Homework 1: Exploring the Design of Single-hop CSMA and TDMA Protocols

# The Problem

In order to achieve develop high-performance wireless networks it is imperative to design an effective media access and control (MAC) layer. Two key approaches to addressing this difficult problem are CSMA and TDMA protocols. This homework will

* allow you to explore various options to implement CSMA and TDMA protocols in a single hop environment
* compare the trade-offs involved by collecting statistics regarding the goodput, packet drop rate, packet delay, and fairness provided by each of the protocols
* analyze the results that you obtained.

# Simulating CSMA and TDMA protocols

The goal of this assignment is to simulate the behavior of the MAC layer in the case of a single hop network. We will make a number of idealized assumptions to simplify their implementation:

* **Connectivity:** We will assume that the nodes form a complete graph (<http://en.wikipedia.org/wiki/Complete_graph>). Accord to the protocol model, the nodes will be able to communicate with each other without losses.
* **Time Synchronization:** Time is divided into slots. Nodes are time synchronized and in each slot they may perform carrier sense or packet transmissions.
* **Packet Lengths:** The packet length is a multiple of a number of slots.
* **Carrier Sense:** The nodes will also be able to perform carrier sense without error. The only case when carrier sense fails is when two nodes decide to transmit within the same slot.
* **Fixed queue length:** each node will have a queue of fixed size

The protocols that you will have to implement are the following:

* **CSMA-linear:** a CSMA protocol whose contention window will be increased linearly (i.e., N slots at a time) when a node determines that the channel is busy
* **CSMA-exponential:** a CSMA protocol whose contention window will be increased exponentially when a node determines that the channel is busy
* **TDMA:** a simple TDMA protocol whose frame contains one slot for each node in the network.

# Single-hop Simulator

To simplify the task of simulating the protocols, I have developed a simple simulator that you may modify to implement the protocols. Currently, the simulator implements a simple CSMA-like protocol in which a node randomly selects to transmit a packet within a constant contention window.

The simulator contains a class node that stores all the state associated with the behavior of a node (e.g., its transmission queue, backoff counter, and contention window). In addition, a class packet contains some additionally minimal state. Most importantly, it keeps track of the its size (in terms of slots) and the number of slots until its transmission is completed.

The core of the simulator is a simple for loop that iterates over a number of slots specified by variable num\_slots. In the for loop, the *execute* method will be called on each node to perform its operating within the slot. The parameters of the method include the clock tick and the carrier sense indicator (i.e, is the channel busy or note). If the node will transmits a packet, it will be returned at the end of the *execute* invocation.

The *execute* method involves the following steps:

1. Determine if the application has a packet to transmit. This is done using the helper method *generate\_traffic.*
2. Determine weather there is enough room to store the generated packet within the queue. Otherwise, the packet is dropped.
3. Implement the media access protocol. For CSMA this involves appropriately configuring the backoff and cw variables based on the carrier sense values. For TDMA, access must be determined based on the position of a node within a frame. You will have to implement this yourself, though it is straightforward!

After invoking *execute* for each node, the simulator proceeds to determine if there were any packet collisions.

# Metrics

Next, we will evaluate the performance of the developed protocols based on the following metrics. In computing the following metrics we assume that that the slot is 10ms.

* **Channel goodput (pkts/s)**: this captures the number of uncorrupted packets that are transmitted over the channel. There are two ways in which packets will not be transmitted in our system: (1) if more than one node decides to transmit in a single time slot and (2) if the queue size is exceeded, packets will be dropped.
* **Packet drop packets (%)**: the ratio between the number of dropped packets and the total number of packets that are transmitted. Packets that are lost due to collisions are not included here.
* **Packet delay (s):** the delay of a packet includes the time from when the packet is generated until it is sent out the mac layer. Note that the packet delay includes the time the packet spends in a queued before its transmission.
* **Fairness (%):** as discussed in class, there are several ways that one may define fairness. In this assignment, we define the fairness of a node N as the radio between goodput of node N (computed as channel goodput but looking at only the packets transmitted by the packets transmitted by N) and the channel goodput.

# Experiment

In the experiment, we will be observing the impact of increasing the number of nodes in the network on the previously defined metrics. The number of nodes should be increased until the fraction of dropped packets will reach at least 70%. From this experiment, I expect that you will be turning in the following graphs:

* Graph 1: channel goodput vs. number of nodes. This is a simple line plot *with 90% confidence intervals*, and will contain 3 lines one per protocol.
* Graph 2: fraction of dropped packets vs. number of nodes. This is a simple line plot *with 90% confidence intervals*, and will contain 3 lines one per protocol.
* Graph 3: packet delay vs. number of nodes. This is a simple line plot *with 90% confidence intervals*, and will contain 3 lines one per protocol.
* Graph 4: fairness of each node. I suggest that you use a boxplot for this type of graph. If that gets too complicated than you can produce a line graph with the avg. fairness and *90% confidence intervals*.

***To gain statistical significant the results must be averages of at least 5 runs with different seeds! Make sure that you add the 90% confidence (you should use the Student-t distribution to compute them).***

# Turn-in

1. Code – zip it
2. A write-up containing the following the four graphs mentioned above and answers. For each graph, I expect a short write-up interpreting the data. Here are some questions to help you out:
	* Which protocol performed the best?
	* Do you have insights why?
	* Did a protocol behave consistently better than others?
	* Did the results surprise you in any way?