Abhay Bhushan
MIT
Bob Braden
UCLA
Will Crowther
Alex McKenzie
BBN
Eric Harslem
John Heafner
Rand
John Melvin
Dick Watson
SRI
Bob Sundberg
HARVARD
Jim White
UCSB
23 June 1971

THE DATA TRANSFER PROTOCOL

I. INTRODUCTION

   A common protocol is desirable for data transfer in such diverse
   applications as remote job entry, file transfer, network mail system,
   graphics, remote program execution, and communication with block data
   terminals (such as printers, card, paper tape, and magnetic tape
   equipment, especially in context of terminal IMPs). Although it
   would be possible to include some or even all of the above
   applications in an all-inclusive file transfer protocol, a separation
   between data transfer and application functions would provide
   flexibility in implementation, and reduce complexity. Separating the
   data transfer function would also reduce proliferation of programs
   and protocols.

   We have therefore defined a low-level data transfer protocol (DTP) to
   be used for transfer of data in file transfer, remote job entry, and
   other applications protocols. This paper concerns itself solely with
   the data transfer protocol. A companion paper (RFC 172) describes
   file transfer protocol.

II. DISCUSSION

   The data transfer protocol (DTP) serves three basic functions. It
   provides for convenient separation of NCP messages into "logical"
   blocks (transactions, units, records, groups, and files), it allows
   for the separation of data and control information, and it includes
   some error control mechanisms.

Bhushan, et al.   [Page 1]

Three modes of separating messages into transactions [1] are allowed
by DTP. The first is an indefinite bit stream which terminates only
when the connection is closed (i.e., the bit stream represents a
single transaction for duration of connection). This mode would be
useful in data transfer between hosts and terminal IMPs (TIPs).

The second mode utilizes a "transparent" block convention, similar to
the ASCII DLE (Data Link Escape). In "transparent" mode,
transactions (which may be arbitrarily long) end whenever the
character sequence DLE ETX is encountered (DLE and ETX are 8-bit
character codes). To prevent the possibility of a DLE ETX sequence
occurring within data stream, any occurrence of DLE is replaced by
DLE DLE on transmission. The extra DLE is stripped on reception. A
departure from the ASCII convention is that "transparent" block does
not begin with DLE STX, but with a transaction type byte. This mode
will be useful in data transfer between terminal IMPs.

The third mode utilizes a count mechanism. Each transaction begins
with a fixed-length descriptor field containing separate binary
counts of information bits and filler bits. If a transaction has no
filler bits, its filler count is zero. This mode will be useful in
most host-to-host data transfer applications.

DTP allows for the above modes to be intermixed over the same
connection (i.e., mode is not associated with connection, but only
with transaction). The above transfer modes can represent transfer
of either data or control information. The protocol allows for
separating data or control information at a lower level, by providing
different "type" codes (see SPECIFICATIONS) for data and control
transactions. This provision may simplify some implementations.

The implementation of a workable [2] subset of the above modes is
specifically permitted by DTP. To provide compatibility between
hosts using different subsets of transfer modes, an initial
"handshake" procedure is required by DTP. The handshake involves
exchanging information on modes available for transmit and receive.
This will enable host programs to agree on transfer modes acceptable
for a connection.

The manner in which DTP is used would depend largely on the
applications protocol. It is the applications protocol which defines
the workable subset of transfer modes. For example, the file
transfer protocol will not work just with the indefinite bit stream
modes. At least, for control information one of the other two modes
is required. Again, the use of information separator and abort
functions provided in DTP (see SPECIFICATIONS) is defined by the
applications protocol. For example, in a remote job entry protocol,
aborts may be used to stop the execution of a job while they may not

cause any action in another applications protocol.

It should also be noted that DTP does not define a data transfer
service. There is no standard server socket, or initial connection
protocol defined for DTP. What DTP defines is a mechanism for data
transfer which can be used to provide services for block data
transfers, file transfers, remote job entry, network mail and
numerous other applications.

There are to be no restrictions on the manner in which DTP is
implemented at various sites. For example, DTP may be imbedded in an
applications program such as for file transfer, or it may be a
separate service program or subroutine used by several applications
programs. Another implementation may employ macros or UUO's (user
unimplemented operations on PDP-10's), to achieve the functions
specified in DTP. It is also possible that in implementation, the
separation between the DTP and applications protocols be only at a
conceptual level.

III. SPECIFICATIONS

   1. Byte Size for Network Connection

       The standard byte size for network connections using DTP is 8-
       bit. However, other byte sizes specified by higher-level
       applications protocols or applications programs are also allowed
       by DTP. For the purpose of this document bytes are assumed to be
       8-bits, unless otherwise stated.

   2. Transactions

       At DTP level, all information transmitted over connection is a
       sequence of transactions. DTP defines the rules for delimiting
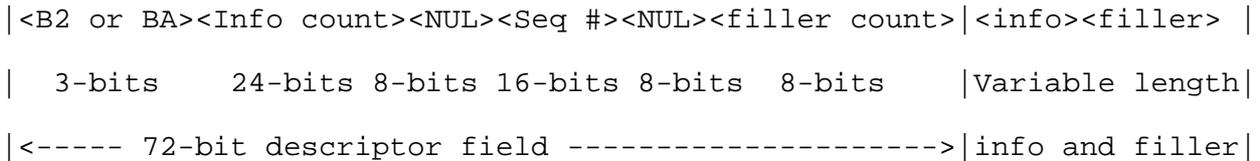   transactions. [3]

   2A. Types

       The first byte of each transaction shall define a transaction
       type, as shown below. (Note that code assignments do not
       conflict with assignments in TELNET protocol.) The transaction
       types may be referred by the hexadecimal code assigned to them.
       The transactions types are discussed in more detail in section
       2B.

```
        Code                         Transaction Type
    Hex        Octal

    B0          260            Indefinite bit stream -- data.
    B1          261            Transparent (DLE) block--data.
    B2          262            Descriptor and counts--data.
    B3          263            Modes available (handshake).
    B4          264            Information separators (endcode).
    B5          265            Error codes.
    B6          266            Abort.
    B7          267            No operation (NoOp).
    B8          270            Indefinite bit stream--control.
    B9          271            Transparent (DLE) block--control.
    BA          272            Descriptor and counts--control.
    BB          273            (unassigned but reserved for data transfer)
    BC          274                  "           "          "
    BD          275                  "           "          "
    BE          276                  "           "          "
    BF          277                  "           "          "
```

2B.   Syntax and Semantics

2B.1   Type B0 and B8 (indefinite bitstream modes) transactions
       terminate only when the NCP connection is "closed". There is
       no other escape convention defined in DTP at this level. It
       should be noted, that closing connection in bitstream mode
       represents an implicit file separator (see section 2B.5).

2B.2   Type B1 and B0 (transparent block modes) transactions terminate
       when the byte sequence DLE ETX is encountered. The sender
       shall replace any occurrence of DLE in data stream by the
       sequence DLE DLE. The receiver shall strip the extra DLE. The
       transaction is assumed to by byte-oriented. The code for DLE
       is Hex '90' or Octal '220' (this is different from the ASCII
       DLE which is Hex '10' or Octal '020). ETX is Hex '03' or Octal
       '03' (the same as ASCII ETX) [4].

2B.3   Type B2 and BA (descriptor and counts modes) transactions have
       three fields, a 9-byte (72-bits) descriptor field [5] and
       variable length (including zero) info and filler fields, as
       shown below. The total length of a transaction is
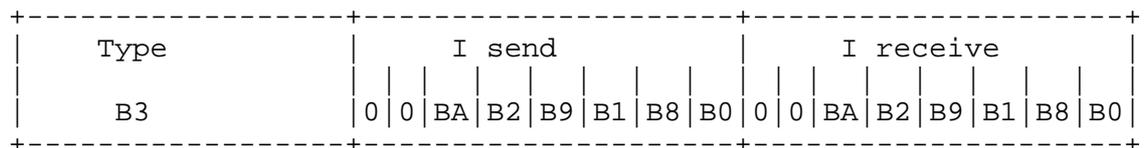   (72+info+filler) bits.

```
|<B2 or BA><Info count><NUL><Seq #><NUL><filler count>|<info><filler> |

|  3-bits    24-bits 8-bits 16-bits 8-bits  8-bits    |Variable length|

|<----- 72-bit descriptor field -------------------->|info and filler|
```

Info count is a binary count of number of bits in info field, not including descriptor or filler bits. Number of info bits is limited to $(2**24 - 1)$, as there are 24 bits in info count field.

Sequence # is a sequential count in round-robin manner of B2 and BA type transaction. The inclusion of sequence numbers would help in debugging and error control, as sequence numbers may be used to check for missing transactions, and aid in locating errors. Hosts not wishing to implement this mechanism should have all 1's in the field. The count shall start from zero and continue sequentially to all 1's, after which it is reset to all zeros. The permitted sequence numbers are one greater than the previous, and all 1's.

Filler count is a binary count of bits used as fillers (i.e., not information) after the end of meaningful data. Number of filler bits is limited to 255, as there are 8 bits in filler count field.

The NUL bytes contain all 0's.

2B.4   Type B3 (modes available) transactions have a fixed length of 3 bytes, as shown below. First byte defines transaction type as B3, second byte defines modes available for send, and third byte defines modes available for receive.

```
+-----------------+--------------------+--------------------+
|      Type       |       I send       |     I receive      |
|                 | | | |  |  |  |  | | | | |  |  |  |  | | |
|       B3        |0|0|BA|B2|B9|B1|B8|B0|0|0|BA|B2|B9|B1|B8|B0|
+-----------------+--------------------+--------------------+
```

The modes are indicated by bit-coding, as shown above. The particular bit or bits, if set to logical "1", indicate that mode to be available. The 2 most significant bits should be set to logical "0". The use of type B3 transactions is discussed in section 3B.

2B.5   Type B4 (information separator) transactions have fixed length of 2 bytes, as shown below. First byte defines transaction type as B4, and second byte defines the separator.

```
+-----------------+-----------------+
|      Type       |    End Code     |
|                 |        | |R|  |
|                 |        |G|E|  |
|       B4        |      F |R|C|U|
|                 |      I |O|O|N|
|                 |      L |U|R|I|
|                 |      E |P|D|T|
+-----------------+-----------------+
```

The following separator codes are assigned:

	Code		Meaning
Hex		Octal	
01		001	Unit separator
03		003	Record separator
07		007	Group separator
0F		017	File separator

Files, groups, records, and units may be data blocks that a
user defines to be so. The only restriction is that of the
hierarchical relationship File>Groups>Records>Units (where
'>' means 'contains'). Thus a file separator marks not only
the end of file, but also the end of group, record, and unit.
These separators may provide a convenient "logical" separation
of data at the data transfer level. Their use is governed by
the applications protocol.

2B.6   Type B5 (error codes) transactions have a fixed length of 3
       bytes, as shown below. First byte defines transaction type as
       B5, second byte indicates an error code, and third byte may
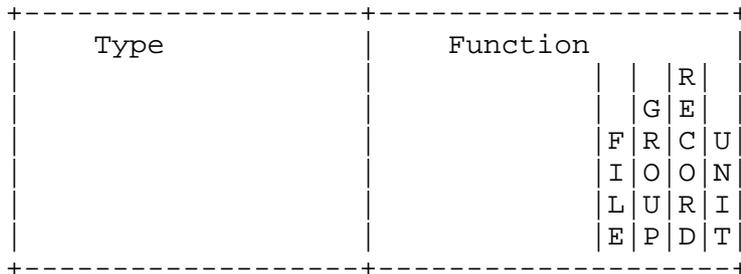   indicate the sequence number on which error occurred.

```
+-----------------+------------------+-----------------+
|      Type       |    Error Code    |    Sequence #   |
|                 |                  |                 |
|       B5        |                  |                 |
+-----------------+------------------+-----------------+
```

The following error codes are assigned:

Error Code		Meaning
Hex	Octal	
00	000	Undefined error
01	001	Out of sync. (type code other than B0 through BF).
02	002	Broken sequence (the sequence # field contains the first expected but not received sequence number).
03	003	Illegal DLE sequence (other than DLE DLE or DLE ETX).
B0 through BF	260 through 277	The transaction type (indicated by by error code) is not implemented.

The error code transaction is defined only for the purpose of
error control. DTP does not require the receiver of an error
code to take any recovery action. The receiver may discard the
error code transaction. In addition, DTP does not require that
sequence numbers be remembered or transmitted.

2B.7 Type B6 (abort) transactions have a fixed length of 2 bytes, as
      shown below. First byte defines transaction type as B6, and
      second byte defines the abort function.

```
+------------------+------------------+
|      Type        |     Function     |
|                  |          | | |R| |
|                  |          | |G|E| |
|                  |          |F|R|C|U|
|                  |          |I|O|O|N|
|                  |          |L|U|R|I|
|                  |          |E|P|D|T|
+------------------+------------------+
```

The following abort codes are assigned:

Abort Code		Meaning
Hex	Octal	
00	000	Abort preceding transaction
01	001	Abort preceding unit
02	002	Abort preceding record
07	007	Abort preceding group
0F	017	Abort preceding file

DTP does not require the receiver of an abort to take specific action, therefore sender should not necessarily make any assumptions. The manner in which abort is handled is to be specified by higher-level applications protocols.

2B.8 Type B7 (NoOp) transactions are one byte long, and indicate no operation. These may be useful as fillers when byte size used for network connections is other than 8-bits.

3. Initial Connection, Handshake and Error Recovery

   3A. DTP does not specify the mechanism used in establishing connections. It is up to the applications protocol (e.g., file transfer protocol) to choose the mechanism which suits its requirements. [6]

   3B. The first transaction after connection is made will be type B3 (modes available). In a full-duplex connection, both server and user will communicate type B3 transactions, indicating modes available for send and receive. In a simplex connection only sender will communicate a type B3 transaction. It is the sender's responsibility to choose a mode acceptable to the receiver. If an acceptable mode is not available or if mode chosen is not acceptable, the connection may be closed. [7]

   3C. No error recovery mechanisms are specified by DTP. The applications protocol may implement error recovery and further error control mechanisms.

END NOTES

[1] The term transaction is used here to mean a block of data defined by the transfer mode.

[2] What constitutes a workable subset is entirely governed by the high-level application protocol.

[3] Transactions suppress the notion of host-IMP messages, and may have
     a logical interpretation similar to that of flags (and data)
     defined by Mealy in RFC 91.

[4] This assignment is made to be consistent with the TELNET philosophy
     of maintaining the integrity of the 128 Network ASCII characters.

[5] A 72-b9t descriptor field provides a convenient separation of
     information bits, as 72 is the least common multiple of 8 and 36,
     the commonly encountered byte sizes on ARPA network host
     computers.

[6] It is, however, recommended that the standard initial connection
     protocol be adopted where feasible.

[7] It is recommended that when more than one mode is available, the
     sender should choose 'descriptor and count' mode (Type B2 or BA).
     The 'bitstream' mode (type B0 or B8) should be chosen only when
     the other two modes cannot be used.

          [This RFC was put into machine readable form for entry]
           [into the online RFC archives by Samuel Etler 08/99]