

Traces link : <https://share.cse.iitg.ernet.in/download.php?f=rzBRnLTfaF> (2.14GB)

Application : ApexDC ++

Functionalities analysed:

1. Uploading
2. Downloading

The trace files are named as follows and would be referred to with those names in entire report:

Name	Time(approx.)	Location	My IP addrss	Action	Target IP	Hub Used
trace_1	7:00PM	CSE Lab	172.16.114.140 (LAN)	Download	10.0.1.53	adc://10.11.2.30:2780
trace_2	8:00PM	CSE Lab	172.16.114.140	Download	10.11.12.54	adc://10.11.2.30:2780
trace_3	10:00PM	Kameng hostel	192.168.1.7 (WiFi router connected to hostel LAN)	Upload	10.9.3.5	adc://10.4.10.3:2780
trace_4	10:00AM	Kameng hostel	192.168.0.112 (WiFi router connected to hostel LAN)	Download	10.9.12.29	adc://10.4.10.3:2780

QUESTION 1

The protocols used by the application while downloading and uploading a large file are as follows:

- Data Link Layer: Ethernet protocol
- Network Layer: IPv4
- Transport Layer: UDP (User Datagram Protocol), TCP (Transmission Control Protocol)
- Application Layer : DRDA, TDS, DCE/RPC, H1

Format of packets involving in upload and download:

- The DRDA and TDS packets were based on the TCP packets. Both the UDP and the TCP packets started with the same headers of the previous layers, the data link layer and the network layer.
- The **Data Link layer** that represented **Ethernet** had 14 bytes in its header. It had the destination address (6 bytes), the source address (6 bytes) and the network layer protocol (2 bytes) in that order.
- The **Network layer** that represented the **IPv4** had 20 bytes in its header. It had the protocol version number, i.e. IPv4, etc. and the header length (both were in 1 byte), the differentiated service field (1 byte), length of the packet after the Ethernet header was removed, i.e. after removing the first 14 bytes (2 bytes), the identification (2 bytes), flags and fragment offset (2 bytes), the time to live (1 byte), the transport layer protocol used (1 byte), the header checksum (2 bytes), the source address (4 bytes) and the destination address (4 bytes) in that order.
- The **Transport layer** header had different sizes for different protocols. The header for TCP had a size of 20 bytes and UDP had a size of 8 bytes.

- In **UDP**, the **total size** of the packet was typically around **160 to 350 bytes**. The transport layer header when representing UDP had **source port (2 bytes)**, the **destination port (2 bytes)**, the **length of the packet after both Ethernet and IP headers were removed (2 bytes)** and the **checksum (2 bytes)** in that order. The rest of the packet was data and this part had varying length.
- In **TCP**, but there were many other packets of **varying length** that had a TCP segment in it (Wireshark named them as “TCP segment of a reassembled PDU”). The transport layer header when representing TCP had the **source port (2 bytes)**, the **destination port (2 bytes)**, the **sequence number (4 bytes)**, the **Acknowledgement number (4 bytes)**, the **header length and the flags (2 bytes)**, the **window size value (2 bytes)**, the **checksum (2 bytes)** and the **urgent pointer (2 bytes)** in that order.

QUESTION 2

Ethernet Protocol

- The destination MAC address was **ac:2b:6e:e8:0e:db (my laptop)** and the source was **98:de:d0:de:0b:57 (WiFi router)**, for all the incoming packets for the traces that were taken in the hostel and vice versa for the outgoing packets.
- **a0:8c:fd:e4:57:ac (lab computer)** and **38:22:d6:0c:ef:99 (the first network device it was connected to)** were the MAC addresses of the destination and source for the incoming packets in the lab.
- The last part of this header had the value 0x0800.

IP Protocol

- The first byte in this header had **0x45** where the **4** indicates **version 4 of the IP protocol** and **5** indicates **the length of the header**.
- Differentiated Services Field was 0x00 for all the downloading/uploading packets.
- The length had two bytes in hex, for example 0x28 for a length of 40, and it varied vastly from different packets. Packet ID had two bytes, e.g. 0x1cca, it is different for different packets.
- The Flag byte was 0x2. The Fragment offset was 0x000.
- The time to live had different values, for example 0x40 is the default value but it can go to 0xff, it is different for different packets.
- **The protocol byte was 0x60 for TCP and 0x11 for UDP.**
- The header checksum had 2 bytes, for example 0x73d2 and it is different for different packets.

Trace	Source IP	Destination IP	Location
trace_1	0x0a000135 (10.0.1.53)	0xac10728c (172.16.114.140)	Lab
trace_2	0x0a0b0c36 (10.11.12.54)	0xac10728c (172.16.114.140)	Lab
trace_3	0x0a090305 (10.9.3.5)	0xc0a80107 (192.168.1.7)	Hostel
trace_4	0x0a090c1d (10.9.12.29)	0xc0a80070 (192.168.0.112)	Hostel

For outgoing packets.

Transmission Control Protocol

- The acknowledgement number was different for different packets.
- The Header length usually stayed the same at 0x5 (20 bytes).
- The flags that followed had 0x010 for ACK, 0x012 for SYN, ACK, and 0x18 for PSH, ACK, and 0x11 for FIN, ACK, and 0x014 for RST, ACK.
- The window size and checksum varied for different packets, 2 bytes each.
- The urgent pointer was at 0x0000.

Trace	Source Port	Destination Port	Location
trace_1	0xcaa3 (51875)	0x98cd (39117)	Lab
trace_2	0xf99e (63902)	0x98cd (39117)	Lab
trace_3	0xc7b7 (51127)	0xc44c (50252)	Hostel
trace_4	0x20e3 (8419)	0xe836 (59446)	Hostel

For outgoing packets.

User Datagram Protocol

- For incoming packets, there were different UDP packets and each packet had a different source port.
- The length and checksum was different for different packets, each had 2 bytes.

Trace	Source Port	Destination Port	Location
trace_1	0x141d (5149)	0x151d (5405)	Lab
trace_2	0x141d (5149)	0x151d (5405)	Lab
trace_3	0xc243 (49731)	0xd508 (54536)	Hostel
trace_4	0xa948 (43336)	0x1d0d (7437)	Hostel

QUESTION 3

- There were TCP packets between the hub and my computer before the download/upload started. Those packets had a “PSH, ACK” from the hub and it was responded with an “ACK”.
- This communication was done to find out the details like the key to access the actual host to download (this key is used to authenticate connections) and some other details along with it.

Source	Destination	Protocol	Length	Info
10.0.1.53	172.16.114.140	TCP	66	51875 → 39117 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM=1
172.16.114.140	10.0.1.53	TCP	66	39117 → 51875 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
10.0.1.53	172.16.114.140	TCP	60	51875 → 39117 [ACK] Seq=1 Ack=1 Win=65536 Len=0

- Before the download established, there were packets from the host computer, i.e. the computer with the file, in the UDP protocol, after some of those, a handshake was established between the host and my computer, where the three-way handshake was established, just like any TCP connection.
- The host sent a ‘SYN’ (Synchronize bit was set), to which my computer replied with a ‘SYN, ACK’ (Synchronize and Acknowledge bit were set) and received an ‘ACK’ in return. This handshake makes TCP a reliable connection.
- UDP on the other hand doesn’t have this system, it directly sends data which makes it unreliable. Hence the downloading and uploading of files were done in TCP.
- After the handshake was established, the download or upload both occurred through TCP packets but on average the size of the packets that were incoming were larger when

compared to the outgoing ones when downloading files and vice versa for uploading files, which makes perfect sense.

- In one of the traces there was a 'PSH, ACK' packet sent by the host and PSH (the push flag) indicates my computer to push the data to the socket used by the program if it wasn't done yet, saying that it has sent all the data.
- This is required to override the standard TCP delay that was created for more efficiency. There were two handshakes in one of the traces .

(1) indicating that the first one failed and packet that doesn't acknowledge within the timeout:

Source	Destination	Protocol	Length	Info
192.168.1.7	10.9.3.5	TCP	1514	[TCP Retransmission] 50584 → 35853 [ACK] Seq=371451588 Ack=3082 Win=64000 Len=1460

(2) indicating when the source gets confirmation that the packet wasn't received :

Source	Destination	Protocol	Length	Info
192.168.1.7	10.9.3.5	TCP	1514	[TCP Fast Retransmission] 50584 → 35853 [ACK] Seq=369952055 Ack=3082 Win=64000 Len=1460

(3) indicating gap seen by receiver in the packets it receives:

Source	Destination	Protocol	Length	Info
10.9.3.5	192.168.1.7	TCP	66	[TCP Dup ACK 368615#6] 35853 → 50584 [ACK] Seq=3082 Ack=371370906 Win=60672 Len=0 SLE=371372366 SRE=371414005
10.9.3.5	192.168.1.7	TCP	66	[TCP Dup ACK 368615#7] 35853 → 50584 [ACK] Seq=3082 Ack=371370906 Win=60672 Len=0 SLE=371372366 SRE=371415465

(4) indicating particular frame was received in a different order from which it was sent :

Source	Destination	Protocol	Length	Info
192.168.1.7	10.9.3.5	TCP	1514	[TCP Out-Of-Order] 50584 → 35853 [ACK] Seq=371450128 Ack=3082 Win=64000 Len=1460

QUESTION 4

DC++ uses TCP and the UDP protocols.

UDP:

- UDP does not require an initial handshake like TCP does (e.g. the 3-way handshake). It also does not retransmit any lost packets.
- This when compared to TCP has a lower reliability but is much faster, as in UDP the application only sends a packet and does not check for acknowledgements, etc.
- This is mainly advantageous for multicast applications, especially for point to multipoint communication.
- DC++ mainly uses UDP for active searches as it serves the purpose of speed and as the same file might be across multiple computers, and reliability is not the main focus but rather speed.
- This fact is clearly noticeable while searching, the results are updated randomly and the list grows with time.
- This happens as the UDP packets are usually out of order and occur from different hosts at different times.

TCP:

- All the rest of the activities like hub connection establishment, downloading and uploading files, etc. on DC++ is done using TCP.
- In these scenarios TCP is preferred over UDP as the data that is to be delivered between the hosts is important and cannot be lost.
- Hub connection, for example, requires a proper connection (in this case a handshake) and the data sent is very valuable and cannot be lost.

- Due to TCPs retransmission and acknowledgements of packets ensure that every piece of the data is received by the host (acknowledgements confirm that the receiver received the packet and if no acknowledgement is received in a certain time, the packet is retransmitted again, this is done several times until the receiver acknowledges).

QUESTION 5

B is my computer and A is the remote computer.

Name	Time(approx.)	Location	My IP address
trace_1	7:00PM	CSE Lab	172.16.114.140 (LAN)
trace_2	8:00PM	CSE Lab	172.16.114.140 (LAN)
trace_3	10:00PM	Kameng hostel	192.168.1.7 (WiFi router connected to hostel LAN)
trace_4	10:00AM	Kameng hostel	192.168.0.112 (WiFi router connected to hostel LAN)

Avg. packet size (in bytes) = (Total bytes transferred from A->B)/(No. of packets from A -> B)

Trace	Throughput (A->B) (bits/s)	Throughput (B->A) (bits/s)	Avg. Packet size (A->B) (bytes)	Avg. Packet size (B->A) (bytes)
trace_1	46M	893k	1533	54
trace_2	40M	777k	1516	54
trace_3	237k	12M	61	1382
trace_4	29M	159k	1483	54

B is my computer and A is the host that B is downloading/uploading from.

Trace	No. of UDP packets (A->B and B->A)	No. of TCP packets (A->B and B->A)	No. of packets lost
trace_1	66 (combined)	132592 (combined)	3
trace_2	61 (combined)	57896 (combined)	2
trace_3	3 (combined)	31574 (combined)	0
trace_4	164 (combined)	1674372 (combined)	1438

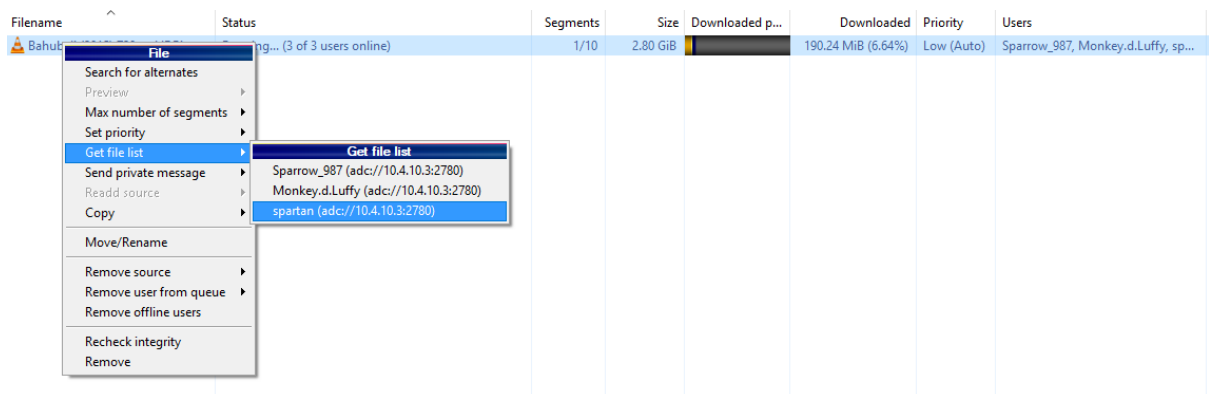
Trace	Avg. RTT (s) (A->B and B->A)
trace_1	0.00254
trace_2	0.00243
trace_3	0.00216
trace_4	0.00319

QUESTION 6

- Before every download there were short communications between the hub server and my computer.
- In trace_1, trace_2 the computer was connected to 10.11.2.30 and for trace_3, trace_4 the computer was connected to 10.4.10.3, as only one hub was active at that time.
- The actual file download or upload was done with only one host. The IP addresses of the content providers are as follows:

Trace_file	Content Provider/Taker	Type
trace_1	10.0.1.53	Download
trace_2	10.11.12.54	Download
trace_3	10.9.3.5	Upload
trace_4	10.9.12.29	Download

- In trace_3 a file was uploaded to the IP address listed from my computer.
- The speed of the file download was above 2.5 Mbps and hence dc++ did not consider downloading from other sources.
- Usually if a connection to a computer is slow, dc++ automatically shifts to other sources that are faster and if possible downloads segments from multiple sources to keep the download speed high.
- For example, in trace_4, there were multiple sources for download, but dc++ kept downloading from only one source as the speed of download was high from it.



Filename	Status	Segments	Size	Downloaded p...	Downloaded	Priority	Users
Bahut...	ng... (3 of 3 users online)	1/10	2.80 GiB	<div></div>	190.24 MiB (6.64%)	Low (Auto)	Sparrow_987, Monkey.d.Luffy, sp...

- Image showing different options DC++ could have downloaded from.