IOSR Journal Of Environmental Science, Toxicology And Food Technology (IOSR-JESTFT)
e-ISSN: 2319-2402,p- ISSN: 2319-2399.Volume 4, Issue 4 (May. - Jun. 2013), PP 75-80
www.Iosrjournals.Org

Analysis of Captured Data on a Typical Tcp Connection

A. S. Mindaudu[1] and I. G. Saidu[2]
[1]Sokoto Energy Research Centre, Usmanu Danfodiyo University, Sokoto.
[2] Dept. of Physics, Usmanu Danfodiyo University,Sokoto

Abstract: *Various data capture tools are in existence today for either intrusive or non-intrusive capturing of data packets. The raw packet trace obtained with such capture tools can be used to determine a lot of information concerning a connection. Usually the data capture tools give insight into the behaviour of the traffic patterns, congestion window evolution and control. Through the possible analysis of the captured data, possible areas of network improvement could be established. One of such capturing tools, Wireshark, was used to capture some packets on an established TCP connection between a client with an IP address 192.168.0.140 and a distant server having IP address 128.227.74.66. In this paper an analysis of the captured data is presented. Phenomenon such as congestion window evolution in the connectionwasobserved on the time sequence graph which also gave an indication of the lost packets in the transmission.These lost packets were identified through the repeated ACKs on the relevant frames.With the captured data, the round trip time of 177ms was calculated, and the average bottleneck bandwidth obtained was* $91.97Mbps$ *which compared favourably with the expected network bandwidth of 100 Mbps.In the captured data, a TCP-out-of-order packet was identified with a time stamp of 2.245038 indicating a lost packet.*

Keywords*: TCP,congestion window, bottleneck bandwidth, round trip time,*

I. Introduction

Wireshark, Earthreal, TCP dump etc. are some of the few examples of data capture tools from which detailed analysis of a connection can be carried out. Using Wireshark several data packets were captured on a TCP connection between a client on IP address 192.168.0.140 in a network located at the University of Plymouth in the UKand a server at the University of Florida in the USA on IP address 128.227.74.66.

TCP Connection Establishment and Closure

In order to establish the TCP connection with the server and subsequently initiate data transfer, the client sends a TCP segment referred to as the SYN segment to the server. At this point in time no application-layer data is sent along with the packet number 1, but the TCP segment sent also has a sequence number 0 which is chosen by the client. This is seen in packet number 1 in Trace 1. So, by this time the server has the SYN and the Seq. number from the client. After receiving the TCP SYN segment from the client, the server then sends a packet to the client (i.e. packet number 2 in Trace 1). This time the server sends its initial sequence number which is 0. The ACK flag is set to 1 implying that in the next packet coming from the client; it is expecting a sequence number 1. At this point also, no application-layer data is sent.

The client having received the SYN ACK sends another TCP segment to the server. This time the server increments its sequence number to 1 and also sets ACK to 1, thus signifying that the TCP connection has been established. Up to this time, however, there is still no application-layer data sent.

With these first three packets, the TCP connection between the server (IP 128.227.74.66) and the client (IP 192.168.0.140) is established. It is noted from the trace that both the client and the server set their receive windows to 65535 bytes.

The first three packets of the capture in Trace 1 describe the TCP connection set up. With the transmission of the first three packets, the client and server start exchanging data. The client first requests for data with the HTTP GET seen in packet 4; and this comes about 171 milliseconds after the first packet was sent.

Closing a TCP connection could be initiated by either the server or the client. For the captured traffic being analysed, the TCP closing session is initiated by the client in packet number 802 (see Trace 1) In this packet a FIN segment is sent by the client to the server. The ACK is set to 594424 and the sequence number is 102. After a round trip time of about 169.6milliseconds (i.e. $6.504665 - 6.335038$), the server acknowledges in packet number 803 with ACK number 103(i.e. acknowledging sequence number 102 by incrementing the ACK to 103). With this acknowledgement, the connection is closed.

Trace1 - Relevant packets for the establishment and closing of the TCP connection

No. Time Source Destination Protocol Pktlen (bytes) Info

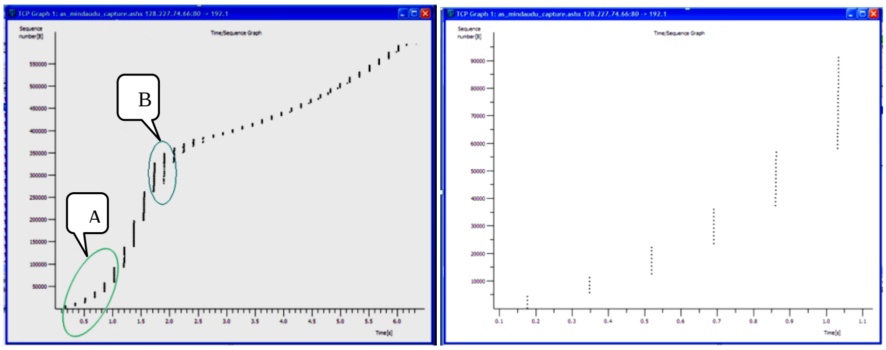
1 0.000000 192.168.0.140 128.227.74.66 TCP 62 hacl-qs> http [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2 0.170360 128.227.74.66 192.168.0.140 TCP 60 http >hacl-qs [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1380
3 0.170402 192.168.0.140 128.227.74.66 TCP 54 hacl-qs> http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
4 0.171422 192.168.0.140 128.227.74.66 HTTP 155 GET /abehnam/DSC01361.JPG HTTP/1.0
5 0.348079 128.227.74.66 192.168.0.140 TCP 1434 [TCP segment of a reassembled PDU]
. . . .
. . . .
. . . .
. . . .

799 6.333002 192.168.0.140 128.227.74.66 TCP 54 hacl-qs> http [ACK] Seq=102 Ack=592942 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
800 6.333065 128.227.74.66 192.168.0.140 HTTP 1434 [TCP Out-Of-Order] HTTP/1.1 200 OK (JPEG JFIF image)
801 6.333086 192.168.0.140 128.227.74.66 TCP 54 hacl-qs> http [ACK] Seq=102 Ack=594424 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
802 6.335038 192.168.0.140 128.227.74.66 TCP 54 hacl-qs> http [FIN, ACK] Seq=102 Ack=594424 Win=65535 [TCP CHECKSUM INCORRECT] Len=0
803 6.504665 128.227.74.66 192.168.0.140 TCP 60 http >hacl-qs [ACK] Seq=594424 Ack=103 Win=65535 Len=0

Congestion Window Evolution

The evolution of congestion windowstarts mainly with establishment of the TCP connection between the sender and receiver. With this connection set up, the TCP sends the first segment into the network and waits for an acknowledgement for it from the other end. Meanwhile, the initial congestion window is set to1 MSS (Maximum Segment Size). If the acknowledgement of the first segment is received before its set time runs out, then the congestion is increased by 1 MSS. Two segments are then sent and their acknowledgement is also awaited. Similarly if the acknowledgement for these two segments is received before their timeouts, the congestion window is increased by two MSS; one for each of the acknowledged segments, thus giving a total size of 4 MSS for the congestion window. This process continues so long as the acknowledgements for sent segments arrive before their timeout and a set congestion window threshold is not reached. So in theory, the segments in the congestion window grow from 2 to 3,5,8,12,17, etc

It is pertinent to mention here that the congestion window is not something that could be read off in the captured data. However, with the use of time-sequence graph, some deductions could be made for the congestion window. The procedure described above is the theoretical expectation which does not necessarily happen all the time in practice. For the captured traffic under analysis, the time sequence graph obtained is as shown in fig.1.



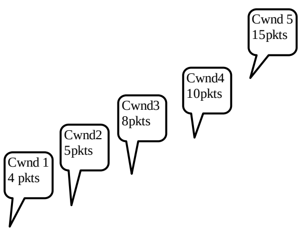


Fig. 1 - Time Sequence Graph (Stevens) for the captured data.Fig2 - Congestion window.

It is seen that the graph initially rises somewhat exponentially before it later becomes linear. This is in agreement with the description in Kurose and Ross (2001) that the congestion window grows exponentially before the threshold is reached and thereafter grows linearly. This linear increase comes about as a result of the TCP congestion avoidance phenomenon. As a result of the exponential growth of the congestion window, the receive buffer gets filled up and packets sent are lost. At this point the congestion avoidance of the TCP comes in to play and reduces the number of packets sent to linear.

For the purpose of identifying the relevant segments in the congestion window the encircled part of the graph marked A in fig.1 is enlarged as shown in fig. 2.

RTT and Bottleneck bandwidth

Round trip time (RTT) is typically the time it takes a packet to be sent from a host to another, and for a response to be received back at the originating host. Consider a simple network of two hosts A and B where a packet from host A takes t_1 seconds to reach host B and the acknowledgement from B takes t_2 seconds to reach A. The round trip time, in this case, is then t_1+t_2 seconds. Strictly speaking, the RTT is made up of various times in the packet transmission scenario such as packet propagation delay, packet queuing delay and packet processing delay (Kurose and Ross, 2001).

The RTT could be determined in one of two ways; either taking the time from either client-server-client transition or server-client-server. It is necessary, however, to ensure that the time stamps of matched SYN and SYN ACK packets are used.

For our captured traffic analysis, client-server-client is used to determine the RTT. A sent packet from the client (IP 192.168.0.140) with “Next Sequence Number” 102 (i.e. packet no.4) is matched with a received packet (no.5) from the server (IP 128.227.74.66) having “Acknowledgement Number” 102, so that the RTT is the difference between the time stamps of the two packets. From the trace shown in Appendix A,
Time stamp of packet 4, t_4 = 0.171422s and
Time stamp of packet 5, t_5 = 0.348079s

Then, $RTT = t_5 - t_4 = 0.348079 - 0.171422 \cong \mathbf{177ms}$

To determine the bottleneck bandwidth, two data back-to-back packets have to be identified and used. Loosely, the bottleneck bandwidth is the ratio of the packet length of the second packets to the difference of the time stamps of the two packets. Several packets are used in computing the bottleneck bandwidth and then averaged to obtain a more realistic value. Consequently, for the captured traffic the following back-to-back packets are chosen for the computation of the bottleneck bandwidth.

- Pair 1; Packets 5&6 with corresponding time stamps,
  t_5=0.348079s & t_6=0.348197s
- Pair 1; Packets 17&18 with corresponding time stamps,
  t_{17}=0.519554s & t_{18}=0.519688s
- Pair 1; Packets 70&71 with corresponding time stamps,
  t_{70}=1.033679s & t_{71}=1.033804s
- Pair 1; Packets 82&83 with corresponding time stamps,
  t_{82}=1.203592s & t_{83}=1.203715s

The time stamps of the identified packets are read off from the part of the trace shown in Appendix A.

Using two back-to-back packets, the bottleneck bandwidth is given by

$$BW = \frac{(Length of the second packet in bytes) X\, 8}{The difference in the time stamps of the packets} (bits/sec)$$

So, for

- Pair 1, Packets 5 & 6, $BW1 = \frac{1434 X 8}{0.348197 - 0.348079} = 97.22 Mbps$
- Pair 2, Packets 17 & 18, $BW2 = \frac{1434 X 8}{0.519688 - 0.519554} = 85.61 Mbps$

- Pair 3, Packets 70 & 71, $BW3 = \frac{1434X8}{1.033804 - 1.033679} = 91.78Mbps$
- Pair 4, Packets 82 & 83, $BW4 = \frac{1434X8}{1.203715 - 1.203592} = 93.27Mbps$

Taking the average, the bottleneck bandwidth is obtained as

$$BW = \frac{BW1 + BW2 + BW3 + BW4}{4}$$

$$BW = \frac{97.22 + 85.61 + 91.78 + 93.27}{4}$$

$$BW = \mathbf{91.97Mbps}$$

Packet Loss

Going through the captured trace, one could see that there appears to be lost packets in the connection. Indication of loss packets, in the captured traffic, is seen in the form of repetitive packets with the same acknowledgement number. An example is shown in Trace 2 where we see that frame 476 is seen out-of-order and the receiver starts sending duplicate acknowledgements as marked in red.

Trace 2 A part of the traffic captured showing packet loss.

```
No. Time     Source         Destination    Protocol  Pkt        Info
                                                     len(bytes)
476 2.245038 128.227.74.66  192.168.0.140  TCP  1434  [TCP Out-Of-Order] [TCP segment of a reassembled PDU]

477 2.245053 192.168.0.140  128.227.74.66  TCP  54    hacl-qs > http [ACK] Seq=102 Ack=351442 Win=65535 [TCP CHECKSUM
INCORRECT] Len=0

478 2.254276 128.227.74.66  192.168.0.140  TCP  1434  [TCP Retransmission] [TCP segment of a reassembled PDU]

479 2.254287 192.168.0.140  128.227.74.66  TCP  54    [TCP Dup ACK 477#1] hacl-qs > http [ACK] Seq=102 Ack=351442 Win=65535
[TCP CHECKSUM INCORRECT] Len=0

480 2.254399 128.227.74.66  192.168.0.140  TCP  1434  [TCP Retransmission] [TCP segment of a reassembled PDU]

481 2.254406 192.168.0.140  128.227.74.66  TCP  54    [TCP Dup ACK 477#2] hacl-qs > http [ACK] Seq=102 Ack=351442 Win=65535
[TCP CHECKSUM INCORRECT] Len=0

482 2.254521 128.227.74.66  192.168.0.140  TCP  1434  [TCP Retransmission] [TCP segment of a reassembled PDU]

483 2.254527 192.168.0.140  128.227.74.66  TCP  54    [TCP Dup ACK 477#3] hacl-qs > http [ACK] Seq=102 Ack=351442 Win=65535
[TCP CHECKSUM INCORRECT] Len=0

484 2.254637 128.227.74.66  192.168.0.140  TCP  1434  [TCP Retransmission] [TCP segment of a reassembled PDU]

485 2.254644 192.168.0.140  128.227.74.66  TCP  54    [TCP Dup ACK 477#4] hacl-qs > http [ACK] Seq=102 Ack=351442 Win=65535
[TCP CHECKSUM INCORRECT] Len=0

486 2.254755 128.227.74.66  192.168.0.140  TCP  1434  [TCP segment of a reassembled PDU]

487 2.254873 128.227.74.66  192.168.0.140  TCP  1434  [TCP Out-Of-Order] [TCP segment of a reassembled PDU]

488 2.254884 192.168.0.140  128.227.74.66  TCP  54    hacl-qs > http [ACK] Seq=102 Ack=352822 Win=65535 [TCP CHECKSUM
INCORRECT] Len=0

489 2.254989 128.227.74.66  192.168.0.140  TCP  1434  [TCP segment of a reassembled PDU]
```

Lost packet

Duplicate ACKs

Looking at the excerpt of the captured traffic in Trace 2, we see the behaviour of the sender which keeps re-transmitting the lost packet in response to the duplicate ACKs from the receiver. For instance, the client sends the first duplicate ACK (no. 351442) in frame 479 and the server responds in frame 480 by re-transmitting the lost segment. Again, the receiver sends another duplicate ACK in frame 481 and again the sender re-transmits the lost packet in frame 482. This continues until frame in frame 488 when the receiver acknowledges the segment in 487.

Sometime it is possible to pick out a lost packet in the time-sequence graph as can be seen in the example shown in fig. 3. This is the enlarged area of fig. 1 marked B.

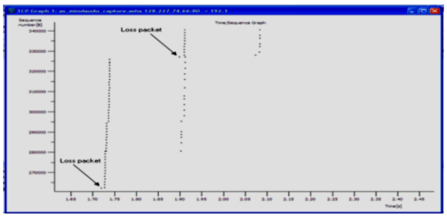


Fig. 3 **-** Example of lost packets in the capture.

II. Conclusion

The use of data capture tools has been demonstrated in this exercise using Wireshark. The packets captured were analysed successfully to establish features of the TCP connection such as congestion window evolution, packet loss and round trip bandwidth. It has been shown that for the data analysed, an average bottleneck bandwidth of 91.97Mbps was obtained, and that was reasonable for an expected network bandwidth of 100Mbps used for the capture. The congestion window phenomenon was clearly observed on the time sequence graph which also gave an indication of the lost packets in the transmission. The lost packets were also identified on the captured trace through the repeated ACKs on the relevant frames.

References

[1]. Derfler F. J. Jr., (2000) *Practical Networking* QUE A Division of Macmillan, Indiana, USA.
[2]. Hunter P. (1994) *Local Area Networks Making the Right Choices* Addison-Wiley Publishing Company, England, UK
[3]. Kurose, J. F. and Ross, K. W. (2001) *Computer Networking A Top-Down Approach Featuring the Internet*, Addison Wiley Longman, Inc., USA.
[4]. Sanders, C. (2007) *Practical Packet Analysis – Using Wireshark to solve real-world Network Problems,No Starch Press Inc., San Francisco, CA 94107.*

APPENDIX A – PACKETS USED FOR THE DETERMINATION OF ROUND TRIP TIME AND BOTTLENECK BANDWIDTH

Appendix A1 - Packets used for the determination of the round trip time (RTT)

No.	Time	Source	Destination	Protocol	Pktlen (bytes)	
1	0.000000	192.168.0.140	128.227.74.66	TCP	62	
2	0.170360	128.227.74.66	192.168.0.140	TCP	60	
3	0.170402	192.168.0.140	128.227.74.66	TCP	54	
4	**0.171422**	**192.168.0.140**	**128.227.74.66**	**HTTP**	**155**	Matched pkts.
5	**0.348079**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	
6	0.348197	128.227.74.66	192.168.0.140	TCP	1434	
7	0.348218	192.168.0.140	128.227.74.66	TCP	54	
8	0.348343	128.227.74.66	192.168.0.140	TCP	1434	

Appendix A2 - Packets used for the determination of the bottleneck bandwidth

No.	Time	Source	Destination	Protocol	Pktlen (bytes)	
1	0.000000	192.168.0.140	128.227.74.66	TCP	62	
2	0.170360	128.227.74.66	192.168.0.140	TCP	60	
3	0.170402	192.168.0.140	128.227.74.66	TCP	54	
4	0.171422	192.168.0.140	128.227.74.66	HTTP	155	
5	**0.348079**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	Pair 1
6	**0.348197**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	
7	0.348218	192.168.0.140	128.227.74.66	TCP	54	
8	0.348343	128.227.74.66	192.168.0.140	TCP	1434	
.	.	.	.	.		
.	.	.	.	.		
16	0.519450	192.168.0.140	128.227.74.66	TCP	54	
17	**0.519554**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	Pair 2
18	**0.519688**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	
19	0.519705	192.168.0.140	128.227.74.66	TCP	54	
20	0.690266	128.227.74.66	192.168.0.140	TCP	1434	
65	1.033291	128.227.74.66	192.168.0.140	TCP	1434	
66	1.033310	192.168.0.140	128.227.74.66	TCP	54	

67	1.033438	128.227.74.66	192.168.0.140	TCP	1434	
68	1.033560	128.227.74.66	192.168.0.140	TCP	1434	
69	1.033577	192.168.0.140	128.227.74.66	TCP	54	
70	**1.033679**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	Pair 3
71	**1.033804**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	
72	1.033816	192.168.0.140	128.227.74.66	TCP	54	
73	1.202847	128.227.74.66	192.168.0.140	TCP	1434	
.	.	.	.	.		
.	.	.	.	.		
79	1.203345	128.227.74.66	192.168.0.140	TCP	1434	
80	1.203469	128.227.74.66	192.168.0.140	TCP	1434	
81	1.203489	192.168.0.140	128.227.74.66	TCP	54	
82	**1.203592**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	Pair 4
83	**1.203715**	**128.227.74.66**	**192.168.0.140**	**TCP**	**1434**	
84	1.203736	192.168.0.140	128.227.74.66	TCP	54	
85	1.203831	128.227.74.66	192.168.0.140	TCP	1434	
86	1.203953	128.227.74.66	192.168.0.140	TCP	1434	
87	1.203974	192.168.0.140	128.227.74.66	TCP	54	
88	1.204069	128.227.74.66	192.168.0.140	TCP	1434	