Computer Networks I
Programming Assignment 3

Due: 30 November 2016

Overview

This project requires you to design and implement a *discrete event network simulator*, to simulate the functionality of a real computer network. A discrete event simulator runs using a global virtual clock. The simulation advances through time by handling pending events at their scheduled times. Each scheduled event gives rise to other events which are scheduled in the future.

Project detail

Your simulator should have the following functionality:

1. The ability to *randomly* generate a network graph of 150 nodes. Each node should be capable of acting as either a flow source, flow destination, or forwarding router.
2. The generated graph should be *connected*; i.e., each node is capable of finding a path to each other node. You can easily check the connectivity of the graph using a depth-first search (DFS), however this is not the only possible solution.
3. Link bandwidth should be uniformly distributed between 0 and 1 Mbps, i.e. for a link (u, v), the bandwidth $b(u, v) \sim \mathcal{U}(0, 1 \text{ Mbps})$.
4. Link propagation delay should be uniformly distributed between 1 and 10 seconds; i.e., $d(u, v) \sim \mathcal{U}(1, 10 \text{ s})$.
5. Each source should generate packets according to the Poisson distribution with mean $\lambda = 0.5$. This is equivalent to the time between packets being determined by the exponential distribution with the same parameter λ, thus its mean $\mu = 1/\lambda = 2$.
6. The size of a packet i, $S(i)$, should be uniformly distributed between 0.1 and 1.0 Mb. Assume packets can be generated instantaneously.
7. The incoming packets at each router are stored in an input queue for the interface they were received on; the packet at the head of the queue is serviced with a processing delay given by the exponential distribution of mean $\mu = 1.0$. Assume the queuing discipline is first-come-first-serve (FCFS), and a maximum of 30 packets can be present in the queue.
8. Each router maintains an output queue for each interface; packets are placed in these queues after being processed and removed from the input queue. A packet will remain in the output queue until it can be transmitted on the outgoing link. The output queues should also be FCFS, with a maximum size of 30 packets.
9. Each router should maintain a routing table. The routing tables can be generated by running a global-state shortest-path algorithm (e.g. Floyd-Warshall, Dijkstra) prior to beginning the simulation. Each router will need to remember the best outgoing interface for each destination node. The network can be assumed to be static; i.e., the routing table will only need to be generated once.

Your simulator and graph generator should be invokable as separate programs.

Graph generator details

The graph generator should always generate a network of 150 nodes, with a randomly-determined number of edges. The network it generates should always be connected (see above). The output of the graph generator should be saved to a file. The first line of the output file should specify the number of nodes and the number of edges in the network. The following lines will define the edges by identifying the nodes that they connect.

For example:

```
5 4
0 1
1 2
2 3
3 4
```

This file specifies a graph with 5 nodes and 4 edges; the edges are $(0, 1)$, $(1, 2)$, $(2, 3)$, and $(3, 4)$.

Network simulator details

Your simulator should accept two command-line parameters: (1) the input graph filename, and (2) a random seed. The random seed is an arbitrary integer which determines the results of your random number generator. Invoking the program with the same seed should always result in the same random numbers being generated. See `man 3 srand` for details.

Once you have loaded the graph, you will randomly select 20 source-destination pairs for your network flows. Then, you will simulate 1000 seconds of network activity. You should collect the following statistics throughout that time:

- The total number of packets generated by sources.
- The total number of packets which reached their destinations successfully.
- The overall average total delay for a packet (measured from source to destination).
- The average delay of the packets within each flow (per source-destination pair).
- The packet loss rate, as a percentage, for each source-destination pair.
- The maximum, minimum, and average number of packets dropped per router.

The statistics should be printed to the console at the end of the simulation.

Other specific requirements

- Use either C or C++ for this assignment.
- Your code must compile and run on the machines in the CS department labs.
- Code should be modular, and must follow the `.h`/`.c` modularization standards and naming conventions discussed in class.
- You must comment your code so that it is understandable. If your code is cryptic and undocumented, points will be deducted.

- Include a Readme which describes how to use your program, and a Makefile to compile your code. The Readme should also explain any interesting design or implementation decisions you have made.
- Do not use any external libraries in your implementation (only the standard C/C++ libraries and basic system libraries). If you are unsure if a particular library is allowed, do not hesitate to ask.
- Test your code throughly. It is also a good idea to run it under `valgrind` to ensure it is free of memory leaks.
- You are to submit a tarball (.tar, .tgz) to Canvas including your code, the Readme, and the Makefile. Name the file using your own full name and the suffix `_Assignment1` (e.g. `JayMisra_Assignment1.tgz`). If you are working as a pair, include both of your names in the filename.
- Do not include the executables in your submission.

Hints

- A standard uniform variable $u \sim \mathcal{U}(0,1)$ can be generated as `(float) rand() / RAND_MAX`.
- A non-standard uniform RV, $v \sim \mathcal{U}(a,b)$, can be generated as `v = a + u*(b-a)`, where $u \sim \mathcal{U}(0,1)$.
- An exponential RV x with parameter λ (mean $1/\lambda$) can be generated as `x = -(1/λ)*log(u)`, where $u \sim \mathcal{U}(0,1)$.
- The simulator is called *discrete* because events occur at discrete points in time. This means there is some quantum of time specifying the resolution of the simulator. The entire simulation can be thought of as a loop over these time quanta, with the loop body containing all of the network logic. The loop begins at time $t = 0$, and advances from there. At each iteration, some events may occur and some new events may be scheduled.
- Your simulator is required to have at least 1-second resolution. However, you may be more precise – you can implement millisecond or microsecond resolution if you would like. In these cases, there will be many time slots (or quanta) which do not have any events scheduled. Allocating memory for each quantum would be very inefficient. Therefore, you might consider implementing a scheduler based on a min-heap.
- An object-oriented design will be very helpful in this project. Note that this does not mean you have to use C++. You can also implement your object-oriented design in plain C.
- Each node object will have an input queue and an output queue for each of its interfaces; thus, you can implement the queues at each node as a set of two two-dimensional arrays. Each node also has a routing table which it will reference when deciding on which interface to forward a packet. Think about what other elements a node object might encapsulate.
- Packets will spend a lot of their time in queues. However, you also have to simulate their transmission over the wire. Some amount of time must elapse between the packet leaving an output queue and arriving at the next input queue. Think about how you might implement this. One way is to store these "on-the-wire" packets in some sort of global data structure until the event occurs which moves them to an input queue.

Grading rubric

Design, delivery, and style		**40 points**
Modularization	Code is modularized according to the standards described in class and used in common practice.	5
Documentation	Code is thoroughly commented. Each method has a comment to describe its functionality and what inputs are expected. More complicated functions have comments throughout to explain the logic.	5
Readme	The Readme explains how to use the programs, and explains major decisions involved in the programs' designs and implementations.	10
Makefile	The Makefile compiles both executables without error.	10
Organization	The code is organized coherently with clear separation of concerns. (*Note: While Modularization refers to the physical partitioning of source code, Organization refers more to the overall design of the program.*)	10

Program functionality		**60 points**
Graph generation	Your program is capable of generating a random network compliant with the requirements set forth above. You store the graph in the specified output format, and you are capable of loading the stored graph back into the simulator.	15
Forwarding behavior	Nodes handle packets correctly. Packets are queued on the input queue, processed, queued on the output queue, then transmitted according to the specifications.	25
Routing behavior	Routing tables are generated, and the correct forwarding decisions are made at each hop.	10
Statistics	The simulator outputs the desired statistics on delay and loss.	10