CS 359

COMPUTER NETWORK LAB

ASSIGNMENT 1

Assignment - 1A

1. List 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.

1. TCP (Transmission Control Protocol)
2. UDP (User Datagram Protocol)
3. QUIC

2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received?

Time of HTTP (GET) request: 15:45:00.630942044

Time of HTTP (OK) response: 15:45:00.946305147

Time taken = 315.36 ms

3. What is the Internet address of the gaia.cs.umass.edu (also known as www net.cs.umass.edu)? What is the Internet address of your computer?

Internet address of the gaia.cs.umass.edu : 128.119.245.12

Internet address of my computer : 192.168.43.180

4. Print the two HTTP messages (GET and OK) referred to in question 2 above.

HTTP (GET) request

/tmp/wireshark_wlp3s0FWT7F1.pcapng 377 total packets, 2 shown

```
No.     Time              Source                Destination           Protocol Length Info
     49 15:45:00.630942044 192.168.43.180        128.119.245.12        HTTP     553    GET /wireshark-
labs/INTRO-wireshark-file1.html HTTP/1.1
Frame 49: 553 bytes on wire (4424 bits), 553 bytes captured (4424 bits) on interface wlp3s0, id 0
Ethernet II, Src: HonHaiPr_87:1d:21 (80:2b:f9:87:1d:21), Dst: XiaomiCo_e1:35:76 (20:a6:0c:e1:35:76)
Internet Protocol Version 4, Src: 192.168.43.180, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 43402, Dst Port: 80, Seq: 1, Ack: 1, Len: 487
Hypertext Transfer Protocol
    GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\n
    Host: gaia.cs.umass.edu\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99
Safari/537.36\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,hi;q=0.7\r\n
    \r\n
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html]
    [HTTP request 1/1]
    [Response in frame: 62]
```

HTTP (OK) Response

/tmp/wireshark_wlp3s0FWT7F1.pcapng 377 total packets, 2 shown

```
No.     Time              Source                Destination           Protocol Length Info
     62 15:45:00.946305147 128.119.245.12        192.168.43.180        HTTP     504    HTTP/1.1 200 OK
(text/html)
Frame 62: 504 bytes on wire (4032 bits), 504 bytes captured (4032 bits) on interface wlp3s0, id 0
Ethernet II, Src: XiaomiCo_e1:35:76 (20:a6:0c:e1:35:76), Dst: HonHaiPr_87:1d:21 (80:2b:f9:87:1d:21)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.43.180
Transmission Control Protocol, Src Port: 80, Dst Port: 43402, Seq: 1, Ack: 488, Len: 438
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Sun, 23 Jan 2022 10:15:00 GMT\r\n
    Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.25 mod_perl/2.0.11 Perl/v5.16.3\r\n
    Last-Modified: Sun, 23 Jan 2022 06:59:01 GMT\r\n
    ETag: "51-5d63a6305fa30"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 81\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.315363103 seconds]
    [Request in frame: 49]
    [Request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html]
    File Data: 81 bytes
Line-based text data: text/html (3 lines)
```

Assignment - 1B

main.cpp

```
#include <bits/stdc++.h>
#include <fstream>
using namespace std;

class Source
{
public:
   int sourceId;
   double rate;
   int bandwidth;
};

class Packet
{
public:
   int sourceId;
   int currentEventId;
   double genTimestamp;
   double transTimestamp;
   double queueTimestamp;
   double sinkTimestamp;
};

// How to compare elements
struct my_comparator
{
   // queue elements are vectors so we need to compare those
   bool operator()(Packet const &a, Packet const &b) const
   {
       return a.sinkTimestamp > b.sinkTimestamp;
   }
};

int totalTime, switchBandwidth, queueLimit, packetSize;
int numOfSources = 4;
vector<Source> sources(numOfSources);
int numOfPackets;
priority_queue<Packet, vector<Packet>, my_comparator> globalQueue;
int numOfSinkedPackets,numOfLostPackets;
vector<Packet> sinkedPackets,lostPackets;
ofstream graph1, graph2;

void getSources()
{
   for (int i = 0; i < numOfSources; i++)
   {
       cout << "\nDetails for Source " << i + 1 << "\n";
       sources[i].sourceId = i + 1;
       cout << "Enter packet sending rate (number of packets send/second):";
       cin >> sources[i].rate;
       cout << "Enter bandwidth (in kb/second):";
       cin >> sources[i].bandwidth;
   }
}
```

```
void getPackets()
{
   for (int i = 0; i < sources.size(); i++)
       for (double j = 0; j < totalTime; j += 1 / (double)(sources[i].rate))
       {
           Packet packet;
           packet.sourceId = i + 1;
           packet.currentEventId = 1;
           packet.genTimestamp = j;
           packet.sinkTimestamp = packet.genTimestamp;
           globalQueue.push(packet);
       }
   numOfPackets = globalQueue.size();
}

void processPackets()
{
   // SwitchTime maintained for switch so that it can transmit one packet at a
time.
   // SourceTime maintained for each source so that one source can transmit
one packet at a time.
   int queueSize = 0;
   double switchTime = 0;
   vector<double> sourceTime(numOfSources, 0);
   while (!globalQueue.empty())
   {
       Packet packet = globalQueue.top();
       // Transmit packet from source
       if (packet.currentEventId == 1)
       {
           packet.currentEventId = 2;
           packet.transTimestamp = sourceTime[packet.sourceId - 1] <
packet.genTimestamp ? packet.genTimestamp : sourceTime[packet.sourceId - 1];
           packet.transTimestamp += (double)packetSize /
sources[packet.sourceId - 1].bandwidth;
           packet.sinkTimestamp = packet.transTimestamp;
           sourceTime[packet.sourceId - 1] = packet.queueTimestamp;
           globalQueue.push(packet);
       }
       // Add packet to queue if space available
       else if (packet.currentEventId == 2)
       {
           if (queueLimit == -1 || queueSize < queueLimit)
           {
               packet.currentEventId = 3;
               packet.queueTimestamp = switchTime < packet.transTimestamp ?
packet.transTimestamp : switchTime;
               packet.sinkTimestamp = packet.queueTimestamp +
(double)packetSize / switchBandwidth;
               switchTime = packet.sinkTimestamp;
               globalQueue.push(packet);
               if (queueLimit != -1)
                   queueSize++;
           }
           else
               lostPackets.push_back(packet);
       }
```

```
        // Transmit packet from switch to sink
        else if (packet.currentEventId == 3)
        {
            packet.currentEventId = 4;
            sinkedPackets.push_back(packet);
            if (queueLimit != -1)
                queueSize--;
        }
        globalQueue.pop();
    }
    numOfLostPackets = lostPackets.size();
    numOfSinkedPackets = sinkedPackets.size();
}

double calculatePacketArrivalRate()
{
    double avgSourceBandwidth = 0, rate = 0, totalPacketRate = 0;
    for (int i = 0; i < numOfSources; i++)
        rate += min(sources[i].rate, (double)sources[i].bandwidth /
packetSize);
    return rate;
}

double calculateSystemTransmissionCapacity()
{
    double rate = (double)switchBandwidth / packetSize;
    return rate;
}

double calculateAverageDelay()
{
    double delay = 0;
    for (int i = 0; i < numOfSinkedPackets; i++)
      delay += sinkedPackets[i].sinkTimestamp - sinkedPackets[i].genTimestamp;
    delay /= numOfSinkedPackets;
    return delay;
}

double calculatePacketLossRate()
{
    int numOfLostPackets = numOfPackets - numOfSinkedPackets;
    double packetLossRate = (double)numOfLostPackets / numOfPackets;
    return packetLossRate;
}

void simulateNetwork()
{
    sinkedPackets.clear();
    lostPackets.clear();
    getPackets();
    processPackets();
    cout << fixed << setprecision(2);
    if (queueLimit == -1)
        graph1 << calculatePacketArrivalRate() /
calculateSystemTransmissionCapacity() << " " << calculateAverageDelay() <<
"\n";
    else
```

```
       graph2 << calculatePacketArrivalRate() /
calculateSystemTransmissionCapacity() << " " << calculatePacketLossRate() <<
"\n";
}

int main()
{

   graph1.open("graph1.txt");
   graph2.open("graph2.txt");
   int ql;
   cout << "Enter time for experiment (in seconds):";
   cin >> totalTime;
   cout << "Switch queue limit:";
   cin >> ql;
   cout << "Enter packet size (in kb):";
   cin >> packetSize;
   getSources();
   for (int i = 2000; i >= 100; i -= 0.05)
   {
       switchBandwidth = i;
       queueLimit = -1;
       simulateNetwork();
       queueLimit = ql;
       simulateNetwork();
   }
   graph1.close();
   graph2.close();
   system("python plotGraphs.py");
   return 0;
}
```

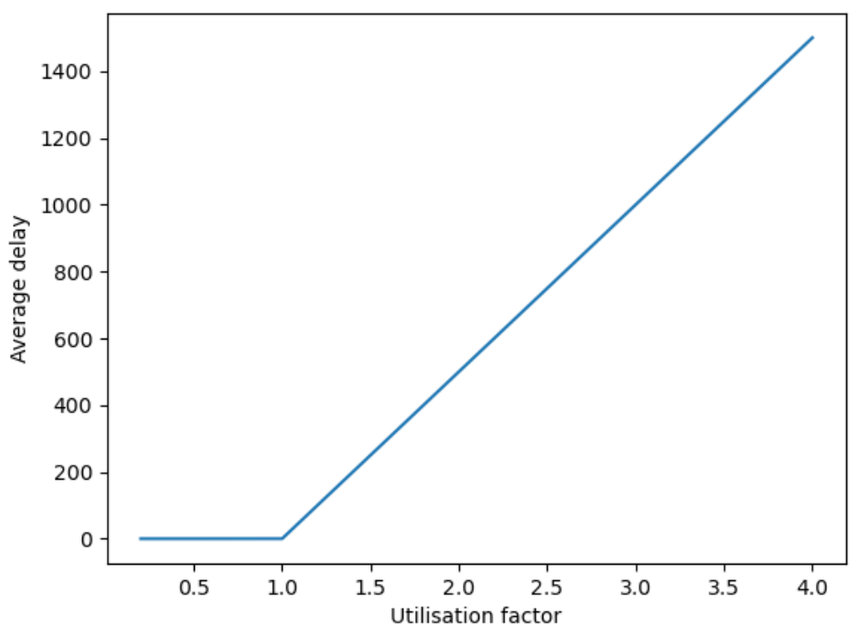
plotGraphs.py

```
import pandas as pd
import matplotlib.pyplot as plt

#Graph-1
df = pd.read_csv("graph1.txt", sep=' ')
df.columns=['x','y']
x = df['x'].tolist()
y = df['y'].tolist()
plt.plot(x,y)
plt.xlabel("Utilisation factor")
plt.ylabel("Average delay")
plt.show()

#Graph-2
df = pd.read_csv("graph2.txt", sep=' ')
df.columns=['x','y']
x = df['x'].tolist()
y = df['y'].tolist()
plt.plot(x,y)
plt.xlabel("Utilisation factor")
plt.ylabel("Packet loss rate")
plt.show()
```

Running the simulation

```
sachan@CLOUD-DESK:~/Documents/VI Semester/CS358/Lab/Lab 1$ g++ ./main.cpp
sachan@CLOUD-DESK:~/Documents/VI Semester/CS358/Lab/Lab 1$ ./a.out
Enter time for experiment (in seconds):1000
Switch queue limit:20
Enter packet size (in kb):20

Details for Source 1
Enter packet sending rate (number of packets send/second):3
Enter bandwidth (in kb/second):100

Details for Source 2
Enter packet sending rate (number of packets send/second):4
Enter bandwidth (in kb/second):80

Details for Source 3
Enter packet sending rate (number of packets send/second):5
Enter bandwidth (in kb/second):100

Details for Source 4
Enter packet sending rate (number of packets send/second):8
Enter bandwidth (in kb/second):200
```

Graph between Average delay & Utilization factor when queue limit is infinite:



Graph between Packet loss rate & Utilization factor when queue size is defined: