Computer Network Lab
CS359

Tanishq Malu Lab:1b 1901CS63

Architecture:

Classes used:

SourceClass

Attributes:
SourceID = unique id of a source
bs = bandwidth speed between source and switch
generation_rate = Rate of generation of packets by the source

Packet Class

Attributes:
Pkt_Id = unique id of a packet
queue_reach_time = Time when packet reaches queue in switch
queue_leave_time = Time when packet leave the queue
ReachSinkTime = Time when packet reaches sink
sourceId = Id of source from which the packet was generated
GenerationTime = Time at which this packet was generated

Switch Class

Attributes:
QueueSize = size of the queue in switch
switch_bs = bandwidth speed between switch and sink

Stage Class

Attributes:
StageId = Id of the stage at which a packet is
packetid = Id of the packet
currenttime = Time at which this packet reached this stage

The basic purpose of the stage class is to break down the whole process of

packet transfer from source to sink via switch into smaller process.

Breaking down the whole process of packet transfer from source to sink via a switch to simulate the process.

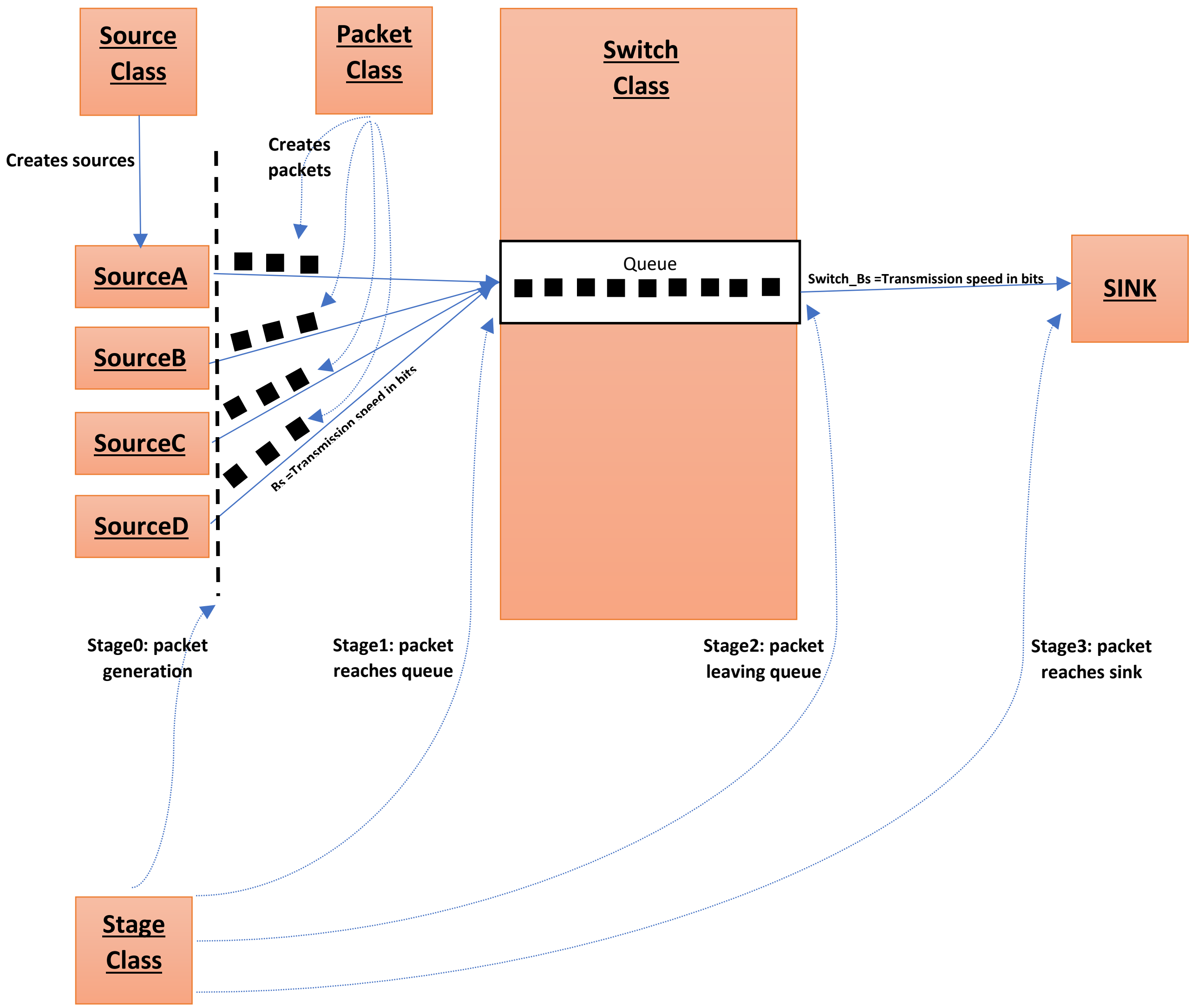
Let us break down the process into 4 stages

Stage id = 0 => The time when packet is generated

Stage id = 1 => The time when packet reaches the queue

Stage id = 2 => The time when packet leaves the queue

Stage id = 3 => The time when packet reaches the sink



In part a: The queue size is infinite and we vary bandwidth speed between switch and sink to see relationship between average delay and utilization factor.

In part b: The queue size is fixed and we maintain a counter of number of packets dropped and arrived to find a relationship between utilization factor and packet drop rate, by varying bandwidth speed between switch and sink.

Walkthrough:

We set the values of different variables for the program either by default or take them as input from user.

After formation of all the source object using source class, the program initializes the generation of first packet for each source using the packet class. These packet, as is evident, are in the stage 0 and *event object* are created for each packet with current time and entered into a priority queue, which sorts the *event objects* based on their time when they were generated. (Initially, we have kept a minute time difference for formation of first packet of each source)

Now we keep popping the top element of our priority queue and pushing new event object formed along the process to simulate the process as per the stage and packet id of the *event object* mentioned in the respective elements: -

When a Packet leaves a source and heads towards the switch, our program changes the stage id of the *event object* associated with this packet, from stage 0 to stage 1 indicating this transfer of packet from source to switch. Simultaneously, a new packet will be generated from the source after (current time + 1/generation rate) time and an *event object* will be created for this packet with stage id 0 and pushed into the priority queue.

When the packet reaches the queue, if the queue size is infinite, there is no case of packet drop and it will simply transit towards the head of the queue. Here the stage id of the *event object* associated with this packet will be changed from stage 1 to stage 2 and the packet is now ready to move towards the sink. (Pushed back into the priority queue)

On the other hand, if the size of the queue is finite, there will be chances of packet drop too and separate counter will be incremented for this purpose.

The event object containing packets with stage id 2 will now move from queue to sink and its stage id will change from stage 2 to stage 3 and a new event object will be pushed back into the priority queue.

Finally, the packet will reach the sink.

We can keep account of time at which the packet was generated and packet reached the sink. Also, we can maintain a count for total number of packets reached to our sink and can use it to find average delay. Further a count of packet drop and packet arrived can be maintained to find packet drop rate.

This whole process is simulated with varying bandwidth speed between source and sink, keeping other things constant to find the relationship between utilization factor and average delay and similarly for utilization factor and packet drop rate too.

Part a ->

The average delay of the sources with respect to the utilization factor

Python Code:

Library used to plot the graph

```
import matplotlib.pyplot as plt

import queue as Qu


#arrival rate of utilization factor

arrivalRate = 0


# A class for source

# Certain terminology

# bs = bandwidth speed

#
```

Different classes to represent source, switch, packet and events

```
class SourceClass:

  def __init__(self, rate, bandwidthSpeed, source_id):

        self.sourceId = source_id

        self.bs = bandwidthSpeed

        self.Pkt_generation_rate = rate



# class for switch

class SwitchClass:

  def __init__(self, bandwidthSpeed):

        self.QueueSize = 0

        self.switch_bs=bandwidthSpeed



# class for packet
```

```
class PacketClass:

    def __init__(self, id=0, Generation_time=0.0, source_id=0):

            self. queue_reach_time = -1

            self. queue_leave_time = -1

            self.ReachSinkTime = -1

            self.Pkt_Id = id

            self.sourceId = source_id

            self.GenerationTime = Generation_time


# class for different Stages

class Stage:

    def __init__(self, Stageid, packetid, currenttime):

        self.StageId = Stageid

        self.pId = packetid

        self.curTime = currenttime


    def __lt__(self, comp):

        return self.curTime < comp.curTime
```

Function to calculate the average delay for a given bandwidth speed

```
# function to calculate avg delay

def calculateAvgDelay(numberOfSource, bs, switch_bs, pktLength, source_array, TotalSimulationTime):

    avgDelay = 0.0


    # Stores packets

    packet = []


    mainSwitch = SwitchClass(switch_bs)
```

```
# PriorityQueue is priority queue on the basis of Stage's time

PriorityQueue = Qu.PriorityQueue()


# Initialisig packet generation from each source

Initialtime = 0.0
```

Initializing packet formation in each source

```
for i in range(numberOfSource):

    packet.append(PacketClass(i, Initialtime, i))

    PriorityQueue.put(Stage(0, i, Initialtime))

    Initialtime = Initialtime + 0.000005


# packet count

pcount = 0

# Initial number of packet = number of sources
TotalPackets = numberOfSource


 lastLeftTime = 0.0

TotalSimulationTime = 5000


#current timer

curTime = 0


# Number of pakcets that reached the sink

pkt_reach_sink = 0


while (pkt_reach_sink < TotalSimulationTime):

    CurrPacket = PriorityQueue.get()

    pid = CurrPacket.pId

    curTime = CurrPacket.curTime
```

```
currStage = CurrPacket.StageId

# Stage 0 -> (Stage0,Stage1)
```

When a packet will leave the source and reach the switch.

```
if currStage == 0:
    # Rate at which a single packet is generated
    rate = source_array[packet[pid].sourceId].Pkt_generation_rate
    singlePktGentime = 1 / rate

    # Since the current packet has left the source, the source will generate a new packet after
(1/generateion rate)
    # Thus a new packet has been formed having stage 0 at the respective source
      nextTime = curTime + singlePktGentime
    PriorityQueue.put(Stage(0, TotalPackets, nextTime))
    packet.append(PacketClass(TotalPackets, nextTime, packet[pid].sourceId))

    # The stage of current packet will change from 0 to 1 and its timer will also be increased
accordingly.

    PriorityQueue.put(Stage(1, pid, curTime + pktLength / bs))

    packet[pid]. queue_reach_time = curTime + pktLength / bs

# Stage 1 -> Stage2
```

When a packet will move from back of the queue to the front

```
elif currStage == 1:
      # the time required by current packet to reach to front of the queue
    reachingTime = (mainSwitch.QueueSize * pktLength) / switch_bs

    var_time = 0
    if packet[pid]. queue_reach_time - lastLeftTime < (pktLength / switch_bs):
```

```
    var_time = max(0, (pktLength / switch_bs) - (packet[pid]. queue_reach_time - lastLeftTime))

  if lastLeftTime == 0:
    var_time = 0
  nextTime = curTime + reachingTime + var_time
  packet[pid]. queue_leave_time =  nextTime

  # The stage of current packet will change from 1 to 2 and its timer will also be increased
accordingly.
  PriorityQueue.put(Stage(2, pid, nextTime))

  avgDelay += packet[pid]. queue_leave_time - packet[pid].GenerationTime +
(pktLength/switch_bs)

  mainSwitch.QueueSize += 1
  pcount += 1


# Stage2 -> Stage3
elif currStage == 2:
  # The current packet will reach to sink in (pktLength / switch_bs) time
  # The stage of current packet will change from 2 to 3 and its timer will also be increased
accordingly.
   nextTime = curTime + (pktLength / switch_bs)
  PriorityQueue.put(Stage(3, pid, nextTime))
       mainSwitch.QueueSize -= 1
  packet[pid].ReachSinkTime = nextTime
  lastLeftTime = curTime
else:
   pkt_reach_sink += 1
```

When a packet will leave the queue to reach the sink

When a packet reaches to sink.

```
    avgDelay /= pcount
    return avgDelay

def main():
        # arrivalRate = numerator of utilization factor
    global arrivalRate

    print ("To run in default mode: Enter 0")
    print ("To run with values defined by you: Enter 1")
    CheckInput = int(input())
    if CheckInput == 0:
        # Number of sources present
        numberOfSource = 4

        # bs = bandwidth speed between source and switch in bit
        bs = 2e6

        # switch_bs = bandwidth between switch and sink in bit
        switch_bs_high = 8e6
        switch_bs_low = 20e3
            # pktLength = Packet Size in bit
        pktLength = 1e4

        #generationrate = packet Pkt_generation_rate
        generationRate = 15

        #simulation time
        TotalSimulationTime = 5000
    else :
```

Program can be run either in default mode or user defined mode

```
    numberOfSource = int(input("Enter the number of sources required :"))

    bs = float(input("Enter the bandwdth between source and switch(bs) in bit:"))

    switch_bs_low = float(input("Enter the bandwidth(lower bound) between switch and 
sink(switch_bs_low) in bit:"))

    switch_bs_high = float(input("Enter the bandwidth(upper bound) between switch and 
sink(switch_bs_high) in bit:"))

    pktLength = int(input("Enter the packet length in bit(pktlength) :"))

    generationRate = int(input("Enter the packet generation rate for source(assuming it to be same for 
all):"))

    while pktLength * generationRate >= bs:

      print("pktLength * genrationRate should be less than bs. Please enter approproate values.")

      pktLength = int(input("Enter the packet length in bit(pktlength) :"))

      generationRate = int(input("Enter the packet generation rate for source(assuming it to be same 
for all):"))

      bs = float(input("Enter the bandwdth between source and switch(bs) in bit:"))


    TotalSimulationTime =int(input("Enter the total simulation time:"))


  # To plot the graph: x and y holds value of average delay and Utilization factor

  x = []

  y = []


  # source array which contains source object

  source_array = []

  for i in range(numberOfSource):

    source_array.append(SourceClass(generationRate, bs, i))

    arrivalRate += generationRate

  curr_switch_bs = switch_bs_low

  increment_rate_gt1 =  (switch_bs_high - switch_bs_low)/2000

  increment_rate_lt0 =  (switch_bs_high - switch_bs_low)/200
```

```
    # varying curr_switch_bs for plotting
    print("Utilization Factor          :          Average Delay")
    while (curr_switch_bs<=switch_bs_high):
        UtilizationFactor_numerator = (arrivalRate * pktLength)
        UtilizationFactor_denominator = curr_switch_bs
```

Varying the speed for finding relationship between average delay and utilization factor

```
        UtilizationFactor = UtilizationFactor_numerator / UtilizationFactor_denominator
        avgDelay = calculateAvgDelay(numberOfSource, bs, curr_switch_bs, pktLength, source_array,
TotalSimulationTime)
```

Calculating average delay for a given transmission speed

```
        x.append(UtilizationFactor)
        y.append(avgDelay)
        print(UtilizationFactor, "                    ", avgDelay)

        if(UtilizationFactor>1):
            curr_switch_bs += increment_rate_gt1
        else:
            curr_switch_bs += increment_rate_lt0
```

plotting the graph

```
    # Plotting the grpah
    plt.plot(x, y)
    plt.xlabel("Utilization Factor")
    plt.ylabel("Average Delay")
    plt.title("Average delay vs UtilizationFactor")
    plt.show()


if __name__ == "__main__":
    main()
```

Output:

To run in default mode: Enter 0

To run with values defined by you: Enter 1

0

Utilization Factor :	Average Delay
30.0	36250.529992333766
25.010421008753646	25022.306826597545
21.44388849177984	18266.831392309345
18.76759461995621	13894.453715260828
16.685205784204673	10905.05527058664
15.018773466833542	8773.111917823202
13.654984069185252	7200.416219931236
12.518255789693303	6008.064864535517
11.556240369799692	5083.316240573897
10.731532820604542	4351.7448482204545
10.01669449081803	3763.3895222718065
9.391141023634372	3283.5700678899707
8.839127872716558	2887.2627807537688
8.348406845693614	2556.373690612329
7.909306617453203	2277.1813876137967
7.514088916718848	2039.6621830656431
7.156488549618321	1835.962848734103
6.831378799954457	1660.0944190437092
6.53452406883032	1507.14368414308
6.2623943220958145	1373.2246042315624

The following output provides as different values of average delay with respect to utilization factor as we vary our bandwidth speed from switch to sink.

Plotting Utilization factor on X

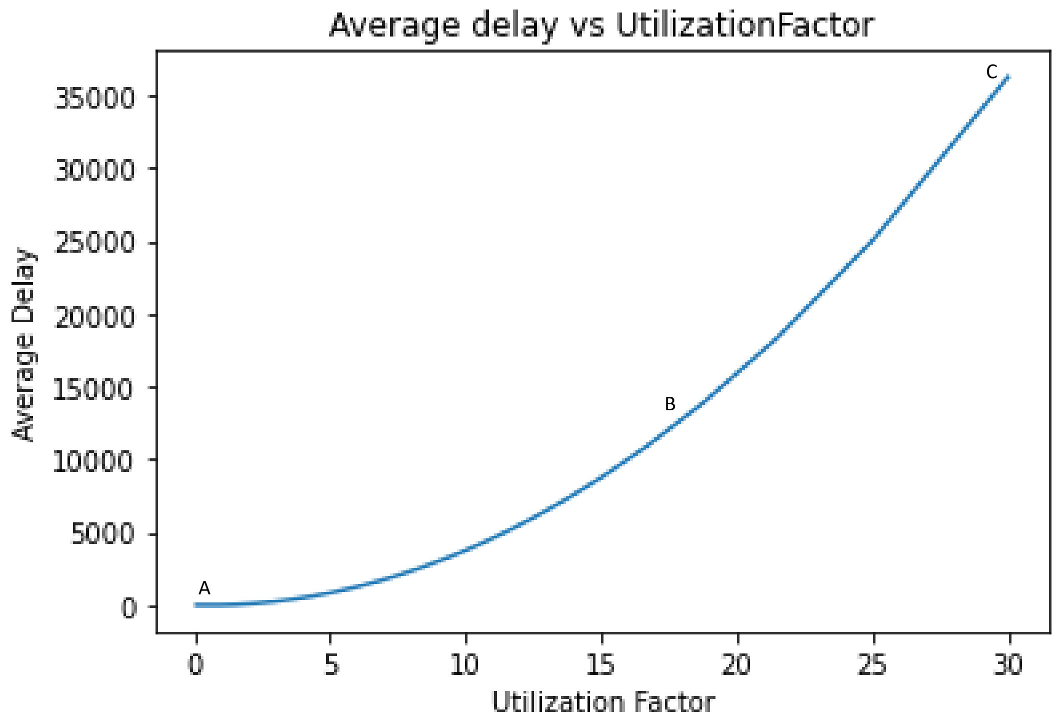
Plotting Average Delay on Y

…. And so on

Note: The output has been clipped due to its large size. Open the link to see the full working code along with the output on google collab:

https://colab.research.google.com/drive/1vh5sWcdA3Oz36XeNKsdjc_y3DbG3q-Zd?usp=sharing

Graph:



Observation:

In our Graph ABC, we observe that while keeping the arrival rate of utilization factor constant and changing the bandwidth speed from source to sink, as the utilization factor increase, the average delay of the sources also increases which is quite evident from the fact that as bandwidth speed increases utilization factor decreases and delay also decreases and vice versa.

--End Of Part 1--

Part b ->

If the queue size is fixed, then the packet loss rate at the switch with respect to the utilization factor

Python Code:

Library used to plot the graph

```
import matplotlib.pyplot as plt

import queue as Qu


#arrival rate of utilization factor

arrivalRate = 0


# A class for source

# Certain terminology

# bs = bandwidth speed

#

class SourceClass:

    def __init__(self, rate, bandwidthSpeed, source_id):

            self.sourceId = source_id

            self.bs = bandwidthSpeed

            self.Pkt_generation_rate = rate




# class for switch

class SwitchClass:

    def __init__(self, bandwidthSpeed):

            self.QueueSize = 0

            self.switch_bs=bandwidthSpeed
```

Different classes to represent source, switch, packet and events

```
# class for packet
class PacketClass:
    def __init__(self, id=0, Generation_time=0.0, source_id=0):
        self. queue_reach_time = -1
        self. queue_leave_time = -1
        self.ReachSinkTime = -1
        self.Pkt_Id = id
        self.sourceId = source_id
        self.GenerationTime = Generation_time
```

```
# The basic purpose of the stage class(below) is to break down the whole process of
# packet transfer from source to sink via switch into smaller process.

# Let us break down the process into 4 stages

# Stage id = 0 => The time when packet is generated
# Stage id = 1 => The time when packet reaches the queue
# Stage id = 2 => The time when packet leaves the queue
# Stage id = 3 => The time when packet reaches the sink
```

Breaking down the whole process of packet transfer from source to sink via a switch to simulate the process.

```
# class for different Stages
class Stage:
    def __init__(self, Stageid, packetid, currenttime):
        self.StageId = Stageid
        self.pId = packetid
        self.curTime = currenttime
```

```
    def __lt__(self, comp):
        return self.curTime < comp.curTime
```

Function to calculate the packet loss rate for a given bandwidth

```
# function to calculate Packet loss rate
def PktLossRate(numberOfSource ,bs, switch_bs, pktLength, source, TotalSimulationTime,
fixedQueueSize):
    avgDelay = 0.0

    # Stores packets
    packet = []

    mainSwitch = SwitchClass(switch_bs)

    # PriorityQueue is priority queue on the basis of Stage's time
    PriorityQueue = Qu.PriorityQueue()

    # Initialisig packet generation from each source
    Initialtime = 0.0
```

Initializing packet formation in each source

```
    for i in range(numberOfSource):
        packet.append(PacketClass(i, Initialtime, i))
        PriorityQueue.put(Stage(0, i, Initialtime))
        Initialtime = Initialtime + 0.000005

            # packet count
    pcount = 0

    # Initial number of packet = number of sources
```

```
TotalPackets = numberOfSource

lastLeftTime = 0.0
TotalSimulationTime = 5000

#current timer
curTime = 0

# Number of pakcets that reached the sink
pkt_reach_sink = 0
packetarrived = 0
pktDrop = 0
while (pkt_reach_sink < TotalSimulationTime):

    CurrPacket = PriorityQueue.get()
    pid = CurrPacket.pId
    curTime = CurrPacket.curTime

    # Stage 0 -> (Stage0,Stage1)
    if currStage == 0:
        # Rate at which a single packet is generated
        rate = source[packet[pid].sourceId].Pkt_generation_rate
        singlePktGentime = 1 / rate

        # Since the current packet has left the source, the source will generate a new packet after
(1/generateion rate)
        # Thus a new packet has been formed having stage 0 at the respective source
        PriorityQueue.put(Stage(0, TotalPackets, curTime + singlePktGentime))
        packet.append(PacketClass(TotalPackets, curTime + singlePktGentime, packet[pid].sourceId))
```

When a packet will leave the source and reach the switch.

```
        TotalPackets += 1

        # The stage of current packet will change from 0 to 1 and its timer will also be increased
accordingly.
        PriorityQueue.put(Stage(1, pid, curTime + pktLength / bs))

        packet[pid]. queue_reach_time = curTime + pktLength / bs
            # Stage 1 -> Stage2
      elif currStage == 1:
        packetarrived += 1
        reachingTime = (mainSwitch.QueueSize * pktLength) / switch_bs
        if(mainSwitch.QueueSize <= fixedQueueSize):
            var_time = 0
            if packet[pid]. queue_reach_time - lastLeftTime < (pktLength / switch_bs):
              var_time = max(0, (pktLength / switch_bs) - (packet[pid]. queue_reach_time -
lastLeftTime))

            if lastLeftTime == 0:
              var_time = 0

            packet[pid]. queue_leave_time = curTime + reachingTime + var_time

            PriorityQueue.put(Stage(2, pid, curTime + reachingTime + var_time))
            #avgDelay += packet[pid]. queue_leave_time - packet[pid].GenerationTime
            mainSwitch.QueueSize += 1
            pcount += 1
        else:
          #counting packet loss/drop
```

When a packet will move from back of the queue to the front

If the queue is full the packet gets dropped

```
            pktDrop += 1

        # Stage2 -> Stage3
        elif currStage == 2:
```

When a packet will leave the queue to reach the sink

```
            # The current packet will reach to sink in (pktLength / switch_bs) time
            # The stage of current packet will change from 2 to 3 and its timer will also be increased accordingly.
            PriorityQueue.put(Stage(3, pid, curTime + (pktLength / switch_bs)))

            mainSwitch.QueueSize -= 1
            packet[pid].ReachSinkTime = curTime + (pktLength / switch_bs)
            lastLeftTime = curTime
        else:
```

When a packet reaches to sink.

```
              pkt_reach_sink += 1

    #avgDelay /= pcount
    #return avgDelay
    return pktDrop/packetarrived


def main():
    print ("To run in default mode: Enter 0")
    print ("To run with values defined by you: Enter 1")
    CheckInput = int(input())
```

Program can be run either in default mode or user defined mode

```
    if CheckInput == 0:
        # Number of sources present
        numberOfSource = 4

        # bs = bandwidth speed between source and switch in bit
        bs = 1e2
```

```
        # switch_bs = bandwidth between switch and sink in bit
        switch_bs_low = 10
        switch_bs_high = 500

        # pktLength = size of each packet in bit
        pktLength = 2

        # Generation Rate of packets
        GenerationRate = 15

        # simulation time
        TotalSimulationTime = 200

        #max queue size in switch
        fixedQueueSize = 50
    else :
        numberOfSource = int(input("Enter the number of sources required :"))
        bs = float(input("Enter the bandwdth between source and switch(bs) in bit:"))
        switch_bs_low = float(input("Enter the bandwidth(lower bound) between switch and sink(switch_bs_low) in bit:"))
        switch_bs_high = float(input("Enter the bandwidth(upper bound) between switch and sink(switch_bs_high) in bit:"))
        pktLength = int(input("Enter the packet length in bit(pktlength) :"))
        GenerationRate = int(input("Enter the packet generation rate for source(assuming it to be same for all):"))
        while pktLength * GenerationRate >= bs:
            print("pktLength * genrationRate should be less than bs. Please enter approproate values.")
            pktLength = int(input("Enter the packet length in bit(pktlength) :"))
```

```
        GenerationRate = int(input("Enter the packet generation rate for source(assuming it to be same for all):"))
        bs = float(input("Enter the bandwdth between source and switch(bs) in bit:"))

    TotalSimulationTime =int(input("Enter the total simulation time:"))
    fixedQueueSize = int(input("Enter Queue size in switch:"))

  # arrivalRate in utilization factor
  global arrivalRate

        # source holds all the source objects
  source = []

  for i in range(numberOfSource):
    source.append(SourceClass(GenerationRate, bs, i))
    arrivalRate += GenerationRate

  # x holds value of Utilization factor
  x = []

  # y holds value of packet loss rate
  y = []

  # varying curr_switch_bs i.e switch_bs for plotting
  curr_switch_bs = switch_bs_low
  increment_amt = (switch_bs_high - switch_bs_low)/800
  print("UtilizationFactor          :          packet_Drop_Rate")
  while(curr_switch_bs <= switch_bs_high):
    UtilizationFactor_numerator = (arrivalRate * pktLength)
```

Varying the speed for finding relationship between average delay and utilization factor

```
        UtilizationFactor_denominator = curr_switch_bs

        UtilizationFactor = UtilizationFactor_numerator / UtilizationFactor_denominator
        x.append(UtilizationFactor)

        packet_loss_Rate = PktLossRate(numberOfSource, bs, curr_switch_bs, pktLength, source, TotalSimulationTime, fixedQueueSize)
        y.append(packet_loss_Rate)

        print(UtilizationFactor,"                    ",packet_loss_Rate)

        # Updating the Bandwidth speed for graph plotting purpose
        curr_switch_bs += increment_amt


    # Plotting the grpah
    plt.plot(x,y)
    plt.xlabel("Utilization Factor")
    plt.ylabel("Packet loss rate")
    plt.title("Packet loss rate vs Utilization Factor")
    plt.show()

if __name__=="__main__":
    main()
```

Calculating Packet loss rate for a given transmission speed

Output:

To run in default mode: Enter 0

To run with values defined by you: Enter 1

0

UtilizationFactor :	packet_Drop_Rate
12.0	0.9158166666666666
11.307420494699645	0.9106650159179343
10.690423162583517	0.9055110745285843
10.137275607180568	0.9003511679292929
9.638554216867467	0.8951987716822973
9.186602870813395	0.8900426680599094
8.775137111517365	0.884880116692497
8.398950131233592	0.8797266406324412
8.053691275167782	0.874577870480731
7.735697018533438	0.8694157187176835
7.441860465116275	0.8642642158443513
7.1695294996265835	0.8590995313546084
6.916426512968297	0.8539498033772843
6.680584551148222	0.8487905640043109
6.460296096904438	0.8436416542842992

The following output provides us different values of packet_Drop_rate with respect to utilization factor as we vary our bandwidth speed from switch to sink.

Plotting Utilization factor on X

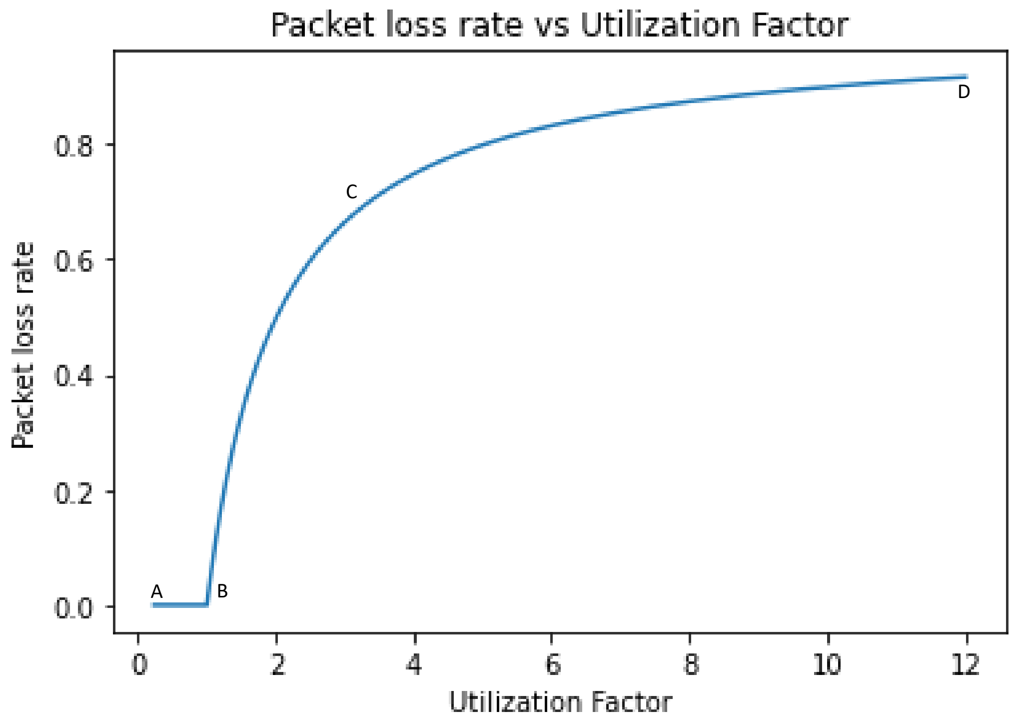
Plotting packet_Drop_rate on Y

…. And so on

Note: The output has been clipped due to its large size. Open the link to see the full working code along with the output on google collab:

https://colab.research.google.com/drive/1vh5sWcdA3Oz36XeNKsdjc_y3DbG3q-Zd?usp=sharing

Graph:



Observation:

In our Graph ABCD, we observe that while keeping the arrival rate of utilization factor constant and fixing the size of our queue; thus on changing the bandwidth speed from source to sink, as the utilization factor increase, the Packet loss rate of the sources also increases which is quite evident from the fact that as bandwidth speed increases, utilization factor decreases and average delay decreases and the queue gets freed faster thus packet loss rate also decreases and vice versa.

Another important observation can be seen by analyzing the line AB in our graph. It suggests as that below a certain utilization factor the packet loss is always zero, thus we can use it to find the lowest bandwidth speed to ensure zero packet drop rate.

---End Of Assignment--