CAC Technical Memorandum No. 86

CCTC-WAD No. 7508

ARPANET RFC No. 725

NIC No. 38316

An RJE Protocol for a Resource Sharing Network

by

John Day

Gary R. Grossman

Prepared for the

Command and Control Technical Center

WWMCCS ADP Directorate

of the

Defense Communications Agency

Washington, D.C. 20305

under contract

DCAl00-76-C-0088

Center for Advanced Computation

University of Illinois at Urbana-Champaign

Urbana, Illinois 61801

March 1, 1977

Approved for Release - Peter A. Alsberg, Principal Investigator

An RJE Protocol for a Resource Sharing Network

For many users of the ARPANET, an RJE protocol is probably as important in terms of utility as a TELNET (VTP) protocol. In fact, the facilities provided by a TELNET and an RJE protocol are probably of most interest to most users of computer networks. For these users, the net provides a fast, cheap RJE surrogate, just as TELNET provides a telephone surrogate for the timesharing user. The collection (and layers) of protocols that provide these services must be organized to efficiently support a wide variety of applications and user needs. They should not pose an undue software burden on the user.

The "official" NETRJE protocol for the ARPANET has met an underwhelming response from both the user and server community. I believe there are two basic reasons. First, a large commitment of resources is necessary to implement NETRJE. Second, the protocol creates serious security problems.

In order to support the ARPA RJE protocol, a user must implement User Telnet, Server FTP, and User RJE, while a server must implement Server Telnet, User FTP, and Server RJE. In addition when an RJE session is going on all three of these protocol implementations will be executing for most of the life of the session. This could entail considerable burden for some systems. Although it may not be out of line to require a service to shoulder such burdens, it is out of line to require a user to assume them in order to gain a rather basic service. Most user installations are oriented toward meeting their user’s needs not toward implementing large amounts of network software. (In fact one of the better aspects of the previous ARPANET protocol designs was that they attempted to minimize the work for the user. (It must be admitted though that compassion for the user was not the reason for this approach.)

In order to support a "hot line printer" (i.e., a job is automatically printed when it is completed), the user must store his user code and password for the output host at the server host. This, of course, presents a rather severe security problem. Although the ARPANET can not be made totally secure without massive revision, there are some basic precautions that can be taken to protect users from being victimized by every first year Computer Science student with access to the net.

The RJE protocol proposed here tries to mitigate the implementation problems and security problems. The protocol is designed to provide three levels of service. A user or server has the perogative to implement the protocol at whatever level their resources allow. The service can then be upgraded to cleaner or more sophisticated approaches when and if the opportunity arises.

This protocol is described in terms of the ARPANET. Several aspects of

the design (such as the reply structure) were made to coincide with existing ARPANET conventions. This was done to facilitate understanding and limit the discussion to the protocol itself. Although the protocol is described in ARPANET terms, it should be applicable to other network environments.

This paper is not considered to be complete in every detail. It was written primarily to elicit comments from the network community and to measure the desire of the community to adopt such a procedure. We have tried to describe enough of the protocol so that the reader can get an idea of how things are to work without getting bogged down in the detail that would be necessary for implementation. Below is an outline of the final protocol document as presently conceived. Sections marked with an asterisk are to be provided later.

Part I THE NETRJE MODELS

This section describes the proposed NETRJE protocol in a narrative form. A formal definition will be included in Part II after review. The narrative should provide the general reader with the flavor of the protocol without getting bogged down in unnecessary detail. The proposed NETRJE protocol provides three different models for job submission and retrieval. The three models can be characterized as 1) RJE using Telnet only, 2) RJE using Telnet and Data Transfer, and 3) RJE using FTP. This approach provides flexibility for both implementors and users. User and server sites constrained by manpower or machine resources may implement only the simpler models. The user may use the different models separately or in any consistent combination which best suits his requirements and convenience. Servers should assume that the minimal implementation of a more sophisticated model includes the minimal implementations of all less sophisticated models. (There are, however, certain minimal requirements that must be supported.) This secton will discuss each of these models in turn, and show each one can be used to provide a useful network RJE functon.

This protocol does not contain the security difficulties of the present protocol. This has been avoided by requiring that the burden of implementing the "hot line printer" or "hot card reader" be put on the user system. Thus, those systems which desire such a facility may still support it. The user implementaton will be slightly more complicated. The trade-off is the increased security of the protocol.

End-to-end protocols are assumed to be available and to provide an ordered, error free bit stream to the RJE protocol. It is also assumed that a suitable virtual terminal protocol such as Telnet, is used to format the control connection.

RJE Using Only Telnet (VTP)

The intent of this model is, bluntly, to provide an official "quick and dirty" form of the protocol. Many organizatons, both users and servers, are often confronted with problem of providing a service quickly or within very tight budgetary constraints. This model is intended for these situations. With this model, the user is required only to be able to establish a Telnet connection via the RJE contact socket. Commands, replies, and data are all sent over the Telnet connecton. Card input or printer output has the appearance of coming from or going to the user’s terminal. The user’s system may allow output to be diverted from the terminal to another device such as the line printer. The technique of diverting terminal output was used with great success in the MARK I ANTS

systems where various devices were not assigned socket numbers as in a TIP. This technique is also useful for hosts that allow program access to the network only through the user's Telnet connection. This situation may exist in the early phases of a server's availability to the network. When data is transferred in this mode an end-of-data marker will be sent to aid the receiving host in determining when to stop diverting the data. This model will have to handle the problems of data traveling on a connection essentially meant for control. The use of this data transfer mechanism is intended as an intermediate measure required by limited resources. For now we let it stand that the designers are aware of the problems inherent in embedding commands or replies in the data stream. We will leave the exact resolution of the problem to the formal definition.

This proposed NETRJE protocol uses a schedule verb, SCHED for job submission. For this model, there are two forms of SCHED that are relevant. First, there is the "SCHED <server pathname>" form. This command indicates to the server that there exists at the server site a file with all necessary job control information and data to define a job. The server will then attempt to place the job in the job queue and reply to the user indicating success or failure and possibly a job-id. This job-id will be used when inquiring about the job status or retrieving the job's output.

When the job finishes, the server will take one of two actions:

a) if the user is still logged in, the server will send a reply notifying the user of his job completion; or,

b) if the user is not logged in, the server will save the status of the job which may later be interrogated via the STATUS command (see below).

The otherform of SCHED of relevance to this model has the syntax:

SCHED INPUT <CRLF><data><CRLF>.<CFLF>

This allows the user to sit down at a terminal and type his own job control or possibly a program. It also allows those users whose local systems provide a facility to transmit files with User TELNET to transmit user input job fles in this way. The RJE Server would insert the job into the local job stream, returning the proper indication of success or failure along with identification of the job.

Just as the SCHED command provides several ways for job submission, the OUTPUT command provides several options for retrieving output. The form

OUTPUT<job-id><server pathname>DISCARD

is sent to the server to initiate the output to the user's site according to output specifications defined by previous OUTDEF commands (see below). The optional DISCARD argument to the OUTPUT command indicates, if present, that the file is to be destroyed after transmission has completed successfully.

The OUTDEF command for a job may be sent at any time after the job has been scheduled and before it is retrieved using the OUTPUT command. This command will specify the parameters necessary to effect the transfer of the output to the user or to define the disposition of the output. We realize that the OUTDEF <job-id><server pathname> command (indicating that output is to be placed in a file described by the pathname) may be difficult for some systems to implement. These systems would merely respond negatively indicating their inability to perform the function.

A scenario is now in order to illustrate the model. The user has logged in to Multics and is ready to submit an RJE job in the following way (XXX will denote the as yet unspecified reply code for the reply):

SCHED MY-JOB>TREK

The system responds with a reply indicating the job has been submitted successfully and returns a job-id, say XA1423.

XXX JOB XA1423 was successfully submitted.

At some later time a message appears.

XXX JOB XA1423 has completed.

The user or user process now sends OUTDEF XA1423 TELNET indicating that the job should be sent on the TELNET connection. A reply returns

XXX last command successful.

The user now sends

OUTPUT XA1423

and the server replies with

XXX Output ready. Type an empty line when ready.

The user then sends an empty line when he is read to receive the output.

This exchange allows the user to effect output diversion at his local site if necessary after he has confirmed the server is ready.

If the user had not wished to wait on his output and had logged off after getting the successful submission, the next time the user logged in he could inquire as to the status of the job or all jobs under his usercode and then proceeded to output any or all of them.

RJE with TELNET and Data Transfer

The previous model provided a minimal implementation for NETRJE. This model provides better data transfer facilities without requiring an FTP implementation. This model requires no new commands, but does manipulate connections differently, so that data is not required to flow on the command connection (see Fig. 2). Data is sent on separate default connections (unless otherwise specified) as in the CCN NETRJS protocol. However, for this protocol the defaults used will be the same offsets from the control connection as those in FTP.

The use of this model is indicated to the Server by either the INDEF command or a SCHED command with no arguments. The INDEF command allows the user to specify a socket other than the default socket as the source of the input. On receipt of the SCHED or INDEF indicating this technique is to be used, the Server will attempt to connect to the appropriate socket. If a SCHED command was sent, the user protocol interpreter could start sending cards as soon as the data connection is established. (It is assumed that the user interface has indicated to the RJE protocol interpreter where the cards are to come from.) If the command was INDEF, then the Server will not start reading until the SCHED is received. Similarly, when the output is ready, either an OUTDEF or OUTPUT command is sent to set up and start the printing. The INDEF and OUTDEF commands used with this mode will also allow moving data to or from a TIP or printer.

This model requires definiton of actual data transfer formats for the reader and printer lines. We propose that the formats and connection schemes of the present FTP be adopted. This solution has the advantage of not requiring extra coding efforts for users with FTP implementations and may allow them to organize their FTP implementations and may allow them to organize their FTP and NETRJE implementations in such a way as to take advantage of common algorithms. One might easily confuse this solution with a revival of the Data Transfer Protocol. Some thought on a more rigorous definition of a Data Transfer Protocol for the common use of FTP and RJE might be worthwhile in the future.

Let us consider a scenario.

The user wishes to submit a card deck to the Server. He then types

SCHED<CRLF>

The Server opens a connection to the user's default card reader socket while sending a reply to the user on the control connection.

XXX attempting connection to card reader.

When the connection is opened, another reply:

XXX transfer started

and when completed:

XXX JOB XA 1423 was successfully submitted.

When the job completes and the completion message is sent to the user, he may wish to send the output to his TIP printer on socket Y. He will then type

OUTDEF XA1423 255, Y (255 being his host address).

The Server will then attempt to connect to the socket and will reply

XXX printer connection successful.

When the user has satisfied himself all is in readiness, he will type

OUTPUT XA1423

and the Server will start sending and reply to the user

XXX print started.

When the transfer is complete the Server will close the data connection and send an appropriate reply.

NETRJE Using FTP

This model (illustrated in Fig. 3) uses FTP to effect the transfer of the files. It may be easier for some systems to use this sort of model for more sophisticated RJE systems. This is especially true if the users desire to take input from the local file system or to send output to the local file system rather than from an actual card reader or to an actual line printer. Although using the local file system is not prohibited by the Data Transfer model, it may be easier to approach through FTP. Using FTP with NETRJE also allows the utilization of the FTP server-server transfer mechanism to generate input from or direct output to a third host.

The only new facility required by this model are the commands INPATH and OUTPATH. When using FTP to transfer input to the Server, the user must know where to send the job so that it enters the job stream. The INPATH command returns as a reply such a legal pathname. Thus the scenario for job submission is as follows: The user sends an INPATH command; the Server responds with a legal Server pathname for the user. The user process starts sending the input to the file using FTP. When transfer is complete, the user sends a SCHED <server pathname> command. When the job has finished, the pathname created for the user may or may not destroy the input file. The OUTPATH command is similarily used to identify the pathname for the output, so that it may be retrieved by FTP. Some systems may define file names in such a way that the user may derive them from the parameters of his job.

Note on Replies

In all of the above examples we have refrained from defining actual reply codes. The intent is to use the same reply structure, and where appropriate the same numbers, as described in RFC 640 "Revised FTP Reply Codes".

Protocol Measurement

An integral part of any good protocol definition is a set of measurements to allow evaluation of both the protocol and its implementation. This provides two functions: 1) It allows the protocol designer to evaluate the protocol and make improvements. 2) It allows the user of the protocol to know how expensive it is and to demand improvements. The proposed NETRJE protocol provides two sets of measures - one for a particular session and one for overall performance. These measurements may be elicited by the MEASURE command which will take an argument with three values: JOB (job statistics and cost measurements), SESSION (measurements taken for this sesson), and GLOBAL

(overall measurements of the performance of the protocol and its implementation). The command will return the measurements in a fixed format reply.

The measurements reported for a job are:

1. CPU time,

2. I/O operations,

3. storage space time product,

4. job cost in dollars,

5. elapsed time the job waited before being executed, and

6. elapsed time for the job to execute.

The measures taken from a sesson are:

1. number of bits transferred,

2. transmission rate of input or output transfers,

3. the amount of CPU time, storage space-time product, and I/O operations for the session.

4. cost in dollars and cents.

The measures to be taken globally are:

1. frequency of commands and possibly command forms,

2. model frequency (which submission/retrieval model used),

3. transmission mode frequency,

4. total number of sessions,

5. transmission rate: average, std. deviation, upper and lower bounds (also by transmission mode),

6. cpu time, storage space-time product, and I/O operations for both the protocol and jobs submitted: average, std. deviation, and upper and lower bounds (overall as well as by model, transfer mode, and file size). (The reason for including job statistics here is so that

management and systems personnel have some indication how the facility is being used.)

It is clear that it may be difficult to acquire some measures (such as transmission rate) when NETRJE is using FTP. This is unavoidable since FTP is not metered. The most straightforward solution is also to meter FTP (hint). For the final definition a close look will be given to the subset that should be required. Comments are welcome. However, we believe strongly that it is very important to know how a facility like this is used as well as how well it performs.

Part II. Preliminary Definition of NETRJE Commands

For purposes of discussion this section gives a very preliminary definition of the NETRJE commands and their replies. The intent is to give a brief but not exhaustive definition of each command and its major replies to give the flavor of the protocol. We do not do this to discourage nit-picking by critics, since we may actually overlook the obvious on occasion, but merely to expedite the writing of this paper.

The reply scheme will follow the model of the revised FTP reply codes described in RFC 640.

Access Control

USER <usercode>

PASS <password>

ACCT <account>

These perform the normal functions to log the user into the system. The replies to them are the standard ones in FTP. It was never clear why "account" was not included in the old NETRJE. Presumably, if it's necessary for an FTP or Telnet user, it will be necessary for an RJE user.

REINIT

This command reinitializes the state of the NETRJE server process so that it is ready for a new user. If the transfer of data is in progress for the previous user, it will be allowed to complete.

ABORT

This command is used to abort the transfer of data. This command is meaningful to the Server only if the data is being transferred over the Telnet connection or the default data sockets. If FTP is being used, the execution of this command is the responsibility of the USER NETRJE process.

BYE

This command causes the Server to log out the user and close the Telnet connection. If the transfer of data is in progress, the action of the command will be delayed until the transfer is complete.

SCHED <input part><output part>

<input part>::= <empty>|<server pathname> [DISCARD]

INPUT <CRLF> <text> <CRLF>.<CRLF>

output part ::= <empty>|<server pathname>[DISCARD]

server pathname ::= {locally recognizable string of characters terminated by an ASCII NULL}

This command causes the input described by the <input part> to be entered into the RJE job stream and the output produced to be disposed of according to the <output part>. The null condition for either argument implies that the information has been previously specified or is the default.

For the <input part>, the <empty> may imply two actions. If an INDEF command has previously specified a <server pathname>, input to the job stream is taken from the file indicated by the file name. If the INDEF command has specified that the input is to come from a CCN-like data transfer socket, the SCHED <empty> command is the signal for the Server to start reading data.

The DISCARD modifier, if present, indicates that the file should be discarded after it has been transmitted or it has been received and executed. If the input stream is to be sent on the Telnet connection, the source may be a local device or a human user. This facility is provided for mini-hosts that can’t use one of the other techniques and for the user who wishes to enter job control directly at his terminal.

The empty for output specifies either the primary output file of the job (the default) or a previously specified server pathname (OUTDEF command).

Successful replies to this command should indicate any job-id assigned by the local RJE system along with other status informaton. Failure would be because files did not exist, access was denied, etc.

OUTPUT <output spec>

<output spec>::= <job-id><xmsn part>|<job-id><server pathname>

<xmsn part>::= <empty>| /<IO params>

<IO params>::= <xmsn params>, <dest>

This command indicates to the Server what output is to be sent to the user, how it is to be sent, and to whom. The <IO params> part will allow the specification of a host and socket so that output may be sent to a TIP printer, or alternatively sent on the Telnet connection or to the default data sockets. This argument also specifies the format and representation of the data.

When the Server receives this command, it will proceed to transmit the output to the host in the prescribed manner. The reply structure of this command will depend on how the output is moved and will be discussed in more detail later.

INPATH

This command returns to the user a legal pathname at the Server. The user may then transfer his input to this pathname for eventual submission to the RJE facility.

OUTPATH

This command performs a similar function to INPATH.

DISCARD <job-file-id> | <server pathname>

This allows the user to destroy input or output files associated with a job.

INDEF <job-id><I/O params>

OUTDEF <job-id><I/O params>

These commands allow the user to specify the parameters necessary to send input or retrieve output. This command specifies how the data will be transferred and specifies format, etc.

CANCEL <job-id>

This command allows a job to be cancelled from the RJE job stream.

STATUS <status arg>

status arg ::= <empty>|<user id>|<job-id>|<job-id><blank><job-file-id>

This command allows the user to determine the status of the RJE session, all jobs under his usercode, a specific job, or the output of a specific job.

ALTER <job-id><site specific option>

SITE <site specific option>

These commands allow site specific commands to be passed to the Server RJE system. The ALTER command is intended to effect specific jobs, while the SITE command is used for commands of more global effect. They could be merged into one.

OP <operator message>

This command allows messages to be sent to the operator at the Server site.

Reply Codes for the Proposed NETRJE

The reply codes for this protocol will follow the model proposed for the new FTP specificaton in RFC 640. As a reminder we insert the pertinent information from that RFC:

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) For implementations where simultaneous monitoring is difficult, this type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used, the user does not change his file access or user name, the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don’t fit any functional category, and unimplemented or superfluous commands.

x1z Information - These are replies to requests for information, such as status or help.

x2z Connection - Replies referring to the Telnet and data connections.

x3z Authentication and accounting - Replies for the logon process and accountng procedures.

x4z Unspecified as yet.

x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is suggestive, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, should strictly follow the specifications. That is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

Below is a list of replies ordered by reply code. Some new replies have been added for RJE; these are marked by asterisks to aid the reader. Following this list is a list of commands with the replies that are possible for that command. This list is not considered complete or final; as usual comments are welcomed.

110 Restart marker reply,

In this case the text is exact and not left to the particular implementation; it must read:

MARK yyyy = mmmm

where yyyy is user-process data stream marker, and mmmm is Server's equivalent marker. (Note the spaces between the markers and "=".)

120 Service ready in nnn minutes

125 Data connection already open; transfer starting

150 File status okay; about to open data connection

200 Command okay

202 Command not implemented, superfluous at this site

211 System status, or system help reply

212 Directory status

213 File status

214 Help message (on how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.)

*215 RJE general status reply

*216 job status reply

*217 RJE user's jobs status reply

220 Service ready for new user

221 Service closing TELNET connecton (logged off if appropriate)

225 Data connection open; no transfer in progress

226 Closing data connection; requested file action successful (for example, file transfer or file abort.)

227 Entering [passive, active] mode

230 User logged in

250 Requested file action okay, completed

*260 Job <job-id> has completed

*261 Output ready. Type an empty line when ready

*262 Job <job-id> IS ALLOCATED pathname

*263 Job <job-id> cancelled as requested

*264 Job <job-id> altered as requested to state status

331 User name okay, need password

332 Need account for login

350 Requested file action held in abeyance, pending further information

354 Start mail input; end with CRLF, CRLF

*360 Job <job-id> successfully submitted

421 Service not available, closing Telnet connecton. (This may be a reply to any command if the service knows it must shut down.)

425 Can't open data connection

426 Connection trouble, closed; transfer aborted

230

202

530

500, 501, 503, 421

332

ACCT

230

202

530

500, 501, 503, 421

BYE

221

500

REINIT

120

220

220

421

500, 502

ABORT

225, 226

500, 501, 502, 421

STATUS

211, 212, 213

450

500, 501, 502, 421, 530

HELP

211, 214

500, 501, 502, 421

SOCK

200

500, 501, 421, 530

BYTE, MODE, TYPE, STRU

200

500, 501, 504, 421, 530

SCHED

360 JOB <job-id> successfully submitted

260 Job <job-id> has completed.

125 500

425, 426 501

226 504, 532

OUTPUT

261 Output ready. Type an empty line when ready.

125 Transfer started

226 500

425, 426 501

110

OUTDEF

225 Data connection opened, no transfer in progress.

425 500

501

504

INDEF

225 500

425 501

504

INPATH/OUTPATH

262 JOB <job-id> IS ALLOCATED PATHNAME >

500 504

501

DISCARD

250 500 530

450 501

550 502

421

CANCEL

263 Job <job-id> Cancelled as requested

500 504

501

502

563 Job <job-id> is not known to the system

564 Requested Alteration is not permitted for the specific job.

STATUS

215 RJE general status reply

216 RJE job status reply

217 RJE user’s jobs status reply

500, 501, 502, 504

ALTER

264 Job <job-id> altered as requested to state status

500, 501, 502, 504 563, 564

SITE

200

500, 501, 502, 504

OP

200

500, 501, 502, 504

References

Braden, R.

1971 "Interim NETRJS Specifications", RFC 189, NIC 7133.

Bressler, R.; Guida, R.; and McKenzie, A.

1972 "Remote Job Entry Protocol", RFC 407

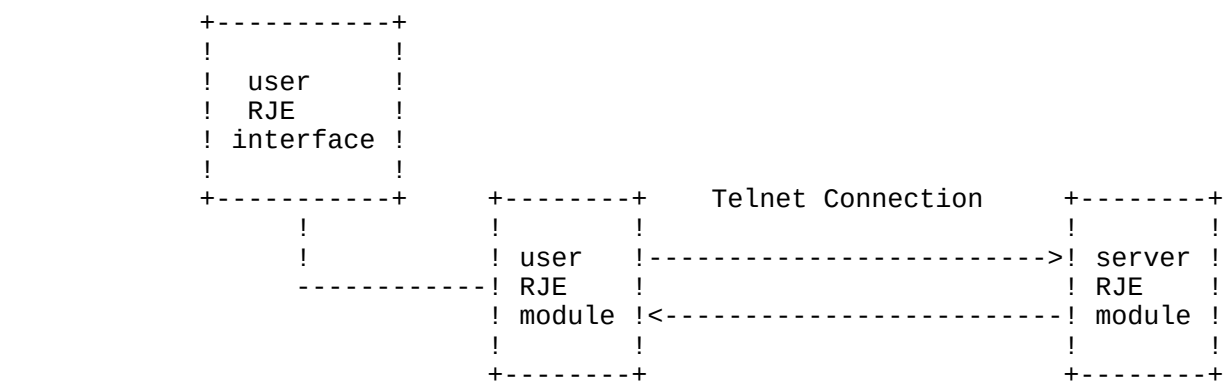
Neigus, N.

1973 "The File Transfer Protocol", RFC 542.

Neigus, N.; Pogran, K.; and Postel, J.

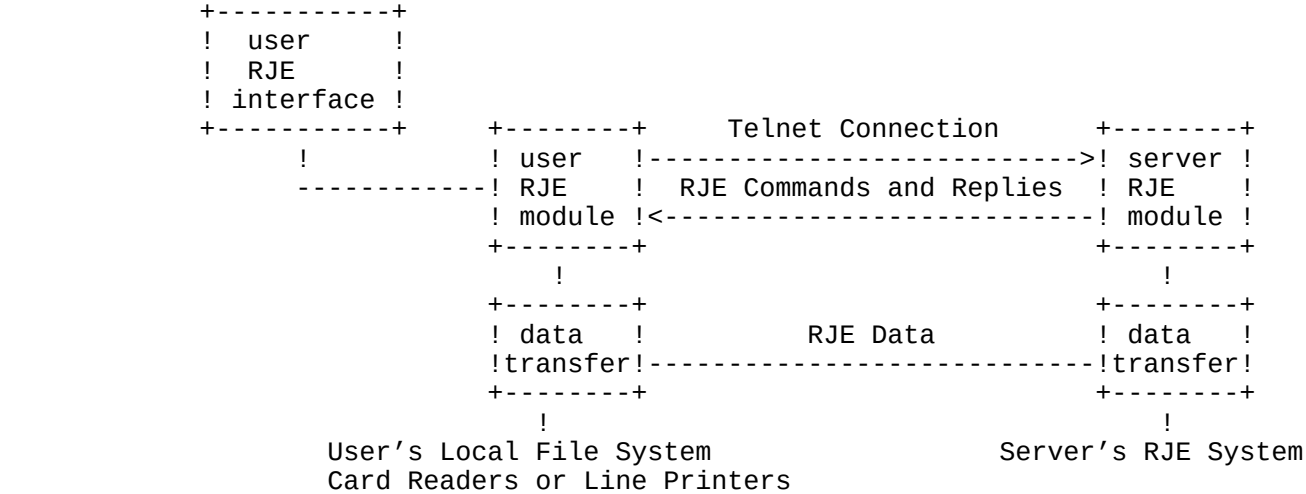
1974 "A New Schema for FTP Reply Codes", RFC 640.

Figures



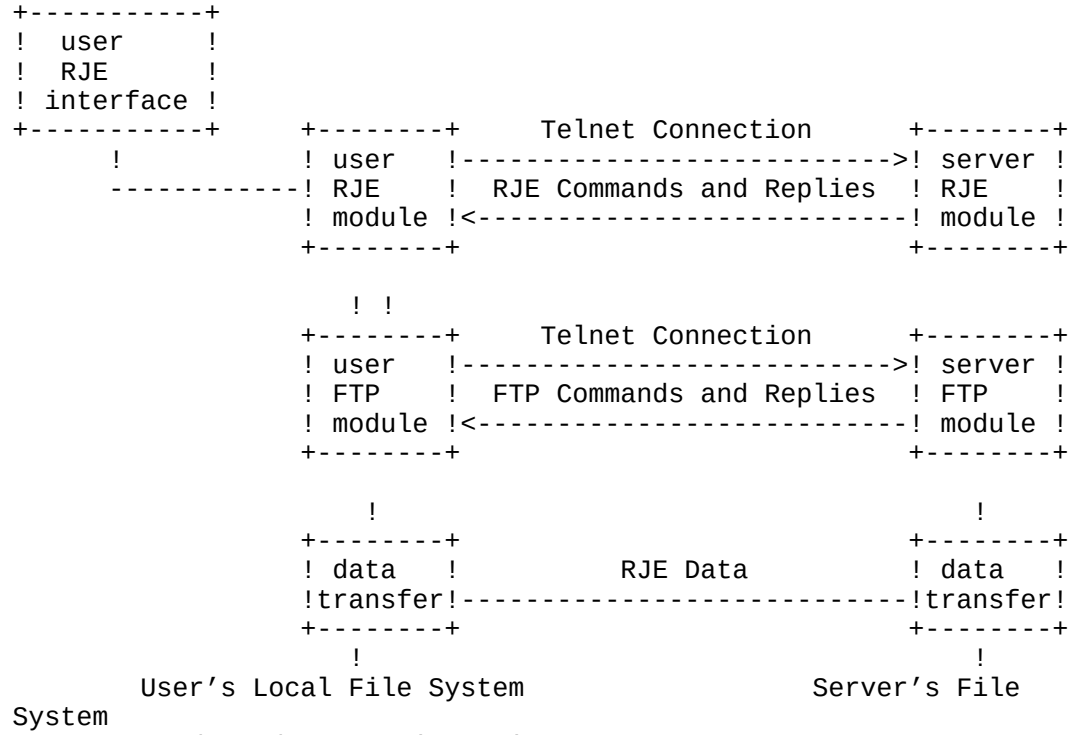
RJE Using Only Telnet

Figure 1.



RJE Using a Separate Data Connection

Figure 2.



RJE Using FTP

Figure 3.