**Introduction:**

This project was about simulating the behavior of routers communicating in two different setups using two different protocols for both for scenario A the nodes will be in the same collision domain and able to “see” each other. The CSMCA protocol will be simulated for this scenario as well as CSMCA with virtual collision sensing to compare their performance. For scenario B the nodes will be in a hidden terminal set up, in other words the nodes will not be able to “see” each other directly. This scenario will also be simulated using CSMCA and virtual collision sensing. Please see the figure below for example of scenario A and scenario B.

A picture containing hanging, photo, table, skiing

Description automatically generated

Jessmer was responsible for writing the simulation for scenario A with CSMAC and scenario B with Virtual sensing as well as the code to generate the Poisson Traffic.

Spencer was responsible for writing the simulation for scenario A with virtual sensing, and scenario B with CSMAC as well as collating the resulting data.

The report was written jointly.

**Description:**

In order to perform the required simulations, we have written different kinds of functions and algorithms, as well as incorporating different data structures. We chose to use C++ as the programming language to write the code for the simulations. We created separate functions for generating uniform distributed numbers, exponentially distributed numbers, slot times, and the arrival slots. We used the arrival slots as the point where packets are sent, instead of using the slot times. These arrival slots are integer values stored in a vector. When generating random numbers, we used the rand() function, and used the srand() for the seeding number. In addition to the four random number generator functions, we also wrote four (4) algorithms that computes and simulates the behaviors of packet transmissions. These four functions are: simulate\_SCD\_CSMACA(), simulate\_SCD\_CSMACA\_VCS(), simulate\_HT\_CSMACA(), and simulate\_HT\_CSMACA\_VCS(). The simulate\_SCD\_CSMACA() simulates the scenario for Single Collision Domain (SCD) without enabling the Virtual Carrier Sensing (VCS). The simulate\_SCD\_CSMACA()\_VCS() simulates the scenario for SCD with VCS enabled. The simulate\_HT\_CSMACA() simulates the scenario for Hidden Terminal (HT) with VCS disabled. The simulate\_HT\_CSMACA()\_VCS() simulates the scenario for HT with VCS enabled. Each of the four algorithms return an unordered map that stores the throughput values for nodes A and C, the number of collisions for A and C, and the fair index value of the simulation. These functions and algorithms are inside a “for” loop that iterates all the LAMBDA values. Each lambda value is passed into the random number generator for each iteration. After the function returns, the results are then printed in the console.

**Results:**

1. **Throughput**
   1. **Node A:**
   2. **Node C:**

From the above graphs we can see that node A & C’s throughputs for the same collision domain using CSMCA (green) and the throughput for the hidden terminal using virtual collision sensing (purple) are very similar with the throughput for the hidden terminal virtual collision sensing being just a bit less than that of the CSMAC throughput at high lambda’s. This is congruent with our predictions since the number of collisions should be similar between the two the throughput will be more influenced by how many frames are used to send a packet. Since the virtual sensing model uses more frames before sending the data it creates a larger window in which the transmitting node will occupy the channel(figure 1). This is mitigated at lower lambdas as there is less chance of a node wanting to transmit while the channel is occupied as well as there being more empty frames available. At higher lambdas the probability of a node wanting to transmit while the channel is occupied is increased and there are less empty frames available to mitigate this. Since the virtual collision sensing method utilized more frames to send the same amount of data as the CSMCA method the throughput will be lower than the CSMCA at high lambdas and the difference will increase as lambda gets larger.

This should have held true for when the virtual sensing method was used in the same collision domain (yellow) but instead we see that the throughput is the same at lower lambdas, greater at lambda equals one thousand, and much less at lambda equals two thousand. From this we can assume that there is an error in the simulation code as the difference between the two should not change so radically at lambda equals two thousand especially since the collision counts between the two (green & yellow) support that yellow should have a higher throughput at lambda equals two thousand as it experiences less collisions.

Diagram

Description automatically generated

Figure 1

1. **Collisions**
   1. **Node A:**
   2. **Node C:**

The above graphs showing collisions vs lambda mostly behave as expected with the exception of the hidden terminal virtual collision detection (purple). It was expected that the collision count of the hidden terminal scenario would closely mirror the same collision domain values(green & yellow) when virtual collision sensing was used. From the graph it can plainly be seen that the number of collisions experienced by the hidden terminal virtual collision detection scenario (purple) was far greater than expected. Upon closer inspection this seems to be the result of an error in the simulation that was not detected when testing with smaller data sets, as at the lower lambda values the collision count was equal to the successful packet count and at higher lambdas the collision count was greater than the successful packet count. It was expected that the collision number would be much closer to that of the same collision domain CSMAC (green) scenario. This is because the VCS also uses DIFFS followed by backoff and then sends an RTS, this will set the NAV(RTS) high at the receiving terminal indicating to all other nodes not to send, in effect making it so that the hidden terminals can “see” when each other are transmitting and can wait until the channel is clear indicated by NAV(RTS) going low before attempting to send. Coupled with this the transmitting node will not send the data packet until it has received a CTS signal from the receiver. These steps help to greatly reduces collisions vs hidden terminal without virtual collision sensing(blue).

The collision values for the hidden terminal without collision sensing(blue) behaved as expected. High numbers of collisions as nodes A and C had no way of knowing if the channel was occupied and would often transmit while the other was also transmitting. As lambda increased the number of collisions did until it reached a saturation point around lambda = 1000 where the combined number of collisions between A and C stays around nine to ten thousand. At this point the growth of the number of collisions slowed considerably. At this point the number of collisions can not grow much larger as most of the frames are occupied by transmissions that are colliding already.

The same collision domain CSMAC (green) and the same collision domain with virtual sensing (yellow) are very close to each other and within expectations. The only discrepancy is that we assumed that these values would be equal or near equal as the virtual sensing model (yellow) also uses DIFS and a backoff like the CSMCA model (figure 1). From the data we see that the lower collision rate of the virtual sensing method is in keeping with its apparent higher throughput seen in 1.a and 1.b up until lambda equals two thousand where the throughput takes a precipitous drop.

1. **Fairness index:**

The fairness index for all the methods of transmission tended to stay close to 1 with the greatest difference between successful packets sent by A and successful packets sent by C being 6%. From the data and graphs the methods and scenarios used do not appear to favor one side over the other and after running the simulation multiple times we found that the fairness values for all lambdas stay very close to 1.0