ECE4893/8893 **Fall Semester, 2012**

Network Simulation

Assigned: Nov 15, 2012 *Due: Nov 30, 2012, 11:59pm*

Introduction In this assignment we will use create a *Discrete Event Simulation* of a simple local area network consisting of end systems (desktops or laptops), routers, network interfaces, network links, queues, and applications that generate network packets. Since we can develop and debug several of the needed classes simultaneously, we will work in teams of 3 or 4 for this project. Once we have created all of the necessary simulation models, we will evaluate the perfomance of our network with several simulation experiments varying traffic intensity, link bandwidth, and queuing parameters.

Classes Needed You will need to create C++ classes for the following models to complete the assignment.

1. `Simulator`. The `Simulator` object maintains the sorted pending event list and a list of all `Node` objects in the simulation. Further, the `Simulator` needs to add `Stop`, `Run`, `StopAt`, `Schedule` and `ComputeRoutes` functions. There is a single instance of the `Simulator` object in every simulation.

2. `Node`. The `Node` object represents a single network node, either an end sysem or a router (it turns out the needed functionality is identical in either case). The `Node` object maintains a list of associated `NetworkInterface` objects, a list of associated `Application` objects, and a representation of the node's address. We will use a simple 32–bit integer for node addressing. It must implement a `Send` and a `Receive` function to model sending and receiving packets. It must implement `AddNeighbor` function to add another network interface and a point to point link to construct the topology. Nodes also need a routing table, discussed later.

3. `NetworkInterface`. A `NetworkInterface` represents a single *Point to Point* network interface. It models a single `NetworkLink` object and a single `Queue` object to queue packets. It also maintains a *busy* flag, indicating whether the interface is busy sending a packet or not. It must implement a `Send` and a `Receive` function to model sending and receiving packets. Finally it needs a pointer to the associated network node containing the interface.

4. `NetworkLink`. A `NetworkLink` represents a single network "wire" connecting two network interfaces together. It must have two `Node` pointers representing the two ends of the point to point link, as well as a `bandwidth` (bits per second) and `delay` (seconds) to model the link's characteristics.

5. `Packet`. A `Packet` represents bytes being transmitted on the network. Your `Packet` model needs a size (length in bytes) a *source* identifier (indicating the packet's origin node address, and a *destination* indicating the ultimate destination address of the packet.

6. `Queue` and `DropTailQueue` model basic network queuing methods. Createn abstract base class `Queue` with pure functions `Enqueue` and `Dequeue`. `Enque` returns a boolean indicating the packet was successfully enqueued, and `Dequeue` returns a packet pointer of the packet being dequeued. The queue class also needs `MaxSize` and `CurrentSize` member functions indicating the maximum queue length and current queue length respectively. The `DropTailQueue` is a simple first-in first-out queue with a hard limit that drops packets if the queue is full (ie. the size has reached the maximum size).

7. `Application`, `OnOffApplication`, and `TrafficSyncApplication` model network appliations that generate network traffic and receive network traffic. The abstract base class `Application` has `Start` and `Stop` pure virtual functions. The `OnOffApplication` creates and sends packets addressed to a specified peer node at a constant rate when in the "on" state. The `TrafficSyncApplication` simply counts the number of packets received for performance metrics.

Memory Management. One of the hardest things to get right in any simulation environment, and in network simulation in particular is to avoid memory leaks. To keep the project relatively simple, we will ignore this issue completely.

Running the Simulation. Once we have all of the necessary models, we will run several simulations to measure network performance on a simple subnetwork under a variety of load conditions. The steps are as follows:

1. First, construct the topology shown in the figure below. For all links excepting those connecting the G nodes assume the bandwidth is 10 Mbps (10 mega bits per second) and the speed-of-light delay is 1ms (one millisecond). For the links connecting G nodes assume 100Mbps and 10ms repectively.

2. Next add a `OnOffApplication` to every host node (those denoted with an H in the title). The `OnOffApplication` chooses a random value in the range 0 to 1 second, and sends data at a constant rate during that interval. Then it chooses another ramdom valu in the same range and remains silent during that interval. The on and off cycles repeat indefinitely. The data generation rate of the application is a parameter (member variable) that specifies how much data (bits per second) are generated during the on cycle. This will be set to ρ times 10Mbps, where ρ is a value between zero and one. The default value of ρ is 0.5. The purpose of ρ is discussed below. For each application, choose a single randomly chosen destination and address all packets to that destination. Be sure the random destination is not the same as the origin (don't send packets to yourself!). Code for the `OnOffApplication` will be provided for you.

3. Next add a `PacketSynkApplication` to every host node. The packet sync applications simply count the number of recieved packets.

4. Schedule the `Start` event for all applications at a random time between 0 and 1 second.

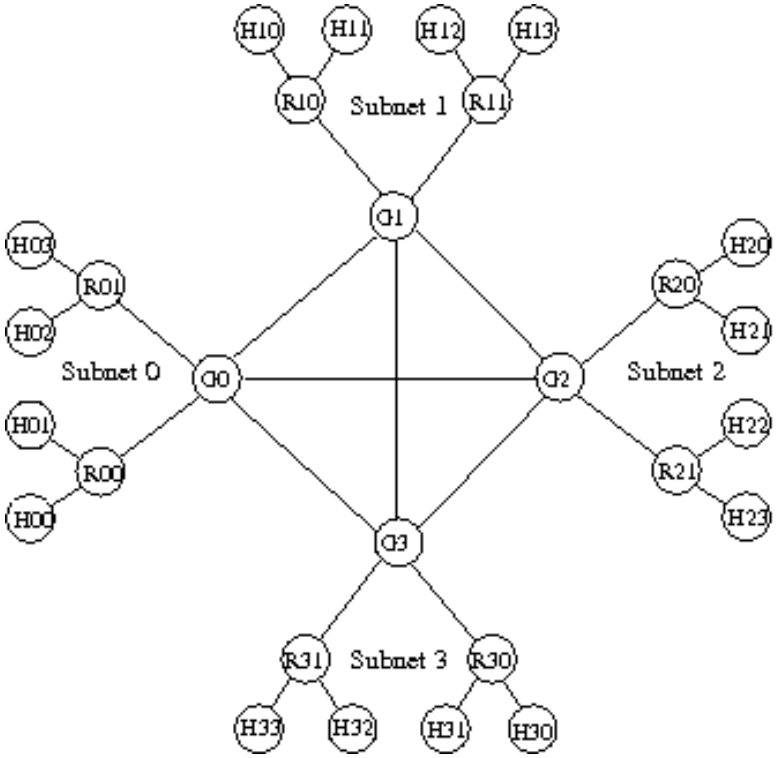
5. Create *Routing Tables* for each node. A routing table specifies the next-hop information for packets for each possible destination. Code will be provided for you to do this.

6. Run the simulation for 100 seconds of simulation time.

7. Report the total number of packets sent and total received, and compute the *Network Efficiency* (total received divided by total sent).

8. The ρ value mentioned above is called the "traffic intensity". It is a measure of the total demand (total amount of data created by the applications) divided by the capacity of the slowest link on the path. In general a ρ value of more than 0.6 or 0.7 will lead to network instability and considerable packet loss and queuing delay.

9. GRAD STudents, run the experiment 10 times with varying values for ρ.



Copying the Project Skeletons

1. Log into `jinx-login.cc.gatech.edu` using `ssh` cd .and your prism log-in name.
2. Copy the files from the ECE8893 user account using the following command:

   ```
   /usr/bin/rsync -avu /nethome/ECE8893/NetSim .
   ```

   Be sure to notice the period at the end of the above commands.
3. Change your working directory to `NetSim`

   ```
   cd NetSim
   ```

4. You should see empty (or nearly empty) implementations of the ".h" and ".cc" files for all of the objects mentioned above. Most of these can be implemented and tested individually with test-code drivers, and then put together to form a working simulation when they have all been developed and debugged.

Turning in your Project. Use the normal `riley-turnin` procedure to turn in your project by the due date.