Data Networks
HW 3: MAC Sublayer Simulation

Dr. Mohammad reza Pakravan

Due on

Problems

Question1

Suppose nodes A and B are at the same 10 Mbps Ethernet segment and the propagation delay between the two nodes is 200 bit times. Suppose A and B send frames at the same time, the frames collide, and then A and B choose different values of K in the CSMA/CD algorithm. Assuming no other nodes are active, can the retransmissions from A and B collide? For our purposes, it suffices to work out the following example. Suppose A and B begin transmission at t = 0 bit times. They both detect collisions at t = 200 bit times. They finish transmitting a jam signal at t = 248 bit times. Suppose $K_A = 0$ and $K_B = 1$.

a) At what time does B schedule its retransmission?
b) At what time does A begin retransmission?
c) At what time does A's retransmitted signal reach B?
d) Does B refrain from transmitting at its scheduled retransmission time? Why or why not?
e) Can the retransmissions from A and B collide? Why or why not?

Question 2

which statement is not true about HDLC protocol?
a) start and end of HDLC frame is identified by flag"01111110"
b) address field has usually 8 bits but it is expandable to 8n bits.
c) if the frame is of supervisory, send sequence number should be controlled in control field
d) to avoid interference, bit stuffing is used and address field (FF) indicates broadcasting

Question 3

Answer the following questions for wireless communication:
a) How can a wireless node interfere with the communications of another node when the two nodes are separated by a distance greater than the transmission range of either node?
b) Why is collision detection more complex in wireless networks than in wired networks such as Ethernet?
c) Why isn't it practical for each node in a sensor network to learn its location by using GPS? Describe a practical alternative.

Question 4

Suppose we use the pure ALOHA method for communication between nodes A and B, where A and B send packets of length 4 ms and X ms, respectively. If a collision occurs at time t_0 during the first transmission, the nodes wait a random interval t (uniformly distributed between 0 ms and 20 ms) before resending the packets. What value of X results in a collision probability of exactly 39.875 percent?

Question 5

Suppose that the ALOHA protocol is used to share a 56 Kbps satellite channel. Suppose that frames are 1000 bits long. Find the maximum throughput of the system in frames/second.

Question 6

In a LAN, which MAC protocol has a higher efficiency: ALOHA or CSMA/CD? What about in a WAN?

Question 7

In a slotted ALOHA network, 100 communication stations use a channel with 100 Mbps rate. size of each frame is 100 bits. if each station produces 1000 frames per second and sends it out, calculate throughput of this system.

Question 8

which one of the followings shows a true statement about comparing throughput (for G=10)(p = persistant)
1) Nonpersistent CSMA > 0.01p CSMA > 0.1-p CSMA > 0.5-p CSMA
2) 0.01p CSMA > Nonpersistent CSMA > 0.1-p CSMA > 0.5-p CSMA
3) 0.5-p CSMA > 0.1-p CSMA > 0.01p CSMA > Nonpersistent CSMA
4) 0.1-p CSMA > 0.01p CSMA > Nonpersistent CSMA > 0.5-p CSMA

Simulation

The Hidden Node Problem(40 points)

In this problem we are going to simulate the hidden node problem and the effect of RTS/CTS using OMNeT++. For this purpose we use one of INET's showcases. The hidden node problem is the collective name of situations where a transmitting node does not know about the existence of another node (the "hidden node") while transmitting to a third node which is within the range of both nodes. Since the node doesn't know when the hidden node is transmitting, normal collision avoidance is not effective, and their transmissions will often collide at the third node. The hidden node problem reduces channel utilization and damages network performance. Hidden nodes may be created by distance, by obstacles that block radio signals, by unequal transmission powers, or by other factors. The situation may be symmetric or asymmetric (the roles of the originator and the hidden node may or may not be interchangeable.)
The 802.11 protocol allows transmissions to be protected against interference from other stations by using the RTS/CTS mechanism. Using RTS/CTS, the node wishing to transmit first sends an RTS (Request To Send) frame to the target node. The target node, if the channel is clear, responds with a CTS (Clear To Send) frame that not only informs the originator that it may transmit but also tells other stations to refrain from using the channel for the specified amount of time. RTS/CTS obviously adds some overhead to the transmission process, so it is normally used to protect longer frames which are more expensive to retransmit. RTS/CTS does not completely solve the hidden node problem, but it can significantly help under certain conditions.

The network used is:

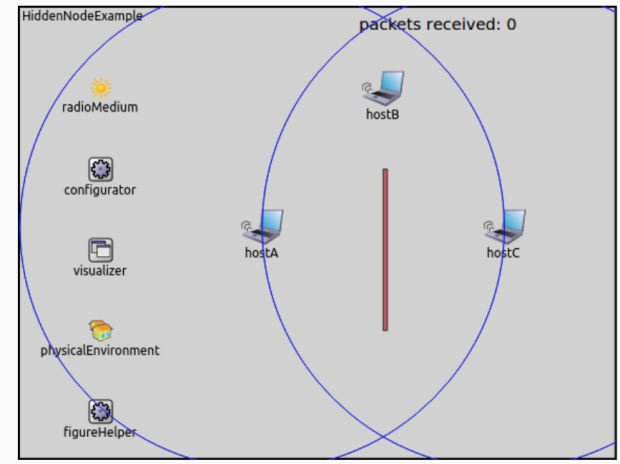

Figure 1: Hidden network problem - network topology

- In this problem, we'll set up a wireless network that contains a hidden node. To ensure that two selected nodes don't hear one another, we'll place an obstacle (a section of wall that blocks radio signals) between them. (10 points)
- now run the simulation without RTS/CTS, with RTS/CTS turned on, and for reference, also with the wall removed. Provide enough screenshots/ recordings to demonstrate the results. Discuss about the results, and the effect of RTS/CTS.(30 points)

MAC Layer Protocols(60 points)

This problem 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.
The important parameters involved 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). 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;
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 retry 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. When it is turned on, at an offered load of 100% (which is what we will use in this problem), 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: This 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 problem. By default, the skew is off, so all nodes generate the same load on average. When the skew option is set, then the total offered load, L, is divided in geometrically-spaced amounts. Node 0 presents a load of $\frac{L}{2}$, node 1 a load of $\frac{L}{4}$, node 2 a load of $\frac{L}{8}$, and so on. The last two nodes each present the same load, $\frac{L}{2^{N-1}}$, where N is the total number of nodes.

5. GUI: The GUI option turns on the graphical user interface, which may be of some use in debugging your code. We recommend that you do not set the parameters for the simulation, as the GUI's parameter setting code may not port well across different python installations.
The default values for these parameters has been assigned in the MAC notebook. please use the default values unless otherwise is stated.
In this problem, you will write the core of three MAC protocols—**TDMA, stabilized Aloha (with back-offs), 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).

TDMA (10 points)

Implement a simple TDMA scheme by suitably filling in the *channel_access*() method. 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. 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, 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: **(5 points)**

- set nodes = 16, packet size = 1 slots, simtime = 2000
- set nodes = 16, packet size = 7 slots, simtime = 14000
- set nodes = 20, skewed load, packet size = 1 slots, simtime = 10000

Now Answer the following questions:

- Please run the following experiments using a skewed load 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.
  - set nodes = 10
  - set nodes = 20
  - set nodes = 40
  - set nodes = 80

  Please report the utilization from each experiment. With a skewed load, as one increases the number of nodes, what happens to the utilization? Why? **(2.5 points)**

- 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.) **(2.5 points)**

Stabilizing Aloha (with Backoff) (15 points)

In this task, we will develop a stabilization method for Aloha using randomized backoffs. Our goal is to adaptively select the transmission attempt probability, p, used in the *channel_access* method. To do that, write your code in the dedicated area in MAC notebook 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 as needed, and are available as $self.network.pmax$ and $self.network.pmin$ respectively. In your code, ensure that $pmin \leq p \leq pmax$. You can use any algorithm you want to set p in these functions. 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. **(10 points)**
Now Answer the following questions:

- Run your code with retry enabled, nodes= 8, pmin= 0 pmax= 1. Would you recommend running a real network with these parameters for pmin and pmax? Briefly explain your answer. **(2.5 points)**

- 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), set the variables as: retry enabled, nodes= 8, pmin= 0.01, pmax= [*value*]

  For the second case (packet size of 10),set the variables as: retry enabled,packet size = 10, simtime= 70000 nodes= 8, pmin= 0.01, pmax= [*value*]

  (2.5 points)

Carrier Sense Multiple Access (CSMA)(10 points)

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 task, 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 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 packet size 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: **(5 points)**
retry enabled,packet size = 10, simtime= 100000 nodes= 8, pmin= [*value*], pmax= [*value*]
Now Answer the following questions:

- What is the utilization and fairness of your protocol when pmin = 0 and pmax = 1? **(2.5 points)**
- 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.] **(2.5 points)**

CSMA with contention windows, as in WiFi (802.11) and Ethernet(802.3) (25 points)

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, 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 stabilized ALOHA and CSMA.
You will first implement the scheme described thus far. It will turn out not to do as well as we would like, and later, you will fix an important weakness.
Implement this scheme by writing the appropriate code for *channel_access*(), *on_collision*(), and *on_xmit_success*() in MAC notebook. 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 your code with the following parameters:
retry enabled,packet size = 10, simtime= 100000 nodes= 16, cwmin= 1, cwmax= 256.
Running the above, you will note that even with carrier sense being used, the utilization is quite a bit lower than in CSMA.

Answer the following questions:

- 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? **(2.5 points)**

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. **(20 points)**
Modify your code to include the above suggestion.

- Report the utilization and fairness for your scheme after including the suggestion and running the same command as before. **(2.5 points)**

What Should I Do?

You must upload (.cc/.ned/omnetpp.ini/.xml/.py/.ipynb/.pdf) files in a folder named after its section. provide enough screenshots or other useful media to demonstrate your results. Complete the MAC notebook. feel free to duplicate whichever cell needed to execute the requirements. provide a brief explanation of your algorithms in your report. You should also answer the questions in your report.
Compress all files and rename the compressed file to STUDENT_ID_HW3.zip.
If you have any questions regarding the problem statement or understanding the concept, feel free to ask in Telegram.