CS359-Assignment 1

1901CS75 Siddharth Sanskritayan

1a)

1. 3 different protocols that appear in the protocol column in the unfiltered packet-listing window:
   HTTP, TCP, QUIC

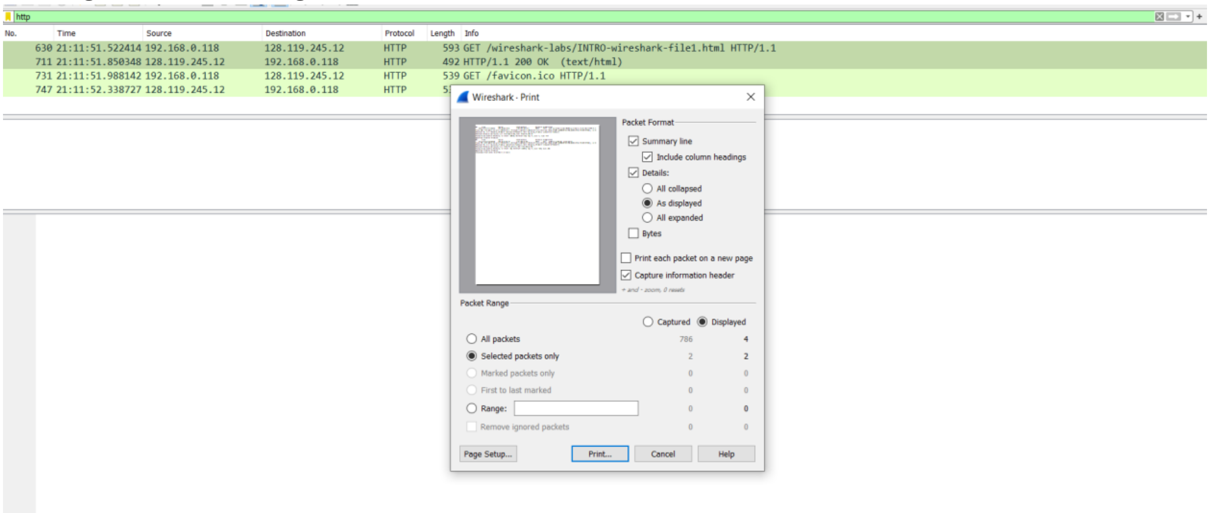
No.	Time	Source	Destination	Protocol	Length	Info
626	21:11:51.511572	192.168.0.118	142.251.12.188	TCP	66	60930 → 5228 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
627	21:11:51.516199	192.168.0.118	142.250.77.67	QUIC	1292	Initial, DCID=aaa60ad78c45cc44, PKN: 1, CRYPTO, CRYPTO, PADDING, PING,
628	21:11:51.522110	128.119.245.12	192.168.0.118	TCP	66	80 → 60926 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1440 SACK_PERM=1
629	21:11:51.522176	192.168.0.118	128.119.245.12	TCP	54	60926 → 80 [ACK] Seq=1 Ack=1 Win=132352 Len=0
630	21:11:51.522414	192.168.0.118	128.119.245.12	HTTP	593	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1

2. HTTP GET message sent at 21:11:51.522414
   Time when OK received 21:11:51.850348
   Total time taken = 51.850348 – 51.522414 = 0.327934 seconds

630	21:11:51.522414	192.168.0.118	128.119.245.12	HTTP	593	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1
711	21:11:51.850348	128.119.245.12	192.168.0.118	HTTP	492	HTTP/1.1 200 OK (text/html)

3. Internet address of the gaia.cs.umass.edu : 128.119.245.12
   Internet address of my computer: 192.168.0.118

4. Printing the HTTP messages:



Contents:

```
No.     Time           Source                Destination           Protocol Length Info
    175 16:17:39.251177    192.168.0.118         128.119.245.12        HTTP     542    GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1

Frame 175: 542 bytes on wire (4336 bits), 542 bytes captured (4336 bits) on interface \Device\NPF_{0D0601F8-576B-486D-9762-1532011E7894}, id 0
Ethernet II, Src: IntelCor_79:08:1f (58:a0:23:79:08:1f), Dst: Tp-LinkT_c5:40:ec (d8:47:32:c5:40:ec)
Internet Protocol Version 4, Src: 192.168.0.118, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 51235, Dst Port: 80, Seq: 1, Ack: 1, Len: 488
Hypertext Transfer Protocol

No.     Time           Source                Destination           Protocol Length Info
    179 16:17:39.586551    128.119.245.12        192.168.0.118         HTTP     492    HTTP/1.1 200 OK  (text/html)

Frame 179: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits) on interface \Device\NPF_{0D0601F8-576B-486D-9762-1532011E7894}, id 0
Ethernet II, Src: Tp-LinkT_c5:40:ec (d8:47:32:c5:40:ec), Dst: IntelCor_79:08:1f (58:a0:23:79:08:1f)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.0.118
Transmission Control Protocol, Src Port: 80, Dst Port: 51235, Seq: 1, Ack: 489, Len: 438
Hypertext Transfer Protocol
Line-based text data: text/html (3 lines)
```

1b)

Case 1: When queue size is infinite

```
//Packet switching network simulation for a fixed duration using n sources(here n = 4) connected to a switch via n links having
//bandwidth R1 and the switch is connected to a sink with a link of bandwidth R2
//In this case queue size is infinite

//priority queue (min heap) to store a vector of 4 doubles :{t_occ,t_gen,src_id,status}
//t_occ is the time of occurrence of the event
//t_gen is the time of generation of the packet
//src_id is the source id of the source from where packet is generated
//status denotes event status having 4 different values:
//0 : packet is generated and ready to transfer to the switch via the link
//1 : packet is in the queue
//2 : packet is ready to dispatch from the queue
//3 : packet is received at the sink

priority_queue<vector<double>,vector<vector<double>>,greater<vector<double>>>q;

double duration=1000;   //simulation time
```

Generating the packets:

```
//generate packets(push into the queue) and return packet arrival rate at the switch from the source
double generate_packets(int src_id, double packet_gen_rate, double t_delay)
{
    double t_one = 1 / packet_gen_rate; //time to generate one packet
    double start= t_one;
    while(start<=duration)
    {
        q.push({start,start,src_id,0}); //push the packet generation event(one packet generated every t_one seconds)
        start+=t_one;
    }
    return ( 1 / (t_delay+t_one) ); //return the packet arrival rate from the source
}
```

Initializing bandwidths of the links, packet generation rate at the sources , and computing arrival rate of the packets, dispatch rate from the switch, transmission delays, and utilization factor. (R1 and R2, the two bandwidths are changed for multiple simulations)

```
double R1=1;  //bandwidth from source to switch in packets/s
double R2=1;  //bandwidth from switch to sink in packets/s
double g1=5;    //packet generation rate from source 1 in  packet/s
double g2=10;   //packet generation rate from source 2 in  packet/s
double g3=15;   //packet generation rate from source 3 in  packet/s
double g4=20;   //packet generation rate from source 4 in  packet/s

double transmission_delay_at_src = 1 / R1;      //in seconds (transmission delay for each packet)
double transmission_delay_at_switch = 1 / R2;   //in seconds

double a1=generate_packets(1,g1,transmission_delay_at_src);
double a2=generate_packets(2,g2,transmission_delay_at_src);
double a3=generate_packets(3,g3,transmission_delay_at_src);
double a4=generate_packets(4,g4,transmission_delay_at_src);

double total_delay=0; //sum of delay of each packet (time to reach sink - time of generation)
long long cnt=0; //number of packets received by sink in the duration of simulation

double arrival_rate= a1+a2+a3+a4;    //adding arrival rate due to each source
double dispatch_rate= R2;            //simply the bandwidth of link from switch to sink
double utilization_factor= arrival_rate / dispatch_rate;
double ready_to_dispatch_time=0;     //time at which the packet at the front of the queue is ready to dispatch
```

Processing the events using the priority queue one by one:

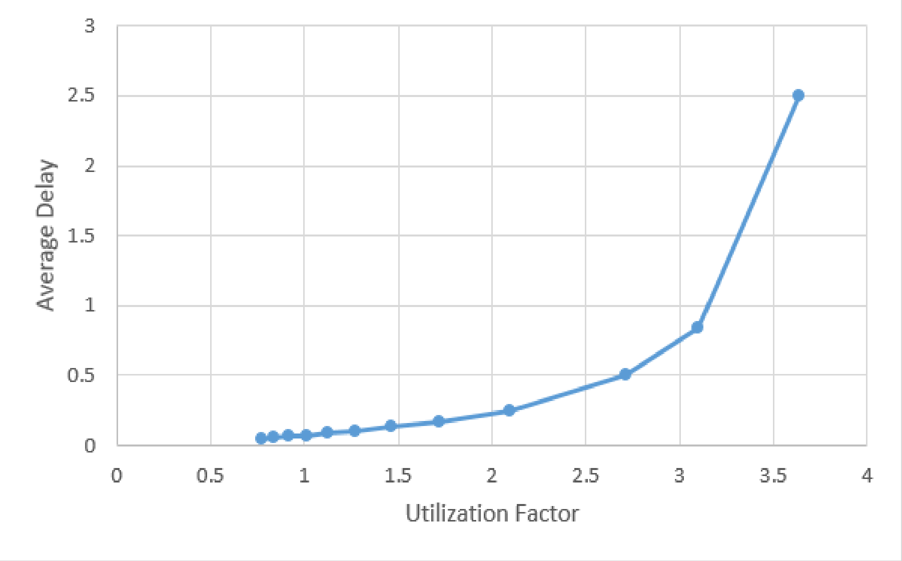
```
while(!q.empty())
{
    vector<double> packet= q.top();
    q.pop();
    double t_occ=packet[0];
    double t_gen=packet[1];
    double src_id=packet[2];
    double status=packet[3];

    if(t_occ > duration)                //simulation time over (discard rest of the events)
        break;

    if(status==0)   //transfer packet from source to switch
    {
        q.push({t_occ+transmission_delay_at_src,t_gen,src_id,1});
    }
    else if(status==1)//from queue to ready to dispatch from queue
    {
        t_occ=max(t_occ,ready_to_dispatch_time);
        q.push({t_occ,t_gen,src_id,2});
    }
    else if(status==2)  //from queue to sink
    {
        q.push({t_occ+transmission_delay_at_switch,t_gen,src_id,3});
        ready_to_dispatch_time=t_occ + transmission_delay_at_switch; //ready to dispatch time for the next packet
    }
    else if(status==3)  //reached the sink
    {
        total_delay+= t_occ - t_gen;    //add total delay for the current packet
        cnt++;                          //increment packets received at sink
    }
}
```

Printing the desired values:

```
    double avg_delay = total_delay / cnt;
    cout<<"Utilization factor = ";
    cout<<utilization_factor<<endl;
    cout<< "Average delay = ";
    cout<<fixed<<setprecision(7)<<avg_delay<<endl;
    return 0;
}
```

Running the simulation multiple times to get the following graph:

Bandwidth (R1=R2)	Utlisation Factor	Average Delay
1	3.63231	2.49966
3	3.09713	0.839886
5	2.71667	0.499648
10	2.1	0.248682
15	1.72143	0.173098
20	1.4619	0.135939
25	1.27183	0.0979997
30	1.12619	0.0878989
35	1.01086	0.0723378
40	0.917172	0.0626288
45	0.83951	0.0566662
50	0.774059	0.0499996



Case 2: When queue size is finite:

Few more variables added

```
long long maxqsize=5;              //maximum size of the queue at output port of the switch
long long curqsize=0;              //current queue size
long long packets_in=0;            //no. of packets entering the queue
long long packets_drop=0;          //no. of packets dropped
```

Processing the events:

```
while(!q.empty())
{
    vector<double> packet= q.top();
    q.pop();
    double t_occ=packet[0];
    double t_gen=packet[1];
    double src_id=packet[2];
    double status=packet[3];

    if(t_occ > duration)                //simulation time over (discard rest of the events)
        break;

    if(status==0)   //transfer packet from source to switch
    {
        q.push({t_occ+transmission_delay_at_src,t_gen,src_id,1});
    }

else if(status==1)//from queue to ready to dispatch from queue
{
    if(curqsize < maxqsize)          //queue not full so add the packet to the queue
    {
        curqsize++;
        packets_in++;
        t_occ=max(t_occ,ready_to_dispatch_time);    //wait for the current packet until previous packet is completely dispatched
        q.push({t_occ,t_gen,src_id,2});
    }
    else                             //queue full so drop the packet
    {
        packets_in++;
        packets_drop++;
    }
}
else if(status==2)  //from queue to sink
{
    q.push({t_occ+transmission_delay_at_switch,t_gen,src_id,3});
    ready_to_dispatch_time=t_occ + transmission_delay_at_switch; //ready to dispatch time for the next packet
    curqsize--;     //packet removed from the queue
}

        else if(status==3)  //reached the sink
        {
            total_delay+= t_occ - t_gen;     //add total delay for the current packet
            cnt++;                           //increment packets received at sink
        }
  }
```

Printing the desired values:

```cpp
double packet_loss_rate= (double)packets_drop / packets_in;
cout<<"Utilization factor = ";
cout<<utilization_factor<<endl;
cout<< "Packet loss rate = ";
cout<<fixed<<setprecision(7)<<packet_loss_rate<<endl;
return 0;
```

Running the simulation multiple times to get the following graph:

Bandwidth (R1=R2)	Utlisation Factor	Packet loss rate
0.5	3.80482	0.949977
1	3.63231	0.899974
2	3.33906	0.799972
3	3.09713	0.699966
4	2.89265	0.599968
5	2.71667	0.49995
6	2.56306	0.39996
7	2.42746	0.299954
8	2.30663	0.199952
9	2.19811	0.1400024
15	1.72143	0
20	1.4619	0