Table of contents

Question 1 2
Question 2 4
CompareNon-Persistent and Persistent 5
Go Through Project Design 6

Name: Sijia Lu
Group Number: 68
UserID: s96lu
StudentID: 20649138

1. Simulate a persistent CSMA/CD protocol. Show the efficiency and throughput (in Mbps) of the LAN as a function of N(20, 40, 60, 80, 100) for A = 7, 10 and 20 Packet/sec, R = 1Mbps, and L = 1500 bits. Comment on the behavior of the graphs

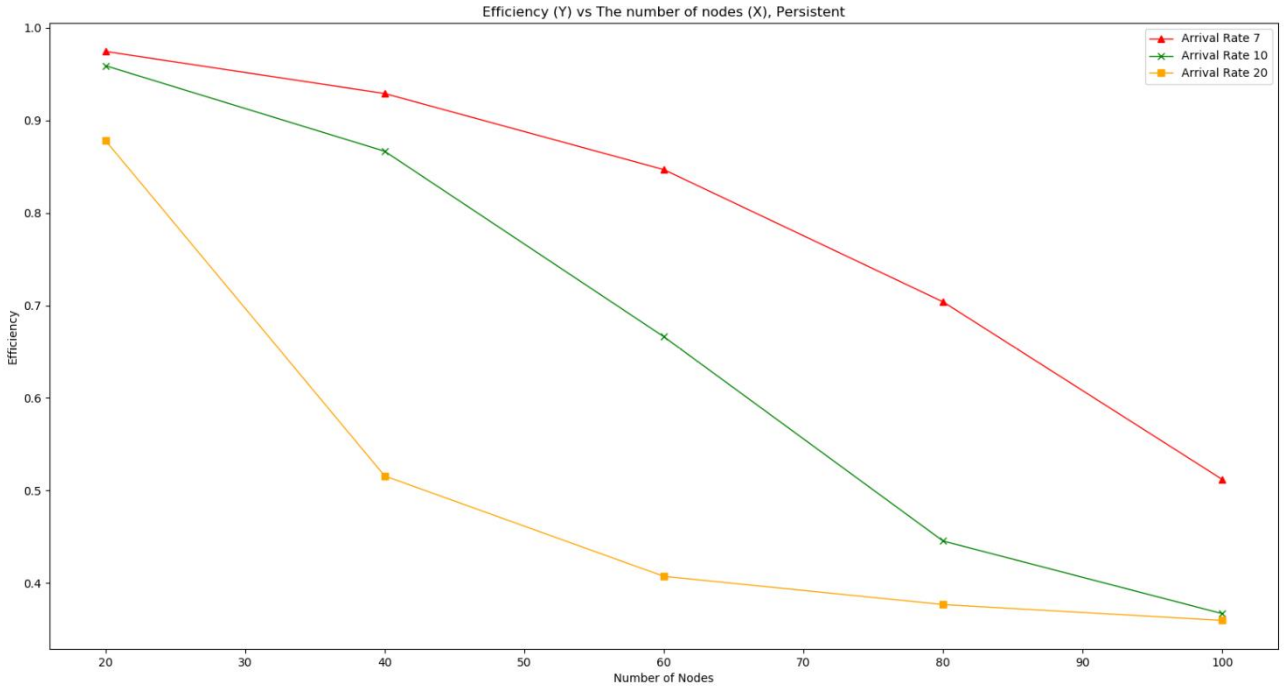

Figure 1. Efficiency vs Nodes, Persistent

Figure 1 presents the persistent CSMA/CD relationship between efficiency and nodes. When the arrival rate is 7, the efficiency decreased comparatively steady as the number of nodes increased. When the arrival rate is 10 and 20, the efficiency showed an apparent decrease, especially when the arrival rate = 20. The efficiency dropped from 95% to 91% when nodes increased from 20 to 40 at A = 20. And when A = 10 and 20, N = 100, the efficiency becomes almost the same.

As a result, efficiency drops if the arrival rate remains constant, and more nodes get involved. The efficiency is the highest when the arrival rate is the lowest, and efficiency is the lowest when the arrival rate is the highest. It is understandable since the higher arrival rate means more collisions and more busy wait.

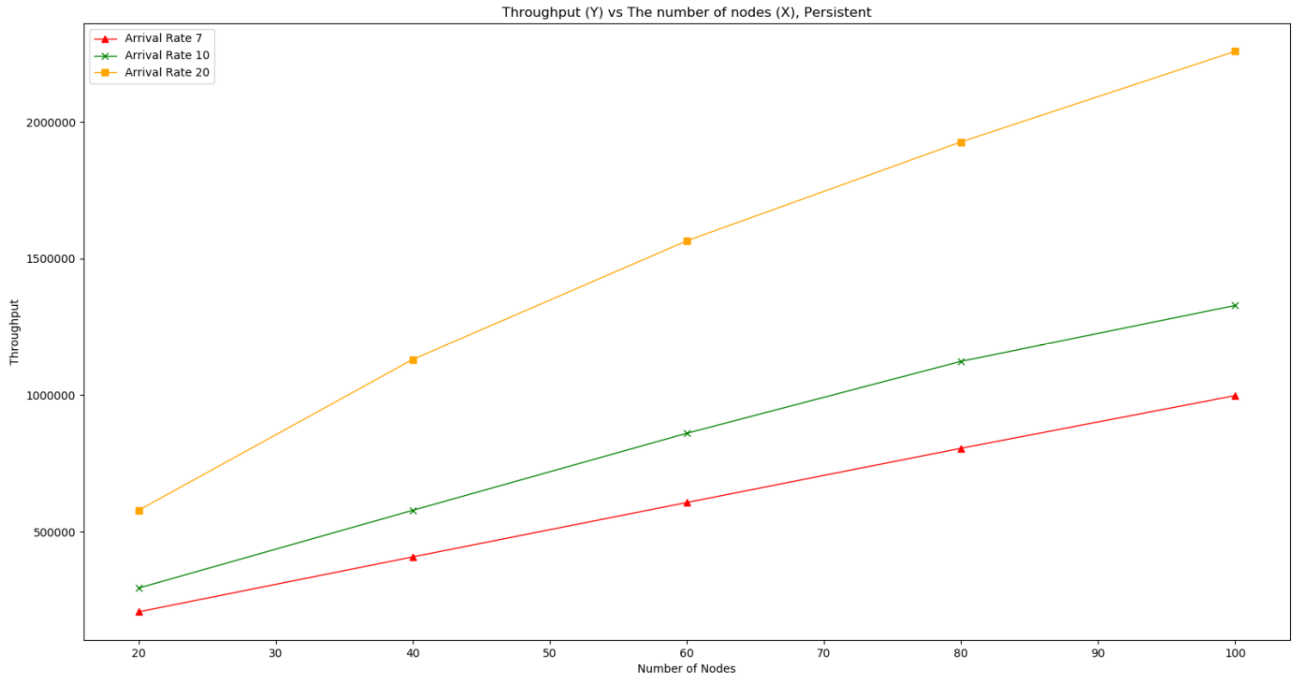

Figure 2. Throughput vs Nodes, Persistent

Figure 2 shows the relationship between throughput and the number of nodes in persistent CSMA/CD. If arrival rate remains constant, throughput increases as more nodes get involved. The throughput when A = 20 is the highest (2625693), which is almost twice the throughput when A = 10 (1289379). It is reasonable because when the number of nodes get larger, there are more packets and successful packets get sent within a period of time.

As a result, when arrival rate remains constant, more nodes will generate higher throughput.

2. Show the efficiency and throughput (in Mbps) of non-persistent CSMA/CD protocol for the same network parameters used in question 1. Also, comment on the graph and compare between the results obtained in question 1 and question 2.

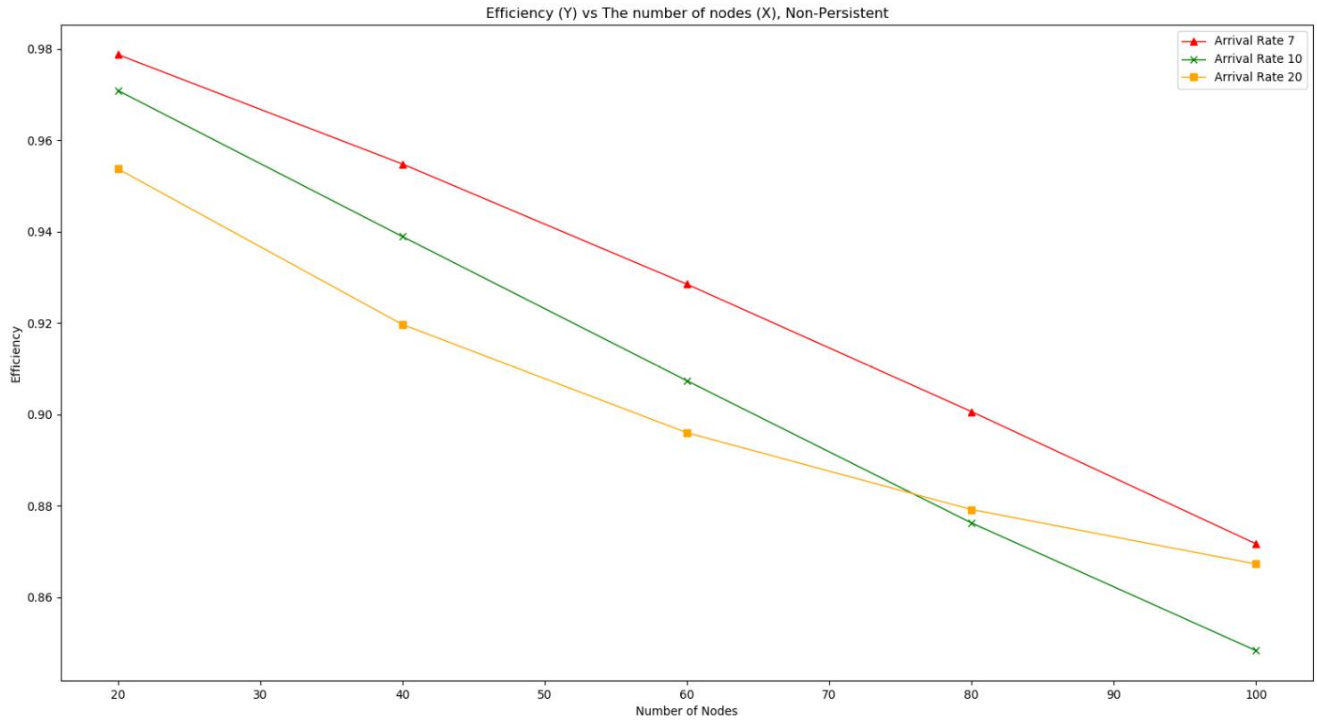

Figure 3. Efficiency vs Nodes, Non_Persistent

Figure 3 gives the relationship between efficiency and the number of nodes under non-persistent CSMA/CD. The efficiency drops a larger amount when the arrival rate is 7, 10 than 20. It is because that when arrival rate = 7 and 10, the efficiency is almost straight lines with a negative slope. However, when the arrival rate becomes 20, the efficiency is even better than A = 10. So, it is reasonable to assume that when the number of nodes gets closer to 100, the efficiency will also be close no matter what the arrival rate is.

As a result, for non-persistent CSMA/CD, more nodes and higher arrival rates seem to have better efficiency.

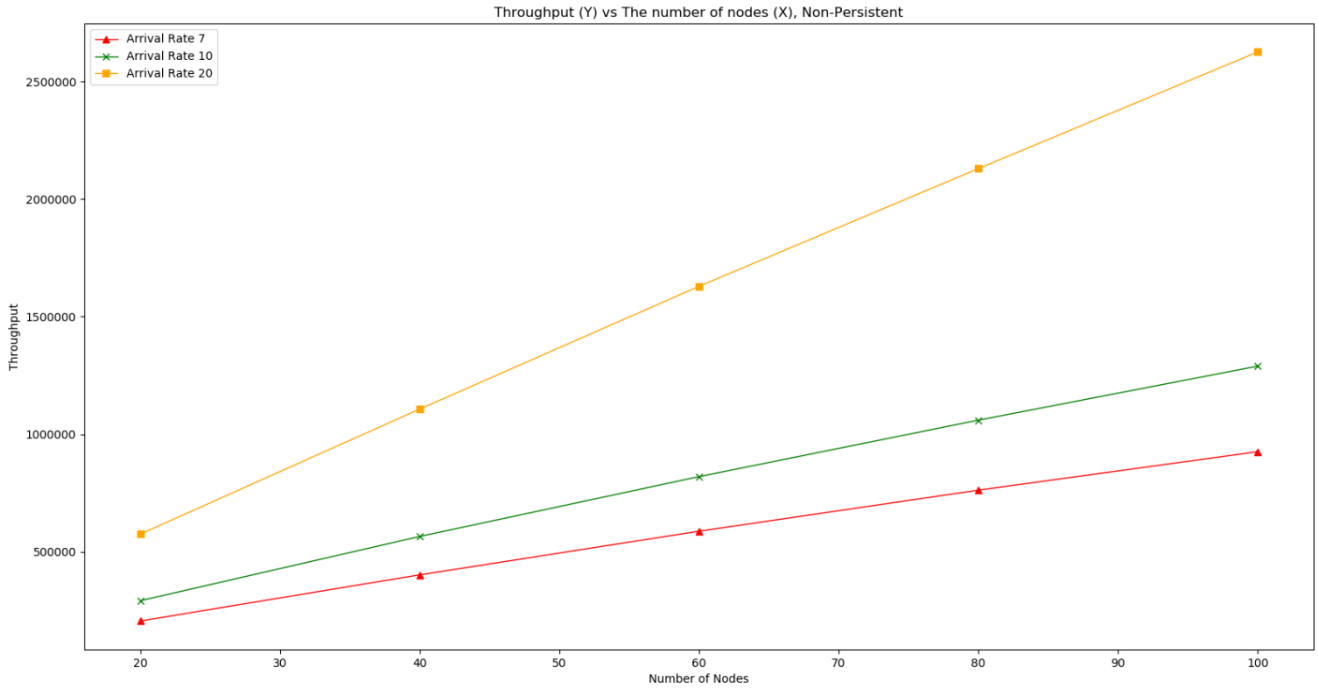

Figure 4. Throughput vs Nodes, Non_Persistent

Figure 4 shows the relationship between throughput and the number of nodes under non-persistent CSMA/CD. The throughput increases as more nodes involved when arrival rate remains constant. In addition, when the number of nodes remains constant, the higher the arrival rate, the higher the throughput is.

Compare Non-Persistent and Persistent:

The efficiency and throughput of non-persistent and persistent CSMA/CD have a similar trend or slope. However, the first difference is efficiency as the arrival rate rises. In persistent, if all other parameters remain the same, the efficiency when A = 20, is lower than A = 10, is lower than A = 7. However, in non-persistent, the efficiency when A = 20 eventually surpasses the efficiency when A = 10.

Throughput is almost identical except that the throughput is higher under non-persistent CSMA/CD. It is because throughput = (number of successful packets * packet length) / (operating time). When a node finds bus busy under non-persistent CSMA/CD, it back off for some time rather than keeps sensing, so the total number of packets sent is less than persistent CSMA/CD

Description:

The design of the project is as following:

Each LAN object has a lan[] array which stores each node. Each node also has an array eventQueue[] which stores the randomly generated arrival time. The first step is to populate eventQueue[] of each node, then append each node into lan[], so that LAN object can track each node and their time.

```
    def populate_node_stack(self):  # populate packet stack of each nod
e
        for i in range(self.N):
            node_time = 0
            node_events = []
            while node_time < self.T:
                random_interval_time = self.get_random_variable(1.0 / s
elf.A)
                node_time = node_time + random_interval_time
                node_events.append(arrival.ArrivalEvent(node_time)) # n
ode_events is an array, elements are ArrivalEvent
            self.lan.append(lan_node.Node(node_events, i))
```

After population, it is necessary to select the earliest time of a node by doing a simple for loop. We call that node “sender_node”, and its earliest time “sender_node”.next_event_time.

```
    def determine_next_sender(self): # find the next sender in LAN by s
imply comparing each node's earliest time
        sender_index = -1
        next_packet_time = self.T
        for i in self.lan:
            if next_packet_time > i.next_event_time and len(i.eventQueu
e) > 0:
                next_packet_time = i.next_event_time
                sender_index = i.index
        return sender_index
```

Then we come to the trick part, busy wait function and collision function.

```
    def have_collision(self, node, new_time): # call this function when
 having collisions
        node.collision_count += 1
        node.total_packets += 1
        node.busy_count = 0
        self.total_collisions += 1
        self.total_packets += 1
        if node.collision_count > self.K_max: # If collision count exce
eds K_max, drop the packet
            node.eventQueue.popleft()
            node.collision_count = 0
```

```
            node.dropped_packets += 1
            self.dropped_packets += 1
            if len(node.eventQueue)>0 and node.eventQueue[0].event_time
 > node.next_event_time:
                node.next_event_time = node.eventQueue[0].event_time
        else:                                   # exponential backoff
            backoff_time = self.exponential_backoff(node.collision_coun
t)
            node.next_event_time = max(new_time + backoff_time, node.ev
entQueue[0].event_time)
```

```
    def have_busy_wait(self, node, new_time): # call this when busy wai
t
        node.busy_count += 1
        backoff_time = 0
        if node.busy_count >= self.K_max:
            node.eventQueue.popleft()
            node.busy_count = 0
            node.dropped_packets += 1
            node.total_packets += 1
            self.dropped_packets += 1
            self.total_packets += 1
        else:                                   # exponential backoff
            node.next_event_time = max(new_time + backoff_time, node.ev
entQueue[0].event_time)
```

Non-persistent and persistent have the same collision function but slightly different busy wait function. The lab manual already provided enough details about the differences. For non-persistent, the node back offs for some time after busy wait, but persistent keeps sensing the channel until it is free. In my design, there are no parameter to show if the channel is busy or not, my approach is before successfully sending any packets, we do a for loop of the entire lan[], and check:

```
if self.lan[i].next_event_time < first_bit_arrival
```

if the above is true, then there is a collision on that node, "lan[i]", we do exponential backoff on that node. The same methodology applies on busy wait,

```
if first_bit_arrival <= self.lan[i].next_event_time <= last_bit_arrival
:
```

if the above is true, then this node will see a busy wait, what we do depends on the type of CSMA/CD.

The results we want to get eventually is throughput, efficiency and arrival rate of the entire LAN object. Then we must keep updating

```
        self.N = N          # Number of nodes
        self.A = A          # Average number of packets per second
```

```
        self.T = T          # Simulation Time
        self.D = 10         # Distance between two neighbouring nodes,
10m
        self.K_max = 10     # max_wait
        self.R = 1e6        # Transmission speed, 1Mb
        self.L = 1500       # packet length is 1500 bits
        self.C = 3e8        # Speed of light
        self.S = (2/3) * self.C  # Propagation speed
        self.t_prop = self.D / self.S # Propagation Time
        self.t_trans = self.L / self.R # Transmission Time
        self.dropped_packets = 0
        self.successful_packets = 0
        self.total_packets = 0
        self.total_collisions = 0
```

these variables.

After all calculations are done:,

```
        print('N =', self.N)
        print('A =', self.A)
        print('Successfully transmitted:', self.successful_packets)
        print('Total packets:', self.total_packets)
        print('Efficiency:', self.successful_packets / self.total_packe
ts)
        print('Throughput:', self.successful_packets * self.L / self.T)
        print('Total collisions:', self.total_collisions)
        print('Dropped packets:', self.dropped_packets)
```

we can simply print the results to the console.

After running my code, you will see four txt files:

"**persistent_data.txt**": This file is the real persistent csma/cd results you get after running code

"**non_persistent_data.txt**": This file is the real non-persistent csma/cd results you get after running code

"**persistent_performance.txt**": This is the sample txt file I got after running my own persistent csma/cd experiments, I used this to draw graph

"**non_persistent_performance.txt**": This is the sample txt file I got after running my own non-persistent csma/cd experiments, I used this to draw graph