

ECE1150 ASSIGNMENT7

Yinhao Qian @ University of Pittsburgh

Q1

```
clearvars -except answer;
addrRang_C = {
    {'11100000 00000000 00000000 00000000'},
    {'11100000 00111111 11111111 11111111'},
    {'11100000 01000000 00000000 00000000'},
    {'11100000 01000000 11111111 11111111'},
    {'11100000 01000001 00000000 00000000'},
    {'11100001 01111111 11111111 11111111'}};
for i = 1:1:numel(addrRang_C)
    puts("link interface "+(i-1),'');
    generateForwardingTable(addrRang_C{i}{1}, ...
        addrRang_C{i}{2});
end
```

```

-> link interface 0
FROM -> 11100000000000000000000000000000
TO   -> 1110000000111111111111111111111111
TABLE -> 11100000000000000000000000000000
RES   -> 224.0.0.0 | 10
-> link interface 1
FROM -> 11100000010000000000000000000000
TO   -> 1110000001000000111111111111111111
TABLE -> 11100000010000000000000000000000
RES   -> 224.64.0.0 | 16
-> link interface 2
FROM -> 11100000010000010000000000000000
TO   -> 1110000101111111111111111111111111
TABLE -> 11100000000000000000000000000000
RES   -> 224.0.0.0 | 7
```

```
puts('all others go to link interface 3','');
```

```
-> all others go to link interface 3
```

Q2A

```
clearvars -except answer;
src = ["138.76.29.7:9001","138.76.29.7:9002","138.76.29.7:9003"];
dest = ["192.168.0.2:1533","192.168.0.3:2202","192.168.0.4:2489"];
dest2src = containers.Map(dest,src);
```

Source IP: port number Destination IP: port number 192.168.0.3:2202
136.142.34.104:80 192.168.0.4:2489 52.25.108.148:443

```
res = ["192.168.0.3:2202", "136.142.34.104:80";  
      "192.168.0.4:2489", "52.25.108.148:443"];
```

Original Table:

```
disp(res);
```

"192.168.0.3:2202"	"136.142.34.104:80"
"192.168.0.4:2489"	"52.25.108.148:443"

```
for i = 1:1:2%NAT translation process
    res(i,1) = convertCharsToStrings( ...
        dest2src(convertStringsToChars(res(i,1))));
end
```

Translated Table:

```
disp(res)
```

"138.76.29.7:9002"	"136.142.34.104:80"
"138.76.29.7:9003"	"52.25.108.148:443"

Q2B

```
src2dest = containers.Map(src,dest);
req = ["136.142.34.104:80", "138.76.29.7:9003";  
      "52.25.108.148:443", "138.76.29.7:9002"];
```

Original Table:

```
disp(req);
```

```
"136.142.34.104:80"    "138.76.29.7:9003"
"52.25.108.148:443"   "138.76.29.7:9002"
```

```
for i = 1:1:2%NAT translation process
    req(i,2) = convertCharsToStrings( ...
        src2dest(convertStringsToChars(req(i,2))));
end
```

Translated Table:

```
disp(req);
```

```
"136.142.34.104:80"    "192.168.0.4:2489"
"52.25.108.148:443"   "192.168.0.3:2202"
```

Q3

```
clearvars -except answer;
```

Reference Table (demonstrating known distance information):

```
V = {'x','y','z','u','v'};
x = [0,3,2,inf,3]';
y = [3,0,inf,2,inf]';
z = [2,inf,0,inf,6]';
u = [inf,2,inf,0,1]';
v = [3,inf,6,1,0]';
T_REF = table(x,y,z,u,v,'RowNames',V);
disp(T_REF);
```

	x	y	z	u	v
	---	---	---	---	---
x	0	3	2	Inf	3
y	3	0	Inf	2	Inf
z	2	Inf	0	Inf	6
u	Inf	2	Inf	0	1
v	3	Inf	6	1	0

Operating Table (used to calculate the shortest distance):

```
R = {'x','z','v'};
T = table(inf(1,3)',inf(1,3)',inf(1,3)',inf(1,3)',inf(1,3)', ...
    'VariableNames',V,'RowNames',R);
disp(T);
```

	x	y	z	u	v
x	Inf	Inf	Inf	Inf	Inf
z	Inf	Inf	Inf	Inf	Inf
v	Inf	Inf	Inf	Inf	Inf

```
isTableChanged_lA = [true,true,true];
for from = 1:1:numel(R)%from this node
    puts("testing row "+R{from},'row');
    T{R{from},R{from}} = 0;
    while true
        doMadeChanges_l = false;
        for gpas = 1:1:numel(V)%going pass this node
            for to = 1:1:numel(V)%to this node
                origCost_d = T{R{from},V{to}};
                newCost_d = T{R{from},V{gpas}}
                    +T_REF{V{gpas},V{to}};
                if newCost_d<origCost_d
                    doMadeChanges_l = true;
                    puts("from "+R{from} ...
                        +" ,it's less costly to go thru " ...
                        +V{gpas}+" to reach "+V{to} ...
                        +" since "+T{R{from},V{to}}+" > " ...
                        + T{R{from},V{gpas}} ...
                        + " + "+ T_REF{V{gpas},V{to}} ,');
                    T{R{from},V{to}} = newCost_d;
                    disp(T(from,:))
                end
            end
        end
    end
end
```

Instead of solely having 3 iterations, I'll keep it running until no more changes to the table are made (to show the final result)

```

        if(doMadeChanges_l==false)
            break
        end
    end
end
end

```

```

ROW  -> testing row x
      -> from x ,it's less costly to go thru x to reach y since Inf > 0 + 3
      x   y   z   u   v
      -   -   ---  ---  ---

x     0   3   Inf   Inf   Inf
      -> from x ,it's less costly to go thru x to reach z since Inf > 0 + 2
      x   y   z   u   v
      -   -   -   ---  ---

x     0   3   2   Inf   Inf
      -> from x ,it's less costly to go thru x to reach v since Inf > 0 + 3
      x   y   z   u   v
      -   -   -   ---  -

x     0   3   2   Inf   3
      -> from x ,it's less costly to go thru y to reach u since Inf > 3 + 2
      x   y   z   u   v
      -   -   -   -   -

x     0   3   2   5   3
      -> from x ,it's less costly to go thru v to reach u since 5 > 3 + 1
      x   y   z   u   v
      -   -   -   -   -

x     0   3   2   4   3
ROW  -> testing row z
      -> from z ,it's less costly to go thru z to reach x since Inf > 0 + 2
      x   y   z   u   v
      -   ---  -   ---  ---

z     2   Inf   0   Inf   Inf
      -> from z ,it's less costly to go thru z to reach v since Inf > 0 + 6
      x   y   z   u   v
      -   ---  -   ---  -

z     2   Inf   0   Inf   6
      -> from z ,it's less costly to go thru v to reach u since Inf > 6 + 1
      x   y   z   u   v

```

```

      -   ---   -   -   -
z    2    Inf    0    7    6
-> from z ,it's less costly to go thru x to reach y since  $\text{Inf} > 2 + 3$ 
    x    y    z    u    v
    -    -    -    -    -

z    2    5    0    7    6
-> from z ,it's less costly to go thru x to reach v since  $6 > 2 + 3$ 
    x    y    z    u    v
    -    -    -    -    -

z    2    5    0    7    5
-> from z ,it's less costly to go thru v to reach u since  $7 > 5 + 1$ 
    x    y    z    u    v
    -    -    -    -    -

z    2    5    0    6    5
ROW -> testing row v
-> from v ,it's less costly to go thru v to reach x since  $\text{Inf} > 0 + 3$ 
    x    y    z    u    v
    -    ---    ---    ---    -

v    3    Inf    Inf    Inf    0
-> from v ,it's less costly to go thru v to reach z since  $\text{Inf} > 0 + 6$ 
    x    y    z    u    v
    -    ---    -    ---    -

v    3    Inf    6    Inf    0
-> from v ,it's less costly to go thru v to reach u since  $\text{Inf} > 0 + 1$ 
    x    y    z    u    v
    -    ---    -    -    -

v    3    Inf    6    1    0
-> from v ,it's less costly to go thru x to reach y since  $\text{Inf} > 3 + 3$ 
    x    y    z    u    v
    -    -    -    -    -

v    3    6    6    1    0
-> from v ,it's less costly to go thru x to reach z since  $6 > 3 + 2$ 
    x    y    z    u    v
    -    -    -    -    -

v    3    6    5    1    0
-> from v ,it's less costly to go thru u to reach y since  $6 > 1 + 2$ 
    x    y    z    u    v

```

	-	-	-	-	-
v	3	3	5	1	0

Final table:

```
disp(T);
```

	x	y	z	u	v
	-	-	-	-	-
x	0	3	2	4	3
z	2	5	0	6	5
v	3	3	5	1	0

Q4

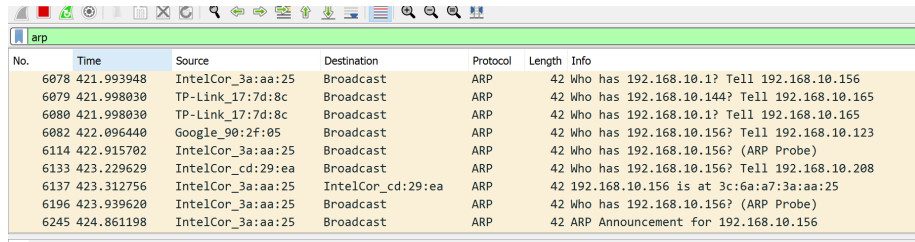
This command presented me withg the resolved MAC addresses saved in cache:

```
PS C:\WINDOWS\system32> arp -a
```

Interface: 192.168.10.208 --- 0x9

Internet Address	Physical Address	Type
192.168.10.1	80-3f-5d-d4-77-db	dynamic
192.168.10.123	ac-67-84-90-2f-05	dynamic
192.168.10.128	28-80-23-f6-0a-d8	dynamic
192.168.10.255	ff-ff-ff-ff-ff-ff	static
224.0.0.22	01-00-5e-00-00-16	static
224.0.0.251	01-00-5e-00-00-fb	static
224.0.0.252	01-00-5e-00-00-fc	static
239.255.255.250	01-00-5e-7f-ff-fa	static
255.255.255.255	ff-ff-ff-ff-ff-ff	static

No.	Time	Source	Destination	Protocol	Length	Info
6331	429.530782	Winstars_d4:77:db	IntelCor_cd:29:ea	ARP	42	Who has 192.168.10.208? Tell 192.168.10.1
6332	429.530802	IntelCor_cd:29:ea	Winstars_d4:77:db	ARP	42	192.168.10.208 is at 08:5b:d6:cd:29:ea
6360	433.974991	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.144? Tell 192.168.10.165
6361	433.985101	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.123? Tell 192.168.10.165
6362	433.985101	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.1? Tell 192.168.10.165
6364	436.022863	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.156? Tell 192.168.10.165
6379	436.944530	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.128? Tell 192.168.10.165
6392	445.955878	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.144? Tell 192.168.10.165
6393	446.058335	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.123? Tell 192.168.10.165



No.	Time	Source	Destination	Protocol	Length	Info
6078	421.993948	IntelCor_3a:aa:25	Broadcast	ARP	42	Who has 192.168.10.1? Tell 192.168.10.156
6079	421.998030	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.144? Tell 192.168.10.165
6080	421.998030	TP-Link_17:7d:8c	Broadcast	ARP	42	Who has 192.168.10.1? Tell 192.168.10.165
6082	422.096440	Google_90:2f:05	Broadcast	ARP	42	Who has 192.168.10.156? Tell 192.168.10.123
6114	422.915702	IntelCor_3a:aa:25	Broadcast	ARP	42	Who has 192.168.10.156? (ARP Probe)
6133	423.229629	IntelCor_cd:29:ea	Broadcast	ARP	42	Who has 192.168.10.156? Tell 192.168.10.208
6137	423.312756	IntelCor_3a:aa:25	IntelCor_cd:29:ea	ARP	42	192.168.10.156 is at 3c:6a:a7:3a:aa:25
6196	423.939620	IntelCor_3a:aa:25	Broadcast	ARP	42	Who has 192.168.10.156? (ARP Probe)
6245	424.861198	IntelCor_3a:aa:25	Broadcast	ARP	42	ARP Announcement for 192.168.10.156

As we can see, in arp process, both receiving and transmitting uses broadcast rather than unicast.

Unfortunately there are too many servers that require internet on my device, so it's very difficult to tell which one is for connecting to my.pitt.edu.

The purpose of arp messages is that it translates IP addresses and MAC addresses so that devices know what is the actual destination addresses so that they can send packets to each other.


Q5A

Transport Control Protocol aka TCP needs a three way handshake to make sure connection is established before sending data, and error checking is also present in TCP protocol (either stop-and-wait or sliding-window). It's more secure than UDP but slower and takes more resources.

User Datagram Protocol aka UDP does not need to establish any connections, and all data is broadcasted and doesn't care of the receiver's reaction. It's easier to use and has less overhead.

Q5B

I use Socket.IO for web programming, and it employs TCP protocol.


Socket.IO

Documentation
Introduction
Delivery guarantees
How it works
Logging and debugging
Testing
Troubleshooting
TypeScript
Server
Client
Events
Adapters


to a WebSocket server, and a WebSocket client will not be able to connect to a plain HTTP server either.

```
// WARNING: the client will NOT be able to connect!
const socket = io("ws://echo.websocket.org");
```

If you are looking for a plain WebSocket server, please take a look at [ws](#) or [uWebSockets.js](#).

There are also [discussions](#) for including a WebSocket server in the Node.js core.

On the client-side, you might be interested in the [robust-websocket](#) package.


CAUTION

Socket.IO is not meant to be used in a background service for mobile applications.

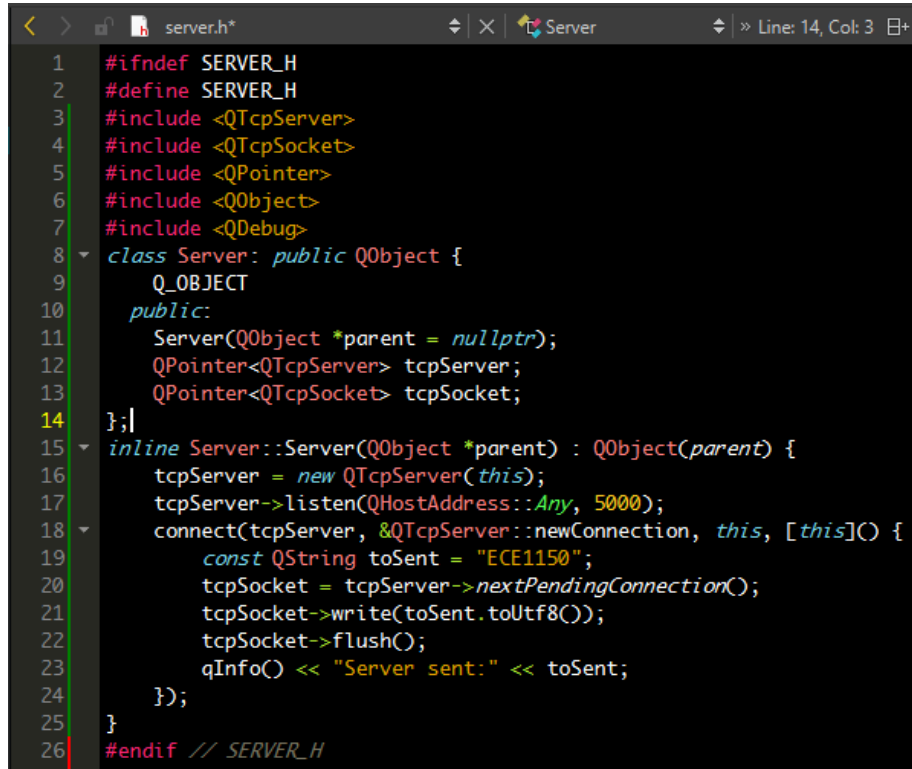
The Socket.IO library keeps an open **TCP connection** to the server, which may result in a high **battery drain for your users**. Please use a dedicated messaging platform like [FCM](#) for this use case.

Q6A

Unfortunately my MATLAB is having some trouble downloading the toolbox, so I used QT Framework instead and created the TCP Client and Server implementation in C++ (just as how it should be done in MATLAB).

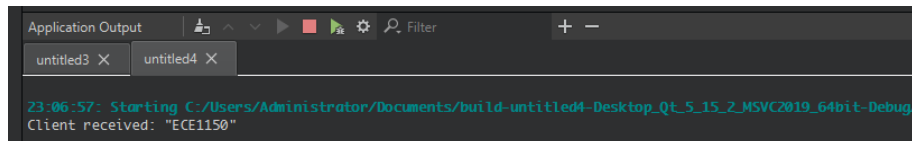
The port I used will be 5000. The message is sent in UTF-8 encoding.

TCP Server Setup:



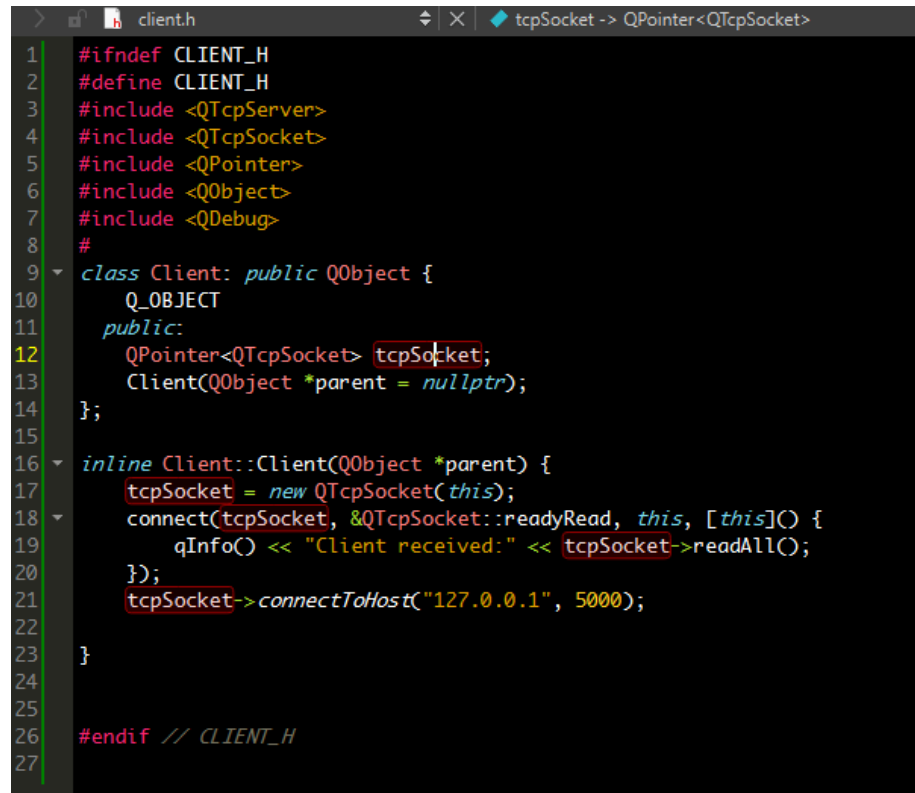
```
1  #ifndef SERVER_H
2  #define SERVER_H
3  #include <QTcpServer>
4  #include <QTcpSocket>
5  #include <QPointer>
6  #include <QObject>
7  #include <QDebug>
8  class Server: public QObject {
9      Q_OBJECT
10     public:
11         Server(QObject *parent = nullptr);
12         QPointer<QTcpServer> tcpServer;
13         QPointer<QTcpSocket> tcpSocket;
14     };
15     inline Server::Server(QObject *parent) : QObject(parent) {
16         tcpServer = new QTcpServer(this);
17         tcpServer->listen(QHostAddress::Any, 5000);
18         connect(tcpServer, &QTcpServer::newConnection, this, [this]() {
19             const QString toSent = "ECE1150";
20             tcpSocket = tcpServer->nextPendingConnection();
21             tcpSocket->write(toSent.toUtf8());
22             tcpSocket->flush();
23             qDebug() << "Server sent:" << toSent;
24         });
25     }
26 #endif // SERVER_H
```

TCP Server Terminal Output:



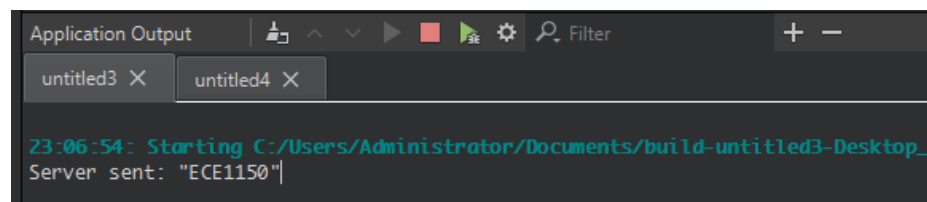
```
Application Output
untitled3 X untitled4 X
23:06:57: Starting C:/Users/Administrator/Documents/build-untitled4-Desktop_Qt_5_15_2_MSVC2019_64bit-Debug/d
Client received: "ECE1150"
```

TCP Client:



```
1  #ifndef CLIENT_H
2  #define CLIENT_H
3  #include <QTcpServer>
4  #include <QTcpSocket>
5  #include <QPointer>
6  #include <QObject>
7  #include <QDebug>
8  #
9  class Client: public QObject {
10     Q_OBJECT
11     public:
12     QPointer<QTcpSocket> tcpSocket;
13     Client(QObject *parent = nullptr);
14 };
15
16 inline Client::Client(QObject *parent) {
17     tcpSocket = new QTcpSocket(this);
18     connect(tcpSocket, &QTcpSocket::readyRead, this, [this]() {
19         qDebug() << "Client received:" << tcpSocket->readAll();
20     });
21     tcpSocket->connectToHost("127.0.0.1", 5000);
22 }
23
24
25
26 #endif // CLIENT_H
27
```

TCP Client Terminal Output:



```
Application Output
untitled3 X untitled4 X
23:06:54: Starting C:/Users/Administrator/Documents/build-untitled3-Desktop_
Server sent: "ECE1150"
```

Q6B

The first three packets are the three-way handshakes. It is used to ensure the connection has been established correctly in TCP protocol.

First packet: The sender generates a random initial sequence number (ISN) and sends it to destination.

Second packet: Destination generates a random initial sequence number and sends it along with the acknowledgment to sender.

Third packet: Sender acknowledges.

Q6C

No.	Time	Source	Destination	Protocol	Length	Info
197	28.976874	127.0.0.1	127.0.0.1	TCP	56	2985 → 5000 [SYN] Seq=4044466882 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
198	28.976912	127.0.0.1	127.0.0.1	TCP	56	5000 → 2985 [SYN, ACK] Seq=2553511635 Ack=4044466883 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
199	28.976930	127.0.0.1	127.0.0.1	TCP	44	2985 → 5000 [ACK] Seq=4044466883 Ack=2553511636 Win=2619648 Len=0
200	28.978050	127.0.0.1	127.0.0.1	TCP	51	5000 → 2985 [PSH, ACK] Seq=2553511636 Ack=4044466883 Win=2619648 Len=7
201	28.978074	127.0.0.1	127.0.0.1	TCP	44	2985 → 5000 [ACK] Seq=4044466883 Ack=2553511643 Win=2619648 Len=0

Q6D

Info

2985 → 5000 [SYN]	Seq=4044466882	Win=65535	Len=0	MSS=65495	WS=256	SACK_PERM=1
5000 → 2985 [SYN, ACK]	Seq=2553511635	Ack=4044466883	Win=65535	Len=0	MSS=65495	WS=256
2985 → 5000 [ACK]	Seq=4044466883	Ack=2553511636	Win=2619648	Len=0		
5000 → 2985 [PSH, ACK]	Seq=2553511636	Ack=4044466883	Win=2619648	Len=7		
2985 → 5000 [ACK]	Seq=4044466883	Ack=2553511643	Win=2619648	Len=0		

Q6E

Wireshark · Packet 200 · Adapter for loopback traffic capture

> Frame 200: 51 bytes on wire (408 bits), 51 bytes captured (408 bits) on interface
 > Null/Loopback
 > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 > Transmission Control Protocol, Src Port: 5000, Dst Port: 2985, Seq: 2553511636,
 > Data (7 bytes)

0000	02 00 00 00 45 00 00 2f 1b e8 40 00 80 06 00 00E../..@....
0010	7f 00 00 01 7f 00 00 01 13 88 0b a9 98 33 7e d43~.
0020	f1 11 aa c3 50 18 27 f9 cc 11 00 00 45 43 45 31	...P.'...ECE1
0030	31 35 30	150

Q7A

```
PS C:\WINDOWS\system32> ipconfig /flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.
PS C:\WINDOWS\system32> ping www.pitt.edu

Pinging pantheon-systems.map.fastly.net [146.75.34.133] with 32 bytes of data:
Reply from 146.75.34.133: bytes=32 time=19ms TTL=53
Reply from 146.75.34.133: bytes=32 time=22ms TTL=53
Reply from 146.75.34.133: bytes=32 time=31ms TTL=53
Reply from 146.75.34.133: bytes=32 time=25ms TTL=53

Ping statistics for 146.75.34.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 19ms, Maximum = 31ms, Average = 24ms
```

The image shows a Wireshark packet capture window titled "udp.stream eq 2". It displays a table of network traffic with columns: No., Time, Source, Destination, Protocol, Length, and Info. Two packets are visible:

No.	Time	Source	Destination	Protocol	Length	Info
79	22.204332	192.168.10.208	75.75.75.75	DNS	72	Standard query 0xfb7f A www.pitt.edu
81	22.214979	75.75.75.75	192.168.10.208	DNS	133	Standard query response 0xfb7f A www.pitt.edu

Below the table, a "Follow UDP Stream" window is open for the selected stream. It shows the raw data of the stream, which is a DNS query and response. The "Find" field is empty, and the "Find Next" button is highlighted.

Q7B

The image shows a Wireshark packet capture window titled "udp.stream eq 2". It displays a table of network traffic with columns: Source, Destination, Protocol, Length, and Info. Four packets are visible:

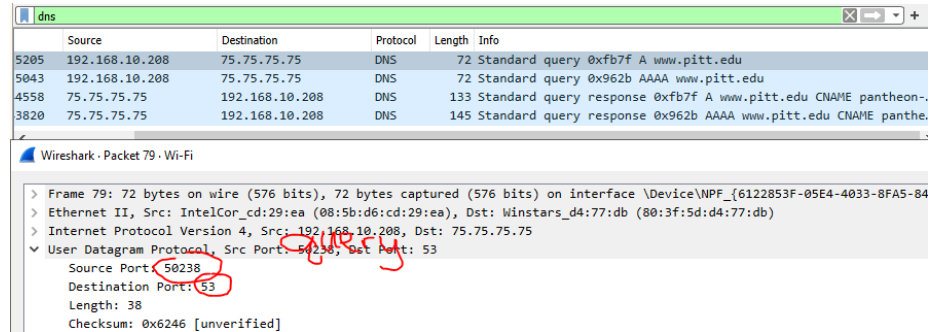
Source	Destination	Protocol	Length	Info
192.168.10.208	75.75.75.75	DNS	72	Standard query 0xfb7f A www.pitt.edu
192.168.10.208	75.75.75.75	DNS	72	Standard query 0x962b AAAA www.pitt.edu
75.75.75.75	192.168.10.208	DNS	133	Standard query response 0xfb7f A www.pitt.edu CNAME pantheon-systems.map.fastly.net.*.....K"
75.75.75.75	192.168.10.208	DNS	145	Standard query response 0x962b AAAA www.pitt.edu CNAME pantheon-systems.map.fastly.net.*.....K"

Below the table, a "Follow UDP Stream" window is open for the selected stream. It shows the raw data of the stream, which is a DNS query and response. The "Find" field is empty, and the "Find Next" button is highlighted.

It is sent over UDP protocol.

Q7C

Query Message:



	Source	Destination	Protocol	Length	Info
5205	192.168.10.208	75.75.75.75	DNS	72	Standard query 0xfb7f A www.pitt.edu
5043	192.168.10.208	75.75.75.75	DNS	72	Standard query 0x962b AAAA www.pitt.edu
4558	75.75.75.75	192.168.10.208	DNS	133	Standard query response 0xfb7f A www.pitt.edu CNAME pantheon-
3820	75.75.75.75	192.168.10.208	DNS	145	Standard query response 0x962b AAAA www.pitt.edu CNAME panthe-

Wireshark · Packet 79 · Wi-Fi

> Frame 79: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \Device\NPF_{6122853F-05E4-4033-8FA5-84...}

> Ethernet II, Src: IntelCor_cd:29:ea (08:5b:d6:cd:29:ea), Dst: Winstars_d4:77:db (80:3f:5d:d4:77:db)

> Internet Protocol Version 4, Src: 192.168.10.208, Dst: 75.75.75.75

▼ User Datagram Protocol, Src Port: 50238, Dst Port: 53

Source Port: 50238

Destination Port: 53

Length: 38

Checksum: 0x6246 [unverified]

Source Port : 50238

Destination Port: 53

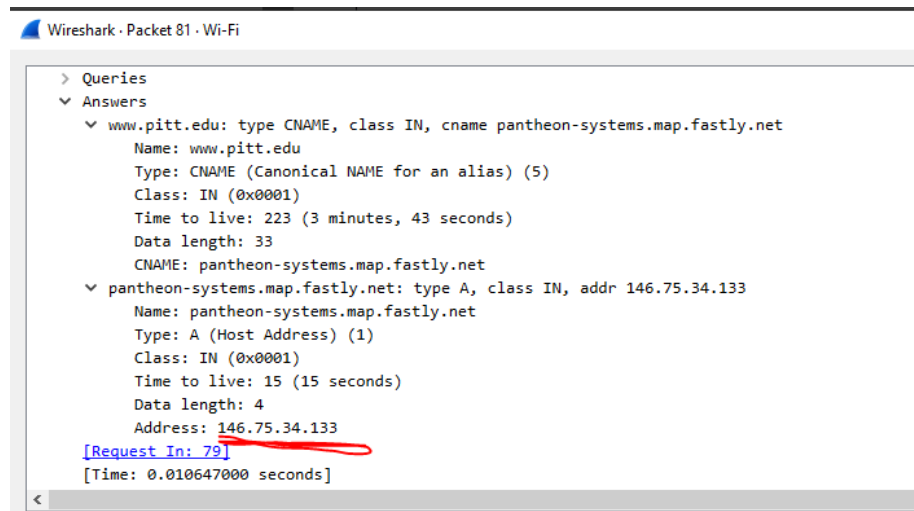
Response Message:

From the picture in Q7B, we are able to tell:

Source Port: 53

Destination Port: 50238

Q7D



Wireshark · Packet 81 · Wi-Fi

> Queries

▼ Answers

▼ www.pitt.edu: type CNAME, class IN, cname pantheon-systems.map.fastly.net

Name: www.pitt.edu

Type: CNAME (Canonical NAME for an alias) (5)

Class: IN (0x0001)

Time to live: 223 (3 minutes, 43 seconds)

Data length: 33

CNAME: pantheon-systems.map.fastly.net

▼ pantheon-systems.map.fastly.net: type A, class IN, addr 146.75.34.133

Name: pantheon-systems.map.fastly.net

Type: A (Host Address) (1)

Class: IN (0x0001)

Time to live: 15 (15 seconds)

Data length: 4

Address: 146.75.34.133

[Request In: 79]

[Time: 0.010647000 seconds]

It is 146.75.34.133.

END OF HOMEWORK

The remaining contents are function definitions used mainly in Question 2.

```
function [addr,nBit] = generateForwardingTable (arg1_cA,arg2_cA)
%for q2
arg1_cA = strrep(arg1_cA,' ','');
arg2_cA = strrep(arg2_cA,' ','');
assert(numel(arg1_cA)==numel(arg2_cA));
nBit = find(arg1_cA~=arg2_cA, 1, 'first')-1;
puts(getEmphCharArray(arg1_cA,1:nBit),'from');
puts(getEmphCharArray(arg2_cA,1:nBit),'to');
% final_cA = bin2dec(reshape(arg1_cA,8,[]))'
final_cA = cat(2,arg1_cA(1:nBit),repmat('0',1,32-nBit));
puts(getEmphCharArray(final_cA,1:nBit),'table');
addr = bin2dec(reshape(final_cA,8,[]))';
puts(sprintf("%i.%i.%i.%i | %i", ...
    addr(1),addr(2),addr(3),addr(4),nBit),'res');
end

function [res] = getEmphCharArray (arg1_cA,ind_dA)
res = "";
for i = 1:1:numel(arg1_cA)
    if(ind_dA(1)<=i&&i<=ind_dA(end))
        res = res+sprintf("<strong>%c</strong>",arg1_cA(i));
    else
        res = res+sprintf("%c",arg1_cA(i));
    end
end
end

function [] = puts(arg1_cA,arg2_cA)
fprintf("<strong>%-5s</strong> -> %s\n",upper(arg2_cA),arg1_cA);
end
```