Tarun Nagineni

ECS 152A

ECS152A Project Report

Functioning:

In order to generate the graph the user must run the program with no argument. The lamda values chosen are 0.001, 0.002, 0.004, 0.008, 0.012, 0.016, 0.02, 0.024, 0.028. In order to get a single throughput value simply run following the prompt ”python ethernet-simulation.py 25 pp 0.001”.

Code summary:

The code had been visualized in simply two classes the node class and the server class

Node_process class:

Variable description:

Each node has an individual Id which can range from 1 to the max number of nodes specified by the user. All nodes contain the same arrival rate which will be used as the time interval between the arrival of consecutive packets. Each node contains a unique packet number, which is simply the total number of packets existing at any given point in the node. The next_slot_number variable specifies the earliest slot number that the node will transmit a packet on. This is simply the slot that the packet at the head of the node will transmit. The final variable is the retransmission_attempts. This is simply the number of times the node tries to transmit/retransmit the packet at the head of the node. It is important to keep in mind that this variable will only be used by the ‘lb’ and ‘beb’ retransmission policies as will be elaborated on shortly.

Process:

There are two cases to consider for the arrival of a new packet in a node: a packet already exists in the node, or the new packet is arriving in an empty node. In the case of other packets already existing in the node, it would mean that there already exists a packet at the head of the node. Thus, values for next_slot_number and retransmission_attempts are already set with reference to this packet. However, if the new packet arrives in an empty packet, it will form the new head of the node. Thus, the values of the variables need to be changed. Retransmission_attempts will continue to be set to the default value of 1, while the next_slot_number is simply the next whole second from the current time. The reason for this is that the server only wakes up at periodic seconds. After we have dealt with this, we increment the packet_number variable by 1.

Server Process:

Variables:

The server process class takes in a dictionary of nodes initialized by the main class. Current_slot keeps track of the current slot(in range from 1 to max number of slots as initialized by the SIM_TIME variable in the G class). Success_count,cllision_count and idle_count variables to keep track of the number of times each of them have occurred over the span of all the slots(probably 100,000).

Process:

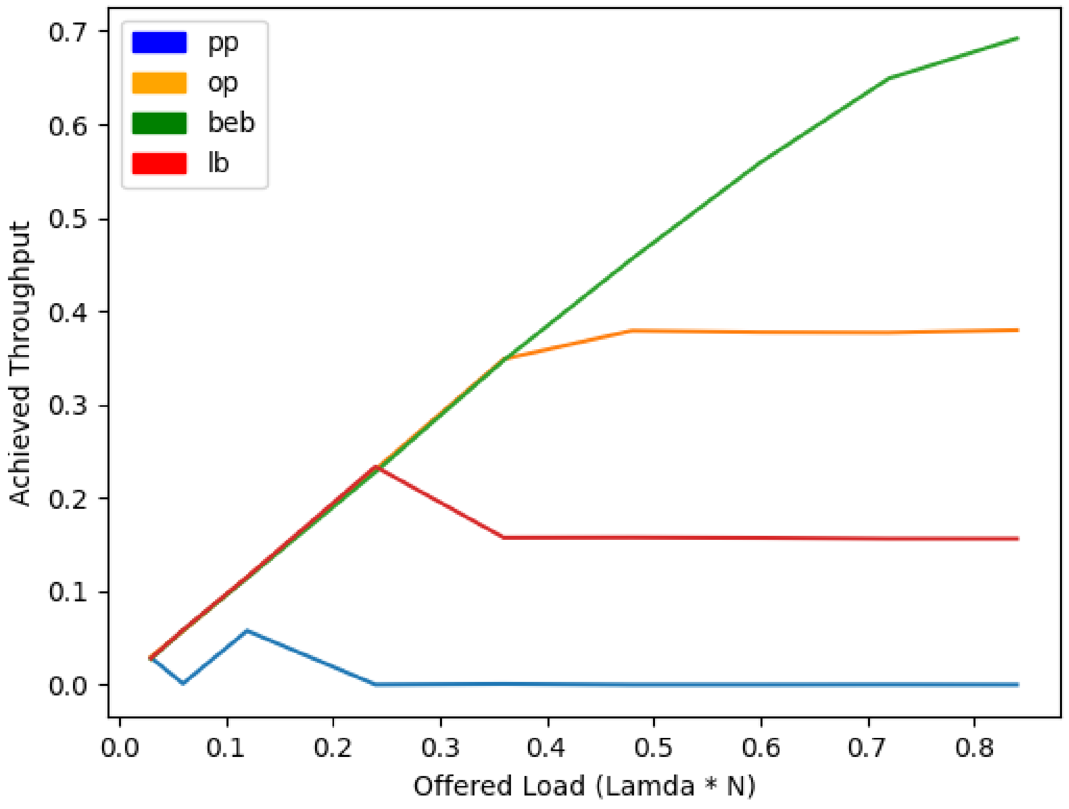
First, the server process tries to determine all the active nodes in the current slot. In order to do this we simply check how many nodes' next_slot_number(earliest transmission for the node) is the same as the current_slot_number. The index's of each of these active nodes will be appended to the active_nodes_index list. The length of this list would be able to determine if the slot was either successful, idle, or collision had occurred: If the length is equal to 1 the slot was successful, if it is equal to zero the slot is idle, if it is more than 1 then collision had occurred.

For dealing with successful and idle slots, the process is the same regardless of the retransmission policy. If the slot is successful we decrement the packet_number of the one packet in the active_nodes_array. And then check if there is another packet in the node(for becoming the new head of the node). If this is true, then set the next_slot_number of the node to the very next time the server will wake up. Finally, we increment the success_count of the server class. If the slot is idle, we simply increment the idle_count.

Things a little more complicated when there is a collision. Each node in the active node class must be given a new next_slot_number. This would depend on the retransmission policy. Finally, we increment collision count.

Main class:

In here we calculate the throughput by taking the success_count , the total total number of slots, and divide in order to get throughput. Depending on the transmission policy, we will append each of these throughputs to its own unique list(this will give us 4 lists in total: throughputList_pp, throughputList_op,throughputList_lb,throughputList_beb). We will use these lists to finally plot on the graph.


pp
op
beb
lb
Achieved Throughput
Offered Load (Lamda * N)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8