CS433 Assignment 1

Shrreya Singh
19110136

Part 1:
GitHub Link: https://github.com/ShrreyaSingh/CS433_A1_RFS.git
c) Wireshark dump and analysis indicating data was correctly encrypted.

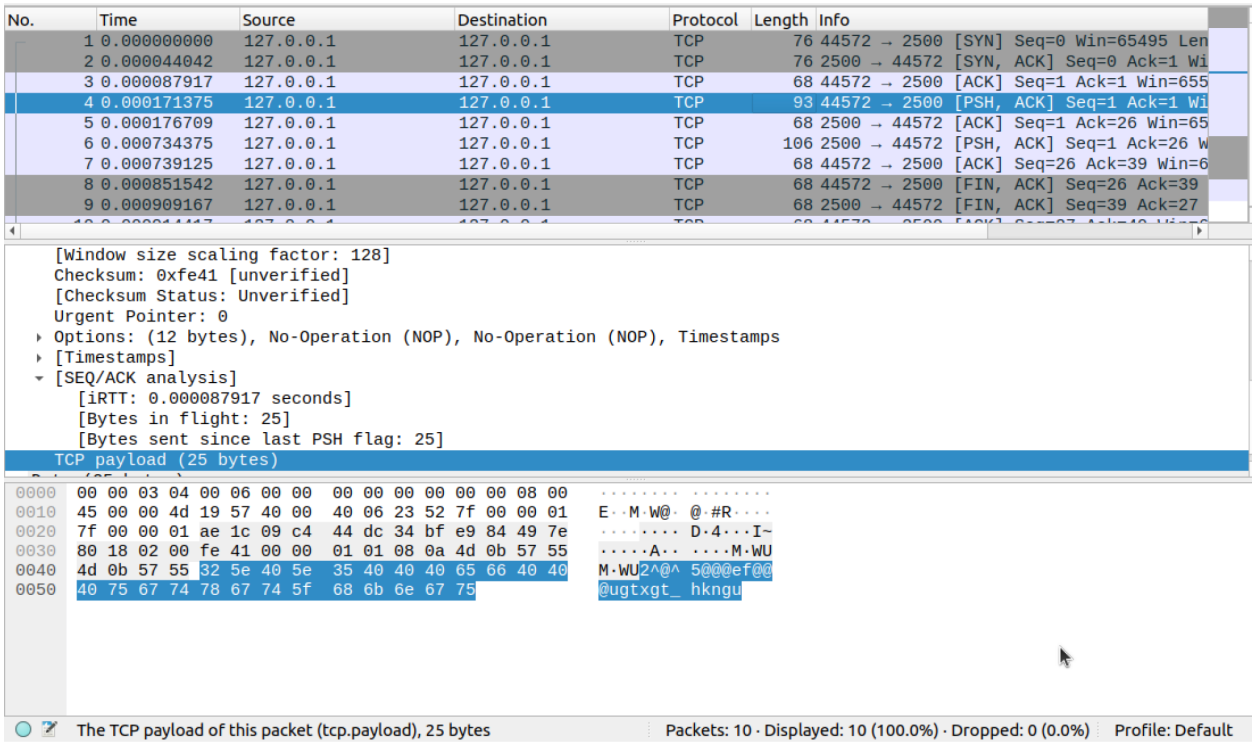
1. Checking Substitution Encryption:
   Command: cd server_files

   The actual encrypted command sent to the server and the encrypted response received are shown below (both hex and encrypted strings):

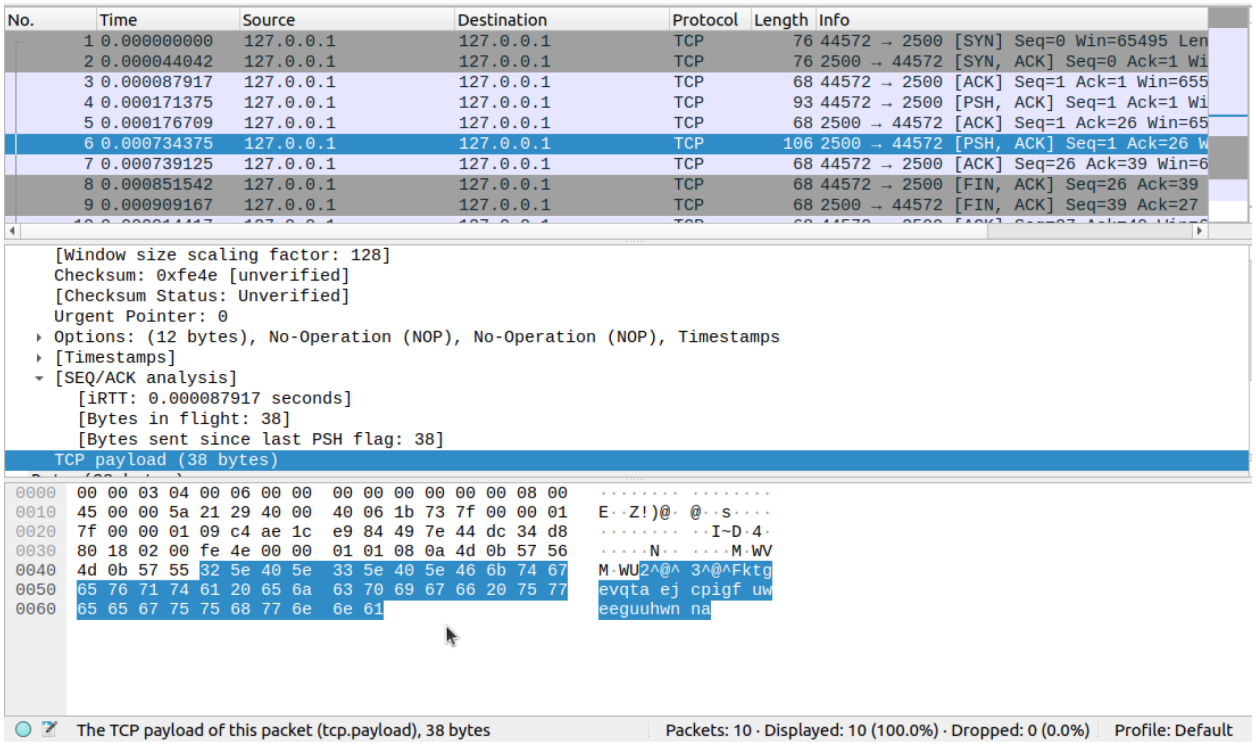
```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py

Enter any one of the below commands:
CWD
LS
CD dir_path
DWD server_file_path
UPD client_file_path

-----------------------------------------
cd server_files

Enter an integer to select the encryption type:
1 Plain
2 Substitute
3 Transpose

-------------------------------------------------
2

Setting up connection ...

Client Request Encrypted String: 2^@^5@@@ef@@@ugtxgt_hkngu

Hex string = 325e405e3540404065664040407567747867745f686b6e6775

Server Response Encrypted String: 2^@^3^@^Fktgevqta ejcpigf uweeguuhwnna

Hex string = 325e405e335e405e466b7467657671746120656a6370696766207577656567757568776e6e61

Connection closed :)
-------------------------------
Received: 'Directory changed successfully'
-------------------------------
STATUS: OK
```

   These results can be verified from the wireshark dump (Q1_A1_cd.pcapng). The snapshots of the request and response payloads (from WireShark dump) match the encrypted string and its hex code.

Client Request Payload:



Server Response Payload:



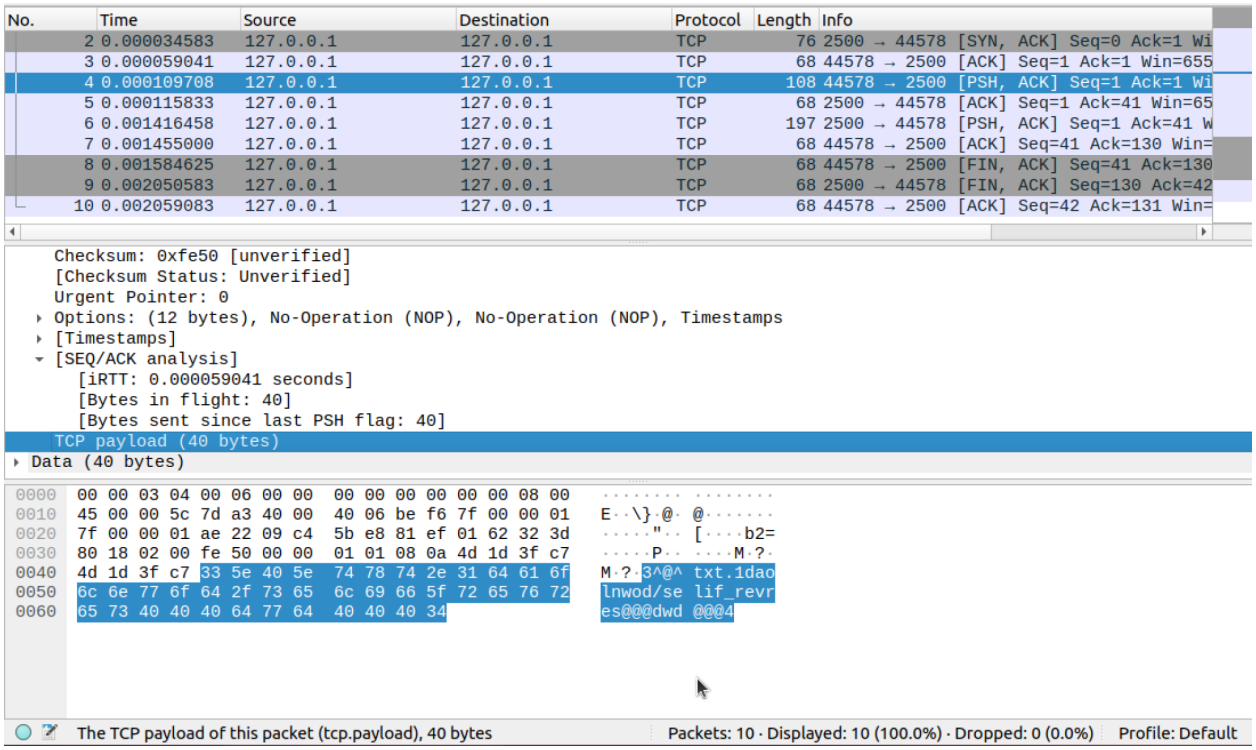
2. Checking Transpose Encryption:
   Command: dwd server_files/download1.txt

   The actual encrypted command sent to the server and the encrypted response received are shown below (both hex and encrypted strings):

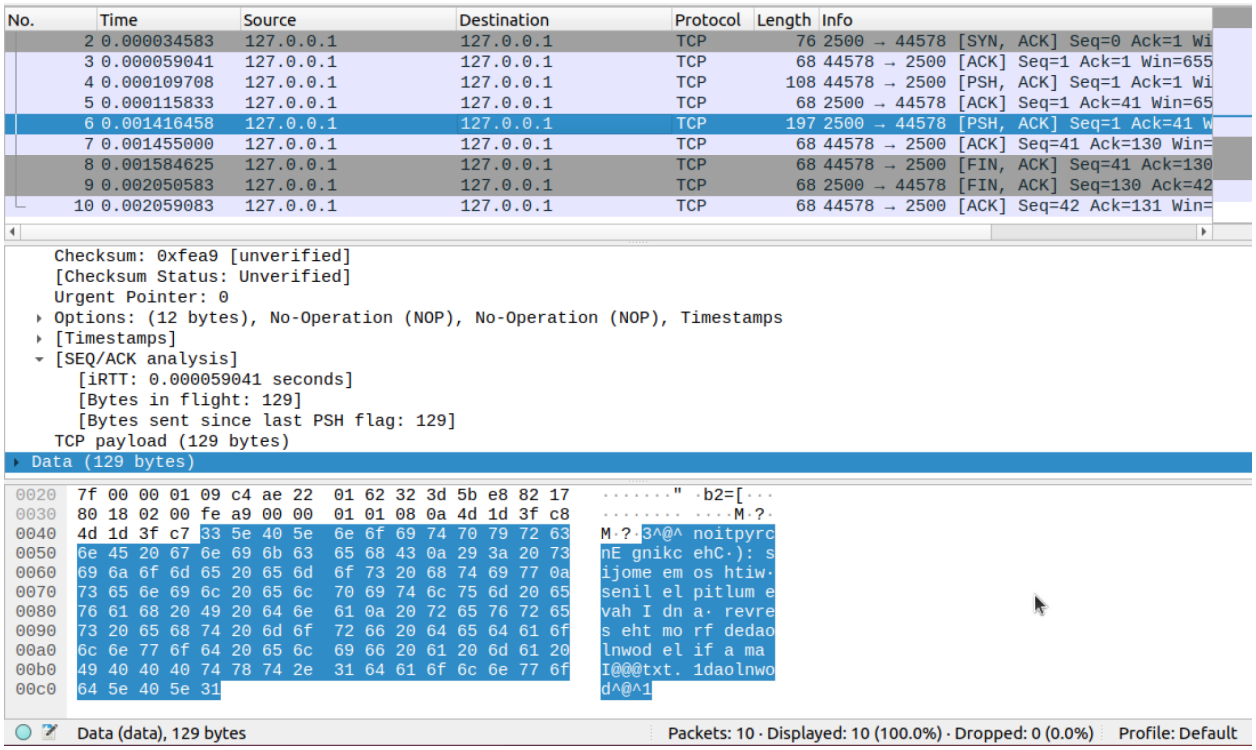
```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py

Enter any one of the below commands:
CWD
LS
CD dir_path
DWD server_file_path
UPD client_file_path

-----------------------------------------
dwd server_files/download1.txt

Enter an integer to select the encryption type:
1 Plain
2 Substitute
3 Transpose

-----------------------------------------------------
3

Setting up connection ...

Client Request Encrypted String: 3^@^txt.1daolnwod/selif_revres@@@dwd@@@4

Hex string = 335e405e7478742e3164616f6c6e776f642f73656c69665f726576726573404064776440404034

Server Response Encrypted String: 3^@^noitpyrcnE gnikcehC
): sijome emos htiw
senil elpitlum evah I dna
 revres eht morf dedaolnwod elif a ma I@@txt.1daolnwod^@^1

Hex string = 335e405e6e6f6974707972636e4520676e696b636568430a293a2073696a6f6d6520656d6f7320687469770a73656e696c20656c7069746c756d20657661682049206 46e610a207265767265732065687420 6d6f726620646564
616f6c6e776f6420656c6966206120 6d6120494040407478742e3164616f6c6e776f645e405e31

Connection closed :)
---------------------------------
Received: 'I am a file downloaded from the server \nand I have multiple lines\nwith some emojis :)\nChecking Encryption'
---------------------------------
STATUS: OK
```

   These results can be verified from the wireshark dump (Q1_A1_dwd.pcapng). The snapshots of the request and response payloads (from WireShark dump) match the encrypted string and its hex code.

Client Request Payload:



Server Response Payload:



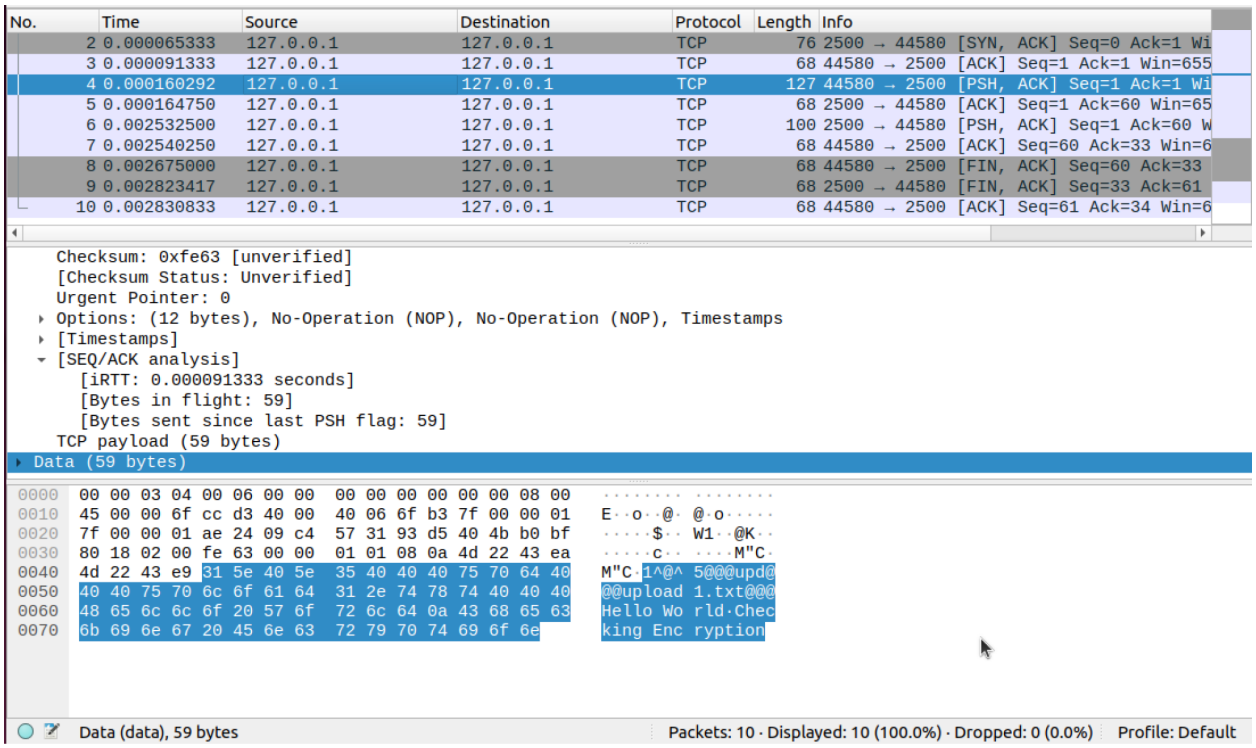
3. Checking Plain Text:
   Command: upd client_files/upload1.txt

   The actual encrypted command sent to the server and the encrypted response received are shown below (both hex and encrypted strings):

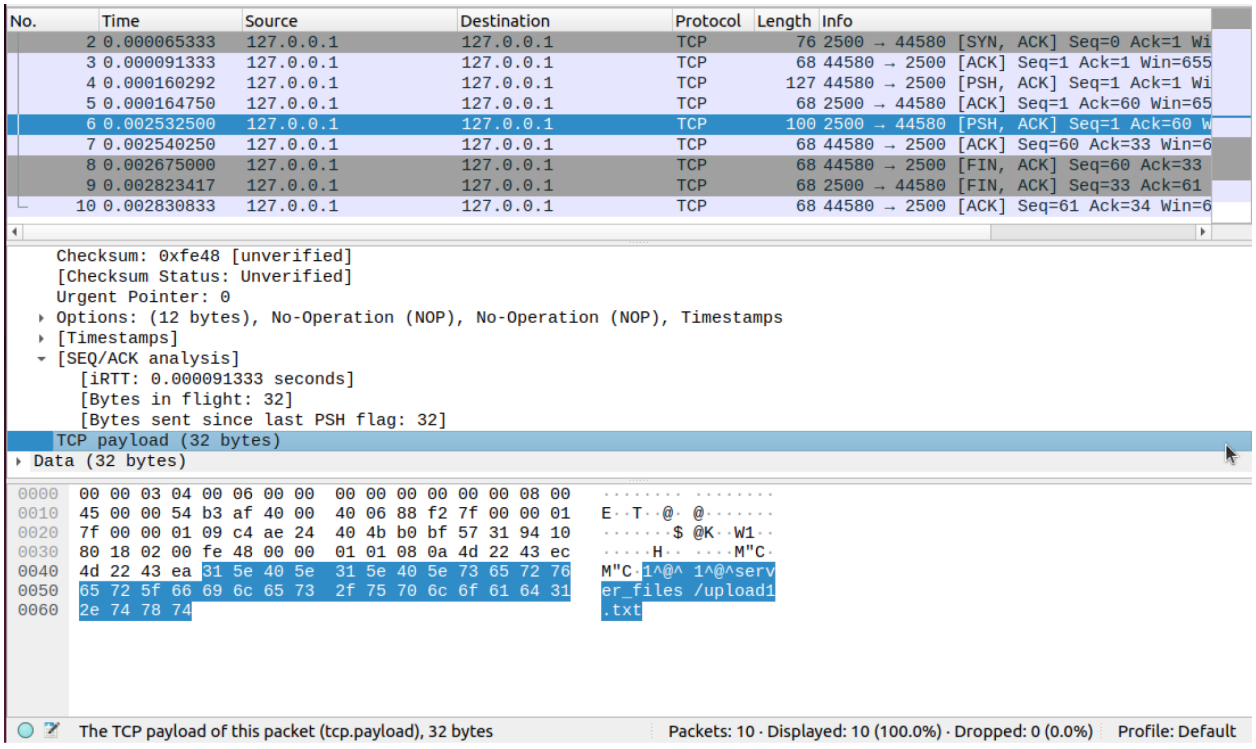
```
(base) shrreyasingh@Shrreyas-MacBook-Pro cs433_A1_rfs % python3 c_app.py

Enter any one of the below commands:
CWD
LS
CD dir_path
DWD server_file_path
UPD client_file_path

-----------------------------------------
upd client_files/upload1.txt

Enter an integer to select the encryption type:
1 Plain
2 Substitute
3 Transpose

-----------------------------------------------------
1

Setting up connection ...

Client Request Encrypted String: 1^@^5@@@upd@@@upload1.txt@@@Hello World
Checking Encryption

Hex string = 315e405e3540404075706440404075706c6f6164312e74787440404048656c6c6f20576f726c640a436865636b696e6720456e6372797074696f6e

Server Response Encrypted String: 1^@^1^@^server_files/upload1.txt

Hex string = 315e405e315e405e7365727665725f66696c65732f75706c6f6164312e747874

Connection closed :)
--------------------------------
Received: 'server_files/upload1.txt'
--------------------------------
STATUS: OK
```

   These results can be verified from the wireshark dump (Q1_A1_upd.pcapng). The snapshots of the request and response payloads (from WireShark dump) match the encrypted string and its hex code.

Client Request Payload:



Server Response Payload:

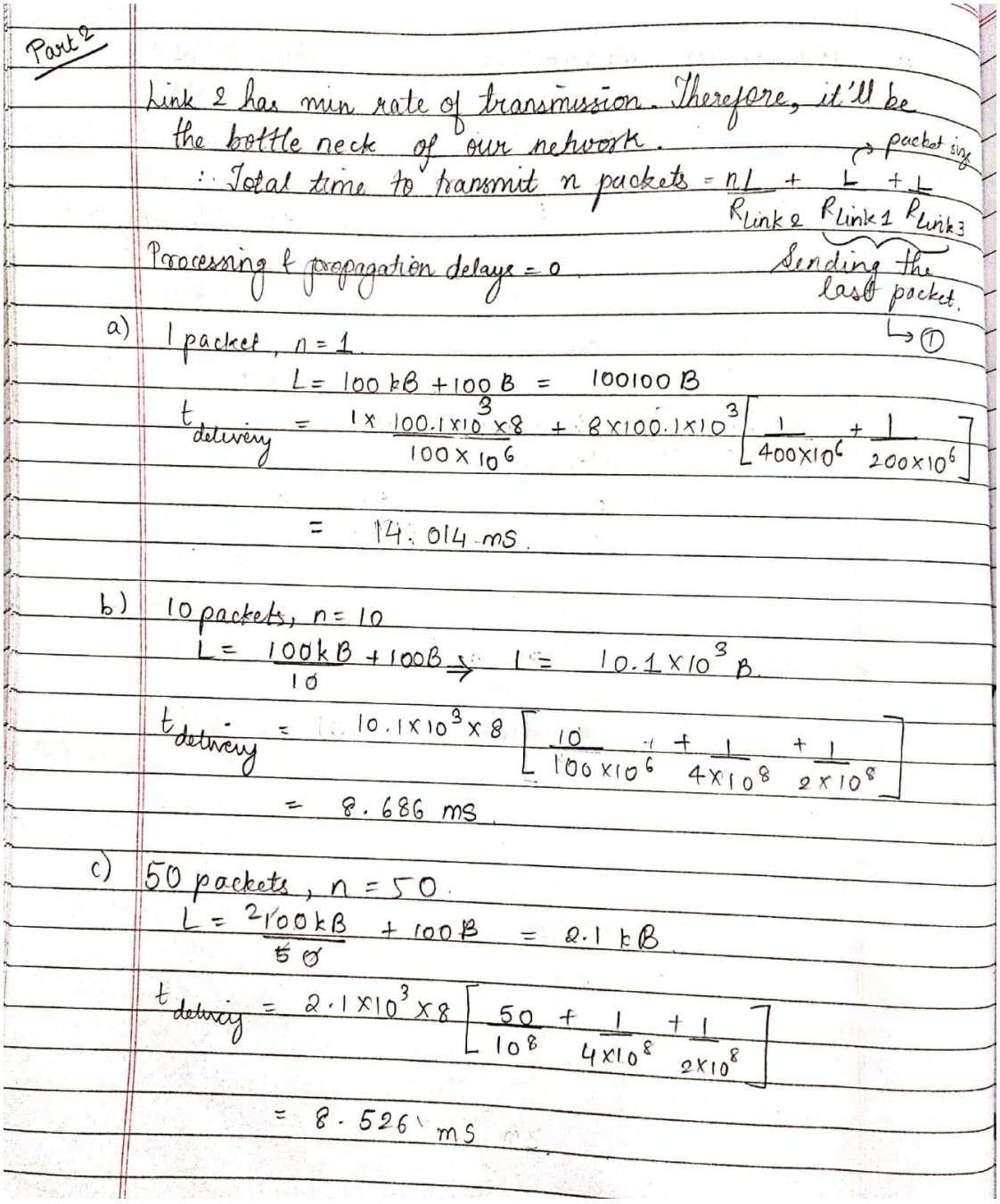


Analysis:

- In all the requests and responses, TCP protocol is used in accordance with our socket definition parameters of clients and servers (SOCK_STREAM).
- In case of encryption (both substitution and transpose) as well as no encryption, the data sent over the network (payload - both hex value and string), as captured by WireShark, exactly corresponds to the encrypted client request and server response (when explicitly printed out to the terminal before transmission).
- The cross verification is performed on three different commands. Thus, the crypto layer is functioning well in our Remote File System.
- The WireShark dump for all the three cases are present in the above mentioned repository.

Part 2:

Answer 2



Part 2

Link 2 has min rate of transmission. Therefore, it'll be the bottle neck of our network.

∴ Total time to transmit n packets = $\frac{nL}{R_{Link\,2}} + \underbrace{\frac{L}{R_{Link\,1}} + \frac{L}{R_{Link\,3}}}_{\text{Sending the last packet.}}$ ← packet sizes

Processing & propagation delays = 0

↳ ①

a) 1 packet, $n = 1$

$L = 100\,kB + 100\,B = 100100\,B$

$t_{delivery} = 1 \times \frac{100.1 \times 10^3 \times 8}{100 \times 10^6} + 8 \times 100.1 \times 10^3 \left[\frac{1}{400 \times 10^6} + \frac{1}{200 \times 10^6}\right]$

$= 14.014\,ms$

b) 10 packets, $n = 10$

$L = \frac{100\,kB}{10} + 100B \Rightarrow L = 10.1 \times 10^3\,B$

$t_{delivery} = 10.1 \times 10^3 \times 8 \left[\frac{10}{100 \times 10^6} + \frac{1}{4 \times 10^8} + \frac{1}{2 \times 10^8}\right]$

$= 8.686\,ms$

c) 50 packets, $n = 50$

$L = \frac{100\,kB}{50} + 100B = 2.1\,kB$

$t_{delivery} = 2.1 \times 10^3 \times 8 \left[\frac{50}{10^8} + \frac{1}{4 \times 10^8} + \frac{1}{2 \times 10^8}\right]$

$= 8.526\,ms$

d) 100 packets, $n = 100$

$L = \frac{100 \text{ kB}}{100} + 100\text{B} = 1.1\text{kB}$.

$t_{delivery} = 1.1 \times 10^3 \times 8 \left[\frac{100}{10^8} + \frac{1}{4\times10^6} + \frac{1}{2\times10^8} \right]$

$= 8.866$ ms

∴ Configuration c gives lowest delivery time (8.526 ms)

3. a) $t_{propagation} = \frac{len\ (link)}{V_{propagation}} = \frac{M}{S}$

$= \frac{10 \times 10^3}{2/3 \times 3 \times 10^8} \Rightarrow 5\times10^{-5}$ s.

$= 50\ \mu s$. [Propagation Delay]

b) Max bits that R_1 can send before 1st bit reaches R_2 $= \frac{RM}{S}$

$= \frac{100\times10^9 \times 10\times10^3}{2\times10^8}$

$= 5\times10^6$ bits

$S = 2/3\,C$ (C = speed of light)

c) Bit width of data in link $= \frac{S}{R} = \frac{2\times10^8}{100\times10^9} = 2\times10^{-3}$ m $= 2$ mm/bit

4) Let the file transfer time be = k sec/1 byte.

a) Non persistant: $2RTT + \underbrace{1000k}_{\text{transferring webpage (10 is less as compared to image download time)}} + \sum_{i=1}^{10} 2RTT + \left(\frac{100\times10^3}{}\right)k$

setting up connection + requesting webpage.

$= 22RTT + k[10^6 + 10^3]$

$= 0.22 + k[10^6 + 10^3]$ s.

b) Persistant connection : Connection need n't be setup again & again

$$2RTT + \frac{10^3}{k} + \sum_{i=1}^{10}\left(RTT + \frac{10^5}{k}\right) \text{ s}$$

$$= 12RTT + k(10^6 + 10^3)$$

$$= 0.12 + k(10^6 + 10^3) \text{ s}$$

c) Persistant + pipelined & dataframes of 1kB each. All images will be packed & sent in 1 RTT

$$2RTT + \frac{10^3}{k} + RTT + \sum_{i=1}^{10}\left(\frac{10^5}{k}\right)$$

$$= 3RTT + k(10^3 + 10^6)$$

$$= 0.03 + k(10^3 + 10^6) \text{ s}$$

Part3: Answer 5:

a) The new protocols are:

i) TLSv1.3 [Transport Layer Security Version 1.3]: It provides data encryption services and secures communication channels between two endpoints. It aims to encrypt handshakes to the maximum possible extent. It's the latest version of the internet's most deployed security channel with better security than previous versions and eliminates the usage of obsolete cryptographic algorithms. It is located between the application protocol layer and TCP/IP layer. Details about this protocol can be found in RFC 8446.

ii) QUIC [Quick UDP Internet Connections]: It's an experimental secure internet transport protocol designed by Google, similar to TCP+TLS+HTTP/2 implemented on UDP, with an aim to reduce the latency in comparison with TCP. It's a transport layer protocol whose details can be found in RFC 9000.

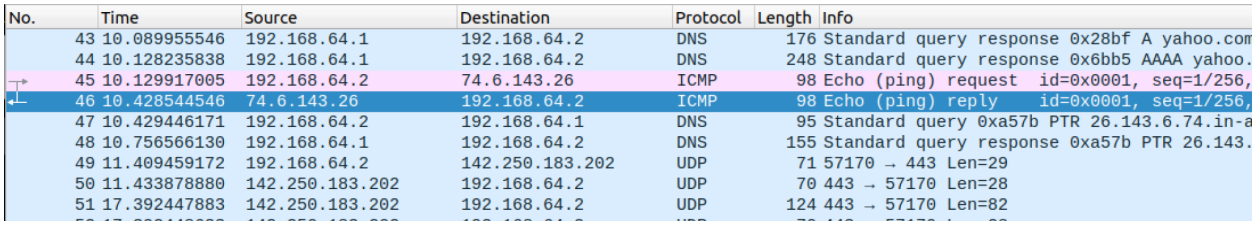
iii) MDNS [Multicast DNS]: It helps in resolution of names on smaller networks without using pre configured server [DNS] like in cases when local DNS doesn't exist. nDNS requests and responses are UDP with port 5353 as both source and destination ports. It is an application layer protocol with details present in RFC 6762.

iv) ICMPv6 [Internet Control Message Protocol]: It is Internet Control Message Protocol for IP version 6 and performs diagnostic and error reporting functions. It is layer 3 [Network Layer] protocol and details about it are present in RFC 4443.

v) NTP [Network Time Protocol]: It is an Application Layer protocol that helps in clock synchronization [aims to coordinate within a few milliseconds of UTC] between computer systems with variable latency data networks following packet switching. Details about its current version can be found in RFC 5905.

vi) OCSP [Online Certificate Status Protocol]: It is used to check digital certificate's validity. Browsers like FireFox use OCSP to validate HTTP certificates. It is Transport layer Internet Protocol with details in RFC 6960.

b) RTT for connection established by pinging yahoo.com (ping yahoo.com -c 1).
RTT = Time(reply {46}) - Time(request {45}) = 10.428544546 - 10.129917005 = 298.6 ms.



No.	Time	Source	Destination	Protocol	Length	Info
43	10.089955546	192.168.64.1	192.168.64.2	DNS	176	Standard query response 0x28bf A yahoo.com
44	10.128235838	192.168.64.1	192.168.64.2	DNS	248	Standard query response 0x6bb5 AAAA yahoo.
45	10.129917005	192.168.64.2	74.6.143.26	ICMP	98	Echo (ping) request id=0x0001, seq=1/256,
46	10.428544546	74.6.143.26	192.168.64.2	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256,
47	10.429446171	192.168.64.2	192.168.64.1	DNS	95	Standard query 0xa57b PTR 26.143.6.74.in-a
48	10.756566130	192.168.64.1	192.168.64.2	DNS	155	Standard query response 0xa57b PTR 26.143.
49	11.409459172	192.168.64.2	142.250.183.202	UDP	71	57170 → 443 Len=29
50	11.433878880	142.250.183.202	192.168.64.2	UDP	70	443 → 57170 Len=28
51	17.392447883	142.250.183.202	192.168.64.2	UDP	124	443 → 57170 Len=82

c)
As soon as I visited ims.iitgn.ac.in, 2 cookies were set up (_ga, _gid) and a third one was set up when I logged into the student portal (RequestToken). Cookies are stored as name-value pairs (specified in the corresponding headers below). Expiry date and time of the first 2 cookies are given under their respective “Expires” tab. The third cookie is a Session cookie i.e. it will expire as soon as the session is closed. Domain attribute species hosts that receive a cookie. Path specifies the webpage or the directory that sets up the cookie. The size of the cookie is specified under the size field. The sizes of cookie 1,2 and 3 are 30B, 31B and 36B respectively. Secure field checked implies the cookie is sent over HTTPS only with an encrypted request and whether the cookie can be retrieved by any server. The first 2 cookies are secure while the third one isn’t. HttpOnly (checked) prevents client’s access to cookie’s data. SameSite Lax implies a more relaxed form of protection of cross-site requests.