Problem Set 7

Your answers will be graded by actual human beings (at least that's what we believe!), so don't limit your answers to machine-gradable responses. Some of the questions specifically ask for explanations; regardless, it's **always a good idea to provide a short explanation for your answer**.

Please read Chapter 15 **before** solving these problems (and you may find it useful to consult these while solving them too). Please also solve the online practice problems on MAC protocols and the problems at the end of Chapter 15.

Problem 1.

Tim D. Vider thinks Time Division Multiple Access (TDMA) is the best thing since sliced bread. In Tim's network with N nodes, the **offered load is not uniform** across the different nodes. The **rate** at which node i generates new packets to transmit is $r\_i = 1/2^i$ packets per time slot ($1 \leq i \leq N$). That is, in each time slot, the application on node i produces a packet to send over the network with probability r_i.

A. Tim runs an experiment with TDMA for a large number of time slots. At the end of the experiment, how many nodes (as a function of N) will have a substantial backlog of packets (i.e., queues that are growing with time)?
B. Let N=20. Calculate the **utilization** of Tim's TDMA protocol for this non-uniform workload.

(points: 3)

Problem 2.

Ben Bitdiddle sets up a shared medium wireless network with one access point (AP) and N client nodes. Assume that both the AP and the N client nodes are backlogged. Each of the N clients wants to send its packets to the access point; the AP's packets are destined to various clients. The network uses slotted Aloha with each packet fitting exactly in one slot. Ben sets the transmission probability, p, of each client node to 1/N and sets the transmission probability of the AP to a value p_a.

A. Determine the utilization of the network in terms of N and p_a.

(points: .5)

B. What is the utilization of the network when N is large?

(points: .5)

C. Suppose N is large. What value of p_a ensures that the aggregate throughput of packets received successfully by the N clients is the same as the throughput of the packets received successfully by the access point?

(points: 1)

From here on, only the client nodes are backlogged -- the access point has no packets to send. Each client node sends with probability p (don't assume it is 1/N).

Ben Bitdiddle comes up with a cool improvement to the receiver at the access point. If exactly one node transmits, then the receiver works as usual and is able to correctly decode the packet. If exactly two nodes transmit, he uses a method to cancel the interference caused by each packet on the other, and is (quite remarkably) able to decode both packets correctly.

D. What is the probability, P2, of *exactly* two of the N nodes transmitting in a slot? Note that we want the probability of *any* two nodes sending in a given slot.

(points: 1)

E. What is the throughput of slotted Aloha with Ben's receiver modification, measured in packets per time slot? Write your answer in terms of N, p, and P2, where P2 is defined in the problem above.

(points: 1)

Problem 3.

Ben Bitdiddle runs the slotted Aloha protocol with stabilization. Each packet is one time slot long. At the beginning of time slot T, node i has a probability of transmission equal to p_i, $1 \leq i \leq N$, where N is the number of backlogged nodes. The increase/decrease rules for p_i are doubling/halving, with $p_{min} \leq p_i \leq p_{max}$, as described in Chapter 15.

Ben notes that exactly two nodes, j and k, transmit in time slot T. Derive an expression for the probability that node either node j or node k will transmit successfully in time slot T+1. Show your work.

(points: 2)

Problem 4.

Token-passing is a variant of a TDMA MAC protocol. Here, the N nodes sharing the medium

are numbered 0, 1, ..., (N-1). The token starts at node 0. A node can send a packet if, and only if, it has the token. When node *i* with the token has a packet to send, it sends the packet and then passes the token to node (*i*+1) mod N. If node *i* with the token does not have a packet to send, it passes the token to node (*i*+1) mod N. To pass the token, a node broadcasts a token packet on the channel and all other nodes hear it correctly.

A data packet occupies the channel for time T_d. A token packet occupies the channel for time T_k. If *s* of the N nodes in the network have data to send when they get the token, calculate the utilization of the channel in terms of the parameters above. Note that the bandwidth used to send tokens is pure overhead; the throughput we want corresponds to the rate at which data packets are sent.

Hint: When 20% of the nodes have data to send (i.e., *s*/N = 0.2) and T_d=10*T_k, the utilization is 2/3.

(points: 2)

Problem 5. Contention windows

Recall the MAC protocol with contention windows from Chapter 15 (Section 15.8). Here, each node maintains a contention window, W, and sends a packet t idle time slots after the current slot, where t is an integer picked uniformly in [1,W]. Assume that each packet is 1 slot long.

Suppose there are two backlogged nodes in the network with contention windows W1 and W2, respectively (W1 ≥ W2). Suppose that both nodes pick their random value of t at the same time. What is the probability that the two nodes will collide the next time they each transmit?

(points: 2)

Problem 6. Two radios.

Eager B. Eaver gets a new computer with **two** radios. There are N other devices on the shared medium network to which he connects, but each of the other devices has only one radio. The MAC protocol is slotted Aloha with a packet size equal to 1 time slot. Each device uses a fixed transmission probability, and only one packet can be sent successfully in any

time slot. All devices are backlogged.

Eager persuades you that because he has paid for two radios, his computer has a moral right to get twice the throughput of any other device in the network. You begrudgingly agree.

Eager develops two protocols:

Protocol A: Each radio on Eager's computer runs its MAC protocol independently. That is, each radio sends a packet with fixed probability p. Each other device on the network sends a packet with probability p as well.

Protocol B: Eager's computer runs a single MAC protocol across its two radios, sending packets with probability 2p, and alternating transmissions between the two radios. Each other device on the network sends a packet with probability p.

A. With which protocol, A or B, will Eager achieve higher throughput?

(points: 1)

B. Which of the two protocols would you allow Eager to use on the network so that his expected throughput is double any other device's?

(points: 1)

Introduction to this week's Python tasks.

This lab uses **WSim**, a simple packet-level network simulator for a shared medium network. You will be writing a small amount of code to develop various MAC protocols and measure how they perform under different conditions. Much of your work will be on experimenting with various parameters and explaining what you observe. The amount of new code you have to write is rather small.

In each experiment, all the nodes run the same MAC protocol. The simulator executes a set of steps every time slot; time increments by 1 each slot.

You can run the python programs for this lab using python from the terminal command line, e.g., `python PS7_???.py`. **This lab may not work well in IDLE (you can use IDLE to edit files, but running them may not work as expected).**

To understand the different parameters one can set in WSim, go to a shell (i.e., command line prompt on a terminal application) and enter:

```
python PS7_tdma.py -h
```

This command prints out the various options; the important ones are:

1. **Packet size:** In any given experiment, the size of a packet is fixed. It has to be an integral number of time slots in size (1 or more). To set the packet size, use the `-s` option; the default is 1. Notice that setting a large packet size (say, 10) emulates an "unslotted" network.

2. **Number of nodes:** The number of nodes in a run of WSim; default is 16. Set using `-n`.

3. **Retry:** If two or more nodes are actively sending a packet in the same time slot, they collide and both packets are considered lost. The `-r` option decides whether the node should *retry* the packet or not upon failure. In WSim, the feedback about whether a packet succeeded or not (i.e., collided) is instantaneous, with the sending node discovering it in the same time slot as the transmission. By default, the retry option is "off". When it is turned on, at an *offered* load of 100% (which is what we will use in this lab), the **actual** load presented to the system **exceeds** the channel's maximum rate. That is, we would expect most queues to be backlogged most of the time with these settings.

4. **Skew:** The -k option specifies whether the load is skewed or not. The load itself is generated according to a random process, whose details aren't important for this lab. By default, the skew is off, so all nodes generate the same load on average. When the `-k` option is set, then the total offered load, L, is divided in geometrically-spaced amounts. Node 0 presents a load of L/2, node 1 a load of L/4, node 2 a load of L/8, and so on. The last two nodes each present the same load, $L/2^{N-1}$, where N is the total number of nodes.

5. **GUI:** The `-g` option turns on the graphical user interface, which may be of some use in debugging your code. **We recommend that you set the parameters for the simulation from the command line and not from the GUI, as the GUI's parameter setting code may not port well across different python installations.**

Experimental method

WSim runs for a specified number of time slots (settable using the `-t` option, with a default of 10000) and prints out some performance numbers (and possibly displays some graphs) at the end. Of interest to us are the **utilization** and **fairness** numbers, which are defined in the lecture notes. The code reports a "weighted" fairness number as well, which is the fairness index calculated over the ratio of the observed throughputs to offered loads. The (unweighted) inter-node fairness is calculated over the throughputs alone, without regard to the offered load.

In this lab, you will write the core of three MAC protocols---**TDMA**, **stabilized Aloha** (with backoffs), and **CSMA**---and understand their performance. Each node is an object, which has

three methods that you can use to implement the core of the MAC protocol:

boolean = *node*.channel_access(time,ptime,numnodes)
: This method is called by WSim every time slot when the node has a packet waiting to be sent in its queue. This method should return `True` if the MAC protocol you're implementing would like a packet sent in the current time slot, and `False` otherwise. `time` is current time, `ptime` is the packet size in time slots, and (for TDMA) `numnodes` is the number of nodes in the network. (You must not use `numnodes` in the other protocols.)

node.on_collision(packet)
: Called every time slot in which the node has experienced a collision.

node.on_xmit_success(packet)
: Called every time slot in which the node has successfully completed the transmission of a packet (i.e., no collisions occurred during transmission).

You can modify any per-node state that you want to in these methods (e.g., the state maintained by the backoff scheme, statistics of interest, etc.). Make sure to add the code to initialize this state in the Node object's `__init__` function, whose body is included in the lab task files.

In many of our experiments, we will use the `-r` option, which cause the nodes to retry upon experiencing a collision. (Of course, the MAC protocol's `channel_access()` method will determine when the retry actually occurs.)

Useful download links:

> Zip archive of all the Python files
> PS7_wsim.py -- packet-level simulator

Python Task #1: Time Division Multiple Access (TDMA)

Useful download link:

> PS7_tdma.py -- template file for this task

Implement a simple TDMA scheme by suitably filling in the `channel_access()` method in the template file PS7_tdma.py. Recall that in a TDMA scheme, time is divided into `numnodes` equal-size slots, each long enough to accommodate the transmission of a single packet and that each node is allocated one of the slots to use when it has a packet to transmit (see §15.3 of Chapter 15). Note that a node can determine its unique node number (an integer between 0 and `numnodes - 1`) by calling `self.get_id()`.

The slightly tricky part of this function is to correctly handle packet sizes that are larger than 1 time slot. We want the TDMA scheme to treat each packet as an atomic unit of transmission; when the protocol determines that a given node can send, that node should send the complete packet. That is, we want the effective size of a time slot in the scheme to be equal to the packet size. You will probably find it easier to first write the function and run it

for a packet size of 1 slot, then modify your code to correctly handle larger packet sizes. Note that when the packet size is set to some value using the -s option, all nodes will use that value.

Run the following, after you test your code with various packet sizes to ensure that there are no collisions:

- `python PS7_tdma.py -t 2000`
  (16 nodes, packet size = 1 slot, simulation time = 2000 slots)
- `python PS7_tdma.py -s 7 -t 14000`
  (16 nodes, packet size = 7 slots, simulation time = 14000 slots)
- `python PS7_tdma.py -k -n 20`
  (20 nodes, skewed load, packet size = 1 slot, simulation time = 10000, the default value)

When you're ready, please submit the file with your code using the field below.

File to upload for Task 1: Browse...

(points: 1)

Questions:

A. Please run the following experiments using a skewed load (`-k`) where the load offered by a node decreases with the node number, i.e., high-numbered nodes have a packet to transmit much less frequently than low-numbered nodes.

```
python PS7_tdma.py -k -n 10
python PS7_tdma.py -k -n 20
python PS7_tdma.py -k -n 40
python PS7_tdma.py -k -n 80
```

Please report the utilization from each experiment (look for `util` in the printout). With a skewed load, as one increases the number of nodes, what happens to the utilization? Why?

(points: 0.5)

B. What is the number of nodes at which the network utilization is smaller than 0.25 for the skewed workload? (Because each run is randomized, run it a few times to be confident of your answer.)

(points: 0.5)

Python Task #2: Stabilizing Aloha (with Backoff)

Useful download link:

PS7_stabaloha.py -- template file for this task

In this task, we will develop a stabilization method for Aloha using randomized backoffs, as described in §15.5 (but also read §15.4 and §15.6 for the full story). Our goal is to adaptively select the transmission attempt probability, `p`, used in the `channel_access` method. To do that, write your code in PS7_stabaloha.py to adjust `p` in the `on_collision` and `on_xmit_success` methods, which are called when a packet transmission fails and succeeds, respectively.

We will use two parameters, `pmax` and `pmin`. These correspond to the maximum and minimum values of the transmission attempt probability, `p`. The values of these parameters can be set from the command line when you run the program, and are available as `self.network.pmax` and `self.network.pmin` respectively (see the `__init__` function of AlohaWirelessNetwork). In your code, ensure that $\texttt{pmin} \leq \texttt{p} \leq \texttt{pmax}$.

You can use any algorithm you want to set `p` in these functions, including the ones discussed in lecture. Good schemes achieve high utilization, but also ensure that fairness is high, and avoid the capture effect. The fairness should be as close to 1 as possible -- when the number of nodes is between 8 and 16, fairness lower than 0.9 is a sign that there is significant unfairness. There is no absolute correct answer (though there are bad methods!), so feel free to be creative if you think you have a good idea. Note that you are not allowed to use the number of backlogged nodes or `numnodes` in your scheme, because that information would not be available in practice.

Run your code as follows for a few different settings of pmin and pmax and observe the utilization and inter-node fairness values.

`python PS7_stabaloha.py -r -n 8 --pmin=`*`value`* `--pmax=`*`value`*

(Note the two dashes in front of the pmax and pmin options.)

When you're ready, please submit the file with your code using the field below.

File to upload for Task 2: Browse...

(points: 2)

Questions:

A. Run `python PS7_stabaloha.py -r -n 8 --pmin=0 --pmax=1`. Would you recommend running a real network with these parameters for `pmin` and `pmax`? Briefly explain your answer.

(points: 0.5)

B. Set `pmin` to 0.01 and pick `pmax` so that the fairness is as large as possible (at least 0.9) when the number of nodes is 8 and there is no load skew. What value of pmax did you pick? What is the utilization of your protocol when the packet size is 1 slot? How does it compare to the utilization when the packet size is 10 slots? For the first case (packet size of 1), run

```
python PS7_stabaloha.py -r -n 8 --pmin=0.01 --pmax=value
```

For the second case (packet size of 10), run

```
python PS7_stabaloha.py -s 10 -t 70000 -r -n 8 --pmin=0.01 --pmax=value
```

(points: 0.5)

Python Task #3: Carrier Sense Multiple Access (CSMA)

Useful download link:

PS7_csma.py -- template file for this task

In this task, we will try to get the best utilization and fairness we can for a MAC protocol that uses CSMA. To check if the channel is idle, you can use the `self.network.channel_idle()` from inside `channel_access()`. This function returns `True` if there is no on-going transmission in the **current time slot**, and `False` otherwise.

Every carrier sensing mechanism has a *detection time*, defined as the time interval between the ending of a previous transmission and the detection of the channel as "idle" by a node. For the purposes of this lab, we will assume that the detection time is 0. Hence, a node can sense that the carrier is idle in the immediate next slot after the termination of the previous transmission, when it does the check for whether the channel is busy. However, it is still possible for collisions to occur, for multiple nodes could simultaneously sense the channel at

the beginning of a slot, and conclude that the channel is "idle", and possibly attempt a transmission in that time slot (or in the same future time slot).

Write your code PS7_csma.py for the `channel_access()` method assuming that the node has the ability to sense the carrier. Obviously, you should fill in the steps for the `on_collision()` and `on_xmit_success()` methods as well.

Test your code as follows. The `-s` option is important because it causes the packets to be longer than 1 slot, allowing a node to sense whether another transmission is in progress during a time slot.

```
python PS7_csma.py -r -n 8 -s 10 -t 100000 --pmin=value --pmax=value
```

Note that you should run the above for 100000 time slots, because we have scaled up the packet size to 10 (from 1).

When you're ready, please submit the file with your code using the field below.

File to upload for Task #3: Browse...

(points: 1)

Questions:

A. What is the utilization and fairness of your protocol when `pmin` = 0 and `pmax` = 1?

(points: 0.5)

B. How would you set `pmin` and `pmax` in your protocol to make the fairness number be over 0.95 consistently while still maintaining good utilization (i.e., about 75% or so)? [This question may not be too easy; one may need some trial-and-error.]

(points: 0.5)

Python Task #4: CSMA with contention windows, as in WiFi (802.11) and Ethernet (802.3)

Useful download link:

PS7_cw.py -- template file for this task

The ALOHA and CSMA schemes in the previous tasks pick a probability `p` for transmitting a packet, and adapt `p` to stabilize the protocol. The advantage of this method is that it is easy to analyze. In practice, however, real-world CSMA protocols like the popular 802.11 WiFi standard and the 802.3 Ethernet standard implement something a bit different, as explained below. Your task will be to write the code for the key parts of this scheme.

Rather than decide whether any given slot should have a transmission with probability `p`, each node maintains a **contention window**, which we denote by **`cw`**. `cw` is initially set to `cwmin`, which is a small positive integer (say, 1). Denote the current time slot by C. If the sender is backlogged, it picks a random integer t in [1, `cw`] and decides to send a packet in time slot C + t.

Of course, with carrier sense in place, the sender should only send a packet if the channel is idle in time slot C + t. So, the sender senses the carrier in that time slot, and then sends a packet only if the channel is idle then. If the channel is not idle, it waits until the channel is idle, and then sends the packet.

Whenever a collision occurs, the node doubles `cw`, but makes sure `cw` never exceeds `cwmax` (say, 512). Whenever a transmission is successful, the node might reset cw to `cwmin`, or might halve its current value of `cw`. You will note that this scheme is similar in spirit to the probabilistic transmission scheme of stabilized Aloha (Task #2), but a crucial difference is that here each backlogged node is **guaranteed** to send a packet within a finite time, unlike in the probabilistic case where there is always a small probability that the node cannot send within any given number of time slots. In mathematical terms, the probability distribution that governs whether a node transmits a packet in a given time slot is uniform in this scheme, while it is geometrically distributed in Tasks #2 and #3.

You will first implement the scheme described thus far. It will turn out not to do as well as we would like, and in Task 4.2, you will fix an important weakness.

Implement this scheme by writing the appropriate code for channel_access(), on_collision(), and on_xmit_success() in PS7_cw.py. Use `self.network.cwmin` and `self.network.cwmax` to access the values of `cwmin` and `cwmax` respectively. How well does it work? To answer this question, measure the utilization and fairness by running

```
python PS7_cw.py -r -s 10 -n 16 -t 100000  -W 256
```

The `-W` option sets the maximum contention window size. The minimum contention window is 1 (you can change it using the `-w` option if you like (lower case "w"). Running the above, you will note that even with carrier sense being used, the utilization is quite a bit lower than in Task 3.

A. Report the utilization and fairness. Briefly explain why the utilization is low. Hint: Think about what happens if more than one node is backlogged and waiting for an on-going transmission to complete; what happens when the on-going one finishes?

(points: 0.5)

To fix this problem, each node needs to ignore the time slots when other nodes are transmitting data. That is, if a node picks a time slot t in [1, cw] to transmit, it should wait for that many **idle** slots before attempting its own transmission. Of course, before transmitting data, it should ensure that the channel is idle.

Modify your code to include the above suggestion and run the same command as before:

```
python PS7_cw.py -r -s 10 -n 8 -t 100000  -W 256
```

B. Report the utilization and fairness for your scheme after including the suggestion and running the same command as before.

(points: 0.5)

Please upload the final version of your Task #4 code:

File to upload for Task 5: Browse...

(points: 4)

6.02 Introduction to EECS II: Digital Communication Systems

Fall 2012