Network Working Group
Request for Comments: 189
Obsoletes: RFC 88 (NIC 5668)
NIC 7133
Category: D

R. T. Braden
UCLA/CCN

15 July 1971

INTERIM NETRJS SPECIFICATIONS

The following document describes the operation and protocol of the remote job entry service to CCN's 360 Model 91. The interim protocol described here will be implemented as a production service before the end of July. Two host sites (Rand and UCLA/NMC) have written user processes for the interim NETRJS, based on the attached document. Questions on it should be addressed to CCN's Technical Liaison.

It is anticipated that the interim protocol will be superseded in a few months by a revised NETRJS, but the changes will be minor. The revision will bring the data transfer protocol of NETRJS into complete conformity with the proposed Data Transfer Protocol DTP (see RFC #171). The present differences between the DTP and NETRJS protocols are:

(a) The format (but not the contents) of the 72 bit transaction header of NETRJS must be changed to conform with DTP.

(b) The End-of-Data marker must be changed from X'FE' to X'B40F'.

(c) The initial "modes available" transaction of DTP must be added.

(d) Some of the DTP error codes will be implemented.

No other protocol changes are presently planned, although some may be suggested by operating experience with the interim protocol. When the revised protocol has been fully specified, it will be implemented with different ICP sockets than the interim protocol. This will allow a site which wants to start using CCN immediately to convert his protocol at leisure.

Some possible future extensions to NETRJS which have been suggested are:

(1) A 7-bit ASCII option of data transfer connections, for the convenience of PDP-10s.

(2) A "transparency" mode for input from ASCII remote sites, to allow the transmission of "binary decks" (object decks) in the job stream from these sites.

(3) More than one simultaneous virtual card read, printer, and punch stream to the same virtual terminal.

Comments on the utility of these proposals or others for your site would be appreciated.

Robert T. Braden
Technical Liaison
UCLA/CCN
(213) 825-7518

REMOTE JOB ENTRY TO UCLA/CCN FROM THE ARPA NETWORK

(Interim Protocol)

A. Introduction

NETRJS is the protocol for the remote job entry service to the 360 Model 91 at the UCLA Campus Computing Network (CCN). NETRJS allows the user at a remote host to access CCN's RJS ("Remote Job Service") sub-system, which provides remote job entry service to real remote batch (card reader/line printer) terminals over direct communications lines as well as to the ARPA Network.

To use NETRJS, a user at a remote host needs a NETRJS user process to communicate with one of the NETRJS server processes at CCN. Each active NETRJS user process appears to RJS as a separate (virtual) remote batch terminal; we will refer to it as a VRBT.

A VRBT may have virtual card readers, printers, and punches. Through a virtual card reader a Network user can transmit a stream of card images comprising one or more OS/360 jobs, complete with Job Control Language, to CCN. These jobs will be spooled into CCN's batch system (OS/360 MVT) and run according to their priority. RJS will automatically return the print and/or punch output images which are created by these jobs to the virtual printer and/or card punch at the VRBT from which the job came (or to a different destination specified in the JCL). The remote user can wait for his output, or he can sign off and sign back on later to receive it.

The VRBT is assumed to be under the control of the user's teletype or other remote console; this serves the function of an RJS remote operator console. To initiate a NETRJS session, the remote user must execute the standard ICP (see RFC #165) to a fixed socket at CCN. The result is to establish a duplex Telnet connection to his console, allowing the user to sign into RJS. Once he is signed in, he can use his console to issue commands to RJS and to receive status, confirmation, and error messages from RJS. The most important RJS commands are summarized in Appendix D.

Different VRBT's are distinguished by 8-character terminal id's. There may be more than one VRBT using RJS simultaneously from the same remote host. Terminal id's for new VRBT's will be assigned by CCN to individual users or user groups who wish to run batch jobs at CCN (contact the CCN Technical Liaison for details).

B. Connections and Protocols

Figure 1 shows conceptually the processes and protocols required to use NETRJS. The operator console uses a duplex connection under the Telnet third-level protocol (see RFC #158). The actual data transfer streams for job input and output are handled over separate simplex connections using a data transfer protocol.

We will use the term channel for one of these NETRJS connections, and designate it input or output with reference to CCN. Each data transfer channel is identified with a particular virtual remote device -- card reader, printer, or punch. The data transfer channels need be open only while they are in use, and different channels may be used sequentially or simultaneously. NETRJS will presently support simultaneous operation of a virtual card reader, a virtual printer, and a virtual punch (in addition to the operator console) on the same VRBT process. RJS itself will support more than one reader, printer, and punch at each remote terminal, so the NETRJS protocol could easily be expanded in the future to allow more simultaneous I/O streams to each Network user.

The remote user needs a local escape convention so he can send commands directly to his VRBT process. These local VRBT commands would allow selection of the files at his host which contain job streams to be sent to the server, and files to receive job output from the server. They would also allow the user to open data transfer channels to the NETRJS server process, and to close these connections to free buffer space or abort a transmission.

When a VRBT starts a session, it has a choice of two ICP sockets, depending upon whether it is an ASCII or an EBCDIC virtual terminal. An EBCDIC virtual terminal transmits and receives its data as transparent streams of 8 bit bytes (since CCN is an EBCDIC installation). It is expected that a user at an ASCII installation, however, will want his VRBT declared ASCII; RJS will then translate the input stream from ASCII to EBCDIC and translate the printer stream back to ASCII. This will allow the user to employ his local text editor for preparing input to CCN and for examining output. The punch stream will always be transparent, for outputting "binary decks".

It should be noted that the choice of code for the operator console connections is independent of declared terminal type; in particular, they always use ASCII under Telnet protocol, even from an EBCDIC VRBT.

NETRJS protocol provides data compression, replacing repeated blanks or other characters by repeat counts. However, when the terminal id is assigned by CCN, a particular network terminal may be specified as using no data compression. In this case, NETRJS will simply truncate trailing blanks and send records in a simple "op code-length-data" form, called truncated format.

C. Starting and Terminating a Session

The remote user establishes a connection to RJS via the standard ICP from his socket U to socket 11 [sub] 10 (EBCDIC) or socket 13 [sub] 10 (ASCII) at host 1, IMP 1. If successful, the ICP results in a pair of connections which are in fact the NETRJS operator control connections.

Once the user is connected, he must enter a valid RJS signon command ("SIGNON terminal-id") through his console. RJS will normally acknowledge signon with a console message; however, if RJS does not recognize the terminal-id or has no available Line Handler for the Network, it will indicate refusal by closing both operator connections. If the user attempts to open data transfer connections before his signon command is accepted, the data transfer connections will be refused by CCN with an error message to his console.

Suppose the operator input connection is socket S at CCN; S is the even number sent in the ICP. Then the other NETRJS channels have sockets at CCN with fixed relation to S, as shown in the table below. Until there is a suitable Network-wide solution to the problem of identity control on sockets, NETRJS will also require that the VRBT process use fixed socket offsets from his initial connection socket U. These are shown in the following table:

Channel		CCN Socket (Server)	Remote Socket (User)	
Telnet	/ Remote Operator Console Input	S	U + 3	\
	\ Remote Operator Console Output	S + 1	U + 2	/ Telnet
Data	/ Card Reader #1	S + 2	U + 5	
Transfer	< Printer #1	S + 3	U + 4	
	\ Punch #1	S + 5	U + 6	

Once the user is signed on, he can open data transfer channels and initiate input and output operations as explained in the following sections. To terminate the session, the remote user may close all connections. Alternatively, the user may enter a SIGNOFF command through his console; in this case, RJS will wait until the current job output streams are complete and then itself terminate the session by closing all connections.

D. Input Operations

A job stream for submission to RJS at CCN is a series of logical records, each of which is a card image. A card image may be at most 80 characters long, to match the requirements of OS/360 for job input. The user can submit a "stack" of successive jobs through the card reader channel with no end-of-job indication between jobs; RJS recognizes the beginning of each new job by the appearance of a JOB card.

To submit a job or stack of jobs for execution at CCN, the remote user must first open the card reader channel. He signals his VRBT process to issue Init (local = U + 5, foreign = S + 2, size = 8). NETRJS, which is listening on socket S + 2, will normally return an RTS command, opening the channel. If, however, it should happen that all input buffer space within the CCN NCP is in use, the request will be refused, and the user should try again later. If the problem persists, call the Technical Liaison at CCN.

When the connection is open, the user can begin sending his job stream using the protocol defined in Appendix A. For each job successfully spooled, the user will receive a confirming message on his console. At the end of the stack, he must send an End-of-Data transaction to initiate processing of the last job. NETRJS will then close the channel (to avoid holding buffer space unnecessarily). At any time during the session, the user can re-open the card reader channel and transmit another job stack. He can also terminate the session and sign on later to get his output.

The user can abort the card reader channel at any time by closing the channel (his socket S + 2). NETRJS will then discard the last partially spooled job. If NETRJS finds an error (e.g., transaction sequence number error or a dropped bit), it will abort the channel by closing the connection prematurely, and also inform the user via his console that his job was discarded (thus solving the race condition between End-of-Data and aborting). The user needs to retransmit only the last job. However, he could retransmit the entire stack (although it would be somewhat wasteful) since the CCN operating system enforces job name uniqueness by immediately "flushing" jobs with names already in the system.

If the user's process, NCP, or host, or the Network itself fails during input, RJS will discard the job being transmitted. A message informing the user that this job was discarded will be generated and sent to him the next time he signs on. On the other hand, those jobs whose receipt have been acknowledged on the operator's console will not be affected by the failure, but will be executed by CCN.

E. Output Operations

The user may wait to set up a virtual printer (or punch) and open its channel until a STATUS message on his console indicates output is ready; or he may leave the output channel(s) open during the entire session, ready to receive output whenever it becomes available. He can also control which one of several available jobs is to be returned by entering appropriate operator commands.

To be prepared to receive printer (or punch) output from his jobs, the user site issues Init (local = U + 4 (U + 6), foreign = S + 3 (S + 5), size = 8), respectively. NETRJS is listening on these sockets and should immediately return an STR. However, it is possible that because of software problems at CCN, RJS will refuse the connection and a CLS will be returned; in this case, try again or call the Technical Liaison.

When RJS has output to send to a particular (virtual) terminal and a corresponding open output channel, it will send the output as a series of logical records using the protocol in Appendix A. The first record will consist of the job name (8 characters) followed by a comma and then the ID string from the JOB card (if any). In the printer stream, the first column of each record will be an ASA carriage control character (see Appendix C); the punch output stream will never contain carriage control characters.

NETRJS will send an End-of-Data transaction and then close an output channel at the end of the output for each complete batch job; the remote site must then send a new RFC (and ALL) to start output for another job. This gives the remote site a chance to allocate a new file for each job without breaking the output within a job. If the user at the remote site wants to cancel (or backspace or defer) the output of a particular job, he enters appropriate RJS commands on the operator input channel (see Appendix D).

A virtual printer in NETRJS has 254 columns, exclusive of carriage control; RJS will send up to 255 characters of a logical record it finds in a SYSOUT data set. If the user wishes to reject or fold records longer than some smaller record size, he can do so in his VRBT process.

If RJS encounters a permanent I/O error in reading the disk data set, it will notify the user via his console, skip forward to the next set of system messages or SYSOUT data set in the same job, and continue. In the future, RJS may be changed to send a Lost Data marker within the data stream as well as a console message to the user. In any case, the user will receive notification of termination of output data transfer for each job via messages on his console.

If the user detects an error in the stream, he can issue a Backspace (BSP) command from his console to repeat the last "page" of output, or a Restart (RST) command to repeat from last SYSOUT data set or the beginning of the job, or he can abort the channel by closing his socket. If he aborts the channel, RJS will simulate a Backspace command, and when the user re-opens the channel the job will begin transmission again from an earlier point in the same data set. This is true even if the user terminates the current session first, and re-opens the channel in a later session; RJS saves the state of its output streams. However, before re-opening the channel he can defer this job for later output, restart it at the beginning, or cancel its output (see Appendix D). Note that aborting the channel is only effective if RJS has not yet sent the End-of-Data transaction.

If the user's process, NCP, or host, or the Network itself fails during an output operation, RJS will act as if the channel had been aborted and the user signed off. In no case should a user lose output from NETRJS.

Appendix A

Data Transfer Protocol in NETRJS

1. Introduction

The records in the data transfer channels (for virtual card reader, printer, and punch) are generally grouped into _transactions_ preceded by headers. The transaction header includes a sequence number and the length of the transaction. Network byte size must be 8 bits in these data streams.

A transaction is the unit of buffering within the Model 91 software. Internal buffers are 880 bytes. Therefore, CCN cannot transmit or receive a single transaction larger than 880 bytes. Transactions can be as short as one record; however, those sites which are concerned with efficiency should send transactions as close as possible to the 880 byte limit.

There is no necessary connection between physical message boundaries and transactions ("logical messages"); the NCP can break the "logical message" arbitrarily into physical messages. At CCN we will choose to have each logical message start a new physical message, so the NCP can send the last part of each message without waiting for an explicit request, but a remote site is not required to follow this convention.

Each logical record within a transaction begins with an "op code" byte which contains the channel identification, so its value is unique to each channel but constant within a channel. This choice provides a convenient way to verify bit synchronization at the receiver, and also allows an extension in the future to true "multileaving" (i.e., multiplexing all channels within one connection in each direction).

The only provisions for transmission error detection in the current NETRJS protocol are (1) this "op code" byte to verify bit synchronization and (2) the transaction sequence number. At the urging of Crowther, we favor putting an optional 16 bit check sum in the unused bytes of the second-level header. It is currently assumed that if an error is detected then the channel is to be aborted and the entire transmission repeated. To provide automatic retransmission we would have to put in reverse channels for ACK/NAK messages.

2. Character Sets

For an ASCII VRBT, NETRJS will map ASCII in the card reader stream into EBCDIC, and re-map the printer stream to ASCII, by the following rules:

1. One-to-one mapping between the three ASCII characters | ~ \ which are not in EBCDIC, and the three EBCDIC characters [vertical bar, not-sign and cent-sign] (respectively) which are not in ASCII.

2. The other six ASCII graphics not in EBCDIC will be translated on input to an EBCDIC question mark (?).

3. The ASCII control DC3 (the only one not in EBCDIC) will be mapped into and from the EBCDIC control TM.

4. The EBCDIC characters not in ASCII will be mapped in the printer stream into the ASCII question mark.

3. Meta-Notation

The following description of the NETRJS data transfer protocol uses a formal notation derived from that proposed in RFC #31 by Bobrow and Sutherland. (The NETRJS format is also shown diagramatically in Figure 2.)

The derived notation is both concise and easily readable, and we recommend its use for Network documentation. The notation consists of a series of productions for bit string variables whose names are capitalized. Each variable name which represents a fixed length field is followed by the length in bits (e.g., SEQNUMB(16)). Numbers enclosed in quotes are decimal, unless qualified by a leading X meaning hex. Since each hex digit is 4 bits, the length is not shown explicitly in hex numbers. For example, '1'(8) and X'FF' both represent a string of 8 one bits. The meta-syntactic operators are:

```
|       :alternative string
[ ]     :optional string
( )     :grouping
+       :catenation of bit strings
```

The numerical value of a bit string (interpreted as an integer) is symbolized by a lower case identifier preceding the string expression and separated by a colon. For example, in "i:FIELD(8)", i symbolizes the numeric value of the 8 bit string FIELD.

Finally, we use Bobrow and Sutherland's symbolism for iteration of a sub-string: (STRING-EXPRESSION = n); denotes n occurrences of STRING EXPRESSION, implicitly catenated together. Here any n >= 0 is assumed unless n is explicitly restricted.

4. Protocol Definition

```
STREAM <-- (TRANSACTION = n) + [END-OF-DATA]
```

That is, STREAM, the entire sequence of data on a particular open channel, is a sequence of n TRANSACTIONS followed by an END-OF-DATA marker (omitted if the sender aborts the channel).

```
TRANSACTION <-- THEAD(72) + (RECORD = r) + ('0'(1) = f)
```

That is, a transaction consists of a 72 bit header, r records, and f filler bits.

```
THEAD <-- X'FF' + f:FILLER(8) + SEQNUMB(16) + LENGTH(32) + X'00'
```

Transactions are to be consecutively numbered in the SEQNUMB field, starting with 0 in the first transaction after the channel is (re-) opened. The 32 bit LENGTH field gives the total length in bits of the r RECORD's which follow. For convenience, the using site may add f additional filler bits at the end of the transaction to reach a convenient word boundary on his machine; the value f is also transmitted in the FILLER field of THEAD.

```
RECORD <-- COMPRESSED | TRUNCATED
```

RJS will accept intermixed RECORD's which are COMPRESSED or TRUNCATED in an input stream. RJS will send one or the other format in the printer and punch streams to a given VRBT; the choice is determined when CCN establishes a terminal id.

```
COMPRESSED  <--   '2'(2) + DEVID(6) + (STRING = p) + '0'(8)

STRING      <--   ('6'(3) + i:DUPCOUNT(5))
                  This form represents a string of i
                  consecutive blanks

                  ('7'(3) + i:DUPCOUNT(5) + TEXTBYTE(8))
                  This form represents string of i consecutive
                  duplicated of TEXTBYTE.
```

('2'(2) + j:LENGTH(6) + (TEXTBYTE(8) = j))
This form represents a string of j characters.

The first two alternatives above in the STRING production begin with count bytes chosen to be distinguishable from the (currently defined) Telnet control characters. In a Telnet stream, the third count byte would not be needed. This is irrelevant to the current NETRJS, but it would allow the use of compression within a Telnet data stream.

TRUNCATED <-- '3'(2) + DEVID(6) + n:COUNT(8) + (TEXTBYTE(8) = n)

DEVID(6) <-- DEVNO(3) + t:DEVTYPE(3)

DEVID identifies a particular virtual device, i.e., it identifies a channel. DEVTYPE specifies the type of device, as follows:

t = 1: Output to remote operator console
2: Input from remote operator console
3: Input from card reader
4: Output to printer
5: Output to card punch
6,7: Unused

DEVNO(3) identifies the particular device of type t at this remote site; at present only DEVNO = 0 is possible.

END-OF-DATA <-- X'FE'
Signals end of job (output) or job stack (input).

APPENDIX B

Telnet for VRBT Operator Console

The remote operator console connections use the ASCII Telnet protocol as in RFC #158. Specifically:

1) The following one-to-one character mappings are used for the three EBCDIC graphics not in ASCII:

ASCII in Telnet	NETRJS
|	[vertical bar]
~	[not-sign]
\	[cent-sign]

2) Initially all Telnet control characters will be ignored. In the future we will implement the Telnet Break facility to allow a remote user to terminate extensive console output from a command.

3) An operator console input line which exceeds 133 characters (exclusive of CR LF) will be truncated by NETRJS.

4) NETRJS will accept BS to delete a character, and CAN to delete the current line. The sequence CR LF terminates each input and output line. HT will be translated to a single space in RJS. All other ASCII control characters will be ignored. NETRJS will translate the six ASCII graphics with no equivalent in EBCDIC into the character question mark ("?") on input.

APPENDIX C

Carriage Control

The carriage control characters sent in a printer channel by NETRJS conform to IBM's extended USASI code, defined by the following table:

CODE	ACTION BEFORE WRITING RECORD
blank	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12

APPENDIX D

Network/RJS Command Summary

Terminal Control and Information Command

SIGNON	First command of a session; identifies VRBT by giving its terminal id.
SIGNOFF	Last command of a session; RJS waits for any data transfer in progress to complete and then closes all connections.
STATUS	Outputs on the remote operator console a complete list, or a summary, of all jobs in the system for this VRBT, with an indication of their processing status in the Model 91.
ALERT	Outputs on the operator console the special "Alert" message, if any, from CCN computer operator. The Alert message is also automatically sent when the user does a SIGNON, or whenever the message changes.
MSG	Sends a message to CCN computer operator or to any other RJS terminal (real or virtual). A message from the computer operator or another RJS terminal will automatically appear on the remote operator console.

Job Control and Routing Commands

Under CCN's job management system, the default destination for output is the input source. Thus, a job submitted under a given VRBT will be returned to that VRBT (i.e., the same terminal id), unless the user's JCL overrides the default destination.

RJS places print and punch output described for a particular remote terminal into either an Active Queue or a Deferred Queue. When the user opens his print or punch output channel, RJS immediately starts sending job output from the Active Queue, and continues this queue is empty. Job output in the Deferred Queue, on the other hand, must be called for by job name, (via a RESET command from the remote operator) before RJS will send it. The Active/Deferred choice for output from a job is determined by the deferral status of the VRBT when the job is entered; the deferral status, which is set to the Active option when the user signs on, may be changed by the SET command.

SET — Allows the remote user to change certain properties of his VRBT for the duration of the current session;

(a) May change the default output destination to be another (real or virtual) RJS terminal or the central facility.

(b) May change the deferral status of the VRBT.

DEFER — Moves the print and punch output for a specified job or set of jobs from the Active Queue to the Deferred queue. If the job's output is in the process of being transmitted over a channel, RJS aborts the channel and saves the current output location before moving the job to the Deferred Queue. A subsequent RESET command will return it to the Active Queue with an implied Backspace (BSP).

RESET — Moves specified job(s) from Deferred to Active Queue so they may be sent to user. A specific list of job names or all jobs can be moved with one RESET command.

ROUTE — Re-routes output of specified jobs (or all jobs) waiting in the Active and Deferred Queues for this VRBT. The new destination may be any other RJS terminal or the central facility.

ABORT — Cancels a job which was successfully submitted and awaiting execution or is current executing in the Model 91. If he cancelled job was in execution, all output it produced ill be returned.

Output Stream Control Commands

BSP (BACKSPACE) — "Backspaces" output stream within current sysout data set. Actual amount backspaced depends upon sysout blocking but is typically equivalent to a page on the line printer.

CAN (CANCEL) — (a) On an output channel, CAN causes the rest of the output in the sysout data set currently being transmitted to be omitted. Alternatively, may omit the rest of the sysout data sets for the job currently being transmitted; however, the remaining system and accounting messages will be sent.

(b) On an input channel, CAN causes RJS to ignore the job currently being read. However, the channel is not aborted as a result, and RJS will continue reading in jobs on the channel.

(c) CAN can delete all sysout data sets for specified job(s) waiting in Active or Deferred Queue.

RST (RESTART) (a) Restarts a specified output stream at the beginning of the current sysout data set or, optionally, at the beginning of the job.

(b) Marks as restarted specified job(s) whose transmission was earlier interrupted by system failure or user action (e.g., DEFER command or aborting the channel). When RJS transmits these jobs again it will start at the beginning of the partially transmitted sysout data set or, optionally, at the beginning of the job. This function may be applied to jobs in either the Active or the Deferred Queue; however, if the job was in the Deferred Queue then RST also moves it to the Active Queue. If the job was never transmitted, RST has no effect other than this queue movement.

REPEAT Sends additional copies of the output of specified jobs.

EAM Echoes the card reader stream back in the printer or punch stream, or both.

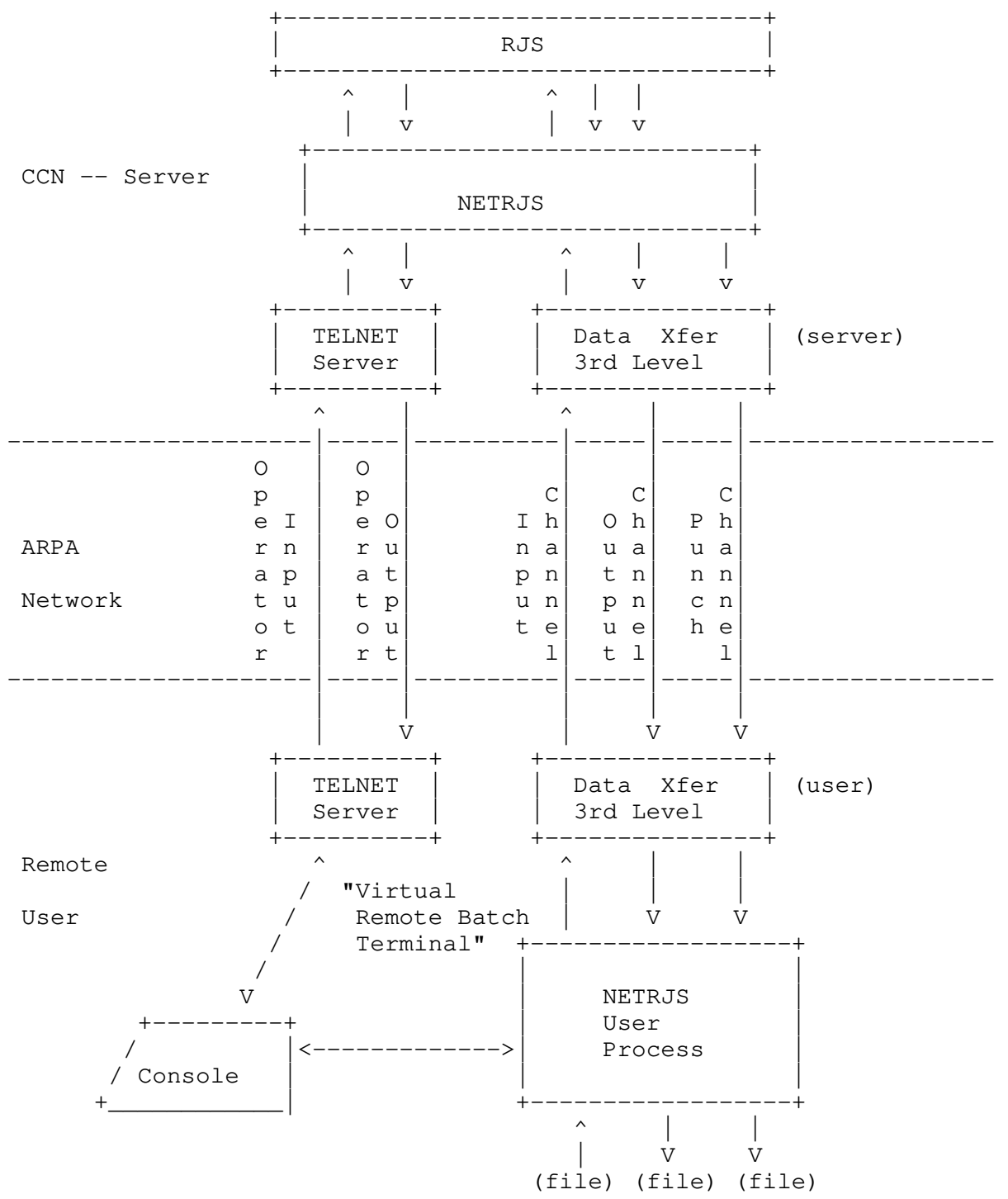

FIGURE 1. SCHEMATIC OF NETRJS OPERATION

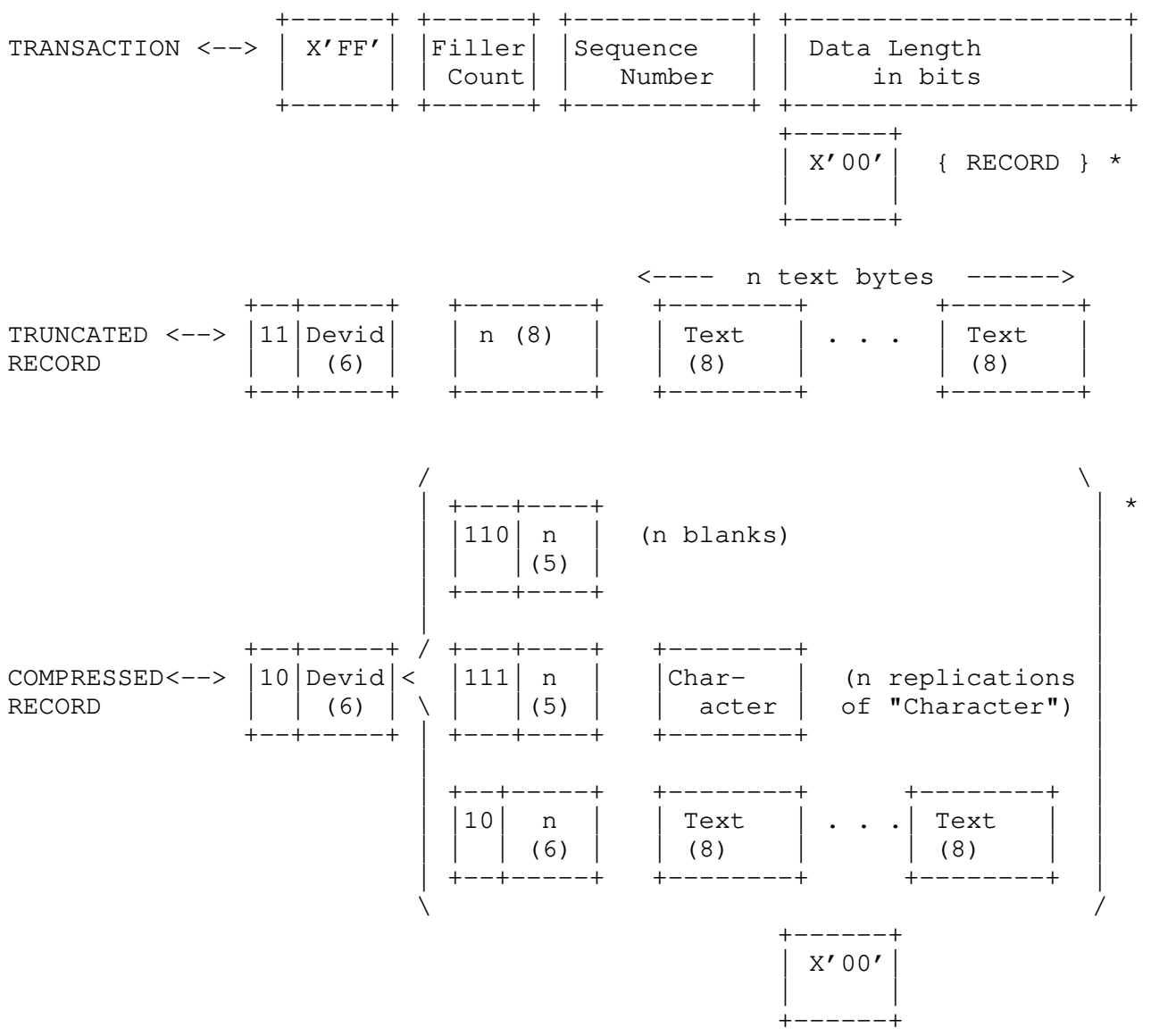

FIGURE 2. DATA TRANSFER PROTOCOL IN NETRJS

[This RFC was put into machine readable form for entry]
[into the online RFC archives by Tony Hansen 11/98]