Introduction:

Max-weight/Back pressure routing:

Max-weight or Backpressure routing is routing algorithm which is explicitly used in multi-hop network by with the help of congestion gradients in order to dynamically route the traffic over a multi-hop network. In a wireless communication networks, this routing algorithms are most widely used in sensor networks, mobile ad hoc networks and heterogeneous networks with wireless and wireline components.

Backpressure routing technology is also used in many fields like processing networks and product assembly and many more. In multi hop network infrastructure the data packets from multiple data streams arrive and it is expected to deliver to an appropriate destination. The backpressure algorithm operates in slotted time. Every time slot it seeks to route data in directions that maximize the differential backlog between neighboring nodes. However, the backpressure algorithm can be applied to multi-commodity networks like the problem statement as given in the figure (where different packets may have different destinations user 1 and user 2), and to networks where transmission rates can be selected from a set of options.

The most important features of the backpressure algorithm are:

(a) It leads to maximum network throughput.

(b) It is probably more stable and robust infrastructure to time-varying network conditions.

(c) it can be implemented without having any knowledge about packet arrival rates or channel state probabilities.

However, the algorithm may introduce large delays, and may be difficult to implement exactly in networks with interference. Modifications of backpressure that reduce delay and simplify implementation are described below under Improving delay and Distributed backpressure.

In practical scenario, ad hoc wireless networks have typically implemented alternative routing methods based on shortest path computations or network flooding, such as Ad Hoc on-Demand Distance Vector Routing (AODV), geographic routing, and extremely opportunistic routing (Ex-OR). However, the mathematical optimality properties of backpressure have motivated recent experimental demonstrations of its use on wireless testbeds at the University of Southern California and at North Carolina State University.

Working Principle:

Backpressure routing is designed to make decisions that (roughly) minimize the sum of squares of queue backlogs in the network from one timeslot to the next. The precise mathematical development of this technique is described in later sections. This section describes the general network model and the operation of backpressure routing with respect to this model.

Consider a multi-hop network with **N** number of nodes The network operates in sloted time t={0,1,2,3,4,5,6,7,…….}. On each slot, new data can arrive to the network, and routing and transmission scheduling decisions are made to deliver all data to its proper destination. Let data that is

destined for node C={1,2,3, 4,….N} be labeled as *commodity c data*. Data in each node is stored according to its commodity. For n={1,….N} and c={1…N}, let Qn(t) be the current amount of commodity *c* data in node *n*, also called the *queue backlog*. A closeup of the queue backlogs inside a node.The units of Qn(t) depend on the context of the problem. For example, backlog can take integer units of *packets*, which is useful in cases when data is segmented into fixed length packets. Alternatively, it can take real valued units of *bits*. It is assumed that for all Qn(t) =0 and c={1…N} all timeslots *t*, because no node stores data destined for itself. Every timeslot, nodes can transmit data to others. Data that is transmitted from one node to another node is removed from the queue of the first node and added to the queue of the second. Data that is transmitted to its destination is removed from the network. Data can also arrive exogenously to the network, and An(t) -Arrival is defined as the amount of new data that arrives to node *n* on slot *t* that must eventually be delivered to node *c*.

From the graph we got to that the empirical average is directly related to the queue length and from the three scenarios it is evident that if all the nodes have the same arrival rate then the empirical average tends to increase exponentially as shown in the plot and it keeps on increasing until the maximum time slot.(Scenario 1).

In case of Scenario 2, we got to know that as the arrival rate changes for the nodes, the empirical average value tries to increase in a discrete manner (in this problem max value is 3 to 3.5) and when it reaches the maximum max value it again drops and reaches the saturation level and thereafter the value of the empirical value does not vary much. It will be saturated as the time slots keeps increasing.

In case of Scenario 3, the empirical average value keeps increasing to the maximum value significantly for the Netflix traffic and FBI traffic for both user 1 and user 2 respectively. And for the both Netflix traffic and FBI traffic at cell tower will always be zero.

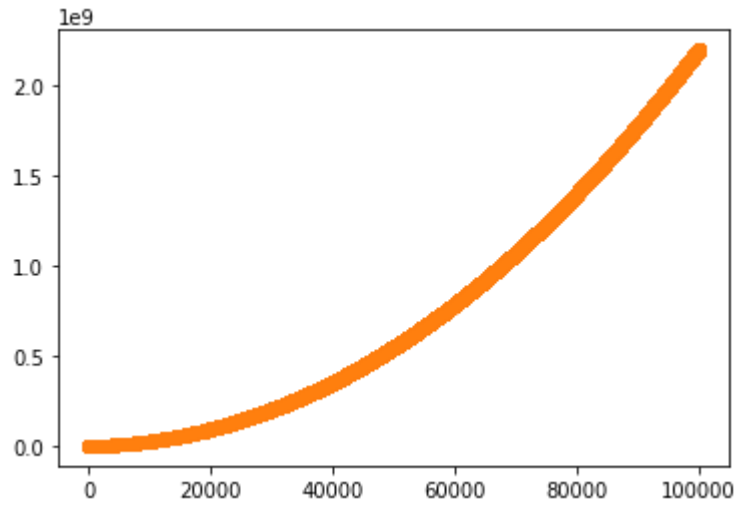
FBI traffic at cell tower increases in a discrete manner and it will be saturated to some value after reaching the maximum value as shown in the graph.

```
import matplotlib.pyplot as plt
import numpy as np


#The following convention is used: node 0 = user 1, node 1 = user 2, node 2 = cell tower,
#node 3 = Netflix server, node 4 = FBI server.

t_nodes = 5 #number of transmitting nodes
r_nodes = 5 #number of receiving nodes
time_slots =100000 #length of simulation



#Queue Lengths
q = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)
empirical_average_q = np.zeros((t_nodes, r_nodes, time_slots))


#Average Arrival Rate
netflix_arrivals_node0 = 0.24
FBI_arrivals_node0 = 0.24
netflix_arrivals_node1 = 0.24
FBI_arrivals_node1 = 0.24


#Feasible Transmission Schedules
M = np.zeros((t_nodes,r_nodes,time_slots), dtype=int)

M[:,:,0] = np.array([[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,1] = np.array([[0,0,0,0,0],[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,2] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,2,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,3] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,2],[0,0,0,0,0],[0,0,0,0,0]])



maxweightM = np.zeros((4,time_slots), dtype=int)

s = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)
```

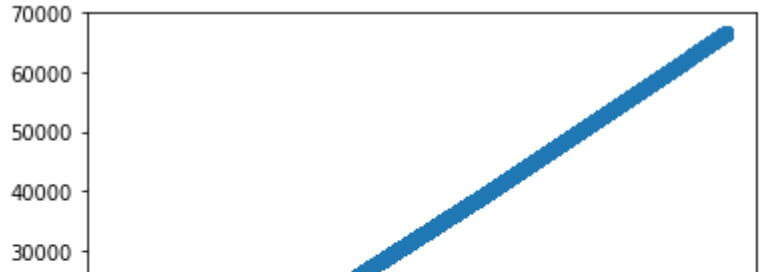
```
c02 = np.zeros(time_slots)
c12 = np.zeros(time_slots)
c23 = np.zeros(time_slots)
c24 = np.zeros(time_slots)



for k in range(time_slots):

    #Find the weights
    if k%2 == 0: #even timeslots
        c02[k] = max(q[0, 3, k] - q[2, 3, k], q[0, 4, k] - q[2, 4, k], 0) #find max queue dif
```

```
        c12[k] = max(q[1, 3, k] - q[2, 3, k], q[1, 4, k] - q[2, 4, k], 0) #find max queue dif

        for l in range(2):
            maxweightM[l, k] = c02[k] * M[0, 2, l] + c12[k] * M[1, 2, l]
    else: #odd time slots
        c23[k] = q[2, 3, k] - q[3, 3, k] #find weight on link 23
        c24[k] = q[2, 4, k] - q[4, 4, k] #find wieght on link 24

        for l in range(2,4):
            maxweightM[l, k] = c23[k] * M[2, 3, l] + c24[k] * M[2, 4, l]

    #Use the maxweight algorithm to find feasible transmission schedules
    s[:, :, k] = M[:, :, np.argmax(maxweightM[:, k])] #find max-weight feasible transmission

    #update the queues
    if k < time_slots-1:

        a03 = np.random.binomial(1, netflix_arrivals_node0) #Netflix traffic at node 0
        a04 = np.random.binomial(1, FBI_arrivals_node0) #FBI traffic at node 0
        a13 = np.random.binomial(1, netflix_arrivals_node1) #Netflix traffic at node 1
        a14 = np.random.binomial(1, FBI_arrivals_node1) #FBI traffic at node 1

        q[0, 3, k + 1] += a03 #Netflix traffic arrving at node 0
        q[0, 4, k + 1] += a04 #FBI traffic arrving at node 0
        q[1, 3, k + 1] += a13 #Netflix traffic arriving at node 1
        q[1, 4, k + 1] += a14 #FBI traffic arrving at node 1

        if k%2==0: #even time slots

            if q[0, 3, k] - q[2, 3, k] > q[0, 4, k] - q[2, 4, k]: #Netflix traffic has max qu
                q[0, 3, k + 1] += max(q[0, 3, k] - s[0, 2, k], 0) #Netflix traffic leaving no
                q[0, 4, k + 1] += q[0, 4, k] #FBI traffic stays at node 0
                q[2, 3, k + 1] += min(s[0, 2, k], q[0, 3, k]) #Netflix traffic arriving at ce
            else:
                q[0, 3, k + 1] += q[0, 3, k] #Netflix traffic stays at node 0
                q[0, 4, k + 1] += max(q[0, 4, k] - s[0, 2, k], 0) #FBI traffic leaving node 0
                q[2, 4, k + 1] += min(s[0, 2, k], q[0, 4, k]) #FBI traffic arriving at cell t

            if q[1, 3, k]-q[2, 3, k] > q[1, 4, k]-q[2, 4, k]: #Netflix traffic has max queue
                q[1, 3, k + 1] += max(q[1, 3, k] - s[1, 2, k], 0) #Netflix traffic leaving no
                q[1, 4, k + 1] += q[1, 4, k] #FBI traffic stays at node 1
                q[2, 3, k + 1] += min(s[1, 2, k], q[1, 3, k]) #Netflix traffic arriving at ce
            else:
                q[1, 4, k + 1] = max(q[1, 4, k] - s[1, 2, k], 0) #FBI traffic leaving node 1
                q[1, 3, k + 1] += q[1, 3, k] #Netflix traffic stays at node 1
                q[2, 4, k + 1] += min(s[1, 2, k], q[1, 4, k]) #FBI traffic arriving at cell t
        else: #odd timeslots

            q[0, 3, k + 1] += q[0, 3, k]
            q[0, 4, k + 1] += q[0, 4, k]
            q[1, 3, k + 1] += q[1, 3, k]
            q[1, 4, k + 1] += q[1, 4, k]
```

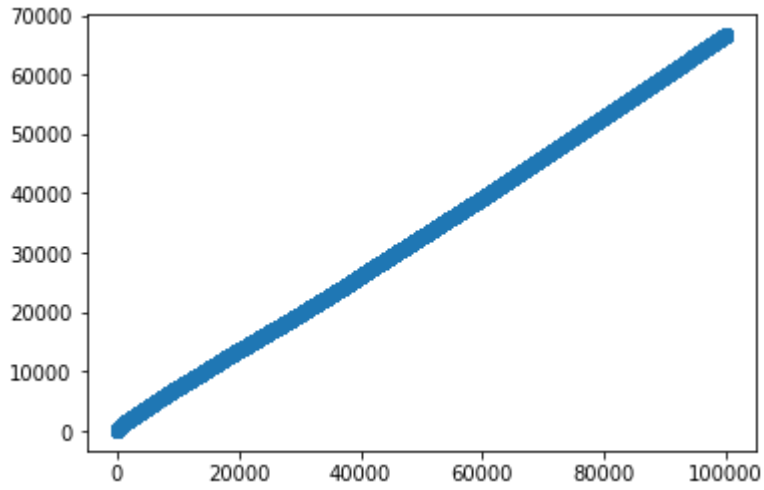
```
            q[2, 3, k + 1] = max(q[2, 3, k] - s[2, 3, k], 0)  # Netflix traffic leaving cell
            q[2, 4, k + 1] = max(q[2, 4, k] - s[2, 4, k], 0)  # FBI traffic leaving cell towe


for i in range(time_slots):
    if i < time_slots - 1:
        empirical_average_q[:,:,i + 1] = ((i + 1) * empirical_average_q[:,:,i] + q[:,:,i + 1]

t = np.arange(0,time_slots,dtype=int)

plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```



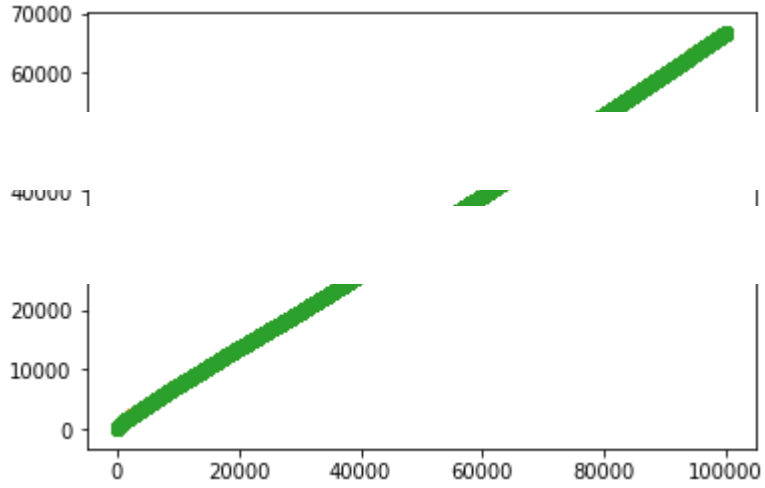

```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```




```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```



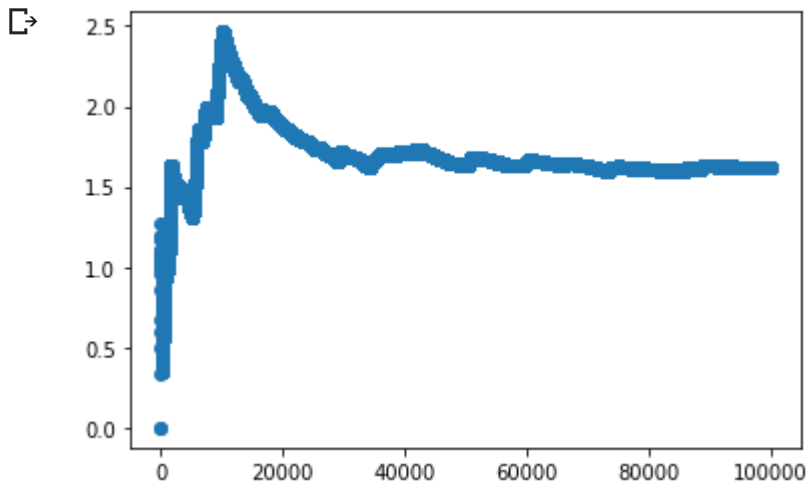

```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```


70000
60000
40000
20000
10000
0
0
20000
40000
60000
80000
100000

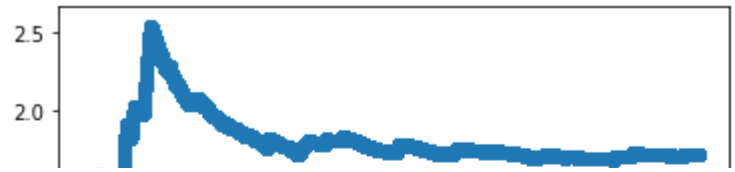
```
import matplotlib.pyplot as plt
import numpy as np


#The following convention is used: node 0 = user 1, node 1 = user 2, node 2 = cell tower,
#node 3 = Netflix server, node 4 = FBI server.

t_nodes = 5 #number of transmitting nodes
r_nodes = 5 #number of receiving nodes
time_slots = 100000 #length of simulation



#Queue Lengths
q = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)
empirical_average_q = np.zeros((t_nodes, r_nodes, time_slots))


#Average Arrival Rate
netflix_arrivals_node0 = 0.10
FBI_arrivals_node0 = 0.38
netflix_arrivals_node1 = 0.38
FBI_arrivals_node1 = 0.10


#Feasible Transmission Schedules
M = np.zeros((t_nodes,r_nodes,time_slots), dtype=int)

M[:,:,0] = np.array([[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,1] = np.array([[0,0,0,0,0],[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,2] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,2,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,3] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,2],[0,0,0,0,0],[0,0,0,0,0]])



maxweightM = np.zeros((4,time_slots), dtype=int)

s = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)


c02 = np.zeros(time_slots)
c12 = np.zeros(time_slots)
c23 = np.zeros(time_slots)
c24 = np.zeros(time_slots)



for k in range(time_slots):

    #Find the weights
    if k%2 == 0: #even timeslots
        c02[k] = max(q[0, 3, k] - q[2, 3, k], q[0, 4, k] - q[2, 4, k], 0) #find max queue dif
        c12[k] = max(q[1, 3, k] - q[2, 3, k], q[1, 4, k] - q[2, 4, k], 0) #find max queue dif
```

```
        for l in range(2):
            maxweightM[l, k] = c02[k] * M[0, 2, l] + c12[k] * M[1, 2, l]
    else: #odd time slots
        c23[k] = q[2, 3, k] - q[3, 3, k] #find weight on link 23
        c24[k] = q[2, 4, k] - q[4, 4, k] #find wieght on link 24

        for l in range(2,4):
            maxweightM[l, k] = c23[k] * M[2, 3, l] + c24[k] * M[2, 4, l]

    #Use the maxweight algorithm to find feasible transmission schedules
    s[:, :, k] = M[:, :, np.argmax(maxweightM[:, k])] #find max-weight feasible transmission

    #update the queues
    if k < time_slots-1:

        a03 = np.random.binomial(1, netflix_arrivals_node0) #Netflix traffic at node 0
        a04 = np.random.binomial(1, FBI_arrivals_node0) #FBI traffic at node 0
        a13 = np.random.binomial(1, netflix_arrivals_node1) #Netflix traffic at node 1
        a14 = np.random.binomial(1, FBI_arrivals_node1) #FBI traffic at node 1

        q[0, 3, k + 1] += a03 #Netflix traffic arrving at node 0
        q[0, 4, k + 1] += a04 #FBI traffic arrving at node 0
        q[1, 3, k + 1] += a13 #Netflix traffic arriving at node 1
        q[1, 4, k + 1] += a14 #FBI traffic arrving at node 1

        if k%2==0: #even time slots

            if q[0, 3, k] - q[2, 3, k] > q[0, 4, k] - q[2, 4, k]: #Netflix traffic has max qu
                q[0, 3, k + 1] += max(q[0, 3, k] - s[0, 2, k], 0) #Netflix traffic leaving no
                q[0, 4, k + 1] += q[0, 4, k] #FBI traffic stays at node 0
                q[2, 3, k + 1] += min(s[0, 2, k], q[0, 3, k]) #Netflix traffic arriving at ce
            else:
                q[0, 3, k + 1] += q[0, 3, k] #Netflix traffic stays at node 0
                q[0, 4, k + 1] += max(q[0, 4, k] - s[0, 2, k], 0) #FBI traffic leaving node 0
                q[2, 4, k + 1] += min(s[0, 2, k], q[0, 4, k]) #FBI traffic arriving at cell t

            if q[1, 3, k]-q[2, 3, k] > q[1, 4, k]-q[2, 4, k]: #Netflix traffic has max queue
                q[1, 3, k + 1] += max(q[1, 3, k] - s[1, 2, k], 0) #Netflix traffic leaving no
                q[1, 4, k + 1] += q[1, 4, k] #FBI traffic stays at node 1
                q[2, 3, k + 1] += min(s[1, 2, k], q[1, 3, k]) #Netflix traffic arriving at ce
            else:
                q[1, 4, k + 1] = max(q[1, 4, k] - s[1, 2, k], 0) #FBI traffic leaving node 1
                q[1, 3, k + 1] += q[1, 3, k] #Netflix traffic stays at node 1
                q[2, 4, k + 1] += min(s[1, 2, k], q[1, 4, k]) #FBI traffic arriving at cell t
        else: #odd timeslots

            q[0, 3, k + 1] += q[0, 3, k]
            q[0, 4, k + 1] += q[0, 4, k]
            q[1, 3, k + 1] += q[1, 3, k]
            q[1, 4, k + 1] += q[1, 4, k]
```

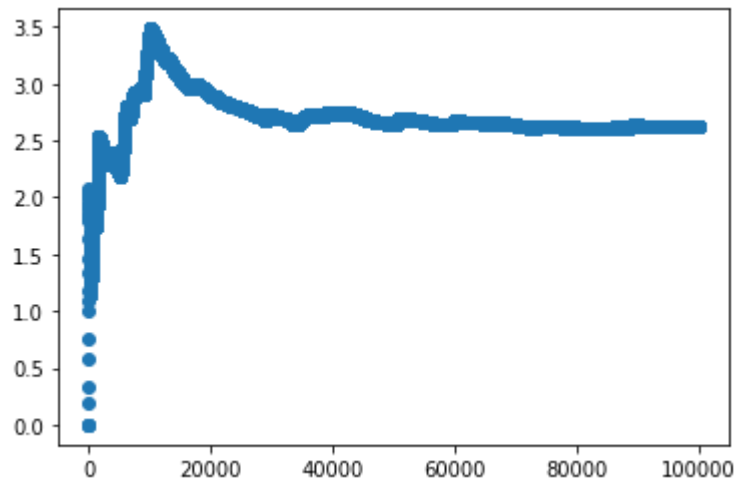
```
            q[2, 3, k + 1] = max(q[2, 3, k] - s[2, 3, k], 0)  # Netflix traffic leaving cell
            q[2, 4, k + 1] = max(q[2, 4, k] - s[2, 4, k], 0)  # FBI traffic leaving cell towe


for i in range(time_slots):
    if i < time_slots - 1:
        empirical_average_q[:,:,i + 1] = ((i + 1) * empirical_average_q[:,:,i] + q[:,:,i + 1]

t = np.arange(0,time_slots,dtype=int)

plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```



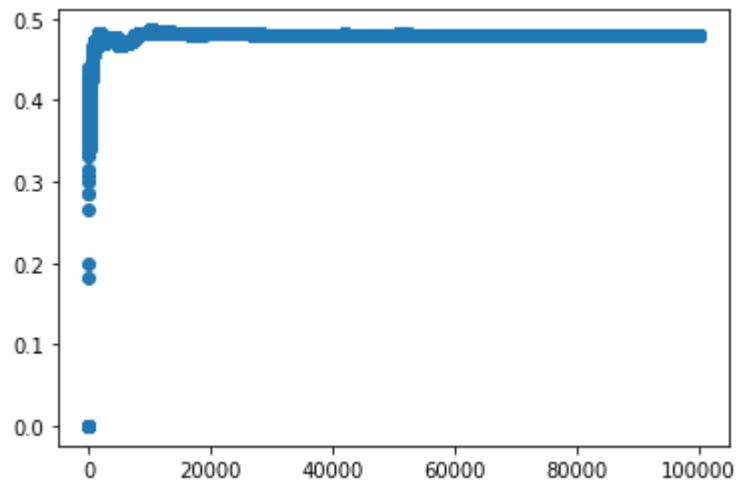

```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```




```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```

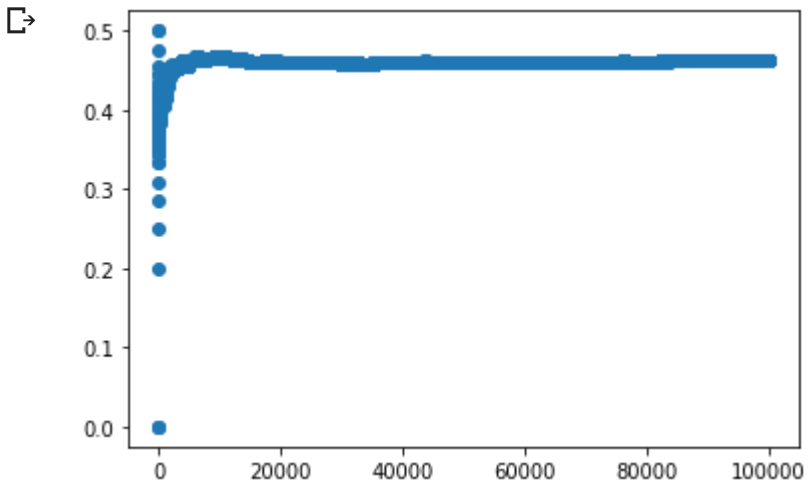



```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```




```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
```

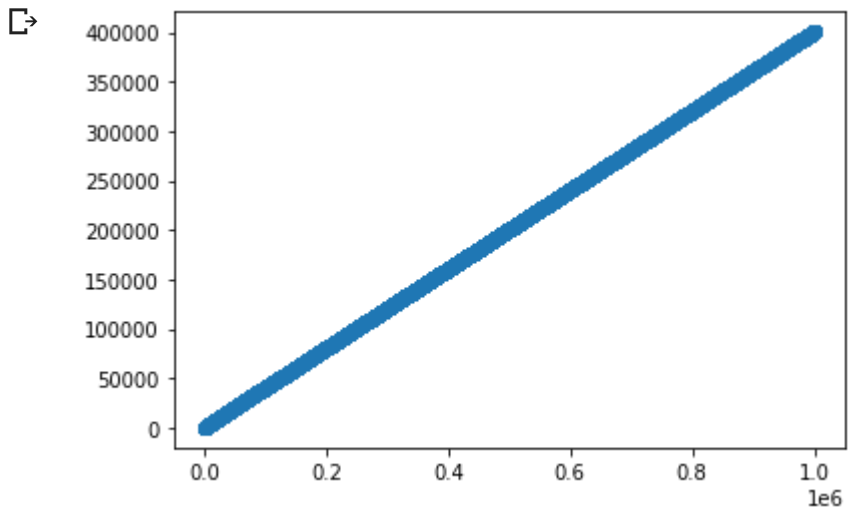
```
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np


#The following convention is used: node 0 = user 1, node 1 = user 2, node 2 = cell tower,
#node 3 = Netflix server, node 4 = FBI server.

t_nodes = 5 #number of transmitting nodes
r_nodes = 5 #number of receiving nodes
time_slots = 1000000 #length of simulation



#Queue Lengths
q = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)
empirical_average_q = np.zeros((t_nodes, r_nodes, time_slots))


#Average Arrival Rate
netflix_arrivals_node0 = 0.10
FBI_arrivals_node0 = 0.38
netflix_arrivals_node1 = 0.10
FBI_arrivals_node1 = 0.38


#Feasible Transmission Schedules
M = np.zeros((t_nodes,r_nodes,time_slots), dtype=int)

M[:,:,0] = np.array([[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,1] = np.array([[0,0,0,0,0],[0,0,2,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,2] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,2,0],[0,0,0,0,0],[0,0,0,0,0]])
M[:,:,3] = np.array([[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,2],[0,0,0,0,0],[0,0,0,0,0]])



maxweightM = np.zeros((4,time_slots), dtype=int)

s = np.zeros((t_nodes, r_nodes, time_slots), dtype=int)


c02 = np.zeros(time_slots)
c12 = np.zeros(time_slots)
c23 = np.zeros(time_slots)
c24 = np.zeros(time_slots)



for k in range(time_slots):

    #Find the weights
    if k%2 == 0: #even timeslots
        c02[k] = max(q[0, 3, k] - q[2, 3, k], q[0, 4, k] - q[2, 4, k], 0) #find max queue dif
        c12[k] = max(q[1, 3, k] - q[2, 3, k], q[1, 4, k] - q[2, 4, k], 0) #find max queue dif
```

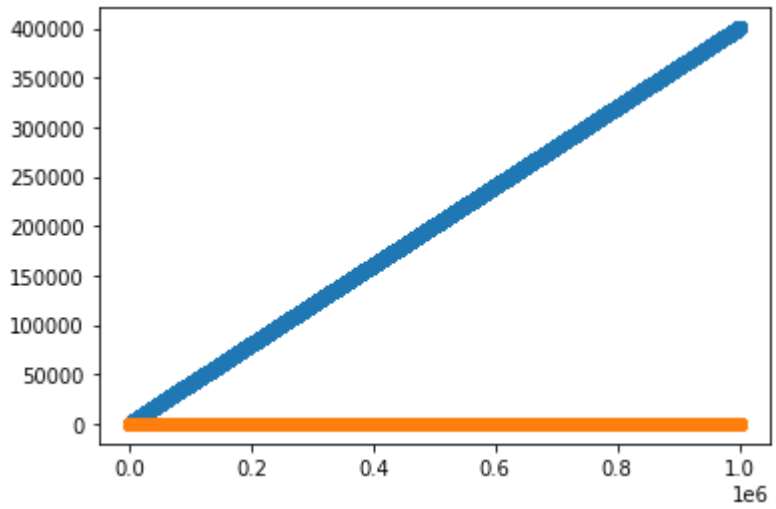
```
    for l in range(2):
        maxweightM[l, k] = c02[k] * M[0, 2, l] + c12[k] * M[1, 2, l]
else: #odd time slots
    c23[k] = q[2, 3, k] - q[3, 3, k] #find weight on link 23
    c24[k] = q[2, 4, k] - q[4, 4, k] #find wieght on link 24

    for l in range(2,4):
        maxweightM[l, k] = c23[k] * M[2, 3, l] + c24[k] * M[2, 4, l]

#Use the maxweight algorithm to find feasible transmission schedules
s[:, :, k] = M[:, :, np.argmax(maxweightM[:, k])] #find max-weight feasible transmission

#update the queues
if k < time_slots-1:

    a03 = np.random.binomial(1, netflix_arrivals_node0) #Netflix traffic at node 0
    a04 = np.random.binomial(1, FBI_arrivals_node0) #FBI traffic at node 0
    a13 = np.random.binomial(1, netflix_arrivals_node1) #Netflix traffic at node 1
    a14 = np.random.binomial(1, FBI_arrivals_node1) #FBI traffic at node 1

    q[0, 3, k + 1] += a03 #Netflix traffic arrving at node 0
    q[0, 4, k + 1] += a04 #FBI traffic arrving at node 0
    q[1, 3, k + 1] += a13 #Netflix traffic arriving at node 1
    q[1, 4, k + 1] += a14 #FBI traffic arrving at node 1

    if k%2==0: #even time slots

        if q[0, 3, k] - q[2, 3, k] > q[0, 4, k] - q[2, 4, k]: #Netflix traffic has max qu
            q[0, 3, k + 1] += max(q[0, 3, k] - s[0, 2, k], 0) #Netflix traffic leaving no
            q[0, 4, k + 1] += q[0, 4, k] #FBI traffic stays at node 0
            q[2, 3, k + 1] += min(s[0, 2, k], q[0, 3, k]) #Netflix traffic arriving at ce
        else:
            q[0, 3, k + 1] += q[0, 3, k] #Netflix traffic stays at node 0
            q[0, 4, k + 1] += max(q[0, 4, k] - s[0, 2, k], 0) #FBI traffic leaving node 0
            q[2, 4, k + 1] += min(s[0, 2, k], q[0, 4, k]) #FBI traffic arriving at cell t

        if q[1, 3, k]-q[2, 3, k] > q[1, 4, k]-q[2, 4, k]: #Netflix traffic has max queue
            q[1, 3, k + 1] += max(q[1, 3, k] - s[1, 2, k], 0) #Netflix traffic leaving no
            q[1, 4, k + 1] += q[1, 4, k] #FBI traffic stays at node 1
            q[2, 3, k + 1] += min(s[1, 2, k], q[1, 3, k]) #Netflix traffic arriving at ce
        else:
            q[1, 4, k + 1] = max(q[1, 4, k] - s[1, 2, k], 0) #FBI traffic leaving node 1
            q[1, 3, k + 1] += q[1, 3, k] #Netflix traffic stays at node 1
            q[2, 4, k + 1] += min(s[1, 2, k], q[1, 4, k]) #FBI traffic arriving at cell t
    else: #odd timeslots

        q[0, 3, k + 1] += q[0, 3, k]
        q[0, 4, k + 1] += q[0, 4, k]
        q[1, 3, k + 1] += q[1, 3, k]
        q[1, 4, k + 1] += q[1, 4, k]
```

```
            q[2, 3, k + 1] = max(q[2, 3, k] - s[2, 3, k], 0)  # Netflix traffic leaving cell
            q[2, 4, k + 1] = max(q[2, 4, k] - s[2, 4, k], 0)  # FBI traffic leaving cell towe


for i in range(time_slots):
    if i < time_slots - 1:
        empirical_average_q[:,:,i + 1] = ((i + 1) * empirical_average_q[:,:,i] + q[:,:,i + 1]

t = np.arange(0,time_slots,dtype=int)

#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```



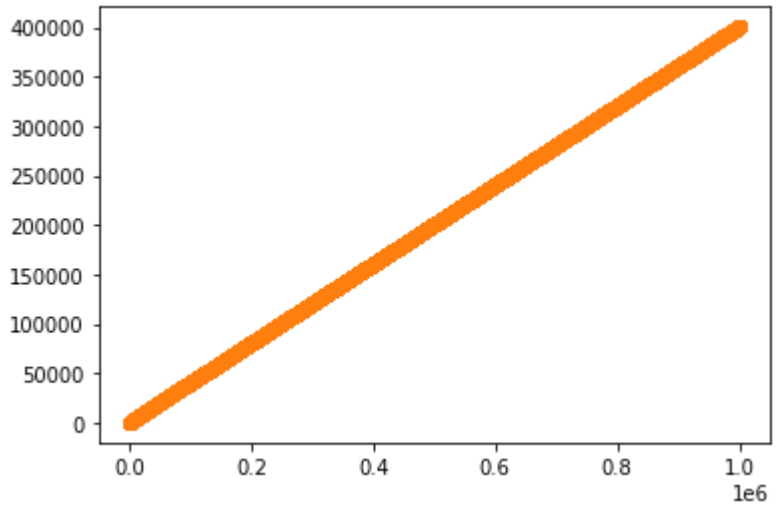

```
plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```


```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```




```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```


```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```




```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```

```
#plt.scatter(t,empirical_average_q[0,3,:]) #netflix traffic at user 1
#plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at user 1
#plt.scatter(t,empirical_average_q[1,3,:]) #netflix traffic at user 2
plt.scatter(t,empirical_average_q[1,4,:]) #FBI traffic at userr 2
#plt.scatter(t,empirical_average_q[2,3,:]) #netflix traffic at cell tower
#plt.scatter(t,empirical_average_q[2,4,:]) #FBI traffic at cell tower
plt.show()
```