

Homework 11, Due Date: 10:00am 5/1/2014, Cutoff Date: 10:00am 5/6/2014**Submission: Upload source code (.java) files to Blackboard, and java programs (.class files) to EC2 instances**

Part I: Write a *Host* program and a *Channel* program to simulate how the CSMA/CD algorithm on a host eventually sends one frame successfully. The channel program is used to simulate the shared “channel” that provides information to NIC. (**Hint:** You may use the `Timer` and `TimerTask` classes for scheduling a task to be performed later. See <http://enos.itcollege.ee/~jpoial/docs/tutorial/essential/threads/timer.html> and JavaDoc. Or feel free to use any class to implement such functions.)

- All inputs/outputs must be read from/written to the **command-line** standard input/output console.
- The *Channel* program needs to support multiple *Host* programs. The *Host* program needs to take the dns/ip of the machine where the *Channel* program runs as the argument and use it later for the communication with the *Channel* program on top of UDP (recommended, since single thread may be used for easily sharing data among hosts) or TCP (multithreads have to be used, and sharing data among threads is a little tricky).
- Your *Host* program needs to
 1. Display prompt messages to collect two user inputs (to keep it simple, let first input ≥ 2 and second input $\geq 10 \times v$, where v is defined in Step 1 for your *Channel* program):
 - How many seconds from now until this frame is ready to be sent out?
 - How long does it take for the host to transmit this frame entirely? (in seconds)
 2. Send a “SIMSTART” message to the *Channel* program, wait for the response. Once receiving the response, report an error and terminate if the response is not “YESSS”. Otherwise, display the response and move on to Step 3.
 3. Initialize the value of a variable for *total-number-of-collisions-so-far* as **zero**.
 4. Use a timer to schedule a *sensing* task to be performed x seconds later, where x is the first user input collected in Step 1.
 5. Perform the *sensing* task scheduled in Step 4, Step 5.1, or Step 6.2.4. Display the following message on the standard output console, send a message “IDLE” to the *Channel* program, and wait for the response from the *Channel* program.
 - NIC senses channel to see whether the channel is idle. Current time is ...
 - 5.1 If the *Channel* program responds “NO”, get the current time in millisecond, display the following message, and use the timer to schedule a new *sensing* task to be performed **1** second later.
 - The channel is busy now. Current time is ...
 - 5.2 If the *Channel* program responds “YES”,
 - 5.2.1 Initialize the value of a variable for *leftover-transmitting-duration* as y seconds, where y is the second user input collected in Step 1.
 - 5.2.2 Send a message “START” to the *Channel* program.
 - 5.2.3 Display a message, “NIC starts transmitting a frame. Current time is ... The leftover-transmission-duration is z seconds.”, where z is the current value of the variable for *leftover-transmitting-duration*.
 - 5.2.4 Use the timer to schedule a *collision-detecting* task to be performed **1** second later.
 6. Perform the *collision-detecting* task scheduled in Step 5.2.4 or Step 6.1.2.3. Display the following message on the standard output console, send a message “COLIDE” to the *Channel* program, and wait for the response.
 - NIC detects collision on channel. Current time is ...
 - 6.1 If the *Channel* program responds “NO”,
 - 6.1.1 If the value of the variable for *leftover-transmitting-duration* (See Step 5.2.1) is **zero**, send a message “DONE” to the *Channel* program, display “Done with transmitting this frame! Current time is ...”, and stop this program.
 - 6.1.2 If the value of the variable for *leftover-transmitting-duration* is **positive**,
 - 6.1.2.1 Subtract $w = \min(1, \text{the value of the variable for } leftover\text{-transmitting-duration})$ seconds from the value of the variable for *leftover-transmitting-duration*,
 - 6.1.2.2 Display a message, “NIC is still transmitting. The leftover-transmission-duration is z sec. Current time is ...”, where z is the current value of the variable for *leftover-transmitting-duration*.
 - 6.1.2.3 Use the timer to schedule a *collision-detecting* task to be performed w (See 6.1.2.1) seconds later.
 - 6.2 If the *Channel* program responds “YES”,
 - 6.2.1 Add **1** to the value of the variable for *total-number-of-collisions-so-far* (see Step 3)
 - 6.2.2 Use binary backoff to determine $k \times y$, where y is the second user input collected in Step 1, k is randomly chosen from $\{0, 1, 2, \dots, 2^m - 1\}$, and m is the current value of the variable for *total-number-of-collisions-so-far*.
 - 6.2.3 Display a message, “NIC detects a collision and aborts transmitting the frame and will sense the channel for re-transmission $k \times y$ seconds later. Local time is ...”, where $k \times y$ is the product calculated in Step 6.2.2.
 - 6.2.4 Send a message “ABORT” to the *Channel* program.
 - 6.2.5 Use a timer to schedule a *sensing* task to be performed $k \times y$ seconds later.
- Your *Channel* program needs to
 1. Take only **ONE** user input as below for collision detection, which is assumed to be much smaller than y .
 - What is the end-to-end propagation delay (in seconds) over the channel?

Let's record this user input as ν and use it for possible data collision. For example, if $\nu = 2.8$, when Host A starts transmitting its frame at time 15 seconds and Host B sends "IDLE" to Channel at time 17 seconds, Channel responds "YES". Then when Host B sends "COLIDE" to Channel at time 18 seconds, Channel responds "YES".

2. Create/Maintain a list/array of host objects. Each object is for a host program that *has communicated* with the Channel program. Each host object has the following fields:

```
String hostIP; //ip of the machine where this host program runs, initial value: NULL
int port;      //port number from which the host program sends the "SIMSTART" message
long t_start;  //time when this host starts transmitting, initial value: -1
long t_stop;   //time when this host aborts/is done with transmitting, initial value: -1
```

3. Display the following prompt message to take one user input,

How many hosts are there on this channel?

Let's record this user input as `numHosts`.

Use an infinite loop to keep waiting for a "SIMSTART" message from each host. AFTER receiving the "SIMSTART" message from ALL `numHosts` hosts, send out a "YESSS" message to EACH host program using the ip & port recorded in each host object. Then move on to Step 4.

4. Use an infinite loop to keep waiting for a message from any host. Whenever it receives a message, set `t_now` as the current time in milliseconds. Display the received message on the standard output console. (*: if UDP is used and a new socket is created by the host program for sending a message, the port number in the currently received message, instead of the port number recorded in the host object, should be used for the following response.) If the message is

- 4.1 a "IDLE" message,

4.1.1 For all host objects in the list/array, if `t_start < 0` or
 $((t_start + 1000 * \nu) > t_now)$ and $((t_stop < 0)$ or $((t_stop + 1000 * \nu) \leq t_now)))$ or
 $((t_start \leq t_stop)$ and $((t_stop + 1000 * \nu) \leq t_now)))$, respond* "YES".

4.1.2 Otherwise, respond* "NO".

- 4.2 a "COLIDE" message,

4.2.1 For all host objects in the list/array **except for** the sender, if `t_start < 0` or
 $((t_start + 1000 * \nu) > t_now)$ and $((t_stop < 0)$ or $((t_stop + 1000 * \nu) \leq t_now)))$ or
 $((t_start \leq t_stop)$ and $((t_stop + 1000 * \nu) \leq t_now)))$, respond* "NO".

4.2.2 Otherwise, respond* "YES".

- 4.3 a "START" message,

4.3.1 Look for the host object in the list/array whose `hostIP` is the ip of the sender of this message. If not found, create a new host object for this sender using its ip and insert it into the list/array.

4.3.2 set `t_start` of the host object for this sender as `t_now`

- 4.4 a "DONE" or "ABORT" message,

4.4.1 Look for the host object in the list/array whose `hostIP` is the ip of the sender of this message. If not found, create a new host object for this sender using its ip and insert it into the list/array (this may occur with a very small chance since UDP packets may be lost or arrive out of order.)


4.4.2 set `t_stop` of the host object for this sender as `t_now`.

and display each message, hostIP, and port.

together with sender's hostIP and port of this message.

Part II: Test your programs with multiple clients using the Amazon EC2 Cloud

1. Make a directory "HW11" under your home directory on your EC2 instances.
2. Upload the Channel program under "HW11" on your instance in N. Virginia and the Host program under "HW11" on your instance in Sydney.
3. Run the Channel program. Meanwhile, run the Host program on your computer or lab computer and the Host program on your EC2 instance to test all the possible cases.

 Please STOP your ec2 instances whenever you are NOT working on them. Please do NOT start or stop ec2 instances whose names do not contain your user name. Thank you!