CS2303 Systems Programming Concepts **C14**

Program 5 **64 Points**

Due: Wednesday March 5, 2014 at 5 p.m.

A Simulation of MANET Source Routing in C++

Note: There is only a one day late period for this assignment. Programs turned after 5 p.m. on Thursday, March 6, 2014 will not be graded at all.

This assignment provides the students in two-person teams with the opportunity to implement a relatively large and complex C++ program that includes data structures while reusing components from previous programming assignments and labs. The goal of this program is to develop an event-driven simulator to model the transmission of network packets sent by **S** source nodes to **R** receiver nodes over a simulated **MANET** (**M**obile **A**d hoc wireless **NET**work). The MANET contains **M** mobile 'data mules' that use Source Routing to forward packets. **All simulated time is represented in 100 millisecond units** (referred to as **'sims**').

Command Line Arguments

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

```
./manetsim sources  receivers  mules  dimension
```

where

- sources indicates **S**, the number of sources
- receivers indicates **R**, the number of receivers
- mules indicates **M**, the number of mules
- dimension indicates one side of the simulated square two-dimensional field (the MANET map) where mules move.

Note1: All the command line arguments are integers.

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

```
./manet-sim 20  10  24  30                                    [1]
```

This indicates that for this simulator run there are 20 sources, 10 receivers and 24 mules where the mules move over a MANET map of 30 by 30 spots. Note, for this simple simulation sources only send packets to **one** receiver, but receivers may get packets from multiple sources. To facilitate further examples, assume sources are identified as nodes 1-20, mules as nodes 21-44 and receivers as nodes 45-54.

Source Routing (SR)

Source Routing is a network packet routing scheme where each packet sent from a source 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 source route (**SR)** of **4 24 40 27 52** indicates packets originating at sender 4 are sent to receiver 52 via a MANET path traversal through mules 24, 40 and 27.

Thus, when a packet leaves source 4, it contains SR equal to 24 40 27 52.
When the same packet leaves mule 24, it contains SR equal to 40 27 52.
When the same packet leaves mule 40, it contains SR equal to 27 52.
When the same packet leaves mule 27, it contains SR equal to 52.
When the packet finally arrives at its destination, receiver 52, it contains SR equal to 52.

Notes: You need to retain the original **sourceID** in the transmitted packet for receiver performance calculations. For this assignment, you can assume that the SR fits on one input line (max 80 characters) and the largest number of hops in the route is 15 (i.e., the maximum number of nodes in the route is 16).

Simulated Node Roles

Sources

Sources 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 1, 2 and 3, respectively. These sizes **magically** absorb any overhead including **sourceID** and **SR** in the packet header and **timestamp** in the packet trailer. This magic means your simulation does not account for the header or trailer in transmission time, but your packet data structure needs to keep this information for routing and for performance calculations at the receiver.

Each source transmits fixed size packets (as specified). A source records the **sourceID** and the **modified SR** in the packet header and the simulated time of the start of a packet's transmission time at the source in the timestamp field in the packet trailer.

The simulated time between initiating sender transmissions is equal to the packet size **(i.e., assume a very small IPG (inter packet gap) that is simulated as zero time).** Sender nodes (sources and mules) are busy during packet transmissions but NOT during packet propagations [read carefully this discussion later in this document!!]

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 sources. When a packet arrives at the receiver, 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 from each source**, **the total number of packets received**, **the mean delay of all packets received from each source** and **the mean delay of all packets received**.

Delay variance per sender produced by each receiver is an optional receiver calculation **(and worth 1 extra credit point if correct)**.

Mules

Mules are **mobile** wireless nodes in the MANET. All mules move through the MANET map at the rate of one hop per second (namely, it takes 10 **sims** to complete a hop). All mules will be positioned at time 0 and subsequently their position changes occur at the **end** of every simulated second.

The directional movement of mules is similar to Program 2. The four possible mule movement directions are: East, South, West and North. These directions are identified by the integers 0 to 3, 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 hop off the edge of the MANET map, it 'bounces' back in the opposite direction where opposite direction pairs are defined as follows:

[East <–> West] and [South <–> North]

Each mule has a **packet transmission queue** that hold packets awaiting wireless transmission by that mule. When a packet arrives at a mule, it is instantly enqueued in the mule's transmission queue (i.e., it takes 0 sims to enqueue a packet in the transmission queue of a mule). Note, if a mule's queue is empty at packet arrival time and the mule is not busy (i.e, no packet is currently being transmitted by that mule), the arriving packet is immediately sent by the mule. As with source transmissions, the time it takes to get the packet out of the transmission queue is equal to the **transmission time** (discussed later). Upon packet arrival at the next hop (at the end of propagation time), the receiving mule removes its **node ID** from the front of the **SR** in the packet and enqueues the packet in its transmission queue. Note, packet size remains **fixed** even though the packet header gets smaller as the packet traverses the mules.

Main Assignment

1. Simulation Initialization

1a. The MANET simulator uses the command line arguments to control the **simulation initialization**. The program reads `one` line of input for each source node (`sources`) from a script file where each input line contains:

```
sourceID  arrival_time  packets pkt_size SR_size SR
```

where

`sourceID` identifies the original source (e.g., for command line [1] above, this is an integer between 1 and 20.)

arrival_time is the source's arrival time to the simulation in sims.
packets specifies the number of packets this source node sends during this simulation.
pkt_size is the integer 1, 2 or 3 to indicate small, medium or large packets.
SR_size is the size of the source route that follows.
SR specifies the full source route through the MANET (including the sourceID at the front of the **SR**).

For example, an input line of:

```
4 35 100 2 5 4 24 40 27 52
```

indicates that for this simulation, source 4 starts sending 100 medium packets at 3.5 seconds of simulated time (or at 35 sims). Each of the 100 packets sent by node 4 to node 52 will be simulated using event driven simulation that models transmitting each packet from node 4 through mule nodes 24, 40 and 27 before being received by node 52, a receiver node. This packet path of 5 nodes is simulated by handling transmission events (packet transmissions and packet arrivals).

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 a new random position such that all mules start at unique spots.

To simplify this assignment, mules are **ONLY** simulated in multiples of **four** to equalize and simplify 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 4** that matches up to the four 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 hop time), 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 that cause mules to die!

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 should combine your previous code such that the simulation always takes events off the front of the event list until the list is empty. In this case

simulated time will jump over **sim** times when no events are simulated to occur and several events on the event list will occur at the same sim time. Once time is moved up, the simulator then goes through a rotation of all events that occur at that time.

To keep everyone's simulator similar, the last thing to happen at sim time, **t, is if **t** is on a **second** (or 10 **sims**) boundary, then all the mules will complete one move. For example, if the simulator reaches sim time 50 (5 seconds or 50 milliseconds), it must process all possible events that occur at time 50 first, then cause all the mules to complete a single move and finally record the new location of all the mules due to completion of a hop.

Choices in your major design decision will be discussed in class.

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 to empty and/or idle.

The simulation starts by recording the arrival time event of each source on the Event List. Whenever simulated time equals the first event on the event list, this event is processed. For each source arrival event, an event corresponding to the arrival of its first packet at the next node on the SR is created and as long as the source still has packets to be sent, an additional event to represent the end of the packet's transmission. Thus, the end of a packet transmission and an arrival of a packet at a receiver are simulated as two distinct events.

When a packet transmission ends, the sender node (source or mule) is no longer busy and it can now begin a new packet transmission.

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 that mule's transmission queue. Whenever a packet is placed on an empty mule queue and the mule is not busy, this packet is immediately transmitted by the mule node and the event list is updated accordingly.

All senders (mules and sources) need to create events for packet arrivals which indicate when the transmitted packet will arrive at the next node in the **SR** and remain in the busy state during the packet transmission interval.

When the next event to be processed on the event list is a packet arrival at a receiver node, the receiver node extracts the sourceID and the timestamp for its required performance calculations. However, **no new event is triggered** and appropriate data structures need to be deleted.

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.

A source leaves the simulation when its last packet is sent (which is before all its packets have been received.)

3. Simulation Completion

The simulation continues until the Event List is empty. This will occur when all the sources 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 and the total number of events processed during the simulation should also be printed out.

Simulating Wireless Packet Transmissions

Packet transmissions require time at the sender (**transmission time**) and time in the air (**propagation time**). These should be modeled as two separate events in the simulator.

Since mules are in motion in this simulation, the packet arrival time at the next node needs to account for the distance between the transmitting node (s_node) and the receiving node (r_node) and the packet size at the current time. The formula to determine arrival_time, the simulated time in the future that a packet sent at time current arrives at a mule or a receiver, is given by:

arrival_time = current + transmission time + propagation time

where

transmission time = pkt_size

propagation time = ceil ($\log_2$ (Euclidean distance(s-node, r-node)))

and

pkt_size is the size integer (1,2, or 3) specified for this packet.
s-node is the coordinates of the sender node at the current time (either a source or a mule).
r-node is the coordinates in spots of the receive node at the current time (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 **immediately next** to the edge of the MANET map.

Example:

For mule 24 located at (6,9) transmitting a medium packet (size = 2) to mule 16 located at (20,22) at current time of 20,

arrival_time = 20 + 2 + 4 = 26 sims

Thus, mule 24 at time 20 creates an end of packet transmission event at time 22 and a packet arrival event at mule 16 to at time 26. Additionally, sending nodes (sources and mules) remain busy until the end of the packet transmission time. Any subsequent packets arriving at mule 24 at time 20 or 21 can be enqueued but they cannot be sent until the mule is not busy at time 22.

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 after it is converted to C++.

For advanced programmers who can handle well the complexity of this assignment, you can earn 10 extra credit points by **additionally** 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 log file for grading analysis. This log file should include one line for each event that occurs during the simulation. 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 (time 0), and after every 10 seconds of simulated time 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. NOTE – as this program comes at the end of the course we will accept programs that do NOT compile for this assignment. In this case, your README file needs to fully discuss what you believe is working and where your current problems are with program 5. Honesty in your README is important to maximizing any partial credit you will receive for a program 5 that only partially works.