NETRJS - A THIRD LEVEL PROTOCOL FOR REMOTE JOB ENTRY

A. Introduction

NETRJS is the name for a message protocol and set of control
conventions which will allow users at remote Hosts to access the RJS
("Remote Job Service") remote batch subsystem of CCN. RJS[1] was
written at CCN to support remote batch (car reader/line printer)
terminals over communications lines.

RJS makes a remote batch terminal's unit record devices operate as if
they were at the central site; thus, a remote user enters OS/360
jobs, complete with JCL, into the remote reader. The jobs are
spooled into the operating system and run in their turn, and the
printed and/or punched output is returned to the remote terminal from
which the jobs originated (unless the user or operator re-routes the
output). The remote terminal may also include a console typewriter
to be used by the remote operator to receive and send messages and to
exert control over his terminal [2].

When RJS is used via the ARPA Network, the "remote terminal" is
expected to be a multiprogrammed user process in a remote Host. We
will use the RJS term "remote site" for such a user process, which
presumably simulates unit record devices by file I/O. Furthermore,
several users at the same remote Host may simultaneously use NETRJS,
acting as independent "remote sites" distinguished by 8-character
names called _terminal-ids_ (because each remote site appears to RJS
as a separate physical terminal). Valid terminal-ids will be
assigned to individual users or user groups at remote Hosts who wish
to use NETRJS.

Under NETRJS, a separate ARPA network connection is opened from this
remote site to CCN for each (simulated) unit record device. Each
such connection will be called a _channel_ and be designated _input_
or _output_ with reference to CCN. We define a _standard_ remote
site in NETRJS to have the following five channels (See Figure 1):

    1._Operator Input Channel_ - Commands and messages entered by
       remote "operator" console.

    2 _Operator Output Channel_ - Message stream which would normally
       be directed to remote operator.

       3._Input Stream_ - One simulated Hollerith card reader for job
          submission.

       4._Printer Stream_ - One simulated line printer to record printed
          output (system messages and SYSOUT data sets) from jobs.

       5._Punch Stream_ - One simulated card punch, capable of recording
          arbitrary (i.e., transparent) binary text.

RJS actually will support more than one reader, printer, and punch at
each remote terminal, so the NETRJS protocol could easily be expanded
to allow multiple simultaneous I/O streams to each Network user.
However, this does not presently appear useful, as the ARPA Network
bandwidth will normally be the limitation on the transmission speed
under NETRJS.

Under NETRJS, the text of a single network message is called a
block. A block is of variable length, up to 900 bytes (except
operator input and output blocks, which may not exceed 130 bytes).
Here the term _byte_ refers to the set of 8 bits representing one
character; each byte is to be aligned on an 8-bit boundary within the
message (and block). Thus we may consider a block to be a string of
bytes. The detailed format of a block will be defined in Sections E,
F, and G, using essentially the formalism suggested by Bobrow and
Sutherland in RFC #31.

Since the central site Host (CCN) is an IBM 360, NETRJS uses the IBM
EBCDIC character code to avoid redundant code conversion at both
hosts in those cases when the remote host also uses EBCDIC
internally. However, the message formats make no assumption about
the code, and in fact, "object decks" sent to the (simulated) card
punch will normally contain arbitrary binary text.

To maximize the use of the available Network bandwidth, we strongly
recommend transmitting input blocks as large as possible; CCN will
always fully block NETRJS output. Furthermore, to avoid excessive
overhead, we urge that all NETRJS users make their marking _a
multiple of 8 bits_, so the messages received at CCN arrive on a byte
boundary.

B. Starting a Session[3]

The initial connection protocol for NETRJS is essentially that of
Crocker in RFC #66 (as restated by Harslem and Heafner in RFC #80),
with some extensions. User U at a remote Host presumably requests
his outgoing logger to make a NETRJS connection to CCN. This

logger does so by first sending an initial RFC to connect socket
(user,aen) = (U,s) to CCN socket (0,5). User 0 at CCN is the
incoming logger, and aen = 5 signifies NETRJS.

The CCN incoming logger will allocate a set of (six) consecutive aen
numbers A, A+1,......A+5, for user U, return a message containing the
socket number (U,A) as specified in RFC #66, and close the initial
connection. The remote and central sites will then open an input
channel between CCN socket (U,A) (socket f in Figure 1) and remote
socket (U, s+1). This is the remote operator input channel. The
other devices have fixed aen's at CCN assigned relative to A, in
particular:

	CCN Socket (User,aen)
Channel	
Operator Input	(U,A)
Operator Output	(U,A+1)
Card Reader (No. 1)	(U,A+2)
Printer (No. 1)	(U,A+3)
Punch (No. 1)	(U,A+5)

Once the operator input channel is open, the remote site must
transmit a valid RJS signon message [2]. This message is free-format
and consists of the command verb "SIGNON" followed by the user's
terminal-id. If RJS does not recognize the terminal-id or has no
available Line Handler for the Network, it will indicate refusal by
closing the operator input channel. Central site issues subsequent
RFC's for the other channels listed above only in response to
corresponding RFC's from the remote site

To terminate the session, the remote site may close the console input
channel (socket "a" in figure 1). Alternatively, the user can enter
a SIGNOFF command through the operator input channel; in this case,
RJS will wait until the current job output streams are complete and
then terminate the session. RJS terminates the session by closing
the console output channel (socket g). Also, if RJS should abend
then socket g will close. If either site terminates the session, all
other connections for this remote site should be closed. Note that a
user can submit a number of jobs, sign off, and later receive his
output when he signs on again.

C. Channel Control

Flow control in NETRJS is handled by the Network protocol ALL
mechanism. Before transmission of a stream of records can begin on a
particular channel, the remote site must issue an RFC and Central
must reply. This allows the central site to determine the remote

configuration dynamically. A particular card reader, printer, or
punch channel is open only while it is active, so the receiver need
not tie up buffer space needlessly. Each of these channels, when
open, assumes a buffer allocation of at least 900 bytes at the
receiver.

The operator input and output channels, on the other hand, are open
throughout the session. On these channels the receiver must provide
an allocation of at least 130 bytes.

After sending the SIGNON command over the operator input channel, the
remote site should send RFC's for all output channels which are ready
to receive data. When output is available for that site, Central
returns an RFC and begins transmission. Central closes an output
channel (socket i and j) at the end of the output for each complete
batch job.[4] The remote site must then send a new RFC and Central
must reply with an RFC to start output for another job to that
device. 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[2] on the
operator input channel.

When the remote site is ready to submit a job (or stack of
consecutive jobs), it issues an RFC for the card reader input
channel. The remote site is not required to close the channel
(socket c) after each job in a "stack" of jobs, but he must close it
following the last job in the stack to initiate its processing.

It may be necessary for the receiver site to abort a particular
channel, perhaps due to a transmission error (see Section D below on
checking) or a disk I/O error. The receiver may abort a channel
(other than console output) by closing it (sockets d, e, f, and h).
This action signals the transmitter to re-transmit the information
after the channel has been reopened (initiated by the remote site, as
always). The transmitter, on the other hand, aborts a channel by
sending a block with a particular bit combination (e = 2 in BCBYTE;
see Section E).

If either site aborts card reader (input) channel, RJS will discard
the text of the last partially-spooled job; the remote site should
re-transmit this job. Note that repeating an entire stack will enter
duplicate jobs into the system, but the second copy of a job will
"flush" due to its duplicate job name.

If a printer or punch (output) channel is aborted, Central will re-
transmit from the beginning of the current SYSOUT data set; the
effect is the same as a RESTART command.[2]

Braden, et. al.   [Page 4]

If the operator input channel is aborted, the remote site must re-
transmit the last _block_. Finally, the operator output channel has
no abort condition defined. Central will never send Channel Abort
message on this channel; if the remote site closes its socket (socket
b), Central will not re-transmit, but simply cease sending messages
until the channel is reopened. Therefore a remote site can operate
without an operator output channel; however we do not recommend this,
as the user will then miss operator advisory messages such as a
warning of an impending IPL.

D. Checking

The nature of remote job entry service is such that a low rate of
undetected errors is mandatory. The IMP's use CRC's and sequence
numbers over the communication lines, so the effective IMP-IMP error
rate should be negligible. Although there is no checking provided
for the IMP-Host interface, it seems likely that these interfaces
will either be reliable or fail catastrophically; it seems unlikely
that "drop-outs" or other random failures will occur. Therefore only
the following simple checks are provided:

1. Each block will (at least initially) contain a fixed bit check
   pattern using both on and off states of each bit path in the 16
   bit PDA interface at CCN.

   It is anticipated that even this crude check on IMP-Host
   transmission will be useful both during the initial checkout of
   hardware and software and also later if the interface becomes
   marginal. However, either site can omit the check pattern if it
   sets a bit in the Block Control Byte (BCBYTE); see Section F.

2. Each block contains a sequence number. Again this is intended for
   initial checkout and to signal catastrophic hardware or software
   problems. If the receiver detects an incorrect check pattern or
   block sequence number, he aborts the channel by closing the
   corresponding network connection; the remote site should then
   issue an RFC to re-establish the network connection. The sequence
   number of the first block after an RFC is 0. The numbers are
   never reset while the connection is open.

E. Block Format

     BLOCK <---- BLOCKHEAD + (RECORD = r) + ENDOFBLOCK

               Here r > 0
                     =
  BLOCKHEAD <-- BCBYTE + [e=0=>CHECK] + DEVBYTE

               The Blockhead field consists of a Block Control Byte,
               a 32-bit check field CHECK, and a Device Byte.

   BCBYTE <---- '1'BIT + e:ERRORCONTROL + b:BLKSEQ

               Here BLKSEQ contains a 5-bit modulo 32 block sequence
               number b. ERRORCONTROL is a 2 bit field with the
               following meanings:

               e=0 : Normal block. Contains a (presumably valid)
                       check field CHECK.

               e=1 : Block contains no check field CHECK.

               e=2 : Abort channel, initiated by transmitter.
                       Channels is not closed, transmission restarts
                       on job-related boundary.

   DEVBYTE <---- '1'BIT + n:DEVNO + t:DEVTYPE

               This byte identifies a particular remote device, i.e.,
               it identifies a stream. 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 is a 3-bit integer which identifies the
               particular device type of type t at this remote site.

   CHECK <--- '10101111'BYTE + 01010000'BYTE + '11111010'BYTE +
                                   '00000101'BYTE
   ENDOFBLOCK<----'0'BYTE

Record Format

        RECORD <------ DATA RECORD | JOBNAMERECORD

   The first record sent on a printer or punch output channel will be a
   JOBNAMERECORD, identifying the OS/360 jobname of the job which
   produced the following output.

   DATARECORD <---   '10'BIT2 + DEVCNTRL + (STRING=p) + ENDOFRECORD

   JOBNAMERECORD <-- '11000000'BYTE + '11001000'BYTE + JOBNAME +
                     ENDOFRECORD

   JOBNAME <----   (TEXTBYTE = 8)

                     This is the 8-character OS/360 jobname for the
                     following job.

   DEVCNTRL <-----   d:BIT2 + k:BIT4

                     DEVCNTRL specifies carriage control for a printer,
                     so if the device is not a printer then DEVCNTRL
                     should be '000000'. For a printer:

                     d=0   : Space k lines after printing; 0 < k < 3
                                                             =   =
                               is allowed

                     d=2   : Immediately space k lines.

                     d=1, k=1: Skip to top of new page after printing.

                     d=3, k=1: Immediately skip to top of new page.

   STRING <---   ('100' + i:DUPCOUNT)| This is a string of i
                     consecutive blanks.

                     ('101' + i:DUPCOUNT + TEXTBYTE)|

                     This is a string of i consecutive duplicates of
                     TEXTBYTE.

                     ('11' + j:LENGTH + (TEXTBYTE=j)| This is an
                     uncompressed string of j characters.

   ENDOFRECORD <---- '0'BYTE

G. Field Definitions

Name*	Meaning	Length (bits)
BIT	1-bit field	1
BIT2	2-bit field	2
BIT4	4-bit field	4
BLKSEQ	Block sequence number	5
BYTE	8-bit field aligned on 8-bit boundary	8
CHECK	Block check number	32
DEVNO	Device number of a given type	3
DEVTYPE	Device type	4
DUPCOUNT	Number of replications of duplicated character in compressed text.	5
ERRORCONTROL	Block transmission error control.	2
LENGTH	Length in bytes of the following string of text.	6
TEXTBYTE	An 8-bit byte of text	8

*Note: All non-terminal fields whose names end in
        "...BYTE" represent bytes in both length and
        alignment.

   H. NOTES AND REFERENCES

   1. Martin, V.A. and Springer, T.W., "Implementation of A Remote Job
      Service", Technical Report TR2, Campus Computing Network, UCLA,
      Los Angeles, (undated).

   2. The RJS operator commands and messages are described in detail in
      Reference 1.

   3. We use the phrase "starting a session" rather than "logging on"
      because RJS has its own log on procedure, which is, we suppose, a
      fourth-level protocol.

   4. Note that NETRJS uses closing of connections as end-of-file
      signals.

```
          REMOTE SITE                CENTRAL SITE (CCN)
     +--------------------+     +-------------------+
     |                a   |     |                   |
     | Console Input  o---------->o f               |
     |                b   |     |                   |
     | Console Output o<---------o g                |
     |                c   |     |                   |
     | Card Reader    o-----------o h               |
     |                d   |     |                   |
     | Printer        o<---------o i                |
     |                e   |     |                   |
     | Card Punch     o<---------o j                |
     |                    |     |                   |
     +--------------------+     +-------------------+
```
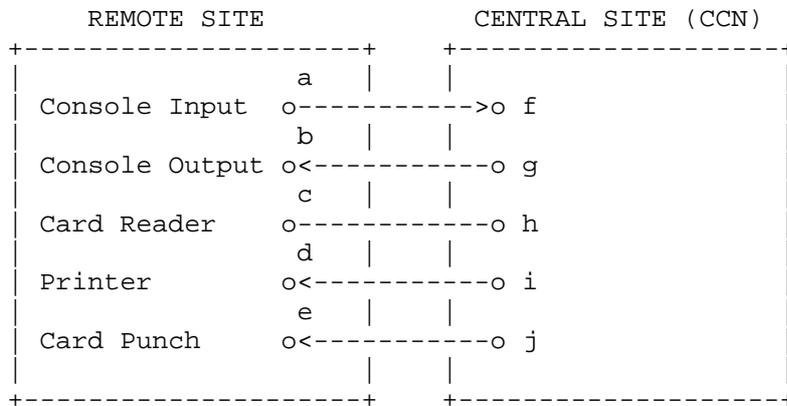
                      FIGURE 1
              ARPA Network Connections (Channels)
             For a Standard Remote Site Under NETRJS

   R.T. Braden/rb.
   S.M. Wolfe

        [This RFC was put into machine readable form for entry]
         [into the online RFC archives by Lorrie Shiota, 10/01]