Project # 1. Simulating the Distributed Coordination Function (DCF) of 802.11

1 Preliminaries

- Read the project description very carefully.
- You can form a group of up to two people. If you do not have a partner please post on D2L to find one. Utilize available online resources (Zoom, github) for collaboration to make the experience smoother.
- You are free to use a programming language of your choice. Matlab, C/C++, or Python should all work for this project.

2 Project Description

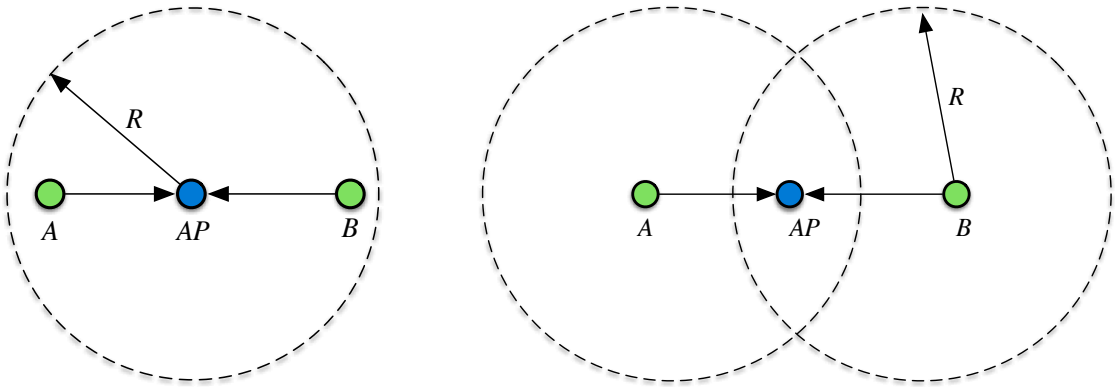
You are to study the performance of multiple access protocols in a wireless setting via event-driven simulation. Consider the network topologies shown in Figure 1. The circles denote the communication range R of each station. We are interested in the following two scenarios:

A. Single Collision Domain, Figure 1(a): Stations A, B, and the access point AP are within the same collision domain (any transmission is received by all). Stations A and B generate traffic that is directed to the AP. The traffic pattern at each station follows a Poisson arrival process with an average arrival rate of λ frames/sec (more on this in the Appendix).

B. Hidden Terminals, Figure 1(b): Stations A, and B are in different collision domains and therefore cannot hear each other's transmissions. Stations A and B generate traffic that is directed to the AP. The traffic pattern at each station follows a Poisson arrival process with an average arrival rate of λ frames/sec.

For each scenario, implement the following multiple access protocols.

1. *CSMA with Collision Avoidance (CSMA/CA) according to the 802.11 DCF function.*
   (a) Time is slotted and transmissions can only start at the beginning of a slot.
   (b) A station Tx is ready to transmit when a frame is written by an application in the upper layers of the network stack. The times that a frame is written to the station's transmission buffer are determined by a Poisson arrival process with rate λ. The Poisson processes at each transmitting station are independent. For a frame at the head-of-line in the transmission buffer, the Tx selects a random backoff value $b \in [0, CW_0 - 1]$ slots. It then senses the channel for an initial period of DIFS slots.
   (c) If the channel becomes busy, the Tx freezes until the end of the transmission round (known from the respective Network Allocation Vector (NAV). When the NAV expires, the Tx repeats the access protocol starting from DIFS. When the channel becomes idle for a period of DIFS



(a) (b)

Figure 1: (a) Topology for parallel transmissions within the same collision domain, (b) topology for parallel transmissions when A and B are hidden terminals.

slots, Tx decrements his backoff counter b by one with every additional idle slot. If the channel becomes busy before the counter b reaches zero, the Tx freezes its backoff counter. The backoff counter resumes to count down after the channel has become idle again (which is determined by sensing for DIFS slots). When the counter reaches zero, Tx transmits its frame.

(d) If the frame is successfully received (no collision) by Rx, the station Rx replies with an ACK frame after SIFS slots. This completes the transmission round and the protocol repeats for the next transmission. For successive transmissions, the station has to sense for DIFS time before starting the countdown.

(e) If a collision occurs, the stations that collide double their contention window CW and repeat the backoff process. After k collisions, the backoff value is selected from $[0, 2^k CW_0 - 1]$. The CW value cannot exceed threshold $CW_{\max}$.

2. *CSMA/CA with virtual carrier sensing enabled:* RTS and CTS frames are exchanged before the transmission of a frame. If RTS transmissions collide, stations invoke the exponential backoff mechanism outlined in Step 1(d). Otherwise, stations that overhear an RTS/CTS message defer from transmission for the time indicated in the NAV vector.

3 Simulation parameters

Parameter	Value	Parameter	Value
Data frame size	1,500 bytes	ACK, RTS, CTS size	3 slots
Slot duration	10 μs	DIFS duration	4 slots
SIFS duration	1 slot	Bandwidth	10 Mbps
CW_0	8 slots	$CW_{\max}$	512 slots
$\lambda_A = \lambda_C$	$\{100, 200, 300, 500, 800, 1000\}$ frames/sec	Simulation time	10 sec

4 Performance Metrics

Evaluate the protocol performance with respect to the following metrics:

Throughput T: The individual station's throughput as a function of λ.

Collisions N: The number of collisions (data and RTS/CTS collisions) as a function of λ. Perceived by each station. For topology (a), you only need to count the collisions at the AP. For topology (b), collisions measured by A and B could differ, so they need to be accounted for independently.

Fairness Index FI: The fraction of time that A transmits over the time that B transmits, as a function of λ. This can simply be the ratio of transmissions (successful and collisions) made by A over the transmissions made by B.

5 Project Report

Include with your report:

- A brief introduction describing the project. In this section, include the responsibilities of each team member with references to which parts/steps he/she completed (no more than 1 page).
- A brief description of how you developed your simulations (no more than 3/4 of a page).
- Graphs for each of the simulated scenarios, according to what is required below.
- Commentary for each graph and comparison between graphs. Sample commentary. "In Figure X, we show the Throughput of A as a function of the arrival rate λ of each station. We observe that initially (for $\lambda < xx$), the throughput grows linearly with the arrival rate. This is because contention is low and the two stations do not collide. This is further verified by Figure Y, which shows the number of collisions as a function of the arrival rate λ. For $\lambda < xx$, the number of collisions is less than Z. As the rate increases, we observe that the throughput ..."

You need to generate 6 graphs in total.

1. Throughput T: Label each line as DCF-topology(a), DCF-topology(b), DCF/VCS-topology(a), and DCF/VCS-topology(b).
   (a) **Station A:** Throughput T (Kbps) vs. rate λ (frames/sec) for DCF-topology(a), DCF-topology(b), DCF/VCS-topology(a), and DCF/VCS-topology(b) (*four lines in total on the same graph*).
   (b) **Station B:** Throughput T (Kbps) vs. rate λ (frames/sec) for DCF-topology(a), DCF-topology(b), DCF/VCS-topology(a), and DCF/VCS-topology(b) (*four lines in total on the same graph*).
2. Collisions N
   (a) **Collisions at the access point AP:** Number of collisions N vs. rate λ (frames/sec) for DCF-topology(a) and DCF/VCS-topology(a) (*two lines in total on the same graph*).
   (b) **Collisions perceived by A and B:** Number of collisions N vs. rate λ (frames/sec) for DCF-topology(b), DCF/VCS-topology(b) (*four lines in total on the same graph*).
3. Fairness Index FI
   (a) Fairness Index FI vs. rate λ (frames/sec) for DCF-topology(a), DCF-topology(b), DCF/VCS-topology(a), and DCF/VCS-topology(b) (*four lines in total on the same graph*).

Justification for the results shown in your graphs. The majority of your grade will be based on your interpretation of the results, not the results themselves, so spend adequate time explaining them.

Appendix 1 - Poisson Traffic

Generating Poisson-distributed traffic: To create and run your simulation you need to generate traffic. This traffic dictates the arrival of packets at the respective Txs for transmission. Since we are doing an event simulation, we are only interested in the exact arrival times (in slots) and will not implement real arrivals and packet transmissions. The times of arrivals will follow the Poisson distribution with parameter λ. To generate Poisson-distributed arrival times, it is sufficient to generate a series of exponentially distributed inter-arrival times with the same parameter λ. Such times can be generated using the inverse CDF transformation method.

Step 1: Generate a series of uniformly distributed numbers $U = \{u_1, u_2, \ldots, u_n\}$ with $u_i \in (0, 1), \forall i$. To compute how many numbers are needed in the series you can simply multiply the rate of arrivals with the simulation time. For instance, when $\lambda = 200$ frames per second, you will need approximately 2,000 frames to fill arrivals for 10 seconds. That means you need to draw 2,000 at random in (0, 1).
Step 2: Convert the series U to series $X = \{x_1, x_2, \ldots, x_n\}$ of exponentially distributed numbers with parameter λ, as

$$X = -\frac{1}{\lambda} \ln(1 - U). \tag{1}$$

Here, you get another series of n numbers that are exponentially distributed. Using X, you can determine the time that each frame is written from the upper layers to the transmitter's queue. Since X corresponds to inter-arrival times, the first frame is ready to transmit at time x_1, frame 2 is ready to transmit at time $x_1 + x_2$, etc. Please note that this are not the actual transmission times, but the time that packets are placed in the queue. Once the queue, the CSMA/CA protocol needs to be executed to determine the exact transmission time. For simplicity of accounting, you can convert all your times into slots. The above process will give you times in seconds. By dividing with the slot duration, you will get the arrivals times in slots.

Numerical Example:

Let the arrival rate be equal to $\lambda = 100$ frames/sec. At this rate, we expect, on average, one frame to arrive every 1,000 slots (Slot is 10 μsec long, so there are 100,000 slots in 1 sec. Given a 100 frames/sec rate, 1 frame will arrive roughly every 1,000 slots). When you draw numbers exponentially-distributed with an average of $\lambda = 100$ frames/sec and convert them to slots by dividing with the slot duration, you will get a sequence of this sort.

$$X = \{985, 789, 1045, 1140...\}$$

Each number in this set represents an inter-arrival time (time between two successive frame arrivals at a given station. To translate these number to the times that frames are at the transmitter's queue ready to be transmitted, you can simply start adding them. Then you come up with a new sequence Y

$$Y = \{985, 985 + 789, 985 + 789 + 1045, 985 + 789 + 1045 + 1140...\}$$

These are the arrival times of the 1st, 2nd, 3rd, and 4th frame, respectively. You need two such sequences, one for A and one for C. Both sequences are randomized by different seeds. Once you have the sequence, you simulation needs to look into which events occur first and account for the protocol implementation.

Appendix 2 - Implementation tips

- You are asked to implement an event-driven simulation. In this simulation, you are focusing your implementation around events of interest. Arrival, transmission, collision, etc. So rather than

counting the time per slot, you advance the simulation time to the next event that occurs. Your algorithmic part is to determine which event will occur next out of the competing possibilities, (e.g., which node will get his backoff timer to zero first).

- DO NOT RUN your simulation for 10 seconds until you have verified that it works correctly. It is very difficult to figure out if the outcomes are correct this way. Initially, rather than working with random arrivals, fix the timings of your arrivals for just a few frames to create test cases for (a) successful transmission from A, (b) successful transmission of C, (c) collision of A and C, (d) collision of RTS with frame in the hidden terminal scenario, (e) other test cases. For each test scenario, draw out the timeline of events by hand and compare it to the timeline in your simulation.

- To get one data point for your plots, you fix the arrival rate λ, generate the appropriate number of frame arrivals for each transmitting station, and then count the number of successful frames for each station and the collisions. You repeat with different λ's to get the next data points in your plots.

- When you get final results use qualitative reasoning to see if they are correct. If the medium capacity is 10Mbps and your simulation yields a sum of rates for A and C that exceeds 10Mbps, something is wrong. If your throughput is low at high lambdas but you have close to zero collisions something is wrong.

Plotting tips:

1. Label your axes and use appropriate units
2. Make sure the scales on both axes are appropriate. If you are to use the same variable on multiple plots (e.g. throughput) use the same scale on all plots so they can be compared
3. Do not superimpose more than 4-5 plot lines on the same plot.
4. If more than one plot line is present in the same plot make sure to individually label each one
5. For individual plot lines use different marker shapes so they can be distinguishable.
6. Keep in mind that colors do not show on a black-and-white printout. So if you color code your lines, use some other distinct labeling such as dashed lines to differentiate between plot lines.

MATLAB code for generating good figures. You can port your data to Matlab and use this code to generate figures.

```
close all; % closes all open figure windows

set(0,'defaulttextinterpreter','latex'); % allows you to use latex math
set(0,'defaultlinelinewidth',2); % line width is set to 2
set(0,'DefaultLineMarkerSize',10); % marker size is set to 10
set(0,'DefaultTextFontSize', 16); % Font size is set to 16
set(0,'DefaultAxesFontSize',16); % font size for the axes is set to 16

figure(1)
plot(X, Y1, '-bo', X, Y2, '–rs', X); % plotting three curves Y1, Y2 for the same X
grid on; % grid lines on the plot
```

```
legend('CSMA', 'CSMA w. virtual Sensing');
ylabel('$T$ (Kbps)');
xlabel('$λ$' (frames/sec));
```