Teaching Tip

Tools and Techniques for Simplifying the Analysis of Captured Packet Data

Thomas P. Cavaiani
Department of Information Technology and Supply Chain Management
College of Business and Economics
Boise State University
Boise, Idaho 83725, USA
tcavaiani@boisestate.edu

ABSTRACT

Students acquire an understanding of the differences between TCP and UDP (connection-oriented vs. connection-less) data transfers as they analyze network packet data collected during one of a series of labs designed for an introductory network essentials course taught at Boise State University. The learning emphasis of the lab is not on the capture of the data, but instead on the analysis that follows. By assisting students in developing techniques to filter large batches of data using open-source tools, they gain considerable insight into the differences between aforementioned protocols.

Keywords: Effective Instruction, Data Filtering Techniques, Packet Analysis, Networking, File Transfer

1. INTRODUCTION

The introductory networking essentials course that I co-teach with one of my colleagues includes a weekly lab session. I have developed a set of thirteen labs for this course (see http://telecomm.boisestate.edu/itm305l.fall.2008/). This teaching tip describes lab 10, which provides students with experience in capturing and analyzing data transferred over a TCP/IP network.

Specifically the lab examines packets transferred from server to client during the download of a "large" file, approximately 1 megabyte in size, using FTP and TFTP. FTP transfers files in a reliable, connection-oriented fashion using TCP as its transport layer protocol, while TFTP transfers files using a connection-less approach supported by the UDP transport layer protocol.

The purpose of the lab is to allow students to collect and analyze data so that they can visualize the differences in complexity of the two transport layer protocols, TCP and UDP. TCP is more complex than UDP. TCP clients establish logical connections to TCP servers before data transmissions take place. All data segments traverse the same logical path between sender and receiver. *TCP segments* consist of a TCP header and a *payload* that contains application data. TCP uses a *sliding window protocol* to determine segment sizes. Acknowledgements are sent after segments have been received (Liebeherr and El Zarki, 2004).

UDP is a very simple protocol. UDP adds a small header to application data. The result is called a *UDP datagram*. When a UDP datagram is transmitted, it is encapsulated with an IP header and delivered using a connection-less approach, i.e., datagrams take whichever path is available from sender to receiver on the network. A separate acknowledgment is sent for each datagram that is received (Liebeherr and El Zarki, 2004).

In general, one might expect that a UDP transfer would take more time than the equivalent TCP transfer, because of the additional number of acknowledgements transmitted by UDP. On the other hand, since TCP must establish a virtual connection between sender and receiver, and supports numerous functions that UDP does not, one might expect that this additional overhead would make TCP transfers slower than UDP transfers. This lab is designed to instruct students on the use of tools and techniques that will help them determine how the features of each protocol affect its performance. Students learn how to use the Wireshark packet analyzer and the UNIX grep command to capture and analyze data to help them determine performance differences in TCP and UDP protocols. Wireshark (http://www.wireshark.org/) is a free program that can be configured to capture all network packets sent to a computer. The grep command is a filtering program that can quickly extract data subsets from large data files. Grep was originally a UNIX utility, but Windows versions are now available at http://gnuwin32.sourceforge.net/packages/grep.htm .

2. PROCEDURES

2.1 UDP File Transfers

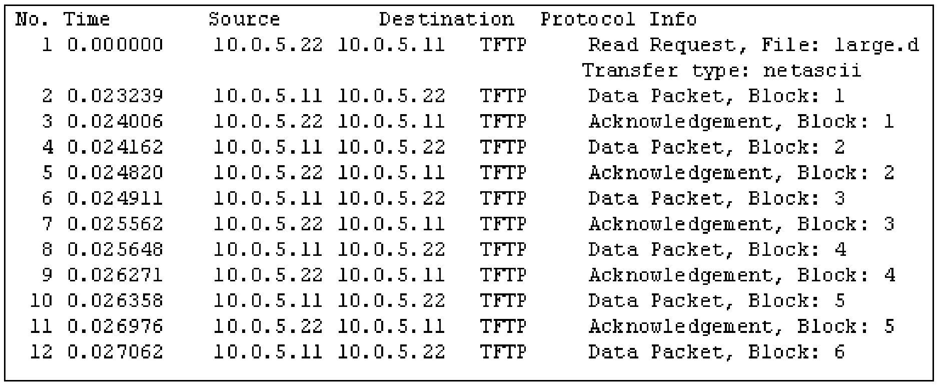
Students begin by configuring Wireshark to capture all packets sent to their client computer. While Wireshark runs in the background, students start a TFTP session and download a “large” one megabyte file stored on a TFTP server. After downloading the file, students stop the Wireshark capture and answer a set of questions pertaining to the number, size, and content of the packets captured, and calculate the actual time required to download the file (Liebeherr and El Zarki, 2004).

2.2 TCP File Transfers

The procedure required for the FTP download is similar to that used for TFTP file transfers. Students start a Wireshark capture, login to the FTP server, and download the “large” file. When the download is complete, students stop the capture and answer a set of questions similar to the TFTP question set described above. Detailed instructions for this part of the lab are available at: http://telecomm.boisestate.edu/itm305l.fall.2008/Lab10/lab_10__packet_capture_and_anal.htm .

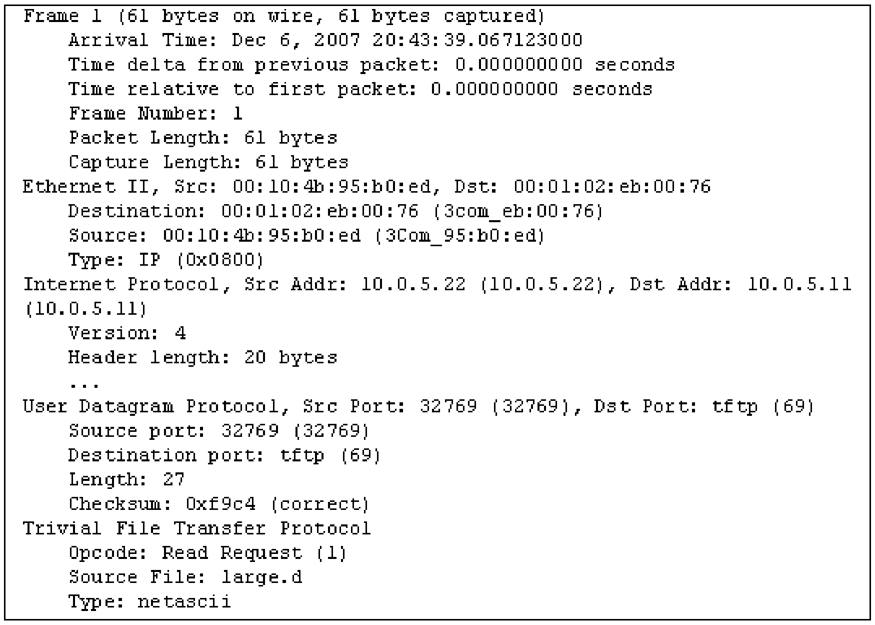
2.3 Captured File Formats

The Wireshark data captures are quite large. Over 35,000 lines are required to display all of the details of the FTP download. By default, Wireshark provides a summarized view of the packets captured. The Wireshark interface supports expansion of the default display so that packet details can be viewed. The detailed view includes all header information for all protocols involved in the file transfer, grouped by Internet Model layer. See Figures 1 and 2 for samples of these views for the TFTP download session. Since the detailed listing is extremely long, only a partial listing of the first frame captured is shown in Figure 2.



```
No. Time          Source        Destination  Protocol Info
  1 0.000000     10.0.5.22 10.0.5.11    TFTP     Read Request, File: large.d
                                                Transfer type: netascii
  2 0.023239     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 1
  3 0.024006     10.0.5.22 10.0.5.11    TFTP     Acknowledgement, Block: 1
  4 0.024162     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 2
  5 0.024820     10.0.5.22 10.0.5.11    TFTP     Acknowledgement, Block: 2
  6 0.024911     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 3
  7 0.025562     10.0.5.22 10.0.5.11    TFTP     Acknowledgement, Block: 3
  8 0.025648     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 4
  9 0.026271     10.0.5.22 10.0.5.11    TFTP     Acknowledgement, Block: 4
 10 0.026358     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 5
 11 0.026976     10.0.5.22 10.0.5.11    TFTP     Acknowledgement, Block: 5
 12 0.027062     10.0.5.11 10.0.5.22    TFTP     Data Packet, Block: 6
```

Figure 1. Partial Wireshark Summary Display



```
Frame 1 (61 bytes on wire, 61 bytes captured)
    Arrival Time: Dec 6, 2007 20:43:39.067123000
    Time delta from previous packet: 0.000000000 seconds
    Time relative to first packet: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 61 bytes
    Capture Length: 61 bytes
Ethernet II, Src: 00:10:4b:95:b0:ed, Dst: 00:01:02:eb:00:76
    Destination: 00:01:02:eb:00:76 (3com_eb:00:76)
    Source: 00:10:4b:95:b0:ed (3Com_95:b0:ed)
    Type: IP (0x0800)
Internet Protocol, Src Addr: 10.0.5.22 (10.0.5.22), Dst Addr: 10.0.5.11
(10.0.5.11)
    Version: 4
    Header length: 20 bytes
    ...
User Datagram Protocol, Src Port: 32769 (32769), Dst Port: tftp (69)
    Source port: 32769 (32769)
    Destination port: tftp (69)
    Length: 27
    Checksum: 0xf9c4 (correct)
Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: large.d
    Type: netascii
```

Figure 2. Partial Wireshark Detailed Display of Frame 1

2.4 Data Analysis
Visual analysis of packet captures is quite tedious. To simply the process of packet analysis, Wireshark provides *display filters* that allow the user to extract and display specific subsets of data captured. These tools have a rather complex syntax, which requires some practice to master. Therefore, in this lab, emphasis is placed on instruction of filtering techniques. The task of gaining insight into the characteristics and performance differences between TCP and UDP is greatly simplified when these filters are applied.

Listed below are examples of the techniques that students learn during this lab:

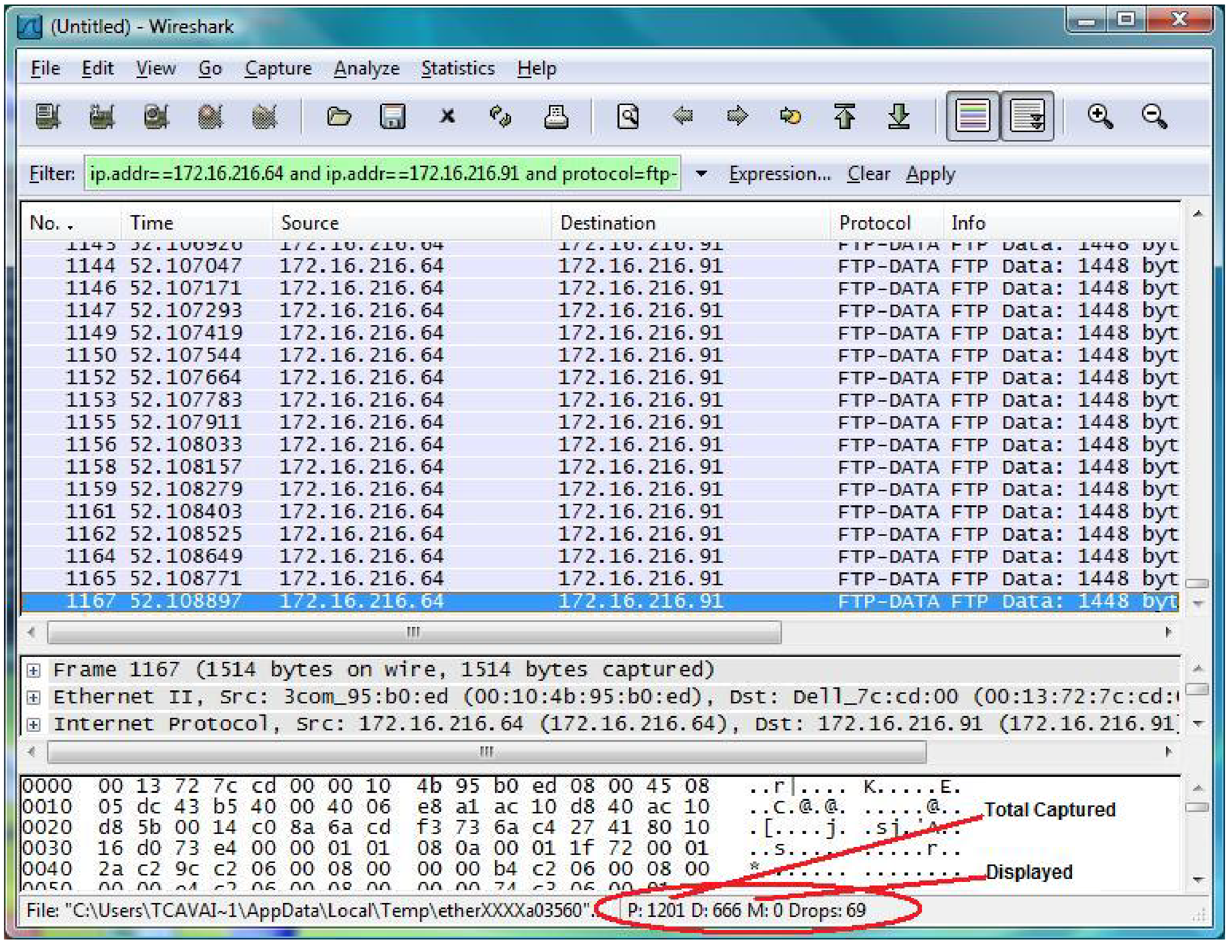
1. Apply display filters to eliminate all packets except those sent between sender and receiver. The filter *ip.addr == clientIPaddress && ip.addr == serverIPaddress*, eliminates all packets transferred to client and server by other computers.

2. Apply a filter to eliminate all protocols except the protocol of interest. For example, the filter *protocol = ftp-data* eliminates all packets that do not include FTP data.

3. Combine filters to further refine the data display – For example, combining the above two filters eliminates all packets not pertaining to the FTP download for the two computers involved. See Figure 3.

4. Check the status bar for packet counts - The Wireshark status bar displays both the total number of packets captured and the number of packets displayed as a result of filtering. See Figure 3.

5. Use the *grep* command. Wireshark capture files can be saved as text files, which can be filtered with the *grep* command. The grep command keys on substrings to extract and display different lines of a file. Grep has a very simple syntax that students learn quickly (an excellent tutorial is available at http://www.panix.com/~elflord/unix/grep.html). For example, the command *grep "ACK" ftpSummaryLarge.rtf,* displays only those lines in *ftpSummaryLarge.rtf* that contain the substring *ACK*. To illustrate the power of grep, consider the following question: *How many of the packets exchanged in the transfer carry a payload?* Issuing *grep –c "Data" tftpSummaryLarge.rtf,* instantly reduces a file of over 2800 lines into a single number, 1404, the number of packets that carry a data payload. Since we know that UDP sends one acknowledgement for each datagram received, students also immediately know the number of acknowledgementsexchanged, as well as the fact that very few of the packets transferred were control (non-data) packets.


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: ip.addr==172.16.216.64 and ip.addr==172.16.216.91 and protocol=ftp-
Expression... Clear Apply
Total Captured
Displayed

```
No.  Time      Source         Destination    Protocol Info
1143 52.106920 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1144 52.107047 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1146 52.107171 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1147 52.107293 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1149 52.107419 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1150 52.107544 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1152 52.107664 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1153 52.107783 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1155 52.107911 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1156 52.108033 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1158 52.108157 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1159 52.108279 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1161 52.108403 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1162 52.108525 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1164 52.108649 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1165 52.108771 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt
1167 52.108897 172.16.216.64  172.16.216.91  FTP-DATA FTP Data: 1448 byt

⊞ Frame 1167 (1514 bytes on wire, 1514 bytes captured)
⊞ Ethernet II, Src: 3com_95:b0:ed (00:10:4b:95:b0:ed), Dst: Dell_7c:cd:00 (00:13:72:7c:cd:
⊞ Internet Protocol, Src: 172.16.216.64 (172.16.216.64), Dst: 172.16.216.91 (172.16.216.91

0000  00 13 72 7c cd 00 00 10  4b 95 b0 ed 08 00 45 08   ..r|.... K.....E.
0010  05 dc 43 b5 40 00 40 06  e8 a1 ac 10 d8 40 ac 10   ..C.@.@. .....@..
0020  d8 5b 00 14 c0 8a 6a cd  f3 73 6a c4 27 41 80 10   .[....j. .sj.'A..
0030  16 d0 73 e4 00 00 01 01  08 0a 00 01 1f 72 00 01   ..s..... .....r..
0040  2a c2 9c c2 06 00 08 00  00 00 b4 c2 06 00 08 00   *....... ........

File: "C:\Users\TCAVAI~1\AppData\Local\Temp\etherXXXXa03560"... P: 1201 D: 666 M: 0 Drops: 69
```

Figure 3. Wireshark interface with Display Filters

Filters based upon the above suggestions permit students to answer all of the questions posed for this lab, as well as others. For a list of additional filters that might be used in this lab see http://telecomm.boisestate.edu/itm305l.fall.2008/Lab10/Lab%2010%20Answers.htm .

2.5 Evaluating Protocol Performance

This lab also helps students learn how to analyze the performance of transport layer protocols. The default Wireshark summary display for the FTP download indicates that about 25.5 seconds were required to complete the transfer. The TFTP display reveals that slightly less than 5 seconds were required to complete that download. Students are initially surprised by this result because they believe that, in general, TCP is a more efficient protocol that should take less time to transfer files than does UDP. Students quickly realize that the summary transfer time for FTP is misleading because it involves virtual circuit establishment and user authentication, both processes that the TFTP transfer did not include.

Apply filtering techniques to the captured data allows students to obtain more realistic time values, thus providing them with the data they need to quantify performance differences between the two protocols. The FTP file contents can be filtered to eliminate both authentication and control packets using a simple grep command. Sorting this listing in ascending order and then subtracting the beginning time value from the final time value provides students with a more realistic download time for the transfer of FTP data. In this case the time was about 0.8 seconds. Grep filtering of the TFTP data reveals a time of about 1.2 seconds. These more realistic values provide evidence to confirm the assumption that, at least for "large" data files, TCP transfers are faster than UDP transfers.

3. CONCLUSION

The above results, although anecdotal, appear to support the initial premise that students can gain insights into the behavior and performance differences between connection-oriented and connection-less data transfers, after they have obtained some experience with selected filtering techniques. At the very least, understanding how to use these filtering techniques greatly simplifies the amount of effort that students must expend to analyze the data obtained from packet captures of this type.

4. REFERENCES

Liebeherr, Jorg and El Zarki, Magda (2004), Mastering Networks – An Internet Lab Manual, Pearson/Addison Wesley, Boston.

AUTHOR BIOGRAPHY

Thomas P. Cavaiani is a Special Lecturer in the Department of Information Technology and Supply Chain Management at Boise State University. He received his Ph.D. in Mathematics Education from Oregon State University in 1988. He has published in the *American Technical Education Association Journal*, the *Journal of Information Systems Education*, and the *Journal of Research on Computing in Education*. He has also published two books on computer and operating system support, and has written and edited numerous computer application training manuals. His teaching interests include Java programming, operating systems, networking, and telecommunications.

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096