Multiple Data Connection FTP Extensions

Mark Allman, Shawn Ostermann
{mallman,ostermann}@cs.ohiou.edu

School of Electrical Engineering and Computer Science
Ohio University

TR-19971
February 13, 1997

This work sponsored in part by NASA Lewis Research Center.

Abstract

The Transmission Control Protocol (TCP) has been shown to yield low utilization over networks with high delay*bandwidth products (DBP). We have extended FTP to assist in the study of such networks. The extensions to FTP outlined in this paper help better utilize these high DBP networks which can lead to insights into transport protocol modifications that will help all applications better utilize these channels.

1 Introduction

Experiments with standard TCP [Pos81] over networks with a high DBP have shown the protocol to be ineffective in utilizing the available bandwidth [Kru95]. TCP is evolving in an effort to better utilize such networks with mechanisms such as large windows [JBB92] and selective acknowledgments [MMFR96]. While these extensions help long-delay networks, further investigation is needed. We have modified the FTP protocol to use multiple data connections. This allows FTP to better utilize high DBP channels and can provide valuable insights into transport protocol modifications that will help all applications over long-delay channels.

Results and insights gained from using this tool are published elsewhere ([AOK95], [AKO96]). This paper is a specification of the modifications made to the FTP protocol. These extensions include three new "FTP commands" (commands exchanged between the FTP client and the FTP server on the control connection), a new user mechanism for requesting multiple data connections, and a mechanism for dividing a file across multiple data connections.

2 FTP Commands

2.1 MULT

The MULT command is a question sent from the client to determine if the server supports the use of multiple data connections. This command has no arguments. If arguments are given, the server MUST return an error code of 500 ("command not understood", as defined by [PR85]). If the server does not implement multiple connections, it will not understand the MULT command and therefore must return an error code of 500 ("command not understood") according to [PR85]. If the server can accept multiple data connections, it MUST respond with a return code of 200 ("Command OK"). The string portion of the response SHOULD contain the maximum number of connections the server supports. If used, the format of the response string is:

```
X data connections available.
```

For example:

```
16 data connections available.
```

2.2 MPRT

The MPRT command MUST be used instead of the PORT command when multiple data connections are to be employed. The syntax of the MPRT command is an extension of the EPRT command, as defined by [AO96]. The syntax of the MPRT command is:

```
MPRT <SPACE> <net-prot>|<trans-prot>|<net-addr>|<trans-addr(s)> <CRLF>
```

All fields except the last are unchanged from the definitions given in [AO96], with one exception: if the <trans-prot> field is TCP, the <trans-addr(s)> can contain any number of TCP port numbers, separated by commas (","). For example:

```
MPRT IP4|TCP|132.235.34.34|2345,6789,1212
```

If the number of ports given exceeds the number of data connections supported by the server (as returned in response the the MULT command), the extra ports MUST be ignored (that is, the server MUST NOT open a data connection on these extra ports).

2.3 MPSV

The MPSV command MUST be used instead of the PASV command when multiple data connections are to be employed. The MPSV command is issued with no arguments. The text response to the MPSV command is similar to the response to EPSV defined in [AO96]. The response MUST be:

```
(<net-prot>|<trans-prot>|<net-addr>|<trans-addr(s)>) \
    <text indicating server is entering passive mode>
```

All fields within the parenthesis are unchanged from the response to EPSV defined in in [AO96] except the <trans-addr(s)> field. If the transport protocol is TCP, the <trans-addr(s)> field can contain any number of TCP port number separated by commas (","). For example:

```
(IP4|TCP|132.235.34.34|3434,7865,9934) Entering passive mode.
```

If the number of ports listed in the MPSV response exceeds the maximum number of connections supported by the host issuing the MPSV command, the extra ports MUST be ignored. That is, no data connection should be opened on these ports.

2.4 Discussion

The MULT command is not required for correct operation of the system, but its use is RECOMMENDED before the MPRT command. In the case when the MULT command is not used and multiple data connections are not supported, an error code of 500 ("command not understood") will be returned. In this case, the client and server must revert to using a single data connection. If the MULT command is used (and both client and server make use of the MULT response string in the recommended way) the host opening the data connections passively will not waste time and resources opening data connections that will be unused.

3 Enabling Multiple Data Connections

For the user to enable the use of multiple connections, a new mechanism must be added to the client. Once the user attempts to activate multiple connections using this mechanism, the client SHOULD send the MULT FTP command to the server. The client may allow the user to specify the number of connections to be used for the transfer. If such a choice is provided, the client SHOULD place a limit on the user's choice (see section 5). Furthermore, the client SHOULD NOT allow the user's choice to be greater than the number of data connections supported by the server (obtained from the response to the MULT command).

For example, a simple ASCII interface might provide a new "user command" (issued to the client by the user) called "multiple". Without an argument, this command would trigger an attempt to activate multiple data connections using a default number of connections. With a numeric argument, the command would attempt to activate the number of data connections requested by the user (subject to the limits imposed by the client and server).

4 Dividing a File Across Multiple Connections

In order to transfer a file across multiple connections, the file MUST be broken up into 8192 byte chunks. Each chunk must be prepended with a 4 byte sequence number to form a record (as shown in figure 1). The sequence numbers begin at 0 and increase by 1 for each subsequent record. The sequence number multiplied by the chunk size (8192 bytes) forms an offset from the beginning of the file to the location of the record's data. The sequence number is necessary to reconstruct the file, as each record may be independently transmitted using any one of the established data connections.

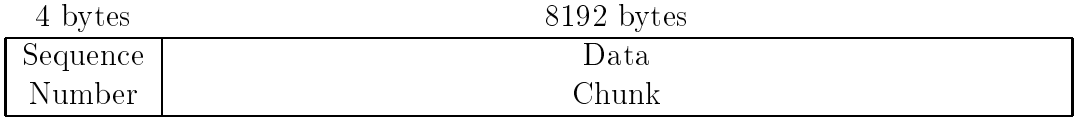

Figure 1: Record Format

5 Recommend Limits

These extensions to FTP can allow a user to inadvertently flood the network with traffic. Therefore, the client and server SHOULD set limits on the number of data connections each is willing to support. The recommended limit for both the client and the server is 8 data connections. Furthermore, it is recommended that if a client employs a default number of connections, this default be 4. These limits will minimize the user's ability to inadvertently flood the network.

6 Conclusions

These changes to the FTP protocol make it possible to better utilize networks with high DBPs. This will allow study of new transport layer mechanisms which will help better utilize these channels.

References

[AKO96] Mark Allman, Hans Kruse, and Shawn Ostermann. An Application-Level Solution to TCP's Inefficiencies. In *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*, November 1996.

[AO96] Mark Allman and Shawn Ostermann. FTP Extensions for Variable Protocol Specification, November 1996. I-D draft-allman-ftp-variable-03.txt (work in progress).

[AOK95] Mark Allman, Shawn Ostermann, and Hans Kruse. Data Transfer Efficiency Over Satellite Circuits Using a Multi-Socket Extension to the File Transfer Protocol (FTP). In *Proceedings of the ACTS Results Conference*. NASA Lewis Research Center, September 1995.

[JBB92] Van Jacobson, Robert Braden, and David Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[Kru95] Hans Kruse. Performance Of Common Data Communications Protocols Over Long Delay Links: An Experimental Examination. In *3rd International Conference on Telecommunication Systems Modeling and Design*, 1995.

[MMFR96] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. RFC 2018.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

[PR85] Jon Postel and Joyce Reynolds. File Tranfer Protocol (FTP), October 1985. RFC 959.