ECE158a Assignment 3

solution

1. Consider a wireless link between a source node and a destination node. Let the time indexed by discrete time slots $t \in \{0, 1, \dots\}$, with one time slot corresponding to the transmission time of a packet. Due to the channel condition, the wireless link could be either in the ON state or the OFF state at each time slot t. If the link is in the ON state, then a packet could be successfully transmitted through the link; on the other hand, if the link is in the OFF state and the source node sends a packet, then that packet would be dropped by the link. Assume the link state to be independent over time and at each time slot, the link is in ON state with probability p, and in OFF state with probability $1 - p$. If a packet is received at the destination, the destination immediately transmits a acknowledgment to the source node (with a negligible transmission time). In contrast, if a packet is dropped in the wireless link, then the source node could only know that when the wait for acknowledgment is timed out. We assume the time out period is two time slots, in other words, if a packet is sent at slot 1 and is dropped, then the source node would only be able to know until the end of slot 2 (and could then attempt retransmission starting from slot 3).

   The packets are enqueued in a first-in-first-out (FIFO) queue at the source node before being transmitted to the destination. The packet arrival in this queue is independent across time, and at each time slot, with probability λ a packet arrives, and with probability $1 - \lambda$ no packet arrives.

   (a) Assume the source node is able to perceive the link state before transmission, and it only sends packet when the link state is ON. Let T be the time required for packet at the head-of-line of the queue to be served (transmitted with success). Determine the distribution of T assuming p is known. Based on the result, what is the condition required for λ such that the queue is stable (queue length does not grow unboundedly)?

   (b) Following (a), and assume the queue is stable. Determine the mean waiting time in the queue for these packets.

   (c) Now consider the source node that is not able to perceive the link state before transmission. At each time slot, if the source node is not waiting for an acknowledgment, then it simply attempts to transmit the head-of-line packet (if the queue is not empty). On the other hand, if the source node is waiting for an acknowledgment, then it does not transmit any packet. Again, determine the distribution of T and derive the condition on λ such that the queue is stable, and determine the mean waiting time in the queue for these packets.

   A:

(a) Since at each time slot, the probability that the packet is transmitted is p, hence the distribution of T is a geometric distribution with success probability p. More specifically, the distribution of T is given by

$$\Pr\{T = k\} = p(1 - p)^{k-1}, \quad \text{for } k = \{1, 2, \dots\}.$$

The mean of T could be derived as

$$\mathbb{E}[T] = \sum_{k=1}^{\infty} kp(1-p)^{k-1} = \frac{1}{p}$$

(Note: if you are not familiar with the evaluation of this summation, notice that $(1-p)\mathbb{E}[T] = \sum_{k=1}^{\infty} kp(1-p)^k = \sum_{k=2}^{\infty}(k-1)p(1-p)^{k-1}$, then $p\mathbb{E}[T] = \mathbb{E}[T] - (1-p)\mathbb{E}[T] = p + \sum_{k=2}^{\infty} p(1-p)^{k-1} = \sum_{k=1}^{\infty} p(1-p)^{k-1} = 1$)

Therefore the service rate is

$$\mu = \frac{1}{\mathbb{E}[T]} = p,$$

hence the condition required for stable queue is $\lambda < p$.

(b) We now determine $\mathbb{E}[T^2]$, by definition we have

$$\mathbb{E}[T^2] = \sum_{k=1}^{\infty} k^2 p(1-p)^{k-1}$$

Multiplying both sides by $(1-p)$, we get

$$(1-p)\mathbb{E}[T^2] = \sum_{k=1}^{\infty} k^2 p(1-p)^k = \sum_{k=2}^{\infty}(k-1)^2 p(1-p)^{k-1}$$

Hence

$$\begin{aligned} p\mathbb{E}[T^2] &= \mathbb{E}[T^2] - (1-p)\mathbb{E}[T^2] = p + \sum_{k=2}^{\infty}(k^2-(k-1)^2)p(1-p)^{k-1} \\ &= p + \sum_{k=2}^{\infty}(2k-1)p(1-p)^{k-1} = \sum_{k=1}^{\infty}(2k-1)p(1-p)^{k-1} = \frac{2}{p} - 1 \\ \Rightarrow \mathbb{E}[T^2] &= \frac{2-p}{p^2} \end{aligned}$$

Then from the M/G/1 result, we have the mean waiting time as

$$\mathbb{E}[W] = \frac{\lambda \mathbb{E}[T^2]}{2(1-\rho)} = \frac{\lambda \frac{2-p}{p^2}}{2(1-\rho)} = \frac{\rho}{1-\rho}\left(\frac{1}{p} - \frac{1}{2}\right)$$

where $\rho = \lambda/p$.

.

Alternative:
If we assume λ and p to be small, then packet arrival and service would be unlikely to occur at the same time slot. Under this scenario, you may also use the M/M/1 analysis introduced in class to obtain the mean queue length as

$$\mathbb{E}[L] = \frac{\rho}{1-\rho}$$

Since the mean service time is $\mathbb{E}[T] = \frac{1}{p}$, the mean waiting time is given by

$$\mathbb{E}[W] = \mathbb{E}[L]\frac{1}{p} = \frac{\rho}{1-\rho}\frac{1}{p}$$

(It is fine if you stop at the mean queue length since this step requires the knowledge that for memoryless arrivals, the mean queue length sampled upon packet arrival is equal to the mean queue length at any time $\mathbb{E}[L]$.)
Note: This result differs with the result obtained from M/G/1 result since this result obtained from M/M/1 analysis requires the assumption that packet arrival and service do not occur at the same time slot (which is still an approximation of this system even for small λ and p). Note that for p very small, we have $\frac{1}{p} - \frac{1}{2} \approx \frac{1}{p}$, and the approximation becomes more accurate as p becomes smaller.

.

(c) Note that for the source node that is not able to perceive the link state, if the link state is OFF upon the transmission attempt, then the transmission would fail, and the next attempt would be 2 slots later. Therefore, the service time T could only take on values $\{1, 3, 5, \dots\}$, and the distribution of T is given by

$$\Pr\{T = 2k-1\} = p(1-p)^{k-1}, \quad \text{for } k = \{1, 2, \dots\}.$$

The mean of T could be derived as

$$\mathbb{E}[T] = \sum_{k=1}^{\infty}(2k-1)p(1-p)^{k-1} = \frac{2}{p} - 1$$

Hence the condition for the queue to be stable is $\lambda < 1/\mathbb{E}[T] = \frac{p}{2-p}$.
We now determine $\mathbb{E}[T^2]$ as follows:

$$\begin{aligned}
\mathbb{E}[T^2] &= \sum_{k=1}^{\infty}(2k-1)^2 p(1-p)^{k-1} \\
(1-p)\mathbb{E}[T^2] &= \sum_{k=1}^{\infty}(2k-1)^2 p(1-p)^{k} = \sum_{k=2}^{\infty}(2k-3)^2 p(1-p)^{k-1} \\
\Rightarrow p\mathbb{E}[T^2] &= \mathbb{E}[T^2] - (1-p)\mathbb{E}[T^2] = p + \sum_{k=2}^{\infty}\left((2k-1)^2 - (2k-3)^2\right)p(1-p)^{k-1} \\
&= p + \sum_{k=2}^{\infty} 8(k-1)p(1-p)^{k-1} = p + \sum_{k=1}^{\infty} 8(k-1)p(1-p)^{k-1} = p + 8(\frac{1}{p} - 1) \\
\Rightarrow \mathbb{E}[T^2] &= 1 + \frac{8(1-p)}{p^2}
\end{aligned}$$

Then from the M/G/1 result, we have the mean waiting time as

$$\mathbb{E}[W] = \frac{\lambda\mathbb{E}[T^2]}{2(1-\lambda\mathbb{E}[T])} = \frac{\lambda\left(1 + \frac{8(1-p)}{p^2}\right)}{2\left(1 - \lambda(\frac{2}{p} - 1)\right)}.$$

2. Due to the complexity of finding good scheduling algorithms, some researchers have proposed practical sub-optimal algorithms such as the following algorithm: At each time, the algorithm selects d permutation matrices chosen *uniformly* at random from amongst the possible $N!$ matchings; let the schedule at time t be the one that has the largest weight of the d selected matchings, where the weight of a matching is defined as the sum of the lengths of the queues selected by the matching. For example, if $\mathbf{X} = [X_{ij}]$ is the matrix of queue lengths, and $\mathbf{S} = [S_{ij}]$ is the matching, then the weight of the matching is defined by $W = \sum_{ij} X_{ij} S_{ij}$.

   (a) Given the schedule length matrix, we define the size of a matching as the number of non-empty queues selected by the matching, and a maximum size matching is a matching that has the maximum size (which may not be unique). Construct an example to show that the algorithm described above can result in a matching which is not of maximum size.

   (b) Consider an $N \times N$ switch. Assuming VOQ$_{ij}$ has a non-zero backlog at a given time; calculate a lower bound on the probability that this VOQ$_{ij}$ remains unserved.
   Hint: Consider the cases that VOQ$_{ij}$ is not served in all randomly selected schedules.

   (c) Discuss what happens as $N \to \infty$, while keeping $d \leq cN$, where $c > 0$ is a constant. In particular, find a condition on λ_{ij} for which, the queue becomes unstable (gets larger and larger).
   Hint: Use the condition $d \leq cN$ and the result from (b) to obtain an upper bound bound on the probability that the VOQ$_{ij}$ is served.

(a) Note that many examples would satisfy the described conditions, here we are giving only one of those. Consider $N = 3$, and $d = 2$, and suppose the queue length matrix is given by

$$X = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 5 & 3 \\ 2 & 2 & 6 \end{bmatrix},$$

while the randomly selected d matchings are

$$S_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The weight of S_1 is $W_1 = 0 + 5 + 6 = 11$, and the weight of S_2 is $W_2 = 1 + 2 + 6 = 9$. According to the approach introduced, S_1 would be selected. However, we could easily check that S_1 is not of maximum size (the size of S_1 is 2, while the size of S_2 is 3).

(b) We assume that the random selection of d matching is with replacement. For each randomly selected matching, the probability that a specific VOQ$_{ij}$ is not served is $(1 - \frac{1}{N})$. Therefore, the probability that a specific VOQ$_{ij}$ is not served in all d randomly selected matchings is $(1 - \frac{1}{N})^d$.
Since VOQ$_{ij}$ would definitely be unserved if all the randomly selected d matchings do not serve VOQ$_{ij}$, hence we have a lower bound on the probability that VOQ$_{ij}$ remains unserved as

$$\Pr\{\text{VOQ}_{ij} \text{ unserved}\} \geq \left(1 - \frac{1}{N}\right)^d$$

(c) From (b), we then derive an upper bound on the probability that VOQ$_{ij}$ is served as

$$\begin{aligned} \mu_{ij} &\triangleq \Pr\{\text{VOQ}_{ij} \text{ served}\} \\ &= 1 - \Pr\{\text{VOQ}_{ij} \text{ unserved}\} \\ &\leq 1 - \left(1 - \frac{1}{N}\right)^d \\ &\leq 1 - \left(1 - \frac{1}{N}\right)^{cN} \qquad (\text{since } d \leq cN \text{ and } 1 - \tfrac{1}{N} < 1) \\ &\to 1 - e^{-c} \quad \text{as } N \to \infty \end{aligned}$$

Therefore, if the arrival rate at VOQ$_{ij}$ satisfies $\lambda_{ij} > 1 - e^{-c}$, then the queue length of VOQ$_{ij}$ is unstable.

3. Represent a directed graph having N vertices with the vertex set $\mathcal{V} = \{1, 2, \ldots, N\}$ and a $N \times N$ weight matrix W, with $w_{ij} = 0$ if there exists no edge from node i to j, and $w_{ij} > 0$ otherwise.

   (a) Write a function in MATLAB (func1.m) or Python (func1.py) which implements Dijkstra's algorithm for finding the minimum weight path to some node T from all the other nodes. This function should take as input the weight matrix W, and the target node T, and output an $N \times 1$ vector D containing the weights of the minimum weight paths from all the nodes to T.

   (b) Write another function in MATLAB (func2.m) or Python (func2.py) which finds the minimum weight path to some node T by the listing out all the paths and then sorting them. This function should also output an $N \times 1$ vector containing the weights of the minimum weight paths from all the nodes to T.

   (c) Write a MATLAB function (func3.m) or Python (func3.py) which takes in as inputs (W, D, T, i) where $i \in \mathcal{V} \setminus \{T\}$; and outputs the minimum weight path from i to T.

   (d) Finally, use these MATLAB (or Python) functions to do the following:

      - Create a graph with $N = 8$, by placing edges between any two nodes randomly with probability $p = 0.7$, and assign weights randomly from the range $[0, 5]$ to the edges created. Generate the vector D for $T = 8$, using the two functions func1.m and func2.m and calculate the ratio of the running time of func2.m to the running time of func1.m
      - Create a graph with $N = 20$ as before, and obtain the vector D for $T = 20$ by using func1(.m or .py). Then use func3 to find the path from node $i = 1$ to T.
      - Create graphs for different values of N in the range $[10, 100]$ as before. For every value of N, use func1.m to obtain D corresponding to $T = 1$ and compute the average running time over 10 trials. Then plot the average running time versus N. Does this plot match the theoretical results?

A: the sample code would be like this (part b is not covered since it is only brute-force), for pat d the average execution time figure is as shown below. The average execution time roughly scales as $O(N^2)$.

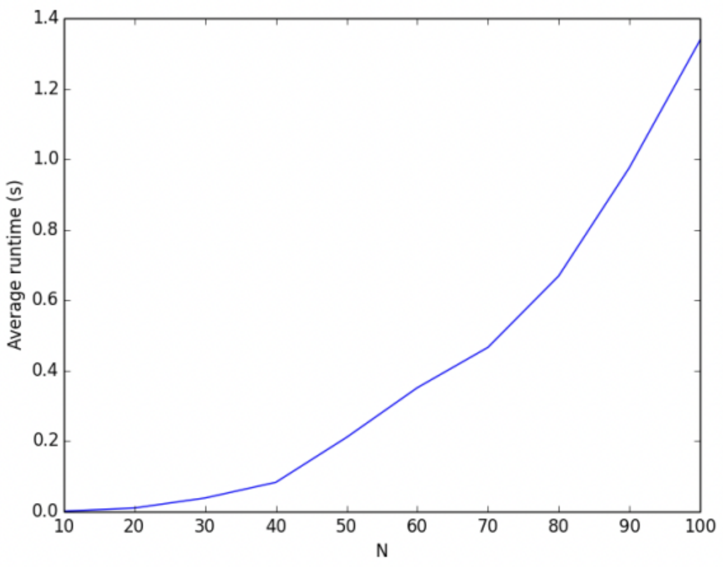

Figure 1: Average execution time plot

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

def func1(W, T):
        """
        W: numpy array representing the graph
        T: Destination node in range [1, ..., N]
        """
        T = T - 1  # Relabel node number in range [0, ..., N-1]
        N = len(W)
        working = range(N)
        working.remove(T)
        weight = np.ones(N) * N * W.max()
        weight[T] = 0;

        while (len(working) > 0):
                tempWeight = weight.copy()
                minWeight = weight[working[0]]
                for i in working:
                        update = weight[i]
                        for j in range(N):
                                if W[i][j] > 0:
                                        update = min(update, W[i][j] + weight[j])
                        tempWeight[i] = update

                        if (update < minWeight):
                                minWeight = update;
                weight = tempWeight
                ind = list(weight).index(minWeight)
                working.remove(ind)

        return weight
```

```
def func2(W, D, T, i):
        ret = []
        N = len(D)
        T = T - 1 # Relabel node number in range [0, ..., N-1]
        i = i - 1 # Relabel node number in range [0, ..., N-1]
        node = i
        while (node != T):
                ret.append(node+1)
                for j in range(N):
                        if (j == node):
                                continue
                        if (W[node][j] > 0 and W[node][j] + D[j] == D[node]):
                                node = j
                                break

        ret.append(node+1)
        return ret

def func3(N, p):
        G = np.random.rand(N, N)
        G = G < p
        weight = np.random.uniform(0, 5, (N, N))

        return G * weight



p = 0.7
timeMeasure = []
NList = range(10, 101, 10)
T = 1
for N in NList:
        print N
        start = time.time()
        for i in range(10):
                W = func3(N, p)
                D = func1(W, T)
        duration = time.time() - start;
        timeMeasure.append(duration / 10)

plt.plot(NList, timeMeasure)
plt.xlabel("N")
plt.ylabel("Average runtime (s)");
plt.show()
```