Network Working Group
Request for Comments: 310
NIC: 9261

A. Bhushan
MIT-MAC
April 3, 1972

Another Look At Data And File Transfer Protocols

Our experience with ad hoc techniques of data and file transfer over the ARPANET together with a better knowledge of terminal IMP (TIP) capabilities and Datacomputer requirements has indicated to us that the Data Transfer Protocol (DTP) (see ref 1) and the File Transfer Protocol (FTP) (see ref 2) could undergo revision. Our effort in implementing DTP and FTP has revealed areas in which the protocols could be simplified without degrading their usefulness.

This paper suggests some specific changes in DTP and FTP that should make them more useful and/or simplify implementation. The attempt here is to stimulate thinking so that we may come up with a better protocol at the forthcoming Data and File Transfer Workshop (see ref 3).

Experience to Date

A number of ad hoc techniques of transmitting data and files across the ARPANET already exist. Perhaps, the most versatile of these existing methods is the TENEX "CPYNET" system. The "CPYNET" system uses an ad hoc or interim file transfer protocol developed by Ray Tomlinson and others at BBN to transmit files among the TENEX systems on the ARPANET. [Private Communication with Bill Crowther, BBN.]

In CPYNET, the using process goes through the Initial Connection Protocol (ICP) to server socket 7, establishing a full-duplex connection with an 8-bit byte size. Control information, including user name, password, command (read, write, or append), file name, and byte size for the data connection is transmitted from the using process to the serving process. The original full-duplex connection is then closed, and a new full-duplex connection is established using the original socket numbers but with possibly a different byte size. The file is now transmitted on this newly established connection. The end-of-file is indicated by closing the connection (the mode of transfer is thus similar to DTP "indefinite bit-stream").

CPYNET has been used quite extensively for transfer of TENEX system files. Because data is not reformatted, and because the optimum connection byte size may be used for data transfer, CPYNET is quite efficient. The PDP-10 (and there are quite a lot in the ARPANET) works more efficiently with a 36 bit byte size which minimizes packing and unpacking of data, and increases effective I/O speed

(bit rate is 36 times the I/O word transfer rate instead of 8 times). The closing and reopening of connections does increase overhead but this is small in TENEX when compared with inefficiency introduced in data transfer using an inappropriate byte size.

Data and file transfer has been achieved at other sites by a simple modification of the user TELNET to enable the transfer of ASCII files as terminal I/O data streams within the constraints of the TELNET protocol. An example of this approach is the use of the "send.file" and "script" features within the MIT-DMCG user-TELNET. Send.file enables the PDP-10 (DMCG) user to transmit his local ASCII files to a receiving process such as an editor at the remote host via a TELNET connection. The program allows for a variable buffer size for transmission, and measures the transfer rate of information bits. Script enables a user to receive an ASCII file from a remote host by essentially printing it out (the terminal output stream is directed to a local file).

Our initial experience with the use of send.file program has affirmed the almost linear relationship between buffer size and transmission rate (inverse relationship to processing cost) until the limits imposed by allocates, NCP sending buffers, the IMP message size, or the receiving process speed, are reached. Our experiments have indicated that TELNET processes in which the receiving process "looks" at each character are slow and expensive. The transfer rate to most TELNET receiving processes ranges between 200 and 2,000 bits per second. The NCP-to-NCP transmission rate however is an order-of-magnitude higher (2,000 to 20,000 bits per second).

A variation of the above method which avoids the character-by-character processing of TELNET, is transmitting files via auxiliary connections (other than the TELNET connections) with or without the use of DTP. We are collecting data on response times, connect times and transfer speeds employing different transfer and buffering strategies.

TIP Capabilities and TIP Users

It appears now that TIPs will not support DTP in its present form. The more elaborate TIPs with magnetic tape units will however, support the DTP block mode (descriptor and counts) [Private Communication with Bill Crowther, BBN.] It is inconvenient, at the very least, for a TIP user to use services based on DTP (such as remote job service, file transfer, mail, and Datacomputer). The TIP philosophy is that "the computational load and storage should be in the hosts or in the terminals and not in the terminal processor." (See ref 4.) To be consistent with this philosophy the protocols should be simple and convenient to use from the user viewpoint.

Ideally, TIP users would like to connect (using the initial connection protocol) to the advertised service socket (including logger socket1) in the remote host and type their commands in a uniform, easy to use, format. Allowing the use of ASCII within DTP would facilitate this. (An alternate approach is extending TELNET to include DTP modes, particularly the indefinite bit-stream mode.) Another step would be to use printable ASCII strings instead of numeric codes for commands and arguments in user-level protocols. Use of standard file system commands (with uniform interpretation and format) will lead towards the existence of a Network Virtual File System, much in the same line as Network Virtual Terminal defined in TELNET protocol.

The transparent mode in DTP was specifically included to allow convenient use by TIPs. Since the TIPs will not support transparent mode, it makes sense to do away with it. This change would lead to a simplier DTP which allows transfer in Block mode, and the indefinite bit-stream mode. (The suggested default which would be acceptable to all including the TIPs, as it involves no overhead.). We can then make optional or do away with the now mandatory modes available handshake. The using process can indicate if it also accepts the block mode for transfer. (Either by modes available transaction, or by an argument in the command string). The server should accept input in DTP mode as well as ASCII. These fundamental changes in DTP will make communication with TIPs a lot easier.

TIP users who do not have a mediating user-FTP process and a file system in their TIP, would probably want to transfer files from input devices or to output devices such as line printer, card reader or punch, or magnetic tape. These devices "listen" on specific "ports" or sockets on a TIP. It would be desirable to modify FTP to allow sending data to a specified socket in a specified mode and type. TIP users would then find it convenient to obtain listing of their files on a high-speed line printer, input their files from a card reader, and keep back-up on cards or magnetic tapes.

Datacomputer Requirements

We have been having a continuing dialogue with CCA personnel (Dick Winter in particular), regarding CCA's plans for data and file transfer on the Datacomputer, and their specific requirements. Dick

Winter will be speaking on this subject at the Data and File Transfer Workshop. This is an attempt to summarize the main points of our discussion, and their implication for data and file transfer.

First, CCA appears quite flexible at this stage regarding the manner in which Datacomputer is to be used. Even the Datalanguage (see ref 5) is flexible and can undergo change. However, CCA would like some changes in the current file transfer protocol and its envisioned use.

Ideally, CCA would like to see a single full-duplex connection for transfer of all control information which is in Datalanguage. This information is generated by a process, which may be a user at a console, or a user program. Ability to inter-mix data and control information would be definite advantage. The Datacomputer would probably support DTP and allow use of TELNET-ASCII.

Data may alternatively be sent to or received from a separate user defined port (which may be a socket). It would be advantageous if a user could instruct the Datacomputer to transfer data to or from a file in remote system via FTP (assuming a server-FTP in remote system). CCA is currently not committed to this idea, but is considering it.

In the CCA view, the Datacomputer represents a data management facility with Datalanguage as its command language. From the viewpoint of Datacomputer as an FTP server, FTP commands be a subset of the Datalanguage. It is therefore desirable that FTP commands be printable ASCII strings instead of numeric codes.

Remote Job Service Requirements

Initially two separate protocols were proposed for Remote Job Service (RJS). One was the NETRJS protocol (see ref 6) for remote job service from large Hosts and the other was the NETRJT Protocol (see ref 7) for remote job service from TIPs (and other mini-Hosts). The current thinking however, is to move towards a single RJS with "as much overlap as possible between the methods of dealing with these two user populations." (See ref 8.) Perhaps inclusion of ASCII within DTP would make this feasible.

The existing proposals for DTP and FTP have been considered somewhat less than optimal for RJS needs. Specific drawbacks of DTP and FTP have been pointed out in the handling of data structures and data types. Most of these problems seem relatively easy to resolve. It would involve making Network ASCII the default data type (acceptable to all hosts) and providing a way in FTP for proposing and rejecting alternative data types and data structures.

Another inadequacy of FTP (which pertains to other applications as well) is in the area of error recovery. Currently there is no way to "restart" transmission if an element in the transmission path fails. One solution suggested has involved the use of sequence number (see ref 9). A number of other solutions exist to the problem. These are discussed later in the section 'FTP Reconsidered'.

DTP Reconsidered

The aspiration for DTP was that it would provide a uniform mechanism for separating information into its logical structure (records, files, and control), and rudimentary error control. The evaluation of DTP and its modes should be on the basis of speed (real-time), efficiency (processing cost), reliability (error control and recovery), and the ease of its use.

It is now clear that unless DTP was significantly revised, the TIP and other mini-Host user would find it difficult to use services based on use of DTP. Allowing the use of ASCII within DTP, and using defaults instead of the "modes available" handshake, would alleviate this problem, but compromise the DTP error control function. Whether error control belongs at the DTP level or at a higher level needs further discussion.

DTP, in its present form, does not provide sufficient error control and recovery procedures. To make DTP more useful, either it should be simplified (at least from a user viewpoint), or it should be extended to include better error control with built in error recovery, and possible handling of data types and data structures.

In the simplified version, DTP would only be a format procedure in which data could be transmitted as ASCII (no format) with escape to an 8-bit transparent (indefinite bit-stream) mode or in data blocks (descriptor and count mode). The choice of which mode to use, and all error control, error recovery, and aborts would be handled by the higher-level protocol.

The utility of the block mode in data transfer has been questioned by many who suggest that it puts a large overhead for providing the simple function of indicating end-of-file, and separating data and control information. The alternative data transfer strategy is to use separate connections for control and data information and/or close and reopen connections. This causes an overhead of a different sort, but has the advantage that the byte size for connection may be chosen to optimize data transfer.

A drawback of present DTP is that it is geared to transfer of 8-bit bytes. Perhaps a good strategy for data transfer would be to allow sending data in an agreed upon transfer mode. The transfer mode chosen should determine the byte size for connection, the data type chosen, and any data structure information. This mode may be chosen at the DTP level, or at the using protocol level.

FTP Reconsidered

The aspiration for FTP was that it would facilitate file management and file transfer in the ARPANET Virtual File System. FTP success should be evaluated by the extent of its use, convenience and efficiency in its use, and its suitability for other applications such as Datacomputer, RJS, and Mail.

Wide use of FTP would be possible if a user could use an FTP-server directly without the help of a mediating DTP/FTP-User process. This would require that commands be ASCII strings instead of numeric codes, and that ASCII characters be an acceptable input. Such a change in FTP would greatly increase its acceptance at the cost of making the server-implementation more complex. Combined implementation, however, would be simplified as the mediating FTP-user process (if used at all) would be simplified.

Efficiency of transfer is an important factor affecting the usefulness of FTP. File transfer may be very expensive (in terms of CPU time) and slow (in real-time) if an inappropriate transfer strategy is used (e.g., inappropriate byte size). Every attempt should be made to optimize transfer of data. A good strategy may be to allow transfer of files over a separate connection or close and reopen connections (using perhaps a different byte size). A problem with indicating end-of-file by closing connection is that is uncertain if the connection was closed because end-of-file was reached, or because of a failure or error condition. Perhaps "NCP interrupts" could be used in addition to a "close" to indicate definite end-of-file condition.

A drawback in the present FTP strategy is that it has no restart procedure. One proposal for restart has involved the use of the sequence numbers used in DTP block mode. Our feeling is that perhaps restart may best be accomplished by incorporating a command in FTP that allows a user to specify the place in file where to begin retransmission. A possible solution is to use the "SPF" command implemented in the UCSB Simple-Minded File System (see ref 10). Another solution may be to have optional arguments for retrieve and store commands that allow selective retrieval and replacement (specified by bits, character, words, lines, pages or segments).

Another useful addition to FTP would be a protocol procedure between user and server to agree to data type, data structure, and mode for file transfer. This would enable the user and server to reach the optimum file transfer strategy acceptable to both.

Concluding Remarks

We have discussed in this paper what we see as the major problem areas in the present DTP and FTP specifications. We hope this discussion will stimulate thinking, so that we can arrive at revised specifications for DTP and FTP that satisfy all the diverse needs in an elegant manner.

REFERENCES

1. The Data Transfer Protocol, Bhushan, et al, NWG/RFC #264, NIC #7212.

2. The File Transfer Protocol, Bhushan, et al, NWG/RFC #265, NIC #7213.

3. Data and File Transfer Workshop Announcement, A. Bhushan, NWG/RFC #309, NIC #9260.

4. The Terminal IMP for the ARPA Compuer Network, Ornstein, et al, SJCC, 1972, NIC #8218.

5. Datalanguage, Computer Operation of America, Datacomputer Project, Working Paper No.3, October 29, 1971, NIC #8208.

6. Interim NETRJS Specifications, R. T. Braden, NWG/RFC #189, NIC #7133.

7. NETRJT - - Remote Job Service Protocol for TIPs, R. T. Braden, NWG/RFC #283, NIC #8165.

8. RJS Protocol Meeting Notes, 25 February 1972, A. McKenzie (limited distribution).

9. A Suggested Addition to File Transfer Protocol, A. McKenzie, NWG/RFC #281, NIC #8163.

10. Network Specifications for UCSB's Simple-Minded Files System, James E. White, NWG/RFC #122, NIC #5834