Program 5 64 Points

Due: February 27, 2008 at 11:59 p.m. {modified February 15, 2008}

A Simulation of MANET Source Routing in C++

Note: There is NO late period partial credit points for this assignment!!!

This assignment provides the students with the opportunity to implement a fairly large and complex C++ program that includes data structures while reusing components from previous programming assignments.

The goal of this program is to develop an event-driven simulator to model the transmission of network packets sent by S senders to R receivers over a simulated MANET (Mobile Adhoc wireless NETwork). The MANET contains M mobile 'data mules' that use Source Routing to forward packets. **All simulated time is represented in 100 milliseconds units** (referred to as '**sims**').

Command Line Arguments

The manetsim simulator takes five required command line arguments in the form:

```
./manetsim senders receivers mules speed dimension
```

where

- senders indicates S, the number of senders
- receivers indicates R, the number of receivers
- mules indicates M, the number of mules
- speed indicates the simulated speed of **ALL** mules.
- dimension indicates one side of the simulated square space (the Manet map) where mules move.

Note1: All the command line arguments are integers.
Note2: **speed** ,**s**, can be converted to a movement rate, **r**, expressed in the number of sims to move one hop as:

r = infinity for s = 0; {immobile mules}
r = (11 – s) sims/hop 1 ≤ s ≤ 10;

While your simulator design should be as general as possible, to provide concrete examples for the remainder of this description, assume the specific command line:

```
./manet-sim 20 10 24 3 39
```
[1]

This indicates that for this simulator run there are 20 senders, 10 receivers and 24 mules where the mules move over a Manet Map of 39 by 39 spots. The mule movement rate is 8 sims/hop. To facilitate further examples, assume senders are identified as nodes 1-20, receivers as nodes 51-60, and mules as nodes 21-44.

Source Routing (SR)

Source Routing is a network packet routing scheme where each packet sent from a sender or forwarded by an intermediate router (namely, a Manet mule) contains the remainder of the packet path in the header of the arriving packet. Thus an input **SR** of **5 34 27 40 59** indicates packets originating at sender 5 are sent to receiver 59 via a Manet path traversal through mules 34, 27 and 40.

Thus, when a packet leaves sender 5, it contains SR equal to 34 27 40 59.
When the same packet leaves mule 34, it contains SR equal to 27 40 59.
When the same packet leaves mule 27, it contains SR equal to 40 59.
When the same packet leaves mule 40, it contains SR equal to 59.
When the packet finally arrives at its destination, receiver 59, it contains SR equal to 59.

Simulated Node Roles

Senders

Senders are immobile wireless nodes located in a column to the West of the Manet map that send packets to receivers through the mule nodes.

For this simplified simulation, packets only come in three sizes (small, median and large) which are identified by the integers 0, 1 and 2, respectively. These sizes magically absorb any overhead including senderID and source route information in the packet header and timestamp in the packet trailer.

Senders transmit fixed size packets with a constant IPG (Inter Packet Gap) of one sim between each packet transmission. A sender records the senderID and the SR in the packet header and the simulated time of the start of a packet transmission time in the timestamp field in the packet trailer.

Receivers

Receivers are immobile wireless nodes located in a column to the East of the Manet map that receive packets sent to them by the senders. When a packet arrives, the receiver computes the end-to-end delay for each packet as:

delay = current time – timestamp (in packet trailer)

Each receiver records the total number of packets received during one simulation run from each sender, the total number of packets received, the mean delay of all packets received from each sender and the mean delay of all packets received.

Delay variance metrics produced by the receivers are optional!

Mules

Mules are mobile wireless nodes in the MANET. All mules move through the Manet map at the same movement rate determined by the speed command line argument. The directional movement of mules is similar but different from Program 2. The eight possible mule movement directions are: East, NorthEast, North, NorthWest, West, SouthWest, South and SouthEast. These directions are identified by the integers 0 to 7, respectively. These directions correspond to the possible movements of each mule on the Manet map. Similar to Program 2, when a mule node attempts to go off the edge of the Manet map, it 'bounces' back in the opposite direction where opposite direction pairs are defined as follows:

[East – West], [NorthEast – SouthWest], [North – South], [NorthWest – SouthEast]

Each mule contains a transmission queue to hold packets awaiting wireless transmission. When a packet arrives at a mule, it is instantly (0 sim time) enqueued in the mule's transmission queue. Note, if the queue is empty, the arriving packet can be immediately be sent by the mule as long as the IPG time has elapsed. Upon packet arrival, the mule removes its node ID from the front of the SR in the packet. Note, the packet size remains fixed.

Main Assignment

1. Simulation Initialization

1a. The simulator uses the command line arguments to control the simulation initialization. The program reads `senders` lines of input from a script file where each input line contains:

```
senderID  arrival_time  packets size  SR
```

where

`senderID` identifies the sender (e.g., for command line [1] above, this is an integer between 1 and 20.)
`arrival_time` is the sender arrival time in sims.
`packets` specifies the number of packets this sender sends during this simulation.
`size` is the integer 0, 1 or 2 to indicate small, medium or large packets.
`SR` specifies the full source route through the MANET (including sender's id at the front)

For example, an input line of:

```
5  35  100  1    5 34 27 40 59
```

indicates that for this simulation, sender 5 starts sending 100 medium packets at 3.5 seconds of simulated time (or at 35 sims). Each of the 100 packets sent by node 5 will be simulated using event driven simulation that models transmitting each packet from node 5 through mule nodes 34, 27 and 40 before being received by node 59, a receiver node. This packet path is simulated by alternating transmission events with enqueueing events that enter the packet on mule transmission queues until the packet reached the receiver.

1b. Senders: Use a random number generator to randomly place each sender in a unique row of a single column of spots directly to the West of the Manet Map. If the random process produces two senders on the same spot, redraw a new random row such that all senders start at unique spots in the West column.

1c. Receivers: Use a random number generator to randomly place each receiver in a unique row of a single column of spots directly to the East of the Manet Map. If the random process produces two receivers on the same spot, redraw a new random row such that all receivers start at unique spots in the East column.

1d. Mules: Use a random number generator to randomly place all the mules on the MANET map. If the random process produces two mules starting on the same spot, redraw new random positions such that all mules start at unique spots.

To simplify this assignment, mules are ONLY simulated in multiples of eight to equalize the movement directions. Thus in command line [1] above, there are 24 mules. This simplifies the assignment of each initial mule direction to be **muleID mod 8** that matches up to the eight possible mule movement directions.

Similar to player movement in program 2, mules bounce in the opposite direction when they attempt to hop off the edge of the Manet map and mules also experience hop boosting. When each mule completes its hop into the next spot on the Manet map (note – the time it takes to hop one spot depends on the mule movement rate), the program must check to see if that spot is currently occupied by another mule. If occupied, the ***hopping mule*** moves again while sim time stands still!! (This effectively causes a ***hyperspace*** hop over any encountered mules in zero sim time). Unlike program 2, there are no mule collisions.

Important Design Decision

While Program 5 combines pieces of Program 2 and Program 3, there is a significant mismatch in that Program 2 was implemented without an Event List. Thus, in designing Program 5 you will have to decide whether to include mule movements as events on the Event List or to always increment the simulated clock to permit handling mule movements with mechanisms developed in Program 2. (This choice will be explained in class.)

The discussion that follows implicitly assumes the second choice.

2. Running the Simulation

Once all the simulation inputs have been processed and all the initial node positions and directions have been recorded, all the mule transmission queues need to be initialized as empty.

The simulation starts by putting the arrival time event of each sender on the Event List.
As time is incremented mule movement is adjusted. Whenever simulated time equals the first event on the event list, this event is processed. For each sender arrival event, an event corresponding to the arrival of its first packet at the next node on the SR is created. Additionally, another event is created one sim later to indicate the time at which the IPG has expired and the next packet from that sender can be transmitted.
[Note – this statement reflects one design choice.] A sender leaves the simulation when all its packets have been transmitted (which is before all its packets have been received.)

When the next event to be processed on the event list is a packet arrival at a mule node, this event triggers enqueueing this packet on the mule's transmission queue. Whenever a packet is enqueued onto an empty mule queue, if the IPG has elapsed, this packet is immediately transmitted by the mule node. The mule then creates an event corresponding to the arrival of this packet at the next node in the SR.

Generally, once a mule completes the creation of an event corresponding to its packet being transmitted, it must then create a check queue event one sim later to correspond to an IPG. When this check queue event occurs, it will cause that mule to check its transmission queue to see if there is another packet to send. If this check queue event finds a non-empty queue, the mule transmits this packet by creating a packet arrival event. If the check queue yields an empty queue, no new event is created but the mule records the fact that an IPG has elapsed.

When the next event to be processed on the event list is a packet arrival at a receiver node, the receiver node extracts the senderID and the timestamp for its required performance calculations. However, no new event is triggered.

Note – There can be chronological ties on the Event List. They can arbitrarily be processed in the order they are placed on the Event List.

3. Simulation Completions

The simulation continues until the event list and all mule transmission queues are empty. This will occur when all the senders have completed transmission of all their packets and all the packets have reached their destination receivers.

At this point, the program should go through all the receivers to calculate and print out individual receiver performance metrics and overall mean delay for all the packets transmitted. The final sim time should also be printed out.

Simulated Wireless Packet Transmissions

Packets are transmitted with an IPG between all transmissions by senders and mules. The mule transmission event is triggered by a packet at the head of a mule's transmission queue. Since mules are in motion in this simulation, the packet transmission time needs to account for the distance between the transmitting node and the receiving node and the packet size at the `current` time . The formula to determine `future`, the simulated time in the future that a packet sent at time `current` is received, is given by:

$$future = size + ceil\left(\log_2 (\text{Euclidean distance}(s\text{-node}, r\text{-node})) \right) + 1 + current$$

where

`size` is the size integer (0,1, or 2) specified for this packet.
`s-node` is the coordinates in spots of the send node (either a sender or a mule).

r-node is the coordinates in spots of the receive node (either a mule or a receiver).

Note: The Euclidean distance calculation assumes both the West column holding senders and the East column holding receivers are one hop away from the edge of the MANET map.

Example:

For mule 24 located at (6,9) transmitting a medium packet (size = 1) to mule 16 located at (20,22) at current time of 20,

future = 1 + 4 + 1 + 20 = 26 sims

Thus, the event created by mule 24 at time 20 to indicate the arrival of the packet at mule 16 occurs at time 26.

Mule Transmission Queues

All the mule transmission queues are to be implemented as FCFS queues. Note, there can be chronological ties in arrival time at the mule transmission queues. Using FCFS permits students to reuse some of the code developed for Program 3.

For advanced programmers who can handle easily the complexity of this assignment, you can earn 10 extra credit points by also implementing the following Priority Queueing scheme at the mule nodes.

Priority Queueing (PQ)

Rather than put all arriving packets of varying sizes (small, medium and large) in one FCFS queue, replace each mule FCFS queue with three Priority queues, one queue for each packet size. Thus, arriving packets are placed in the small, medium or large queues. Mules then give transmission priority to the smallest non-empty queue. Namely, the mule will transmit all packets in the small queue before transmitting any medium packets and transmit all packets in the medium queue before transmitting any large packets

PQ is known to reduce mean delay over FCFS at the expense of increasing unfairness. Hence, if you implement PQ, the extra required output from the receivers is the variance of the mean transmission delay.

Program Output

Your output routines need to be designed to produce line oriented output to be sent to the screen or to an output file for grading analysis.

Given that mules move, your output of the Manet map should be similar to the output from program 2. Namely, output the position of all the nodes at the beginning of the simulation, after every N^{th} sim time period (where N is something reasonable like 50), and output the final position of all nodes when the simulation ends.

Additionally, your final output should include all performance results collected by the receivers.

What to turn in for Program 5

An official test file of sender parameters for one specific set of command line arguments will be made available a few days before the due date. Turn in your assignment using the ***turnin*** program on the CCC machines. You should turn in a tarred file that includes your source code, a make file and a README file. The README file should provide any information to help the TA or SA test your program for grading purpose.