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    -> 11100000001111111111111111111111
TABLE -> 11100000000000000000000000000000
RES   -> 224.0.0.0 | 10
      -> link interface 1
FROM  -> 11100000010000000000000000000000
TO    -> 11100000010000001111111111111111
TABLE -> 11100000010000000000000000000000
RES   -> 224.64.0.0 | 16
      -> link interface 2
FROM  -> 11100000010000010000000000000000
TO    -> 11100001011111111111111111111111
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 (demostrating 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
```

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
```

```
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 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 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 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 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 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:

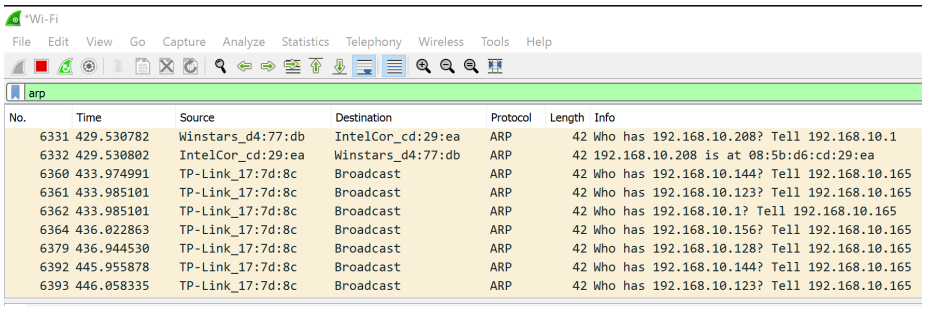
```
disp(T);
```

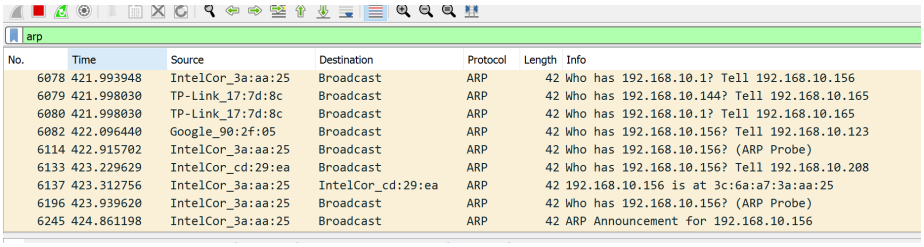
```
        x     y     z     u     v
        -     -     -     -     -

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
```





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

Unfortunately there are too many servers that requires 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.

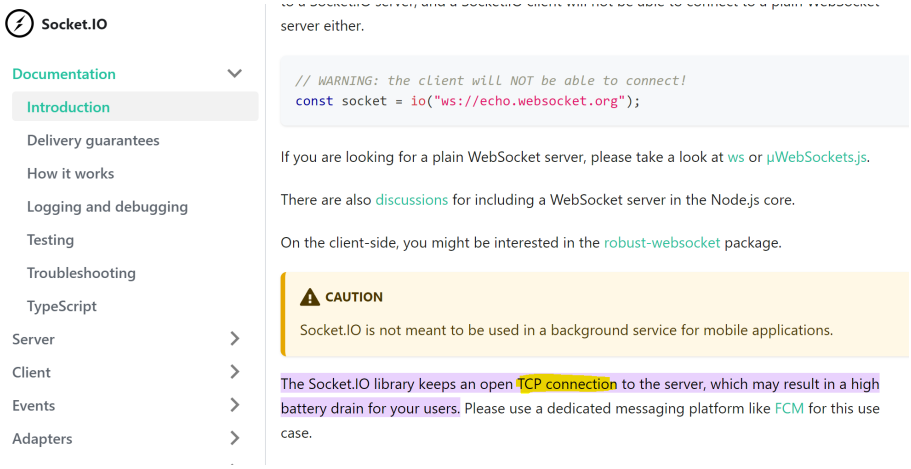
Q5A

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 it 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.



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).

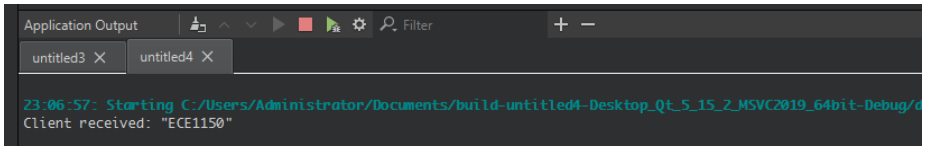
The port I used will be 5000. The message is sent in UTF-8 encoding.

TCP Server Setup:

server.h* | Server | » Line: 14, Col: 3

```
#ifndef SERVER_H
#define SERVER_H
#include <QTcpServer>
#include <QTcpSocket>
#include <QPointer>
#include <QObject>
#include <QDebug>
class Server: public QObject {
    Q_OBJECT
  public:
    Server(QObject *parent = nullptr);
    QPointer<QTcpServer> tcpServer;
    QPointer<QTcpSocket> tcpSocket;
};
inline Server::Server(QObject *parent) : QObject(parent) {
    tcpServer = new QTcpServer(this);
    tcpServer->listen(QHostAddress::Any, 5000);
    connect(tcpServer, &QTcpServer::newConnection, this, [this]() {
        const QString toSent = "ECE1150";
        tcpSocket = tcpServer->nextPendingConnection();
        tcpSocket->write(toSent.toUtf8());
        tcpSocket->flush();
        qInfo() << "Server sent:" << toSent;
    });
}
#endif // SERVER_H
```

TCP Server Terminal Output:

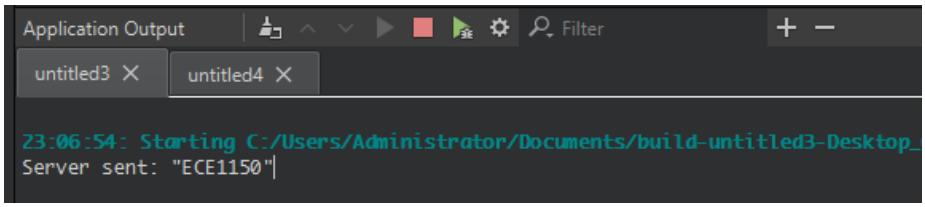


TCP Client:

client.h | tcpSocket -> QPointer<QTcpSocket>

```
#ifndef CLIENT_H
#define CLIENT_H
#include <QTcpServer>
#include <QTcpSocket>
#include <QPointer>
#include <QObject>
#include <QDebug>
#
class Client: public QObject {
    Q_OBJECT
  public:
    QPointer<QTcpSocket> tcpSocket;
    Client(QObject *parent = nullptr);
};

inline Client::Client(QObject *parent) {
    tcpSocket = new QTcpSocket(this);
    connect(tcpSocket, &QTcpSocket::readyRead, this, [this]() {
        qInfo() << "Client received:" << tcpSocket->readAll();
    });
    tcpSocket->connectToHost("127.0.0.1", 5000);

}


#endif // CLIENT_H

```

TCP Client Terminal Output:



Q6B

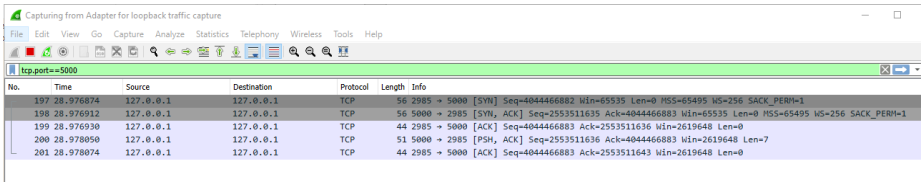
The first three packets are the three-way handshakes. It is used to ensure the connection has been established correctly in TCP protocol.

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

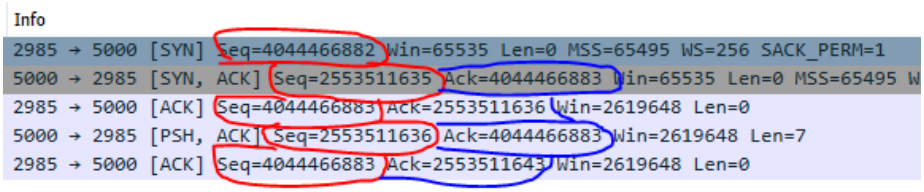
Second packet: Destination generates a random initial sequence number and sends it along with the acknowledgment to sender.

Third packet: Sender acknowledge.

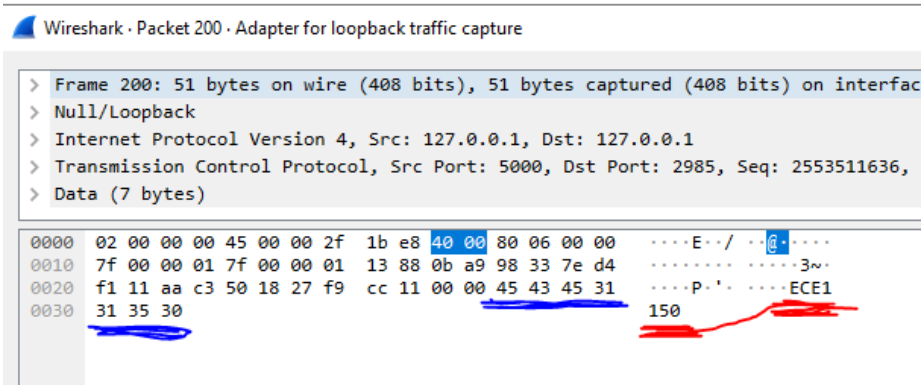
Q6C



Q6D

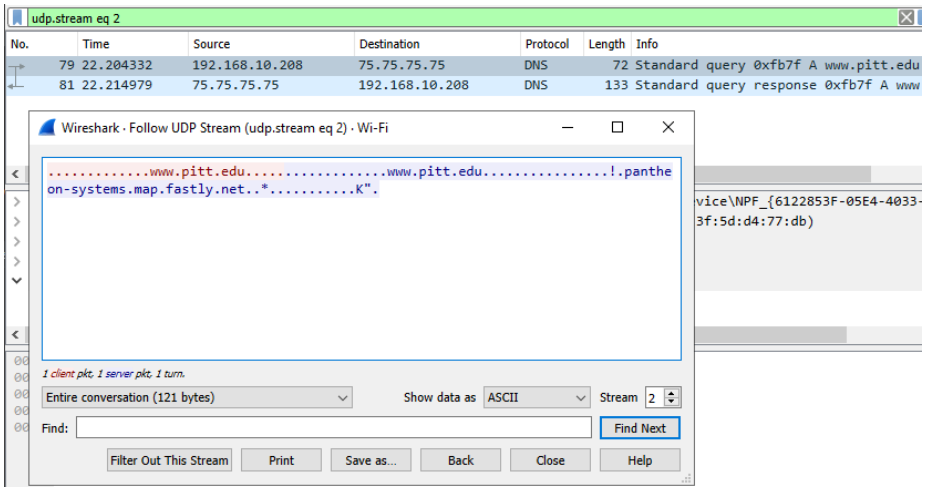


Q6E

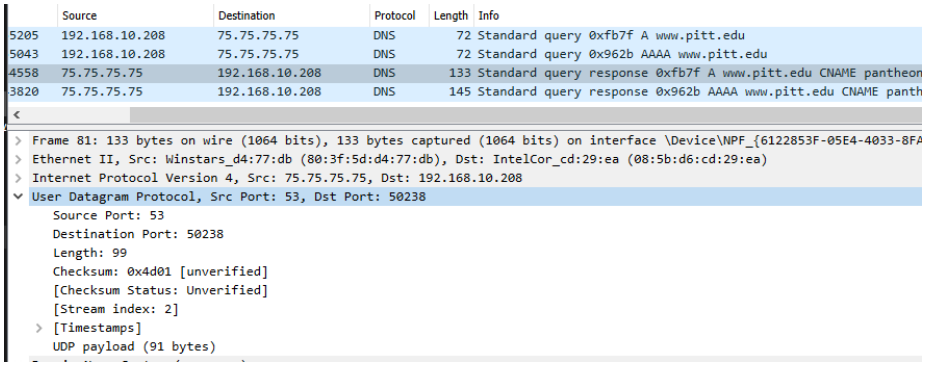


Q7A

```
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
```



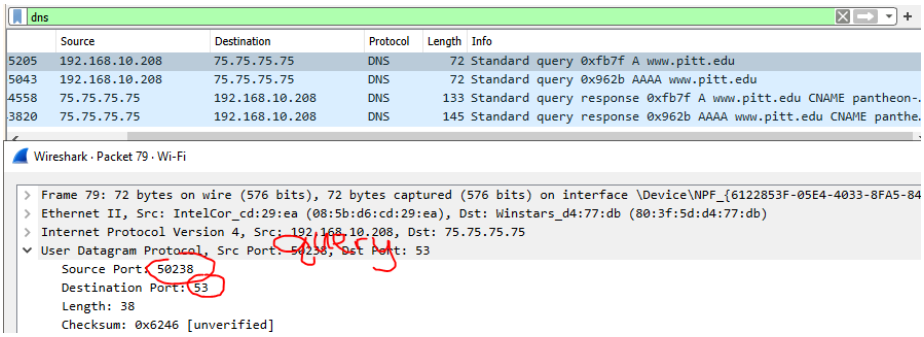
Q7B



It is sent over UDP protocol.

Q7C

Query Message:



Source Port : 50238

Destination Port: 53

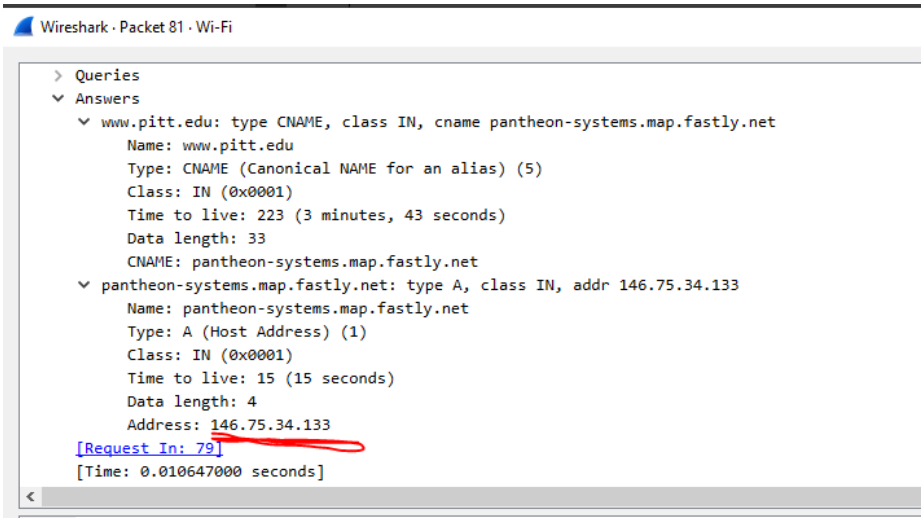
Response Message:

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

Source Port: 53

Destination Port: 50238

Q7D



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
```