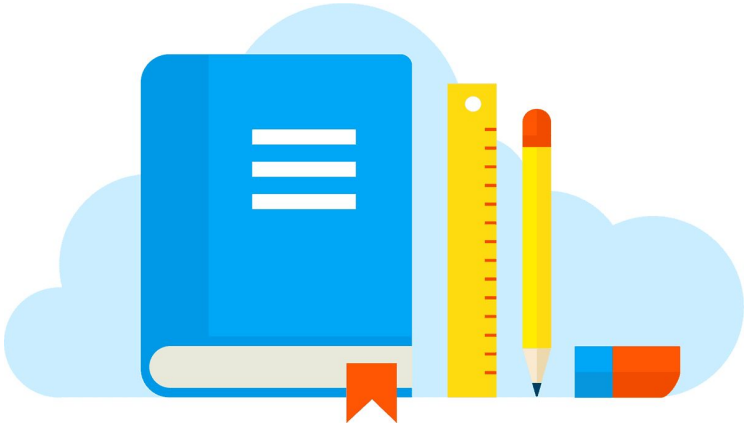


Computer Networks

Assignment 3

Soumalya Kundu

Roll: 001810501033

Date: 14/12/2020

Problem Statement:

In this assignment, we have to implement 1-persistent, non-persistent and p-persistent CSMA techniques. Measure the performance parameters like throughput and forwarding delay experienced by the CSMA frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by varying p.

Approach and Description:

I implemented this assignment as an extension of the previous assignment of Flow control. The whole structure is based upon the GoBackN protocol of Data Link Control Layer. Standard Ethernet 802.3 frame format is used to achieve this. In this assignment I developed a **dynamic approach** of creating any number of sender and receiver nodes.

The numbers need to be declared in the **const.py** file in the **package module**. All sender and receiver nodes along with the channel node and both of the dispatcher nodes are separate processes and can run parallely without any dependencies with each other. For each sender receiver pair, channel and dispatcher nodes create **separate threads** (denoted as threadpool) inside to handle all the transmission. This is to achieve an independent environment for each of the sender receiver pairs so that we don't get into any deadlock situation.

The **CSMA protocols have been implemented into the sender.py** file in the form of 3 functions. For achieving the desired result I used 2 global variables shared among all the sender nodes namely **collision** and **idle**. The idle variable is used to notify all the senders that the channel (implemented with a pipe) is idle at that point of time and they can send data over the channel. The collision variable is to notify that the data already sent by one sender has been collided with any one of the data packets sent by another sender. The rest of the protocol is very simple and straightforward. The underlying flow control protocol is GoBack N ARQ.

First of all one sender senses the channel whether the channel is busy or not(using the help of idle variable). If it finds the channel is idle and no one is sending the data to the channel, the sender will grab the channel and go to sleep for **vulnerableTime** (a constant defined in package/const file). This Vulnerable time imitates the time required for a sender node to upload the data frame to the channel (frame Format Time) and the time to reach the data to travel to that media to the furthest of the node so that it can sense the channel as busy. In this vulnerableTime period if any sender comes, it will find the channel as idle and may start to send data packets, resulting in collision. Next if any collision doesn't happen the sender will set the idle variable as False denoting the sender is busy at that point of time and goes to sleep for **propagationTime.** This time denotes the frame time that will take to reach to the furthest station located from the current sending station. Then the sender actually sends the data to the **senderToChannel pipe.** This pipe is shared across all the sender processes and acts as a unified shared channel for sending data.

Below is a general overall flow diagram of the current program structure.

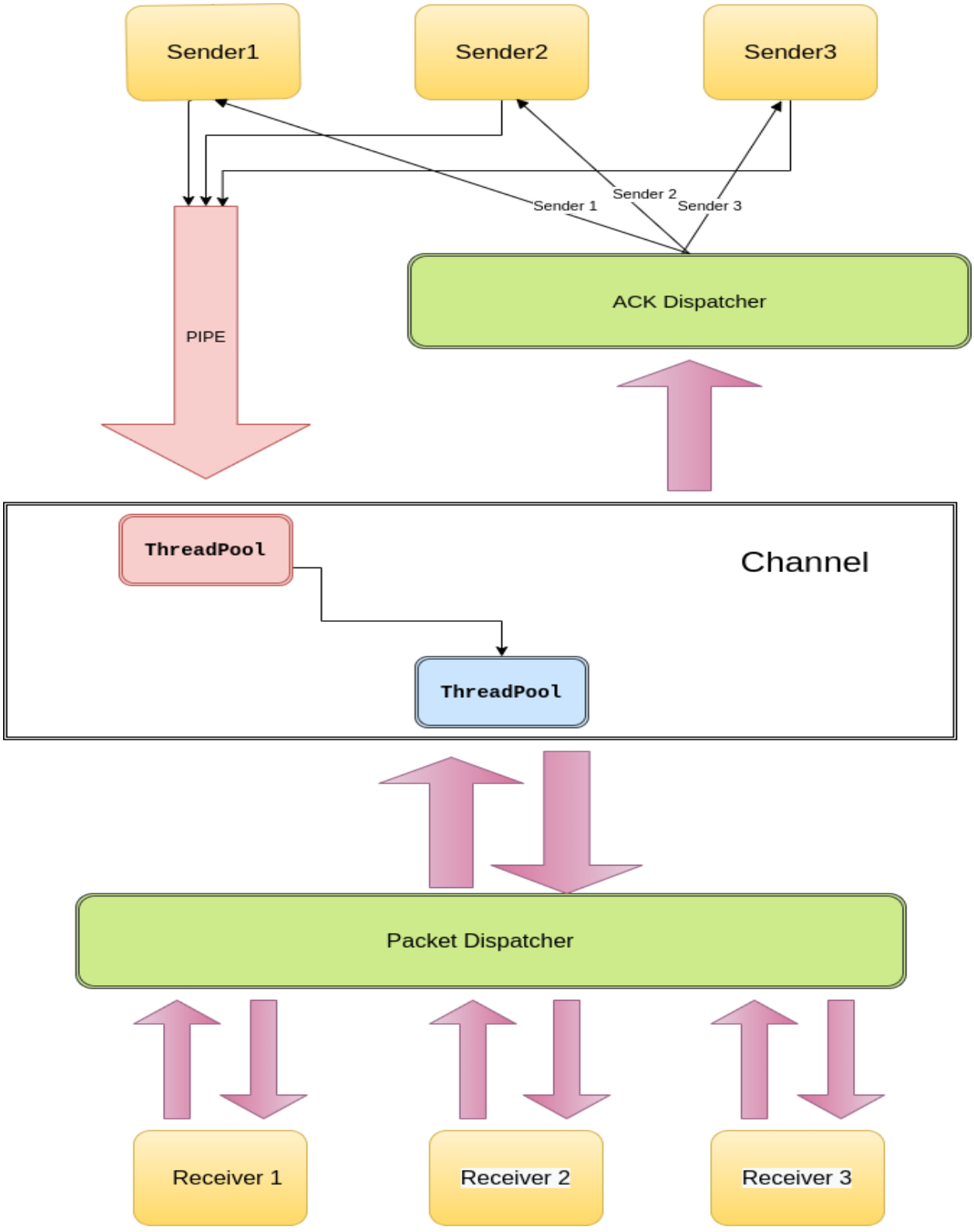


Fig1: A General Flow Diagram

Different CSMA methods and Implementation

P persistent CSMA:

This is the method that is used when channel has time-slots and that time-slot duration is equal to or greater than the maximum propagation delay time. When the station is ready to send the frames, it will sense the channel. If the channel found to be busy, the channel will wait for the next slot. If the channel found to be idle, it transmits the frame with probability p, thus for the left probability i.e. q which is equal to 1-p the station will wait for the beginning of the next time slot. In case, when the next slot is also found idle it will transmit or wait again with the probabilities p and q. This process is repeated until either the frame gets transmitted or another station has started transmitting,

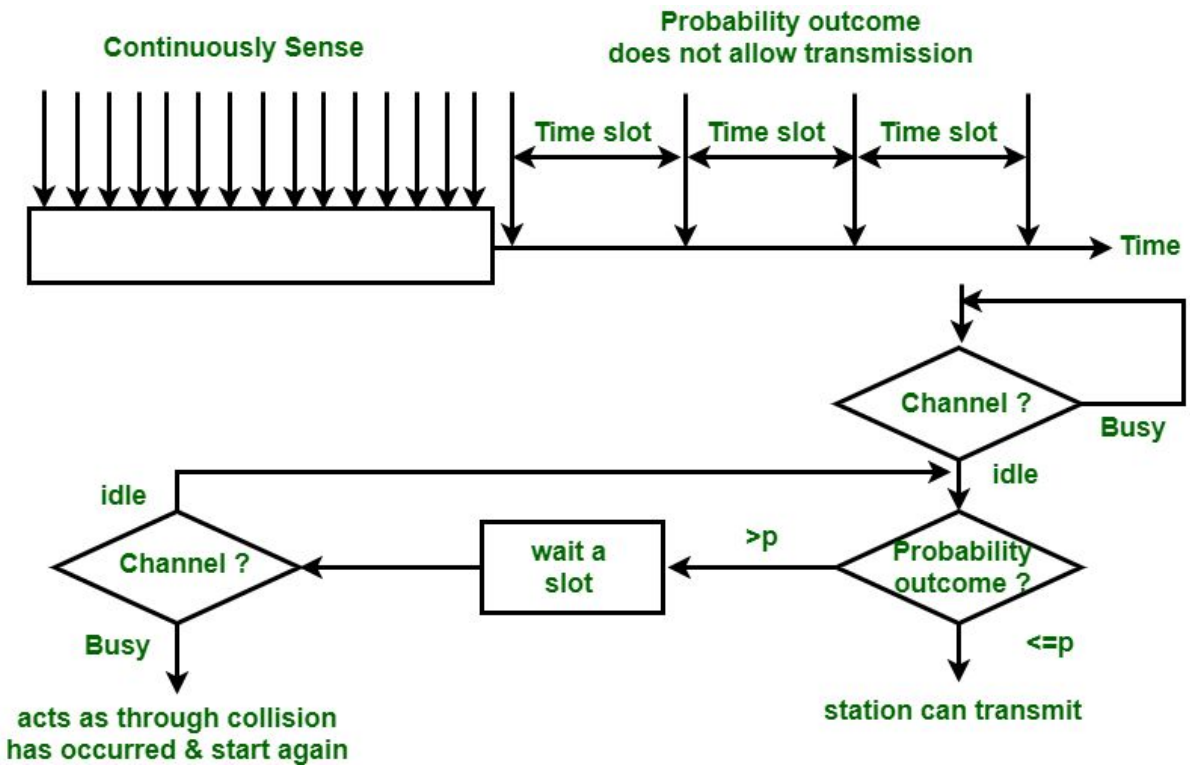
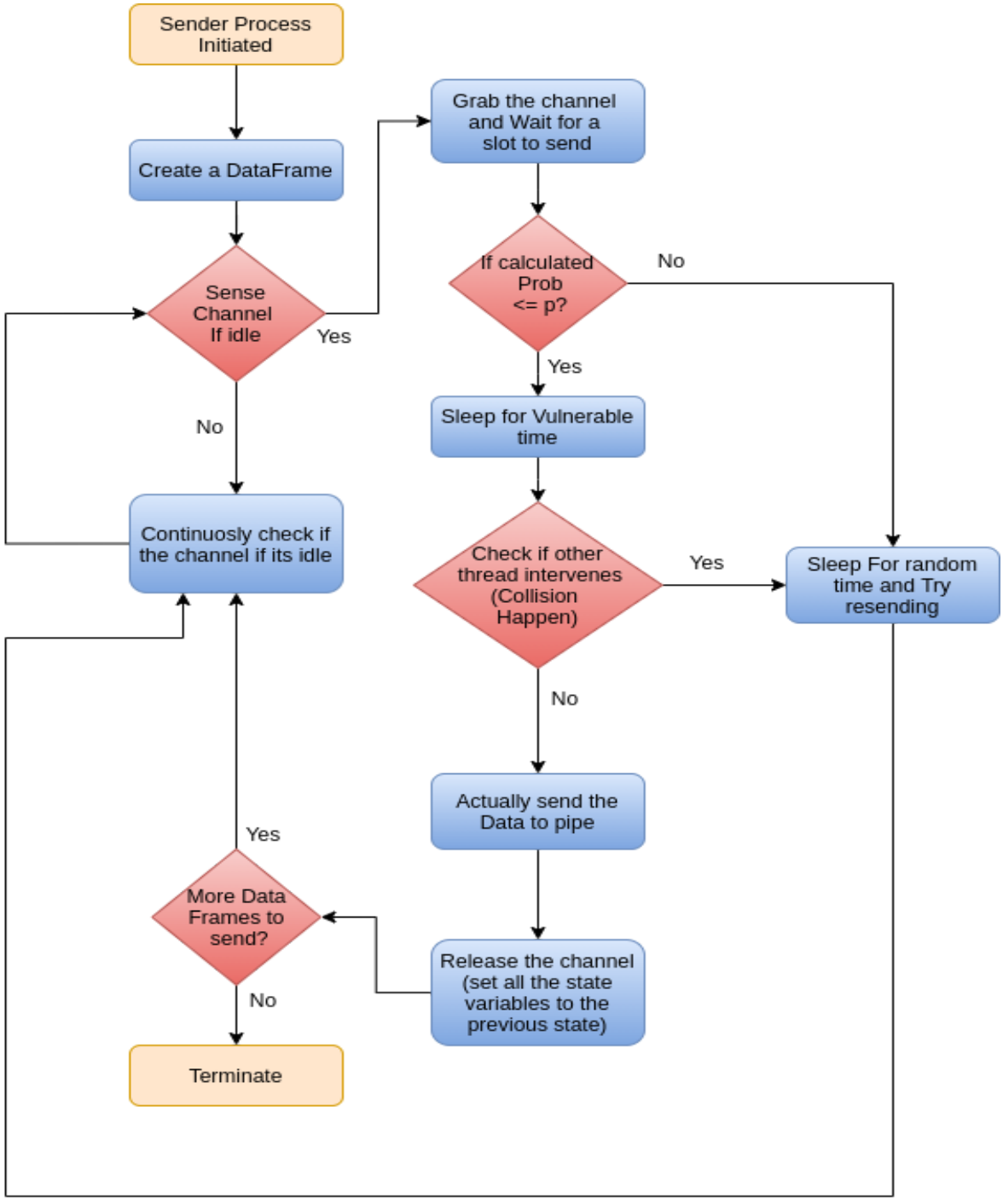


FIg:2 P persistent CSMA

Flowchart



[Implementation of P-persistent CSMA]

Non-persistent CSMA:

In this method, the station that has frames to send, only that station senses for the channel. In case of an idle channel, it will send frame immediately to that channel. In case when the channel is found busy, it will wait for the random time and again sense for the state of the station whether idle or busy. In this method, the station does not immediately sense for the channel for only the purpose of capturing it when it detects the end of the previous transmission. The main advantage of using this method is that it reduces the chances of collision. The problem with this is that it reduces the efficiency of the network

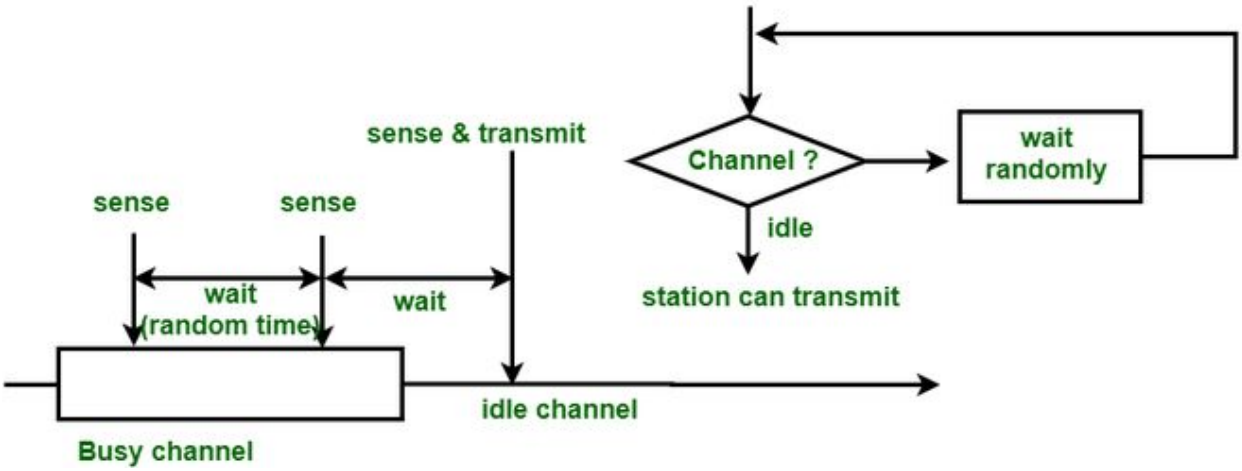
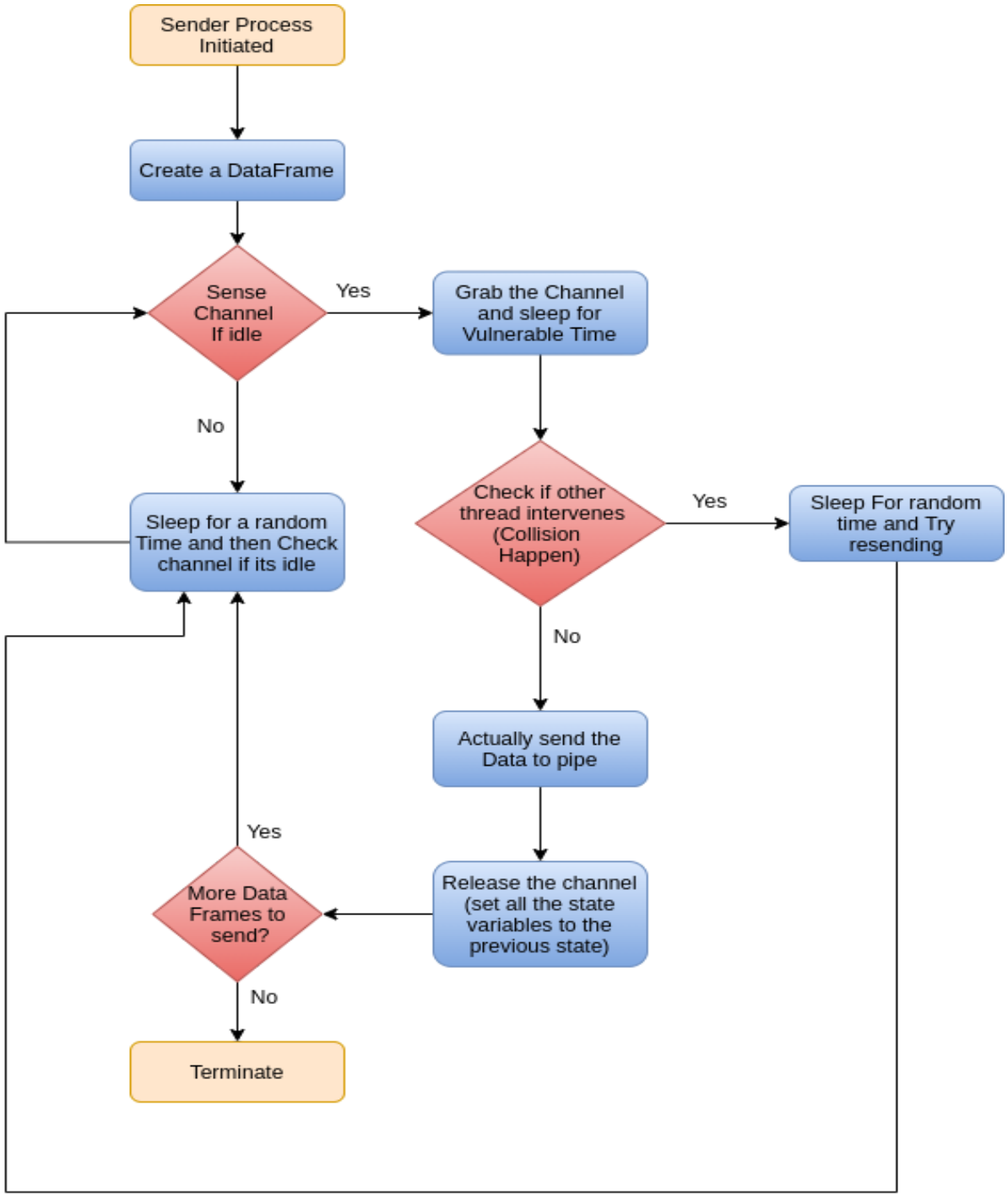


FIg:3 Non persistent CSMA

Flowchart



[Implementation of Non-persistent CSMA]

One-persistent CSMA:

This is the method that is used when channel has time-slots and that time-slot duration is equal to or greater than the maximum propagation delay time. When the station is ready to send the frames, it will sense the channel. If the channel found to be busy, the channel will wait for the next slot. If the channel found to be idle, it transmits the frame with probability p, thus for the left probability i.e. q which is equal to 1-p the station will wait for the beginning of the next time slot. In case, when the next slot is also found idle it will transmit or wait again with the probabilities p and q. This process is repeated until either the frame gets transmitted or another station has started transmitting.

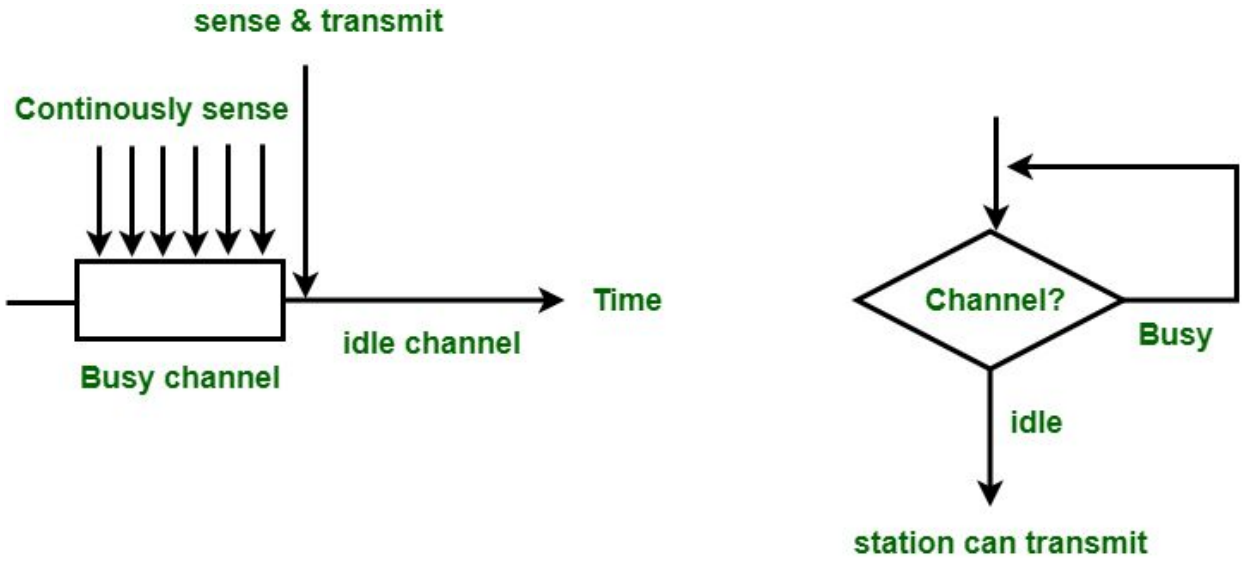
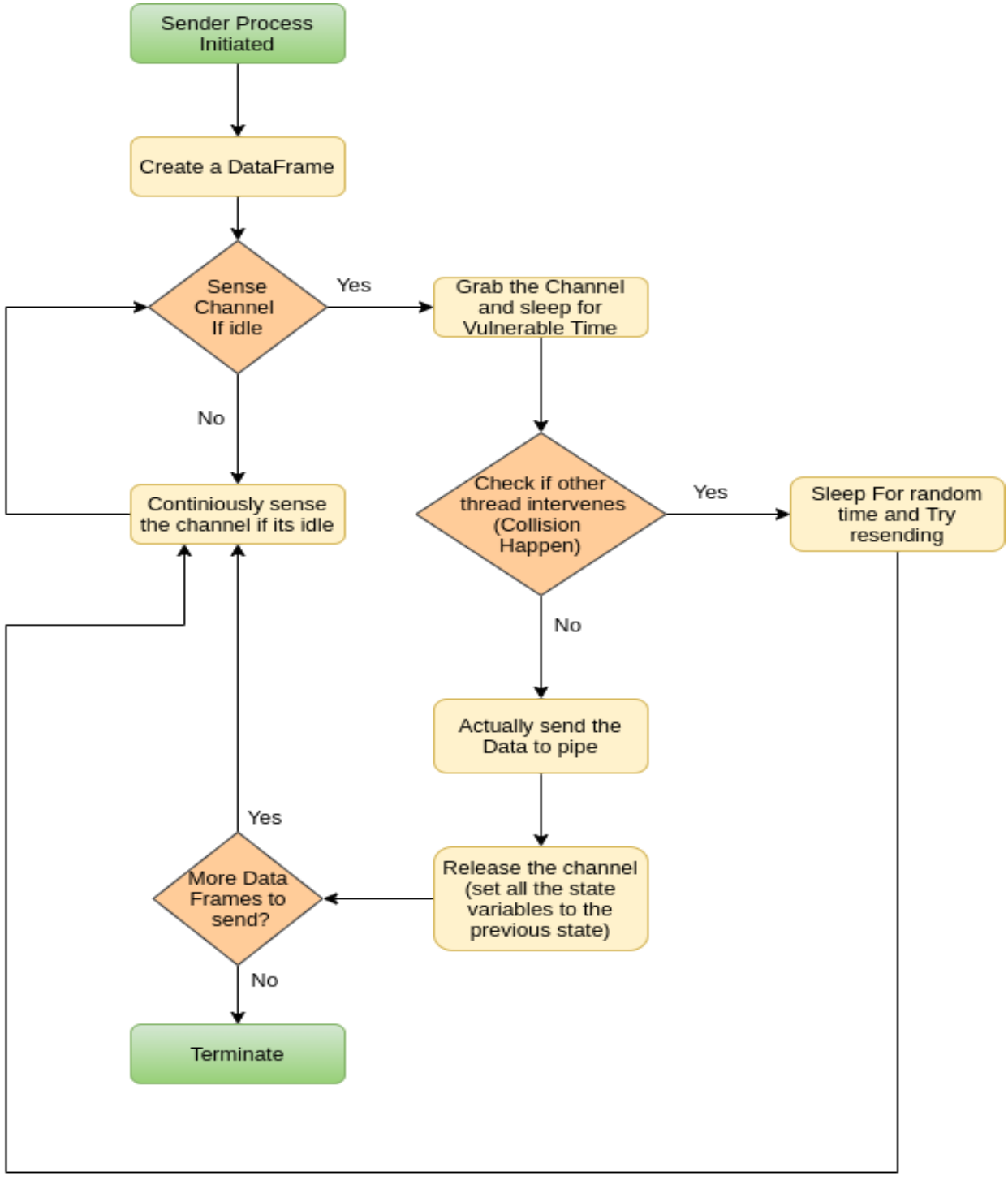


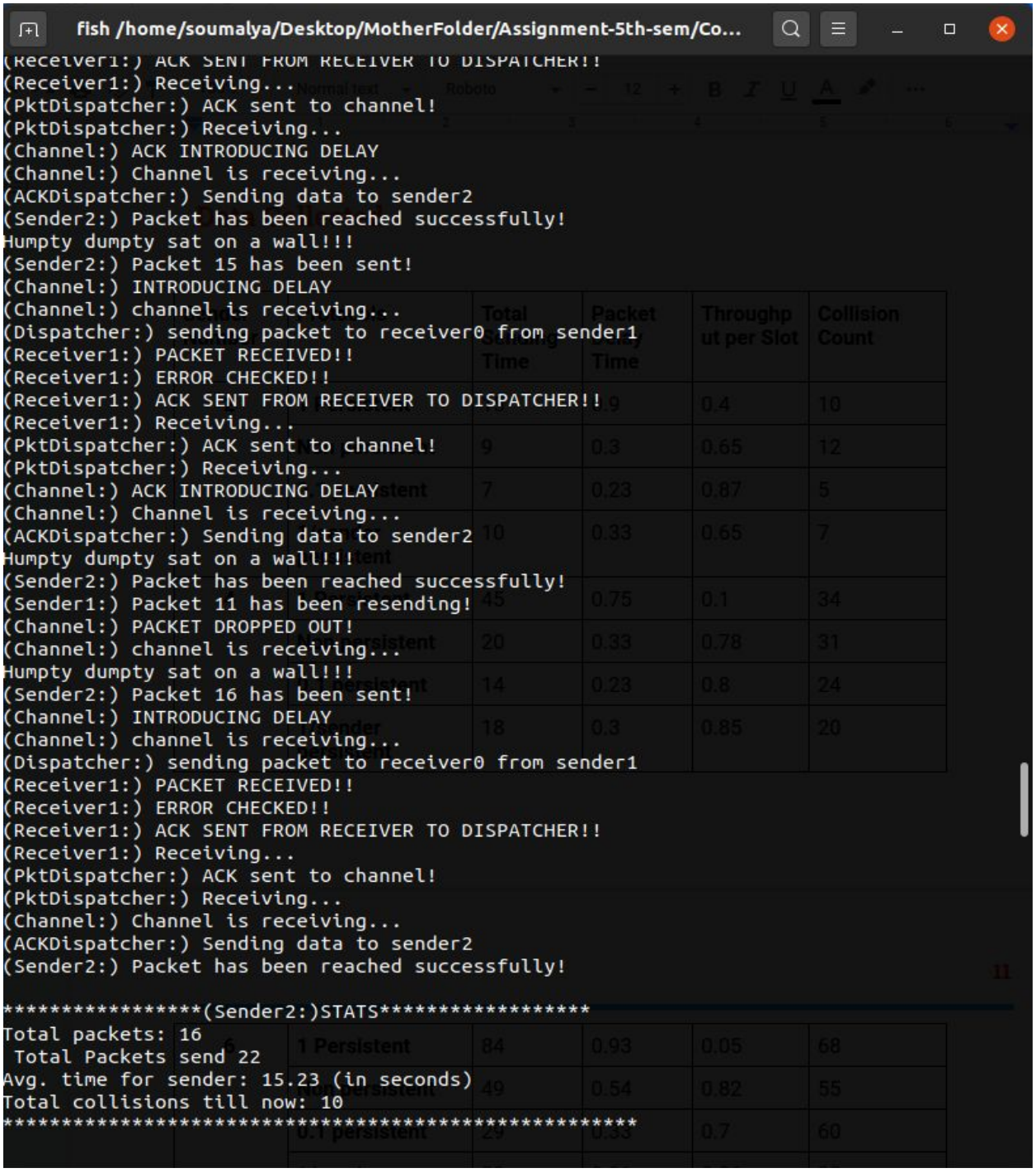
FIg:4 One persistent CSMA

Flowchart

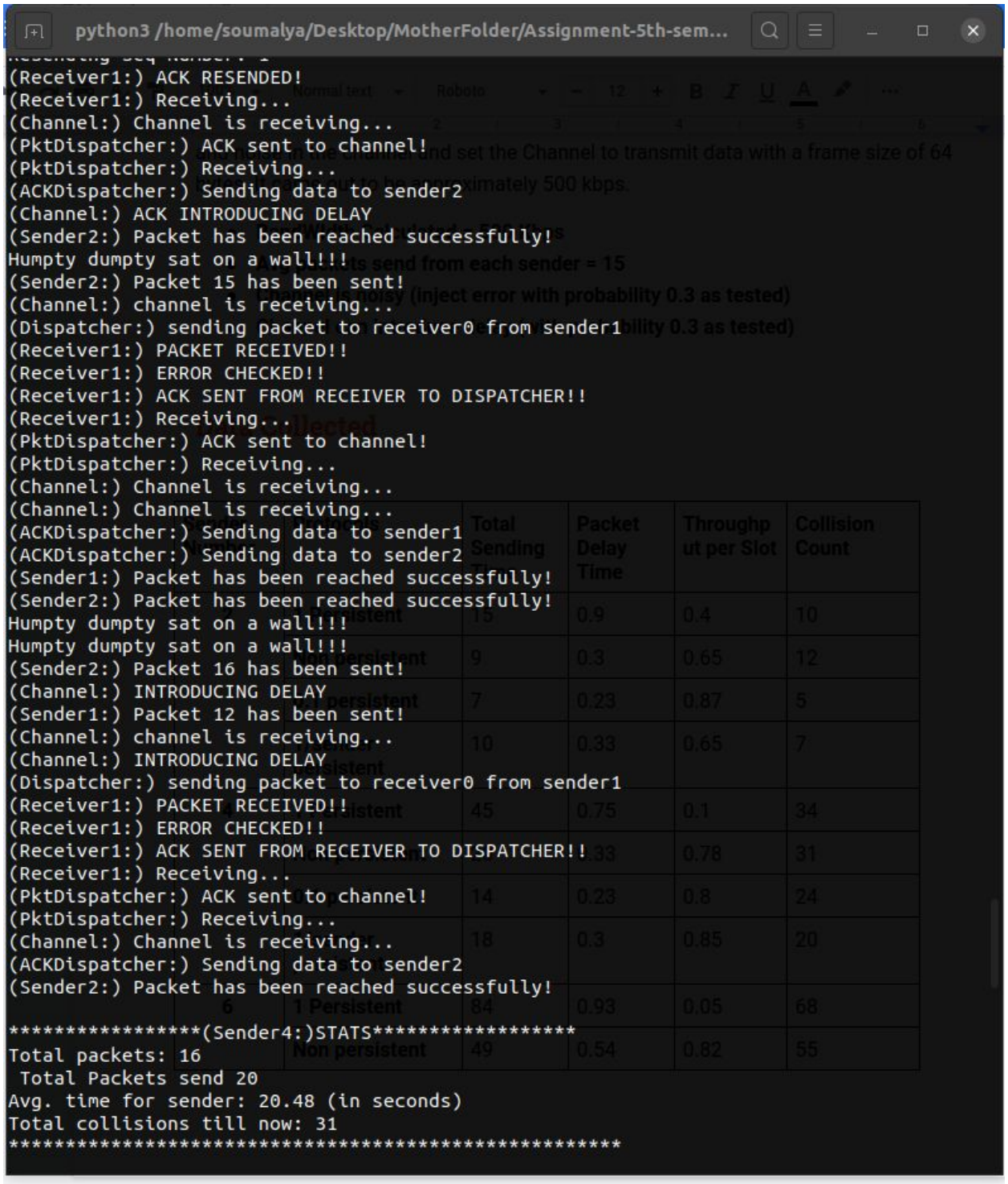


[Implementation of One-persistent CSMA]

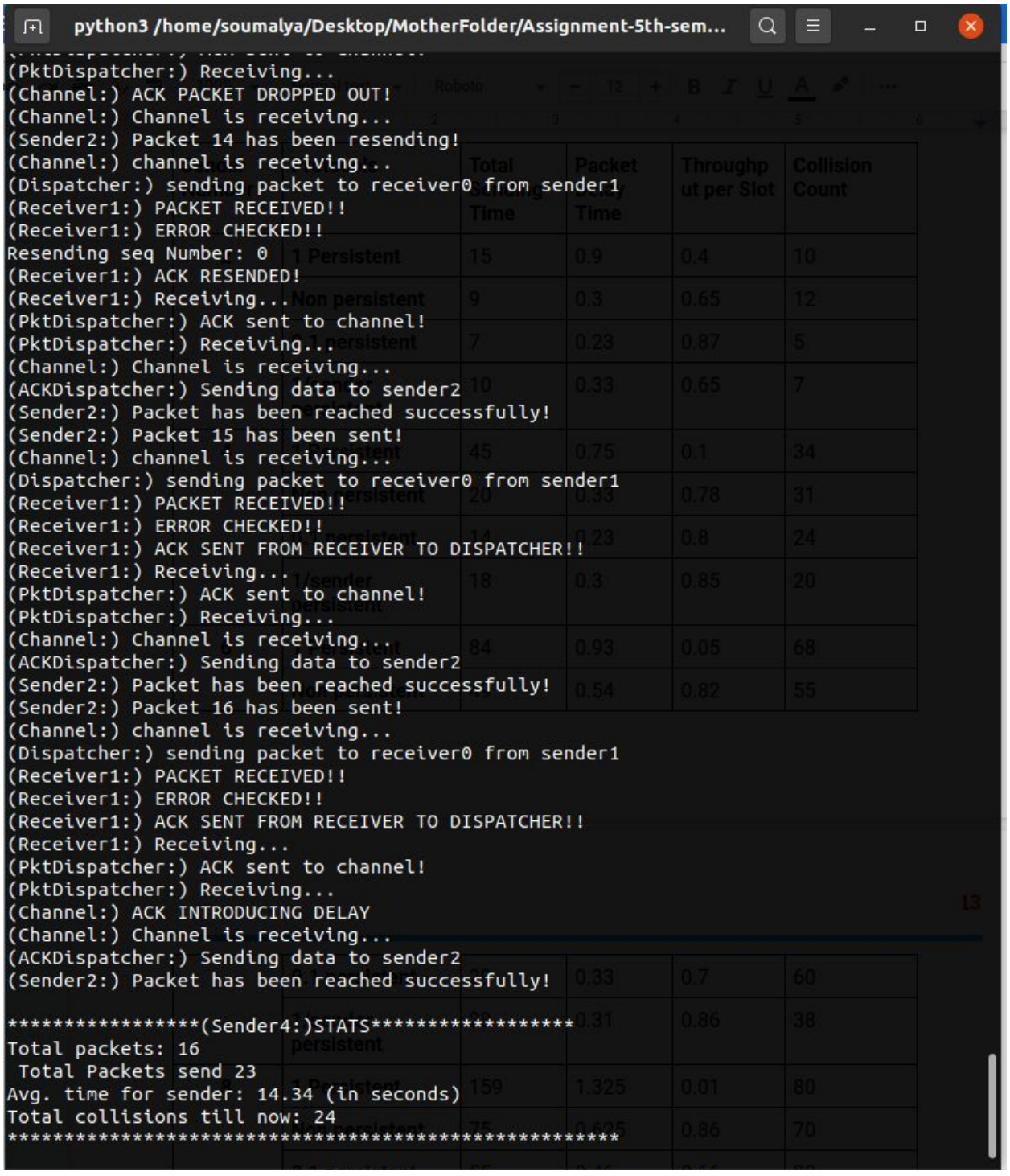
Output and Screenshots



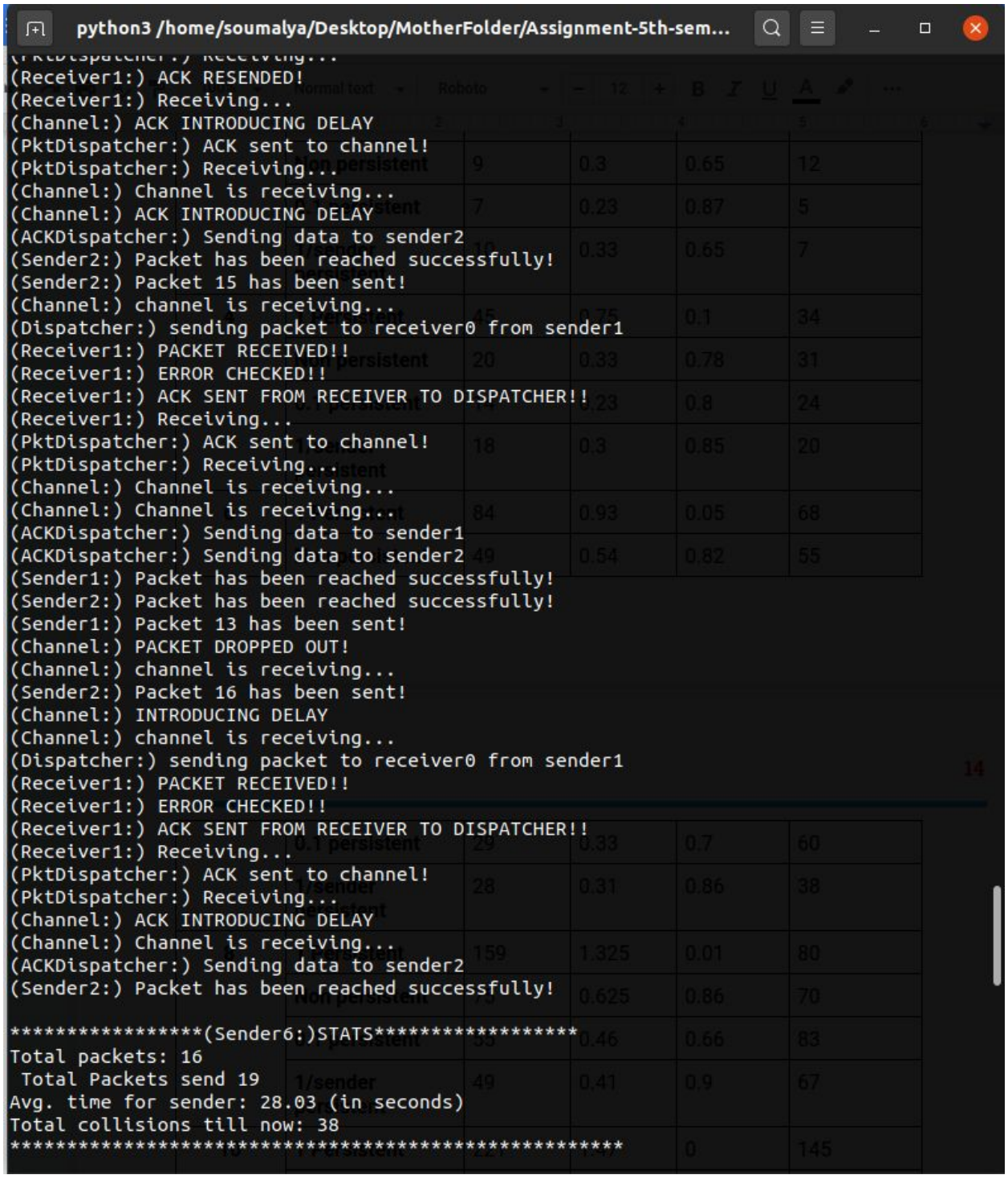
[One persistent CSMA with 2 senders]



[Non persistent CSMA with 4 senders]



[0.1 persistent CSMA with 4 senders]



[1/n persistent CSMA with 6 senders]

Analysis

To calculate and get an idea about the avg bandwidth, I turn off all the delay elements and noise in the channel and set the Channel to transmit data with a frame size of 64 bytes. It came out to be approximately 500 kbps.

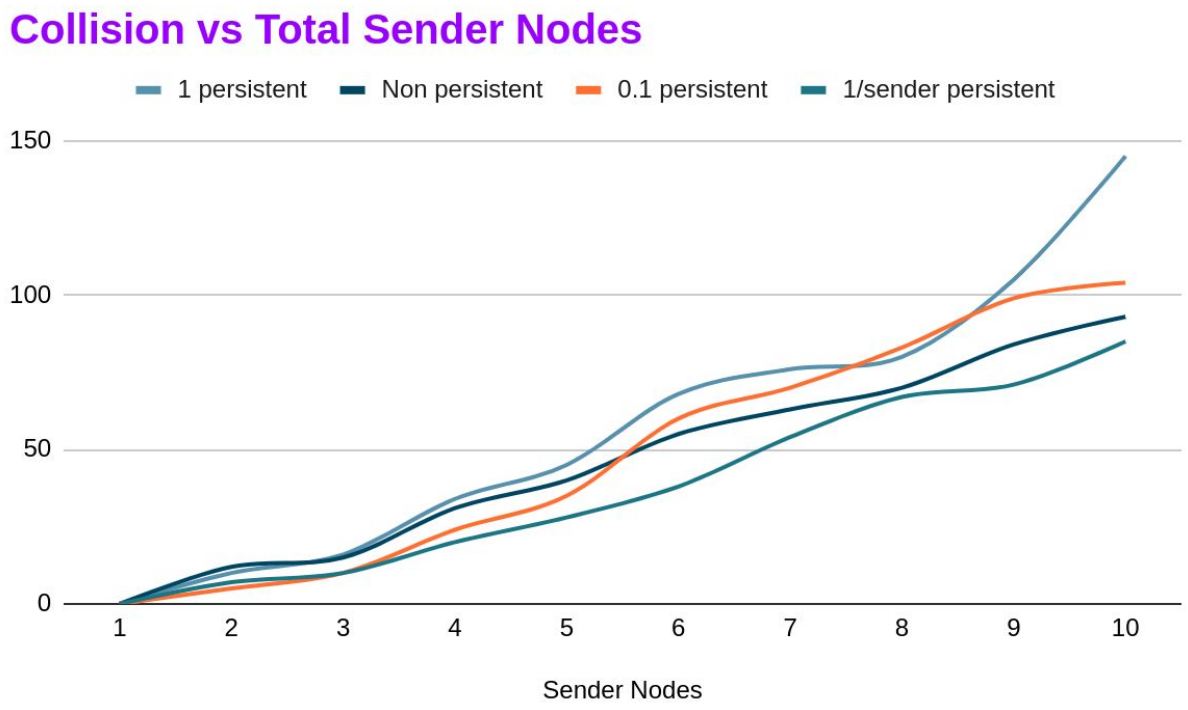
- **BandWIdth Calculated = 500 Kbps**
- **Avg packets send from each sender = 15**
- **Channel is noisy (inject error with probability 0.3 as tested)**
- **Channel can introduce delay (with probability 0.3 as tested)**

Data Collected

Sender Number	Protocols	Total Sending Time(s)	Packet Delay Time(s)	Throughput per Slot	Collision Count
2	**1 Persistent**	15	0.9	0.4	10
	Non persistent	9	0.3	0.65	12
	0.1 persistent	7	0.23	0.87	5
	1/sender persistent	10	0.33	0.65	7
4	**1 Persistent**	45	0.75	0.1	34
	Non persistent	20	0.33	0.78	31
	0.1 persistent	14	0.23	0.8	24
	1/sender persistent	18	0.3	0.85	20
6	**1 Persistent**	84	0.93	0.05	68
	Non persistent	49	0.54	0.82	55
	0.1 persistent	29	0.33	0.7	60

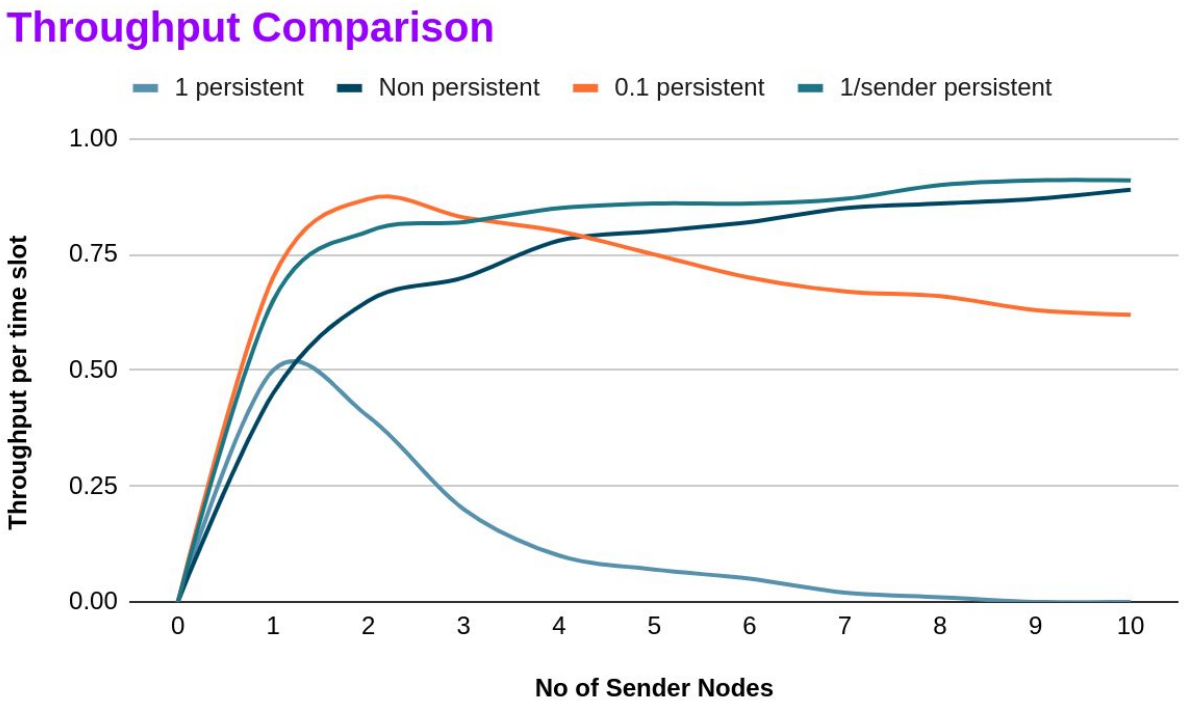
	1/sender persistent	28	0.31	0.86	38
8	**1 Persistent**	159	1.325	0.01	80
	Non persistent	75	0.625	0.86	70
	0.1 persistent	55	0.46	0.66	83
	1/sender persistent	49	0.41	0.9	67
10	**1 Persistent**	221	1.47	0	145
	Non persistent	95	0.63	0.89	93
	0.1 persistent	79	0.53	0.62	104
	1/sender persistent	72	0.48	0.91	85

Collision and Total Sender Nodes:



One persistent method shows the worst number of collisions and with a greater number of senders it just abruptly increases and almost shows exponential growth. But rest 3 of the graphs shows very interesting patterns. 0.1 persistent method actually performs best at the lower number of senders but 1/n persistent method outperformed it gradually in the long run. The non persistent method actually shows greater consistency throughout and managed to endup in between 1/n persistent method and 0.1 persistent method.

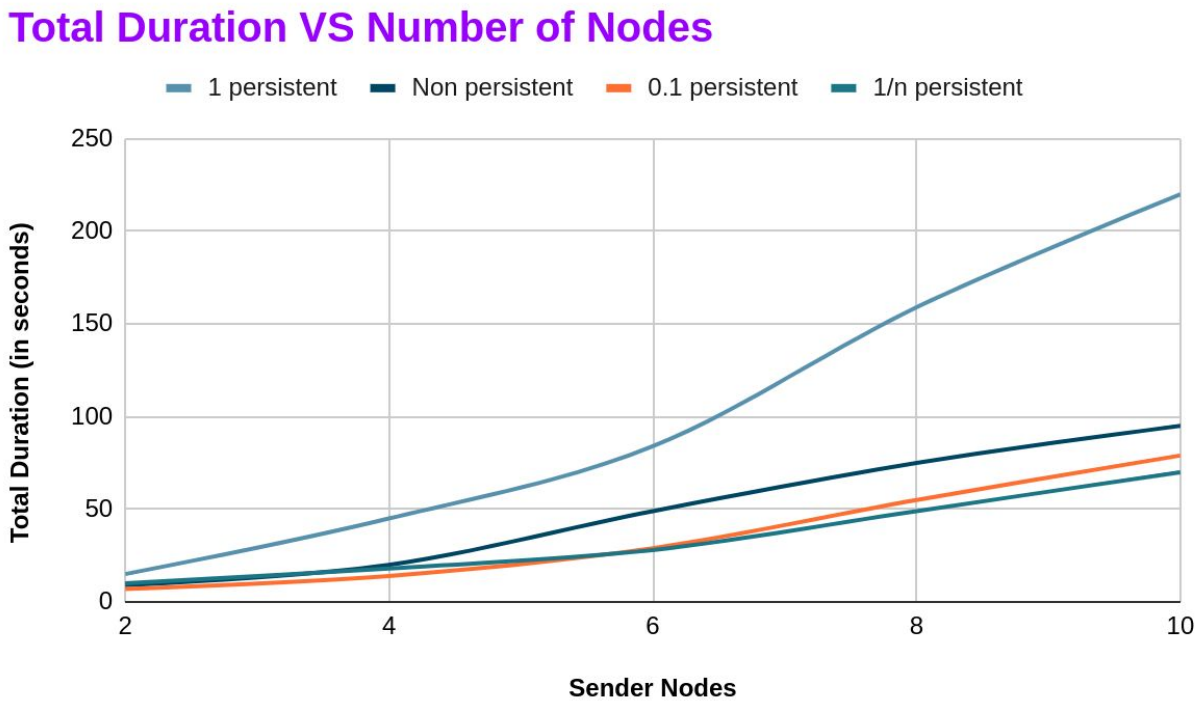
Comparison of Throughputs



The throughput of a network system is defined as the number of successful transmissions per frame time. The performance is inline with the collision chart. The one persistent method shows the worst possible throughput and quickly dies to zero as the number of senders increases. Interestingly enough non persistent method works pretty well at higher numbers of senders. Although 0.1 persistent method dies off quite

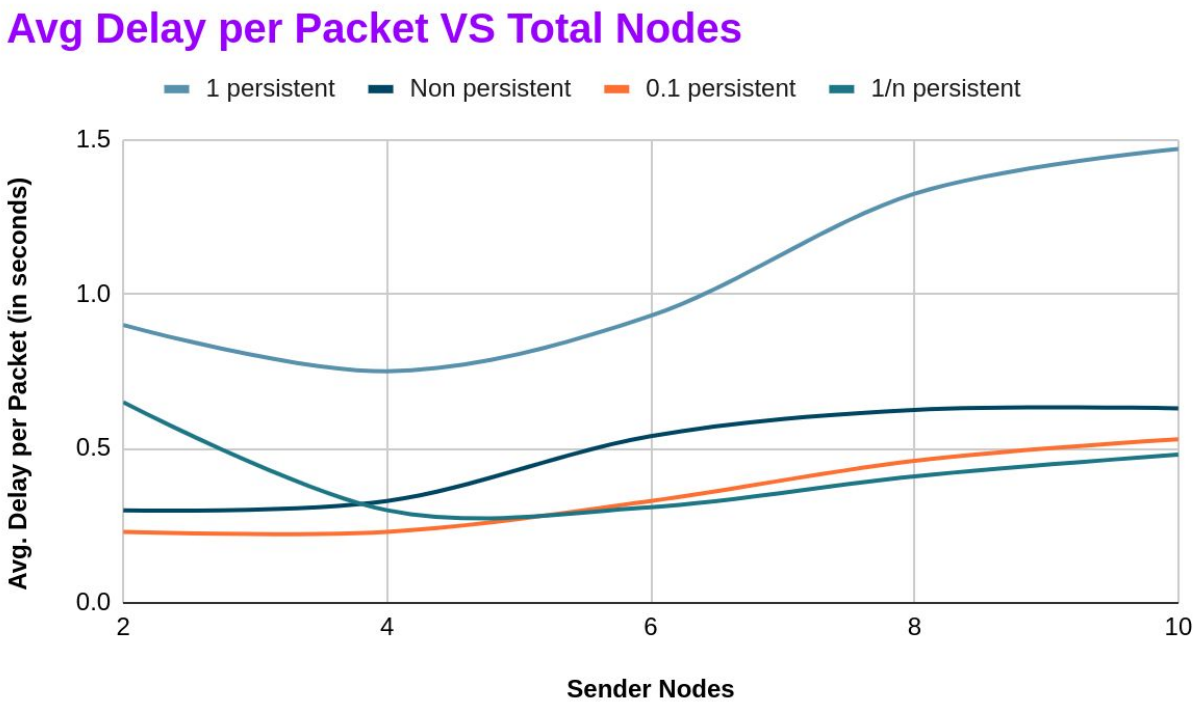
quickly and settles at 0.65 mark, it outperforms every single method at a lower number of nodes. The best performing of the bunch is 1/n persistent method which achieved nearly 0.9 throughput which is quite impressing.

Total Duration



This chart is also in par with the previous charts. One persistent method performed worst and showed exponential growth in duration. 0.1 persistent is best for lower number of nodes but outperformed by both 1/n persistent. Non-persistent method lies in between 1 persistent and 0.1 persistent method.

Average Delay and Total number of nodes



Again same story. One persistent method shows the greatest delay of the bunch. For lower numbers of nodes 0.1 persistent method is the best. In the lower region Non persistent method actually able to beat 1/n persistent method. But in the longer run, 1/n persistent method shows the best result and actually is the best of the bunch.

Conclusion

Considering overall performance, if the number of nodes in the network is relatively high then

1/n persistent > 0.1 persistent > Non persistent > one Persistent

Scopes of Improvement

- In a relatively small network the number of nodes falls somewhere between 10-30. So I should've considered more data points at a higher number of nodes.
- A more real life approach would have been to consider the receiver nodes also a part of this sensing algorithms. It would've provided more data points.
- I should've considered more data points regarding non persistent CSMA varying the range of random sleep.