CS 349: Networks Lab Spring 2018

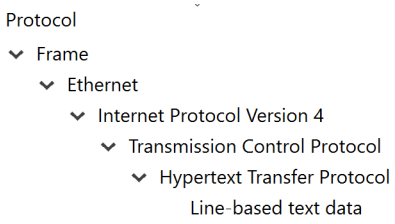
Assignment 2 — 19 Feb, 2018

Prof. T. Venkatesh *Roll: 150123051*

Link to the traces `goo.gl/Phtygb`

Q1 Protocols used by the application at different layers

I have traced car game on disney games. The protocols found in the traces which are used by the application at different layers is attached alongside. The protocol hierarchy of the protocols in application Ethernet,Internet Protocol Version 4,Transmission Control Protocol and Hyper Text Transfer Protocol.

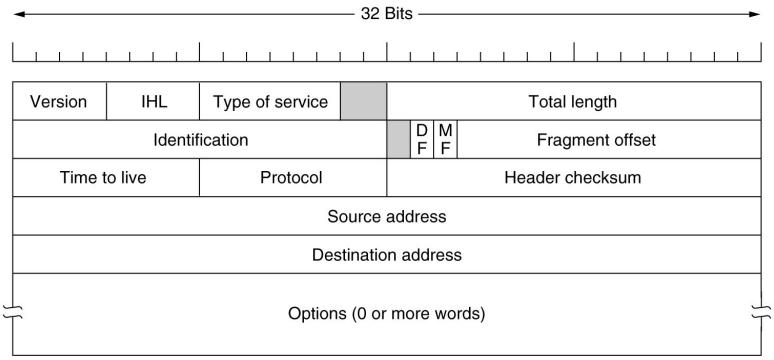


HTTP(Application Layer): The Hyper Text Transport Protocol is a text-based request-response client-server protocol. A HTTP client performs a HTTP request to a HTTP server, which in return will issue a HTTP response. The HTTP protocol header is text-based, where headers are written in text lines.

TCP(Transport Layer): TCP, Transmission Control Protocol provides reliable,ordered, and error-checked delivery of stream of octets between applications running on hosts communicating by an IP network. TCP header includes the Source and Destination ports, Sequence and Acknowledgement numbers,data offset,reserved bits,flags,window size,checksum and urgent pointer which are mandatory,an optional extension field.

Source port	Dest. port	Seq number	ACK number	Data offset	Reserved	Code/ Flags	Window size	Checksum	Urgent Pointer	Options	Padding	Data begins here
16	16	32	32	4	3	9	16	16	16	24	8	...

IPv4(Network Layer): IPv4 stands for Internet Protocol version 4, the fourth version of the Internet Protocol. IPv4 is a connectionless protocol for use on packet-switched networks. It uses 32-bit logical address. IPv4 has the following format which consists of IP version, the IHL(Internet Header Length), Total Length,TTL(Time to Live),Protocol(tells destination host, to which protocol this packet belongs to),Source and Destination Address of packet.



Ethernet(Data Link Layer): Each Ethernet frame starts with an Ethernet header containing source and destination MAC address, then comes data payload coming from upper layers. The frame ends with frame check sequence which is 32-bit Cyclic Redundancy Check(CRC) used for error detection. Ethernet packet format(with length in bytes) is given below. *Since Ethernet hardware filters the preamble and the FCS, these fields are not available to Wireshark.*

Preamble	Destination Addr.	Source Addr.	Type/Length	Data	Frame Check Sequence (FCS)
8	6	6	2	46-1500	4

Q2 Observed values for Protocols

Ethernet(Data Link Layer): From Ethernet information we can find the 48-bit(6 byte) Ethernet address/ hardware address/ MAC address of my computer which in this case is `1c:39:47:40:8b:f8`. Here is also 48-bit Ethernet address which is referred as Destination address but this is not the final destination,but the next router/switch the packet reaches. The first few bytes of this indicate the Manufacturer. This also gives the two-byte Frane type field which indicates the type of Protocol in next layer, here it is `IPv4(0x0800)`.

```
Ethernet II, Src: CompalIn_40:8b:f8 (1c:39:47:40:8b:f8), Dst: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
  Destination: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
       Address: Cisco_97:1e:ef (4c:4e:35:97:1e:ef)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: CompalIn_40:8b:f8 (1c:39:47:40:8b:f8)
       Address: CompalIn_40:8b:f8 (1c:39:47:40:8b:f8)
       .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
       .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

IPv4(Network Layer): IPv4 shows information according to format mentioned in Q1. We can see Source(`10.11.3.5`) and Destination IP address(`202.141.80.24`,because of proxy server). We can also see Version(4), Internet header length(`20 bytes`), Total Length of packet (`52`), Time to live(`128`), Transport layer protocol(`TCP`), details about Fragmentation and other similar things.

```
Internet Protocol Version 4, Src: 10.11.3.5, Dst: 202.141.80.24
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x3db8 (15800)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.11.3.5
    Destination: 202.141.80.24
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
```

TCP(Transport Layer): TCP gives details about Source and Destination port. Here it is `50030(used by firefox)` and `3128(port of our proxy server)` respectively. We can also find Relative Sequence and Acknowledgement number, Header length, Flags (Here it is `SYN` which is used to setup connection between hosts),Window size, Checksum,etc.

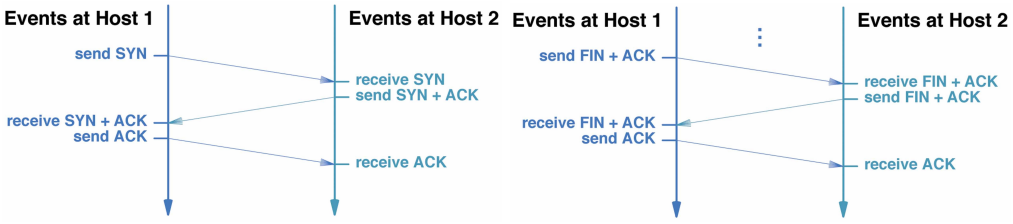
```
Transmission Control Protocol, Src Port: 50030, Dst Port: 3128, Seq: 0, Len: 0
    Source Port: 50030
    Destination Port: 3128
    [Stream index: 66]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x27dc [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
```

HTTP(Application Layer): HTTP gives host name which in this case is `'games.disney.ph'`. It also tells about Proxy authorization. HTTP/1.1 allows for client-server connections to be pipelined,whereby multiple requests can be sent without waiting for a response from the server. We can also find Accept Language(*en-US*), Encoding etc.

```
Hypertext Transfer Protocol
  > GET http://games.disney.ph/demolition-derby-cars-3 HTTP/1.1\r\n
    Host: games.disney.ph\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
  > [truncated]Cookie: SWID=4902b25c-1411-45cc-afa5-a54bc4d2d091; ctoLocalVisitor={%22localVisitorId%22:%221518949146021-3171712148778%22};
  > Proxy-Authorization: Basic cmVtaWRpLnJlZGR5OjcwODY4NjczNzk=\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
    \r\n
    [Full request URI: http://games.disney.ph/demolition-derby-cars-3]
    [HTTP request 1/10]
    [Response in frame: 645]
    [Next request in frame: 678]
```

Q3 Sequence of messages and handshaking

I have observed the following TCP handshaking protocols during the experiments.



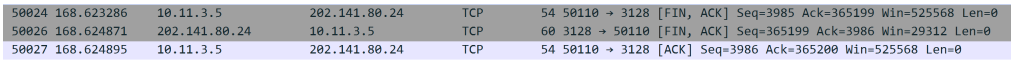
Create connection Close connection

Establishment of connection(Game start): There is a three-way TCP handshake(as mentioned above) going on when we select a game and start playing.

```
602 2.311999    10.11.3.5        202.141.80.24    TCP    66 50030 → 3128 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
603 2.312573    202.141.80.24    10.11.3.5        TCP    66 3128 → 50030 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=128
604 2.312623    10.11.3.5        202.141.80.24    TCP    54 50030 → 3128 [ACK] Seq=1 Ack=1 Win=525568 Len=0
```

(i) The client `10.11.3.5` sends `[SYN]` requesting server for synchronization.
(ii) The server `202.141.80.24`(proxy server) sends acknowledgement `[ACK]` to client's `[SYN]`. It also sends its own `[SYN]`. Thus the server sends `[SYN,ACK]` in a single segment.
(iii) The client `10.11.3.5` must acknowledge the server's(`202.141.80.24`) `[SYN]`.

Termination of connection(Game end): One application calls `close` first. This end's TCP sends a `FIN` segment, which means it is finished sending data. A three-way handshake is also used to terminate a connection. In the following,



(i) The client `10.11.3.5` terminates the connection by transmitting a segment with the `FIN` flag set.
(ii) The server `202.141.80.24`(proxy server) acknowledges this `[FIN]` and sets its own `[FIN]` flag. Thus the server sends `[FIN,ACK]` in a single segment.
(iii) The client `10.11.3.5` acknowledges the server's(`202.141.80.24`) `[FIN]` flag.

Similar thing when the other party wants to terminate the connection. So the above mentioned works in general for Host 1 and Host 2 as mentioned in the figure start of the answer.

On Pause/Restart/level change of Game: I have played many online games and have observed the following different approaches. Some applications used the create connection handshaking (`[SYN]`,`[SYN,ACK]`,`[SYN]` sequence) as if the game was started again whenever there was restart. Some games just continues sending and receiving packets and I couldn't observe any major difference.On Pause there wasn't major changes in the tcp stream. Whenever there was level change/progress to new level some games terminated the current tcp stream(using `[FIN,ACK]`, `[FIN,ACK]`,`[ACK]` sequence) and started new tcp stream using the above handshaking. `[RST]` flag was found sometimes to reset/restart the connection,it can also be for rejecting a segment.

Q4 Protocols and functioning of Application

Data Link Layer Protocol: The Data link layer protocol which in our case is *Ethernet*(which is most common in local area network) is used to send/get data in frames from client's machine(source) to server(destination). It also takes care of error detection to discard the damaged frames.

Network Layer Protocol: In our case it is *IPv4*. Internet protocol packets consisting of source and destination IP address helps in connectivity between source and destination. The IP packet structure essentially encapsulates the data which is to be delivered and routes from our local computer through many routers to find the host in a distant land.

Transport Layer Protocol: This in most of the cases will be TCP or UDP, in our case it was *TCP*. Generallay, Games may use either of them, TCP is stream based(Connection based), UDP is packet based. In many games, for example input and character postions, it really doesn't matter what happened a second ago, the game only need the most recent data. TCP was simply not designed with this in mind. In such cases we need UDP. TCP is a reliable stream delivery service which guarantees that all bytes received will be identical with bytes sent and in the correct order.

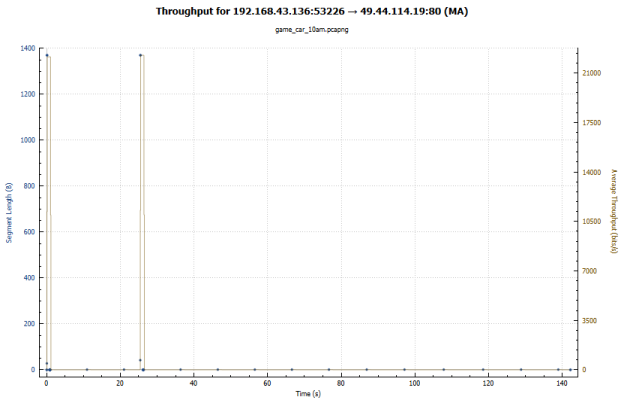
Application Layer Protocol: *Hypertext Transfer Protocol (HTTP/1.1)* is an application layer protocol used in our case. This is the protocol game uses. This is going to be different for different games. These messages are the ones we usually hear about, such as events about game getting started, or players joining and parting in case of multiplayer game, or players updating their positions or weapons or inventory. From the traces, we can find that we use `HTTP/1.1` which allows for client-server connections to be pipelined(reuses a connection multiple times to download images,GIF,media, etc. after the page has been delivered). This is used for getting resources from the game as explained below.

In Multiplayer games based on Client-Server networking architecture. Usually a server is a dedicated host that runs the game and is authoritative about world simulation, game rules, and player input processing. A client is a player's computer coonected to a game server. The client and server communicate with each other by sending small data packets at high frequency(usually 20 to 30 packets per second). A client receives the current world state from the server and generates video and audio output based on these updates. The client also samples data from input devices(keyboard, mouse, microphone, etc.) and sends these input samples back to the server for further processing. Clients only communicate with the game server and not between each other(like in peer-to-peer application). In contrast to single player game, a multiplayer game has to deal with a variety of new problems caused by packet-based communication.

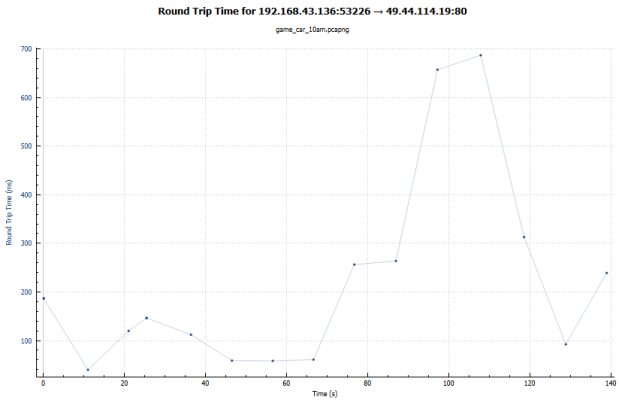
Q5 Statastics from traces

	Sample 1	Sample 2	Sample 3
Time of Day	10am	8pm	10pm
Network used	Mobile Hotspot	Hostel LAN	WIFI Router
Host A	`192.168.43.136:53226`	`10.11.3.5:50030`	`192.168.0.5:52488`
Host B	`49.44.114.19:80`	`202.141.80.24:3128`	`202.141.80.24:3128`
Throughput from A to B (bps)	`459`	`13k`	`1303`
Throughput from B to A (bps)	`4256`	`430k`	`20k`
RTT(ms)	`53.6`	`0.57`	`116.71`
Avg. packet size(bytes)	`680.327`	`1049.046`	`897.67`
No. of packets lost	`0`	`2`	`3`
No. of TCP packets	`122`	`367`	`430`
No. of UDP packets	`0`	`0`	`0`
No. of Responses/request	`1.541`	`3.5875`	`2.282`

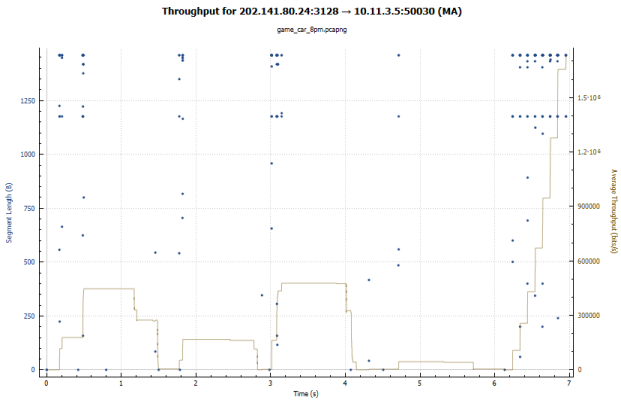
The following images can also be found along with the traces in the google drive.



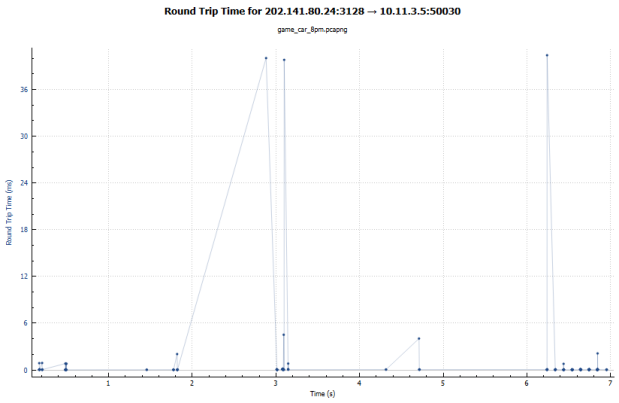
Sample 1 : Throughput



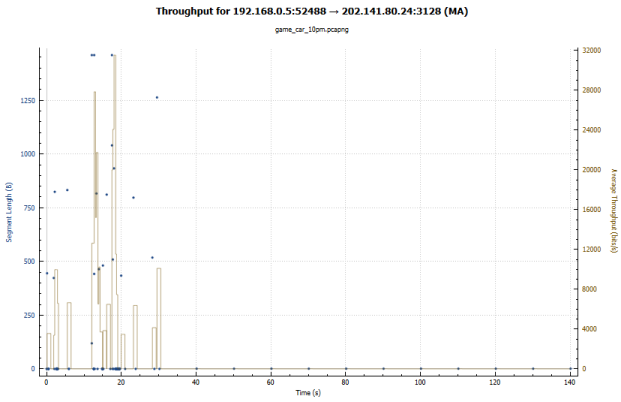
Sample 1 : RTT



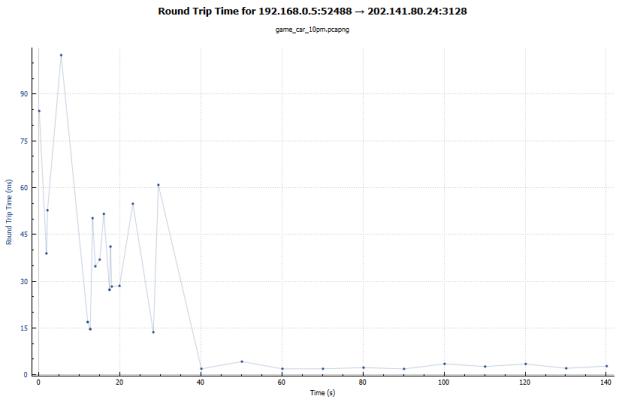
Sample 2 : Throughput



Sample 2 : RTT



Sample 3 : Throughput



Sample 3 : RTT

Q6 IP address of content providers and location

By using Institute LAN, one *cannot* find the IP of the game content providers because of *Proxy server*. Thus it is not possible to find the IP using Wireshark using Institute LAN or WIFI. So while using LAN, Destination IP was always `202.141.80.24`

Using mobile hotspot and online resources(`https://network-tools.com/`), I found the following IP addresses for `games.disney.ph` : `2.21.246.146,2.21.246.148,184.25.56.75,184.25.56.66,184.105.25.7` and `49.44.114.19`.

The IP address of the same content was observed to be different at different times.

The reason for multiple sources can be to balance the number of page requests and serve the end user faster – *load balancing*.

Round Robin DNS is a technique of load distribution, load balancing, or fault-tolerance provisioning multiple, redundant Internet Protocol service hosts. For example, a company has one domain name and three identical copies of the same web site residing on three servers with three different IP addresses.

The website might also use a Content Delivery Network(CDN, a system of distributed servers) to host static content.