NTIA Report 95-319 (3)

Performance Evaluation of Data Communication Services: NTIA Implementation of American National Standard X3.141
Volume 3. Data Extraction

Martin J. Miles
David R. Wortendyke

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

Larry Irving, Assistant Secretary
for Communications and Information

August 1995

CONTENTS

CONTENTS (Cont'd)

FIGURES

FIGURES (Cont'd)

TABLES

PERFORMANCE EVALUATION OF DATA COMMUNICATION SERVICES: NTIA IMPLEMENTATION OF AMERICAN NATIONAL STANDARD X3.141

VOLUME 3: DATA EXTRACTION

Martin J. Miles and David R. Wortendyke[1]

The six volumes of this report are:

Volume 1. Overview
Volume 2. Experiment Design
Volume 3. Data Extraction
Volume 4. Data Reduction
Volume 5. Data Analysis
Volume 6. Data Display

This volume explains how to conduct a data communication session. Specifically, it explains how to determine the commands and expected responses of a protocol (for access and disengagement functions), how to determine the responsibility of the participating entities for producing each reference event, and how to draw a profile of the session (which demonstrates the flow of information between the participating entities and across user/system interfaces). It explains how to create a file containing the commands and expected responses of the protocol, the code that causes the times at which they cross interfaces to be recorded, and a code number that indicates the state of the entities at each interface. This volume also explains how to modify the transmitting program to agree with the protocol. It explains how to create files that support the on-line data extraction software. Specifically, these files are the end user identification files, the clock calibration file, and the protocol file. This volume then explains how to execute a shell script that conducts a test, and how to execute a shell script that processes the test data.

Key words: access; communication state codes; disengagement; reference events; protocol; satellite clock receiver; session profile; user information transfer; user/system interfaces

1. INTRODUCTION

The extraction of information from a data communication system requires a set of hardware and software to access and disengage, terminate connections,

[1]The authors are with the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, Boulder, CO 80303.

transmit and receive user information, and record system-independent interface events (called reference events) at the user/system interfaces. The data extraction software is written in the C programming language. Figure 1 is a schematic diagram of a data communication system, the participating entities, interfaces, and interface events[2].

Figure 2 is a structured design diagram that describes the data extraction procedure for the experimenter. The activities shown in this diagram correspond to the sections of this volume.

Section 2 shows how the commands and expected responses (i.e., the protocol) for the access and disengagement functions can be determined. A schematic protocol that could serve as a template for others is also discussed.

Section 3 shows that it is necessary to record which participating entities are responsible for each reference event.

Section 4 shows how to assign a state code to each entity after each reference event. This code indicates the communication state of the entity and its responsibility for producing the next reference event at each interface.

Section 5 shows how to draw a profile of the data communication session. The session profile shows the flow of information among the participating entity-

---

[2]Sessions can involve either two-way (duplex) transmission or multiple pairs if the on-line data extraction procedures are modified:

- Each session is treated as the superposition of two simplex sessions.
- A given user acts as the source in one session and as the destination in the other.
- Each interface monitor generates one set of extracted data files in which the local user acts as the source and another set in which the local user acts as the destination.
- These sets of extracted data are input to separate data conversion runs to produce two performance data batches - one for each direction of transmission.
- The two data batches are then reduced and analyzed in the usual manner.

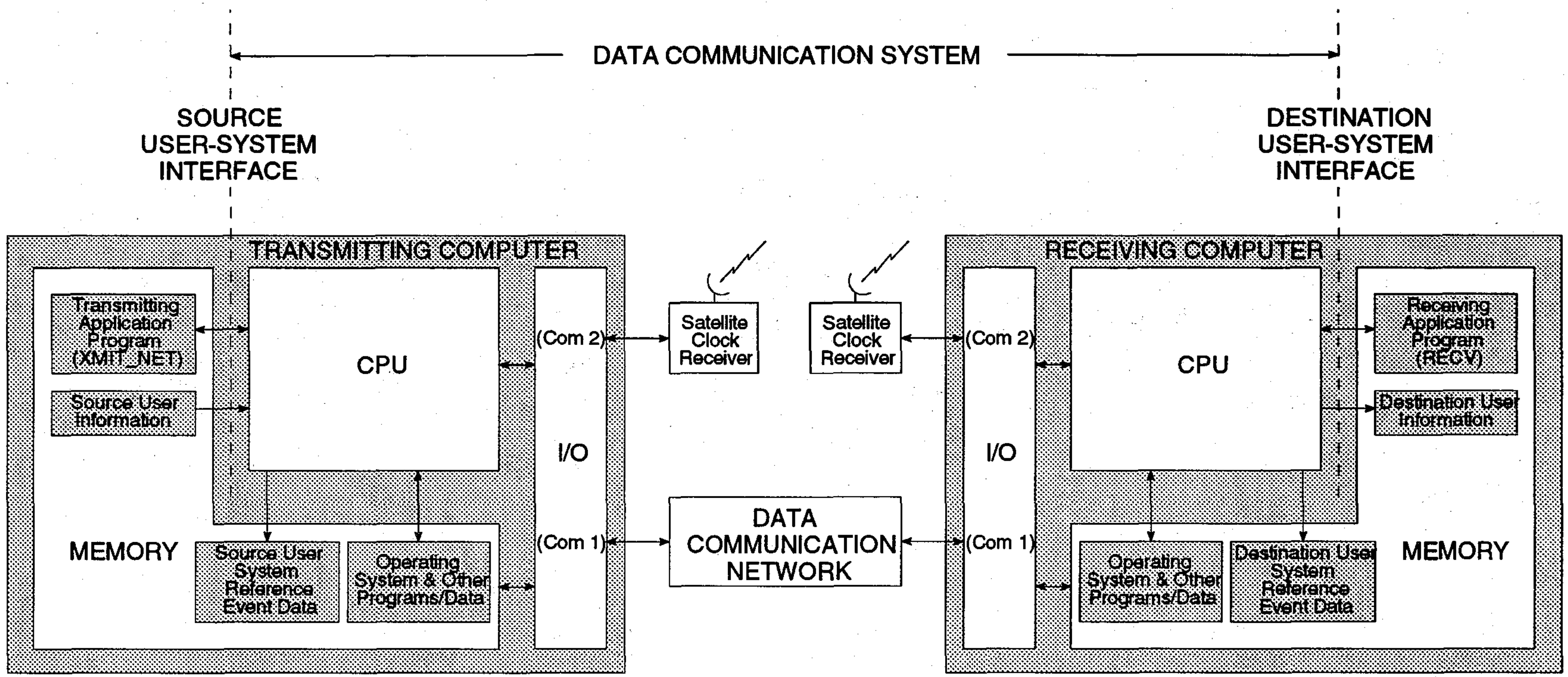

Figure 1. Schematic diagram of a data communication system and end users.

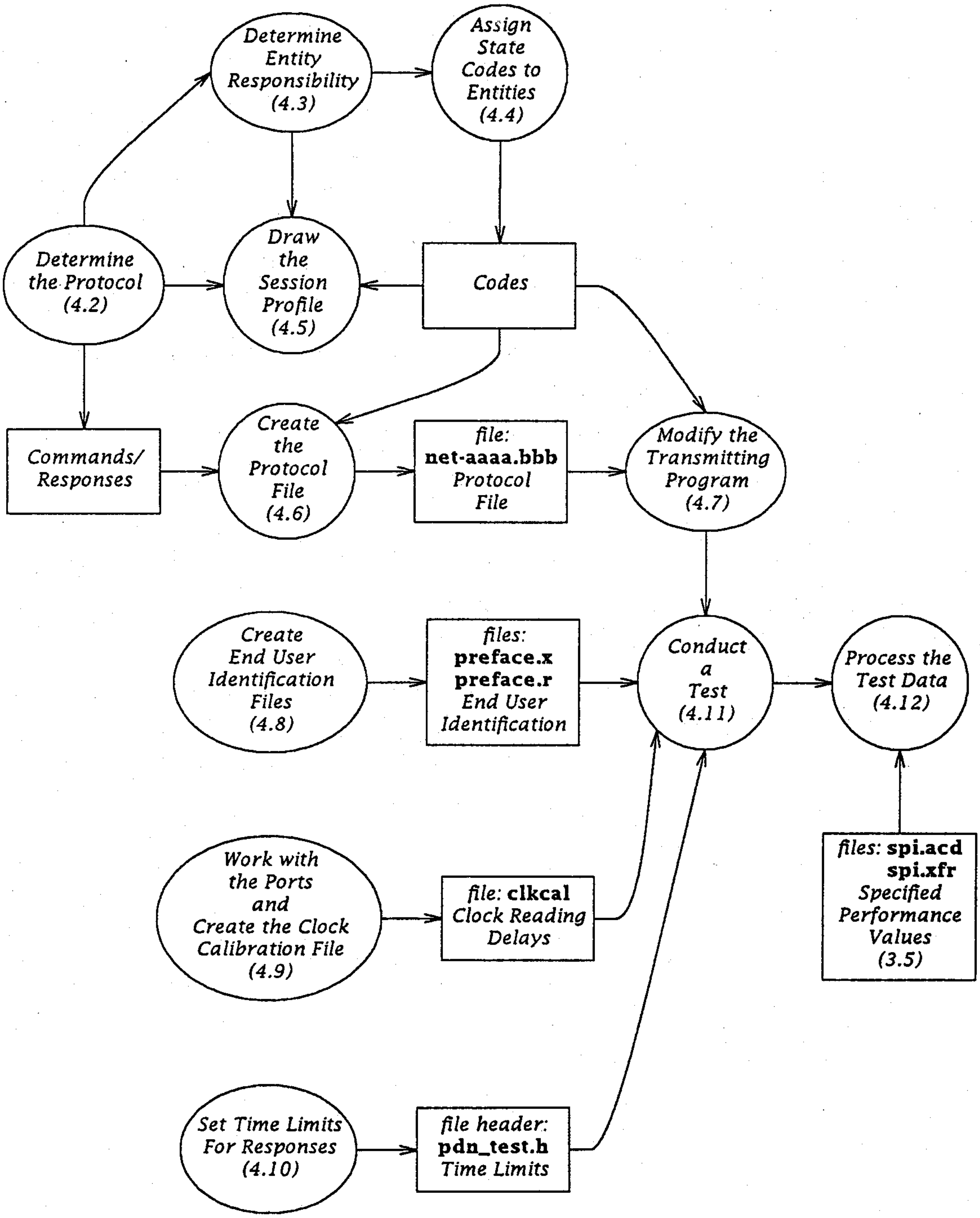

Figure 2. Structured design diagram of the operator procedures in data extraction.

interfaces, lists the entity state codes, and shows how the time of occurrence of reference events are to be recorded. The session profile that matches the schematic protocol is also shown.

After the session profile is drawn, the protocol file that matches it must be created. Section 6 shows the schematic protocol file and a sample protocol file for a direct dial network. It then shows how to create a protocol file.

Section 7 discusses the transmitting program that must also match both the session profile and the protocol file. It describes its function and provides hints for modifying it for other protocols.

Section 8 shows how to create files that identify each end user, define the type of session, and define the type of disengagement.

Section 9 shows how to link and set the communication ports, and how to calibrate the satellite clock receivers.

Section 10 shows how to set the time limits for responses; these limits are related to the performance values that are specified as part of the design of the experiment (Volume 2).

Section 11 shows how to conduct a test (called on-line processing). Before the experiment begins, it is necessary to synchronize the UNIX[tm] clock with the satellite clock receiver.

Then a test can be conducted by simply typing (at the source computer), a command such as

runxt o <network > <opt 1> <opt 2>

for access-disengagement tests, and

runxt u <network > <opt 1> <opt 2>

for user information transfer tests. The options (i.e., opt 1 and opt 2) refer to levels of variable conditions that are determined in Section 4.2 of Volume 2.

Section 12 shows how to process the test data (called off-line processing). After the test is completed, the data files must be consolidated in one computer, and the data processed by typing a command such as

do nnnn

where **nnnn** is the test number. This single command activates a set of command files and UNIX[tm] utilities that

- convert the data to text files,

- reduce the data to performance data (i.e., times and failures) (Volume 4),
- analyze the data (Volume 5), and
- create files for various graphs of primary time parameters (Volume 6).

The on-line software is described in Appendix A, and the off-line software is described in Appendix B. The appendices contain structured design diagrams that show the relationship among programs, shell scripts, and other files.

The procedure described in this volume is augmented by a report of the NTIA experiments (Spies et al., 1988) and a text on the C programming language (Kernighan and Richie, 1978).

2. DETERMINE THE PROTOCOL

A protocol is a set of rules governing the interaction between pairs of communicators. For our purposes, a protocol is a sequence of commands and expected responses required to complete the functions of access and disengagement. The protocol can be determined from protocol analyzers, manual operation, user's manuals, and experts.

2.1 Protocol Analyzer

Following is an example of using a protocol analyzer:

- Connect a protocol analyzer to the remote computer communication port.

- Call the remote computer (which is identified by a connect line) by typing

  cu -scccc -l/dev/coml

  where **cccc** is the transmission rate (baud) as set by the super user, **l** (i.e., "el") indicates the connect line, and **/dev/coml** specifies the device at the first communication port.

- Log in to the network.

- Connect to the remote computer.

- Issue any command (e.g., **sys**).

- Log out of the computer.

- Log out of the network (if necessary).

- Disconnect from **cu** by typing the following two characters

2.2 Manual Operation

If a protocol analyzer is not used, simply record all keystrokes and screen responses provided by the port when accessing the system, logging into it, and disconnecting from it.

2.3 Schematic Protocol

Reference events are those interface events that are system-independent. Primary reference events are reference events that define primary performance parameters (and are given generic names, such as Access Request). A schematic protocol containing five nonspecified commands and their nonspecified expected responses is discussed here. It will be used to demonstrate a session profile and its corresponding protocol file. Table 1 lists the primary data communication states, primary reference events, and symbols for selected commands from the source end user (i.e., C_1, ..., C_5) and their expected responses (i.e., R_1, ..., R_5). The primary reference events that result from blocking are not identified by a symbol.

Table 1. Commands and Expected Responses and Their Corresponding Primary Reference Events

PRIMARY DATA COMMUNICATION STATES/FUNCTIONS	PRIMARY REFERENCE EVENTS	COMMANDS/ EXPECTED RESPONSES
(Initial) Idle	-	C_1
	-	R_1
Access	Access Request	C_2
	-	R_2
	Nonoriginating User Commitment	C_3
	System Blocking Signal	-
	User Blocking Signal	-
	-	R_3
	Access Confirmation	(read Block j)*
User Information Transfer	Start of Block Transfer	(read Block j)*
	End of Block Transfer	(ready)
Disengagement	Source Disengagement Request	C_4
	Source User Disengagement Blocking Signal	-
	Destination Disengagement Request	(ready)
	Destination User Disengagement Blocking Signal	-
	Destination Disengagement Confirmation	R_4
	-	C_5
	Source Disengagement Confirmation	R_5
(Final) Idle	-	-

*The number of blocks to be transferred is j=1,..., N.

3. DETERMINE ENTITY RESPONSIBILITY

To define the sequence of events during a data communication session, it is necessary to identify, after each reference event, the entity (or entities) responsible for producing the next reference event.

In the NTIA implementation of ANS X3.141, the participating entities are the source end user (the application program, **xmit_net**), the system, and the destination end user (the application program, **recv**). The three entities define two interfaces: a source user-system interface and a destination user-system interface. Arbitrarily call one interface the local interface, and call the other interface the remote interface. Then there are four entity-interface combinations:

- local user at the local interface
- system at the local interface
- system at the remote interface
- remote user at the remote interface.

For any entity at either interface, there are two states of responsibility for producing the next reference event: either the entity is responsible or it is not. Therefore, there may be 16 combinations of entity-interface responsibilities:

$$(\text{responsibility states})^{(\text{entity-interface combinations})} = 2^4 = 16.$$

However, we can make two logical observations that reduce the number of combinations to six. The first observation is that the remote user cannot be responsible for the next reference event.[3] Hence, there are three remaining entity-interface combinations:

- local user at the local interface
- system at the local interface
- system at the remote interface.

[3]The remote user cannot be responsible for producing the next reference event because a reference event must occur at the remote interface before the remote user is (can be) responsible for producing a reference event.

Now there are eight possible combinations of entity-interface responsibilities:

$$(\text{responsibility states})^{(\text{entity-interface combinations})} = 2^3 = 8.$$

Table 2 lists the three entity-interface pairs and all possible combinations of responsibility for them. The words "yes" and "no" indicate responsibility and non-responsibility, respectively. The second observation is that the seventh and eighth combinations cannot exist (as indicated by the lack of shading) because the user and the system cannot both be responsible for the next reference event at an interface.

Since the remaining six combinations are not illogical, it is conceivable that some protocol permits them.

Table 2. Combinations of Responsibilities for the Three Entity-Interface Pairs

	LOCAL USER RESPONSIBLE AT LOCAL INTERFACE	SYSTEM RESPONSIBLE AT LOCAL INTERFACE	SYSTEM RESPONSIBLE AT REMOTE INTERFACE
1	No	No	No
2	No	No	Yes
3	No	Yes	No
4	Yes	No	No
5	No	Yes	Yes
6	Yes	No	Yes
7	Yes	Yes	No
8	Yes	Yes	Yes

4. ASSIGN STATE CODES TO ENTITIES

NTIA software requires each entity-interface pair to have a code number that identifies, following each reference event, its primary state (when knowable) and its secondary state.

4.1 Primary States

An entity can exist in one of three primary states: Idle (before and after the session), Access-User Information Transfer, and Disengagement. (Because User Information Transfer does not require a protocol, it is combined with Access for the purpose of assigning state codes.) Following are the conditions under which entities exist in the primary states:

- (Initial) Idle State. An entity is in the Idle State if it is not participating in the communication session. If the system's performance time is within the service time, an entity is responsible for the next reference event, otherwise it is not responsible.

- Access-User Information Transfer State. An entity is in the Access-User Information Transfer State if it is involved in the communication session with the intent to transfer user information. In this state, an entity may or may not be responsible for the next reference event.

- Disengagement State. An entity is in the Disengagement State if it is involved in the communication session with the intent to terminate involvement without transferring additional user information.

- (Final) Idle State. As stated above, an entity is in the Idle state if it is not participating in the communication session. An entity is returned to the Idle State following the Disengagement Confirmation reference event.

4.2 Secondary States

Within each primary state, an entity has a secondary state. This state is the state of responsibility for producing the next reference event at a given interface.

Table 3 lists the entity-interface state codes. The six possible entity-interface states at the local interface are assigned the numbers 0-5. However, the entity-interface states of the system at the remote interface are assigned the numbers 0-1 (indicating the secondary state only) because unpredictable transmission delays render unknowable the primary state of the system at that interface.

These state codes will be used in the session profile (Section 5) and in the protocol file (Section 6).

Table 3. Entity-Interface State Codes

STATE		LOCAL USER RESPONSIBILITY AT LOCAL INTERFACE	SYSTEM RESPONSIBILITY AT LOCAL INTERFACE	SYSTEM RESPONSIBILITY AT REMOTE INTERFACE
Primary	Secondary			
Idle	No	0	0	0
	Yes	1	1	1
Access-UIT	No	2	2	0
	Yes	3	3	1
Disengage.	No	4	4	0
	Yes	5	5	1

5. DRAW THE SESSION PROFILE

A session profile is a diagram of the flow of information among the connected entities during a data communication session. A session profile could contain the following elements:

- Rectangles. Rectangles contain commands and expected responses from the entities.
- Directed Line. Directed line segments connect rectangles and indicate the flow of information. When a line segment crosses a user-system interface, an interface event has occurred.
- Entity-Interface State Codes. These three code numbers indicate the state of each of the three entities (about each interface) following the preceding reference event. The order of the three codes at each interface is as follows:
    - Source User-System Interface. At the source user-system interface, the order, from left to right, is
        - source user at source interface,
        - system at source interface, and
        - system at destination interface.
    - Destination User-System Interface. At the destination user-system interface, the order, from left to right is
        - system at source interface,
        - system at destination interface, and
        - destination user at destination interface.
- Primary Reference Events. These events may be shown at the interfaces.

The data communication session begins when the source user issues the first command while the communication system is in the idle state. Then, from each command/response rectangle, one or two line segments extend to the next command/response rectangle(s) (depending upon the responsibilities whose possibilities are listed in Table 2). This sequence continues until the system returns to the Idle state. Figure 3 is the schematic session profile

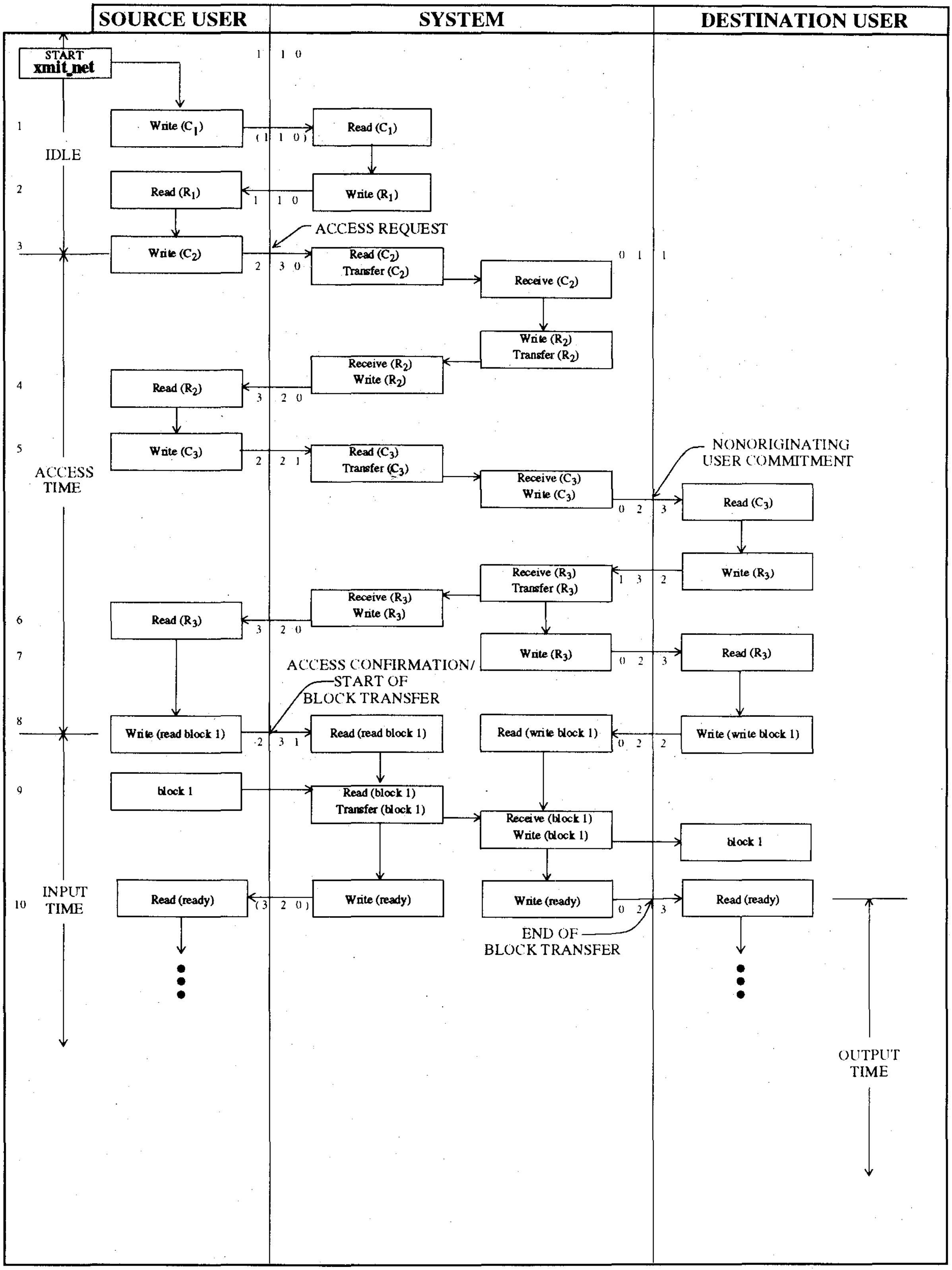

Figure 3. Session profile for the schematic protocol.

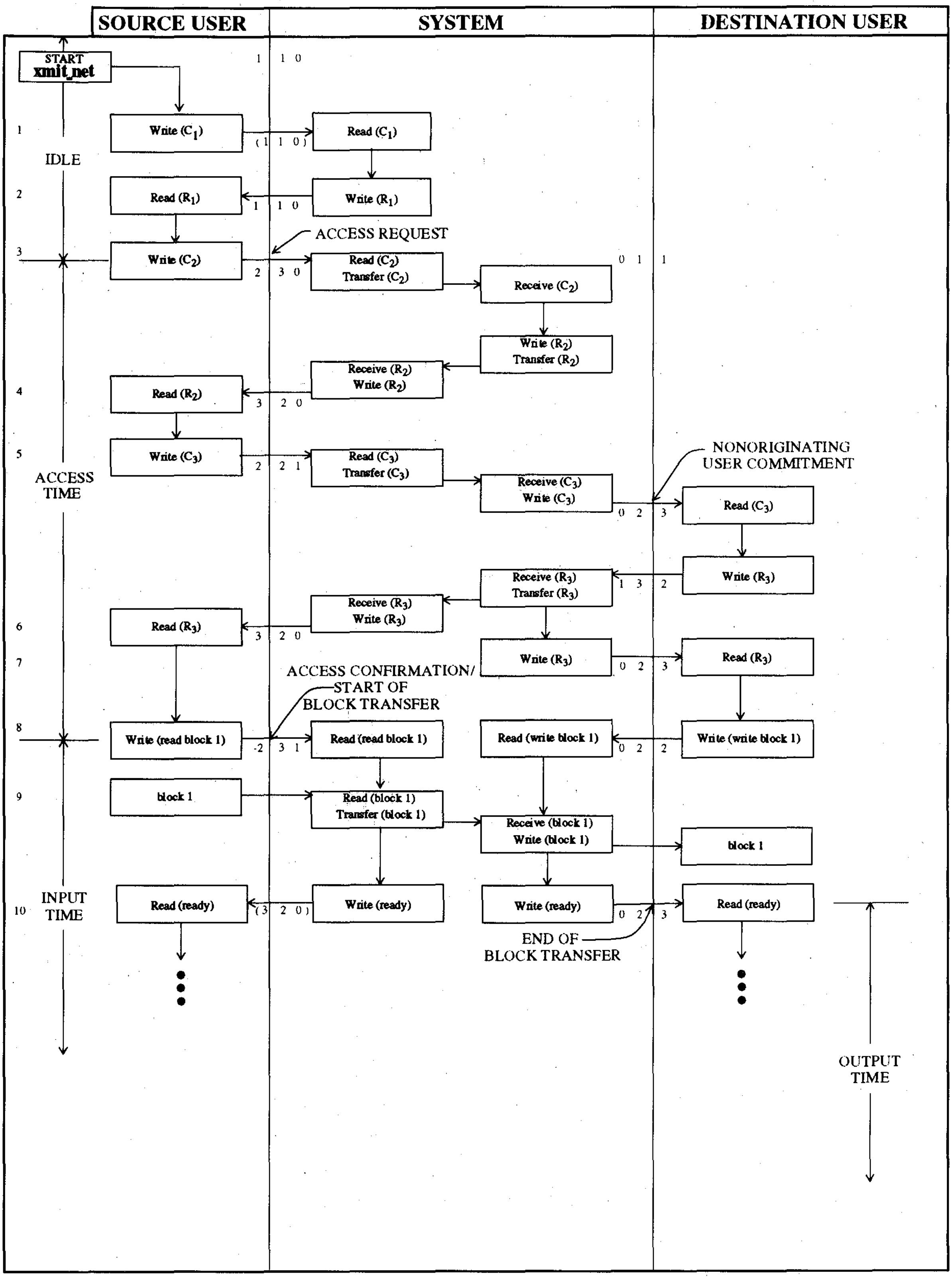

Figure 3. (cont'd). Session profile for the schematic protocol.

corresponding to the command and expected responses listed in Table 1. Along with the activities, it shows the primary reference events (except for blocking events), the state codes of the three entities about each interface, the duration of the communication states, and the Input Time and Output Time (which appear equal in duration only because this is a schematic diagram).

Before discussing the schematic session profile further, the entities and their operations are described.

A. Active Peer Entities

The participating entities are the two end users and the system. End users are active in one sense: they provide logic (i.e., application program instructions in RAM cause the system to perform operations). Systems are active in two senses: they provide logic (i.e., operating system instructions in ROM cause the system to perform operations), and they provide the (electric and magnetic) means to execute instructions from both the end user application programs and their operating systems. Because logic is provided by both entities, <u>the schematic session profile portrays both the end users and the system as active peer entities</u>.

B. Entity Operations

Four types of entity operations are specified in the schematic session profile: read, write, transfer, and receive. Each operation has an operand that is either an instruction or user information. The read and write operations represent user-system interface events as operands that are passed between each end user and its proximal portion of the system. However, the transfer and receive operations do not create user-system interface events; they represent activities within the system.

A data communication session may require thousands of instructions to be read or written. Moreover, several steps (i.e., interface events) are required to complete each of them.[4] However, the schematic session profile shows only

---

[4]As CPUs are currently designed, each instruction must be fetched (from program memory), decoded (by the instruction decoding unit), and executed (by the timing and control unit). Fetching requires a few program memory-CPU interface events, and execution requires a few data memory-CPU interface events (if the instruction has a data operand).

those user-system interface events that are necessary to evaluate performance (except blocking events).

Since the schematic session profile is intended to be expository, it states the fact that all instructions are read or written - even if the instruction, itself, causes the system to read or write. For example, the symbolic instruction, "read block 1," which causes the system to read block 1, must first be written by the end user to the system. Hence, this write operation has the instruction, "read block 1," as its operand, and it is listed in the schematic session profile as

Write (read block 1).

5.1 Initial Idle State

5.1.1 Source User-System Interface

The communication system is initially in the Idle state. After the application program **xmit_net** is started, it can issue zero or more commands prior to the Access Request (all commands denoted by C_1) and receive corresponding responses (all responses denoted by R_1).

5.1.2 Destination User-System Interface

There is no activity at the destination user-system interface during the initial idle state.

5.2 Access State

5.2.1 Source User-System Interface

The Access State begins when **xmit_net** attempts to access the remote computer by writing the command, C_2. In connection-oriented sessions, this command is the primary reference event, Access Request. After the response, R_2, is read at the source user-system interface, the source user invokes the destination user by writing the command, C_3. The application program at the destination site is called **recv**. The source user then reads the response, R_3.

The primary reference event, End of Access, occurs when the source user writes the command to read the first block of user information.

5.2.2 Destination User-System Interface

When the command, C_3, is detected at the destination user-system interface, control is transferred from the source computer operating system to the application program **recv**. The destination user is committed to participate in the session; this is the primary reference event, Nonoriginating User Commitment.

In connection-oriented sessions, user information is entered only after this event is confirmed at the source user-system interface (i.e., after R_3 is detected).

In connectionless sessions (e.g., message-switched and datagram services), user information can be entered before the nonoriginating user is committed.

The destination user writes, R_3, to the system. The system writes a response to the destination user.

The next (and last) reference event at this interface during the Access State is caused when the destination user writes a command to the system to write the first block of user information to memory.

5.3 User Information Transfer State

5.3.1 Source User-System Interface

The source user writes the command to the system to read block 1. The User Information Transfer State begins with the Start of Block Transfer for the first user information block. The system reads block 1 and transfers it to the destination portion of the system. The system writes "ready" to the source user. This sequence of "write" commands and their responses is repeated for each of the N user information blocks that is transferred.

5.3.2 Destination User-System Interface

Receipt of the block at the destination user-system interface is the primary reference event, End of Block Transfer. It is received when the system indicates that the block has been written; the destination portion of the system writes "ready" to the destination user. This sequence of commands and responses is repeated for each user information block that is received.

The state codes at the destination interface are 22 only for the first block because the destination interface does not know if the first block has been written. The state codes alternate as 23, 32, ... after the first block.

5.4 Disengagement State

5.4.1 Source User-System Interface

The Source Disengagement State begins when the source user writes an end-of-text character, C_4. This is the primary reference event, Source Disengagement Request. After it reads the UNIX[tm] prompt, R_4, it writes a command, C_5, to disengage. The response, R_5, from the source portion of the system is the primary reference event, Source Disengagement Confirmation; it marks the end of Source Disengagement Time.

5.4.2 Destination User-System Interface

Destination Disengagement Time begins when the destination portion of the system writes "ready" to the destination user. Destination Disengagement Time ends when the end of the **recv** program is reached. This is the primary reference event, Destination Disengagement Confirmation.

5.5 Final Idle State

5.5.1 Source User-System Interface

After Source Disengagement Confirmation, R_5, is read, the source portion of the system is returned to the Idle State.

5.5.2 Destination User-System Interface

After the end of the program is detected (Destination Disengagement Confirmation), the destination portion of the system is returned to the Idle State.[5]

Draw the session profile.

[5]Throughout this six-volume report, action required of the experimenter is described in a shaded block.

6. CREATE THE PROTOCOL FILE

Protocol files are read by the source end user (the application program, **xmit_net**). One protocol file exists for each network (and also for each site that is accessed by a telephone number), and it must agree with the session profile at the source user-system interface.[6]

This section describes the contents of a protocol file, describes both the schematic protocol file and a sample protocol file for a direct-dial network, and shows how to create a protocol file.

6.1 Contents of a Protocol File

A protocol file is a text file that contains three types of lines:

- Nonexecutable Comment Lines. They have a # in column one and can be placed anywhere.
- Executable Command/Response Lines. They contain commands on the left side, the (expected) response on the right, and the Δ|Δ character sequence between them. The Δ character represents either the blank character or a tab.
- Executable Time Stamp and Entity State Code Lines. These lines cause the time to be recorded. They have an * in column one. The * is followed by three digits, the Δ|Δ character sequence, and three more digits.
  - The left-hand three digits are the entity state codes of the three entity-interface responsibilities about the source interface following the previous response.
  - The right-hand three digits are the entity state codes of the three entity-interface responsibilities about the source interface following the next command.

6.2 Schematic Protocol File

Figure 4 is the schematic protocol file. It shows the sequence of commands, expected responses, and entity state codes at the source user-system

[6]A network can be defined by a unique combination of pairs of end users, telephone numbers (e. g., numbers of the public data network at each destination site), bauds, window sizes, etc. If there is more than one protocol file, the proper file will be selected by the network argument of the **runxt** shell script.

interface as shown in the schematic session profile. The codes in parentheses in the session profile are not listed in the protocol file, but are embedded in **xmit_net**. The minus sign preceding the state codes 231 indicates the end of the access state.

```
      C1            Δ|Δ        R1
*     110           Δ|Δ       230
#     Beginning of access
      C2            Δ|Δ        R2
*     320           Δ|Δ       221
 C3  = recv\r       Δ|Δ R3  = READY
*     320           Δ|Δ       -231
#     End of access - Beginning of user information transfer
#     End of user information transfer - Beginning of disengagement
 C4  = \e           Δ|Δ R4  = %\s
*     540           Δ|Δ       450
      C5            Δ|Δ        R5
#     End of disengagement
```

Figure 4. Schematic protocol file.

Figure 5 relates the schematic session profile and the schematic protocol file to each other:

- <u>Figure 3</u>. Figure 3 is a schematic session profile that shows time increasing down the user-system interfaces, and the interface events are numbered along the left margin. Figure 5 shows the source-user interface as an undulating curve and those interface event numbers are encircled.

- <u>Figure 4</u>. Figure 4 is a schematic protocol file. Figure 5 shows its executable code and state codes within the two rectangles.

6.3 Sample Protocol File

Figure 6 is a sample protocol file for a direct-dial network. In this case, two computers are connected to a public data network by modems. Each executable line is numbered; this number is not part of the file, but is for reference only. The commands are listed on the left, and the last few characters of the expected responses are listed on the right. These characters are unique among all possible expected response character strings. This protocol file contains no unnecessary time stamps. During a test, each command will be written to the system, and the response will be compared with the expected response from

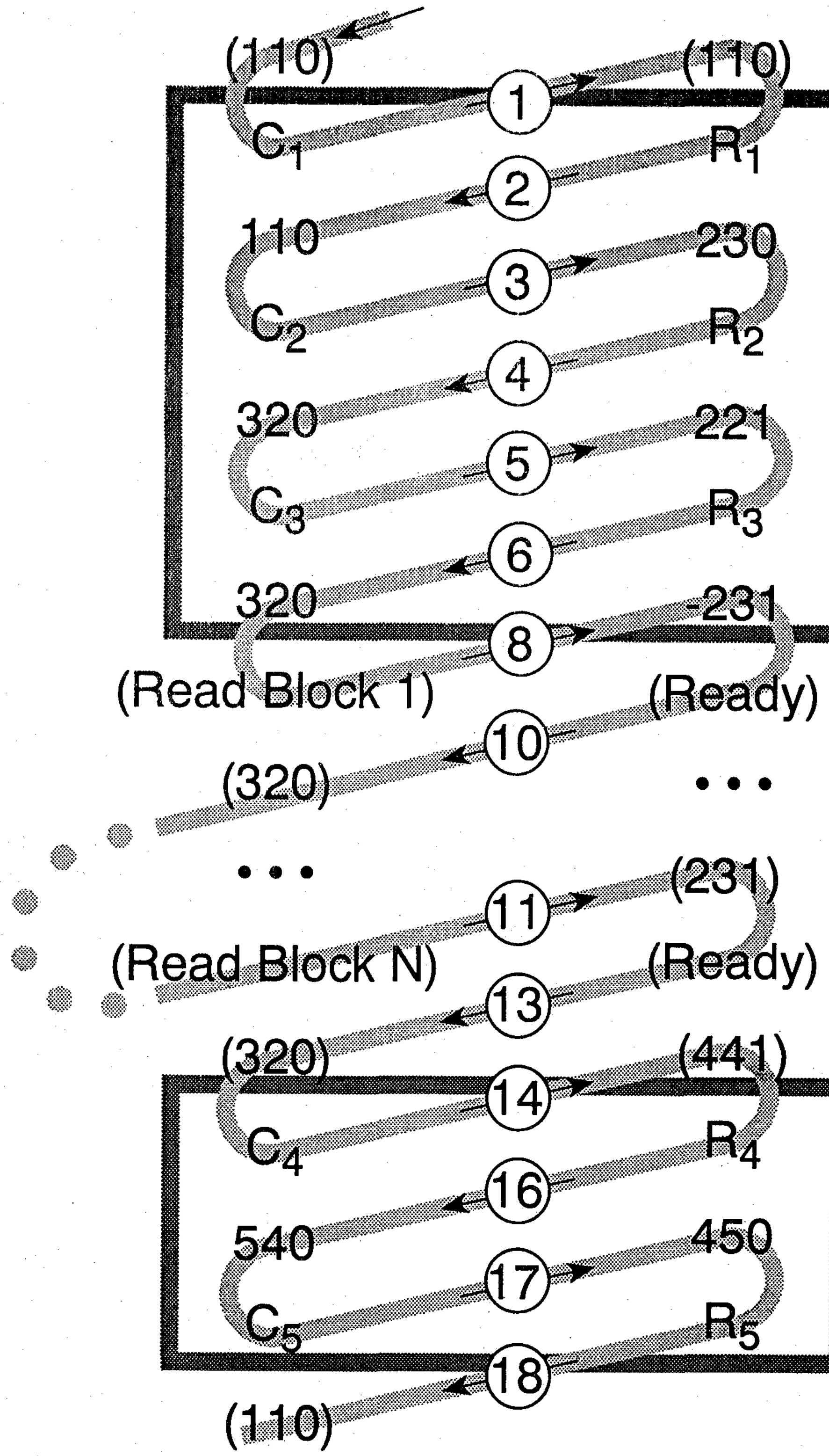

Figure 5. Sequence of commands, expected responses, and state codes at the source user-system interface of the schematic session profile.

```
# Protocol File for Direct Dial From Laramie to Boulder
AT&V\r                  Δ|Δ     OK
* 110                   Δ|Δ     230
ATDT9,303-497-2134\r    Δ|Δ     00
\r\d\d                  Δ|Δ     ogin:
net\r                   Δ|Δ     ord:
test\r                  Δ|Δ     %\s
* 320                   Δ|Δ     221
recv\r                  Δ|Δ     READY
* 320                   Δ|Δ     -231
# logout sequence
\e                      Δ|Δ     %\s
* 540                   Δ|Δ     450
logout\r                Δ|Δ     RRIER
```

Figure 6. Protocol file for a direct dial network.

the protocol file. Each command and response will be discussed in the order they occur in the protocol file.

6.3.1 Preliminary Activities

As stated in Section 5.1, any command/response lines prior to the first time stamp

(i.e., * 110 | 230)

represent activities that occur prior to the Access State. For example, the first executable line in the sample protocol file contains the command

AT&V\r[7]

---

[7]The \ followed by a character is an escape sequence that is considered to be a single character (not two). The following escape sequences may be used in the protocol file:

\r : carriage return.
\d : delay the command for 2 seconds.
\s : provide a space.
\e : end of text.

which requests a listing of parameter settings from a modem. Since this command is not needed to access the destination site, it is placed before the first time stamp. The expected response for this command is

OK.[8]

6.3.2 Access

The second executable line is the mandatory time stamp mentioned above. The third, fourth, fifth, and sixth lines contain the following command/response pairs:

ATDT,303-497-2134\r	Δ|Δ	**00**
\r\d\d	Δ|Δ	**ogin:**
net\r	Δ|Δ	**ord:**
test\r	Δ|Δ	**%\s**[9]

A more complete expected response to the fourth line would be

login:

and a more complete expected response to the fifth line would be

password:.

However, the last few unique characters of these strings are sufficient. The expected response to line six is the UNIX[tm] prompt (i.e., %), followed by a space

---

[8]Three sets of time stamps have been incorporated in the transmitting application program xmit_net. The remainder must be in the protocol file. The three sets incorporated in the transmitting program are:

- Initial set-up time stamps with the state codes 110 and 110.
- User Information Transfer time stamps which follow the negative state code at the end of Access. The entity state codes are 231 and 320 for each block transferred and 441 for Source Disengagement Request.
- End of Disengagement time stamp with entity state codes 110.

It is assumed that the first line in the protocol does not start the Access function, but is placed there to set parameters in a modem or packet assembler/disassembler (PAD). The access function should not be started until after encountering time stamps and a set of entity state codes such as 110 and 230, respectively.

[9]**%** is the operating system prompt established in the C shell file, **.cshrc**..

(i.e., **\s**). The next command/response line (i.e., line 8) is mandatory. It invokes the destination user (i.e., the application program **recv**) which responds with the string **READY**. The end of Access is indicated by the negative state codes, -231.

6.3.3 User Information Transfer

The ninth line, mentioned above as marking the end of Access, is a function delimiter line. It also serves to start the User Information Transfer function. User Information Transfer ends with the eleventh executable line:

\e | %\s

The **\e** (i.e., end of text) character is sent to the destination site, ending the User Information Transfer function. It also causes the primary reference event, Source Disengagement Request.

6.3.4 Disengagement

The twelfth executable line is the time stamp line. The thirteenth executable line contains

logout\r | RRIER

where the command **logout** is followed by a carriage return (i.e., **\r**) and the string **RRIER** is the last few characters of the expected response to logging out of the source computer: **NO CARRIER** is returned by the local modem.

6.4 Create a Protocol File for Each Network

Protocol files are contained in directory **usr/net/proto**. They are named **net-aaaa.bbb** where **aaaa** identifies the network and **bbb** identifies the source site. (A unique identification is required if the source site is accessed by a telephone number.) Both the network **aaaa** and the source site **bbb** must be defined in **netcodes** (as described in Section 4.3.3 of Volume 2).

Each command and its expected response are listed on a line. The command is listed first (i.e., a set of contiguous characters). This is followed by the character sequence Δ|Δ (where Δ is the blank character or tab), and some or all of the expected response. (A set of unique contiguous characters – usually only the last three, four, or five.)

Command/response lines cannot exceed 75 characters unless **MAXLINE** is reset. **MAXLINE** is in the header file **pdn_test.h** which is located in directory **usr/net/src/d3a**. Additionally, there can be no more than 75 lines unless **MAXCMD** is reset in **pdn_test.h**.

Create a protocol file for each network and source site (if a telephone number is required).

7. MODIFY THE TRANSMITTING PROGRAM

The transmitting program, **xmit_net**, is a C program that is located in the directory **/usr/net/src/d3a**. It reads the protocol files and must agree with them. Specifically, **xmit_net** contains a sequence of **status** statements. They have the following form:

status = net_access(city, net, PHASE, &fd_netin, &fd_netout);

where **net_access** is a C function (contained in the file **connect.c**), and **PHASE** is a dummy argument representing one of the four phases: **SET_UP**, **REWIND**, **CONTINUE**, and **CLEAN_UP**.

When a **status** statement is executed, control passes from **xmit_net** to **net_access**. Control then passes from **net_access** to other subprograms, depending upon the value of the **PHASE** argument:

- PHASE = SET_UP. When **PHASE** = **SET_UP**, the protocol file is interpreted by subprogram **parsecmd** and stored in an array called **Dialogue**. This array contains alternating lines of commands and expected responses as well as lines having an * in column one.

- PHASE = REWIND. When **PHASE** = **REWIND**, the global variable **Response** is set to **0** so that **Dialogue** can be read from the beginning.

- PHASE = CONTINUE. When **PHASE** = **CONTINUE**, subprogram **converse** reads a command and its response from the **Dialogue** array.

- PHASE = CLEAN_UP. Finally, when **PHASE** = **CLEAN_UP**, the file is closed.

Figure 7 is a Venn diagram that shows the calling relationship among several important subprograms of **xmit_net**. Figure 8 is a structured design diagram showing the procedures of **xmit_net** during a test.

Modify xmit_net to match the protocol.

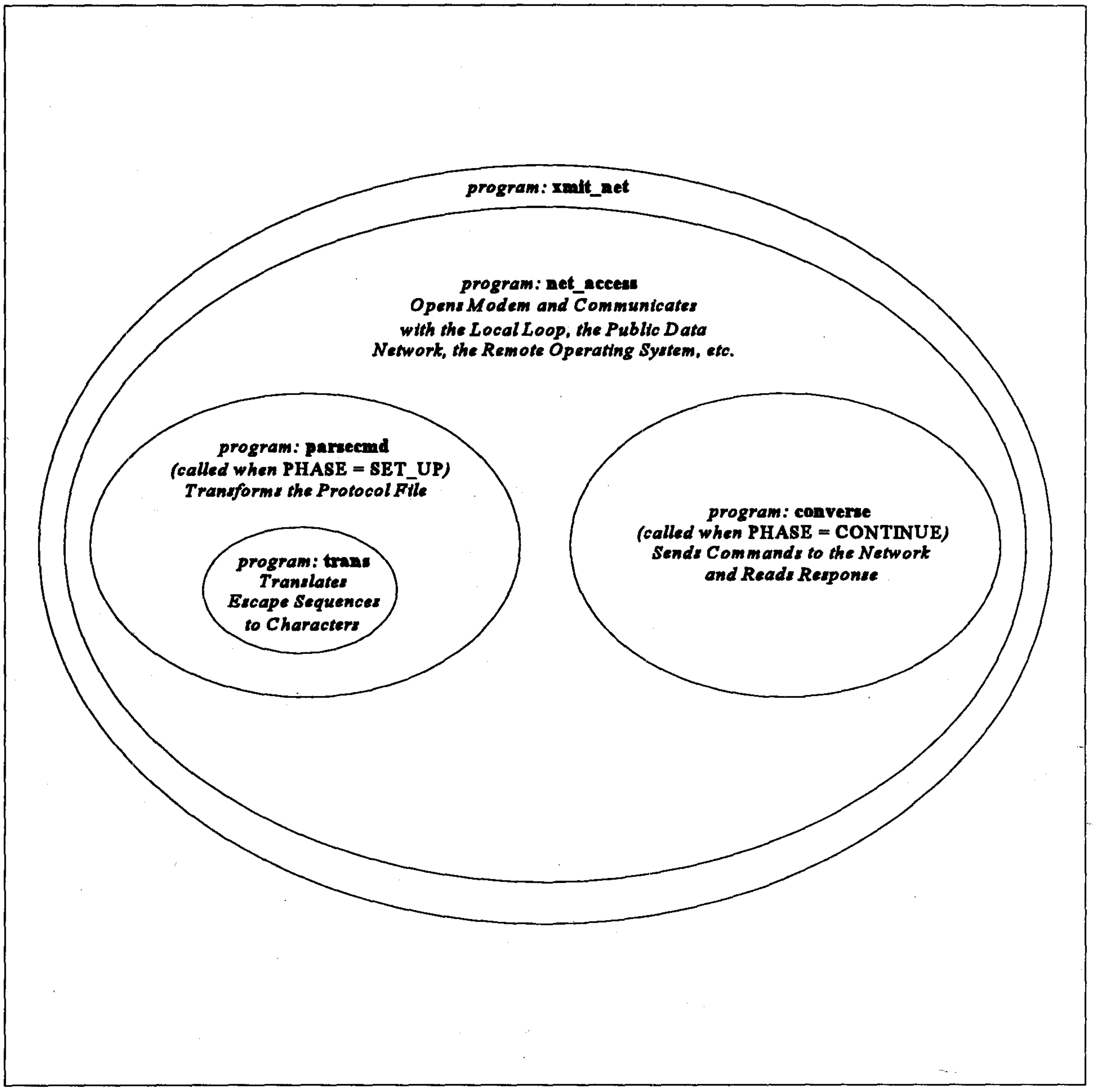

Figure 7. Venn diagram of the major subprograms of xmit_net.

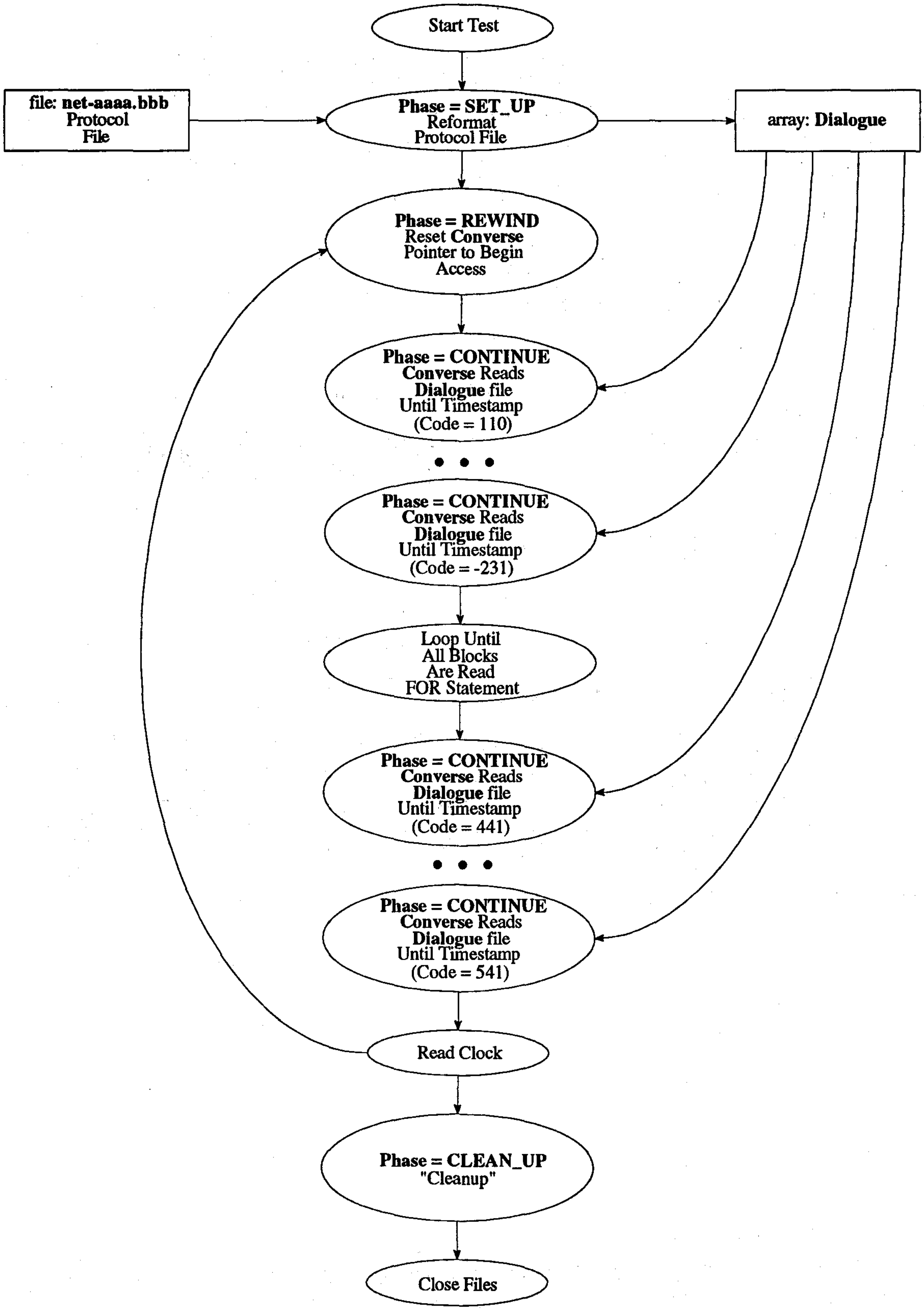

Figure 8. Structured design diagram of **xmit_net**.

8. CREATE THE END USER IDENTIFICATION FILES

There is an end user identification file for each end user. Each end user must log in to the home directory of **net**, and create the file. They will be read by subroutine **preface** in the C program **access**.

8.1 Source End User File

This file is called **preface.x**. The seven lines of identification are as follows:

1. The first line identifies the experimenter.
2. The second line is the four-digit test number **nnnn** which is assigned by the shell script **runumb** (it must be 1000 or greater).
3. The third line is **Source**.
4. The fourth identifies the type of test.[10]
5. The fifth line is the name of the source site computer.
6. The sixth line is the name of the destination site computer.
7. The seventh line contains two digits. The first digit is either 1 or 2, depending upon whether the session is connectionless or connection-oriented, respectively. The second digit is either 1 or 2, depending upon whether the disengagement is independent or negotiated, respectively.

Figure 9 is an example of this file.

---

[10]There are two types of tests: user information transfer tests and access-disengagement tests. However, the word User always appears here. The type of test is also entered as an argument when the shell script **runxt** is invoked; it is this argument that determines that the data extraction software performs correctly - not the fourth line of the **preface.x** file.

```
NTIA-ITS (Boulder)
2260
Source
User
NTIA - crestone
NTIA - eldiente
22
```

Figure 9. Example of a source end user identification file.

Create the source identification file, **preface.x**, and store it in the destination computer.

8.2 Destination End User File

This file is called **preface.r**. The seven lines of identification are as follows:

1. The first line identifies the experimenter.
2. The second line is the four digit test number which is assigned by the shell script **runumb**.
3. The third line is **Destination**.
4. The fourth line identifies the type of test. See the footnote concerning this line for **preface.x**.
5. The fifth line is the name of the source site machine.
6. The sixth line is the name of the destination site machine (called destination identifier).
7. The seventh line contains two digits. The first digit is either 1 or 2, depending upon whether the session is connectionless or connection-oriented, respectively. The second digit is either 1 or 2, depending upon whether the disengagement is independent or negotiated, respectively.

Figure 10 is an example of this file.

```
NTIA-ITS (Boulder)
2260
Destination
User
NTIA - eldiente
NTIA - crestone
22
```

Figure 10. Example of a destination end user identification file.

Create the destination identification file, **preface.r**.

9. CONFIGURE THE PORTS AND CREATE THE CLOCK CALIBRATION FILE

9.1 Link the Communication Ports

Log in as net. Determine if com1 and com2 exist in directory dev by typing

```
cd /dev
ls
```

If they don't exist, create them and link them to the serial ports (i.e., tty00 and tty01) by typing

```
ln com1 tty00
ln com2 tty01
```

In any case, type,

```
cd
```

9.2 Set the Communication Ports

Figure 1 is a schematic diagram showing how the ports are to be connected. NTIA software assumes the clock receiver is a Kinemetrics True Time™ Model 468 Satellite Receiver.

Connect the two computers to the data communication system at their communication ports 1 (i.e., com1). Connect a satellite clock receiver to each computer at its communication port 2 (i.e., com2).

9.3 Create the Clock Calibration File

The time of each reference event is recorded. However, each recorded time must be corrected due to the transmission delay between each computer and its satellite clock receiver. Let T_0 be the reference event time as reported by the satellite clock receiver.

9.3.1 Response Reference Event Time

After a response is read, a single character is transmitted to the satellite clock receiver to obtain the time of the reference event, T_a. This transmission requires some time, say t_1. Therefore, the time obtained from the satellite clock receiver, T_0, is later than the reference event time, T_a. The actual time (after) the response is

$$T_a = T_0 - t_1.$$

9.3.2 Command Reference Event Time

Before a command is written, a single character could again be transmitted to the satellite clock receiver to obtain the time, and adjust it for transmission time. However, it is simpler to use the previously recorded time, T_0 and adjust it. A number of characters, say k characters, is required to define the time. They are transmitted to the computer and read individually.[11] The required time is t_2 (which will be approximately k times t_1). The actual time (before) the command is

$$T_b = T_0 + t_2.$$

Each end user must log in to the directory /usr/net. Type
calibr > clkdata.cal .

The C program **calibrate** corrects the time to within approximately 5 ms. The time must be calibrated only once – before the experiment begins.[12] For example, if t_1 = 2 ms and t_2 = 24 ms, the file is as follows:

2
24.

[11]It is slower but more reliable to read the characters individually than as a block.

[12]However, a clock calibration file must be created whenever there is a change in hardware or software, such as a change in the serial board, buffer, or operating system.

Enter the times, t_1 and t_2 into the text file clkcal as two lines.

These times will agree for each end user only if their hardware and software are identical.

10. SET THE TIME LIMITS FOR RESPONSES

A timeout is a factor of three times a specified primary performance parameter. There is a timeout value for each primary delay performance parameter (i.e., Access Time, Block Transfer Time, Destination Disengagement Time, and Source Disengagement Time). To assure that the extraction software is not unduly suspended awaiting a response, another type of "timeout" value should be specified. It should be longer than the maximum of the above four timeouts. (The timeout variable is called **toutcccc** where **cccc** is the baud.)

To set **timeout** (used in **converse**), enter directory **/usr/net/src/d3a**. Assign a value for **TOUT** in a **define** statement in the header file **pdn_test.h**. It is set once (for the entire experiment).

11. CONDUCT A TEST

When the experiment is designed, some of the variable conditions and levels necessary to conduct an experiment are defined. Many of the variable conditions are listed in the **default** file (Section 4.3 of Volume 2), which is located in net's home directory.[13] The performance values are specified in the **spi.acd** and **spi.xfr** files (Section 5 of Volume 2). Although these values are specified for the preliminary characterization test (Section 6 of Volume 2), perhaps they should be reconsidered at this time.

During User Information Transfer tests, blocks of characters are transmitted in a psuedorandom order from the following set of 64 characters: **A-Z**, **a-z**, **0-9**, **:**, and **;**.

> Before conducting a test, see that the connections and equipment are functioning properly. Log in as **net**.

11.1 Synchronize the Clocks

Once for each experiment, each end user must synchronize the UNIX[tm] clock time with the satellite clock time using the C program **st** (i.e., set time). The super user must change the ownership and mode of **st**.

> After logging in as **root**, the super user must change to the home directory of **net**, and set the clock by typing
>
> ```
> cd bin
> chown root st
> chmod +s st
> st
> ```
>
> This is required only once - after making the files.

[13]Variable conditions listed in default are: Source Site, Destination Site, Block Size, Interblock Delay, Number of Accesses, and Interaccess Delay. The number of blocks is also listed, but this is considered to be a fixed condition (due to the precision that is specified).

The program **st** will list the initial satellite clock time, and it indicates that the computer clock is set.

If they are not synchronized, something is wrong with transmission, the receiver, the antenna, or even the satellite; a delimiter (e.g., ., #, etc.) will appear to the right of the time in the first line.

If they are synchronized, the UNIX[tm] clock is automatically set to the time listed in the second line.

After the system clock has been set, the C program tt (i.e. true time) can determine the discrepancy. Type

tt

The screen will return a message such as

```
System time = Tue Feb 28 14:04:19 1989
NBS Satellite time =  28 14:04:19
System Time off =                0 seconds.
```

The discrepancy (i.e., **System Time off**) is considered inconsequential if it is less than 1 s.

Log out as root.

11.2 Select the Appropriate runx Command

Four commands are available to conduct a test: **runx**, **runxt**, **runxf**, and **runxtf**. Although each serves a slightly different function, they use the same arguments and files. Table 4 is a list of the four commands regarding flow control and adjustment of the reference event times (for transmission times from the computer to the satellite clock receiver and back to the computer).

Table 4. Available **runx** Commands

	Flow Control Option: On	Flow Control Option: Off
Adjust Times	**runxtf**	**runxt**
Do Not Adjust Times	**runxf**	**runx**

The following discusses the features of these options:

- <u>Examine Test Results Before Reduction</u>. If the **runxf** or **runx** commands are selected, the test data for three primary delay performance parameters (Access Time, Source Disengagement Time, and Block Transfer Time) can be examined immediately after disengagement. However, the reference event times will not have been adjusted for the transmission delays from the computer to the satellite clock receiver and back to the computer (Section 9.3).[14] This option is an artifact: the correlation algorithm that checks for bit failures (Volume 4) was originally quite slow, and these commands allowed this algorithm to be by-passed; trading accuracy for speed. See Section 12.4.

- <u>Adjust Times</u>. If the **runxtf** or **runxt** commands are selected, the test data are adjusted for the transmission times to and from the satellite clock receiver (as they should be).

- <u>Enable Software Flow Control</u>. If the **runxtf** or **runxf** commands are selected, flow control will be implemented. That is, the software implements **xon** and **xoff** - regardless of whether the network does also. If this option is selected, generally an even number of characters are recorded at the end of **log** file (indicating **xon** and then **xoff**). However, **xmit_net** can "panic" and send an extra **xoff**.

- <u>Do Not Enable Software Flow Control</u>. If the **runxt** and **runx** commands are selected, flow control will not be implemented (unless the network implements it).

[14]Reference event times preceding a command are increased by t_2 and those following a response are decreased by t_1. Documentation often refers to these adjustments as "tweaking".

The **runxt** command is the most commonly used of the four commands, and it will be used in all further discussion.

Select a **runx** command.

11.3 Start the Test

Each test is conducted by invoking a UNIX[tm] shell script, such as **runxt**. This shell script requires two arguments and allows two more. Specifically, the arguments are:

- Type of Test. **o** or **u**, depending upon whether the test is an access-disengagement (i.e., overhead) test or a user information transfer test, respectively.
- Network. <network> is the four-letter abbreviation, **aaaa** (which also exists in file **netcodes**). (The network, **aaaa**, specified here and the source site, **bbb**, specified in the **default** file uniquely identify the protocol file, **net-aaaa.bbb**.)
- Other Arguments. <levels of O_7 or U_8> and <levels of O_8 or U_9> are two optional arguments consisting of three characters each (Section 4 of Volume 2). Each argument represents a level of a variable condition not listed elsewhere. These variable conditions will be used to analyze the data (Volume 5).

The shell script activates the appropriate data extraction software in the source computer and in the destination terminal.

To conduct an access-disengagement test, type

runxt o <network> <level of O_7> <level of O_8>

where O_7 and O_8 are levels of optional variable conditions.

To conduct a user information transfer test, type

runxt u <network> <level of U_8> <level of U_9>

where U_8 and U_9 are levels of optional variable conditions.

The shell script **mover** moves the data collected at the destination site into permanent storage; the data extraction phase is now complete, and the data at the source site is automatically saved in permanent storage.

When execution of **runxt** is complete, the destination operator logs into net's main directory (**/usr/net**) and invokes the shell script **mover** by typing

mover.

11.4 Example of Data Extraction Using runxt

Example: Conduct a user information transfer test from Laramie to Boulder for the public telephone network through 9600-baud modems. Forty blocks of 512 characters are to be transferred. As an experiment, attempt to reduce the autocorrelation between blocks by adding a 1-second interblock delay.

Solution: Before conducting the test, make sure that all files are current. Then, do the following:

- Check the **preface.x** file and the **preface.r** file.
- Make sure that the values in the **spi.acd** and **spi.xfr** files are appropriate.
- Examine the **default** file. Before running this test, the **default** file would looked like this:

```
lar  source site
bol  destination site
512  block size (bytes or chars)
--- xfr info ---
40    number of blocks
0     interblock delay (sec)
--- ovh info ---
21    number of accesses
55    interaccess delay (sec)
```

  Since the default block size and the number of blocks are identical to those desired for this test, they needn't be changed. However, the default interblock delay is 0 s. We could change this delay to 1 s in the **default** file, but we could pass that value to **runxt** as an optional parameter, **-i 1**.

- Make sure the appropriate protocol file exists. That is, check the **netcodes** file to see that the abbreviation **lar** exists for Laramie, **bol** for Boulder, and **pt96** for Public Telephone at 9.6 kbps. Therefore, the protocol file would be named **net-pt96.lar**.

- After this information has been checked, conduct the test by typing

  runxt u pt96 -i 1

  at the source site. The operator at the source site can observe the progress of the test via **runxt**'s output to the monitor.

- When execution of **runxt** ends, the destination operator logs into the computer, enters **net**'s home directory (**/usr/net**), and invokes the shell script **mover** by typing

  mover.

This completes data extraction of the sample test by moving the data collected at the destination site into permanent storage.

11.5 Check the Results

Before proceeding, the operator should be as confident as possible that the test is valid.

> If the test is known to be flawed, the data should be discarded by erasing the line corresponding to this test in the **log file** and all files corresponding to this test in **/usr/data/3x** directory. This is accomplished by typing
>
> **rm /usr/data/3x/nnnn***
>
> where nnnn is the number of the flawed test.

The character * prompts the UNIX^{tm} command **rm** to remove all files from the **/usr/data/3x** directory whose names begin with **nnnn**.

11.6 Test the Data Extraction Software

After critical portions of the data extraction software have been developed or changed, they should be tested. The easiest method is to run a test. In order to avoid saving useless test data, four more run commands are available: **runx.tst**, **runxt.tst**, **runxf.tst**, and **runxtf.tst**. They are identical to their counterparts, **runx**, **runxt**, **runxf**, and **runxtf**, except that the data are not moved to permanent storage. Instead, they remain in **net's** home directories of both computers. After such a test, the operator must either save or erase the test data.

11.6.1 Save Test Data

If the data are to be saved (in permanent storage), enter the following two commands from net's home directory (**/usr/net**) in the source host:

movex
mklog nnnn

where **nnnn** is the test number. Then, enter the following command from **net's** home directory (**/usr/net**) in the destination host:

mover.

11.6.2 Erase Test Data

If the data collected are not to be saved, the files in the home directory of both hosts must be removed. This may be done by entering the command

cmd/repeat

from **net's** home directory in each host.

12. PROCESS THE TEST DATA

The data from a test may be processed by consolidating the extracted data from the two computers, merging the log files, copying the extracted data, and activating a **do** shell script.

12.1 Consolidate the Extracted Data

Data from a single test can be processed by consolidating the data into one computer and activating a comprehensive shell script. However, it is usually more efficient to consolidate the data after a number of tests, say 10, have been completed. The test data is in **usr/data/3x**. The contents of this directory and the file **log** (which is in the home directory of **net**) must be consolidated into one computer.

The 10 files from on-line data extraction and the **data.x** file are consolidated. That is, they are stored in one computer either manually by transporting magnetic tape or disks or electronically by the using the UNIX[tm] utility **uucp**, the public domain utility **Kermit**, or a network utility such as **FTP**. One of three types of programs is then used on the files: **merge**, **show-o**, and **reform**.

The following discussion assumes that the data are copied to a diskette, the diskette is transferred to the disk drive in the computer that is chosen for data processing, and the data are copied.

Format the diskette by typing,

```
format /dev/rfh0
cd ../data/3x
unalias ls.
```

12.2 Merge the log Files

The files may not be identical.

Verify that **log** and **log.bak** are identical in both computers type

diff log log.bak .

12.2.1 log Files Identical

If they are identical, copy log to a diskette by typing

cpio -o <log> /dev/fh0.

12.2.2 log Files Not Identical

If they are not identical, they can be made so, for example, by typing

```
sort -u log.bak > log.tmp
mv log.tmp log
cp log log.bak.
```

Copy log to a diskette by typing

cpio -o <log> /dev/fh0.

12.3 Copy the Extracted Data

The numbered data files for each test must be copied from a diskette into the **../data/3x** directory with their companion files and the **log** file (merged into the existing **log** file) in the home directory of **net**.

- Place a high-density diskette in the disk drive of one host.
- View the files on the diskette by typing,

  ls >list.

- In **list**, replace the line containing the word **list** with **../../net/log.**
- Copy the files in directory **3x** to the diskette by typing,

  cpio -o <list> /dev/fh0.

- Check contents of the diskette by typing,

  cpio -ivt </dev/fh0.

- Remove the diskette and place it in the disk drive of the host that will process the test(s).
- Change directory to **../data/3x.**
- Type

  cpio -ivf </dev/fh0.

The checksum files (**nnnncksm**) generated at each site can be compared to verify the integrity of data transfer.

12.4 Activate a do Shell Script

After the data of one or more tests are consolidated into one computer, one of the **do** shell scripts processes one test at a time. There are three such shell scripts:

- <u>do</u>. The shell script **do** is usually used. Regardless of which "runx" command was used to conduct the test, a code is passed to **do** indicating whether the times should be adjusted or not.[15] This shell script causes the extracted data from each test to be converted, reduced (Volume 4), and analyzed (Volume 5). The implementation of this shell script is described by Figure B-1a in Appendix B.

- **doqik**. The shell script **doqik** allows the experimenter to view files of some performance parameters prior to reduction and analysis. It provides a "quick look" at Access Time, Block Transfer Time, and Source Disengagement Time. This shell script is an artifact created because the original algorithm for detecting bit failures was quite slow, and it was often desirable to view these performance parameters prematurely (before adjustments for transmission delays between computer and satellite clock receiver), particularly Block Transfer Time. This shell script also knows which "runx" command was used to conduct the test. **doqik** produces the file **nnnninfo**:

    - Access-Disengagement Tests. For access-disengagement tests, this file contains Access Time and Source Disengagement Time.

    - User Information Tests. For user information transfer tests, this file contains Block Transfer Time.

    The file **nnnninfo** can be used by data display software to produce graphs of these performance parameters. The implementation of this shell script is described by Figure B-1b.

- **dopre**. The shell script **dopre** processes data for a preliminary characterization test (Volume 2, Section 6). This shell script is to be used with **runxt** or **runxtf**: It contains no provision for adjusting times. The implementation of this shell script is described in Figures D-1 and D-2 of Volume 2.

[15] **do** calls either **tweakall** or **tweaknon**, depending upon whether the times are to be adjusted or not, respectively.

From the home directory of net, type one of these three commands

```
do nnnn
doqik nnnn
dopre nnnn
```

where **nnnn** is the test number.

13. ACKNOWLEDGMENTS

Many people contributed to this volume. Dwight Melcher, Scott Seebass, Tim Gardner, Dan Byers, Chris Bogart, Darin Schwartz, John Waber, Margaret H. Morris, and Rob Reichart - all students working part-time - contributed to many aspects of this volume, particularly the software. Mike Eubanks, an ITS consultant, provided a key breakthrough in the design of the network access software. Lorna L. Kent, Rob Reichart, and D. J. Atkinson drew the structured design diagrams. Rob Reichart prepared many of the figures and tables in this volume. The authors are truly indebted to all of these contributors.

14. REFERENCES

ANSI (1987), American National Standard for Information Systems - Data communication systems and services - measurement methods for user-oriented performance evaluation, ANSI X3.141-1987 (American National Standards Institute, Inc., New York, NY).

ANSI (1983), American National Standard for Information Systems - Data communication systems and services - user-oriented performance parameters, ANSI X3.102-1983 (American National Standards Institute, Inc., New York, NY).

Kernighan, B.W., and D.M. Richie (1978), *The C Programming Language*, 228 pp. (Prentice-Hall, Englewood Cliffs, NJ)

Spies, K.P., D.R. Wortendyke, E.L. Crow, M.J. Miles, E.A. Quincy, and N.B. Seitz (1988), User-oriented performance communication services: Measurement design, conduct, and results, NTIA Report 88-238, August, 294 pp. (NTIS Order Number PB 89-117519/AS).

APPENDIX A: ON-LINE DATA EXTRACTION

The following two sections describe the on-line data extraction for each end user.

A.1 Source End User

Figure A-1 is a structured design diagram of the on-line data extraction software as implemented by **runxt** at the source site. This section describes the major functions of this software and some files it produces.

A.1.1 The xmit_net Program

Program **xmit_net** reads the source end user identification file (**preface.x**), reads the protocol file (**net-aaaa.bbb**), reads the arguments supplied by **runxt**, reads the two delays (resulting from transmission delays to and from the satellite clock receiver) (file **clkcal**), transfers a block of pseudorandom bytes, updates the **log.x** file, generates and monitors reference events, starts the destination user application program (**recv**)[16], generates two binary files of reference events, and logs out. The five files that it generates are **data.x**, **history.x**, **overhead.x**, **logn.x**, and **log.x**. These files are discussed below.

A. data.x File

This ASCII text file contains a user information block consisting of a set of 64 characters (**A-Z**, **a-z**, **0-9** ; and :) generated in a psuedorandom order. This file resides in **net**'s home directory in the source computer. Figure A-2 is a sample **data.x** file.

B. history.x File

This binary file contains

- the first six lines of **preface.x** (plus a seventh line that contains the month, day, and year of the test),
- the type of session,
- the type of disengagement, and

[16]It sends four values to recv: the number of blocks, the block size, the number of access attempts, and the test number.

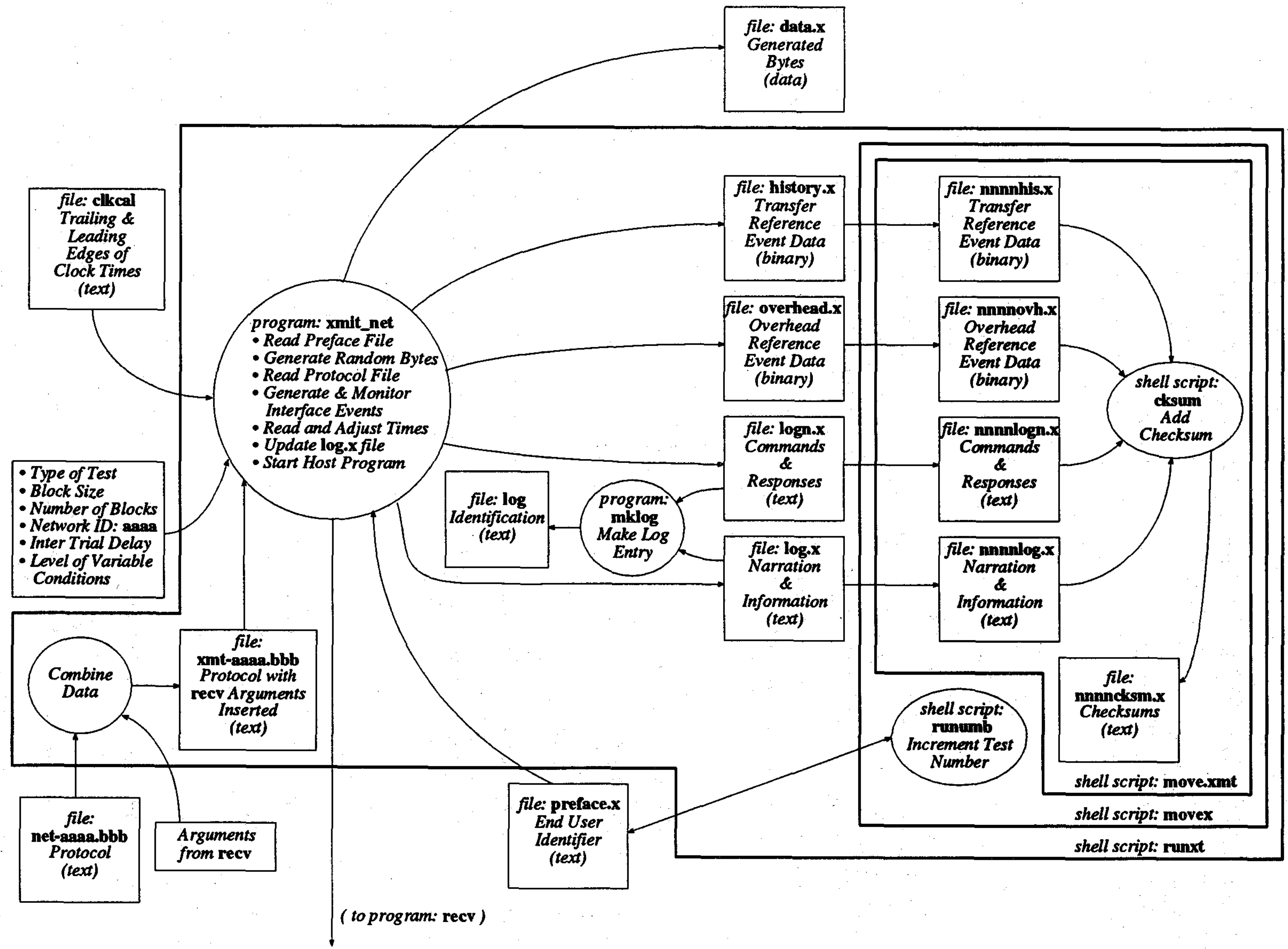

Figure A-1. Structured design diagram of on-line data extraction at the source site.

```
OOE2W4Y4YAhEHEEsxssERFFEOGGOu77UoeFORPMOHWaOQnx1TXr26UI6m668
daRYrJ4exmliJD;8EvJB;cw:qbGGvSQ8OmRB2N3;1LwBaRzAiI72EGkQxAiwOwwO
DVzsfuml8rjH7Ov4TPcik3aIS1RpRZr26cJj.rP5PXbUcds;aJyJRtAyqNZLKOGGO
mi;gYcnNhikfHj7Zw3jRSME14qyNoRREVK6GeLr4K2WgteeETFFsBqrEYCBEHkoO
ixmVuPA2dBw6bgV9R2vToRphHzRYD5FGspNhiUFyheEb6I2mKwZfVFCnjPK1QJ14

...

ivuMox1NpBL32UApjPqhPbLS13Mnp8TOcKDHvxUHaKS1vLxZv3781dp
PpJYactWvvMAua3Q3vrVmuduW3wrVmeP6XgNCWB5F22szOTvQ4v7789sqNZzXXMD
eovVHi7RnpstOry9a5AVBZZEKOGWz:jYlIuBiweGTFVyXPLNcIhDoTphPK14SH9H
;;kjfknslr6iJDfgckP5vzUHaSrhv2jxuskV2oytkNRkZO7He7p6SvYfdsWPj6BV
BxSo5qaeFLkwd2hhkfvWnGgT6nive258NWLaaOYQz4bom87FONO;FXCpLoQxAiYF
qokMppcH72kKYil;7N3v7jZVOgBoD7FeTKtxUXEizeGbyH5G2mifs2LmoUmmM5;1
N;mb2QnBrj3JebVkuF5:Vh9orsNwgeUFqYmOpRZL6miPaTLN1Pn5C98sSYruVPz4
```


Figure A-2. Pseudorandom, 64-character ASCII data used for transmission of the user data and stored in **data.x** at the source site.

- the start time.

Then, for each block, it contains

- the record number,
- the number of bytes in the block,
- the start time of the block transfer (i.e., hr, min, sec, fraction), and
- the end time of the block transfer (i.e., hr, min, sec, fraction).

Figure A-3 is a text version of this binary file (i.e., **nnnnhis.x** located in **data/3x**)[17]. (To read this file, type **show-h 1280 nnnnhis.x**.)

C. overhead.x File

This binary file contains

- the first six lines of **preface.x** (plus a seventh line that contains the month, day, and year),
- the type of session,

[17]It is produced during data conversion by the show-h program.

```
History•information files:

Perfor. measur. ID      - UW to NTIA.ITS
Run number              - 2134
Type                    - Source
Information ID          - User
Source                  - UW - eldiente
Destination             - NTIA - crestone
Mo/Day/Yr               - 3/6/89
Session Category        : Connection oriented
Disengagement Category  : Negotiated
Start time (Hr:Min:Sec) - 14:48:18
              Data from file     /usr2/net/../data/3x/2134his.x
      Record Bytes      Start time               End time

      128    128        14:48:18:431      14:48:18:436
      128    128        14:48:18:478      14:48:18:484
      128    128        14:48:18:524      14:48:18:704
      128    128        14:48:18:744      14:48:18:750
      128    128        14:48:18:791      14:48:18:973
      128    128        14:48:19:013      14:48:19:018
      128    128        14:48:19:058      14:48:19:241
      128    128        14:48:19:281      14:48:19:287
      128    128        14:48:19:327      14:48:19:510
      128    128        14:48:19:551      14:48:19:557
      128    128        14:48:19:597      14:48:19:779
      128    128        14:48:19:819      14:48:19:824
      128    128        14:48:19:864      14:48:20:048

                                    ...

      128    128        14:48:27:075      14:48:27:080
      128    128        14:48:27:121      14:48:27:304
      128    128        14:48:27:344      14:48:27:350
      128    128        14:48:27:391      14:48:27:572
      128    128        14:48:27:612      14:48:27:618
      128    128        14:48:27:658      14:48:27:841
      128    128        14:48:27:881      14:48:27:886
      128    128        14:48:27:927      14:48:28:110
      128    128        14:48:28:150      14:48:28:155
      128    128        14:48:28:195      14:48:28:379
      128    128        14:48:28:419      14:48:28:424
      128    128        14:48:28:464      14:48:28:648
      128    128        14:48:28:688      14:48:28:694
      128    128        14:48:28:734      14:48:28:916
      128    128        14:48:28:961      14:48:28:966
```

Figure A-3. Text version of the **history.x** file produced by **show-h**.

- the type of disengagement, and
- the start time.

Then, for each block, it contains

- the record number,
- the state code of the three entities about the source user/system interface,
- the code indicating the order of time stamping, and
- the event time (i.e., hr, min, sec, fraction).

Figure A-4 is a text version of this binary file (i.e., **nnnnovh.x** in **data/3x**).

D. **logn.x File**

Figure A-5 is a sample file. This text file is a log of all commands /and responses. After the files have been moved and the test number prefix added, the name of this file is changed to **nnnnlogn.x**.

E. **log.x File**

This text file contains messages that are sent to the source operator via the console. Figure A-6a is an example of **log.x** for access-disengagement tests and Figure A-6b is an example of **log.x** for user information transfer tests.

A.1.2 Make a Log Entry

Program **mklog** uses information from files **logn.x** and **log.x** to append one record per test to the file **log**. Each line of **log** contains the test number, the date, the day of the month and time of day, the three-letter identification of the source site (**bbb**), the four-letter identification of the network (**aaaa**), the type of test (**ovh** or **xfr**), the number of access attempts, the number of blocks transferred, the block size, the number of seconds between access attempts, the number of seconds between blocks, and the destination site. Figure A-7 is an example of the **log** file. It will be used by the C program **qklog** to produce the **log.acc** and **log.xfr** files (for analysis of multiple tests).

Overhead•information files:

```
Perfor. measur. ID              - UW to NTIA.ITS
Run number                      - 2115
Type                            - Source
Information ID                  - User
Source                          - UW - eldiente
Destination                     - NTIA - crestone
Mo/Day/Yr                       - 3/6/89
Session Category                : Connection oriented
Disengagement Category          : Negotiated
Start time (Hr:Min:Sec)         - 13: 7:14

        Data from file          /usr2/net/../data/a3x/2115ovh.x

    Record  Code        Clock Time

      1     0110        13:07:14.829
      2     0110        13:07:16.049
      3     0230        13:07:16.066
      4     0320        13:07:55.493
      5     0221        13:07:55.510
      6     0320        13:07:58.739
      7     0441        13:07:58.799
      8     0540        13:07:59.182
      9     0450        13:07:59.237
     10     0110        13:08:00.103
     11     0110        13:08:00.120
     12     0110        13:08:01.046
      1     0110        13:09:00.256
      2     0110        13:09:01.470
                ...

     11     0450        13:39:43.755
     12     0110        13:39:44.658
      1     0110        13:40:44.174
      2     0110        13:40:45.386
      3     0230        13:40:45.403
      4     0320        13:41:26.443
      5     0221        13:41:26.460
      6     0320        13:41:29.273
      7     0231        13:41:29.313
      8     0320        13:41:29.716
      9     0441        13:41:29.762
     10     0540        13:41:30.646
     11     0450        13:41:30.663
     12     0110        13:41:31.593
Total # times - 240
```

Figure A-4. Text version of the **overhead.x** file produced by **show-h**.

```
crestone 1%
.......
out:    recv -t 80 128 1 2134
 in:    recv -t 80 128 1 2134
READY
.......
out:
 in:     crestone 2%
.......
out:     mover
  in:    mover
moving net test files to ..data for test 2134
crestone 3%
.......
out:    logout
  in:   logout
HJ
NO CARRIER
```

Figure A-5. Contents of **logn.x** (renamed 2134logn.x).

```
......... network transmission ........... From: lar  via: vm96  to:  bol
Start test 2115 (Satellite time - 13:07:14)     Mon Mar  6  13:07:13  1989

1 blocks of 512 chars to be sent for each of 20 accesses, - 10240 total chars
Attempt open # 1 13:07:14, Open,  Xmit complete, Transact. complete  13:07:59
Attempt open # 2 13:09:00, Open,  Xmit complete, Transact. complete  13:09:43
Attempt open # 3 13:10:45, Open,  Xmit complete, Transact. complete  13:11:30
Attempt open # 4 13:12:32, Open,  Xmit complete, Transact. complete  13:13:16
Attempt open # 5 13:14:18, Open,  Xmit complete, Transact. complete  13:15:01
Attempt open # 6 13:16:03, Open,  Xmit complete, Transact. complete  13:16:50

                       ...

Attempt open # 15 13:31:54, Open,  Xmit complete, Transact. complete  13:32:37
Attempt open # 16 13:33:40, Open,  Xmit complete, Transact. complete  13:34:24
Attempt open # 17 13:35:26, Open,  Xmit complete, Transact. complete  13:36:11
Attempt open # 18 13:37:13, Open,  Xmit complete, Transact. complete  13:37:57
Attempt open # 19 13:38:59, Open,  Xmit complete, Transact. complete  13:39:42
Attempt open # 20 13:40:44, Open,  Xmit complete, Transact. complete  13:41:29

10240 characters transmitted
test completed                  Mon Mar 6  13:41:29  1989
................
Time stamps have been tweaked with T1 - 2 & T2 - 15
```

Figure A-6a. Screen display of **log.x** for access-disengagement tests.

```
......... network transmission ........... From:   lar via:  vx96  TO:  bol
Start test 2134 (Satellite time - 14:47:33)  Mon Mar 6 14:47:34  1989

80 blocks of 128 chars to be sent for each of 1 accesses, -10240 total chars
Attempt open # 1 14:47:33, Open, Xmit complete, Transact. complete 14:48:28

10240 characters transmitted
test completedMon Mar 6 14:48:30 1989
................
Time stamps have been tweaked with T1 - 2 & T2 - 15
```

Figure A-6b. Screen display of **log.x** for user information transfer tests.

A.1.3 Add recv's Arguments

The arguments from the **recv** application program are added to the protocol file **net-aaaa.bbb**. This new protocol file **xmt-aaaa.bbb** is indicated as the beginning of the **logn.x** file in Figure A-5.

A.1.4 Increment the Test Number

The test number is incremented by the shell script **movex**. **movex** invokes the shell script **runumb** to increment the test number in the **preface.x** file.

A.1.5 Move the Files and Add the Test Number

The shell script **move.xmt** moves the files and adds the test number (as a prefix) to **history.x**, **overhead.x**, **logn.x**, and **log.x**: they are renamed **nnnnhis.x**, **nnnnovh.x**, **nnnnlogn.x**, and **nnnnlog.x**, respectively.

A.1.6 Add the Check Sum

The shell script **move.xmt** calls the shell script **cksum** which, adds the check sum to **nnnnhis.x**, **nnnnovh.x**, **nnnnlogn.x**, and **nnnnlog.x**. It then creates the file **nnnncksm.x**.

A.2 Destination End User

Figure A-8 is a structured design diagram of the on-line data extraction as accomplished by the destination end user application program **recv**.

```
211103/06/891258larvx96xfr 1 20512A55b00bol
211203/06/891259larvx96xfr 1 80128A55b00bol
211303/06/891301larvx96xfr 1 80128A55b00bol
211403/06/891302larvx96xfr 1160 64A55b00bol
211503/06/891307larvm96ovh20  1512A55b00bol
211603/06/891345larvx96xfr 1 20512A55b00bol
211703/06/891346larvx96xfr 1 80128A55b00bol
2118  03/06/89   1348  lar  vx96  xfr   1   160   64  A55  b00  bol
2119  03/06/89   1350  lar  vx96  xfr   1    80  128  A55  b01  bol
2120  03/06/89   1353  lar  vx96  xfr   1   160   64  A55  b01  bol
2121  03/06/89   1357  lar  vx96  xfr   1    20  512  A55  b01  bol
2122  03/06/89   1358  lar  vx96  xfr   1   160   64  A55  b00  bol
2123  03/06/89   1400  lar  vx96  xfr   1    20  512  A55  b00  bol
2124  03/06/89   1402  lar  vx96  xfr   1    80  128  A55  b00  bol
2125  03/06/89   1414  lar  vx12  xfr   1    20  512  A55  b00  bol
2126  03/06/89   1417  lar  vx12  xfr   1   160   64  A55  b00  bol
2127  03/06/89   1420  lar  vx12  xfr   1    80  128  A55  b00  bol
2128  03/06/89   1430  lar  vx19  xfr   1   160   64  A55  b00  bol
2129  03/06/89   1432  lar  vx19  xfr   1    20  512  A55  b00  bol
2130  03/06/89   1433  lar  vx19  xfr   1    80  128  A55  b00  bol
2131  03/06/89   1435  lar  vx19  xfr   1    20  512  A55  b01  bol
2132  03/06/89   1436  lar  vx19  xfr   1   160   64  A55  b01  bol
2133  03/06/89   1440  lar  vx19  xfr   1    80  128  A55  b01  bol
2134  03/06/89   1447  lar  vx96  xfr   1    80  128  A55  b00  bol
2135  03/06/89   1448  lar  vx96  xfr   1   160   64  A55  b00  bol
2136  03/06/89   1450  lar  vx96  xfr   1    20  512  A55  b00  bol
2137  03/06/89   1452  lar  vm96  ovh  12     1  512  A55  b00  bol
2138  03/06/89   1525  lar  mc96  xfr   1    20  512  A55  b00  bol
2139  03/06/89   1527  lar  sp96  xfr   1    20  512  A55  b00  bol
2140  03/06/89   1528  lar  sp96  xfr   1    80  128  A55  b00  bol
2141  03/06/89   1530  lar  mc96  xfr   1    80  128  A55  b00  bol
2142  03/06/89   1531  lar  mc96  xfr   1   160   64  A55  b00  bol
2143  03/06/89   1533  lar  sp96  xfr   1   160   64  A55  b00  bol
2145  03/13/89   0900  bol  bakx  xfr   1    80  128  A55  b00  bol
2146  03/13/89   0901  bol  bakx  xfr   1   160   64  A55  b00  bol
2147  03/13/89   0902  bol  bakx  xfr   1    20  512  A55  b00  bol
2148  03/13/89   0913  bol  bakx  xfr   1    20  512  A55  b01  bol
2149  03/13/89   0914  bol  bakx  xfr   1   160   64  A55  b01  bol
2150  03/13/89   0917  bol  bakx  xfr   1    80  128  A55  b01  bol
2151  03/13/89   0920  bol  bx19  xfr   1    80  128  A55  b00  bol
2153  03/13/89   0925  bol  bx19  xfr   1    80  128  A55  b00  bol
2154  03/13/89   0926  bol  bx19  xfr   1   160   64  A55  b00  bol
2155  03/13/89   0927  bol  bx19  xfr   1    20  512  A55  b00  bol
```

Figure A-7. A portion of the log file generated by the mklog program.

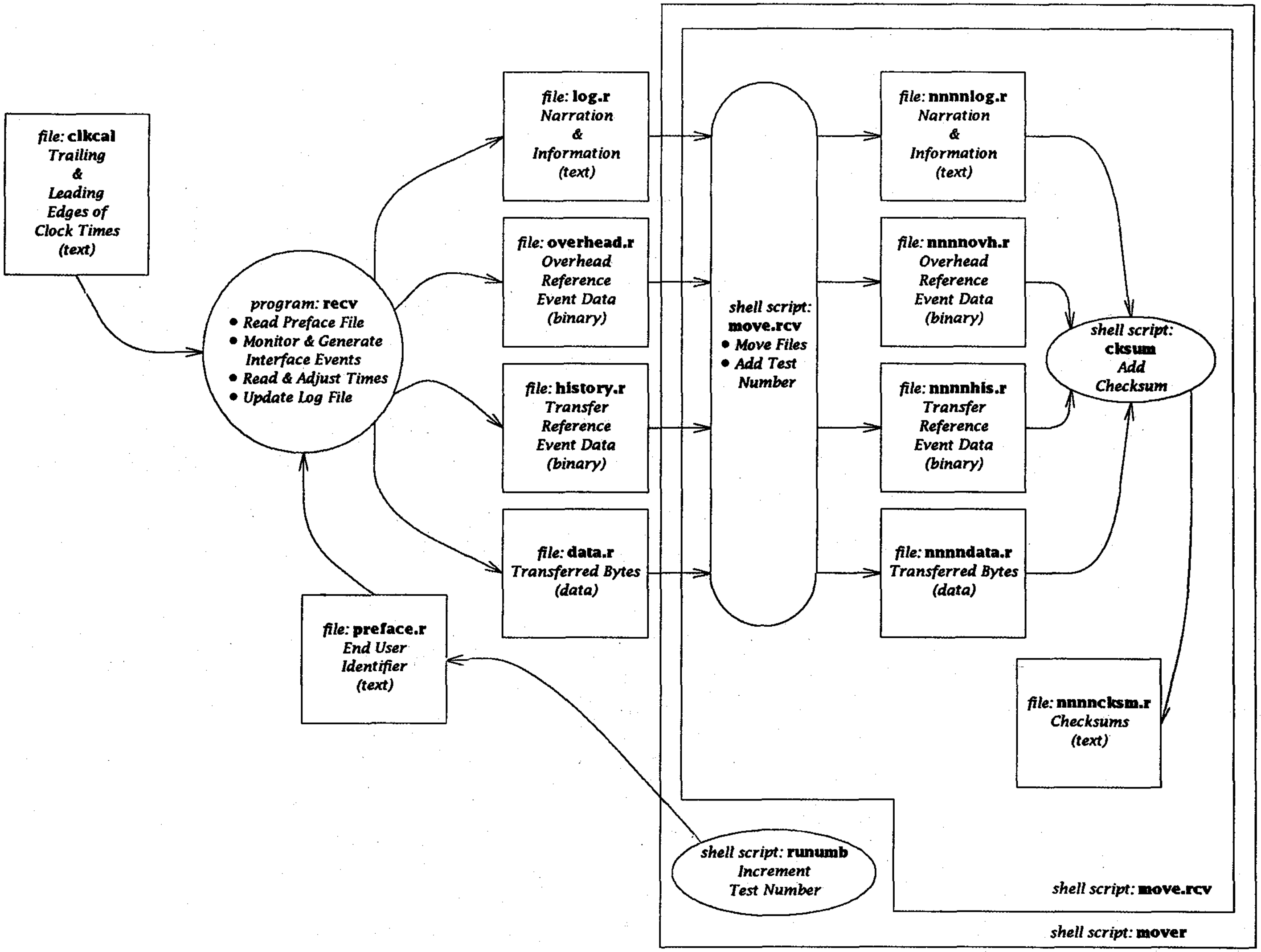

Figure A-8. Structured design diagram of on-line data extraction at the host terminal.

A.2.1 recv Program

recv reads the destination end user identification file (**preface.r**), generates and monitors reference events, calibrates the satellite clock times (**clkcal**), updates the **log.r** file for each test, passes its arguments to the protocol file (i.e., **net-aaaa.bbb**), and generates four files: **log.r**, **overhead.r**, **history.r**, and **data.r**. These files are described below.

A. log.r File

This text file contains the information shown on the destination terminal's console.

B. history.r File

This binary file contains

- the first six lines of **preface.r** (plus a seventh line that contains the month, day, and year),
- the type of session,
- the type of disengagement, and
- the start time.

Then, for each block, it contains

- the record number,
- the number of characters in the block,
- the start time of the block transfer (i.e., hr, min, sec, fraction), and
- the end time of the block transfer (i.e., hr, min, sec, fraction).

Figure A-9 is a text version of this binary file (i.e., **nnnnhis.r** in **data/3x**).

History•information files:

```
Perfor. measur. ID        - UW to NTIA.ITS
Run number                - 2134
Type                      - Destination
Information ID            - User
Source                    - UW - eldiente
Destination               - NTIA - crestone
Mo/Day/Yr                 - 3/6/89
Session Category          : Connection oriented
Disengagement Category    : Negotiated
Start time (Hr:Min:Sec)   - 14:48:18

  Data from file  /usr2/net/../data/3x/2134his.r
      Record      Bytes       Start time       End time

        1         128         14:48:18:347     14:48:18.797
        2         128         14:48:18:814     14:48:18.931
        3         128         14:48:18:948     14:48:19.065
        4         128         14:48:19:082     14:48:19.239
        5         128         14:48:19:256     14:48:19.373
        6         128         14:48:19:390     14:48:19.508
        7         128         14:48:19:525     14:48:19.642
        8         128         14:48:19:659     14:48:19.776
        9         128         14:48:19:793     14:48:19.910
       10         128         14:48:19:927     14:48:20.045
       11         128         14:48:20:062     14:48:20.179
       12         128         14:48:20:196     14:48:20:314
       13         128         14:48:20:331     14:48:20:448
       14         128         14:48:20:465     14:48:20:583
       15         128         14:48:20:600     14:48:20:717
       16         128         14:48:20:734     14:48:20:851
       17         128         14:48:20:868     14:48:20:986
       18         128         14:48:21:003     14:48:21:120

                              ...

       70         128         14:48:27:991     14:48:28:108
       71         128         14:48:28:125     14:48:28:243
       72         128         14:48:28:260     14:48:28:243
       73         128         14:48:28:394     14:48:28:512
       74         128         14:48:28:529     14:48:28:646
       75         128         14:48:28:663     14:48:28:780
       76         128         14:48:28:797     14:48:28:915
       77         128         14:48:28:932     14:48:28:057
       78         128         14:48:29:074     14:48:29:183
       79         128         14:48:29:200     14:48:29:318
       80         128         14:48:29:335     14:48:29:452
```

Figure A-9. Text version of the **history.r** file produced by **show-h**.

C. **overhead.r File**

This binary file contains

- the first six lines of **preface.r** (plus a seventh line that contains the month, day, and year),
- the type of session,
- the type of disengagement, and
- the start time.

Then, for each block, it contains

- the record number,
- the state code of the three entity-interface combinations about the destination user-system interface and the code indicating the order of time stamping, and
- the event time (i.e., hr, min, sec, fraction).

Figure A-10 is a text version of this binary file (i.e., **overhead.r** in **data/3x**). Note that the state code of the destination portion of the system is listed first (i.e., to the left) and the code for the destination end user is listed second; this order is opposite that in **overhead.x**. (See Section 5.)

D. **data.r File**

This ASCII text file contains the psuedorandom characters of **data.x** as received by the destination end user. It will be identical to **data.x** if there have been no failures.

A.2.2 Increment the Test Number

This is implemented by the shell script **mover**. It invokes the shell script **runumb** to increment the test number in the **preface.r** file.

A.2.3 Move the Files and Add the Test Number

The shell script **move.rcv** moves the files and adds the test number (as a prefix) to **history.r**, **overhead.r**, **data.r**, and **log.r**: they are renamed **nnnnhis.r**, **nnnnovh.x**, **nnnndata.r**, and **nnnnlog.r**, respectively.

Overhead•information files:

```
Perfor. measur. ID       - UW to NTIA.ITS
Run number               - 2115
Type                     - Destination
Information ID           - User
Source                   - UW - eldiente
Destination              - NTIA - crestone
Mo/Day/Yr                - 3/6/89
Session Category         : Connection oriented
Disengagement Category   : Negotiated
Start time (Hr:Min:Sec)  - 13:07:58

  Data from file   /usr2/net/../data/3x/2115ovh.r
      Record        Code        Clock time

        1           0023        13:07:58:120
        2           0132        13:07:58:620
        3           0023        13:07:58:623
        4           0022        13:07:58:640
        5           0023        13:07:59:545
        6           0032        13:07:59:562
        7           0045        13:07:59:563
        8           0111        13:07:59:607
        1           0023        13:09:43:091
        2           0132        13:09:43:143
        3           0023        13:09:43:146
        4           0022        13:09:43:163
        5           0023        13:09:44:090
        6           0032        13:09:44:107
        7           0045        13:09:44:109
        8           0111        13:09:44:153
        1           0023        13:11:30:114
        2           0132        13:11:30:166

                    ...

        1           0023        13:41:29:111
        2           0132        13:41:29:163
        3           0023        13:41:29:166
        4           0022        13:41:29:183
        5           0023        13:41:30:082
        6           0032        13:41:30:099
        7           0045        13:41:30:100
        8           0111        13:41:30:144
Total # times - 160
```

Figure A-10. Text version of the **overhead.r** file produced by **show-o**.

A.2.4 Add the Check Sum

The shell script **move.rcv** calls the shell script **cksum** which adds the check sum to **nnnnlog.r**, **nnnnhis.r**, **nnnnovh.r**, and **nnnndata.r**. It then creates the file **nnnncksm.r**.

APPENDIX B: OFF-LINE DATA EXTRACTION (DATA CONVERSION)

The off-line data extraction results in a set of ASCII text files of reference event times. Figure B-1 is a two-part structured design diagram depicting this procedure. Figure B-1a is a diagram of the shell script **do** which accomplishes all processing of a test. Figure B-1b is a diagram of the shell script **doqik** which omits reduction and analysis software to produce "quick" estimates of three performance parameters without adjustments for delays reading the satellite clock receiver: Access Time and Source Disengagement Time (from access-disengagement tests) and Block Transfer Time (from user information transfer tests).

The files produced by **do**, the **data.x** file, and the specification files, **spi.acd** and **spi.xfr**, are input to data reduction processing.

B.1 Consolidate the Files

The ten files from on-line data extraction and the **data.x** file are consolidated. That is, they are stored in one computer either manually by transporting magnetic tape or disks, or electronically by using the UNIX^{tm} utility **uucp**, the public domain utility **Kermit**, or a network utility such as **FTP**. Then one of four types of programs are used on the files: **merge**, **show-o**, **show-h**, and **reform**.

B.2 Merge User Information and Transfer Reference Event Data and Reformat

For each end user, program **merge** has two functions:

- It merges transfer reference event data and user information.
- It reformats the two files of merged data from binary to text.

B.2.1 SUI File

The source transfer information file (**data.x**) and the transfer reference event data (**nnnnhis.x**) are merged and reformatted to create the text file, **SUI** (i.e., source user information data). It obtains the block transfer start time and the block size from **nnnnhis.x**. It then extracts the transmitted ASCII characters from **data.x**.

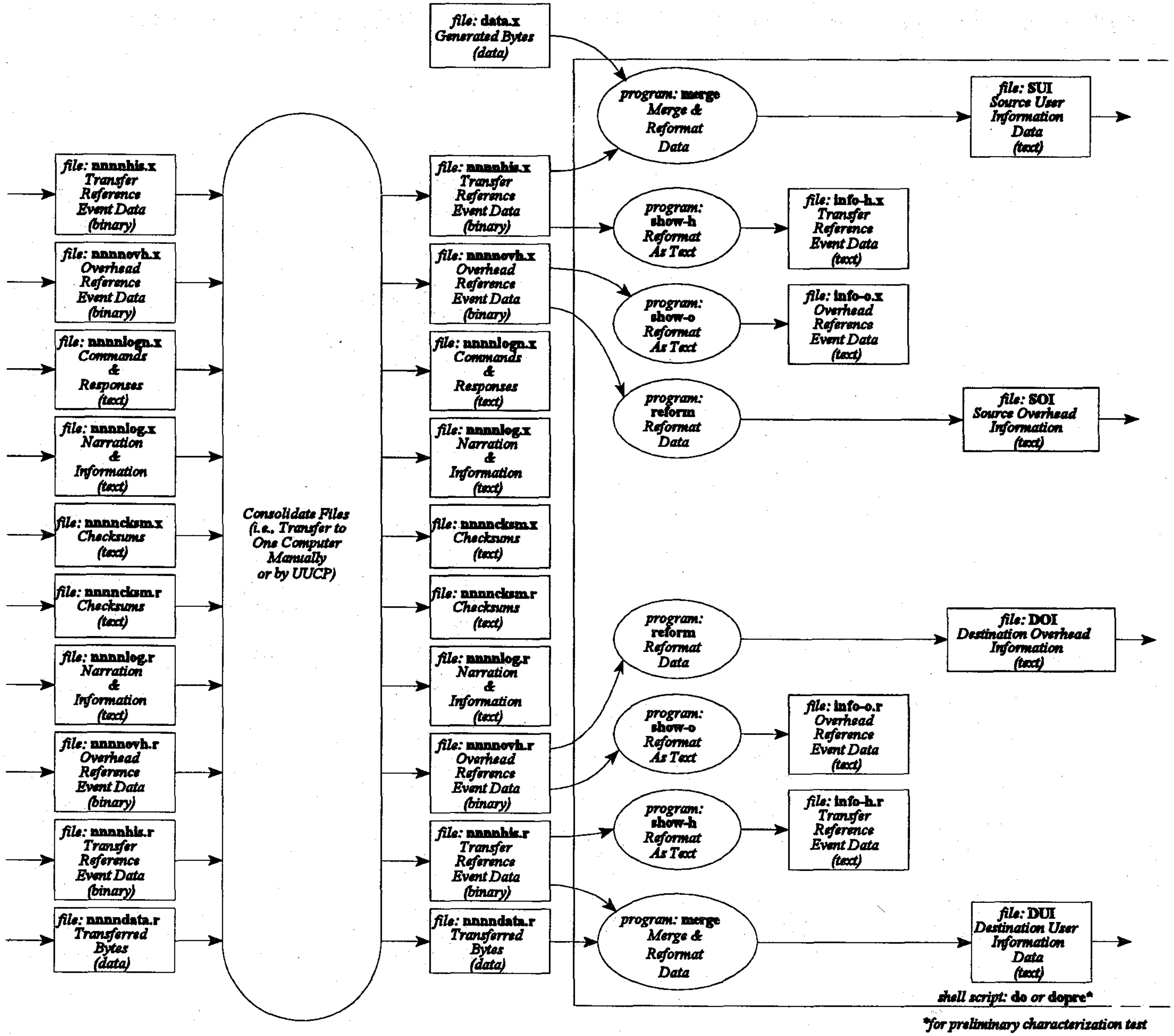

Figure B-1a. Structured design diagram of off-line data extraction for **do** or **dopre**.

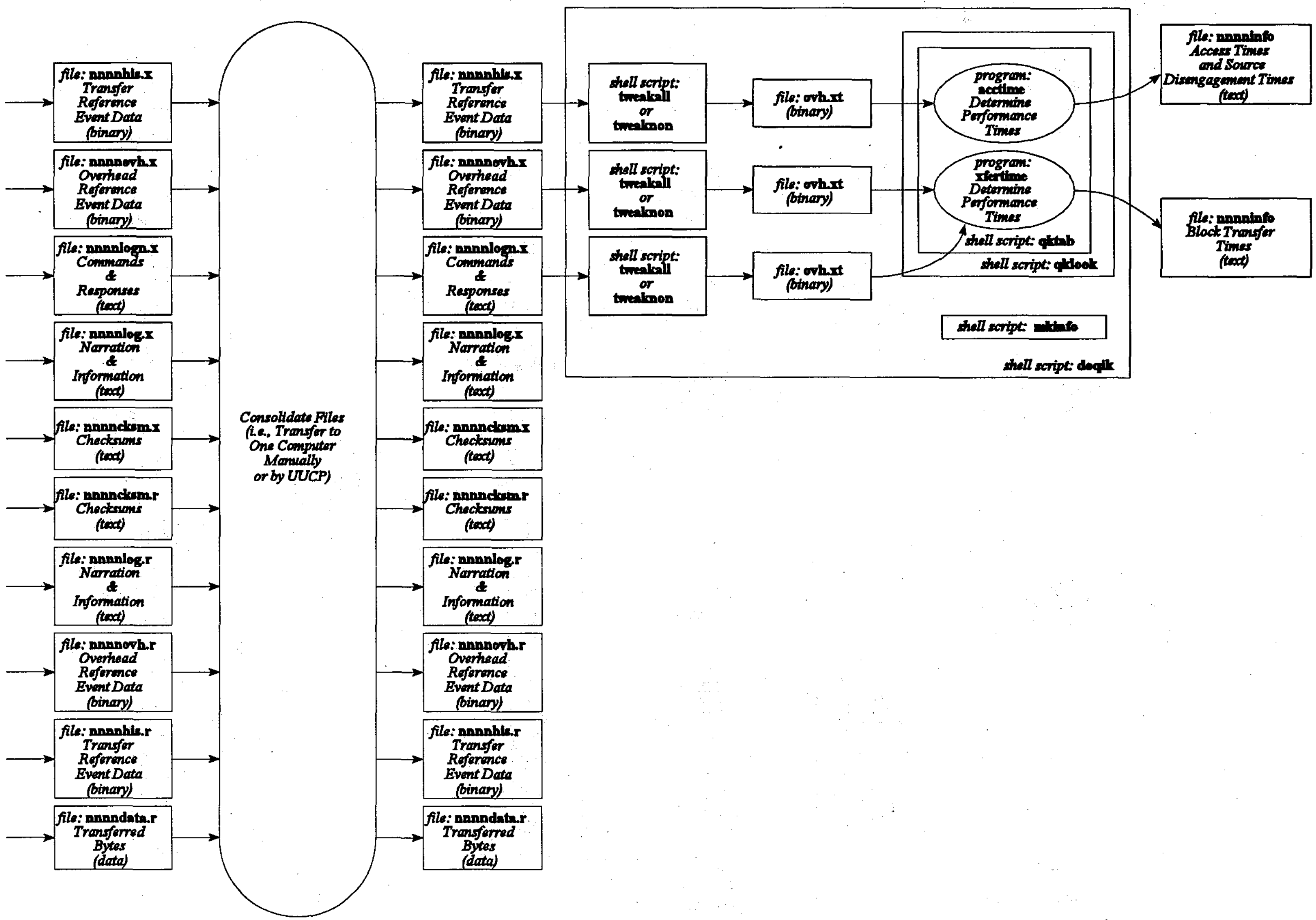

Figure B-1b. Structured design diagram of off-line data extraction for **doqik**.

These characters are converted to machine-independent ASCII characters by dividing the binary representation of the transmitted characters into a sequence of 15-bit strings.

The structure of this file is shown in Figure B-2. If necessary, the last string in the block is completed with binary zeros. Each string is regarded as the binary representation of a decimal integer, where the bit of the lowest index is the most significant bit. The user information block is thus mapped into a sequence of decimal integers in the range 0-32,767. The digits for each decimal integer are stored in **SUI**. Figure B-3 shows the format of **SUI** and **DUI**. An example of the **SUI** file is shown in Figure B-4.

B.2.2 DUI File

Similarly, the received transfer information file (**data.r**) and the transfer reference event data (**nnnnhis.r**) are merged and reformatted to create the text file **DUI** (destination user information data). An example of the **DUI** file is shown in Figure B-5.

B.3 Convert the Transfer Reference Event and Overhead Information into Text Data

Program **show-h** reads the binary transfer reference event files (**nnnnhis.x** and **nnnnhis.r**) and prints the text user information times.

Program **show-o** reads the binary overhead reference event files (**nnnnovh.x** and **nnnnovh.r**) and prints the overhead information in text format.

B.4 Reformat the Overhead Reference Event Data

Program **reform** reformats the files **nnnnovh.x** and **nnnnovh.r** and produces the files **SOI** and **DOI**, respectively. These files contain some preface information, followed by a sequence of overhead reference event data lines. Figure B-6 portrays the structure of **SOI** and **DOI**. Figure B-7 lists the detailed format of the **SOI** and **DOI** files. Examples of the **SOI** and **DOI** files are shown in Figures B-8 and B-9, respectively.

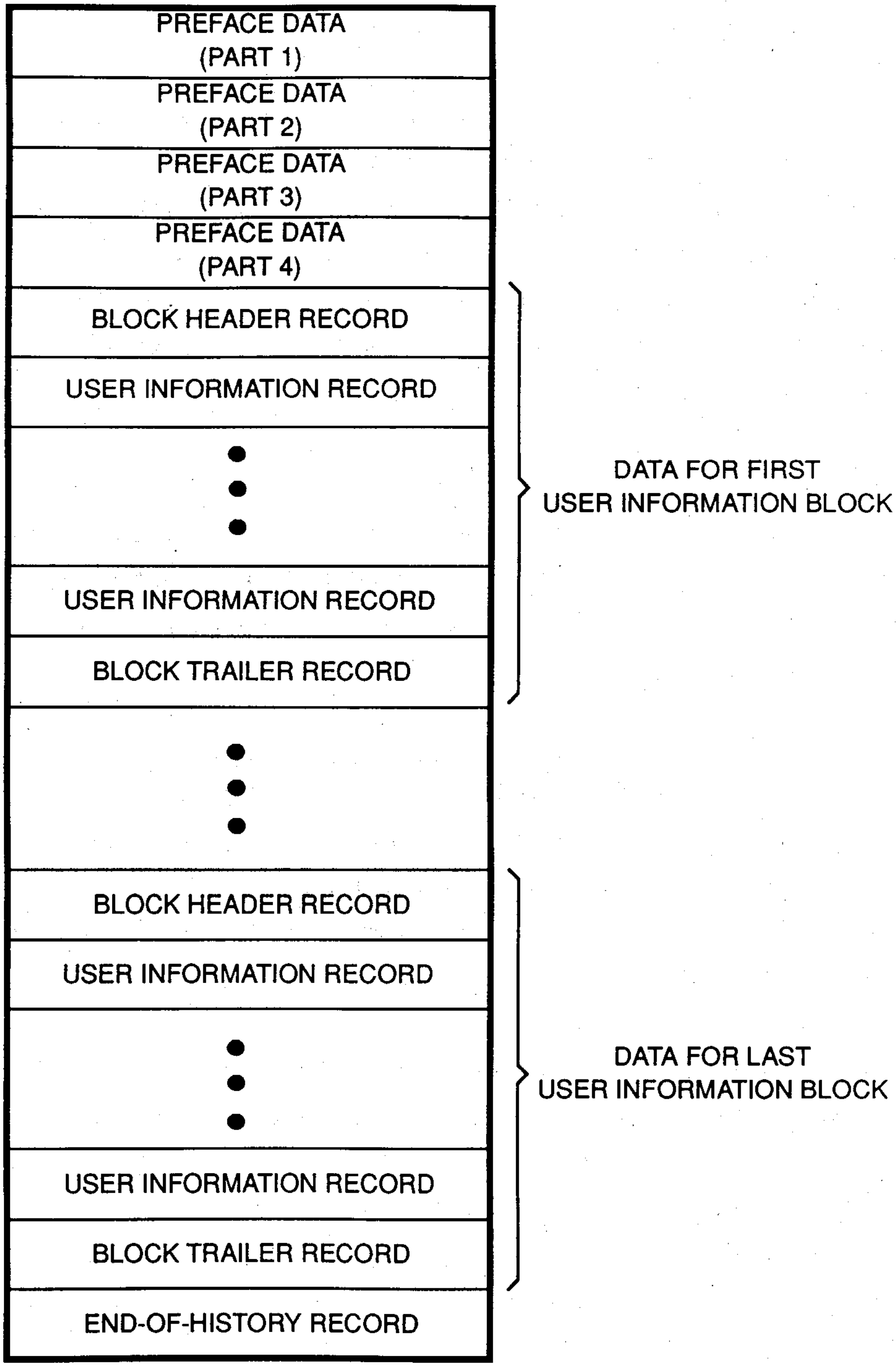

Figure B-2. Structure of the source (SUI) and destination user information (DUI) files.

CHARACTER FIELD	EDIT DESCRIPTOR	CONTENTS	
PREFACE DATA (PART 1):			
1-32	A32	FILE DESCRIPTOR	
PREFACE DATA (PART 2):			
1-64	A64	BATCH IDENTIFIER	
PREFACE DATA (PART 3):			
1-32	A32	SOURCE USER IDENTIFIER	
33-64	A32	DESTINATION USER IDENTIFIER	
PREFACE DATA (PART 4):			
1-4	I4	YEAR	REFERENCE TIME (DATE AT ORIGINATING USER SITE)
5-8	I4	MONTH	
9-12	I4	DAY	
13-16	I4	HOURS	REFERENCE TIME (LOCAL TIME-OF-DAY AT ORIGINATING USER SITE)
17-20	I4	MINUTES	
21-28	F8.0	SECONDS	
BLOCK HEADER/TRAILER RECORD:			
1-8	F8.0	BLOCK INDEX	
9-16	F8.0	INITIAL BIT INDEX	
17-24	F8.0	BLOCK SIZE (BITS)	
25-40	D16.0	EVENT TIME FOR START OF BLOCK INPUT (SECONDS AFTER REFERENCE TIME)	
41-56	D16.0	EVENT TIME FOR START OF BLOCK TRANSFER (SECONDS AFTER REFERENCE TIME)	
USER INFORMATION RECORD:			
1-5	I5	USER INFORMATION FIELD	
6-10	I5	USER INFORMATION FIELD	
		• • •	
76-80	I5	USER INFORMATION FIELD	
END-OF-HISTORY RECORD:			
1-8	F8.0	ZERO OR A NEGATIVE NUMBER	
9-16	F8.0	ZERO	
17-24	F8.0	ZERO	
25-40	D16.0	ZERO	
41-56	D16.0	ZERO	

a. Source User Information File

CHARACTER FIELD	EDIT DESCRIPTOR	CONTENTS	
PREFACE DATA (PART 1):			
1-32	A32	FILE DESCRIPTOR	
PREFACE DATA (PART 2):			
1-64	A64	BATCH IDENTIFIER	
PREFACE DATA (PART 3):			
1-32	A32	SOURCE USER IDENTIFIER	
33-64	A32	DESTINATION USER IDENTIFIER	
PREFACE DATA (PART 4):			
1-4	I4	YEAR	REFERENCE TIME (DATE AT ORIGINATING USER SITE)
5-8	I4	MONTH	
9-12	I4	DAY	
13-16	I4	HOURS	REFERENCE TIME (LOCAL TIME-OF-DAY AT ORIGINATING USER SITE)
17-20	I4	MINUTES	
21-28	F8.0	SECONDS	
BLOCK HEADER/TRAILER RECORD:			
1-8	F8.0	BLOCK INDEX	
9-16	F8.0	INITIAL BIT INDEX	
17-24	F8.0	BLOCK SIZE (BITS)	
25-40	D16.0	EVENT TIME FOR END OF BLOCK TRANSFER (SECONDS AFTER REFERENCE TIME)	
USER INFORMATION RECORD:			
1-5	I5	USER INFORMATION FIELD	
5-10	I5	USER INFORMATION FIELD	
		• • •	
76-80	I5	USER INFORMATION FIELD	
END-OF-HISTORY RECORD:			
1-8	F8.0	ZERO OR A NEGATIVE NUMBER	
9-16	F8.0	ZERO	
17-24	F8.0	ZERO	
25-40	D16.0	ZERO	

b. Destination User Information File

Figure B-3. Format of the source (SUI) and destination user information (DUI) files.

```
SOURCE USER INFORMATION
UW to NTIA.ITS                                                  .2134
UW - eldiente                    NTIA . crestone
      89    3     6        00      00   00.000
            1.     1.      1024. 53298.4310+0        53298.4310+0
061680442816620958708905208370072016705088680443311887018470669818713032101235 9
0911207501263462262110571671308346123601118419476117270078908899186322781817 42
139700347003208055250501010449191842795313477043662376223722100928045282762 0-26
091212395627175012702728202249072701512097872063210831092381901723753027022747 3
153922313327118305790000000000000000000000000000000000000000000000000000000000
      1.      1.           1024  53298.4310+0        53298.4310+0
      2.     1025.         1024. 53298.4780+0        53298.4780+0
0874705788278542223025027188570420620342066980514427949131100263419889094402108 2
146170348026957100210042616737175782544147490622620265095110871526053073481953 1
061792090035010504615051209210736269871309206797274702949421138197650277613425
155270712419016218602296207573063721338706443230050324421589087540660501250292 53
114252062510509140670000000000000000000000000000000000000000000000000000000000
      2.     1025. 1024. 53298.4780+0       53298.4780+0
      3.     2049. 1024. 53298.5240+0       53298.5240+0
135000699720138011043048980254278442645407337032291908529991008340868109394174 61
089952377202509016851080326017190822514209369069942839509829128500351321664193 08
104050692503114051901113008669093942160213611050201927109253108341166121674297 55
106742170919209051730093923893278681001063080508503558052992129825913176521518 7
101612171228333057171024000000000000000000000000000000000000000000000000000000
      3.     2049. 1024. 53298.5240+0       53298.5240+0

                                   ...

      79.    79873.        1024. 53308.7340+0        53308.7340+0
075812325819661140550653918649211401751013233232520171026533108190135725808302 58
136280751628010253823169120909073322748210139210170996600869070902600918662223 52
135950424518511013343114717797190841956315282032260334113927129871462120136119 34
134990647618087012531494701412247542085806705071310999029796313451662103248172 64
097832159802093055241228800000000000000000000000000000000000000000000000000000
      79.    79873.        1024. 55308.7340+0        53308.7340+0
      80.    80897.        1024. 55308.9610+0        53308.9610+0
145192325111790011588168251886105810269880757921388283583037321169167970124617 46
090102177828303013651693909705190862520909242209401988505735065461272524234280 13
098822017210695140380449817849012522718709522226771982221603107062294429406292 99
100432001109202638919306158090939619510140042152810889176350269114549016501445 15
106682370910954019550819200000000000000000000000000000000000000000000000000000
      80.    80897.        1024. 53308.9610+0        53308.9610+0
      -1.    0.            0.          0.0000+0                          0.0000
```

Figure B-4. Example of source user information (SUI) files.

```
DESTINATION USER INFORMATION
UW to NTIA.ITS                                              2134
UW - eldiente                 NTIA . crestone
  89  3     6      00     00     00.000
      1.     1.     1024. 53298.7970+0
06168044281866209587089052083700720167050886804433118870184706698187130321012359
09112075012634622621105716713083461236011184194761172700789088991863227818188742
13979034700320805525050101044919184279531347704366263762237221009280452827629026
09121239562717501270272820224907270152120978720632108310923819017237530270227473
15392231332711830579000000000000000000000000000000000000000000000000000000000000
      1.     1.     1024. 52309/7970+0
      2.     1025.  1024. 53298.9310+0
08747057882785422230250271885704206203420669805144279491311002634198890944021082
14617034802695710021004261673717578254441474906226202650961108715260530734819531
061703480269571002100426150512092107387269871309067972747029421138197650277613425
155270712419016218602296207573063721338706443230050324421589087540660501250295253
11425206251050914067000000000000000000000000000000000000000000000000000000000000
      2.     1025.  1024. 53309.9320+0

                              ...

      79.    79873.       1024.53309.3180+0
075812322581966114055065918649211401751013233232520171026533108190125728082025844
136280751628010253823169120909073322748210139210170996600869070902600918662222352
135950424518511013343114717797190841956315282032603341139271462120136139341359545
134990647618087012531494701412247542085806705071310999029796313451662103241726555
097832115980209305524122880000000000000000000000000000000000000000000000000000000
      79.    79873.       1024. 53309.3180+0
      80.    80897.       1024. 55309.4520+0
145192325111790015881682251886105810269880759213882835830373211691679701246174633
090102177828303013651693909705190862509092422094019885057350654612725242342801333
098822072106951440380449817849012522718709522226771982221603107062294429406292299
100432303011092026389193061580909396195101400421528108891763502691145490165014411
'10668237091094019550819200000000000000000000000000000000000000000000000000000000
0000
      80.    80897.       1024. 53309.4540+0
      -1.    0.           0.    0.0000+0
```

Figure B-5. Example of destination user information (DUI) files.

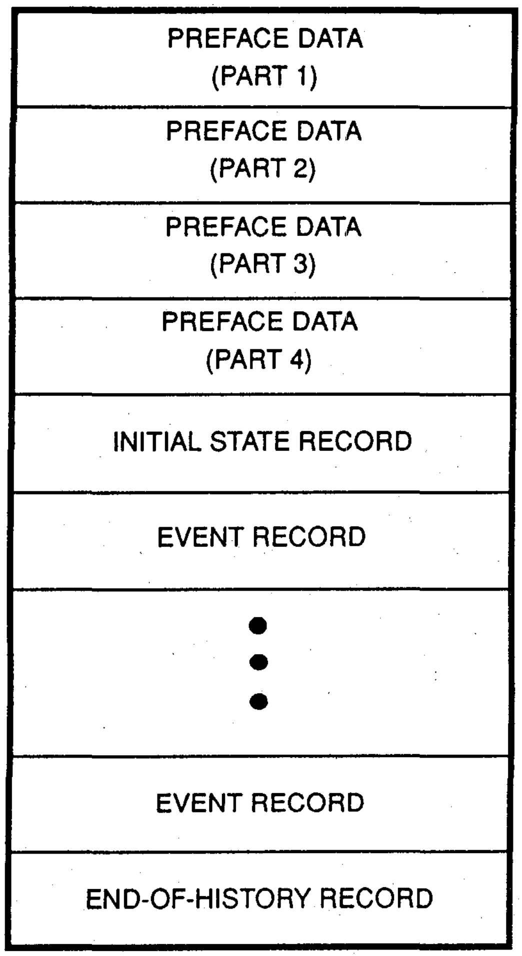

Figure B-6. Structure of the source (SOI) and destination overhead information (DOI) files.

CHARACTER FIELD	EDIT DESCRIPTOR	CONTENTS	
PREFACE DATA (PART 1):			
1-32	A32	FILE DESCRIPTOR	
PREFACE DATA (PART 2):			
1-64	A64	BATCH IDENTIFIER	
PREFACE DATA (PART 3):			
1-32	A32	SOURCE USER IDENTIFIER	
33-64	A32	DESTINATION USER IDENTIFIER	
PREFACE DATA (PART 4):			
1-4	I4	CATEGORY CODE FOR DATA COMMUNICATION SESSION	
5-8	I4	CATEGORY CODE FOR INITIAL DISENGAGEMENT ATTEMPT IN SESSION	
9-12	I4	POINTER TO ORIGINATING USER	
13-16	I4	YEAR	REFERENCE TIME (DATE AT ORIGINATING USER SITE)
17-20	I4	MONTH	
21-24	I4	DAY	
25-28	I4	HOURS	REFERENCE TIME (LOCAL TIME-OF-DAY AT ORIGINATING USER SITE)
29-32	I4	MINUTES	
33-40	F8.0	SECONDS	
INITIAL STATE RECORD:			
1-4	I4	INITIAL COMMUNICATION STATE CODE FOR SOURCE USER	
5-8	I4	INITIAL COMMUNICATION STATE CODE FOR SOURCE HALF-SYSTEM	
EVENT RECORD:			
1-16	D16.0	EVENT TIME (SECONDS AFTER REFERENCE TIME)	
17-20	I4	COMMUNICATION STATE CODE FOR SOURCE USER	
21-24	I4	COMMUNICATION STATE CODE FOR SOURCE HALF-SYSTEM	
25-28	I4	REMOTE INTERFACE EFFECT CODE	
END-OF-HISTORY RECORD:			
1-16	D16.0	A NEGATIVE NUMBER	
17-20	I4	ZERO	
21-24	I4	ZERO	
25-28	I4	ZERO	

a. Source Overhead Information File

CHARACTER FIELD	EDIT DESCRIPTOR	CONTENTS	
PREFACE DATA (PART 1):			
1-32	A32	FILE DESCRIPTOR	
PREFACE DATA (PART 2):			
1-64	A64	BATCH IDENTIFIER	
PREFACE DATA (PART 3):			
1-32	A32	SOURCE USER IDENTIFIER	
33-64	A32	DESTINATION USER IDENTIFIER	
PREFACE DATA (PART 4):			
1-4	I4	CATEGORY CODE FOR DATA COMMUNICATION SESSION	
5-8	I4	CATEGORY CODE FOR INITIAL DISENGAGEMENT ATTEMPT IN SESSION	
9-12	I4	POINTER TO ORIGINATING USER	
13-16	I4	YEAR	REFERENCE TIME (DATE AT ORIGINATING USER SITE)
17-20	I4	MONTH	
21-24	I4	DAY	
25-28	I4	HOURS	REFERENCE TIME (LOCAL TIME-OF-DAY AT ORIGINATING USER SITE)
29-32	I4	MINUTES	
33-40	F8.0	SECONDS	
INITIAL STATE RECORD:			
1-4	I4	INITIAL COMMUNICATION STATE CODE FOR DESTINATION HALF-SYSTEM	
5-8	I4	INITIAL COMMUNICATION STATE CODE FOR DESTINATION USER	
EVENT RECORD:			
1-16	D16.0	EVENT TIME (SECONDS AFTER REFERENCE TIME)	
17-20	I4	REMOTE INTERFACE EFFECT CODE	
21-24	I4	COMMUNICATION STATE CODE FOR DESTINATION HALF-SYSTEM	
25-28	I4	COMMUNICATION STATE CODE FOR DESTINATION USER	
END-OF-HISTORY RECORD:			
1-16	D16.0	A NEGATIVE NUMBER	
17-20	I4	ZERO	
21-24	I4	ZERO	
25-28	I4	ZERO	

b. Destination Overhead Information File

Figure B-7. Format of the source (SOI) and destination overhead information (DOI) files.

```
SOURCE OVERHEAD INFORMATION
UW to NTIA.ITS                                                2115
UW - eldiente                  NTIA - crestone
     2     2     1     89     3     6     00     00     00.00
     1     1
000047234.829D+0              1     1     0
000047236.049D+0              1     1     0
000047236.066D+0              2     3     0
000047275.493D+0              3     2     0
000047275.510D+0              2     2     0
000047278.739D+0              3     2     0
000047278.779D+0              2     3     0
000047279.182D+0              3     2     0
000047279.237D+0              4     4     0
000047280.103D+0              5     4     0
000047280.120D+0              4     5     0
000047281.046D+0              1     1     0
000047340.256D+0              1     1     0
000047341.470D+0              1     1     0
000047341.487D+0              1     3     0
000047380.478D+0              3     2     0
000047380.495D+0              2     2     0
000047383.286D+0              3     2     0
000047383.326D+0              2     3     0
000047383.729D+0              3     2     0
000047383.776D+0              3     3     0
000047384.651D+0              5     3     0
000047384.668D+0              4     5     0
000047385.594D+0              1     1     0
                  ...

000049244.174D+0              1     1     0
000049245.386D+0              1     1     0
000049245.829D+0              2     3     0
000049245.403D+0              3     2     0
000049286.443D+0              2     2     0
000049286.460D+0              3     2     1
000049289.273D+0              2     3     0
000049289.313D+0              3     2     1
000049289.716D+0              4     4     0
000049289.762D+0              5     4     0
000049290.646D+0              4     5     0
000049291.593D+0              1     1     0
      -1.000D+0               0     0     0
```

Figure B-8. Example of source overhead information (SOI) file.

```
SOURCE OVERHEAD INFORMATION
UW to NTIA.ITS                                                2115
UW - eldiente                  NTIA - crestone
      2     2     1     89     3     6     00     00     00.00
      1     1
000047278.120D+0               0     2     3
000047278.620D+0               1     3     2
000047278.623D+0               0     2     3
000047278.640D+0               0     2     2
000047279.545D+0               0     2     3
000047279.562D+0               0     3     2
000047279.563D+0               0     4     5
000047279.607D+0               1     1     1
000047383.143D+0               0     2     3
000047383.146D+0               1     3     2
000047383.163D+0               0     2     3
000047383.090D+0               0     2     2
000047384.090D+0               0     2     3
000047384.107D+0               0     3     2
000047384.109D+0               0     4     5
000047384.153D+0               1     1     1
                  ...

000049244.174D+0               0     2     3
000049245.386D+0               1     3     2
000049245.829D+0               0     2     3
000049245.403D+0               0     2     2
000049286.443D+0               0     2     3
000049286.460D+0               0     3     2
000049289.273D+0               0     4     5
000049289.313D+0               1     1     1
000049289.716D+0               0     2     3
000049289.762D+0               1     3     2
000049290.646D+0               0     2     3
000049291.593D+0               0     2     2
000047384.090D+0               0     2     3
000047384.107D+0               0     3     2
000047384.109D+0               0     4     5
000047384.153D+0               1     1     1
     -1.000D+0                 0     0     0
```

Figure B-9. Example of destination overhead information (DOI) file.

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.
	95-319 (3)		

4. TITLE AND SUBTITLE Performance Evaluation of Data Communication Services: NTIA Implementation of American National Standard X3.141, Volume 3. Data Extraction	5. Publication Date August 1995 6. Performing Organization Code NTIA/ITS.N3
7. AUTHOR(S) Martin J. Miles and David R. Wortendyke	9. Project/Task/Work Unit No.
8. PERFORMING ORGANIZATION NAME AND ADDRESS National Telecommunications and Information Admin. Institute for Telecommunication Sciences 325 Broadway Boulder, CO 80303	10. Contract/Grant No.
11. Sponsoring Organization Name and Address National Telecommunications and Information Admin. Herbert C. Hoover Building 14th and Constitution Avenue, NW Washington, DC 20230	12. Type of Report and Period Covered 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 volume explains how to conduct a data communication session. Specifically, it explains how to determine the commands and expected responses of a protocol (for access and disengagement functions), how to determine the responsibility of the participating entities for producing each reference event, and how to draw a profile of the session (which demonstrates the flow of information between the participating entities and across user/system interfaces). It explains how to create a file containing the commands and expected responses of the protocol, the code that causes the times at which they cross interfaces to be recorded, and a code number that indicates the state of the entities at each interface. This volume also explains how to modify the transmitting program to agree with the protocol. It explains how to create files that support the on-line data extraction software. Specifically, these files are the end user identification files, the clock calibration file, and the protocol file. This volume then explains how to execute a shell script that conducts a test, and how to execute a shell script that processes the test data.

Key words: access; communication state codes; disengagement; reference events; protocol; satellite clock receiver; session profile; user information transfer; user/system interfaces

17. AVAILABILITY STATEMENT	18. Security Class. *(This report)*	20. Number of pages.
☒ UNLIMITED.	Unclassified	88
☐ FOR OFFICIAL DISTRIBUTION.	19. Security Class. *(This page)* Unclassified	21. Price:

*U.S. GOVERNMENT PRINTING OFFICE: 1995-676-490/25108