|  |  |  |
| --- | --- | --- |
|  | **COLLEGE OF ENGINEERING & TECHNOLOGY** | |
| Golden | **Department**  **Lecturer**  **Assoc. Teacher**  **Course Title**  **Course Code** | : Computer Engineering  : Dr. Yehya Z. Mohasseb  : Dr. Nada Mostafa  : Eng. Nourhan Tarek  : Advanced Networks  : CC513  :2022 |

**CSMA/CA project**

**Objective:**

You will need to implement a CSMA/CA protocol on wireless nodes for both the base station and the mobile stations.

Requirements

1. Your final implementation should produce results similar to the applet from the Kurose & Ross website called "802.11 CSMA/CA with Hidden Terminals (Chapter 6)": https://gaia.cs.umass.edu/kurose\_ross/interactive/collisions.php
2. Their should be separate code files for MobileStation, BaseStation.
3. BaseStation file should implement two functions: activeAction and receptionAction.
   1. These two functions implement the CSMA/CA protocol at the base station.
   2. The activeAction function keeps track of the state of the base station and it makes calls to receptionAction which is another state machine that handles the reception of packets.
4. MobileStation file should implement three functions: activeAction, reception Action, and BEB.
   1. activeAction and reception Action serve the same purpose as the functions in BaseStation, namely maintaining the state of the device and handling packet receptions.
   2. BEB is the exponential backoff which will be called by activeAction and receptionAction when necessary.
5. The receptionAction method (BaseStation) needs to handle incoming packets to the node. If the node is idle, you should check that the packet in not corrupted and the packet type is RTS. If this is so, set the currentSender to the owner of the incoming packet and change state to SIFS\_before\_emitCTS. The other option is that we are currently receiving a packet and should check for a corrupted packet and the current owner should be the current packet's owner and the packet type should be PKT. If this is so, change state to SIFS\_before\_emitACK. If not, go back to idle and remove current sender.
6. Appropriate delays before each case are allowed to proceed:
   1. SIFS\_before\_emitCTS: 3
   2. emitCTS: 15
   3. SIFS\_beforercvPKT: 3
   4. rcvPKT: 61
   5. SIFS\_before\_emitACk: 3
   6. emitACK: 10
7. The activeAction (MobileStation):

If the node is in an idle state and there are packets to send. Check to see if the channel is busy and that nav is not zero and if either fails, perform a random backoff. So long as there are packets to send, change to the DIFS\_beforeCountdown state. Once in the DIFS\_beforeCountdown state, see if enough time has elapsed before moving along to the next state. You should also check if either the channel is busy or nav is not zero and if so, change to the DIFS\_beforeCountdownState. Within the coundown state, if enough time has elapsed, decrement the backoff value (backoffValue) and change to the current state. You should also check if either the channel is busy or nav is not zero and if so, change to the countdown state. Case emitRTS will wait for its time and make a call to receptionAction to send the packet and change to a new state. SIFS\_before\_rcvCTS only needs to wait and then change to the next state. rcvCTS will wait then call the random backoff before going back to the DIFS\_beforeCountdown state. SIFS\_before\_rcvPKT will simple wait then change to the next state. emitPKT is responsible for transmitting the packet after the required wait then moving on to the SIFS\_before\_rcvACk state. This state simply waits and then moves on. rcvACK will wait and then perform a random backoff before going back to DIFS\_beforeCountdown.

1. MobileStation: Appropriate delays before parts of each case are allowed to proceed:
   1. DIFS\_beforeCountdown: 5
   2. Countdown: 5
   3. emitRTS: 15
   4. SIFS\_before\_revCTS: 3
   5. rcvCTS: 16
   6. SIFS\_before\_emitPKT: 3
   7. emitPKT: 60
   8. SIFS\_before\_rcvACK: 3
   9. rcvACK: 11
2. receptionAction: The idle, DIFS\_beforeCountdown, and countdown states all do the same thing. They check for a corrupted packet and that packet.getNav() is larger than the nav variable. If so, set nav to the packet's nav. If the state is rcvCTS then check for a corrupted packet and the owner is the current node (this) and the packet type is CTS. If so, change state to SIFS\_before\_emitPKT. If something fails, backoff and change to state DIFS\_beforeCountdown. The last case that must be handled is rcvACK, so check for a corrupted packet and the owner is the current node (this) and the packet type is ACK. If so, decrement (packetToSend) because we are done and if there are more packets to send, set CWorder to 3 and backoff randomly before changing to the DIFS\_beforeCountdown state. If there are no more packets to transmit, go to the idle state. If something was wrong, backoff and change to the DIFS\_beforeCountdown state

**Deliverables:**

1. You should hand in all of the files in a zipped folder, along with some documentation to help the instructor understand your code.
2. The files should compile with the standard java compiler (any other programming language is also aceeptable).
3. For the General implementation:
   1. activeAction() is called every 100 ms. This should handle most state changes, sending packets, and timeouts.
   2. receptionAction(packet) is called whenever a packet is received. It must handle responding to packets by changing state. It should not be sending any packets.
   3. BEB() rnadomly backs off (sets backoffValue).
   4. State is a class (that should be an enum) simply used to define various possible states.
   5. elapsedTime(i) will return true if i ticks (roughly 100 ms) have passed since the last change of state.
   6. changeState(s) changes the state to s, and does some other important things, like setting the start time of the state and setting up emmitedPacket. Always use this to change your state.
   7. myChannel.getBusy() returns true if the channel for this station is busy.
   8. myChannel.reception Action(emmitedPacket) sends out emmitedPacket. This should only be done in "emit" states (emitRTS, emitCTS, emitPKT, emitACK). If you are in the right state, and changed to there with changeState, you will send out the correct packet type.
   9. packet.getCorrupted(this) returns true if a packet is unreadable at this terminal. Corruption happens if there is interference from other terminals. You should only need this in receiveAction.
   10. packet.getType() returns an int that represents the type of the packet (ACK, CTS, RTS, PKT). Compare it to Packet.RTS, Packet.CTS, etc. to determine the type.
   11. packet.getOwner() returns the MobileStation associated with this packet. For a CTS, this is the MobileStation being allowed through. Otherwise, it's the intended recipient.
   12. packet.getNav() returns the Network Allocation Vector (NAV) of the packet. NAV is an estimate of how long the channel will be busy.
4. For BaseStation:
   1. currentSender is used to keep track of the MobileStation you are currently communicating with. You must set this when you send CTS, and check against it when receiving PKT with packet.getOwner().
   2. activeAction() should check the state, and act accordingly. -- If it's idle, do nothing. -- If it's SIFS\_before\_emitCTS, delay (use if (elapsedTime(i)) with the value given in the project) and then change to emitCTS. -- If it's emitCTS, delay, send your CTS and change to SIFS\_before\_revPKT. -- If it's SIFS\_before\_rcvPKT, delay and then change to revPKT. -- If it's revPKT, delay and then change to idle -- If the state is SIFS\_before\_emitACK, delay and change to emitACK. -- If it's emitACK, send out the ACK (just like in emitCTS) and change to idle.
   3. receptionAction(packet) handles any packets that are received. There are only two states where Base Station will accept packets. -- If the state is idle, check that the packet is not corrupted and is an RTS packet . If so, set curSender and change to SIFS\_before\_emitCTS. -- If the state is revPKT, you follow a very similar process. Check that the packet is not corrupted and that the packet is a PKT packet and that the packet's owner is curSender. If so, change to SIFS\_before\_emitACK.
5. For MobileStation:
6. packetToSend is incremented by hitting the button for this station. It keeps track of how many packets are queued up to be sent.
7. BackOffCount is used by BEB(). It's used to keep track of how many times this station has had to backoff for this packet.
8. backoffValue is how long we're waiting to try to send again.
9. nav is the current Network Allocation Vector. It is an approximation of how long the channel is going to be busy.
10. BEB() should set backoffValue to a random integer. However, it should be in the range of 0 to 2^n. n is BackOffCount , and should be incremented each time you have to back off. This way, if you have to back off a lot, due to congestion, you will wait longer and longer so that something can get through