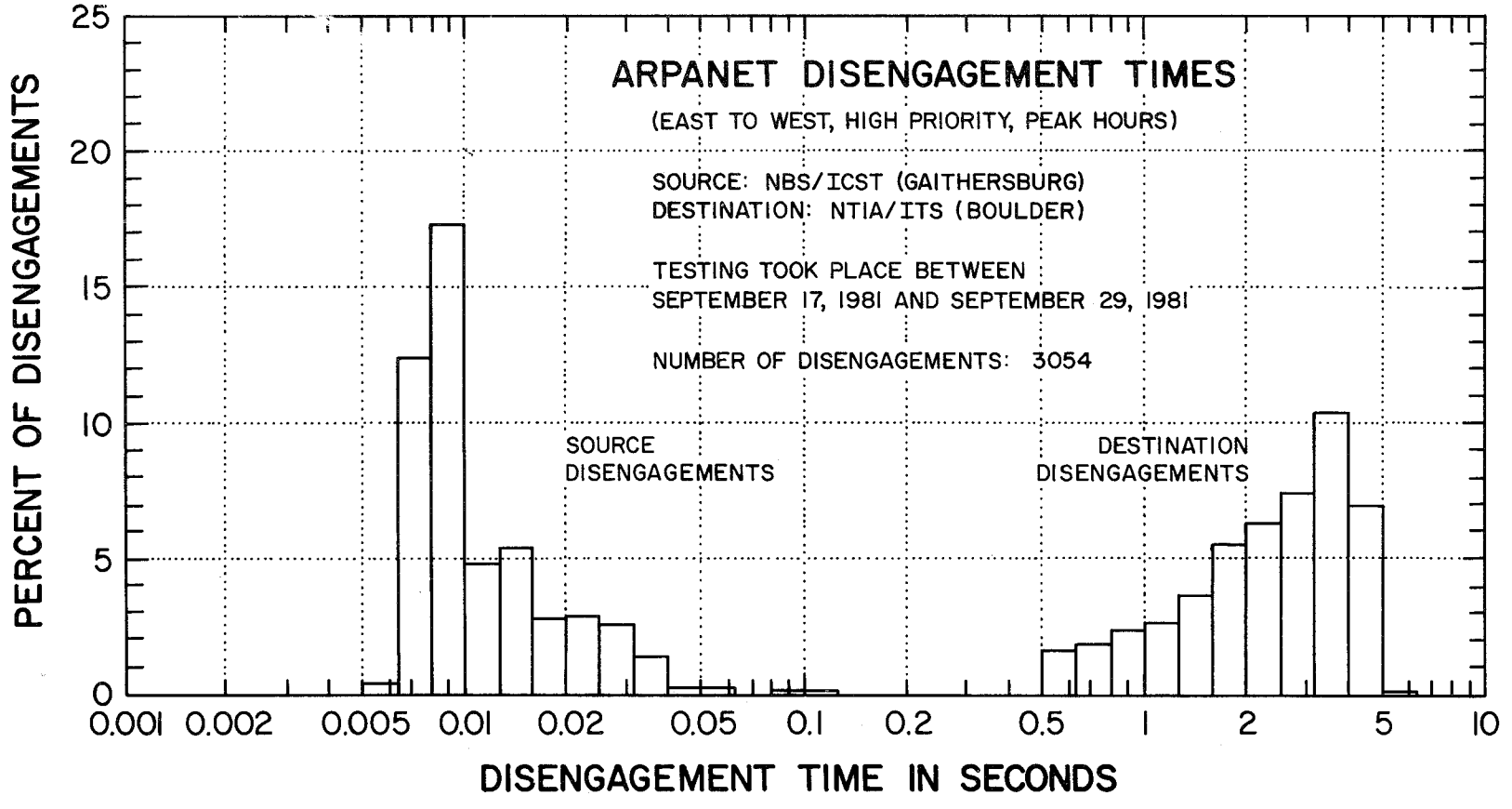


Figure 41. Disengagement time histogram.

evident from the data communication session profile (Figure 25). Disengagement of the originating user is a simple local function which is accomplished entirely within that user's host computer operating system. All that program has to do to disengage an originating user is to remove the Open Request parameters from its local tables and issue a CLOSE COMPLETE response. Once the disengagement request has been issued, the originating user is not involved in the process at all. In contrast, disengagement of the nonoriginating user requires first, the transmission of a disconnect request from the originating NCP through the network to the nonoriginating user; second, that user's issuance of a CLOSE request; and finally, issuance of the CLOSE COMPLETE response by the nonoriginating host operating system.

Based on the Block Transfer Time measurements reported earlier, it appears that only about 10% of the nonoriginating user Disengagement Time is occupied by the subnetwork's disconnect request transmission. About 10% of the remaining time is nonoriginating user time, associated with issuance of the second CLOSE request. The remaining time is apparently occupied by NCP and OS processing.

As in the case of Access Time, a statistical analysis of the collected Disengagement Times was conducted about midway through the test to check dependence assumptions and refine the sample size requirements. Major steps in this analysis are briefly summarized below.

The destination Disengagement Times were analyzed in the same way as the Access Times, except that the full autocorrelation function was estimated for each of the 10 samples available using the last 40 times of each sample. The autocorrelation function of the destination Disengagement Times was determined because those times were known to have substantial within-sample variance. The autocorrelations of lags 1 through 10, averaged over the 10 samples of 40 each, are:

0.176 -0.018 0.026 0.085 -0.020 0.031 0.035 -0.052 -0.048

All of these except the first are quite negligible and in fact are not statistically significant, judged by comparison with the approximate standard error $n^{-1/2} = 400^{-1/2} = 0.05$. This draws some confirmation from the fact that 5 of the 10 are positive and 5 negative. They are consistent with the Markov model $\rho_k = \rho_1^k$.

The mean destination Disengagement Time, estimated from the last 40 trials in each of the 10 samples, is 2.496 seconds. The root-mean-square

within-sample standard deviation is 1.139 second. To get confidence limits for the true mean we proceed as with the Access Times, and get the variance of the 10 means, 0.028808. The independent estimate of the individual observation variance is 40 times this, 1.15232, the square root of which, 1.073 seconds, is actually smaller than the above within-sample standard deviation. Hence there is no additional between-sample variation, in contrast to the situation for Access Times.

Thus, following the Access Time analysis, we take $s_a^2 = 0$, $s_{\bar{x}}^2 = (1.139)^2/400 = 0.003243$. The autocorrelation factor, 1.176/0.824 = 1.427, increases this to 0.004629, so the approximate 95% confidence limits for mean destination disengagement time are

$$2.496 \pm 0.136 = \begin{matrix} 2.632 \\ 2.360 \end{matrix} \text{ seconds} \quad .$$

Since there was no additional between-sample variance, this precision could have been doubled by simply analyzing all 1600 times in the 10 samples rather than only 400. All available data were used in calculating the overall disengagement time values presented above.

2. What is the likelihood that the system will fail to detach a user from an established data communication session (within a specified maximum disengagement time) on any given request? How sensitive is that likelihood to the specified maximum time?

These questions are answered by the measured Disengagement Denial Probability values. As in the case of the Disengagement Times, the Disengagement Denial Probabilities were substantially different for the originating and nonoriginating users and were calculated separately. The originating user disengagement process exceeded the "3 times nominal" timeout threshold 38 times during the 2993 ARPANET access/disengagement trials, producing an originating user Disengagement Denial Probability value of 1.3×10^{-3}. The nonoriginating user disengagement process never exceeded its (separate) timeout during the tests, producing a nonoriginating user Disengagement Denial Probability estimate of zero. The system ultimately completed the disengagement function without corrective action on every trial, including the 38 trials that were "timed out" for performance assessment purposes. Thus, in this case, the timeout thresholds completely determined the disengagement Denial Probability values; and those values simply measured the spread of their associated Disengagement Time distribution.

The ARPA network disengagement measurements were also intended to answer two questions regarding the detailed definition of the disengagement performance parameters. These questions are restated and answered below.

1. The ANSI X3.102 definitions for Disengagement Time and Disengagement Denial Probability differ from their Interim FED STD 1033 counterparts in one important respect: they allow the specification of separate parameter values for each end user in cases where the disengagement functions at the user interfaces are dissimilar. Is this distinction important in the case studied? How meaningful are the "aggregate" parameter values representing both user interfaces?

The distinction between originating and nonoriginating user disengagement proved to be very significant in the ARPANET case. In fact, the extreme bimodality of the Disengagement Time distribution (Figure 41) indicates that the "average" performance of the two ends would be completely misleading in characterizing either. This finding substantiates the ANSI X3S35 decision that the standard should encourage separate specification of disengagement performance in cases where the disengagement process differs significantly at the two user interfaces.

2. How strongly are the values for Disengagement Denial Probability influenced by the choice of maximum disengagement time? Would a different algorithm for determining maximum disengagement time be more appropriate in the case studied?

As discussed earlier, the system ultimately completed the disengagement function without corrective action on every trial, including those that were "timed out". Unlike access and block transfer, only timeout failures were observed in the case of disengagement; and thus, the measured Disengagement Denial Probabilities indicate the "spread" of the associated distributions rather than the likelihood of "hard" failures.

These results lead one to question the timeout thresholds used in defining Disengagement Denial in this experiment. The "3 times nominal" rule is particularly unappealing in the originating user case. For the results illustrated in Figure 41, for example, a strict application of this rule forces one to call any disengagement trial lasting longer than 36 milliseconds a Disengagement Denial! It appears, intuitively at least, that the allowable variation in performance time should be greater in cases where the normal performance time is very small.

The threshold defining rules specified in X3.102 allow for such exceptions. That standard acknowledges, in its introduction, that users may

wish to define communication failure probabilities on the basis of different timeout or threshold criteria in certain situations; and requires only that such departures from the normal failure-defining criteria be explicitly identified in performance specifications. The latter requirement is necessary to ensure the validity of performance comparisons.

The X3.102 approach appears to provide a reasonable compromise between the conflicting needs for uniformity and flexibility in the definition of performance failure thresholds.

6.2.4 Summary of Measured Values

Figures 42 and 43 summarize the user-to-user performance of the ARPA network in terms of measured values for the Interim FED STD 1033 and ANSI X3.102 performance parameters, respectively. Confidence limits associated with the various parameter values have been indicated in the preceding sections.

7. CONCLUSIONS AND RECOMMENDATIONS

As stated in Section 3, the ARPA network measurements project had two overall objectives. The primary objective was to verify and demonstrate proposed Federal Standard 1043 by implementing it in a representative measurement situation. A secondary objective was to obtain some typical values to characterize the data communication service provided by the ARPA network and its host computers to end users. It was anticipated that these values would be useful first, in understanding differences between subnetwork and user-perceived performance; and second, in assessing proposed refinements to the parameter definitions themselves. This section briefly summarizes the major conclusions and recommendations of the ARPA network measurements project in the context of these objectives.

With respect to the trial implementation of proposed FED STD 1043, the most significant conclusion is that it was, in fact, successful: that is, it was possible to directly apply proposed Federal Standard 1043 in obtaining Interim FED STD 1033 and ANSI X3.102 parameter values to characterize the performance of a typical modern data communication service. Future implementations of the proposed 1043 standard should be facilitated by the tools and techniques described here. It is recommended, on the basis of this successful implementation, that the FTSC and ANSI Task Group X3S35 proceed apace with the remaining steps required to prepare proposed FED STD 1043 for

PERFORMANCE MEASUREMENT SUMMARY

Part A - Primary Parameters

1.	Access Time		1.8	Seconds
2.	Incorrect Access Probability		0	*
3.	Access Denial Probability		8.6×10^{-3}	*
		(512-bit blocks)	709	Milliseconds
4.	Bit Transfer Time	(4096-bit blocks)	262	Milliseconds
5.	Bit Error Probability		0	*
6.	Bit Misdelivery Probability		†	*
7.	Bit Loss Probability		6×10^{-3}	*
8.	Extra Bit Probability		0	*
		(512-bit blocks)	709	Milliseconds
9.	Block Transfer Time	(4096-bit blocks)	262	Milliseconds
10.	Block Error Probability		0	*
11.	Block Misdelivery Probability		†	*
12.	Block Loss Probability		6×10^{-3}	*
13.	Extra Block Probability		0	*
14.	Bit Transfer Rate	(4096-bit blocks)	4872	Bits/Second
15.	Block Transfer Rate	(4096-bit blocks)	1.19	Blocks/Second
16.	Bit Rate Efficiency		9.7	%
17.	Block Rate Efficiency		9.7	%
		Originator	12	Milliseconds
18.	Disengagement Time	Nonoriginator	2.5	Seconds
19.	Disengagement Denial Probability	Originator	1.3×10^{-3}	*
		Nonoriginator	0	

Part B - Secondary Parameters

20.	Service Time Between Outages	†	Hours
21.	Outage Duration	†	Hours
22.	Outage Probability	4.7×10^{-2}	*

Part C - Ancillary Parameters

23.	User Access Time Fraction		0.15	*
24.	User Block Transfer Time Fraction	(East-to-West)	0.13	*
25.	User Message Transfer Time Fraction		0.13	*
26.	User Disengagement Time Fraction	Originator	0	*
		Nonoriginator	0.13	

*Note: The probabilities and user performance time fractions are dimensionless numbers between zero and one.

†Value not measured.

Figure 42. FED STD 1033 performance measurement summary.

SERVICE PERFORMANCE SPECIFICATION

Part A - Primary Parameters

1. Access Time		1.8	Seconds
2. Incorrect Access Probability		0	*
3. Access Denial Probability		6.0×10^{-3}	*
4. Access Outage Probability		2.6×10^{-3}	*
5. Bit Error Probability		0	*
6. Bit Misdelivery Probability		†	*
7. Extra Bit Probability		0	*
8. Bit Loss Probability		2.0×10^{-3}	*
9. Block Transfer Time	(512-bit blocks)	709	Milliseconds
	(4096-bit blocks)	262	Milliseconds
10. Block Error Probability		0	*
11. Block Misdelivery Probability		†	*
12. Block Loss Probability		2.0×10^{-3}	*
13. Extra Block Probability		0	*
14. User Information Bit Transfer Rate		4872	Bits/ Second
15. Disengagement Time	Originator	12	Milliseconds
	Nonoriginator	2.5	Seconds
16. Disengagement Denial Probability	Originator	1.3×10^{-3}	*
	Nonoriginator	0	
17. Transfer Denial Probability		4.7×10^{-2}	*

Part B - Ancillary Parameters

18. User Fraction of Access Time		0.15	*
19. User Fraction of Block Transfer Time	(East-to-West)	0.13	*
20. User Fraction of Sample Input/Output Time		0.13	*
21. User Fraction of Disengagement Time	Originator	0	*
	Nonoriginator	0.13	

*Note: The probabilities and user performance time fractions are dimensionless numbers between zero and one.

†Value not measured.

Figure 43. X3.102 performance measurement summary.

formal coordination and approval as a joint Federal/American National Standard.

Although the trial implementation of proposed FED STD 1043 revealed no need for major changes to that standard, it did identify a number of opportunities to improve and refine it. Specific recommendations are the following:

1. Provide a summary of the ARPANET Data Extraction subsystem developed during this experiment with the standard, either as an appendix or in an associated user reference manual. Such a summary will greatly facilitate future Data Extraction subsystem designs.

2. Determine and state the constraints, if any, which exist on the transfer of ASCII character records between machine-dependent Data Extraction programs and the standard FORTRAN Performance Assessment program. A clear statement of any such constraints will prevent possible difficulties in transferring recorded performance information between these two performance measurement system elements.

3. Revise the draft 1043 standard to reflect a partitioning of the Performance Assessment program into the three sequentially executed program modules described in Section 4 of this report: PROLOG, ASSESS, and EPILOG. This partitioning ensures a Performance Assessment program size commensurate with the memory capacities of smaller computer systems.

4. Provide additional guidance in the standard to assist users in developing realistic measurement precision objectives. Such guidance will simplify test design process and will lead, in many cases, to less stringent measurement objectives and more economical tests.

While not essential, additional trial applications of proposed FED STD 1043 would be useful to Task Group X3S35 in developing the corresponding ANSI standard. Measurements involving instrumentation of an operator/terminal interface would be particularly helpful, since the ARPA network measurements did not directly address that interface. Comparative measurements of several public data networks would also be valuable, both in further refining the 1043 standard and in promoting more accurate user assessment of competing service alternatives.

A comparison of the measurement results obtained in this experiment with earlier results revealed some significant differences between the performance delivered to typical ARPA network end users and the performance of the subnetwork. Performance delays observed at the end user interfaces were two

to four times greater than the corresponding subnetwork delays, and the observed throughputs were proportionally lower. It appears that most of the additional delay is introduced by communication support software in the hosts (e.g., the NCP's and operating systems) rather than by the user programs themselves.

The measurement results obtained in this experiment confirmed that transmission errors are extremely rare in the ARPANET. However, the loss of user data in transit between application programs is relatively common. Such failures appear to be caused by hardware and software "crashes" and subnetwork delays in addition to the previously reported "tardiness" of ARPANET hosts in accepting transferred messages. Data loss was by far the most serious network imperfection observed during these measurements. That one phenomenon completely determined the measured service availability values.

In almost every case where differences between the Interim FED STD 1033 and X3.102 performance parameter definitions were examined, the measurement results demonstrated that the latter definitions are preferable. This was particularly true in the case of Disengagement Time, where separate specifications for the two user interfaces were clearly required; and in the case of the "secondary" MTBF and MTTR parameters, whose evaluation would have required a measurement period five to ten times longer than the time available. On the basis of this trial implementation, it is recommended that X3.102 be adopted as both the final form of the standard parameter definitions and the basis for revised FED STD 1043 performance measurement methods.

8. ACKNOWLEDGEMENTS

Many individuals contributed to the ARPA network measurements project and the development of this report. Among the former are Peter McManamon of ITS, who conceived and initiated the project; Greg Noel of the Naval Ocean Systems Center, who assisted in shoehorning UNIX into the NTIA/ITS and NBS/ICST host computers; Evi Nemeth and Dwight Melcher of the University of Colorado, who provided supplemental data reduction programs; Rob Rosenthal of NBS, who gave the authors a much-needed early tutorial on UNIX and "C"; and Marty Miles of ITS, who participated in statistical analysis of the measurement data. Among the latter are Bob Linfield, Don Glen, and Evi Gray, who reviewed the draft report in ITS; Ray Moore and Jerry Lynn, who reviewed the draft report in NBS/ICST; and Cathy Edgar, who typed and edited the entire report.

9. REFERENCES

ANSI (1966), USA Standard FORTRAN, X3.9-1966, Computer and Business Equipment Manufacturers' Association, Washington, DC 20036.

ANSI (1967), USA Standard Code for Information Exchange, X3.4-1967, Computer and Business Equipment Manufactures' Association, Washington, DC 20036.

ANSI (1982), Proposed American National Standard No. X3.102, Data communication user oriented performance parameters, April. Available from the authors of this report.

BBN (1977), Terminal interface message processor user's guide, Bolt, Beranek and Newman Report No. 2183, NIC No. 10916, July.

Carr, C. S., S. D. Crocker, and V. G. Cerf (1970), Host-host communication protocol in the ARPA network, AFIPS Conference Proc., 1970 Spring Joint Computer Conference, Vol. 36, pp. 589-597.

Chesson, G. L. (1975), The network UNIX system, Operating Systems Review 7, No. 5.

Cole, G. C. (1971), Computer network measurements: techniques and experiments, School of Engineering and Applied Science, University of California, Los Angeles, Engineering Report UCLA-ENG-7165, October.

Crow, E. L. (1974), Confidence limits for digital error rates, Office of Telecommunications Report 74-51, November.

Crow, E. L. (1979), Statistical methods for estimating time and rate parameters of digital communication systems, NTIA Report 79-21, June.

Crow, E. L., and M. J. Miles (1977), Confidence limits for digital error rates from dependent transmissions, Office of Telecommunications Report 77-118, March.

DCA (1980), ARPANET Directory, NIC 48000, published for the Defense Communication Agency by the Network Information Center, SRI International, Menlo Park, CA 94025.

U.S. Department of Agriculture (1980), Departmental telecommunications network request for proposals, U.S. Department of Agriculture, RFP No. OF-81-R-12, November.

Dorros, I. T. (1981), ISDN, A challenge and opportunity for the 80's, IEEE Commun. Mag., March.

Drukarch, C. Z., P. M. Karp, K. G. Knightson, L. Lavandera, A. M. Rybczynski, and N. Sone (1980), X.25:the universal packet network interface, Proceedings of the Fifth International Conference on Computer Communications, October.

EPA (1980), Request for quotations, telecommunications network service, U.S. Environmental Protection Agency, RFQ No. WA-80-D289/ldm, May.

GAO (1977), Better management of defense communications would reduce costs, Report to the Congress by the Comptroller General of the United States, General Accounting Office LCD-77-106, December.

GSA (1979), Interim Federal Standard 1033, Telecommunications: digital communication performance parameters, General Services Administration, August. Available from Office of the Manager, National Communications System, Technology and Standards Division, Washington, DC 20305.

GSA (1980), Federal procurement regulations, Title 41 - Public contracts, property management, General Services Administration.

ISO (1982), Reference model of open systems interconnection, International Organization for Standardization Draft International Standard 7498, June.

Kamas, G., and S. L. Howe (1979), Time and frequency user's manual, National Bureau of Standards Special Publication No. 559, November.

Kernighan, B. W., and D. M. Ritchie (1978), The C programming language, (Prentice-Hall, Inc., Englewood Cliffs, NJ 07632).

Kleinrock, L., and W. E. Naylor (1974), On measured behavior of the ARPA network, Proceedings of the National Computer Conference, May 6-10.

Kleinrock, L., and H. Opderbeck (1975), Throughput in the ARPANET - protocols and measurement, Proceedings of the Fourth Data Communications Symposium, Quebec, Canada, October.

Kleinrock, L., W. F. Naylor, and H. Opderbeck (1976), A study of line overhead in the ARPA network, Commun. ACM 10, No. 1, January.

Kleinrock, L. (1976), Queueing Systems, Vol. II, Computer Applications (John Wiley & Sons, Inc., New York, N.Y.).

Ku, H. H. (1969), Precision measurement and calibration, selected NBS papers on statistical concepts and procedures, National Bureau of Standards Special Publication 300, Volume 1, February.

McFadyen, H. J. (1976), Systems network architecture: an overview, IBM Systems J. 15, No. 1.

National Research Council (NCR) Committee on Telecommunications (1977), Summary of Office of Telecommunications Study Panel meeting, Boulder, Colo., April 11-13.

NBS (1978), NBS time via satellites, National Bureau of Standards Publication No. TES-602, January.

NBS (1981), Solicitation of comments on impact and applicability of planned user-oriented data communication performance parameters standard, Federal Register 46, No. 58, March.

Roberts, L. G. (1967), Multiple computer networks and inter-computer communications, Proc. of the ACM Symposium on Operating Systems, Gatlinburg, Tenn.

Roberts, L. G., and B. D. Wessler (1970), Computer network development to achieve resource sharing, AFIPS Conference Proc., 1970 Spring Joint Computer Conference, Vol. 36, pp. 543-549.

Seitz, N. B., and P. M. McManamon (1978), Digital communication performance parameters for proposed Federal Standard 1033, NTIA Report 78-4, Vol. I, standard parameters, May.

Seitz, N. B., and D. Bodson (1980), Data communication performance assessment, Telecommunications 14, No. 2, February.

Seitz, N. B. (1980a), Interim Federal Standard 1033 reference manual, NTIA Report 80-55, December.

Seitz, N. B. (1980b), Measuring communication availability with Federal Standard 1033, Proc. 1980 Reliability and Maintainability Symposium, San Francisco, Calif., January.

Seitz, N. B., K. P. Spies, and E. L. Crow (1981a), Telecommunications: digital communication performance measurement methods, proposed Federal Standard 1043, Version 5, May. Available from the authors of this report.

Seitz, N. B., K. P. Spies, and E. L. Crow (1981b), Data communication performance measurement--a proposed Federal Standard, Proc. 1981 National Telecommunications Conference, New Orleans, Louisiana, November.

Sunshine, C. A. (1975), Interprocess communication protocols for computer networks, Digital Systems Laboratory, Department of Electrical Engineering, Stanford University, Technical Report No. 105, December.

Thompson, K., and D. M. Ritchie (1975), UNIX programmer's manual, Bell Telephone Laboratories, Inc., May.

APPENDIX A: NARRATIVE PARAMETER DEFINITIONS

This appendix provides brief narrative definitions for the Interim FED STD 1033 and ANSI X3.102 parameters. The parenthetical symbols in these definitions are explained in the referenced sample space diagrams and tables.

A.1 Interim Federal Standard 1033 Parameters

The 26 performance parameters specified in Interim Federal Standard 1033 are defined as follows. Refer to the standard and its supporting reports for more explicit definitions of the associated outcomes.

A.1.1 Access Parameters

Interim Federal Standard 1033 specifies access performance in terms of the three primary parameters identified in Figure A-1. Narrative definitions for the selected access parameters are provided below.

Access Time $W(a_s)$ - Average value of elapsed time between the start of an access attempt $t(a)$ and Successful Access $t(a_s)$. Elapsed time values are calculated only on access attempts that result in Successful Access.

Incorrect Access Probability $P(a_m)$ - Ratio of total access attempts that result in Incorrect Access (A_m) to total access attempts included in the reduced sample (A').

Access Denial Probability $P(a_\ell)$ - Ratio of total access attempts that result in Access Denial (A_ℓ) to total access attempts included in the reduced sample (A').

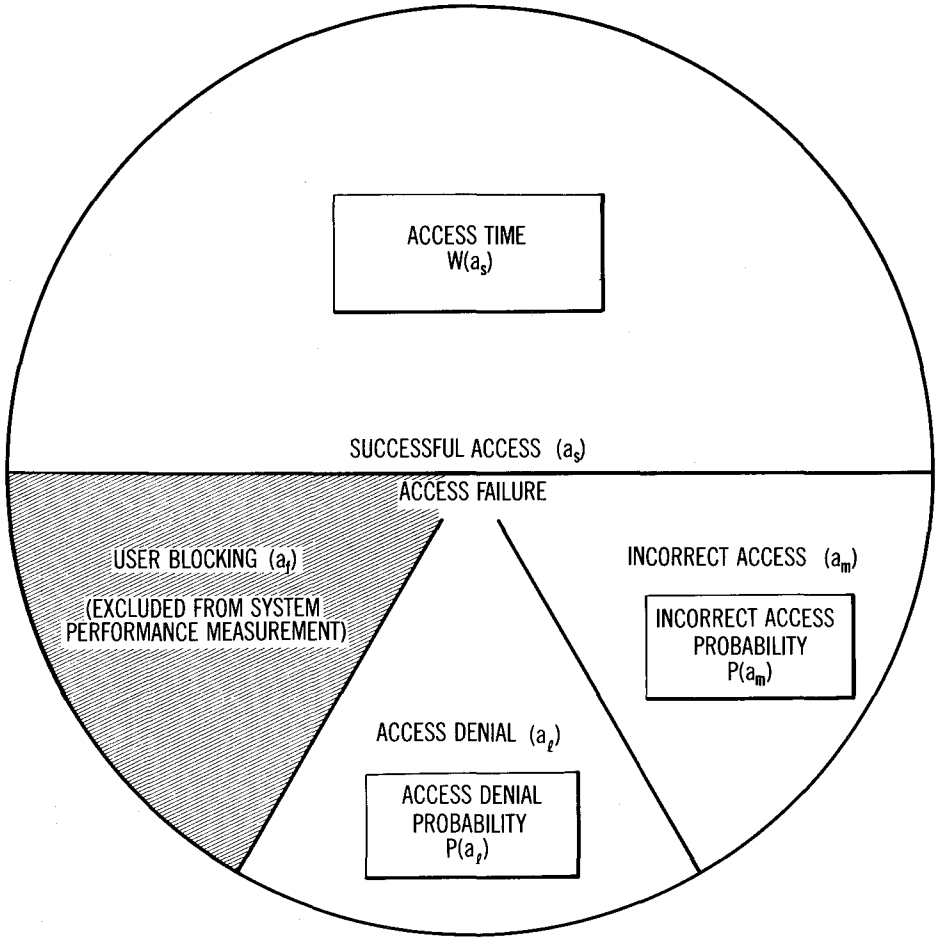
A maximum access time equal to three times the nominal value of the parameter Access time, $3W_N(a_s)$, is defined for performance assessment purposes. Access trials whose performance time exceeds this maximum should be counted as Access Failures.

A.1.2 User Information Transfer Parameters

Interim Federal Standard 1033 specifies bit transfer, block transfer, and message transfer performance in terms of the 14 primary parameters identified in Figure A-2.[23] Narrative definitions for the 10 selected bit transfer and

---

[23]The separate bit transfer, block transfer, and message transfer sample spaces are represented by a single "pie diagram" to simplify the presentation.



ACCESS PARAMETERS

1. Access Time = $W(a_S) = \frac{1}{A_S} \sum_{a_S=1}^{A_S} w(a_S)$
2. Incorrect Access Probability = $P(a_m) = A_m/A'$
3. Access Denial Probability = $P(a_l) = A_l/A'$

DEFINITIONS

A_S = Total number of Successful Access outcomes counted during an access parameter measurement.

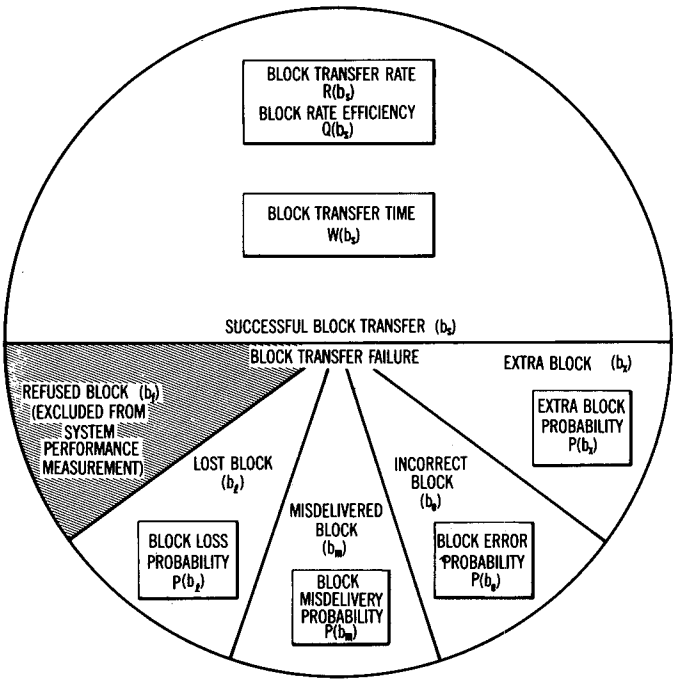
A_m = Total number of Incorrect Access outcomes counted during an access parameter measurement.

A_l = Total number of Access Denials counted during an access parameter measurement.

A' = Total number of access attempts counted during an access parameter measurement: $A_S+A_m+A_l$.

$w(a_S)$ = Value of access time measured on a particular successful access attempt.

Figure A-1. Access parameter definitions.



BIT TRANSFER PARAMETERS

1. Bit Transfer Time $= W(b1_s) = \frac{1}{2(B2_s)} \sum_{b2_s=1}^{B2_s} w_1(b2_s) + \frac{1}{2(B2_s)} \sum_{b2_s=1}^{B2_s} w_2(b2_s)$
2. Bit Error Probability $= P(b1_e) = B1_e/(B1_s+B1_e)$
3. Bit Misdelivery Probability $= P(b1_m) = B1_m/(B1'-B1_\ell-B1_x)$
4. Bit Loss Probability $= P(b1_\ell) = B1_\ell/(B1'-B1_x)$
5. Extra Bit Probability $= P(b1_x) = B1_x/(B1'-B1_\ell)$

BLOCK TRANSFER PARAMETERS

1. Block Transfer Time $= W(b2_s) = \frac{1}{B2_s} \sum_{b2_s=1}^{B2_s} w(b2_s)$
2. Block Error Probability $= P(b2_e) = B2_e/(B2_s+B2_e)$
3. Block Misdelivery Probability $= P(b2_m) = B2_m/(B2'-B2_\ell-B2_x)$
4. Block Loss Probability $= P(b2_\ell) = B2_\ell/(B2'-B2_x)$
5. Extra Block Probability $= P(b2_x) = B2_x/(B2'-B2_\ell)$

MESSAGE TRANSFER PARAMETERS

1. Bit Transfer Rate $= R(b1_s) = \frac{B1_s}{w(b3^*)}$
2. Block Transfer Rate $= R(b2_s) = \frac{B2_s}{w(b3^*)}$
3. Bit Rate Efficiency $= Q(b1_s) = \frac{R(b1_s)}{R_{max}}$
4. Block Rate Efficiency $= Q(b2_s) = \frac{R(b2_s)\cdot n}{R_{max}}$

DEFINITIONS

$B1_s(B2_s)$ = Total number of Successful Bit (Block) Transfer outcomes counted during a UIT performance measurement.

$B1_e(B2_e)$ = Total number of Incorrect Bit (Block) outcomes counted during a UIT performance measurement.

$B1_m(B2_m)$ = Total number of Misdelivered Bit (Block) outcomes counted during a UIT performance measurement.

$B1_\ell(B2_\ell)$ = Total number of Lost Bit (Block) outcomes counted during a UIT performance measurement.

$B1_x(B2_x)$ = Total number of Extra Bit (Block) outcomes counted during a UIT performance measurement.

$w_1(b2_s)$ = Value of bit transfer time measured on the last bit of a particular successful block transfer attempt.

$w_2(b2_s)$ = Value of bit transfer time measured on the first bit of a particular successful block transfer attempt.

$w(b2_s)$ = Value of block transfer time measured on a particular successful block transfer attempt.

$w(b3^*)$ = Duration of a particular UIT performance measurement period.

$B1'$ = Total number of bit transfer outcomes to be included in an individual UIT performance measurement: $B1_s+B1_e+B1_m+B1_\ell+B1_x$.

$B2'$ = Total number of block transfer outcomes to be included in an individual UIT performance measurement. $B2_s+B2_e+B2_m+B2_\ell+B2_x$.

R_{max} = Signaling rate (bits per second).

n = Average block length.

Figure A-2. User information transfer parameter definitions.

block transfer parameters are provided below. For conciseness, only the term "block" is used in these definitions; but it is emphasized that an exactly analogous set of parameters is defined for the bit transfer function. The general symbols b and B refer to either the bit transfer or the block transfer function.[24]

Block Transfer Time $W(b_s)$ - Average value of elapsed time between the start of a block transfer attempt t(b) and Successful block Transfer $t(b_s)$. Elapsed time values are calculated only on block transfer attempts that result in Successful Block Transfer.

Block Error Probability $P(b_e)$ - Ratio of total Incorrect Blocks (B_e) to total successfully delivered blocks (B_s+B_e).

Block Misdelivery Probability $P(b_m)$ - Ratio of total Misdelivered Blocks (B_m) to total transferred blocks ($B'-B_\ell-B_x$).

Block Loss Probability $P(b_\ell)$ - Ratio of total Lost Blocks (B_ℓ) to total transmitted blocks ($B'-B_x$).

Extra Block Probability $P(b_x)$ - Ratio of total Extra Blocks (B_x) to total received blocks ($B'-B_\ell$).

A maximum block transfer time equal to three times the nominal value of the parameter Block Transfer Time, $3W_N(b2_s)$, is defined for performance assessment purposes. Block Transfer trials whose performance time exceeds this maximum should be counted as Block Transfer Failures.

All of the above definitions apply equally to the bit transfer parameters, with two exceptions:

1. The maximum bit transfer time is defined to be equal to the maximum block transfer time, $3W_N(b2_s)$.

2. Bit Transfer Time values are calculated only for the first and last bits of each block, since these are the only bits for which SU output and DU input times are normally available.

The denominators of the bit transfer and block transfer probability parameters have been chosen to normalize the maximum value of each probability to one. Successful Block Transfers and Lost Blocks are expressed as a proportion of total blocks transmitted ($B'-B_x$); Extra Blocks are expressed as a proportion of total blocks received ($B'-B_\ell$); Misdelivered Blocks are expressed as a proportion of total blocks transferred between the source and

---

[24]Separate equation definitions for the bit transfer and block transfer parameters are provided in Figure A.2.

destination in question ($B'-B_\ell-B_x$); and Incorrect Blocks are expressed as a proportion of total blocks transferred between the source and destination in question, excluding misdelivered blocks ($B'-B_\ell-B_x-B_m$). The latter subset is expressed more simply as (B_s+B_e), the "successfully delivered" blocks.

Narrative definitions for the four primary message transfer parameters are provided below. Again, only the term "block" is used in these definitions; but an exactly analogous set of parameters is defined for the bit transfer function.

> Block Transfer Rate $R(b_s)$ - Total number of Successful Block Transfers (B_s) counted during a performance measurement period, divided by the duration of the period w(b3*). The duration w(b3*) is measured in User Information Transfer Time.

> Block Rate Efficiency $Q(b_s)$ - Ratio of the product of the Block Transfer Rate $R(b_s)$ and the Average Block Length (n) to the Signalling Rate of the communication service, R(max).

R_{max} is a constant characteristic of the communication service interconnecting the users, as described in Section A.2.2.

A.1.3 Disengagement Parameters

Interim Federal Standard 1033 specifies disengagement performance in terms of the two primary parameters identified in Figure A-3. Narrative definitions for the selected disengagement parameters are provided below.

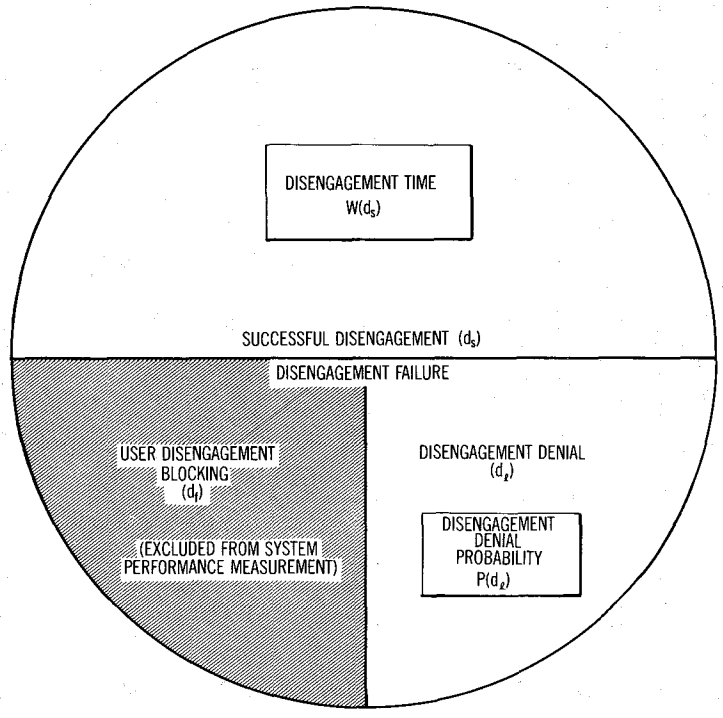
> Disengagement Time $W(d_s)$ - Average value of elapsed time between the start of a disengagement attempt t(d) and Successful Disengagement $t(d_s)$. Elapsed time values are calculated only on disengagement attempts that result in Successful Disengagement.

> Disengagement Denial Probability $P(d_\ell)$ - Ratio of total disengagement attempts that result in Disengagement Denial (D_ℓ) to total disengagement attempts included in the reduced sample (D').

A maximum disengagement time equal to three times the nominal value of the parameter Disengagement Time, $3W_N(d_s)$, is defined for performance assessment purposes. Disengagement trials whose performance time exceeds this maximum should be counted as Disengagement Failures.

A.1.4 Secondary Parameters

Interim Federal Standard 1033 specifies the availability performance of a data communication service in terms of the three "secondary" parameters



DISENGAGEMENT PARAMETERS

1. Disengagement Time $= W(d_S) = \frac{1}{D_S} \sum_{d_S=1}^{D_S} w(d_S)$

2. Disengagement Denial Probability $= P(d_f) = D_\ell / D'$

DEFINITIONS

D_S = Total number of Successful Disengagement outcomes counted during a disengagement parameter measurement.

D_f = Total number of Disengagement Denials counted during a disengagement parameter measurement.

D' = Total number of disengagement attempts counted during a disengagement parameter measurement: $D_s + D_\ell$.

$w(d_S)$ = Value of disengagement time measured on a particular successful disengagement attempt.

Figure A-3. Disengagement parameter definitions.

identified in Figure A-4. Narrative definitions for these three parameters are provided below.

Service Time Between Outages W(y*) - Average value of elapsed user information transfer time between the start t(y) and the end t(z) of the secondary function service continuation.

Outage Duration W(z*) - Average value of elapsed user information transfer time between the start t(z) and the end t(y) of the secondary function service restoral.

Outage Probability $P(b3_z)$ - Ratio of total message transfer attempts resulting in the secondary outcome outage state ($b3_z$) to total message transfer attempts included in the reduced sample (B3').

The secondary function "service continuation" consists of maintaining the telecommunication service in the operational service state continuously (during user information transfer time), without transition to the outage state. The secondary function "service restoral" consists of returning the telecommunication service from the outage state to the operational state.

A.1.5 Ancillary Parameters

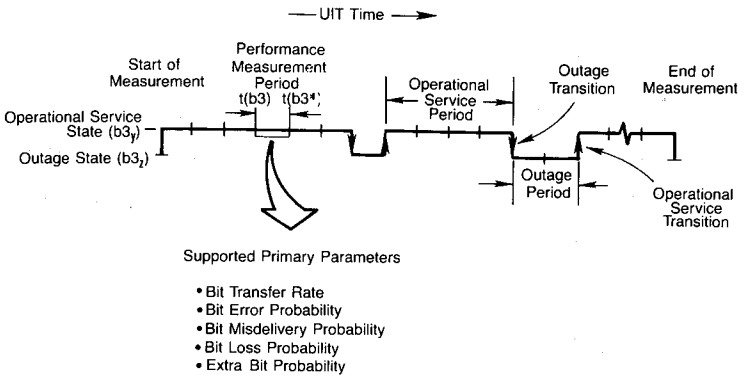
Interim Federal Standard 1033 defines the four ancillary parameters on the basis of three variables defined as follows.

$w(g^*)$ = Elapsed time between the start and the end of an individual primary function performance period.

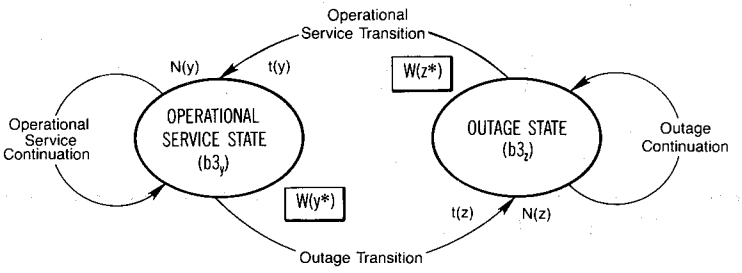
$w_u(g^*)$ = Total time (within the primary function performance period) during which aggregate responsibility for advancing the function to completion rests with the users.

$w_v(g^*)$ = Total time (within the primary function performance period) during which aggregate responsibility for advancing the function to completion is "split" between a user and a half-system.

The ancillary parameters are calculated on the basis of a sample of successful performance trials, G_s. The random variables $w(g^*)$, $w_u(g^*)$, and $w_v(g^*)$ are thus specialized to their "successful outcome" counterparts, $w(g_s)$, $w_u(g_s)$, and $w_v(g_s)$. The average values of these variables are represented by the symbols $W(g_s)$, $W_u(g_s)$, and $W_v(g_s)$, respectively. The ancillary parameters $p(g)$ are defined as ratios of these average values, as follows:



a. Secondary State Sequence



b. Secondary State Diagram

SECONDARY PARAMETERS

1. Service Time Between Outages $= W(y^*) = \frac{1}{Y} \sum_{y=1}^{Y} w(y^*)$

2. Outage Duration $= W(z^*) = \frac{1}{Z} \sum_{z=1}^{Z} w(z^*)$

3. Outage Probability $= P(b3_z) = B3_z/B3'$

DEFINITIONS

$B3_z$ = Total number of Outage state outcomes counted during a secondary parameter measurement.

$t(y)$ = Time most recent Operational Service transition occurred.

$t(z)$ = Time most recent Outage transition occurred.

$w(y^*)$ = Value of Service Time between Outages measured on a particular transition to the Outage state: $t(z) - t(y)$.

$w(z^*)$ = Value of Outage Duration measured on a particular transition to the Operational Service state: $t(y) - y(z)$.

Y = Total number of Operational Service periods counted during a secondary parameter measurement.

Z = Total number of Outage periods counted during a secondary parameter measurement.

$B3'$ = Total number of message transfer trials counted during a secondary parameter measurement.

c. Secondary Parameter Definitions

Figure A-4. Secondary model and parameter definitions.

$$p(g) = \frac{W_u(g_s) + 0.5\ W_v(g_s)}{W(g_s)}$$

The parameters p(g) are termed "user performance time fractions"; they express the average proportion of successful primary function performance time that is attributable to user delay. Periods of unilateral user responsibility contribute to W_u, and are "weighted" at their full value in computing p(g); periods of "split" responsibility contribute to W_v, but are "weighted" at only half their full value to reflect the equal division of responsibility between the user and system entities.

Narrative definitions for the four specific ancillary parameters included in the standard are provided below.

User Access Time Fraction p(a) - Ratio of average user access time, $W_u(a_s) + 0.5\ W_v(a_s)$, to average Access Time, $W(a_s)$, measured over a sample of successful access attempts, A_s.

User Block Transfer Time Fraction p(b2) - Ratio of average user block transfer time, $W_u(b2_s) + 0.5\ W_v(b2_s)$, to average Block Transfer Time, $W(b2_s)$, measured over a sample of successful block transfer attempts, $B2_s$.

User Message Transfer Time Fraction p(b3) - Ratio of average user message transfer time, $W_u(b3_y) + 0.5\ W_v(b3_y)$, to average message transfer time, $W(b3_y)$, measured over a sample of successful message transfer attempts, $B3_y$. "Successful" message transfer attempts are those encountering the secondary outcome Operational Service state ($b3_y$).

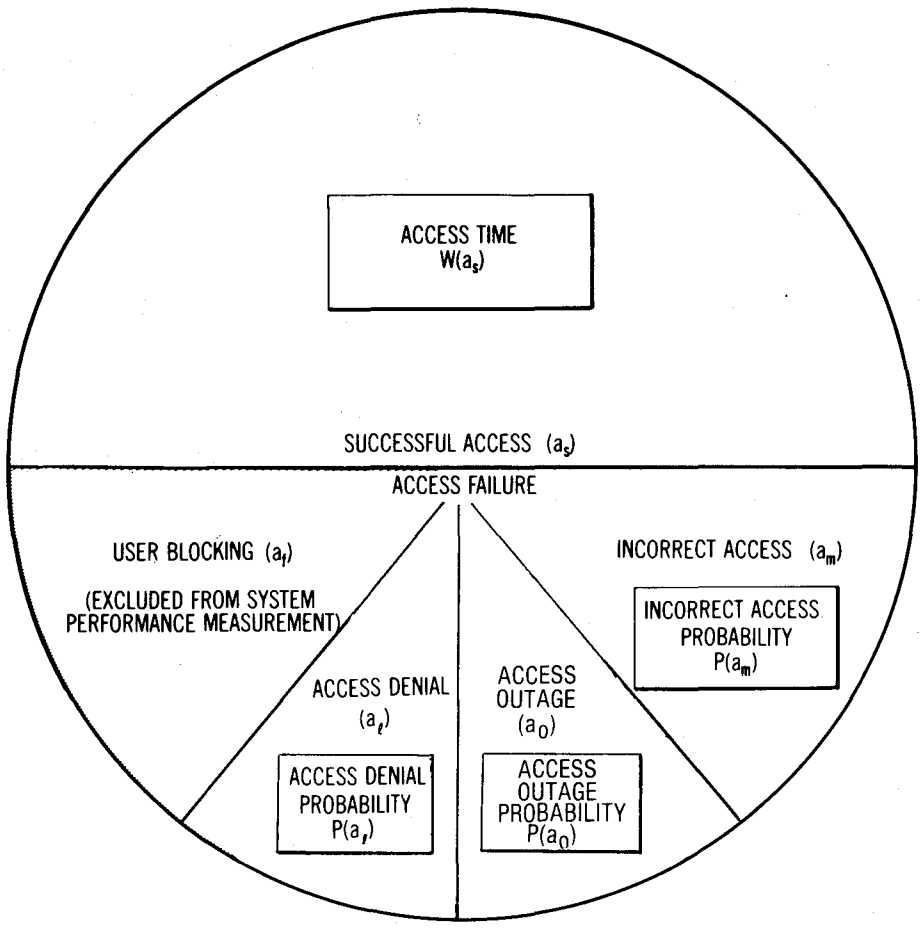
User Disengagement Time Fraction p(d) - Ratio of average user disengagement time, $W_u(d_s) + 0.5\ W_v(d_s)$, to average Disengagement Time, $W(d_s)$, measured over a sample of successful disengagement attempts, D_s.

A.2 ANSI X3.102 Parameters

The 21 performance parameters specified in proposed American National Standard X3.102 are defined as follows. Refer to the standard and its supporting reports for more explicit definitions of the associated outcomes.

A.2.1 Access Parameters

Figure A-5 summarizes the possible access outcomes and their associated performance parameters. Narrative definitions for the selected parameters are provided below.



ACCESS PARAMETERS

1. Access Time = $W(a_S) = \frac{1}{A_S} \sum_{a_S=1}^{A_S} w(a_S)$
2. Incorrect Access Probability = $P(a_m) = A_m/A'$
3. Access Denial Probability = $P(a_\ell) = A_\ell/A'$
4. Access Outage Probability = $P(a_O) = A_O/A'$

DEFINITIONS

A' = Total number of access attempts counted during an access parameter measurement: $A_S+A_m+A_\ell+A_O$

A_S = Total number of Successful Access outcomes counted during an access parameter measurement.

A_ℓ = Total number of Access Denials counted during an access parameter measurement.

A_O = Total number of Access Outage outcomes counted during an access parameter measurement.

A_m = Total number of Incorrect Access outcomes counted during an access parameter measurement.

$t(a)$ = Time a particular access attempt starts

$t(a_S)$ = Time Successful Access is attained on a particular access attempt.

$w(a_S)$ = Value of access time measured on a particular successful access attempt: $t(a_S)-t(a)$

Figure A-5. Access parameter definitions.

Access Time $w(a_s)$. Access Time is the average value of elapsed time between the start of an access attempt t(a) and Successful Access $t(a_s)$. Elapsed time values are calculated only on access attempts that result in Successful Access.

Incorrect Access Probability $P(a_m)$. Incorrect Access Probability is the ratio of total access attempts that result in Incorrect Access (A_m) to total access attempts included in the reduced access sample (A').

Access Denial Probability $P(a_\ell)$. Access Denial Probability is the ratio of total access attempts that result in Access Denial (A_ℓ) to total access attempts included in the reduced access sample (A').

Access Outage Probability $P(a_o)$. Access Outage Probability is the ratio of total access attempts that result in Access Outage (A_o) to total access attempts included in the reduced access sample (A').

Access timeout occurs (i.e., an access attempt is declared a failure for performance assessment purposes) whenever the duration of an individual access attempt exceeds $3W_N(a_s)$, i.e., three times the specified value of the parameter Access Time.

A.2.2 User Information Transfer Parameters

Figures A-6a, A-6b, and A-6c summarize the possible user information transfer outcomes and the associated user information transfer parameters. Narrative definitions for the selected parameters are provided below.

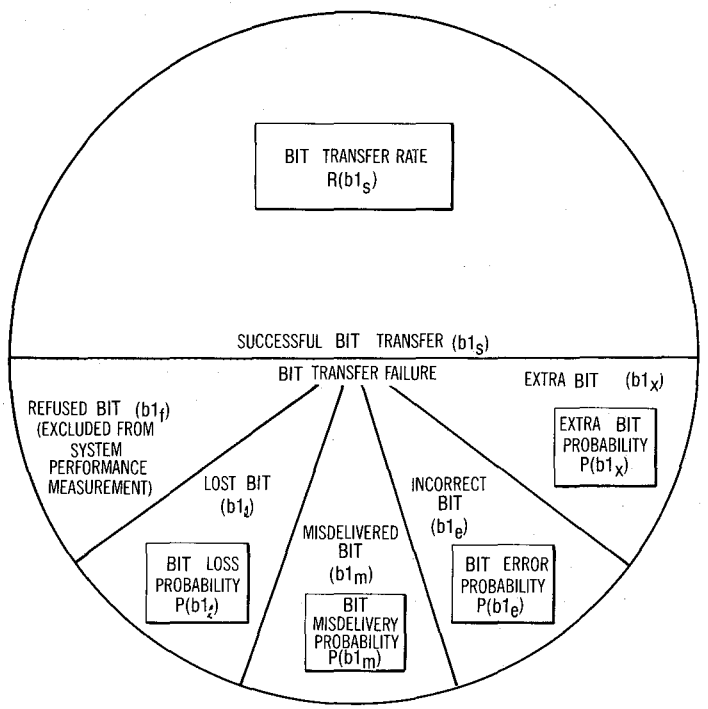
The defined user information transfer parameters include both bit-oriented and block-oriented parameters. The bit-oriented parameters are included primarily to facilitate comparison of systems with different block lengths.

Block Transfer Time $W(b2_s)$. Block Transfer Time is the average value of elapsed time between the start of a block transfer attempt t(b2) and Successful Block Transfer $t(b2_s)$. Elapsed time values are calculated only on blocks that are successfully transferred.

Bit Error Probability $P(b1_e)$. Bit Error Probability is the ratio of total Incorrect Bits ($B1_e$) to total successfully transferred bits plus Incorrect Bits ($B1_s + B1_e$).

Block Error Probability $P(b2_e)$. Block Error Probability is the ratio of total Incorrect Blocks ($B2_e$) to total successfully transferrd blocks plus Incorrect Blocks ($B2_s + B2_e$).

Bit Misdelivery Probability $P(b1_m)$. Bit Misdelivery Probability is the ratio of total Misdelivered Bits ($B1_m$) to total bits transferred between a specified source and destination ($B1' - B1_\ell - B1_x$).



BIT TRANSFER PARAMETERS

1. Bit Loss Probability = $P(b1_\ell) = B1_\ell/(B1'-B1_X)$
2. Bit Misdelivery Probability = $P(b1_m) = B1_m/(B1'-B1_\ell-B1_X)$
3. Bit Error Probability = $P(b1_e) = B1_e/(B1_S+B1_e)$
4. Extra Bit Probability = $P(b1_X) = B1_X/(B1'-B1_\ell)$
5. User Information Bit Transfer Rate = $R(b1_S) = \frac{B1_S}{w(b3^*)}$

DEFINITIONS

$B1'$ = Total number of bit transfer outcomes to be included in an individual UIT performance measurement (All bit transfer outcomes except $b1_1$).

$B1_S$ = Total number of Successful Bit Transfer outcomes counted during a UIT performance measurement.

$B1_f$ = Total number of Refused Bit outcomes counted during a UIT performance Measurement.

$B1_\ell$ = Total number of Lost Bit outcomes counted during a UIT performance measurement.

$B1_m$ = Total number of Misdelivered Bit outcomes counted during a UIT performance measurement.

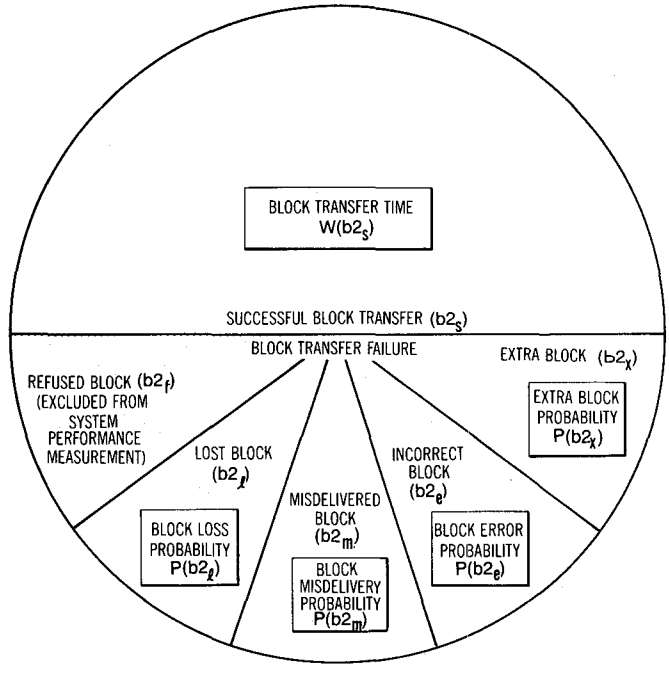
$B1_e$ = Total number of Incorrect Bit outcomes counted during a UIT performance measurement.

$B1_X$ = Total number of Extra Bit outcomes counted during a UIT performance measurement.

$w(b3^*)$ = Greater of input time $w(b3_i)$ or output time $w(b3_o)$ required to transfer sample to/from the system.

UIT = User Information Transfer

Figure A-6a. User information bit transfer parameter definitions.



BLOCK TRANSFER PARAMETERS

1. Block Transfer Time = $W(b2_s) = \frac{1}{B2_s} \sum_{b2_s=1}^{B2_s} w(b2_s)$
2. Block Loss Probability = $P(b2_\ell) = B2_\ell/(B2'-B2_x)$
3. Block Misdelivery Probability = $P(b2_m) = B2_m/(B2'-B2_\ell-B2_x)$
4. Block Error Probability = $P(b2_e) = B2_e/(B2_s+B2_e)$
5. Extra Block Probability = $P(b2_x) = B2_x/(B2'-B2_\ell)$

DEFINITIONS

$B2'$ = Total number of block transfer outcomes to be included in an individual UIT performance measurement (All block transfer outcomes except $b2_f$).

$B2_s$ = Total number of Successful Block Transfer outcomes counted during a UIT performance measurement.

$B2_f$ = Total number of Refused Block outcomes counted during a UIT performance measurement.

$B2_\ell$ = Total number of Lost Block outcomes counted during a UIT performance measurement.

$B2_m$ = Total number of Misdelivered Block outcomes counted during a UIT performance measurement.

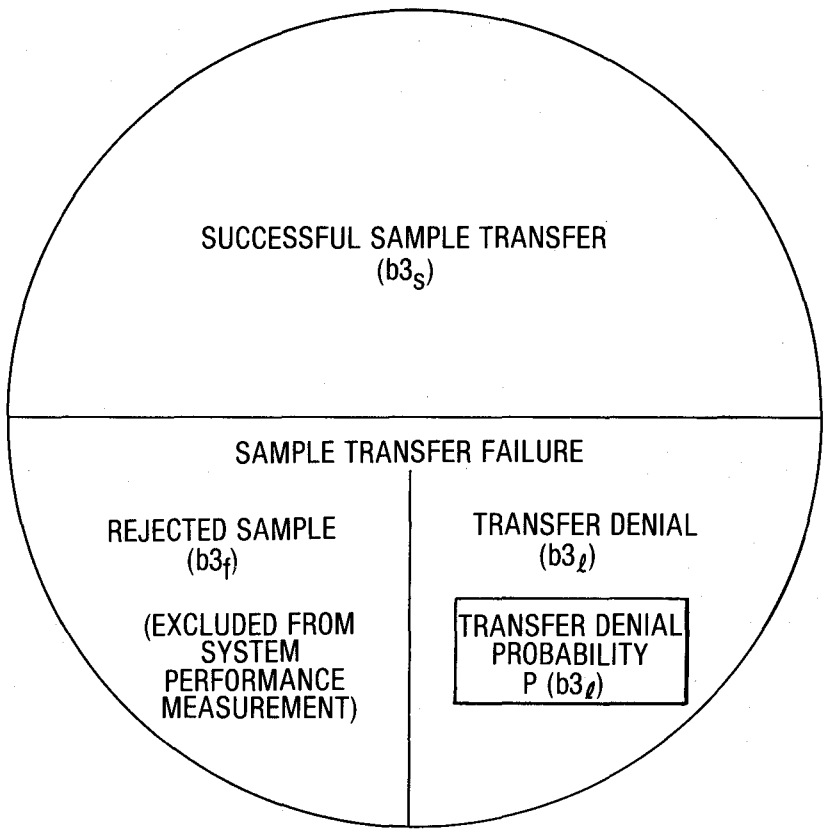
$B2_e$ = Total number of Incorrect Block outcomes counted during a UIT performance measurement.

$B2_x$ = Total number of Extra Block outcomes counted during a UIT performance measurement.

$w(b2_s)$ = Value of block transfer time measured on a particular successful block transfer attempt.

UIT = User Information Transfer.

Figure A-6b. User information block transfer parameter definitions.



Transfer Denial Probability = $B3_\ell/B3'$

DEFINITIONS

B3′ = Total number of transfer samples to be included in a Transfer Denial Probability determination.

$B3_\ell$ = Total number of Transfer Denial outcomes counted in a Transfer Denial Probability determination.

Figure A-6c. Transfer denial probability definition.

Block Misdelivery Probability $P(b2_m)$. Block Misdelivery Probability is the ratio of total Misdelivered Blocks ($B2_m$) to total blocks transferred between a specified source and destination ($B2' - B2_\ell - B2_x$).

Extra Bit Probability $P(b1_x)$. Extra Bit Probability is the ratio of total Extra Bits ($B1_x$) to total bits received by a particular destination user ($B1' - B1_\ell$).

Extra Block Probability $P(b2_x)$. Extra Block Probability is the ratio of total Extra blocks ($B2_x$) to total received blocks ($B2' - B2_\ell$).

Bit Loss Probability $P(b1_\ell)$. Bit Loss Probability is the ratio of total Lost Bits ($B1_\ell$) to total transmitted bits ($B1' - B1_x$).

Block Loss Probability $P(b2_\ell)$. Block Loss Probability is the ratio of total Loss Blocks ($B2_\ell$) to total transmitted blocks ($B2' - B2_x$).

User Information Bit Transfer Rate $R(b1_s)$. User Information Bit Transfer Rate is the total number of Successful Bit Transfer outcomes ($B1_s$) in an individual transfer sample divided by the input/output time for that sample. The Input/Output Time $w(b3^*)$ for a transfer sample is the larger of the input time $w(b3_i)$ or the output time $w(b3_o)$ for that sample (Figure A-7).

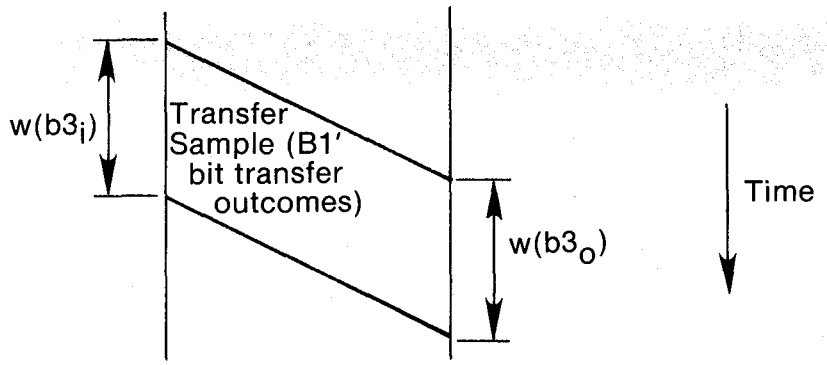
Transfer Denial Probability $P(b3_\ell)$. Transfer Denial Probability is the ratio of total Transfer Denials ($B3_\ell$) to total transfer samples counted ($B3'$). Transfer Denial ($b3_\ell$) occurs whenever the performance as determined for a transfer sample is worse than the threshold of acceptability for any of four supported transfer parameters as a result of system degradation. Transfer Denial Probability is a sampled measure of unavailability.

Block transfer timeout occurs (i.e., a block transfer attempt is declared a failure for performance assessment purposes) whenever the duration of an individual block transfer period exceeds $3W_N(b2_s)$, i.e., three times the specified value of the parameter Block Transfer Time.

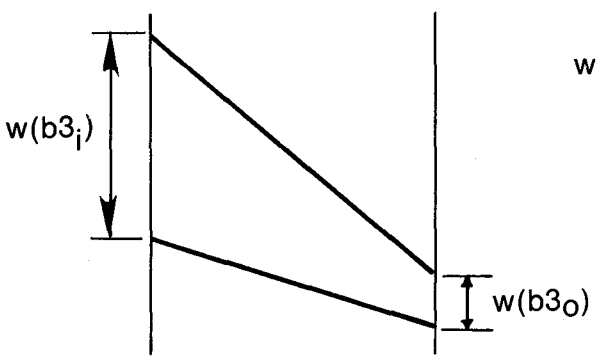
A.2.3 Disengagement Parameters

Figure A-8 summarizes the possible disengagement outcomes and the associated disengagement parameters. Narrative definitions for the selected parameters are provided below.

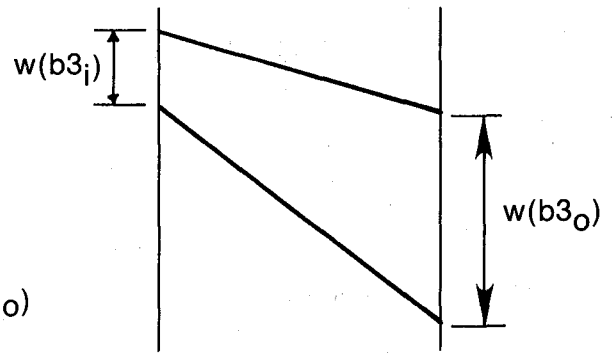
There is a separate disengagement function associated with each end user. The outcomes are determined separately for each function, and the outcomes for all users in a data communication session are weighted equally in determining the disengagement parameter values. Separate disengagement parameters may be



Case 1. No rate conversion : $w(b3_i) = w(b3_o)$



Case 2. Rate increase:
$w(b3_i) < w(b3_o)$

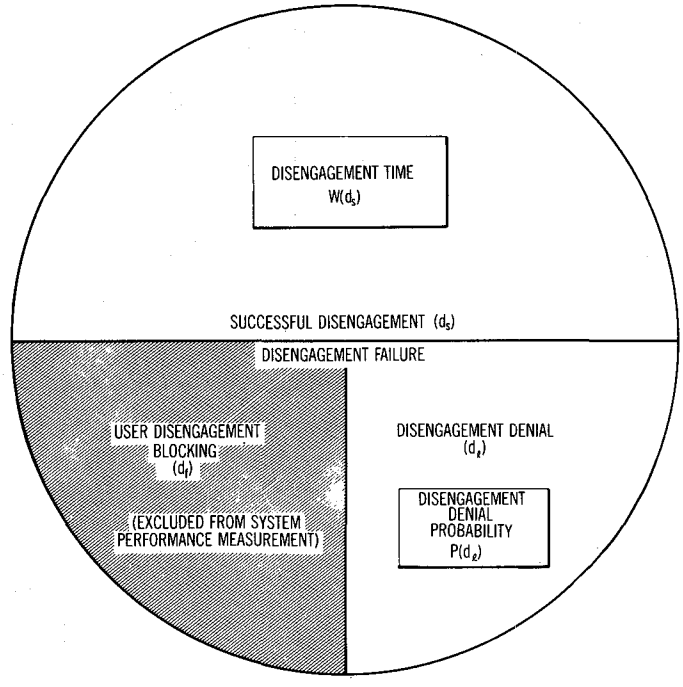


Case 3. Rate reduction :
$w(b3_i) > w(b3_o)$

$$\text{User Information Bit Transfer Rate } R(b1_s) = \frac{B1_s}{\text{Max}\left[w(b3_i) \text{ or } w(b3_o)\right]}$$

$B1_s$ = Total Successful Bit Transfer outcomes in the transfer sample.

Figure A-7. User information bit transfer rate.



DISENGAGEMENT PARAMETERS

1. Disengagement Time $= W(d_s) = \frac{1}{D_s} \sum_{d_s=1}^{D_s} w(d_s)$

2. Disengagement Denial Probability $= P(d_f) = D_f/D'$

DEFINITIONS

D_s = Total number of Successful Disengagement outcomes counted during a disengagement parameter measurement.

D_f = Total number of Disengagement Denials counted during a disengagement parameter measurement.

$t(d)$ = Time a particular disengagement attempt starts.

$t(d_s)$ = Time Successful Disengagement is attained on a particular disengagement attempt.

$w(d_s)$ = Value of disengagement time measured on a particular successful disengagement attempt: $t(d_s)-t(d)$

Figure A-8. Disengagement parameter definitions.

calculated for each end user interface in cases where significant performance differences are expected.

Disengagement Time $W(d_s)$. Disengagement Time is the average value of elapsed time between the start of a disengagement attempt t(d) for a particular user and Successful disengagement $t(d_s)$ of that user. Elapsed time values are calculated only on disengagement attempts that result in Successful Disengagement.

Disengagement Denial Probability $P(d_\ell)$. Disengagement Denial Probability is the ratio of total disengagement attempts that result in Disengagement Denial (D_ℓ) to total disengagement attempts included in the reduced disengagement sample (D').

Disengagement timeout occurs (i.e., a disengagement attempt is declared a failure for performance assessment purposes) whenever the duration of an individual disengagement attempt exceeds $3W_N(d_s)$, i.e., three times the specified value of the parameter Disengagement Time.

A.2.4 Ancillary Parameters

The ancillary parameters are developed by dividing the overall performance time for an associated primary function into subintervals, each corresponding to a period of user or system responsibility. Each ancillary parameter expresses the average proportion of the total primary function performance time that is attributable to user delay.

There are ancillary parameters for: access, block transfer, and disengagement. In addition, there is an ancillary parameter that expresses the user influence on User Information Bit Transfer Rate. No ancillary parameter is defined for the bit transfer function, since the Lost Bit (system nonperformance) and Refused Bit (user nonperformance) outcomes are distinguished from each other by reference to the associated block transfer outcomes. Narrative definitions for the selected ancillary parameters are provided below.

User Fraction of Access Time $U(a_s)$. The User Fraction of Access Time $U(a_s)$ is the ratio of the average access time for which the user is responsible to the average total access time, measured over a series of successful access attempts.

User Fraction of Block Transfer Time $U(b2_s)$. The User Fraction of Block Transfer Time $U(b2_s)$ is the ratio of the average block transfer time for which the user is responsible to the average total block transfer time, measured over a series of successful block transfer attempts.

User Fraction of Input/Output Time $U(b3_s)$. The User Fraction of Input/Output Time $U(b3_s)$ is the ratio of the average user input/output time to the average total input/output time for a transfer sample, measured over a series of successful transfer samples. The User Input/Output Time for a transfer sample is that portion of the sample input/output time for which the user is responsible.

User Fraction of Disengagement Time $U(d_s)$. The User Fraction of Disengagement Time $U(d_s)$ is the ratio of the average disengagement time for which the user is responsible to the average total disengagement time, measured over a series of successful disengagement attempts.

APPENDIX B: LISTINGS OF THE ON-LINE DATA EXTRACTION PROGRAMS

This appendix contains listings of the XMIT and RECV programs used for on-line data extraction in the ARPANET experiment. These programs were written in the "C" Language for a UNIX Version 6 compiler on a PDP-11/40. Many of the constants in both programs were identical, so they were incorporated in a pair of header files called "include" files. Since these files are part of the compiled program, they are shown following the program listings. The first "include" file is NET.C and defines the constants used for opening and establishing the network connection. The second "include" file relates to both the clock (satellite receiver) parameters and formats and the default parameters for the test. Corresponding to the two types of tests (access/disengagement and user information transfer), there were two header files called CLOCK.OVH.H and CLOCK.USR.H with only minor differences between the two. Inclusion of one or the other determined the character of the compiled software. In following the accepted convention, all source text files for "C" were named, for example,

xmit.c , or recv.c ,

and the compiled versions of the same program were

xmit and recv .

A simplified flow chart of the two programs and their interaction is presented in Figure B-1. Since "C" is a modular language, the main program follows the flow sequence in the chart in a straightforward manner.

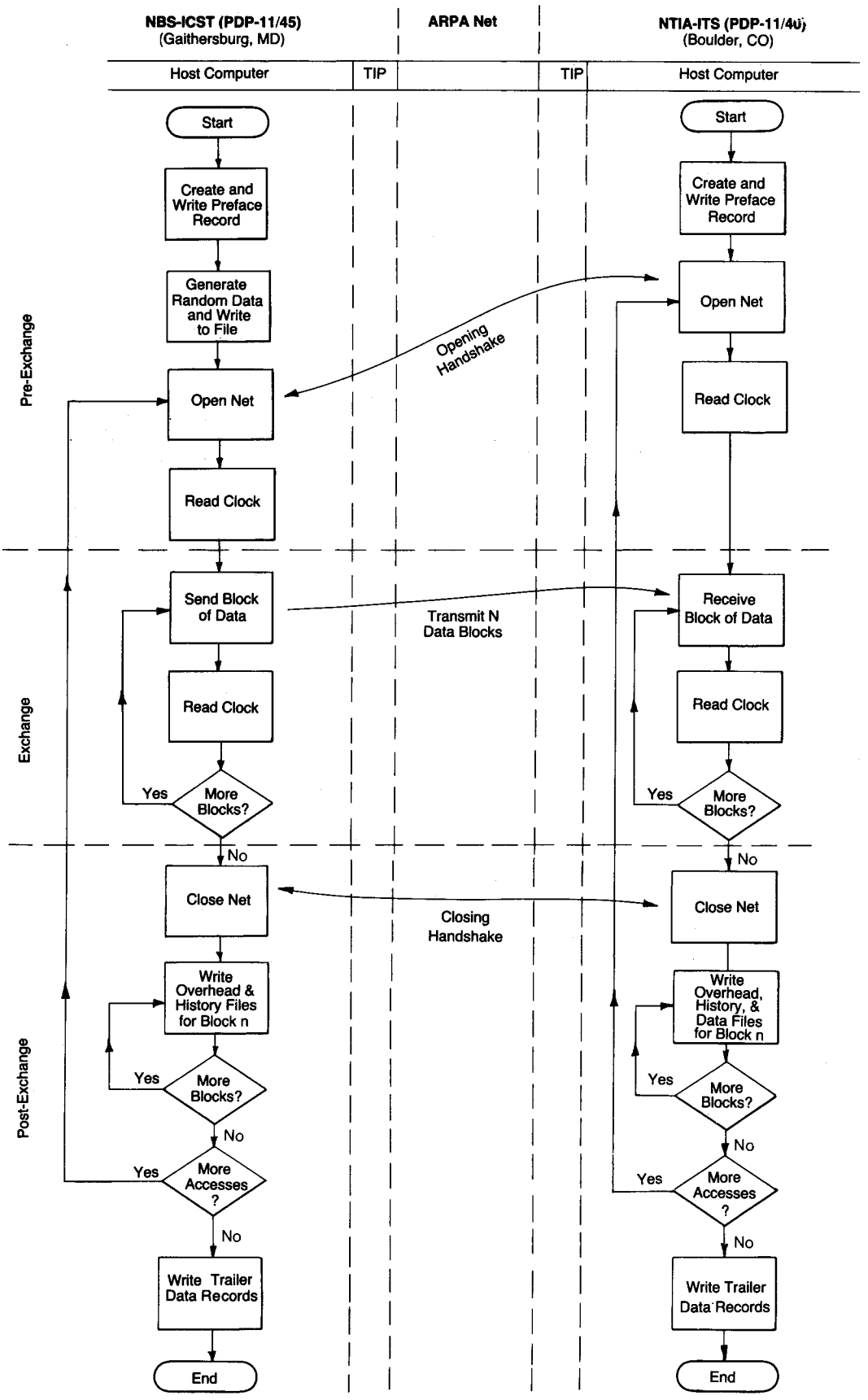


Figure B-1. Flow chart of on-line data extraction programs.

```
#
/*      main program to output (send) data from source over arpanet
*       written:        David Wortendyke        september 30, 1980
*       modified:       evi nemeth              december 1980
*       modified:       evi nemeth              february 1981
*                       to read preface header from a file.
*       commented:      Dana Grubb              april 1981
        rewritten:      Dave Wortendyke         april-may 1981
                        Version 2.0 -- Overhead measurements
*       added quirk:    dave wortendyke         oct 20, 1981
                        solves problem at nbs-unix of close net = -1
*       changed:        dave wortendyke         oct 27, 1981
                        dropped net warm-up (for >64 bytes)
*
*       Inputs:  Program- # blocks,  # of bytes/block, # transactions
                 Files- One: preface.x
*                Hardware- WWV Satellite Receiver clock
*                Library-
*       Outs:    Program- N blocks of BLOCKSIZE bytes in each of
                          T transactions
                 Files- Three:  history.x, overhead.x, data.x
*                Hardware- Network controller/TIP
*                Library-printf
*/

#include "clock.usr.h"

/*      clock.usr.h --> BLOCKSIZE=512, NBLOCKS=20, NTRANSACTIONS=1     */
/*      clock.ovh.h --> BLOCKSIZE= 64, NBLOCKS= 1, NTRANSACTIONS=160    */

#include "net.h"    /* open type bit assignments */

#define xmit main   /* calling "xmit" gives the "main" program */

struct openparams               /* struct for making parameterized connections */
{
        char o_op;              /* opcode for kernel & daemon - unused here */
        char o_type;            /* type for connection see defines below */
        int  o_id;              /* id of file for kernel & daemon - unused here */
        int  o_lskt;            /* local socket number either abs or rel */
        int  o_fskt[2];         /* foreign skt either abs or rel */
        char o_frnhost;         /* for connection to specific host nums */
        char o_bsize;           /* bytesize of connection telnet demands 8 */
        int o_nomall;           /* initial allocation bits and msgs */
        int o_timeo;            /* num of secs to wait before timing out */
        int  o_relid;           /* fid of file to base a data connection on */
} openparams;

struct time             /* for satelite clocks time entries */
{
        int hour;       /* nours */
        int min;        /* minutes */
        int sec;        /* seconds */
        int frac;       /* fractional seconds in milliseconds */
} sattime;
```

```
char    databuf[BUFFRSIZE];     /* data buffer for transmission across the network */
char    timebuf[2*MAXBLOCKS+5][CLOCKRECORDLENGTH+1];    /* buffer for satellite
                        clock times, +1's needed for null, and +5 for overhead times */

int     date[3];        /* date: year, month, day */
int     ttyline[3];     /* line characteristics of ttyX (from gtty) as a tty */
int     debug;          /* flag for debug-trace printouts       */
int     overhdbuf[2*MAXBLOCKS+5];       /* buffer to store transaction state changes */



xmit(argc,argv)
                /* Arguments are for when program begins executing.
                 * argc = number of command line arguments, counting
                 *        0 for "xmit"and 1 for number of blocks.
                 * argv = pointer to array of character strings that
                 *        contain arguments (one per string). In this case,
                 *        argv[1] contains the number of blocks to xmit.
                 *        (The default = 20)
                 */

int argc;
char *argv[];
{

        register int i;
        int     j;
        int     byteswritten;
        int     bytessent[MAXBLOCKS];   /* number bytes actually sent/block */
        int     numblocks;              /* number of blocks to be sent */
        int     bytes_block;            /* number of bytes per block   */
        int     numaccesses;            /* number of network accesses  */
        int     transactnum;            /* acccess or transaction number */
        int     totalbytes;             /* total # bytes to be sent     */
        int     fd_net;                 /* file descriptor, arpanet network */
        int     fd_history;             /* file descriptor, history file */
        int     fd_overhead;            /* file descriptor, overhead file */
        int     fd_clock;               /* file descriptor, satellite clock */
        int     ivec[2];                /* system time vector, before transmission */
        int     tvec[2];                /* system time vector, after transmission */
        char    *pbuf;                  /* pointer to random data buffer */
        char    *filename;              /* pointer to file name         */
        int     baud;

        int     time();                 /* time of day, system clock */
        char    *ctime();               /* converts system time to ascii */
        int     inchar;                 /* input character from operator */
        int     nclockreads;            /* number of satellite clock reads */
```

```
time(ivec);   /* Read UNIX system clock. */
printf ("\n---------- network transmission ---------- ");
printf ("\ntest beginning,");
printf (" %s ", ctime(ivec));

numblocks = (argc > 1) ? atoi(argv[1]) : NBLOCKS;

        /* numblocks made equal to desired number of blocks per run
         * This is a "conditional expression", meaning:
         * if the term before the question mark is not = 0,
         * then the term after the question is the value;
         * otherwise, the term after the colon is the value.
         * In other words: if argc >1, then the integer form of
         * argv[1] is the value of "numblocks"; otherwise,
         * set = NBLOCKS (in the "clock.h" file), which = 20.
         */


bytes_block = (argc > 2) ? atoi(argv[2]) : BLOCKSIZE;
        /* Sets the number of bytes per block to BLOCKSIZE
         * if there is no second argument, otherwise it sets
         * the bytes per block = the value of the second arg  */

numaccesses = (argc > 3) ? atoi(argv[3]) : NTRANSACTIONS;

if (numblocks > MAXBLOCKS)
{
        printf("/nMust use %d blocks or fewer, program terminating.", MAXBLOCKS);
        exit();
}

totalbytes = numblocks * bytes_block * numaccesses;
if (totalbytes > BUFFRSIZE )
{
        printf("\n ---> Request to send more data than in buffer !");
        printf("\n      Program terminating.\n");
        exit();
}

printf ("\n%d blocks of %d bytes to be sent ",numblocks, bytes_block);
printf ("for each of %d accesses,  ", numaccesses);
printf ("= %d total bytes", totalbytes);

debug = NO;     /* Change this to YES if debug is desired */

pbuf = &databuf[0];        /* pbuf points to contents of databuf */

fd_clock = openclock(); /* set tty interface to proper mode for
                         * sat clock, and return file descriptor */

initializeclock(fd_clock);      /* reset sat clock, format sat clock,
                                 * read initial record from both
                                 * clocks, and compare clocks */

filename = "history.x";
fd_history = preface(filename);    /* Write preface record for history
```

```
                                * file. The preface routine returns
                                * a file descriptor for history file. */

        filename = "overhead.x";
        fd_overhead = preface(filename);    /* Write preface record for
                                             * overhead file    */


        randomdata();   /* Creates 10240 bytes of pseudo random test data */
                /* unless the data already exists */

        openparams.o_type = O_SEND;
        openparams.o_lskt   = 277;
        openparams.o_fskt[1] = 10;

        time(ivec);   /* Read UNIX system clock. */
                      /* ivec is the time for network open. */

        transactnum = 0;

      start:
      while (transactnum < numaccesses)
      {
        sleep(10);      /* Wait 10 seconds to be sure the recv program is up */
        nice(-45);      /* Have UNIX give us higher priority for service */

        printf("\nAttempting open # %d, ", transactnum + 1);


        readclock(fd_clock,0);  /* Read satellite clock. */
        overhdbuf[0] = 23;
        nclockreads = 1;

        fd_net = open("/dev/net/ntia-its",&openparams);
                /* try to open connection to ntia-its */

        readclock(fd_clock,1);  /* Read satellite clock. */

        if (fd_net <= 0)
        {
                overhdbuf[1] = 54;

                dupclocktime(2);        /* Duplicate the last clock time into
                                         * row 1 (2nd time)
                                         */
                overhdbuf[2] = 11;
                nclockreads = 3;

                record_ovrhd(fd_overhead,nclockreads);
                printf ("Open denied, a(bort) or c(ontinue) ? ");
                if (answer() == 0)
                        exit();         /* Abort = 0 = False    */
                else
                        goto start;     /* Continue = 1 = True  */
        }
```

```
        else
        {
                overhdbuf[1] = 32;
                nclockreads = 2;
                printf("Opened, ");
        }


        readclock(fd_clock,2);  /* Read satellite clock.  */
                                /* The 2 means put the time at the
                                * 2nd row of the buffer used by readclock. */
        /* DATA TRANSMISSION LOOP FOLLOWS */

        for (i=0;i<numblocks;i++)
        {
                overhdbuf[nclockreads]= 23;
                nclockreads =+ 1;

                bytessent[i] = write(fd_net,pbuf+bytes_block*(transactnum*numblocks+i),bytes_block);

                        /* write arguments are the file descriptor,
                         * pointer to row of "databuf",
                         * and the number of bytes to send. */
                        /* bytessent is the number of bytes actually sent */

                readclock(fd_clock,nclockreads);        /* Read sat clock. */
                                                /* Put the time in next
                                                 * row of "timebuf". */
                overhdbuf[nclockreads] = 32;
                nclockreads =+ 1;

                if (bytessent[i] != bytes_block)
                {       printf("Write error, a(bort) or c(ontinue) ? ");
                        if (answer() == 0)
                                exit();         /* Abort = 0    */
                        else                    /* Continue = 1  */
                        {
                                readclock(fd_clock,nclockreads);
                                printf("\n\t\tattempting next write, ");
                                continue;
                                        /* A continue statement quits the existing
                                         * iteration of the for loop and starts
                                         * the next incremented loop value
                                         */
                        }
                }

                /* make the last end time be the next start time  */
                dupclocktime(nclockreads);
        }

        printf("Xmit complete, ");
        readclock(fd_clock,nclockreads);
        overhdbuf[nclockreads] = 45;
        nclockreads =+ 1;
        close (fd_net); /* Close connection to ntia-its */
```

```
        readclock(fd_clock,nclockreads);
        overhdbuf[nclockreads] = 11;
        nclockreads =+ 1;

        nice(10);       /* Set the UNIX nice routine to normal value */

        decodetime (2,&sattime);        /* Decode char buffer containing
                                         * time returned by sat clock and
                                         * convert it to integers.
                                         * 2 for location in buffer */

        for (j=1;j<numblocks+1;j++)     /* WRITE HISTORY FILE */

                /* write arguments = file descriptor, buffer, # bytes
                 * History file to be record number, number of bytes,
                 * and all start and end times. */

        {
                byteswritten = write (fd_history,&j,2);
                        /* Counter j written to history file as record no. */
                byteswritten = write (fd_history,&(bytessent[j-1]),2);
                        /* bytessent is an array built from the returned
                         * arguments of the write routines.
                         * It is the number of bytes actually sent for
                         * each block. */
                byteswritten = write (fd_history,&sattime,sizeof(sattime));
                        /* start time of each block. */
                decodetime (2*j+1,&sattime);

                byteswritten = write (fd_history,&sattime,sizeof(sattime));

        }

        printf("Transaction complete ");

        record_ovrhd(fd_overhead,nclockreads);

        transactnum =+ 1;       /* Increment the number of transactions and
                                 * do another  if the while condition is met. */
}

        time(tvec);     /* Read UNIX clock. */
                        /* tvec time is for close network. */

        closeclock(fd_clock);   /* Close sat clock file */

        if (debug == YES)
        {
                for(i=0;i<numblocks;i++)
                        printf ("\nbytessent[%d] =: %d",i,bytessent[i]);
        }

        close(fd_history);
        close(fd_overhead);

        baud = (totalbytes * 8) / (tvec[1]-ivec[1]);
```

```
        printf ("\n\n%d characters at %d baud ", totalbytes,baud);

        printf ("\ntest completed, %s\n----------------\n",ctime(tvec));
                /* ctime converts system time to ascii */

}
/*      PREFACE() routine:
*       Function to define and write preface record
*       for history file, ie file of before and after times
*       for transmission of each block.  returns file descriptor
*       of history file.
*
*       Dave Wortendyke                 July 3, 1980
*       modified        evi nemeth      february 1981
*               to read preface header record from a file.
*/
preface(file)

char    *file;
{
        int     fd;
        int     fd_preface;
        int     byteswritten;

        char    *p_ident;       /* Performance Measurement Identifier */
        char    *p_run;         /* Run Number           */
        char    *p_type;        /* Type Identifier source/dest */
        char    *p_info;        /* Information type identifier  */
        char    *p_source;      /* Source identifier            */
        char    *p_destin;      /* Destination identifier       */

        char    *readpreface();


        fd = creat(file,0664);
        if (fd < 0)
        {
                printf ("\ncannot create file\nprogram preface exiting\n");
                exit();
        }

        fd_preface = open("preface.x",READ);  /* READ is a 0 for UNIX read */
        if (fd_preface < 0)                   /* UNIX gives -1 for error */
        {
                printf ("\ncannot open preface file\nprogram preface exiting\n");
                exit();
        }

        byteswritten = 0;


        /* Following commands read pieces of preface
         * and write to history.x */
        /* The form A =+ B is equivelent to A = A + B.*/

        p_ident = readpreface(fd_preface,IDENTSIZE);
```

```
        byteswritten =+ write(fd, p_ident, IDENTSIZE);


        p_run   = readpreface(fd_preface,RUNSIZE);      /* batch no. */
        byteswritten =+ write(fd, p_run, RUNSIZE);

        p_type  = readpreface(fd_preface,TYPESIZE);
        byteswritten =+ write(fd, p_type, TYPESIZE);

        p_info  = readpreface(fd_preface,INFOSIZE);
        byteswritten =+ write(fd, p_info, INFOSIZE);

        p_source= readpreface(fd_preface,SOURCESIZE);
        byteswritten =+ write(fd, p_source, SOURCESIZE);

        p_destin= readpreface(fd_preface,DESTINSIZE);
        byteswritten =+ write(fd, p_destin, DESTINSIZE);

        byteswritten =+ write(fd, date, sizeof(date));

        if (debug == YES)
                printf("\npreface written to file history.x (%d bytes)",byteswritten);

        close(fd_preface);

        return(fd);
}


/*
*       READPREFCE () routine.
*       function to read the preface header information from the
*       file "preface.x" (preface.r).  the argument fd is its file
*       descriptor; length is the length of the string to be read.
*       the function returns a pointer to this character string.
*
*       if a line of the preface header file preface.x (preface.r)
*       is too long it is truncated, if it is too short it is padded
*       with blanks.  the proper sizes are defined constants at the
*       beginning of this file, actually they are in the include
*       file "clock.h".
*
*/


char *readpreface(fd,length)

int fd;
int length;

{
        register int i;
        register int j;
        int bytesread;
        int totalbytes;
```

```
        char s[MAXPREFACESIZE];

        if (length <= 0 || length > MAXPREFACESIZE)
        {
                printf ("\ninvalid length argument to function readpreface,");
                printf (" length:  %d",length);
                printf ("\nprogram preface exiting\n\n");
                exit();
        }

        totalbytes = 0;

        /* READ PREFACE LOOP FOLLOWS */

        for (i=0;i<MAXPREFACESIZE;i++)
        {
                bytesread = read(fd,&s[i],1);  /* read 1 byte, put in s[i] */
                if (bytesread == 0)
                {
                        printf ("\npreface.x file format wrong, eof encountered.");
                        printf ("\nprogram preface exiting\n\n");
                        exit();
                }

                totalbytes =+ bytesread;

                if (s[i] == NL)
                {
                        if (totalbytes < length)  /* blanks in rest of file */
                                for (j=totalbytes-1;j<length;j++)
                                        s[j] = BLANK;
                        s[length] = NULL;  /* last buffer char set = null */
                        return(s);      /* normal return */
                }
        }
        printf ("\npreface header record contains too many characters");
        printf ("\nno newline encountered, format probably wrong");
        printf ("\nprogram preface continuing\n\n");

        if (debug == YES)
                printf ("\npreface record:  %s",s);

        s[length] = NULL;
        return(s);

}
/*      READCLOCK () routine.
*       procedure to read time from satellite clock and store it
*       in a buffer, timebuf. there are two arguments: fd and n,
*       fd is the file descriptor of the clock line; n is the
*       row in the buffer to store the current clock reading.
*/

        /* GENERAL INFORMATION ON I/O:
         * First argument is the file descriptor.
         * Second argument is the buffer location.
```

```
         * Third argument is the number of bytes. */

        /* The return argument for a write is the number of
         * bytes sent, which must = the number supposedly sent
         * or there is an error. */

        /* The return argument for a read is a number greater
         * than zero, unless it was an end of file; or a
         * -1 for a read error. */

readclock(fd,n)

int fd;
int n;  /* n = offset in "timebuf" for particular clock reading. */

{
        int byteswritten;
        int bytesread;

        char *buffer;


        buffer = TRIGGER;    /* TRIGGER = "T" */
        byteswritten = write (fd,buffer,1);
        bytesread = read (fd,timebuf+n,CLOCKRECORDLENGTH);
        if (bytesread != CLOCKRECORDLENGTH)
                readerror(bytesread);

        /*The output from this routine is the sat clock time
         * in the specified row of "timebuf". */
}


/*
*       create psuedo-random data for test (10240 bytes)
*       unless the data already exists.
*       Dave Wortendyke                         July 3, 1980
*/

randomdata()
{
        register int     i;              /* loop counter         */
        int     fd_data;            /* file designator      */
        int byteswritten;

        int     temp;           /* temporary variable   */
        int     a;              /* n-1 register         */
        int     b;              /* n-2 register         */

        a = 0;          /* initial a value */
        b = 1;          /* initial b value */

        for (i=0;i<BUFFRSIZE;i++)      /* loop simulating circular shift */
        {
                temp = a;               /* save a */
                a = a ^ b;              /* a excl. or b */
```

```
                b = temp;               /* b = original a */

                temp = a;               /* save new a */
                a = a >> 3;             /* shift 3 to right */
                a = a & 01777;          /* a masked with octal 01777 */
                temp = temp << 13;      /* flush all but 3 bits */
                temp = temp >> 3;       /* align 3 bits */
                a = a | temp;           /* a incl. or temp */
                databuf[i] = (a & 0377); /* store lower 8 bits */
        }

        fd_data = open("data.x", 0);
        while(1)
        {
                if (fd_data > 0 )
                break;

                fd_data = creat("data.x", 0644);
                if (fd_data < 0)
                {
                        printf ("\ncould not create random data file\nprogram randomdata exiting\n");
                        exit();
                }
                byteswritten = write(fd_data, &databuf, sizeof(databuf));

                if (debug == YES)
                {
                printf ("\nrandom data now in databuf[%d]",byteswritten);
                }
                break;
        }

        close(fd_data);

}


/*
*       OPENCLOCK() routine.
*       dave wortendyke and evi nemeth
*       nov 1980
*
*       sets tty interface to proper mode for satellite clock.
        returns file descriptor of the clock line.
*       in order to open a port for the clock, the user must be
*       superuser, or the program must have mode 4755 and must be owned
*       by the root, or the port, that is the device in /dev, must
*       be owned by the owner of the program.
*/

int openclock()

{
```

```
        int fd;
        int clockline[3];       /* line characteristics for the clock */

        /* open clock port */

        fd = open (CLOCK,READWRITE);
        if (fd < 0)
        {
                printf ("\nunable to open tty port to read clock.\nexiting\n");
                exit();
        }

        /* set line characteristics */

        gtty (fd,ttyline);
        if (debug == YES)
                printmode(ttyline);

        clockline[0] = CLOCKSPEED;      /* speeds */
        clockline[1] = ttyline[1];      /* erase and kill characters */
        clockline[2] = CLOCKMODE;       /* any parity,hup,no echo,no cr/lf */

        stty (fd,clockline);
        if (debug == YES)
        {
                gtty (fd,clockline);
                printmode(clockline);
        }


        return(fd);

}


/*
*       INITIALIZCLOCK() routine.
*       resets, formats and initializes clock.
*/

initializeclock(fd)

int fd;                         /* file descriptor of the clock line */

{

        register int j;         /* loop counter */

        int byteswritten;       /* value returned by the write routine */
        int bytesread;          /* value returned by the read routine */


        int printmode();        /* function to print line characteristics */
```

```
        int timevec[2];         /* time vector returned by time system call */
        int *systime;           /* pointer to array returned by gmtime sys call */
        int timeoff;            /* difference between satelite and system clocks */
        int satday;             /* day returned by satellite clock */

        char *buffer;           /* buffer for clock writes */
        char temp[4];           /* temporary storage for char to int conversions */


/*      turn the clock on and format it for initial record,
*       which includes the date.
*/


        buffer = RESET;                         /* reset code for clock */
        byteswritten = write (fd,buffer,1);
        if (byteswritten != 1)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nbuffer: %s",buffer);

        buffer = IFORMAT;                       /* format code including day */
        byteswritten = write (fd,buffer,FORMATLENGTH);
        if (byteswritten != FORMATLENGTH)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nbuffer: %s",buffer);


/*
*       read an initial record on the satelite clock
*       read an initial record on the system clock
*/

        j = 0;
        while (1)
        {
                bytesread = read(fd,&(timebuf[0][j]),1);
                if (bytesread != 1)
                        readerror(bytesread);

                if (timebuf[0][j] == NL)
                        break;

                else
                        j++;

        }

        if (debug == YES)
                printf ("\ninitial satellite time:    %s",timebuf);

        time (timevec);
        systime = gmtime(timevec);
```

```
        if (debug == YES)
        {
                printf ("\nsystime -- %d %d %d:%d:%d\n",*(systime+YEAR),
                   *(systime+JULIANDAY) + 1,*(systime+HOURS),*(systime+MINUTES),
                   *(systime+SECONDS));
        }


/*
*       decode the satellite clock record and compare system clock and
*       satellite clock values.  exit if the two times are too far off.
*/

        temp[0] = timebuf[0][IDAYBEGIN];
        temp[1] = timebuf[0][IDAYBEGIN + 1];
        temp[2] = timebuf[0][IDAYBEGIN + 2];
        temp[3] = NULL;
        satday = atoi(temp);

        if (satday != 1 + *(systime + JULIANDAY))       /* system uses 0-364, satellite uses 1-365 */
        {
                printf ("\ncheck the date on your system\nprogram exiting\n");
                exit();
        }

        temp[0] = timebuf[0][IHOURBEGIN];
        temp[1] = timebuf[0][IHOURBEGIN + 1];
        temp[2] = NULL;
        sattime.hour = atoi(temp);

        temp[0] = timebuf[0][IMINBEGIN];
        temp[1] = timebuf[0][IMINBEGIN + 1];
        temp[2] = NULL;
        sattime.min = atoi(temp);

        timeoff = ( *(systime + HOURS) * 60 + *(systime + MINUTES) ) - ( sattime.hour * 60 + sattime.min);
        if (debug == YES)
                printf ("\ntimeoff =: %d minutes\n",timeoff);
        if (timeoff > TIMEDIFFLIMIT || timeoff < -TIMEDIFFLIMIT)
        {
                printf ("\ncheck the time on your system\nprogram exiting\n");
                exit();
        }

        date[0] = *(systime + YEAR);
        date[1] = (*(systime + MONTH)) + 1;      /* system call returns 0-11, we need 1-12 */
        date[2] = *(systime + DAY);


/*
*       reformat the clock to return only the time.  read the record
*       generated by formatting it.
*/

        buffer = FORMAT;                        /* format code, time only */
        byteswritten = write (fd,buffer,FORMATLENGTH);
```

```
        if (byteswritten != FORMATLENGTH)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nclock format: %s",buffer);

        j = 0;
        while (1)
        {
                bytesread = read(fd,&(timebuf[0][j]),1);
                if (bytesread != 1)
                        readerror(bytesread);

                if (timebuf[0][j] == NL)
                        break;

                else
                        j++;

        }

        if (debug == YES)
                printf ("\nreformatted clock record:    %s\n",timebuf);

}

/*      function to decode the character buffer containing the time
*       returned by the satellite clock and convert it to integers
*/


decodetime(n,sat)

int n;
struct time *sat;

{
        char hour[3];
        char min[3];
        char sec[3];
        char fractionalsec[4];


        hour[0] = timebuf[n][HOURBEGIN];
        hour[1] = timebuf[n][HOURBEGIN + 1];
        hour[2] = NULL;

        min[0] = timebuf[n][MINBEGIN];
        min[1] = timebuf[n][MINBEGIN + 1];
        min[2] = NULL;

        sec[0] = timebuf[n][SECBEGIN];
        sec[1] = timebuf[n][SECBEGIN + 1];
        sec[2] = NULL;

        fractionalsec[0] = timebuf[n][FRACBEGIN];
        fractionalsec[1] = timebuf[n][FRACBEGIN + 1];
```

```
        fractionalsec[2] = timebuf[n][FRACBEGIN + 2];
        fractionalsec[3] = NULL;

        sat->hour = atoi(hour);
        sat->min = atoi(min);
        sat->sec = atoi(sec);
        sat->frac = atoi(fractionalsec);


        if (debug == YES)
        {
                printf ("\nn =: %d",n);
                printf ("\ntimebuf =: %s",timebuf+n);
                printf ("\ncharacter arrays =:  %s:%s:%s.%s",hour,min,sec,fractionalsec);
                printf ("\ninteger arrays   =:  %d,%d,%d+%dms\n",sat->hour,sat->min,sat->sec,sat->frac);
        }

}


/*      RECORD_OVERHEAD() information on transaction states and times  */

record_overhd(fd,n)

int     fd;
int     n;      /* number of states and times to record to overhead file */

{
        int     j;
        int     bytes;

        for (j=1; j < n+1; j++)

        {
                decodetime(j-1,&sattime);
                bytes = write(fd, &j, 2);
                bytes = write(fd, &(overhdbuf[j-1]), 2);
                bytes = write(fd, &sattime, sizeof(sattime));
        }
}


printmode(line)

int line[3];

{
        printf ("\ngtty line mode: %o %o %o \n",line[0],line[1],line[2]);
        return(0);
}
```

```
writerror(bytes)

int bytes;
{
        printf ("\ndevice write error ");
        printf ("\nnumber of bytes written = %d",bytes);
        exit();
}



readerror(bytes)

int bytes;
{
        printf ("\nclock read error ");
        printf ("\nnumber of bytes read = %o \n",bytes);
        exit();
}



/*
*       reset ttyX line for Satellite Clock back to normal tty mode
*/

closeclock(fd)

int fd;

{

        stty (fd,ttyline);

        if (debug == YES)
        {
                gtty (fd,ttyline);
                printf ("\nin closeclock routine:");
                printmode(ttyline);
        }

        close(fd);

}

/*      ANSWER() routine.
 *      returns 1 if true or continue and 0 if false or abort.
 */

int answer()
{
        int a;
```

```
                switch(getchar())
                        {
                           case 'a':
                                printf("\n Test terminated by operator \n");
                                a = 0;
                                break;
                           case 'c':
                                a = 1;
                                break;
                           default:
                                printf(" 'a' or 'c'? ");
                        }
        return(a);
}

/*      DUPCLOCKTIME() routine
 *      duplicates a previous clock reading into the next row of the buffer
 */

dupclocktime(n)

int     n;
{

        register        int     k;

                /* make the last end time be the next start time  */

                for (k = 0; k < CLOCKRECORDLENGTH + 1; k++)
                {
                        timebuf[n][k] = timebuf[n-1][k];
                }
}
```

```
#
/*      main program to receive data from sender over arpanet
*       written:        David Wortendyke        september 30, 1980
*       modified:       evi nemeth              december 1980
*       modified:       evi nemeth              february 1981
*                       to read preface header from a file.
*       commented:      Dana Grubb              april 1981
        rewritten:      Dave Wortendyke         april-may 1981
                        Version 2.0 -- Overhead measurements
*       changed:        dave wortendyke         oct 27, 81
                        dropped warm-up for large blocks (>64 bytes)
*       corrected:      dave wortendyke         nov 12, 81
*                       insufficient # of writes to history file when blocks
*                       are split by net software into two smaller blocks
*

*       Inputs:  Program-  # of bytes/block expected, # transactions
                 Files- One: preface.r
*                Hardware- WWV Satellite Receiver clock
*                Hardware- Network controller/TIP
*                Library-
*       Outs:    Program-
                 Files- Three:  history.r, overhead.r, data.r
*                Library-printf
*/

#include "clock.usr.h"

/*      clock.usr.h --> BLOCKSIZE=512, NBLOCKS=20, NTRANSACTIONS=1    */
/*      clock.ovh.h --> BLOCKSIZE= 64, NBLOCKS= 1, NTRANSACTIONS=160   */

#include "net.h"    /* open type bit assignments */

#define recv main   /* calling "recv" gives the "main" program */

struct openparams               /* struct for making parameterized connections */
{
        char o_op;              /* opcode for kernel & daemon - unused here */
        char o_type;            /* type for connection see defines below */
        int  o_id;              /* id of file for kernel & daemon - unused here */
        int  o_lskt;            /* local socket number either abs or rel */
        int  o_fskt[2];         /* foreign skt either abs or rel */
        char o_frnhost;         /* for connection to specific host nums */
        char o_bsize;           /* bytesize of connection telnet demands 8 */
        int o_nomall;           /* initial allocation bits and msgs */
        int o_timeo;            /* num of secs to wait before timing out */
        int  o_relid;           /* fid of file to base a data connection on */
} openparams;

struct time             /* for satelite clocks time entries */
{
        int hour;       /* hours */
        int min;        /* minutes */
        int sec;        /* seconds */
        int frac;       /* fractional seconds in milliseconds */
} sattime;
```

```
char    databuf[BUFFRSIZE];     /* data buffer for transmission across the network */
char    timebuf[2*MAXBLOCKS+7][CLOCKRECORDLENGTH+1];    /* buffer for satellite
                        clock times, +1 needed for null and +7 for overhd times */

int     date[3];        /* date: year, month, day */
int     ttyline[3];     /* line characteristics of ttyX (from gtty) as a tty */
int     debug;          /* flag for debug-trace printouts       */
int     overhdbuf[2*MAXBLOCKS+7];       /* buffer to store transaction state changes */



recv(argc,argv)
                /* Arguments are for when program begins executing.
                 * argc = number of command line arguments, counting
                 *        0 for "xmit"and 1 for number of blocks.
                 * argv = pointer to array of character strings that
                 *        contain arguments (one per string). In this case,
                 *        argv[1] contains the block size in bytes and
                 *        argv[2] contains the number of network listens.
                 *        (The default = 20)
                 */

int argc;
char *argv[];
{

        register int i;
        int     j;
        int     byteswritten;
        int     bytesrecd[MAXBLOCKS];   /* number bytes actually recvd/block */
        int     numblocks;              /* number of blocks expected */
        int     netreads;               /* number actually read      */
        int     bytes_block;            /* number of bytes per block   */
        int     numaccesses;            /* number of network accesses  */
        int     transactnum;            /* acccess or transaction number */
        int     totalbytes;             /* total # bytes recvd  */
        int     subtotalbytes;          /* total # bytes recvd/transaction      */
        int     bytes_to_read;          /* number of bytes to read each time
                                         * from network = 3 * number expected
                                         * except for the fake test = exact #
                                         */
        int     fd_net;                 /* file descriptor, arpanet network */
        int     fd_history;             /* file descriptor, history file */
        int     fd_overhead;            /* file descriptor, overhead file */
        int     fd_clock;               /* file descriptor, satellite clock */
        int     fd_data;                /* file descriptor, recvd data file */
        int     ivec[2];                /* system time vector, before transmission */
        int     tvec[2];                /* system time vector, after transmission */
        char    *pbuf;                  /* pointer to random data buffer */
        char    *filename;              /* pointer to file name          */
```

```
int     baud;

int     time();                 /* time of day, system clock */
char    *ctime();               /* converts system time to ascii */
int     nclockreads;            /* number of satellite clock reads */


time(ivec);   /* Read UNIX system clock. */
printf ("\n---------- network reception ---------- ");
printf ("\ntest beginning,");
printf (" %s ", ctime(ivec));

bytes_block = (argc > 1) ? atoi(argv[1]) : BLOCKSIZE;

numaccesses = (argc > 2) ? atoi(argv[2]) : NTRANSACTIONS;

        /* This is a "conditional expression", meaning:
         * if the term before the question mark is not = 0,
         * then the term after the question is the value;
         * otherwise, the term after the colon is the value.
         * In other words: if argc >2, then the ascii form of
         * argv[2] is the value of "number of transactions"; otherwise,
         * set = NTRANSACTIONS (in the "clock.h" file), which = 1.
         */

numblocks = BUFFRSIZE / (bytes_block * numaccesses);
bytes_to_read = 3 * bytes_block;   /* Allow room for more than expected
                                    */


printf ("\nTest to access network with %d listens\n",numaccesses);
debug = NO;     /* Change this to YES if debug is desired */

pbuf = &databuf;        /* pbuf points to contents of databuf */

fd_clock = openclock(); /* set tty interface to proper mode for
                         * sat clock and return file descriptor
                         */

initializeclock(fd_clock);      /* reset sat clock, format sat clock,
                                 * read initial record from both
                                 * clocks, and compare clocks
                                 */

filename = "history.r";
fd_history = preface(filename);    /* Write preface record for history
                                    * file. The preface routine returns
                                    * a file descriptor for history file.
                                    */


filename = "overhead.r";
fd_overhead = preface(filename);    /* Write preface record for
                                     * overhead file
                                     */
```

```
        fd_data = creat("data.r", 0664);
        if (fd_data < 0)
        {
                printf("\nCannot create file data.r for recvd data");
                printf("\nProgram exiting \n");
                exit();
        }


        openparams.o_type = O_RCV;
        openparams.o_lskt   = 10;
        openparams.o_nomall = 1024;

        time(ivec);   /* Read UNIX system clock. */
                      /* ivec is the time for network open. */

        transactnum = 0;
        totalbytes = 0;

      start:
      while (transactnum < numaccesses)
      {
        nice(-85);      /* Have UNIX give us higher priority for service */

        printf("\nAttempting open # %d, ", transactnum + 1);


        readclock(fd_clock,0);  /* Read satellite clock. */
        overhdbuf[0] = 32;
        nclockreads = 1;

        fd_net = open("/dev/net/anyhost",&openparams);
        /* try to open connection to anyhost */

        readclock(fd_clock,1);  /* Read satellite clock. */

        if (fd_net <= 0)
        {
                overhdbuf[1] = 11;
                nclockreads = 2;
                record_ovrhd(fd_overhead,nclockreads);
                printf("Open denied, a(bort) or c(ontinue) ? ");
                if (answer() == 0)
                        exit();         /* Abort = 0 = False    */
                else
                        goto start;     /* Continue = 1 = True  */
        }
        else
        {
                overhdbuf[1] = 23;
                nclockreads = 2;
                printf("Opened, ");
        }

        subtotalbytes = 0;
        netreads = 0;
```

```
/* Now receive the data for real this time. */

readclock(fd_clock,2);  /* Read satellite clock.  */
                         /* The 2 means put the time at the
                         * 2nd row of the buffer used by readclock. */
/* DATA RECEPTION LOOP FOLLOWS */

for (i=0; i < numblocks + 4; i++)
{
        overhdbuf[nclockreads]= 32;
        nclockreads =+ 1;

        bytesrecd[i] = read(fd_net,pbuf+totalbytes+subtotalbytes,bytes_to_read);


        readclock(fd_clock,nclockreads);        /* Read sat clock. */
                                                /* Put the time in next
                                                 * row of "timebuf". */
        overhdbuf[nclockreads] = 23;
        nclockreads =+ 1;

        netreads =+ 1;                  /* increment # of blocks read */
        if (bytesrecd[i] > 0)
        {
                subtotalbytes =+ bytesrecd[i];
                /* make the last end time be the next start time  */
                dupclocktime(nclockreads);
        }
        else
        {
                if (bytesrecd[i] == 0)  /* EOF means xmit closed */
                {
                        overhdbuf[nclockreads - 1] = 45;
                        break;
                }
                else    /* bytesrecd[i] < 0 and error */
                {
                        printf("Read error, a(bort) or c(ontinue) ? ");
                        if (answer() == 0)
                                exit();         /* Abort = 0    */
                        else
                        {
                                                /* Continue = 1  */
                                printf("\n\t\tattempting next read, ");
                                readclock(fd_clock,nclockreads);
                        }
                }
        }

}

printf("Recv complete, ");
readclock(fd_clock,nclockreads);
overhdbuf[nclockreads] = 54;
nclockreads =+ 1;
```

```
        close (fd_net); /* Close connection to anyhost  */
        readclock(fd_clock,nclockreads);
        overhdbuf[nclockreads] = 11;
        nclockreads =+ 1;

        nice(10);       /* Set the UNIX nice routine to normal value */

        decodetime (2,&sattime);        /* Decode char buffer containing
                                         * time returned by sat clock and
                                         * convert it to integers.
                                         * 2 for location in buffer */

        for (j=1;j <= netreads;j++)     /* WRITE HISTORY FILE */

                /* write arguments = file descriptor, buffer, # bytes
                 * History file to be record number, number of bytes,
                 * and all start and end times. */

        {
                byteswritten = write (fd_history,&j,2);
                        /* Counter j written to history file as record no. */
                byteswritten = write (fd_history,&(bytesrecd[j-1]),2);
                        /* bytesrecd is an array built from the returned
                         * arguments of the read routines.
                         * It is the number of bytes actually sent for
                         * each block. */
                byteswritten = write (fd_history,&sattime,sizeof(sattime));
                        /* start time of each block. */
                decodetime (2*j+1,&sattime);

                byteswritten = write (fd_history,&sattime,sizeof(sattime));

        }
        byteswritten = write(fd_data, pbuf + totalbytes, subtotalbytes);

        printf("Transaction complete ");

        record_ovrhd(fd_overhead,nclockreads);

        totalbytes =+ subtotalbytes;  /* Update the grand total count */
        transactnum =+ 1;       /* Increment the number of transactions and
                                 * do another  if the while condition is met. */
}


        time(tvec);     /* Read UNIX clock. */
                        /* tvec time is for close network. */

        closeclock(fd_clock);   /* Close sat clock file */

        if (debug == YES)
        {
                for(i=0;i<numblocks;i++)
                        printf ("\nbytesrecd[%d] =: %d",i,bytesrecd[i]);
        }
```

```
        close(fd_history);
        close(fd_overhead);
        close(fd_data);

        baud = (totalbytes * 8) / (tvec[1]-ivec[1]);
        printf ("\n\n%d characters at %d baud ", totalbytes,baud);

        printf ("\ntest completed, %s\n----------------\n",ctime(tvec));
                /* ctime converts system time to ascii */

}
/**/
/*      PREFACE() routine:
*       Function to define and write preface record
*       for history file, ie file of before and after times
*       for transmission of each block.  returns file descriptor
*       of history file.
*
*       Dave Wortendyke                 July 3, 1980
*       modified        evi nemeth      february 1981
*               to read preface header record from a file.
*/
preface(file)

char    *file;
{
        int     fd;
        int     fd_preface;
        int     byteswritten;

        char    *p_ident;       /* Performance Measurement Identifier */
        char    *p_run;         /* Run Number                         */
        char    *p_type;        /* Type Identifier source/dest */
        char    *p_info;        /* Information type identifier  */
        char    *p_source;      /* Source identifier            */
        char    *p_destin;      /* Destination identifier       */

        char    *readpreface();


        fd = creat(file,0664);
        if (fd < 0)
        {
                printf ("\ncannot create file\nprogram preface exiting\n");
                exit();
        }

        fd_preface = open("preface.r",READ);  /* READ is a 0 for UNIX read */
        if (fd_preface < 0)                   /* UNIX gives -1 for error */
        {
                printf ("\ncannot open preface file\nprogram preface exiting\n");
                exit();
        }

        byteswritten = 0;
```

```
        /* Following commands read pieces of preface
         * and write to history.r */
        /* The form A =+ B is equivelent to A = A + B.*/

        p_ident = readpreface(fd_preface,IDENTSIZE);

        byteswritten =+ write(fd, p_ident, IDENTSIZE);


        p_run   = readpreface(fd_preface,RUNSIZE);      /* batch no. */
        byteswritten =+ write(fd, p_run, RUNSIZE);

        p_type  = readpreface(fd_preface,TYPESIZE);
        byteswritten =+ write(fd, p_type, TYPESIZE);

        p_info  = readpreface(fd_preface,INFOSIZE);
        byteswritten =+ write(fd, p_info, INFOSIZE);

        p_source= readpreface(fd_preface,SOURCESIZE);
        byteswritten =+ write(fd, p_source, SOURCESIZE);

        p_destin= readpreface(fd_preface,DESTINSIZE);
        byteswritten =+ write(fd, p_destin, DESTINSIZE);

        byteswritten =+ write(fd, date, sizeof(date));

        if (debug == YES)
                printf("\npreface written to file history.r (%d bytes)",byteswritten);

        close(fd_preface);

        return(fd);
}


/*
*       READPREFCE () routine.
*       function to read the preface header information from the
*       file "preface.x" (preface.r).  the argument fd is its file
*       descriptor; length is the length of the string to be read.
*       the function returns a pointer to this character string.
*
*       if a line of the preface header file preface.x (preface.r)
*       is too long it is truncated, if it is too short it is padded
*       with blanks.  the proper sizes are defined constants at the
*       beginning of this file, actually they are in the include
*       file "clock.h".
*
*/


char *readpreface(fd,length)

int fd;
int length;
```

```
{
        register int i;
        register int j;
        int bytesread;
        int subtotalbytes;
        char s[MAXPREFACESIZE];

        if (length <= 0 || length > MAXPREFACESIZE)
        {
                printf ("\ninvalid length argument to function readpreface,");
                printf (" length:  %d",length);
                printf ("\nprogram preface exiting\n\n");
                exit();
        }

        subtotalbytes = 0;

        /* READ PREFACE LOOP FOLLOWS */

        for (i=0;i<MAXPREFACESIZE;i++)
        {
                bytesread = read(fd,&s[i],1);  /* read 1 byte, put in s[i] */
                if (bytesread == 0)
                {
                        printf ("\npreface.r file format wrong, eof encountered.");
                        printf ("\nprogram preface exiting\n\n");
                        exit();
                }

                subtotalbytes =+ bytesread;

                if (s[i] == NL)
                {
                        if (subtotalbytes < length)  /* blanks in rest of file */
                                for (j=subtotalbytes-1;j<length;j++)
                                        s[j] = BLANK;
                        s[length] = NULL;  /* last buffer char set = null */
                        return(s);      /* normal return */
                }
        }
        printf ("\npreface header record contains too many characters");
        printf ("\nno newline encountered, format probably wrong");
        printf ("\nprogram preface continuing\n\n");

        if (debug == YES)
                printf ("\npreface record:  %s",s);

        s[length] = NULL;
        return(s);

}
/**/
/*      READCLOCK () routine.
*       procedure to read time from satellite clock and store it
*       in a buffer, timebuf. there are two arguments: fd and n,
```

```
*       fd is the file descriptor of the clock line; n is the
*       row in the buffer to store the current clock reading.
*/

        /* GENERAL INFORMATION ON I/O:
         * First argument is the file descriptor.
         * Second argument is the buffer location.
         * Third argument is the number of bytes. */

        /* The return argument for a write is the number of
         * bytes sent, which must = the number supposedly sent
         * or there is an error. */

        /* The return argument for a read is a number greater
         * than zero, unless it was an end of file; or a
         * -1 for a read error. */

readclock(fd,n)

int fd;
int n;  /* n = offset in "timebuf" for particular clock reading. */

{
        int byteswritten;
        int bytesread;

        char *buffer;


        buffer = TRIGGER;     /* TRIGGER = "T" */
        byteswritten = write (fd,buffer,1);
        bytesread = read (fd,timebuf+n,CLOCKRECORDLENGTH);
        if (bytesread != CLOCKRECORDLENGTH)
                readerror(bytesread);

        /*The output from this routine is the sat clock time
         * in the specified row of "timebuf". */
}

/**/

/*
*       OPENCLOCK() routine.
*       dave wortendyke and evi nemeth
*       nov 1980
*
*       sets tty interface to proper mode for satellite clock.
        returns file descriptor of the clock line.
*       in order to open a port for the clock, the user must be
*       superuser, or the program must have mode 4755 and must be owned
*       by the root, or the port, that is the device in /dev, must
*       be owned by the owner of the program.
*/


int openclock()
```

```
{

        int fd;
        int clockline[3];       /* line characteristics for the clock */

        /* open clock port */

        fd = open (CLOCK,READWRITE);
        if (fd < 0)
        {
                printf ("\nunable to open tty port to read clock.\nexiting\n");
                exit();
        }

        /* set line characteristics */

        gtty (fd,ttyline);
        if (debug == YES)
                printmode(ttyline);

        clockline[0] = CLOCKSPEED;      /* speeds */
        clockline[1] = ttyline[1];      /* erase and kill characters */
        clockline[2] = CLOCKMODE;       /* any parity,hup,no echo,no cr/lf */

        stty (fd,clockline);
        if (debug == YES)
        {
                gtty (fd,clockline);
                printmode(clockline);
        }


        return(fd);

}

/**/

/*
*       INITIALIZCLOCK() routine.
*       resets, formats and initializes clock.
*/

initializeclock(fd)

int fd;                         /* file descriptor of the clock line */

{

        register int j;         /* loop counter */

        int byteswritten;       /* value returned by the write routine */
        int bytesread;          /* value returned by the read routine */
```

```
        int printmode();        /* function to print line characteristics */


        int timevec[2];         /* time vector returned by time system call */
        int *systime;           /* pointer to array returned by gmtime sys call */
        int timeoff;            /* difference between satelite and system clocks */
        int satday;             /* day returned by satellite clock */

        char *buffer;           /* buffer for clock writes */
        char temp[4];           /* temporary storage for char to int conversions */


/*      turn the clock on and format it for initial record,
*       which includes the date.
*/


        buffer = RESET;                         /* reset code for clock */
        byteswritten = write (fd,buffer,1);
        if (byteswritten != 1)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nbuffer: %s",buffer);

        buffer = IFORMAT;                       /* format code including day */
        byteswritten = write (fd,buffer,FORMATLENGTH);
        if (byteswritten != FORMATLENGTH)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nbuffer: %s",buffer);


/*
*       read an initial record on the satelite clock
*       read an initial record on the system clock
*/

        j = 0;
        while (1)
        {
                bytesread = read(fd,&(timebuf[0][j]),1);
                if (bytesread != 1)
                        readerror(bytesread);

                if (timebuf[0][j] == NL)
                        break;

                else
                        j++;

        }

        if (debug == YES)
                printf ("\ninitial satellite time:    %s",timebuf);
```

```
        time (timevec);
        systime = gmtime(timevec);

        if (debug == YES)
        {
                printf ("\nsystime -- %d %d %d:%d:%d\n",*(systime+YEAR),
                   *(systime+JULIANDAY) + 1,*(systime+HOURS),*(systime+MINUTES),
                   *(systime+SECONDS));
        }


/*
*       decode the satellite clock record and compare system clock and
*       satellite clock values.  exit if the two times are too far off.
*/

        temp[0] = timebuf[0][IDAYBEGIN];
        temp[1] = timebuf[0][IDAYBEGIN + 1];
        temp[2] = timebuf[0][IDAYBEGIN + 2];
        temp[3] = NULL;
        satday = atoi(temp);

        if (satday != 1 + *(systime + JULIANDAY))       /* system uses 0-364, satellite uses 1-365 */
        {
                printf ("\ncheck the date on your system\nprogram exiting\n");
                exit();
        }

        temp[0] = timebuf[0][IHOURBEGIN];
        temp[1] = timebuf[0][IHOURBEGIN + 1];
        temp[2] = NULL;
        sattime.hour = atoi(temp);

        temp[0] = timebuf[0][IMINBEGIN];
        temp[1] = timebuf[0][IMINBEGIN + 1];
        temp[2] = NULL;
        sattime.min = atoi(temp);

        timeoff = ( *(systime + HOURS) * 60 + *(systime + MINUTES) ) - ( sattime.hour * 60 + sattime.min);
        if (debug == YES)
                printf ("\ntimeoff =: %d minutes\n",timeoff);
        if (timeoff > TIMEDIFFLIMIT || timeoff < -TIMEDIFFLIMIT)
        {
                printf ("\ncheck the time on your system\nprogram exiting\n");
                exit();
        }

        date[0] = *(systime + YEAR);
        date[1] = (*(systime + MONTH)) + 1;      /* system call returns 0-11, we need 1-12 */
        date[2] = *(systime + DAY);


/*
*       reformat the clock to return only the time.  read the record
*       generated by formatting it.
*/
```

```
        buffer = FORMAT;                        /* format code, time only */
        byteswritten = write (fd,buffer,FORMATLENGTH);
        if (byteswritten != FORMATLENGTH)
                writerror(byteswritten);
        if (debug == YES)
                printf("\nclock format: %s",buffer);

        j = 0;
        while (1)
        {
                bytesread = read(fd,&(timebuf[0][j]),1);
                if (bytesread != 1)
                        readerror(bytesread);

                if (timebuf[0][j] == NL)
                        break;

                else
                        j++;

        }

        if (debug == YES)
                printf ("\nreformatted clock record:    %s\n",timebuf);
}

/**/
/*      function to decode the character buffer containing the time
*       returned by the satellite clock and convert it to integers
*/


decodetime(n,sat)

int n;
struct time *sat;

{
        char hour[3];
        char min[3];
        char sec[3];
        char fractionalsec[4];


        hour[0] = timebuf[n][HOURBEGIN];
        hour[1] = timebuf[n][HOURBEGIN + 1];
        hour[2] = NULL;

        min[0] = timebuf[n][MINBEGIN];
        min[1] = timebuf[n][MINBEGIN + 1];
        min[2] = NULL;

        sec[0] = timebuf[n][SECBEGIN];
        sec[1] = timebuf[n][SECBEGIN + 1];
```

```
        sec[2] = NULL;

        fractionalsec[0] = timebuf[n][FRACBEGIN];
        fractionalsec[1] = timebuf[n][FRACBEGIN + 1];
        fractionalsec[2] = timebuf[n][FRACBEGIN + 2];
        fractionalsec[3] = NULL;

        sat->hour = atoi(hour);
        sat->min = atoi(min);
        sat->sec = atoi(sec);
        sat->frac = atoi(fractionalsec);


        if (debug == YES)
        {
                printf ("\nn =: %d",n);
                printf ("\ntimebuf =: %s",timebuf+n);
                printf ("\ncharacter arrays =:  %s:%s:%s.%s",hour,min,sec,fractionalsec);
                printf ("\ninteger arrays   =:  %d,%d,%d+%dms\n",sat->hour,sat->min,sat->sec,sat->frac);
        }

}

/**/


/*      RECORD_OVERHEAD() information on transaction states and times  */


record_overhd(fd,n)

int     fd;
int     n;      /* number of states and times to record to overhead file */

{
        int     j;
        int     bytes;

        for (j=1; j < n+1; j++)

        {
                decodetime(j-1,&sattime);
                bytes = write(fd, &j, 2);
                bytes = write(fd, &(overhdbuf[j-1]), 2);
                bytes = write(fd, &sattime, sizeof(sattime));
        }
}

/**/


printmode(line)

int line[3];

{
```

```
        printf ("\ngtty line mode: %o %o %o \n",line[0],line[1],line[2]);
        return(0);
}



writerror(bytes)

int bytes;
{
        printf ("\ndevice write error ");
        printf ("\nnumber of bytes written = %d",bytes);
        exit();
}



readerror(bytes)

int bytes;
{
        printf ("\nclock read error ");
        printf ("\nnumber of bytes read = %o \n",bytes);
        exit();
}


/**/


/*
*       reset ttyX line for Satellite Clock back to normal tty mode
*/

closeclock(fd)

int fd;

{

        stty (fd,ttyline);

        if (debug == YES)
        {
                gtty (fd,ttyline);
                printf ("\nin closeclock routine:");
                printmode(ttyline);
        }

        close(fd);

}
/**/
```

```
/*      ANSWER() routine.
 *      returns 1 if true or continue and 0 if false or abort.
 */

int answer()
{
        int a;

                switch(getchar())
                        {
                           case 'a':
                                printf("\n Test terminated by operator \n");
                                a = 0;
                                break;
                           case 'c':
                                a = 1;
                                break;
                           default:
                                printf(" 'a' or 'c'? ");
                        }
        return(a);
}

/*      DUPCLOCKTIME() routine
 *      duplicates a previous clock reading into the next row of the buffer
 */

dupclocktime(n)

int     n;
{

        register        int     k;

                /* make the last end time be the next start time  */

                for (k = 0; k < CLOCKRECORDLENGTH + 1; k++)
                {
                        timebuf[n][k] = timebuf[n-1][k];
                }
}
```

```
/*      defined constants for programs testing network transmission */

#define OPNPARAMSIZE    18

/* open type bits */
#define O_DIRECT        01      /* icp ¦ direct */
#define O_SERVER        02      /* user ¦ server */
#define O_INIT          04      /* listen ¦ init */
#define O_SPECIFIC      010     /* general ¦ specific ( for listen ) */
#define O_DUPLEX        020     /* simplex ¦ duplex */
#define O_RELATIVE      040     /* absolute ¦ relative */
#define O_SEND          05      /* direct user init general smplex absolute */

#define O_RCV           01      /* direct user listen general smplex absolute */
```

```
/*      defined constants for programs using the satellite clock */


#define BLOCKSIZE       512
#define NBLOCKS         20
#define MAXBLOCKS       30
#define NTRANSACTIONS   1
#define BUFFRSIZE       10240

#define MAXPREFACESIZE  80
#define IDENTSIZE       32
#define RUNSIZE         4
#define TYPESIZE        12
#define INFOSIZE        8
#define SOURCESIZE      32
#define DESTINSIZE      32

#define NO              0
#define YES             1
#define READ            0
#define READWRITE       2
#define ZILLION         200

#define NL              '\n'
#define NULL            '\0'
#define BLANK           ' '
#define EOF             -1

#define CLOCK           "/dev/tty3"
#define TTYMODE         0532            /* any parity,echo,cr/lf,tabs */
#define CLOCKMODE       0501            /* any parity,hup */
#define CLOCKSPEED      05413           /* set in/out speeds at 11 = 2400 baud */

#define FORMATLENGTH        18          /* to format the clocks output */
#define IFORMAT         "Fddd hh:mm:ss.sssT"    /* format for initial clock record, includes day */
#define FORMAT          "FXXXXnnXmmXssXsssT"    /* format for other clock records, time only */
#define RESET           "R"
#define TRIGGER         "T"

#define MAXCLOCKRECORDLENGTH    20      /* to read initial clock record */
#define CLOCKRECORDLENGTH       13      /* to read other clock records */

#define YEAR            5               /* offset in time array returned by gmtime */
#define MONTH           4               /* offset in time array returned by gmtime */
#define JULIANDAY       7               /* offset in time array returned by gmtime */
#define DAY             3               /* offset in time array returned by gmtime */
#define HOURS           2               /* offset in time array returned by gmtime */
#define MINUTES         1               /* offset in time array returned by gmtime */
#define SECONDS         0               /* offset in time array returned by gmtime */

#define TIMEDIFFLIMIT   30              /* maximum difference allowed between
                                           system and satellite clocks (minutes) */

#define IDAYBEGIN       1               /* position in string returned by clock */
#define IHOURBEGIN      5               /* position in string returned by clock */
#define HOURBEGIN       1               /* position in string returned by clock, short format */
```

```
#define IMINBEGIN       8               /* position in string returned by clock */
#define MINBEGIN        3               /* position in string returned by clock, short format */
#define ISECBEGIN       11              /* position in string returned by clock */
#define SECBEGIN        5               /* position in string returned by clock, short format */
#define IFRACBEGIN      14              /* position in string returned by clock */
#define FRACBEGIN       7               /* position in string returned by clock, short format */
```

```
/*      defined constants for programs using the satellite clock */

#define BLOCKSIZE       64
#define NBLOCKS         1
#define MAXBLOCKS       10
#define NTRANSACTIONS   160
#define BUFFRSIZE       10240

#define MAXPREFACESIZE  80
#define IDENTSIZE       32
#define RUNSIZE         4
#define TYPESIZE        12
#define INFOSIZE        8
#define SOURCESIZE      32
#define DESTINSIZE      32

#define NO              0
#define YES             1
#define READ            0
#define READWRITE       2
#define ZILLION         200

#define NL              '\n'
#define NULL            '\0'
#define BLANK           ' '
#define EOF             -1

#define CLOCK           "/dev/tty3"
#define TTYMODE         0332            /* any parity,echo,cr/lf,tabs */
#define CLOCKMODE       0301            /* any parity,hup */
#define CLOCKSPEED      05413           /* set in/out speeds at 11 = 2400 baud */

#define FORMATLENGTH        18          /* to format the clocks output */
#define IFORMAT         "Fddd hh:mm:ss.sssT"    /* format for initial clock record, includes day */
#define FORMAT          "FXXXXhhXmmXssXsssT"    /* format for other clock records, time only */
#define RESET           "R"
#define TRIGGER         "T"

#define MAXCLOCKRECORDLENGTH    20      /* to read initial clock record */
#define CLOCKRECORDLENGTH       13      /* to read other clock records */

#define YEAR            5               /* offset in time array returned by gmtime */
#define MONTH           4               /* offset in time array returned by gmtime */
#define JULIANDAY       7               /* offset in time array returned by gmtime */
#define DAY             3               /* offset in time array returned by gmtime */
#define HOURS           2               /* offset in time array returned by gmtime */
#define MINUTES         1               /* offset in time array returned by gmtime */
#define SECONDS         0               /* offset in time array returned by gmtime */

#define TIMEDIFFLIMIT   30              /* maximum difference allowed between
                                           system and satellite clocks (minutes) */

#define IDAYBEGIN       1               /* position in string returned by clock */
#define IHOURBEGIN      5               /* position in string returned by clock */
#define HOURBEGIN       1               /* position in string returned by clock, short format */
```

```
#define IMINBEGIN       8               /* position in string returned by clock */
#define MINBEGIN        3               /* position in string returned by clock, short format */
#define ISECBEGIN       11              /* position in string returned by clock */
#define SECBEGIN        5               /* position in string returned by clock, short format */
#define IFRACBEGIN      14              /* position in string returned by clock */
#define FRACBEGIN       7               /* position in string returned by clock, short format */
```

APPENDIX C: SAMPLE DATA FILES

This appendix contains a sample and brief description of each of the various types of the data files mentioned in Section 4.1.3 and the bubble chart in Figure 26. The data presented should be considered representative only; in the case of long files, only the beginning of the file is shown. When the transmitted and received files have the same format, only one is included. Since a binary list of integer and floating point numbers would be meaningless, the binary files have been converted to ASCII characters and formatted for presentation. The files summarized below are presented as a complete set following this description.

1. PREFACE.R and RREFACE.X (Figure C-1) are text files for on-line input to the XMIT and RECV programs.

2. DATA.X (Figure C-2) is a binary file containing pseudorandom data generated on-line by XMIT. A small portion is shown as a hexidecimal dump using the UNIX "od -h" command.

3. HISTORY.X (Figure C-3) is a binary file generated on-line by the XMIT program and contains the block size and start and end times for each block transfer. The example shown is the text version of the file (H.INFO.X) generated by the program SHOW.

4. OVERHEAD.X (Figure C-4) is a binary file generated on-line by the XMIT program and contains transaction state codes and event times. The example shown is the text version of the file (O.INFO.X) generated by the program OVHD.SHOW.

5. HISTORY.X (Figure C-5), now located in the data directory, is a binary file consisting of the time-corrected version of the raw data file of Figure C-3. The text was obtained from the SHOW program.

6. OVERHEAD.X (Figure C-6), now located in the data directory, is a binary file consisting of the time-corrected version of the raw data file of Figure C-4. The text was obtained from the OVHD.SHOW program.

7. FORT14 (Figure C-7) is an ASCII character file reformatted from the OVERHEAD file of Figure C-6. This file is one of four used by the standard FORTRAN performance assessment program.

8. FORT17 (Figure C-8) is an ASCII character file created by merging the HISTORY.X file of Figure C-5 with the DATA.X file of Figure C-2. This file and its receive companion (FORT18) are used to transfer data from the binary files to the standard FORTRAN performance assessment programs.

Access/Disengagement Tests

9. ACCESS.INFO (Figure C-9) is a text file with information in the same format as 0.INFO.X, plus the following data: time differences (in seconds) between adjacent and alternate events, the file (with dtg preface) from whence the data originated, and an access attempt number. This file is created by the ACCESTIME program from data in the time-corrected OVERHEAD.X file.

10. TFIL (FIGURE C-10) is a temporary text file obtained by editing out all lines from ACCESS.INFO except for those in the third record. This results in a file which contains chronologically arranged access times in the fifth column of data.

11. TABLE.ACC (Figure C-11) is a text file produced by sorting the chronological access times in TFIL into ascending access times. This file is archived with the raw data for the access/disengagement tests.

12. PLOT16ACC (Figure C-12) is a text file containing x-y coordinates for the plottable access time histogram. It is created by the OVH.HISTO program from the TABLE.ACC file.

User Information Transfer Tests

13. XFER.INFO (Figure C-13) is a text file which contains, for each user information block transferred during a test, the source start time, the destination end time, and the time difference derived from the time-corrected HISTORY.X and HISTORY.R files. In addition, the date-prefixed file name and a received block number appears to the right of the times. At the end of the listing are calculations based on both 1043 definitions and X3.102 definitions of rates and transmission times.

14. TFILE (Figure C-14) is a temporary text file obtained by editing out all lines of the XFER.INFO file except those that have a block transfer time.

15. TABLE.XFR (Figure C-15) is a text file produced by sorting the chronological transfer times in TFIL into ascending transfer times. This file is archived with the raw data for the user information transfer tests.

16. PLOT10ACC (Figure C-16) is a text file containing x-y coordinates for the plottable block transfer time histogram. It is created by the HISTO program from the TABLE.XFR file.

a. Preface file preface.r

```
NBS/ITS ARPANET TEST
 126
DESTINATION
USER
NBS-UNIX (Gaithersburg)
NTIA-ITS (Boulder)
```

b. Preface file preface.x

```
NBS/ITS ARPANET TEST
 126
SOURCE
USER
NTIA-ITS (Boulder)
NBS-UNIX (Gaithersburg)
```

Figure C-1. Listing of PREFACE.R and PREFACE.X files.

```
0000000 8000 0290 0212 0422 0424 0c64 0c6d 108c
0000020 1093 38d0 38fd 50b8 51dd d0d1 d240 00d2
0000040 073a a007 aaf4 40ab 5b9d c058 e2d3 00e6
0000060 739c 817d 239f 0237 a006 0694 0632 0846
0000100 4c69 1da4 95b7 2ac4 b2bd 6ec1 4f15 880b
0000120 fb50 cd95 688b 8a3c 27b6 8e52 9e7b 08dc
0000140 729a 0d1d d9c2 0b83 57b1 0d3c 1de6 0c7f
0000160 94ae 0287 1290 1cf0 0c7d 3cee 3cda 403c
0000200 614f f8a5 ba2b 7132 b7c8 132f 1ac7 c43b
0000220 5b1f ae28 83b0 d426 b15e f51d 65dd 4277
0000240 a806 6fd5 9bb7 5b45 6763 6820 f8a9 66ca
0000260 3e95 1dd5 8c79 f63e 65d9 d6d7 d240 40d2
0000300 6e32 2c0b 79e4 d9b3 2ead eb70 af93 6587
0000320 037c 1def 58de f1d0 b644 d97e dd34 505d
0000340 d6e1 1246 17aa 8437 8296 2ce2 2a79 50aa
0000360 51df f891 f64d 5077 4e24 50cd 70d3 0074
0000400 7dee a172 1b7a 420c cd29 c6fc aca7 09e1
0000420 c21d 9f3b dd74 2db5 3fd3 641d c54f 9251
0000440 f538 ed99 60ee fe91 2aad 2710 9486 b202
0000460 7c56 2b25 d1a1 334e 5bef 31d6 159c 0431
0000500 9ce6 6acf 13f4 ddfc 1f24 6f67 19a1 6537
0000520 0d0a 15a0 2856 e02f 4bb9 ea1e c6de 1463
0000540 1bce fcfa bb20 d133 41dc 5a93 f4f9 1641
0000560 6b4a 99a4 3f67 1ecb a85a 0ede 2e7a 202a
0000600 0c41 5ce9 4576 f886 926f 49df af32 63b3
0000620 7bba 54d8 a811 6fd7 a757 771e 1f8d 3192
0000640 70b4 8d18 df72 7915 5e8d e81a 02fe 231f
0000660 b8e7 960b 7433 6b28 98a8 fb46 d937 20dd
0000700 833f b637 b410 2cd4 6e5f 35c6 b59e 3225
0000720 b682 0ec6 0639 78c7 6177 ec02 ce9d 288a
```

Figure C-2. Hexidecimal dump of DATA.X file.

```
History-information file:

Perfor. measur. ID       = NBS/ITS ARPANET TEST
Run number               =  126
Type                     = SOURCE
Information ID           = USER
Source                   = NBS-UNIX (Gaithersburg)
Destination              = NTIA-ITS (Boulder)
Mo/Day/Yr                = 11/24/81
Start time (Hr:Min:Sec) = 20:54:37

  Data from file    nov/24-2054his.x

**File prefix (dd-hhmm) = 24-2054

        Record Bytes Start time     End time

          1   512  20:54:37.248   20:54:40.977
          2   512  20:54:40.977   20:54:41.797
          3   512  20:54:41.797   20:54:42.730
          4   512  20:54:42.730   20:54:43.383
          5   512  20:54:43.383   20:54:44.069
          6   512  20:54:44.069   20:54:44.746
          7   512  20:54:44.746   20:54:45.463
          8   512  20:54:45.463   20:54:46.083
          9   512  20:54:46.083   20:54:46.869
         10   512  20:54:46.869   20:54:47.503
         11   512  20:54:47.503   20:54:48.120
         12   512  20:54:48.120   20:54:48.773
         13   512  20:54:48.773   20:54:49.397
         14   512  20:54:49.397   20:54:49.994
         15   512  20:54:49.994   20:54:50.670
         16   512  20:54:50.670   20:54:51.310
         17   512  20:54:51.310   20:54:51.928
         18   512  20:54:51.928   20:54:52.541
         19   512  20:54:52.541   20:54:53.251
         20   512  20:54:53.251   20:54:53.874
Total sets of times(blocks) = 20
```

Figure C-3. Text version (H.INFO.X) of HISTORY.X file.

```
Overhead-information files:

Perfor. measur. ID        = NBS/ITS ARPANET TEST
Run number                =  126
Type                      = SOURCE
Information ID            = USER
Source                    = NBS-UNIX (Gaithersburg)
Destination               = NTIA-ITS (Boulder)
Mo/Day/Yr                 = 11/24/81
Start time (Hr:Min:Sec)   = 20:54:35

  Data from file    nov/24-2054ovh.x

        Record Code  Clock time

           1    23  20:54:35.026
           2    32  20:54:37.145
           3    23  20:54:37.248
           4    32  20:54:40.977
           5    23  20:54:40.977
           6    32  20:54:41.797
           7    23  20:54:41.797
           8    32  20:54:42.730
           9    23  20:54:42.730
          10    32  20:54:43.383
          11    23  20:54:43.383
          12    32  20:54:44.069
          13    23  20:54:44.069
          14    32  20:54:44.746
          15    23  20:54:44.746
          16    32  20:54:45.463
          17    23  20:54:45.463
          18    32  20:54:46.083
          19    23  20:54:46.083
          20    32  20:54:46.869
          21    23  20:54:46.869
          22    32  20:54:47.503
                ~~~~~~~~~~~~~~~~
                ~~~~~~~~~~~~~~~~
          36    32  20:54:51.928
          37    23  20:54:51.928
          38    32  20:54:52.541
          39    23  20:54:52.541
          40    32  20:54:53.251
          41    23  20:54:53.251
          42    32  20:54:53.874
          43    45  20:54:53.992
          44    11  20:54:54.078
Total # times = 44
```

Figure C-4. Text version (O.INFO.X) of OVERHEAD.X file.

```
History-information files:

Perfor. measur. ID       = NBS/ITS ARPANET TEST
Run number               =  126
Type                     = SOURCE
Information ID           = USER
Source                   = NBS-UNIX (Gaithersburg)
Destination              = NTIA-ITS (Boulder)
Mo/Day/Yr                = 11/24/81
Start time (Hr:Min:Sec) = 20:54:37

  Data from file    history.x

**File prefix (dd-hhmm) = 24-2054
```

Record	Bytes	Start time	End time
1	512	20:54:37.313	20:54:40.970
2	512	20:54:41.042	20:54:41.790
3	512	20:54:41.862	20:54:42.723
4	512	20:54:42.795	20:54:43.376
5	512	20:54:43.448	20:54:44.062
6	512	20:54:44.134	20:54:44.739
7	512	20:54:44.811	20:54:45.456
8	512	20:54:45.528	20:54:46.076
9	512	20:54:46.148	20:54:46.862
10	512	20:54:46.934	20:54:47.496
11	512	20:54:47.568	20:54:48.113
12	512	20:54:48.185	20:54:48.766
13	512	20:54:48.838	20:54:49.390
14	512	20:54:49.462	20:54:49.987
15	512	20:54:50.059	20:54:50.663
16	512	20:54:50.735	20:54:51.303
17	512	20:54:51.375	20:54:51.921
18	512	20:54:51.993	20:54:52.534
19	512	20:54:52.606	20:54:53.244
20	512	20:54:53.316	20:54:53.867

Figure C-5. Text version of time-corrected HISTORY.X file.

```
Overhead-information files:

Perfor. measur. ID       = NBS/ITS ARPANET TEST
Run number               =  126
Type                     = SOURCE
Information ID           = USER
Source                   = NBS-UNIX (Gaithersburg)
Destination              = NTIA-ITS (Boulder)
Mo/Day/Yr                = 11/24/81
Start time (Hr:Min:Sec) = 20:54:35

  Data from file    overhead.x

        Record Code  Clock time

          1    23  20:54:35.091
          2    32  20:54:37.138
          3    23  20:54:37.313
          4    32  20:54:40.970
          5    23  20:54:41.042
          6    32  20:54:41.790
          7    23  20:54:41.862
          8    32  20:54:42.723
          9    23  20:54:42.795
         10    32  20:54:43.376
         11    23  20:54:43.448
         12    32  20:54:44.062
         13    23  20:54:44.134
         14    32  20:54:44.739
         15    23  20:54:44.811
         16    32  20:54:45.456
         17    23  20:54:45.528
         18    32  20:54:46.076
         19    23  20:54:46.148
         20    32  20:54:46.862
         21    23  20:54:46.934
         22    32  20:54:47.496
                  ~~~~~~~~~~~~
                  ~~~~~~~~~~~~
         36    32  20:54:51.921
         37    23  20:54:51.993
         38    32  20:54:52.534
         39    23  20:54:52.606
         40    32  20:54:53.244
         41    23  20:54:53.316
         42    32  20:54:53.867
         43    45  20:54:54.057
         44    11  20:54:54.071
Total # times = 44
```

Figure C-6. Text version of time-corrected OVERHEAD.X file.

```
NBS/ITS ARPANET TEST                                              0126SOURCE      USER
NBS-UNIX (Gaithersburg)   NTIA-ITS (Boulder)          11     50000.011811124
00000075275.0912377
00000075277.1383277
00000075277.3132377
00000075280.9703277
00000075281.0422377
00000075281.7903277
00000075281.8622377
00000075282.7233277
00000075282.7952377
00000075283.3763277
00000075283.4482377
00000075284.0623277
00000075284.1342377
00000075284.7393277
00000075284.8112377
00000075285.4563277
00000075285.5282377
00000075286.0763277
00000075286.1482377
00000075286.8623277
00000075286.9342377
00000075287.4963277
00000075287.5682377
00000075288.1133277
00000075288.1852377
00000075288.7663277
00000075288.8382377
00000075289.3903277
00000075289.4622377
00000075289.9873277
00000075290.0592377
00000075290.6633277
00000075290.7352377
00000075291.3033277
00000075291.3752377
00000075291.9213277
00000075291.9932377
00000075292.5343277
00000075292.6062377
00000075293.2443277
00000075293.3162377
00000075293.8673277
00000075294.0574577
00000075294.0711177
        -99.9990000
```

Figure C-7. Listing of FORT14 standard ASCII character file.

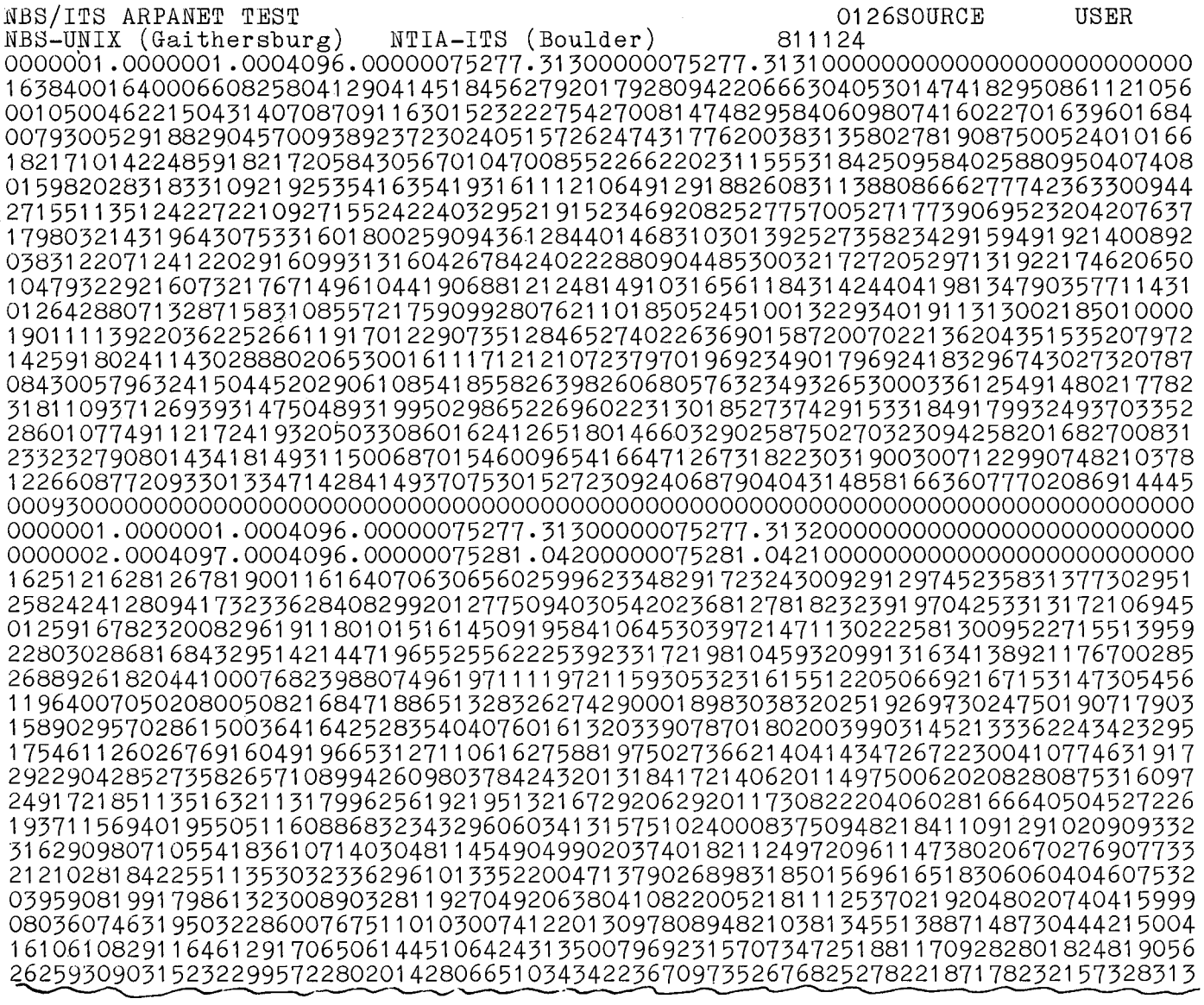


Figure C-8. Listing of FORT17 standard ASCII character file.

```
Overhead-information files:

Perfor. measur. ID      = NBS/ITS INITIAL ARPANET TEST
Run number              =   73
Type                    = SOURCE
Information ID          = USER
Source                  = NBS-UNIX (Gaithersburg)
Destination             = NTIA-ITS (Boulder)
Mo/Day/Yr               = 9/17/81
Start time (Hr:Min:Sec) = 22:14:18

  Data from file    sep/17-2214ov.x

        Record Code  Clock time       Differences

          1    23  22:14:18.196                          sep/17-2214ov.x    1
          2    32  22:14:19.942       1.746     1.746    sep/17-2214ov.x    1
          3    23  22:14:20.113       0.171     1.917    sep/17-2214ov.x    1
          4    32  22:14:20.428       0.315     0.486    sep/17-2214ov.x    1
          5    45  22:14:20.639       0.211     0.526    sep/17-2214ov.x    1
          6    11  22:14:20.646       0.007     0.218    sep/17-2214ov.x    1
          1    23  22:14:31.198      10.552    10.559    sep/17-2214ov.x    2
          2    32  22:14:32.552       1.354    11.906    sep/17-2214ov.x    2
          3    23  22:14:32.724       0.172     1.526    sep/17-2214ov.x    2
          4    32  22:14:33.046       0.322     0.494    sep/17-2214ov.x    2
          5    45  22:14:33.245       0.199     0.521    sep/17-2214ov.x    2
          6    11  22:14:33.255       0.010     0.209    sep/17-2214ov.x    2
          1    23  22:14:43.201       9.946     9.956    sep/17-2214ov.x    3
          2    32  22:14:43.635       0.434    10.380    sep/17-2214ov.x    3
          3    23  22:14:43.807       0.172     0.606    sep/17-2214ov.x    3
          4    32  22:14:44.196       0.389     0.561    sep/17-2214ov.x    3
          5    45  22:14:44.387       0.191     0.580    sep/17-2214ov.x    3
          6    11  22:14:44.394       0.007     0.198    sep/17-2214ov.x    3
          1    23  22:46:25.203       9.859     9.866    sep/17-2214ov.x  159
          2    32  22:46:26.711       1.508    11.367    sep/17-2214ov.x  159
          3    23  22:46:26.882       0.171     1.679    sep/17-2214ov.x  159
          4    32  22:46:27.324       0.442     0.613    sep/17-2214ov.x  159
          5    45  22:46:27.520       0.196     0.638    sep/17-2214ov.x  159
          6    11  22:46:27.528       0.008     0.204    sep/17-2214ov.x  159
          1    23  22:46:38.311      10.783    10.791    sep/17-2214ov.x  160
          2    32  22:46:40.168       1.857    12.640    sep/17-2214ov.x  160
          3    23  22:46:40.339       0.171     2.028    sep/17-2214ov.x  160
          4    32  22:46:40.747       0.408     0.579    sep/17-2214ov.x  160
          5    45  22:46:40.937       0.190     0.598    sep/17-2214ov.x  160
          6    11  22:46:40.945       0.008     0.198    sep/17-2214ov.x  160
```

Figure C-9. Listing of ACCESS.INFO text file.

3	23	22:14:20.113	0.171	1.917	sep/17-2214ov.x	1
3	23	22:14:32.724	0.172	1.526	sep/17-2214ov.x	2
3	23	22:14:43.807	0.172	0.606	sep/17-2214ov.x	3
3	23	22:14:55.725	0.172	1.527	sep/17-2214ov.x	4
3	23	22:15:06.853	0.171	0.653	sep/17-2214ov.x	5
3	23	22:15:18.795	0.171	1.596	sep/17-2214ov.x	6
3	23	22:15:30.702	0.171	1.506	sep/17-2214ov.x	7
3	23	22:15:41.899	0.171	0.700	sep/17-2214ov.x	8
3	23	22:15:52.873	0.180	0.670	sep/17-2214ov.x	9
3	23	22:16:05.641	0.171	1.439	sep/17-2214ov.x	10
3	23	22:16:16.930	0.186	0.724	sep/17-2214ov.x	11
3	23	22:16:28.762	0.171	1.557	sep/17-2214ov.x	12
3	23	22:16:39.952	0.171	0.749	sep/17-2214ov.x	13
3	23	22:16:52.672	0.171	1.466	sep/17-2214ov.x	14
3	23	22:17:04.617	0.171	1.414	sep/17-2214ov.x	15
3	23	22:17:15.865	0.171	0.657	sep/17-2214ov.x	16
3	23	22:17:26.907	0.187	0.696	sep/17-2214ov.x	17
3	23	22:17:38.687	0.171	1.481	sep/17-2214ov.x	18
3	23	22:17:50.004	0.191	0.792	sep/17-2214ov.x	19
3	23	22:18:02.640	0.171	1.432	sep/17-2214ov.x	20
3	23	22:18:14.696	0.171	1.487	sep/17-2214ov.x	21
3	23	22:18:26.749	0.186	1.539	sep/17-2214ov.x	22
3	23	22:18:39.365	0.171	2.033	sep/17-2214ov.x	23
3	23	22:18:51.459	0.170	1.161	sep/17-2214ov.x	24
3	23	22:42:43.860	0.171	1.649	sep/17-2214ov.x	141
3	23	22:42:55.953	0.171	1.744	sep/17-2214ov.x	142
3	23	22:43:08.890	0.171	1.682	sep/17-2214ov.x	143
3	23	22:43:21.946	0.171	1.735	sep/17-2214ov.x	144
3	23	22:43:34.763	0.186	1.554	sep/17-2214ov.x	145
3	23	22:43:47.902	0.171	1.690	sep/17-2214ov.x	146
3	23	22:44:00.814	0.171	1.602	sep/17-2214ov.x	147
3	23	22:44:12.741	0.171	1.534	sep/17-2214ov.x	148
3	23	22:44:24.888	0.171	1.680	sep/17-2214ov.x	149
3	23	22:44:36.615	0.172	1.409	sep/17-2214ov.x	150
3	23	22:44:48.903	0.171	1.693	sep/17-2214ov.x	151
3	23	22:45:01.804	0.171	1.594	sep/17-2214ov.x	152
3	23	22:45:13.653	0.171	1.446	sep/17-2214ov.x	153
3	23	22:45:25.932	0.171	1.722	sep/17-2214ov.x	154
3	23	22:45:38.715	0.171	1.510	sep/17-2214ov.x	155
3	23	22:45:50.886	0.171	1.680	sep/17-2214ov.x	156
3	23	22:46:02.834	0.172	1.630	sep/17-2214ov.x	157
3	23	22:46:14.682	0.171	1.480	sep/17-2214ov.x	158
3	23	22:46:26.882	0.171	1.679	sep/17-2214ov.x	159
3	23	22:46:40.339	0.171	2.028	sep/17-2214ov.x	160

Figure C-10. Listing of temporary text file TFIL.

22:14:43.807	0.606	sep/17-2214	3
22:23:39.861	0.642	sep/17-2214	48
22:24:39.872	0.650	sep/17-2214	53
22:15:06.853	0.653	sep/17-2214	5
22:17:15.865	0.657	sep/17-2214	16
22:25:02.888	0.664	sep/17-2214	55
22:20:50.877	0.665	sep/17-2214	34
22:25:13.889	0.666	sep/17-2214	56
22:23:28.888	0.668	sep/17-2214	47
22:15:52.873	0.670	sep/17-2214	9
22:17:26.907	0.696	sep/17-2214	17
22:15:41.899	0.700	sep/17-2214	8
22:19:27.928	0.717	sep/17-2214	27
22:16:16.930	0.724	sep/17-2214	11
22:16:39.952	0.749	sep/17-2214	13
22:20:26.971	0.760	sep/17-2214	32
22:22:03.997	0.778	sep/17-2214	40
22:17:50.004	0.792	sep/17-2214	19
22:20:03.037	0.825	sep/17-2214	30
22:19:16.263	1.054	sep/17-2214	26
22:18:51.459	1.161	sep/17-2214	24
22:21:52.406	1.190	sep/17-2214	39
22:21:02.570	1.357	sep/17-2214	35
22:33:34.787	1.362	sep/17-2214	97
22:30:29.067	1.832	sep/17-2214	82
22:33:22.211	1.875	sep/17-2214	96
22:19:04.110	1.894	sep/17-2214	25
22:41:28.243	1.916	sep/17-2214	135
22:14:20.113	1.917	sep/17-2214	1
22:42:31.216	1.948	sep/17-2214	140
22:21:40.176	1.951	sep/17-2214	38
22:46:40.339	2.028	sep/17-2214	160
22:18:39.365	2.033	sep/17-2214	23
22:41:53.288	2.077	sep/17-2214	137
22:34:01.352	2.129	sep/17-2214	99
22:34:14.863	2.298	sep/17-2214	100
22:42:18.575	2.362	sep/17-2214	139
22:38:20.621	2.386	sep/17-2214	120
22:34:40.877	2.406	sep/17-2214	102
22:32:33.669	2.429	sep/17-2214	92
22:22:41.715	2.485	sep/17-2214	43
22:35:31.713	2.487	sep/17-2214	106
22:26:50.862	2.628	sep/17-2214	64
22:40:39.166	4.836	sep/17-2214	131

Figure C-11. Listing of TABLE.ACC text file.

```
 Number of items:  160
Maximum:        4.836    Minimum:        0.606
Mean =    1.543    Std Deviation =    0.466

Histogram for Access Time File

        0.000              0.00
        0.000              0.00
        0.100              0.00
        0.100              0.00
        0.100              0.00
        0.200              0.00
        0.200              0.00
        0.200              0.00
        0.300              0.00
        0.300              0.00
        0.300              0.00
        0.400              0.00
        0.400              0.00
        0.400              0.00
        0.500              0.00
        0.500              0.00
        0.500              0.00
        0.600              0.00
        0.600              0.00
        0.600              6.87
        0.700              6.87
        0.700              0.00
        0.700              4.37
        0.800              4.37
        0.800              0.00
        0.800              0.62
        0.900              0.62
        0.900              0.00
        0.900              0.00
        1.000              0.00
        1.000              0.00
        1.000              0.62
        1.100              0.62
        1.100              0.00
        1.100              1.25
        1.200              1.25
        1.200              0.00
        1.200              0.00
        1.300              0.00
        1.300              0.00
        1.300              2.50
        1.400              2.50
        1.400              0.00
        1.400             23.13
        1.500             23.13
        1.500              0.00
        1.500             16.88
        1.600             16.88
        1.600              0.00
        1.600             26.88
        1.700             26.88
        1.700              0.00
        1.700              4.37
        1.800              4.37
```

Figure C-12. Listing of PLOT16ACC text file for access time histogram.

```
History-information files:   Transmit                          Receive

Perfor. measur. ID       = NBS/ITS ARPANET TEST                NBS/ITS ARPANET TEST
Run number               =  126                                 126
Type                     = SOURCE                              DESTINATION
Information ID           = USER                                USER
Source                   = NBS-UNIX (Gaithersburg)             NBS-UNIX (Gaithersburg)
Destination              = NTIA-ITS (Boulder)                  NTIA-ITS (Boulder)
Mo/Day/Yr                = 11/24/81                            11/24/81
Start time (Hr:Min:Sec)  = 20:54:37                            20:54:36
**File prefix (dd-hhmm)  = 24-2054

Times shown to nearest 1/1000 sec

 1   512  20:54:37.313          1    64  20:54:37.588
                                2   448  20:54:38.480          1.167   nov/24-2054hs.r   1
 2   512  20:54:41.042          3   512  20:54:41.837          0.795   nov/24-2054hs.r   2
 3   512  20:54:41.862          4    64  20:54:42.111
                                5   448  20:54:42.748          0.886   nov/24-2054hs.r   3
 4   512  20:54:42.795          6   512  20:54:43.418          0.623   nov/24-2054hs.r   4
 5   512  20:54:43.448          7   512  20:54:44.027          0.579   nov/24-2054hs.r   5
 6   512  20:54:44.134          8   512  20:54:44.768          0.634   nov/24-2054hs.r   6
 7   512  20:54:44.811          9   512  20:54:45.462          0.651   nov/24-2054hs.r   7
 8   512  20:54:45.528         10   512  20:54:46.120          0.592   nov/24-2054hs.r   8
 9   512  20:54:46.148         11   512  20:54:46.772          0.624   nov/24-2054hs.r   9
10   512  20:54:46.934         12   512  20:54:47.544          0.610   nov/24-2054hs.r   10
11   512  20:54:47.568         13   512  20:54:48.162          0.594   nov/24-2054hs.r   11
12   512  20:54:48.185         14   512  20:54:48.765          0.580   nov/24-2054hs.r   12
13   512  20:54:48.838         15   512  20:54:49.434          0.596   nov/24-2054hs.r   13
14   512  20:54:49.462         16   512  20:54:50.039          0.577   nov/24-2054hs.r   14
15   512  20:54:50.059         17   512  20:54:50.653          0.594   nov/24-2054hs.r   15
16   512  20:54:50.735         18   512  20:54:51.350          0.615   nov/24-2054hs.r   16
17   512  20:54:51.375         19   512  20:54:51.972          0.597   nov/24-2054hs.r   17
18   512  20:54:51.993         20   512  20:54:52.587          0.594   nov/24-2054hs.r   18
19   512  20:54:52.606         21   512  20:54:53.214          0.608   nov/24-2054hs.r   19
20   512  20:54:53.316         22   512  20:54:53.913          0.597   nov/24-2054hs.r   20

+++ block xmit time            = 16.003  with block rate =   1.25 blocks/sec, and bit rate =  5119.1 bits/sec
+++ block recv time            = 15.433  with block rate =   1.30 blocks/sec, and bit rate =  5308.2 bits/sec
+++ block transmission time    = 16.600  with block rate =   1.20 blocks/sec, and bit rate =  4935.0 bits/sec
```

Figure C-13. Listing of XFER.INFO text file.

2	448	20:54:38.480	1.167	nov/24-2054hs.r	1
3	512	20:54:41.837	0.795	nov/24-2054hs.r	2
5	448	20:54:42.748	0.886	nov/24-2054hs.r	3
6	512	20:54:43.418	0.623	nov/24-2054hs.r	4
7	512	20:54:44.027	0.579	nov/24-2054hs.r	5
8	512	20:54:44.768	0.634	nov/24-2054hs.r	6
9	512	20:54:45.462	0.651	nov/24-2054hs.r	7
10	512	20:54:46.120	0.592	nov/24-2054hs.r	8
11	512	20:54:46.772	0.624	nov/24-2054hs.r	9
12	512	20:54:47.544	0.610	nov/24-2054hs.r	10
13	512	20:54:48.162	0.594	nov/24-2054hs.r	11
14	512	20:54:48.765	0.580	nov/24-2054hs.r	12
15	512	20:54:49.434	0.596	nov/24-2054hs.r	13
16	512	20:54:50.039	0.577	nov/24-2054hs.r	14
17	512	20:54:50.653	0.594	nov/24-2054hs.r	15
18	512	20:54:51.350	0.615	nov/24-2054hs.r	16
19	512	20:54:51.972	0.597	nov/24-2054hs.r	17
20	512	20:54:52.587	0.594	nov/24-2054hs.r	18
21	512	20:54:53.214	0.608	nov/24-2054hs.r	19
22	512	20:54:53.913	0.597	nov/24-2054hs.r	20

Figure C-14. Listing of temporary text file TFIL.

16	512	20:54:50.039	0.577	nov/24-2054hs.r	14
7	512	20:54:44.027	0.579	nov/24-2054hs.r	5
14	512	20:54:48.765	0.580	nov/24-2054hs.r	12
10	512	20:54:46.120	0.592	nov/24-2054hs.r	8
13	512	20:54:48.162	0.594	nov/24-2054hs.r	11
17	512	20:54:50.653	0.594	nov/24-2054hs.r	15
20	512	20:54:52.587	0.594	nov/24-2054hs.r	18
15	512	20:54:49.434	0.596	nov/24-2054hs.r	13
19	512	20:54:51.972	0.597	nov/24-2054hs.r	17
22	512	20:54:53.913	0.597	nov/24-2054hs.r	20
21	512	20:54:53.214	0.608	nov/24-2054hs.r	19
12	512	20:54:47.544	0.610	nov/24-2054hs.r	10
18	512	20:54:51.350	0.615	nov/24-2054hs.r	16
6	512	20:54:43.418	0.623	nov/24-2054hs.r	4
11	512	20:54:46.772	0.624	nov/24-2054hs.r	9
8	512	20:54:44.768	0.634	nov/24-2054hs.r	6
9	512	20:54:45.462	0.651	nov/24-2054hs.r	7
3	512	20:54:41.837	0.795	nov/24-2054hs.r	2
5	448	20:54:42.748	0.886	nov/24-2054hs.r	3
2	448	20:54:38.480	1.↑67	nov/24-2054hs.r	1

Figure C-15. Listing of TABLE. XFR text file.

```
 Number of items:   20
Maximum:        1.167   Minimum:        0.577
Mean =   0.656   Std Deviation =   0.139

Histogram for Block Transfer File

        0.000           0.00
        0.000           0.00
        0.050           0.00
        0.050           0.00
        0.050           0.00
        0.100           0.00
        0.100           0.00
        0.100           0.00
        0.150           0.00
        0.150           0.00
        0.150           0.00
        0.500           0.00
        0.500           0.00
        0.550           0.00
        0.550           0.00
        0.550          50.00
        0.600          50.00
        0.600           0.00
        0.600          30.00
        0.650          30.00
        0.650           0.00
        0.650           5.00
        0.700           5.00
        0.700           0.00
        0.700           0.00
        0.750           0.00
        0.750           0.00
        0.750           5.00
        0.800           5.00
        0.800           0.00
        0.800           0.00
        0.850           0.00
        0.850           0.00
        0.850           5.00
```

Figure C-16. Listing of PLOT10ACC text file for block transfer time histogram.

APPENDIX D: SATELLITE RECEIVER TIME CORRECTIONS

As described in Section 4.1, performance-significant events at the monitored user/system interfaces in the ARPANET tests corresponded to system calls or system responses. The time of each interface event was obtained by including, in the XMIT and RECV programs, a command to read the GOES satellite clock receiver either immediately before the event, in the case of system calls, or immediately after the event, in the case of system responses. Because time information was transferred serially from the clock receiver to the host, there was a significant difference between the time read from the clock and the time at which the corresponding interface event actually occurred. This appendix contains a detailed description of the procedures used to correct event times for such differences, and is divided into five parts:

1. Clock function via the serial port.
2. Definition of time correction terms.
3. Time correction approach.
4. Calibration program and measurements.
5. Calibration results.

The corrections took into account both the time required for serial transmission of clock information and the average operating system response time. The serial transmission delay depended on the RS-232-C bit rate setting on the host computer communication cards, and these were different at the two receiver locations. This necessitated separate calibrations for the ITS and NBS sites. As a result of the calibrations, it is believed that the corrected event times had an absolute accuracy normally better than 5 milliseconds.

Prior to running the ARPANET tests, the time available at the serial communication port of each GOES satellite receiver and decoder was adjusted to account for the mean transmission path delay from the Wallops Island, VA transmitter to the Eastern GOES satellite and back down to the receiver site. Including the daily cyclic variation, the absolute time accuracy of the clock was about 1 millisecond.

1. Clock function via the serial port

In executing the command to read the clock, the computer software sends the single character "T" to the clock. At the receipt of the stop bit of the "T" character, the clock immediately takes a "snapshot" of the time (called

the time hack). The clock then transmits the time in a predefined format back to the computer communication card. The character format used by the clock varies, depending on the format statement last received from the computer. The format may include Julian day, hour, minute, seconds, and thousandths of seconds, as well as delimiters such as spaces and colons. The returned character string might appear, for example, as follows:

123 17:52:07.894 .

A very short format of only minutes, seconds, and milliseconds (without delimiters) may also be specified, resulting for instance, in the character string

5207894 .

The format is specified by sending an "F" followed by a string of characters which define the "picture" of the returned time.

2. Definition of time correction terms

A model representing the process of executing the command to read the satellite clock is depicted in Figure D-1. The command is immediately preceded and followed by interface events A and B, respectively, whose times T_A and T_B are to be deduced from the observed time hack T_H. After an initial delay t_1 by the operating system, the trigger character "T" is sent to the clock, requiring a transmission time t_2. The time hack T_H is recorded, then sent back to the computer communication card as a character string (according to the specified format), requiring a transmission time t_3. The process ends after a delay t_4 by the operating system in returning from the call.

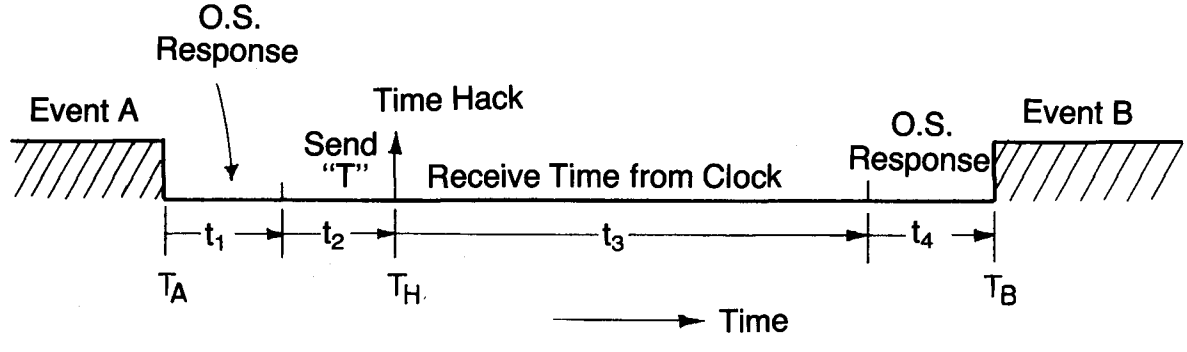


Figure D-1. Time correction diagram.

The desired event times T_A and T_B are related to the observed time hack T_H by

$$t_A = t_H - (t_1 + t_2) \qquad (D.1)$$

and

$$t_B = t_H + (t_3 + t_4) \quad . \qquad (D.2)$$

Thus, the time of an event immediately preceding a clock read is obtained by subtracting the correction $t_1 + t_2$ from the observed clock time, and the time of a event immediately following a clock read is obtained by adding the correction $t_3 + t_4$ to the observed clock time.

3. Time correction approach

To determine the time corrections for a given serial line speed and format, it was necessary to estimate the four times t_1, t_2, t_3, and t_4. An empirical approach was followed by writing a calibration program which read the clock 100 times in succession. In running this program, the event corresponding to A in Figure D-1 was the return from the previous clock read, and the event corresponding to B was the issuance of the subsequent clock read. The average (mean) time $(t_1 + t_2 + t_3 + t_4)$ to execute a clock read with n bytes of received clock data is equal to the average $\overline{\Delta t_n}$ of the observed differences between successive time hacks:

$$t_1 + t_2 + t_3 + t_4 = \overline{\Delta t_n} \quad . \qquad (D.3)$$

A second equation was obtained by relating the reception time t_3 of the n bytes of clock data to the transmission time t_2 of the single trigger character "T":

$$t_3 = nt_2 \quad . \qquad (D.4)$$

Using (D.4) and assuming that the two operating system delays (t_1 and t_4) are equal, (D.3) becomes

$$2t_1 + (n + 1)\, t_2 = \overline{\Delta t_n} \quad . \qquad (D.5)$$

The latter assumption was not critical (since $t_1 + t_4 \ll t_2$ at the serial speeds being used) and could cause only about a 5% error in the correction as a worst case.

The time corrections were obtained by running the calibration program twice, with two different values of n (the number of bytes of received clock

data), and observing the corresponding time hack difference $\overline{\Delta t_n}$. The resulting pair of equations, obtained from (D.5) by substituting the appropriate values of n and $\overline{\Delta t_n}$, was then solved for t_1 and t_2.

The time correction $(t_1 + t_2)$ for events preceding a clock read is independent of format length, whereas the correction $(t_3 + t_4)$ for events following a clock read are format dependent.

4. Calibration program and measurements

The calibration program consisted essentially of two statements, one defining a loop, and the other a "read clock" subroutine call within the loop. The "read clock" subroutine simply sent the trigger "T" to the clock and read the characters as they were returned by the clock. This software is listed in Figure D-2. The "read clock" subroutine is identical to that used in the XMIT and RECV test programs. Samples of calibration runs at a serial rate of 1200 bps are shown in Figure D-3. The first 12 readings are for 10-byte formats, the next 12 are for the 13-byte format used in the XMIT and RECV test programs, and the last 12 readings are for the full 20-byte format. Any two sets of readings serve to determine the time corrections, as described above; the third serves as a consistency check.

For the 13-byte extraction format, the data shown in Figure D-3 yield the values

$$t_1 = 3 \text{ ms}, \; t_2 = 9 \text{ ms}, \; t_3 = 117 \text{ ms}, \; t_4 = 3 \text{ ms} \quad ,$$

a total of 132 ms at the 1200 bps rate.

5. Calibration results

Calibration results for both for both a 1200 bps and 2400 bps serial interface are shown below in Table D-1.

Table D-1. Time Corrections for Observed Clock Times

Serial Rate (bps)	$t_1 + t_2$ (Event precedes clock read)	$t_3 + t_4$ (Event follows clock read)	Total Clock delay
1200 (at ITS)	12 ms	120 ms	132 ms
2400 (at NBS)	7 ms	65 ms	72 ms

```
/*  Main part of loop to read clock     */

        for (i=0;i<numb;i++)
        {
                readclock(fd_clock,i);
        }


/*      procedure to read time from satellite clock and store it
*       in a buffer, timebuf. there are two arguments: fd and n,
*       fd is the file descriptor of the clock line; n is the
*       row in the buffer to store the current clock reading.
*/


readclock(fd,n)

int fd;
int n;

{
        int byteswritten;
        int bytesread;

        char *buffer;

                /* the clock trigger is a "T"          */

        buffer = TRIGGER;
        byteswritten = write (fd,buffer,1);
        bytesread = read (fd,timebuf+n,CLOCKRECORDLENGTH);
        if (bytesread != CLOCKRECORDLENGTH)
                readerror(bytesread);

}
```

Figure D-2. Clock test program.

```
initial satellite time:     338 20:45:57.144
The clock is now initialized and ready to read times when triggered.

  0:  57:972
  1:  58:077
  2:  58:182
  3:  58:287
  4:  58:392
  5:  58:497
  6:  58:602
  7:  58:707
  8:  58:812     Serial rate = 1200 bps
  9:  58:917
 10:  59:022     Average time difference = 105 ms
 11:  59:127
 12:  59:232

  0:  204647915
  1:  204648047
  2:  204648179
  3:  204648313
  4:  204648446
  5:  204648578
  6:  204648710
  7:  204648842
  8:  204648974    Serial rate = 1200 bps
  9:  204649106
 10:  204649238    Average time difference = 132 ms
 11:  204649370
 12:  204649502

  0:  338 20:46:27:259
  1:  338 20:46:27:454
  2:  338 20:46:27:649
  3:  338 20:46:27:844
  4:  338 20:46:28:039
  5:  338 20:46:28:234
  6:  338 20:46:28:429
  7:  338 20:46:28:624
  8:  338 20:46:28:818    Serial rate = 1200 bps
  9:  338 20:46:29:013
 10:  338 20:46:29:208    Average time difference = 195 ms
 11:  338 20:46:29:403
 12:  338 20:46:29:598
```

Figure D-3. Clock calibration test runs.

Note that the correction $t_1 + t_2$ is applied to all even-numbered times (i.e., to T_2, T_4, T_6 and R_2, R_4, R_6, R_8) in the session profile in Figure 25, whereas $t_3 + t_4$ is applied to all odd-numbered times (i.e., to T_1, T_3, T_5 and R_1, R_3, R_5, R_7). At ITS, times for events preceding (or following) a clock read are obtained by subtracting 12 ms from (or adding 120 ms to) the observed time hack. At NBS, the corresponding corrections are 7 ms and 72 ms, respectively.

APPENDIX E: FILE STRUCTURE

This appendix provides a brief description of the file structure used in the ARPANET experiment. The UNIX file structure is hierarchical (i.e., it is tree structured) in that directories may contain subdirectories in addition to programs and data files. This structure can be nested with additional directories, programs, and files in the subdirectories, etc.

As stated in Section 5.4, files used in the ARPANET measurements were stored on a single disk. The attendant file structure began with a main directory (called /DISK2) entered from the system directory. The main directory in turn contained a set of subdirectories, as listed below:

main directory:	/disk2	ARPANET measurement subdirectories and files
subdirectories:	/disk2/assess	directory for standard FORTRAN performance assessment files
	/disk2/progs	"C" and FORTRAN source programs
	/disk2/doc	ARPANET measurement documentation
	/disk2/sh	command files to run test
	/disk2/data	subdirectories and files used in post-test processing and aggregation
	/disk2/data/sh	command files used in post-test processing
	/disk2/data/temp	text version of temporary INFO, PLOT, and TABLE files
	/disk2/data/sep	monthly directory for raw data
	/disk2/data/oct	monthly directory for raw data
	/disk2/data/nov	monthly directory for raw data
	. . .	
	. . .	
	. . .	

A comprehensive pictorial representation of the UNIX file structure for the NTIA/ITS host is presented in Figure E-1. Representative listings of the first eight subdirectories, plus a typical monthly directory (NOV), are shown in Figures E-2 through E-10.

Ideally, all files (source and compiled programs, command files, raw and processed data) used in the ARPANET experiment would have been stored on the dedicated disk. However, as described in Section 5.4, the 2-1/2 Megabyte storage of the disk was inadequate. To make some space available for conducting tests and processing the resulting data, source versions of the "C" and FORTRAN processing programs were moved to another disk from the /DISK2/PROGS subdirectory.

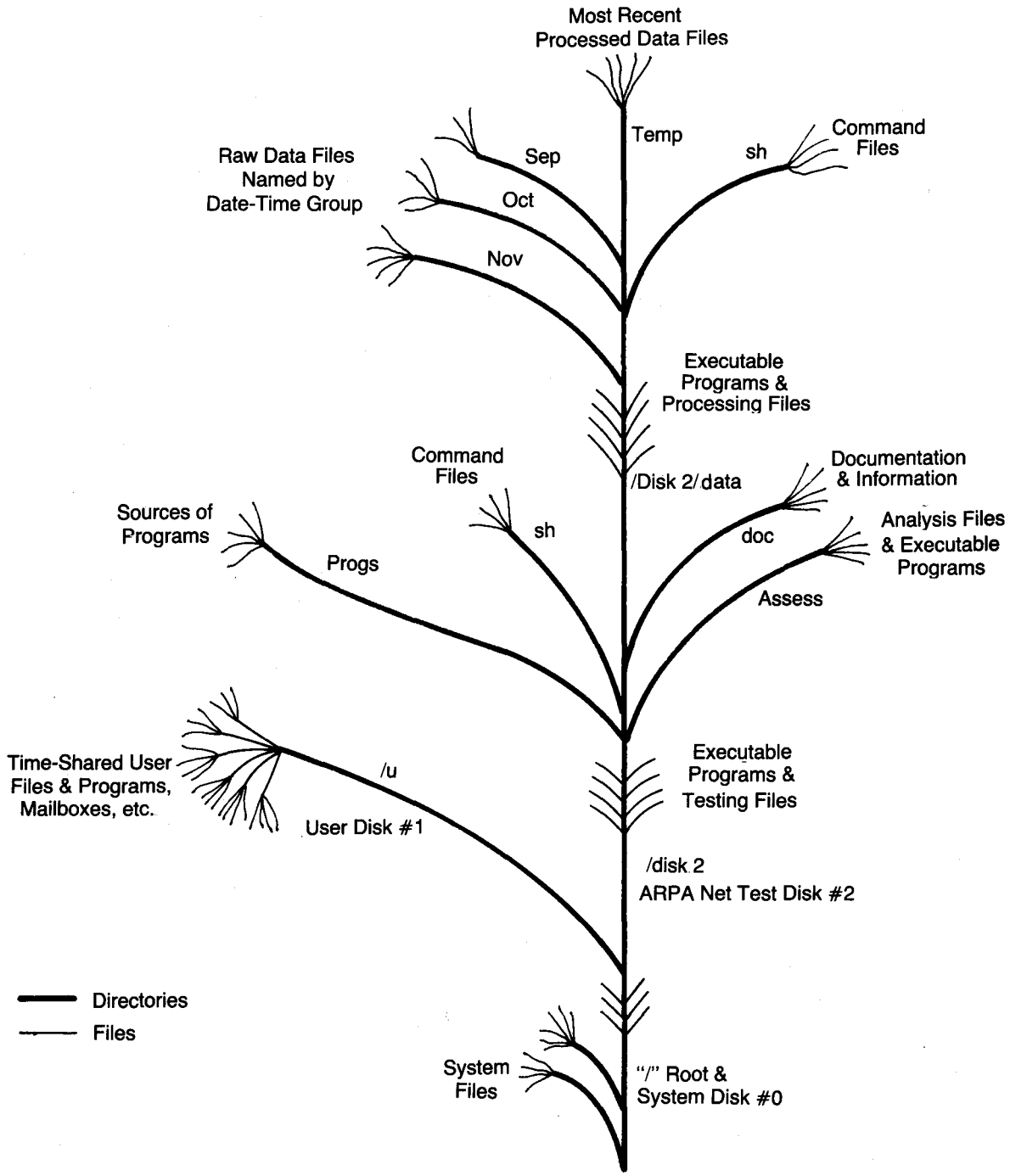


Figure E-1. UNIX file structure at the NTIA/ITS host.

```
Jan 13 09:19 1982  /disk2 Page 1

drwxrwxr--  2 spies  grp2     1072 Jan  5 18:32 assess
drwxrwxr--  8 davew  grp2     7184 Jan 12 15:21 data
-rw-rw-r--  1 davew  grp2    10112 Dec 31 13:15 data.r
-r--r--r--  2 davew  grp2    10240 Oct  1 19:18 data.x
drwxrwxr--  2 davew  grp2      208 Jan 11 12:42 doc
-rw-rw-rw-  1 dwight grp2       87 Dec 30 13:47 preface.r
-rw-rw-r--  1 davew  grp2       81 Dec 30 13:08 preface.x
drwxrwxr--  2 davew  grp2      800 Dec 31 17:37 progs
-rwsr-sr--  1 root   grp2     5926 Oct 14 18:12 qik_clock
-rwsr-sr--  1 root   grp2     7394 Dec 28 17:02 recv.o
-rwsr-sr--  1 davew  grp2     7394 Dec 28 17:02 recv.oslo
-rwsr-sr--  1 root   grp2     7256 Dec 28 17:03 recv.u
-rwsr-sr--  1 davew  grp2     7256 Dec 28 17:03 recv.uslo
-rwxr-xr--  1 root   grp2     7134 Dec 28 17:03 recv_nbs.u
-rwxr-xr-x  1 davew  grp2     3852 Apr 14 12:03 runupdate
drwxrwxr--  2 davew  grp2      208 Jan 12 18:08 sh
-rwsr-xr-x  1 root   grp2     4232 Dec 28 17:05 testclock
-rwsr-sr--  1 root   grp2     7822 Dec 28 17:01 xmit.o
-rwsr-sr--  1 davew  grp2     7822 Dec 28 16:59 xmit.oslo
-rwsr-sr--  1 root   grp2     7738 Dec 28 17:01 xmit.u
-rwsr-sr--  1 davew  grp2     7738 Dec 28 16:59 xmit.uslo
-rwxr-xr--  1 davew  grp2     7822 Dec 28 16:59 xmit_nbs.o
-rwxr-xr--  1 davew  grp2     7738 Dec 28 16:59 xmit_nbs.u
```

Figure E-2. Main directory /DISK2.

```
May 14 11:37 1982  /disk2/assess Page 1

-rw-rw-r--  1 davew  grp2     1062 Mar  9 14:23 24-2054fort14
-rw-rw-r--  1 davew  grp2     1182 Mar  9 14:23 24-2054fort15
-rw-rw-r--  1 davew  grp2    32643 Mar  9 14:23 24-2054fort17
-rw-rw-r--  1 davew  grp2    33129 Mar  9 14:23 24-2054fort18
-rwxr-xr-x  1 spies  grp2    40448 Mar 25 17:36 ANALYZ.out
-rwxr-xr-x  1 spies  grp2    32256 Mar  9 11:23 EPILOG.out
-rwxr-xr-x  1 spies  grp2    37888 Mar  9 11:21 PROLOG.out
-rw-rw-r--  1 spies  grp2      359 Mar 25 17:15 fort10
-rw-r--r--  1 spies  grp2      412 Mar  9 18:40 fort11
-rw-r--r--  1 spies  grp2      412 Mar 25 17:37 fort12
-rw-rw-r--  1 davew  grp2     1062 Mar 17 18:44 fort13
-rw-rw-r--  1 davew  grp2     1102 Mar 17 18:44 fort14
-rw-r--r--  1 spies  grp2     2805 Mar 25 17:37 fort15
-rw-rw-r--  1 davew  grp2    32643 Mar 17 18:44 fort16
-rw-rw-r--  1 davew  grp2    32400 Mar 17 18:47 fort17
-rw-r--r--  1 spies  grp2      182 Mar  9 18:42 fort18
-rw-r--r--  1 spies  grp2      182 Mar 25 17:44 fort19
-rw-r--r--  1 spies  grp2     3010 Mar 25 17:44 fort20
-rw-r--r--  1 spies  grp2      156 Mar 25 17:44 fort21
-rw-r--r--  1 spies  grp2     4946 Mar 25 17:44 fort25
-rw-r--r--  1 spies  grp2      250 Mar 25 17:44 fort35
-rwxr-xr--  1 spies  grp2      278 Oct  8 17:24 passrun
-rw-r--r--  1 spies  grp2      250 Mar 25 17:44 ttimes
```

Figure E-3. Subdirectory /DISK2/ASSESS.

```
Jan 14 09:50 1982  /disk2/progs Page 1

-rw-r--r--  1 davew  system   3858 Jan  6 17:23 accesstime.c
-rw-rw-r--  1 davew  system   2313 Jan  6 17:23 clock.ovh.h
-rw-r--r--  1 davew  system   2313 Jan  6 17:24 clock.usr.h
-rw-r--r--  1 davew  system   9260 Jan  6 17:23 distr.c
-rw-r--r--  1 davew  system   9464 Jan  6 17:23 histo.c
-rw-rw-r--  1 davew  system   9464 Jan  6 17:24 histo2.c
-rw-r--r--  1 davew  system  24458 Jan  6 17:23 merge.c
-rw-r--r--  1 davew  system    522 Jan  6 17:23 net.h
-rw-rw-r--  1 davew  system   9460 Jan  6 17:24 ovh.histo.c
-rw-rw-r--  1 davew  system   9460 Jan  6 17:24 ovh.histo2.c
-rw-rw-r--  1 davew  system   9220 Jan  6 17:25 qik_clock.c
-rw-rw-r--  1 davew  system  22660 Jan  6 17:23 recv_its.c
-rw-rw-r--  1 davew  system  22386 Jan  6 17:23 recv_nbs.c
-rw-r--r--  1 davew  system  19690 Jan  6 17:24 reform.c
-rw-r--r--  1 davew  system   4660 Jan  6 17:24 runupdate.c
-rw-rw-r--  1 davew  system   3056 Jan  6 17:23 show.c
-rw-r--r--  1 davew  system   2920 Jan  6 17:24 show.overhd.c
-rw-rw-r--  2 dwight grp2      502 Jan 13 17:01 statacc.f
-rw-rw-r--  1 davew  grp2      502 Jan 13 17:35 statxfr.f
-rw-r--r--  1 davew  system   3496 Jan  6 17:23 tell.f
-rw-r--r--  1 davew  system   3200 Jan 14 12:24 tell.overhd.f
-rw-r--r--  1 davew  system   3941 Jan  6 17:23 times.c
-rw-rw-r--  1 davew  system   4644 Jan  6 17:25 tweak.h12.c
-rw-rw-r--  1 davew  system   4642 Jan  6 17:24 tweak.h24.c
-rw-rw-r--  1 davew  system   4563 Jan  6 17:24 tweak.o12.c
-rw-rw-r--  1 davew  system   4561 Jan  6 17:24 tweak.o24.c
-rw-rw-r--  1 davew  system   6299 Jan  6 17:24 xfertime.c
-rw-rw-r--  1 davew  system  24277 Jan  6 17:23 xmit.c
-rw-rw-r--  1 davew  system  24259 Jan  6 17:24 xmit_its.c
-rw-rw-r--  1 davew  system  24337 Jan  6 17:23 xmit_nbs.c
```

Figure E-4. Subdirectory /DISK2/PROGS.

```
May 14 10:25 1982  /disk2/doc Page 1

-rw-rw-r--   1 davew  grp2      5823 Jun 18 12:33 HOWTODOIT
-rw-rw-r--   1 davew  grp2      3348 Jun 18 12:33 README
-rw-r--r--   1 davew  grp2       635 Dec 31 18:08 clockcal.doc
-rw-rw-r--   1 davew  grp2       580 Jan 11 12:38 grfiles
-rw-rw-rw-   1 davew  grp2      7818 Jun 18 12:34 how_to_run
-rw-rw-r--   1 davew  grp2      5823 Jun 18 12:33 howtodoit
-rw-rw-r--   1 davew  grp2      3348 Jun 18 12:33 readme
-rw-rw-r--   1 davew  grp2       615 Dec 18 13:23 report
-rw-rw-r--   1 davew  grp2      1274 Aug 17 17:43 whatfiles
```

Figure E-5. Subdirectory /DISK2/DOC.

```
Jan 13 09:20 1982  /disk2/sh Page 1

-rwxr-xr--   1 davew  grp2       183 Oct  6 09:32 movefort
-rwxr-xr--   1 spies  grp2       177 Oct  8 18:19 moveinput
-rwxr-xr--   1 davew  grp2       469 Dec 30 11:57 runrecv
-rwxr-xr--   1 davew  grp2       400 Dec 30 11:57 runxmit
```

Figure E-6. Subdirectory /DISK2/SH.

Jan 14 09:50 1982 /disk2/data Page 1

```
-rwxr-xr-x  1 davew  grp2     7312 Oct 29 16:53 accesstime
-rw-rw-r--  1 davew  grp2    10240 Jan 12 12:54 data.r
-r--r--r--  2 davew  grp2    10240 Oct  1 19:18 data.x
drwxr-xr-x  2 davew  grp2      368 Dec 30 15:43 dec
-rwxr-xr-x  1 davew  grp2     9766 Nov 18 17:27 distr
-rw-r--r--  1 davew  grp2    26478 Dec 30 14:13 dtg
-rw-rw-rw-  1 davew  grp2     3974 Dec 30 15:39 dtg.short
-rw-r--r--  1 davew  grp2     1314 Dec 21 12:52 h.info.r
-rw-r--r--  1 davew  grp2     1314 Dec 21 12:51 h.info.x
-rwxr-xr-x  1 davew  grp2    10210 Jan  4 14:01 histo
-rwxr-xr-x  1 davew  grp2     9940 Nov 23 17:11 histo2
-rw-rw-r--  2 davew  grp2      586 Jan 12 12:40 history.r
-rw-rw-r--  2 davew  grp2      526 Jan 12 12:40 history.x
-r-xr-xr--  1 davew  grp2    15320 Sep  9 12:22 merge
drwxr-xr-x  2 davew  grp2     2560 Jan 12 13:25 nov
-rw-r--r--  1 davew  grp2     1824 Dec 21 12:53 o.info.r
-rw-r--r--  1 davew  grp2     1656 Dec 21 12:52 o.info.x
drwxr-xr-x  2 davew  grp2     1472 Jan 11 12:24 oct
-rw-rw-r--  1 davew  grp2      726 Jan 12 12:30 overhead.r
-rw-rw-r--  1 davew  grp2      654 Jan 12 12:36 overhead.x
-r-xr-xr--  1 davew  grp2     9936 Oct  1 19:19 ovh.histo
-rwxr-xr-x  1 davew  grp2     7150 Jan  4 18:40 ovh.histo2
-r-xr-xr--  1 davew  grp2    13234 Oct  1 19:17 reform
drwxr-xr-x  2 davew  grp2     2400 Jan  5 16:32 sep
drwxr-xr-x  2 davew  grp2      400 Jan 12 17:57 sh
-r-xr-xr-x  1 davew  grp2     6962 Nov 12 17:04 show
-r-xr-xr-x  1 davew  grp2     6872 Oct  1 19:16 show.overhd
-r-xr-xr--  1 davew  grp2    10752 Dec 30 12:26 tell
drwxr-xr-x  2 davew  grp2      416 Jan 12 12:45 temp
-r-xr-xr--  1 davew  grp2     7564 Oct  8 10:09 times
-r-xr-xr--  1 davew  grp2     7870 Oct  1 19:17 tweak.h12
-r-xr-xr--  1 davew  grp2     7870 Oct  1 19:17 tweak.h24
-r-xr-xr--  1 davew  grp2     7648 Oct  1 19:17 tweak.o12
-r-xr-xr--  1 davew  grp2     7648 Oct  1 19:17 tweak.o24
-rwxr-xr-x  1 davew  grp2     8768 Nov 20 20:19 xfertime
```

Figure E-7. Subdirectory /DISK2/DATA.

```
Jan 14 09:45 1982  /disk2/data/sh Page 1

-rwxr-xr--  1 davew  grp2      1266 Jan  4 12:09 doit.o
-rwxr-xr--  1 davew  grp2      1332 Jan  4 12:10 doit.u
-rwxr-xr--  1 davew  grp2       231 Sep 30 17:21 getdtg
-rwxr-xr--  1 davew  grp2       486 Oct 14 16:12 graph.acc
-rwxr-xr--  1 davew  grp2       431 Nov 17 11:56 graph.xfr
-rwxr-xr--  1 davew  grp2       264 Oct 15 23:36 mover
-rwxr-xr--  1 davew  grp2       242 Oct 15 23:36 movex
-rwxr-xr--  1 davew  grp2       787 Nov 10 10:14 process_for
-rwxr-xr--  1 davew  grp2       924 Nov 10 09:51 process_qik
-rwxr-xr--  1 davew  grp2       347 Sep 30 17:24 storem
-rwxr-xr--  1 davew  grp2        77 Oct 22 15:26 tab.acc
-rwxr-xr--  1 davew  grp2       151 Nov 17 11:37 tab.xfr
-rwxr-xr--  1 davew  grp2       103 Oct  7 18:44 trim.table
-rw-rw-r--  1 davew  grp2       246 Oct 23 10:49 wipeout
```

Figure E-8. Subdirectory /DISK2/DATA/SH.

```
May 13 13:23 1982  /disk2/data/temp Page 1

-rw-rw-rw-  1 davew  system  67919 May 12 18:29 access.info
-rw-r--r--  1 davew  grp2    13224 May 12 18:33 bigtable.acc
-rw-r--r--  1 davew  grp2     6988 May 13 14:16 bigtable.xfr
-rw-r--r--  1 davew  grp2       12 May 13 14:14 date
-rw-r--r--  1 davew  system   1477 May 13 14:09 h.info.r
-rw-r--r--  1 davew  system   1348 May 13 14:11 h.info.x
-rw-r--r--  1 davew  system   1825 May 13 14:09 o.info.r
-rw-r--r--  1 davew  system   1657 May 13 14:12 o.info.x
-rw-r--r--  1 davew  system   1010 May 13 14:16 plot10xfr
-rw-r--r--  1 davew  system   1428 Mar 23 20:02 plot16acc
-rw-rw-rw-  1 davew  grp2     6612 May 12 18:33 table.acc
-rw-r--r--  1 davew  system   1371 May 13 14:15 table.xfr
-rw-r--r--  1 davew  system   1371 May 13 14:15 tfil
-rw-r--r--  1 davew  system   2840 May 13 14:14 xfer.info
```

Figure E-9. Subdirectory /DISK2/DATA/TEMP.

Jan 14 08:47 1982 /disk2/data/nov Page 1

```
-rw-r--r--  1 davew  grp2     1233 Dec 18 17:42 03-1432tab.xfr
-rw-r--r--  1 davew  grp2     1302 Jan  6 11:52 03-1435tab.xfr
-rw-r--r--  1 davew  grp2     1233 Jan  6 12:13 04-1625tab.xfr
-rw-r--r--  1 davew  grp2     1233 Jan  6 12:28 04-1627tab.xfr
-rw-r--r--  1 davew  grp2     1233 Jan  6 12:46 05-2135tab.xfr
-rw-r--r--  1 davew  grp2     1302 Jan  6 13:05 05-2137tab.xfr
-rw-r--r--  1 davew  grp2     1233 Jan  6 13:46 10-1418tab.xfr
-rw-r--r--  1 davew  grp2     1302 Jan  6 14:09 10-1421tab.xfr
-rw-r--r--  1 davew  grp2     1371 Jan 11 17:32 12-1945tab.xfr
-rw-r--r--  1 davew  grp2     1371 Jan 11 13:26 13-1447tab.xfr
-rw-r--r--  1 davew  grp2     1371 Jan 11 14:38 13-1449tab.xfr
-rw-r--r--  1 davew  grp2     1422 Jan 12 13:21 18-1553tab.xfr
-rw-r--r--  1 davew  grp2     1371 Jan 11 15:11 18-1557tab.xfr
-rw-r--r--  1 davew  grp2      966 Jan 12 13:24 18-1645tab.xfr
-rw-r--r--  1 davew  grp2     1371 Nov 19 18:01 18-1651tab.xfr
-rw-r--r--  1 davew  grp2     1371 Nov 20 13:59 20-1815tab.xfr
-rw-r--r--  1 davew  grp2     1371 Nov 20 15:13 20-1818tab.xfr
-rw-rw-r--  1 davew  grp2     2560 Dec  1 10:42 21-0131data.r
-rw-rw-r--  1 davew  grp2      966 Nov 20 20:45 21-0131his.r
-rw-rw-rw-  1 davew  grp2      926 Nov 20 20:45 21-0131his.x
-rw-rw-rw-  1 davew  grp2      349 Nov 20 20:45 21-0131log.r
-rw-rw-rw-  1 davew  grp2      390 Nov 20 20:45 21-0131log.x
-rw-rw-r--  1 davew  grp2     1230 Nov 20 20:45 21-0131ovh.r
-rw-rw-rw-  1 davew  grp2     1182 Nov 20 20:45 21-0131ovh.x
-rw-r--r--  1 davew  grp2     1923 Nov 30 19:12 21-0131tab.xfr
-rw-rw-r--  1 davew  grp2     2560 Dec 18 17:34 21-0138data.r
-rw-rw-r--  1 davew  grp2      706 Nov 20 20:46 21-0138his.r
-rw-rw-rw-  1 davew  grp2      926 Nov 20 20:46 21-0138his.x
-rw-rw-rw-  1 davew  grp2      349 Nov 20 20:46 21-0138log.r
-rw-rw-rw-  1 davew  grp2      390 Nov 20 20:45 21-0138log.x
-rw-rw-r--  1 davew  grp2      918 Nov 20 20:46 21-0138ovh.r
-rw-rw-rw-  1 davew  grp2     1182 Nov 20 20:46 21-0138ovh.x
-rw-r--r--  1 davew  grp2     1190 Dec  1 10:51 21-0138tab.xfr
-rw-rw-r--  2 davew  grp2      586 Jan 12 12:40 24-2054hs.r
-rw-rw-r--  2 davew  grp2      526 Jan 12 12:40 24-2054hs.x
-rw-r--r--  1 davew  grp2     1371 Jan 12 12:45 24-2054tab.xfr
-rw-r--r--  1 davew  grp2      894 Jan 12 13:22 24-2057tab.xfr
```

Figure E-10. Representative monthly directory /DISK2/DATA/NOV.

APPENDIX F: TEST LOG FILE

This appendix provides a brief description and a sample of a LOG file recorded onto disk during an ARPANET test. The information in the LOG file consists of:

1. System date and time.
2. Who is logged on.
3. The process status.
4. Clock times and comparisons prior to the test.
5. The name of the test program.
6. The narrative from the test.
7. Clock times and comparison following the test.
8. The comparison between the transmitted and received user information.

All items above (except 2, 3, 4, and 7) are also displayed to the operator. The LOG files become part of the permanent archive, with the raw data. A sample LOG file from a user information transfer test, recorded at the destination (ITS) site, is shown in Figure F-1.

```
Tue Nov 24 12:52:23 PST 1981
davew   tty1 Nov 24 12:47
F   S UID   PPID   PID NCE CPU PRI   ADDR   SZ WCHAN TTY TIME COMMAND
  3S   0      0     0   0 145-100   1376    2 110456 ?  0:00 scheduler
  1S   0      0     1   0   0  40   1453    4 110524 ?  0:00 /etc/init
  1S   0      1  4866   0   0  10   1721    7 134762 ?  0:00 - -
  1S   0      1 17411   0   0  10   2134    7 135062 ?  0:00 - D
  1R   0      1     4   0   0 -50   1505    3        ?  0:00 /etc/update
  1S  41      1 24325   0   0  40   2220   11 110674 ?  0:00 -Shell
  1S  41  24325 27657   0   4  40   2453   11 111044 ?  0:00 /bin/sh sh/runre
  1R  41  27657 14858   0  80 104   3126   18        ?  0:00 ps axl
  0S   0      1   774   0   0  10  10131    7 135162 ?  0:00 - D
  1S   0      1  4103   0   0  13   1531    4 135376 ?  0:00 /etc/init
  0S   0      1 11528   0   0  13  10101    4 135436 ?  0:00 /etc/init
  1S   0      1  4620   0   0   2   2005    5 117352 ?  0:00 ftpsvr
  1S   1      1  1294   0   0   1   5470   60 137216 ?  0:00 -Largedaemon
  1S   1   1294 22288 -15   0 -25   2655    5 137166 ?  0:00 smalldaemon
  1S   0      1   529   0   0  13   1667    4 135476 ?  0:00 /etc/init
  0S   0      1    18   0   0  13  10356    4 135536 ?  0:00 /etc/init
  1S   0      1 17427   0   0   2   3614   14 117302 ?  0:00 svrtel
  1S   0      1 19988   0   0   2   3767    6 117272 ?  0:00 srvrtelnet
  1S   0      1 24341   0   0  90   5170   24 140000 ?  0:00 mailer 600 5 yes
  1S   0      1 32279   0   0   2   4672   21 117322 ?  0:00 netpr

20 used,  10 unused

Initial satellite time:     328 20:53:28.345
timeoff =: -1 minutes
Computer clock reads Tue Nov 24 13:52:45 1981

 starting recv.u

---------- network reception -------------
test beginning, Tue Nov 24 13:52:50 1981

Test to access network with 1 listens

Attempting open # 1, Opened, Recv complete, Transaction complete

10240 characters at 202 baud
test completed, Tue Nov 24 13:54:15 1981

---------------

Initial satellite time:     328 20:55:08.078
timeoff =: -1 minutes
Computer clock reads Tue Nov 24 13:54:25 1981

 compare data.r and data.x
```

Figure F-1. Sample LOG file.

APPENDIX G: LISTINGS OF COMMAND FILES

This appendix presents a brief summary and a listing of each of the command files used in driving the data through the bubble chart of Figure 26. The command files are grouped under three major functions: On-Line (Real-Time) Data Extraction, File Consolidation, and Post-Test Processing and Archiving. Figure numbers refer to listings, which are presented as a complete set following the command file summaries.

I. On-Line (Real-Time) Data Extraction

A. RUNRECV (Figure G-1) runs RECV program:

1. Writes date and who is using the system into specified LOG file.
2. Initializes and reads clocks, storing times in LOG file.
3. Runs RECV program with output going to temporary file.
4. Concatenates temporary file to LOG file and removes temporary file.
5. Gets times from clocks and stores in LOG file.
6. Compares the received and transmitted data files for equality, writing results in LOG file.
7. Increments test number in the PREFACE files.

B. RUNXMIT (Figure G-2) runs XMIT program:

1. Writes date and who is using the system into specified LOG file.
2. Initializes and reads clocks, storing times in LOG file.
3. Runs XMIT program with output going to temporary file.
4. Concatenates temporary file to LOG file and removes temporary file.
5. Gets times from clocks and stores in LOG file.
6. Increments test number in the PREFACE files.

II. File Consolidation

A. MOVER (Figure G-3) moves the four receive files created by a test into the data directory, and appends test number.

B. MOVEX (Figure G-4) moves the three transmit files created by a test into the data directory and appends test number.

C. GETDTG (Figure G-5) determines the date-time group prefix for use in naming raw data files.

D. STOREM (Figure G-6) archives the seven files from the data directory to a month directory with the dtg as a prefix to their name.

III. Post-test Processing and Archiving

A. DOIT.O (Figure G-7) performs post-test processing of access/disengagement tests:

1. Compares transmitted and received data files and outputs results to line printer.
2. Removes all temporary HISTORY, OVERHEAD and INFO files left from previous processing.
3. Runs PROCESS_QIK on transmit and receive files and produces ASCII files for standard FORTRAN processing (PROCESS_FOR).
4. Moves FORTRAN files to ASSESS directory.
5. Produces tables used for generating histograms of access times (GRAPH.ACC) and outputs sorted table to line printer and stores with raw data on disk.

B. DOIT.U (Figure G-8) performs post-test processing of user information transfer tests:

1. Compares transmit and receive data files and outputs results to line printer.
2. Removes all temporary HISTORY, OVERHEAD, and INFO files left from previous processing.
3. Runs PROCESS_QIK on transmit and receive files and produces ASCII files for standard FORTRAN processing (PROCESS_FOR).
4. Moves FORTRAN files to ASSESS directory.
5. Produces tables used for generating histograms of block transfer time data with GRAPH.XFR and outputs sorted table to line printer and stores with raw data on disk.

C. PROCESS_QIK (Figure G-9) produces "quick-look" tables and the time-corrected HISTORY and OVERHEAD files:

1. Prints out LOG files.
2. Creates temporary files containing corrected times.
3. Prints temporary information files on line printer.

D. PROCESS_FOR (Figure G-10) creates FORTRAN-readable files from HISTORY and OVERHEAD files:

1. Merges HISTORY and DATA file.
2. Reformats OVERHEAD file.
3. Moves four pertinent FORT files to ASSESS directory.

E. GRAPH.ACC (Figure G-11) creates and maintains data files of access times for histograms:

1. Runs ACCESSTIME on OVERHEAD file to obtain text files with access times.
2. Selects only times that pertain to access times and sorts them in ascending order of access time (TABLE.ACC).
3. Creates histogram data by running OVH.HISTO.
4. Updates cumulative access time tables.

F. GRAPH.XFR (Figure G-12) creates and maintains data files of block transfer times for histograms.

1. Runs XFERTIME on HISTORY files to obtain test files with block transfer times; prints table.
2. Selects only transfer times and sorts in ascending order of transfer time (TABLE.XFR).
3. Creates histogram files by running HISTO.
4. Updates cumulative transfer time tables.

G. TAB.ACC (Figure G-13) edits and sorts table produced by ACCESSTIME for use in making histograms.

H. TAB.XFR (Figure G-14) edits and sorts table produced by XFERTIME for use in making histograms.

I. TRIM.TABLE (Figure G-15) edits TABLE.ACC to a more condensed form.

J. WIPEOUT (Figure G-16) removes all raw data from referenced test,leaving the processed table of times.

```
Dec 30 08:57 1981  runrecv Page 1


if $N != 1 goto start
echo "usage:  % sh sh/runrecv <file name> <test #>"
goto end
: start
echo $1 program for test $2 will go into log.r$2
(date;who;ps axl) >log.r$2
qik_clock >>log.r$2
echo starting $1 for test $2 >>log.r$2
echo "----------------------"
$1 | tee log.r
echo $1 done
cat log.r >>log.r$2
rm log.r
qik_clock >>log.r$2
echo compare data.r and data.x >>log.r$2
cmp data.r data.x >>log.r$2
echo "   "
runupdate preface.x; runupdate preface.r
: end
echo
```

Figure G-1. Command file RUNRECV.

```
Dec 30 08:57 1981  runxmit Page 1


if $N != 1 goto start
echo "usage:  % sh sh/runxmit <file name> <test #>"
goto end
: start
echo $1 program for test $2 will go into log.x$2
(date;who;ps axl) >log.x$2
qik_clock >>log.x$2
echo starting $1 for test $2 >>log.x$2
echo "----------------------"
$1 | tee log.x
echo $1 done
cat log.x >>log.x$2
rm log.x
qik_clock >>log.x$2
echo "   "
runupdate preface.x; runupdate preface.r
: end
echo
```

Figure G-2. Command file RUNXMIT.

```
Oct 15 20:36 1981  mover Page 1


if -r log.r$1 goto error
echo moving recv $1 files to data dir
mv ../data.r data.r$1
mv ../history.r history.r$1
mv ../overhead.r overhead.r$1
mv ../log.r$1 .
lsm *.[rx]$1
su chown davew *.[rx]$1
echo
exit
: error
echo files exist with $1 suffix, no moving done
```

Figure G-3. Command file MOVER.

```
Oct 15 20:36 1981  movex Page 1


if -r log.x$1 goto error
echo moving xmit $1 files to data dir
mv ../history.x history.x$1
mv ../overhead.x overhead.x$1
mv ../log.x$1 .
lsm *.[rx]$1
su chown davew *.[rx]$1
echo
exit
: error
echo files exist with $1 suffix, no moving done
```

Figure G-4. Command file MOVEX.

```
Mar 24 08:38 1982  sh/getdtg Page 1


set a = $1
: loop
echo getting dtg for test $a
show 0 history.x$a | tee junk
cat junk >>dtg
set a + 1
echo looking for history.x$a
if -r history.x$a goto loop
rm junk
ed dtg
v/^[* ][* ][DF]/d
w dtg.short
q
echo done
```

Figure G-5. Command file GETDTG.

```
Sep 30 14:24 1981  storem Page 1


if $N = 4 goto start
echo "usage:  % sh/storem <month> <dd-hhmm> <test #>"
goto end
: start
echo moving test# $3 files to $1/$2names
mv log.x$3 $1/$2log.x
mv log.r$3 $1/$2log.r
mv history.x$3 $1/$2his.x
mv history.r$3 $1/$2his.r
mv overhead.x$3 $1/$2ovh.x
mv overhead.r$3 $1/$2ovh.r
mv data.r$3 $1/$2data.r
echo done all archiving
: end
echo " "
```

Figure G-6. Command file STOREM.

```
if $N = 4 goto start2
echo "usage:  % sh sh/doit <mon> <dd-hhmm> [<receive site>]"
if $N = 3 goto start1
echo recv site default = its, otherwise use nbs
goto end
: start1
set b = its
goto start
: start2
set b = $3
: start
echo comparing xmit and recv data files
set a = log.compare
date >$a
echo comparing $1/$2data.r with data.x >>$a
cmp $1/$2data.r data.x >>$a
echo done >>$a
print $a
rm $a
echo compare done
if -r history.r rm -f history.r
if -r history.x rm -f history.x
if -r overhead.r rm -f overhead.r
if -r overhead.x rm -f overhead.x
if -r $1/*ov.x rm -f $1/*ov.x
if -r data.r rm -f data.r
ln $1/$2data.r data.r
echo process recv $1/$2 files for $b
sh sh/process_qik r $1/$2 $b
echo create ascii fortran files for recv
sh sh/process_for r $1/$2
echo process xmit $1/$2 files - the recv site was $b
sh sh/process_qik x $1/$2 $b
echo create ascii fortran files for xmit
sh sh/process_for x $1/$2
echo "   "
echo done all processing,  moving fortran files to assess/$2fort
mv fort14 ../assess/$2fort14
mv fort15 ../assess/$2fort15
mv fort17 ../assess/$2fort17
mv fort18 ../assess/$2fort18
echo starting graphical analysis
echo $1 $2 >temp/date
sh sh/graph.acc $1 $2 16
cp temp/table.acc $1/$2tab.acc
pr -2 -w120 $1/$2tab.acc >/dev/lp
echo done
: end
echo " "
```

Figure G-7. Command file DOIT.O

```
if $N = 4 goto start2
echo "usage:  % sh sh/doit <mon> <dd-hhmm> [<receive site>]"
if $N = 3 goto start1
echo recv site default = its, otherwise use nbs
goto end
: start1
set b = its
goto start
: start2
set b = $3
: start
echo comparing xmit and recv data files
set a = log.compare
date >$a
echo comparing $1/$2data.r with data.x >>$a
cmp $1/$2data.r data.x >>$a
echo done >>$a
print $a
rm $a
echo compare done
if -r history.r rm -f history.r
if -r history.x rm -f history.x
if -r overhead.r rm -f overhead.r
if -r overhead.x rm -f overhead.x
if -r $1/*ov.x rm -f $1/*ov.x
if -r $1/*hs.x rm -f $1/*hs.x
if -r $1/*hs.r rm -f $1/*hs.r
if -r data.r rm -f data.r
ln $1/$2data.r data.r
echo process recv $1/$2 files for $b
sh sh/process_qik r $1/$2 $b
echo create ascii fortran files for recv
sh sh/process_for r $1/$2
echo process xmit $1/$2 files - the recv site was $b
sh sh/process_qik x $1/$2 $b
echo create ascii fortran files for xmit
sh sh/process_for x $1/$2
echo "   "
echo done all processing,  moving fortran files to assess/$2fort
goto bypass
mv fort14 ../assess/$2fort14
mv fort15 ../assess/$2fort15
mv fort17 ../assess/$2fort17
mv fort18 ../assess/$2fort18
: bypass
echo starting graphical analysis
echo $1 $2 >temp/date
sh sh/graph.xfr $1 $2 10
cp temp/table.xfr $1/$2tab.xfr
print $1/$2tab.xfr
echo done
: end
echo " "
```

Figure G-8. Command file DOIT.U.

```
if $N != 4 goto message
set b = $3
goto begin
: message
echo "useage:  % sh sh/process_qik <x or r> <mon/dd-hhmm> [<recv site>]"
if $N != 3 goto end
set b = its
: begin
echo process for a test with $b as recv site
set a = $1
set c = $2
set d = 12
set e = recv
set f = 18
if $a = r goto start2
if $a = x goto start1
echo bad 1st parameter, use 'r' for recv or 'x' for xmit
goto end
: start1
set e = xmit
set f = 17
if $b != its goto start
set d = 24
goto start
: start2
if $b != nbs goto start
set d = 24
: start
echo $e $clog.$a printing out
pr $clog.$a >/dev/lp &
echo starting quick-look file info for $e at $d00 baud clock speed
tweak.h$d 160 $chis.$a >temp/h.info.$a
mv hist.tweaked history.$a
echo file temp/h.info.$a now done
tweak.o$d 1280 $covh.$a >temp/o.info.$a
mv ovhd.tweaked overhead.$a
echo file temp/o.info.$a now done
: wrapup
echo all $e tweaking of times done
pr temp/h.info.$a >/dev/lp
: end
echo
```

Figure G-9. Command file PROCESS_QIK.

```
if $N = 3 goto begin
echo "useage:  % sh sh/process_for <x or r> <mon/dd-hh> "
goto end
: begin
set a = $1
set c = $2
set e = recv
set f = 18
if $a = r goto start2
if $a = x goto start1
echo bad 1st parameter, use 'r' for recv or 'x' for xmit
goto end
: start1
set e = xmit
set f = 17
: start2
date >log.merge.$1
echo start of $e merge for file history.$a and data.$a
if -r data.$a goto continue
ln $cdata.$a data.$a
: continue
echo starting $e merge
merge history.$a data.$a >>log.merge.$1
echo $e merge done
mv fort90 fort$f
: overhd
reform overhead.$a
if $a = r mv fort80 fort15
if $a = x mv fort80 fort14
echo overhead.$a file reformatted to fortran readable file
: wrapup
echo all $e processing done
echo "directory of appropo files: "
ls -l fort* history.* overhead.*
: end
echo
```

Figure G-10. Command file PROCESS_FOR.

```
Oct 14 13:12 1981  graph.acc Page 1


if $N = 4 goto start
echo "useage % sh sh/graph.acc <mon> <dd-hhmm> <#bars>"
goto end
: start
ln overhead.x $1/$2ov.x
accesstime 1280 $1/$2ov.x >temp/access.info
sh sh/tab.acc temp/access.info
wc temp/table.acc
ovh.histo $3 temp/table.acc >temp/plot$3acc
if -r temp/slimtable.acc rm temp/slimtable.acc
sh/trim.table
mv temp/slimtable.acc temp/table.acc
mv temp/bigtable.acc temp/big.acc
cat temp/table.acc >>temp/big.acc
sort -n +1 temp/big.acc >temp/bigtable.acc
rm temp/big.acc
: end
```

Figure G-11. Command file GRAPH. ACC.

```
Nov 17 08:56 1981  graph.xfr Page 1


if $N = 4 goto start
echo "useage % sh sh/graph.xfr <mon> <dd-hhmm> <#bars>"
goto end
: start
ln history.x $1/$2hs.x
ln history.r $1/$2hs.r
xfertime 24 $1/$2hs.x $1/$2hs.r >temp/xfer.info
print temp/xfer.info &
sh sh/tab.xfr temp/xfer.info
wc temp/table.xfr
histo $3 temp/table.xfr >temp/plot$3xfr
mv temp/bigtable.xfr temp/big.xfr
cat temp/table.xfr >>temp/big.xfr
sort -n +3 temp/big.xfr >temp/bigtable.xfr
rm temp/big.xfr
: end
```

Figure G-12. Command file GRAPH.XFR.

```
Oct 22 12:26 1981  tab.acc Page 1

ed -  $1 >temp/tfil
1,$g/     3    23/p
q
sort -n +4 temp/tfil >temp/table.acc
```

Figure G-13. Command file TAB.ACC.

```
Nov 17 08:37 1981  tab.xfr Page 1

ed -  $1 >temp/tfil
g/hs\.r/p
q
ed temp/tfil
g/^[ 12][1234567890]/s/^..    ...  ..:..:.....          //
g/^                              /s///
w
q
sort -n +3 temp/tfil >temp/table.xfr
```

Figure G-14. Command file TAB.XFR.

```
Oct  7 15:44 1981  trim.table Page 1


ed temp/table.acc
1,$g/     3    23  /s///
1,$g/ov.x /s///
1,$g/     ......./s///
w temp/slimtable.acc
q
```

Figure G-15. Command file TRIM.TABLE.

```
Oct 23 07:49 1981  wipeout Page 1


if $N = 2 goto start
echo "useage:  % sh sh/wipeout <mon/dd-hhmm>"
goto end
: start
echo removing $1 files
rm -f $1log.[xr]
rm -f $1ovh.[xr]
rm -f $1his.[xr]
rm -f $1data.r
rm -f $1ov.x
echo all $1 files for log, his, ovh, and data removed
: end
```

Figure G-16. Command file WIPEOUT.

FORM **NTIA-29**
(4-80)

U.S. DEPARTMENT OF COMMERCE
NAT'L. TELECOMMUNICATIONS AND INFORMATION ADMINISTRATION

BIBLIOGRAPHIC DATA SHEET

	1. PUBLICATION NO.	2. Gov't Accession No.	3. Recipient's Accession No.
	NTIA Report 82-112		

4. TITLE AND SUBTITLE

USER-ORIENTED PERFORMANCE MEASUREMENTS ON THE ARPANET: THE TESTING OF A PROPOSED FEDERAL STANDARD

5. Publication Date: November 1982

6. Performing Organization Code: NTIA/ITS

7. AUTHOR(S) D. R. Wortendyke, N. B. Seitz, K. P. Spies, E. L. Crow, and D. S. Grubb

9. Project/Task/Work Unit No.: 910 4120

8. PERFORMING ORGANIZATION NAME AND ADDRESS

U.S. Department of Commerce, National Telecommunications and Information Administration, Institute for Telecommunication Sciences, 325 Broadway, Boulder, CO 80303

10. Contract/Grant No.

11. Sponsoring Organization Name and Address

U.S. Department of Commerce, National Telecommunications and Information Administration, Institute for Telecommunication Sciences, 325 Broadway, Boulder, CO 80303

12. Type of Report and Period Covered: NTIA Report

13.

14. SUPPLEMENTARY NOTES

15. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

This report presents the results of a trial implementation of a newly developed data communication performance measurement methodology which has been proposed as Federal Standard 1043. In this experiment, a prototype data communication performance measurement system was developed in accordance with specifications defined in the standard. The system was used to assess the data communication service provided to a typical pair of ARPANET end users (host computer application programs). These user-oriented measurements differed from earlier ARPANET measurements in that the host computer operating systems and network control programs were regarded as providers of an end-to-end data communication service, rather than as users of the subnetwork.

(continued)

16. Key Words *(Alphabetical order, separated by semicolons)*

American national standard; ARPANET; computer networks; data communications; end user; federal standards; performance measurement

17. AVAILABILITY STATEMENT

☒ UNLIMITED.

☐ FOR OFFICIAL DISTRIBUTION.

18. Security Class. *(This report)*: Unclassified

20. Number of pages: 293

19. Security Class. *(This page)*: Unclassified

21. Price:

15. ABSTRACT (continued)

Results of the experiment will be useful in three ways. First, the prototype performance measurement system developed in this experiment will facilitate future implementations of the measurement standard. Second, the experience of implementing the measurement standard identified a number of ways in which that standard could be improved. These improvements will be incorporated in a future revision. Finally, the user-oriented performance values measured in this experiment will assist communication system planners in relating end-to-end performance objectives to the performance of subsystems.

☆U.S. GOVERNMENT PRINTING OFFICE:1983—677-279 / 77