CS 152A/EEC173A : Project 2

Simulation Analysis of an IEEE 802.x Based Network

Leon Hakimi: lahakimi@ucdavis.edu
Peter Nelson: phnelson@ucdavis.edu
Yang Ye: yaye@ucdavis.edu
Spring 2020

Project Logic:

Our attempt at the implementation involved a basic initialization core, then a loop that pulls Event classes out of the global GEL queue. All of the variables used in the calculations for waiting times were the ones provided, 0.05ms for SIFS, 0.1ms for DIFS, and the minimum interval was not used, as this was a true discrete event machine in the works. The size of packets was determined by a negative exponential function with a very small lambda to draw out a large enough curve to cover the entire range of 0 to 1544 and included a modulus operation to keep everything within these bounds. This was originally kept in chronological order via a priority queue, but was discovered that this would require a change of implementation to provide another required feature.

The countdown timers stopping for packets waiting to enter the channel were accomplished with two functions, FreezeTimers and UnfreezeTimers. These kept track of the amount of time the channel was occupied by another packet, and then incremented the stop_time of events in the GEL that were waiting for access.

Collisions were handled by a counter that kept track of the number of packets in the channel. If multiple packets were occupying the channel in an instant in time, they were allowed to complete, but a function would iterate through the GEL and make any event with its packet in the broadcasting state be labeled as collided, where it would be dropped after the event reached it’s stop time.

Code and Description of Code

There is a brief initialization function that takes in a lambda value, as well as a value for the number of host users through the command prompt. Then, a vector of hosts is populated, and an Arrival event is created for each of these with different arrival times and stored in the GEL. The program then continues into its main loop, where it removes the event with the lowest stop_time and calls a sub function based on the category of Event it is.

Arrival events create a new packet with a random destination and place it in the queue of a random host, but it cannot be generated to be delivered from its home host. The buffer is checked, and if it is not occupied by any other packets, it begins the departure process. The departure time is calculated by adding the DIFS to the current time, and if the channel is currently occupied, a random value from the BinaryExponentialBackoff function is added as well. With the buffer occupied, the packet is simply pushed into the back of the buffer.

Departure events happen when the end_time is reached, but this value will be incremented whenever another packet occupies the channel, so it's expected to have increased. In a departure event, the FreezeTimers function is called to get the time interval oft the channels occupancy, and a check is done to verify that only one packet is in the channel. Otherwise a MarkAllCollision function is called to toggle a boolean value in the Event so it will be dropped upon arriving at the destination host.

Sent events are activated when a packet reaches its destination, and sets up the process of sending an ACK back to the source host. Similarly to departure events, it checks the buffer capacity of the host and cuts in line to prepare for departure after a SIFS interval. And if the channel is occupied, an added BinaryExponentialBackoff. It also creates an AckTimer event, and if the event is reached before the ACK event is completed, begins the cycle all over again with the same packet and the same tracked values for queue delay and transmission time delay.

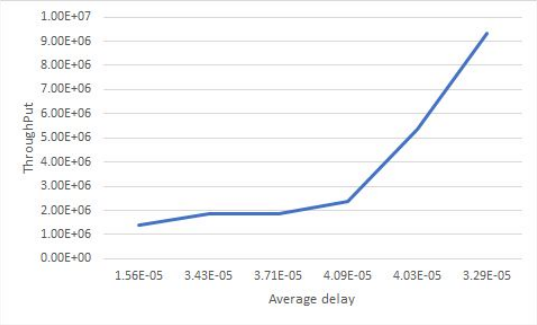
It is built to loop for a large number of events, but we have had issues with the pointers that unfortunately prevented us from producing a fully fledged working build.

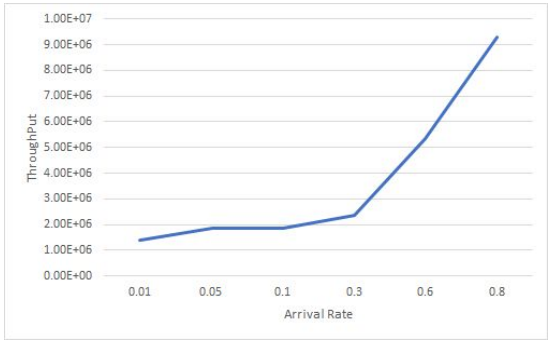
Data Analysis:

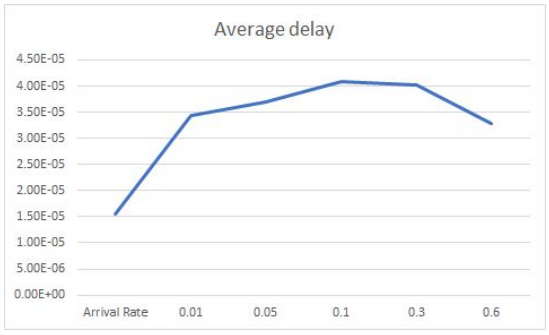
```
N = 10, arrivalRate = 0.05 :
throughPut: 1.87e+06
average delay:3.42781e-05

N = 10, arrivalRate = 0.1 :
throughPut: 1.87e+06
average delay:3.70535e-05

N = 10, arrivalRate = 0.3 :
throughPut: 2.38333e+06
average delay:4.08923e-05

N = 10, arrivalRate = 0.6 :
throughPut: 5.35333e+06
average delay:4.03207e-05

N = 10, arrivalRate = 0.8 :
throughPut: 9.31333e+06
average delay:3.28701e-05

N = 50, arrivalRate = 0.01 :
throughPut: 5.5e+06
average delay:1.54909e-05

N = 50, arrivalRate = 0.05 :
throughPut: 9.09333e+06
average delay:3.52951e-05

N = 50, arrivalRate = 0.1 :
throughPut: 9.68e+06
average delay:3.94783e-05

N = 50, arrivalRate = 0.3 :
throughPut: 1.199e+07
average delay:4.65938e-05

N = 50, arrivalRate = 0.6 :
throughPut: 7.62667e+06
average delay:4.43234e-05

N = 50, arrivalRate = 0.8 :
throughPut: 7.62667e+06
average delay:4.43234e-05
```

In these experiments, we tried different arrivalRate, different lambda, and recorded the average delay and the throughput.

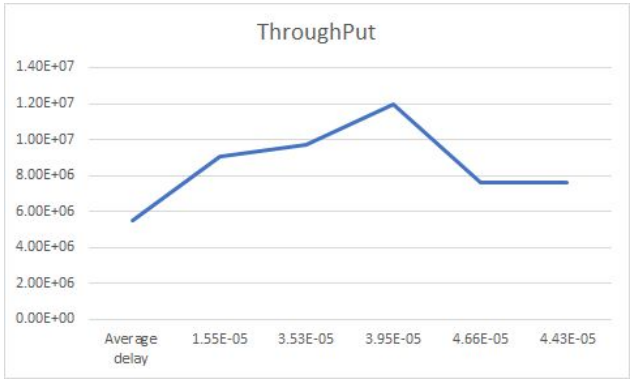
When Lambda = 10

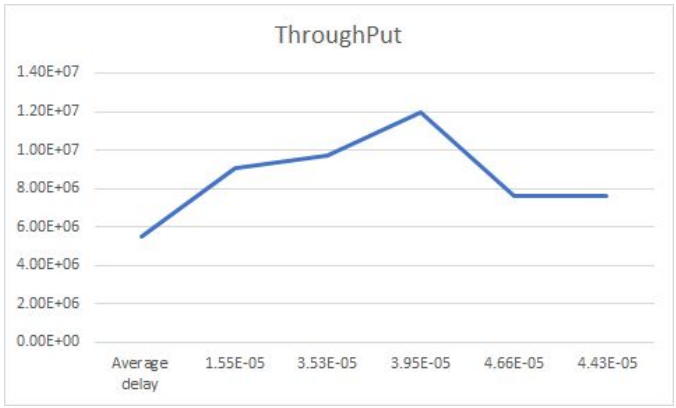


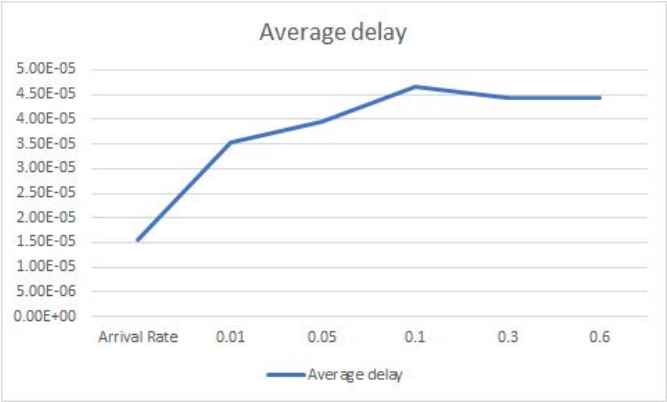




When Lambda = 50







By observing the graph, we found that there is a strongly positive correlation between Average delay and arrivalRate. The higher the arrivalRate, the higher delay there will be, but when the arrival is becoming more close to 1, the delay curve will flatter.

When the lambda is 10, the average delay increases when the throughput increases.
When the lambda is 50, the average delay increases when throughput increases, but then when the throughput keeps increasing. When the average delay is too high, the throughput will drop.

Contribution:

Leon Hakimi(lahakimi@ucdavis.edu)

Contribution: Writing Code/Debugging

Insight: I felt like I understood the logic of the CSMA/CA protocol but it was much harder to actually implement the simulation

Yang Ye(Yaye@ucdavis.edu):

Contribution: Writing code, Testing the code for different parameters, Testing the code on bash shell, wrote a part of the report.

Insight: By looking at the date, I get a better understanding about how the WLAN works in general situations.

Peter Nelson(phnelson@ucdavis.edu)

Contribution: Writing code, debugging, Program Logic and Explanation of Code Sections.

Insight: This was a fun project to go through, my only real regret is not being able to put out a polished version to get correct results. It was initially very difficult to grasp how this would be built so it could function, and although this version had the unfortunate issues of pointer problems, I feel I did the best that I could do with the time I was able to spend on it.