In this project, we will add distributed scheduling to the multi-linecard version of the \texttt{wunet} router. This version adds two new methods to the \texttt{wuLinecard} class. The \texttt{sendVOQstatus()} method sends a VOQ status message to one of the “other” linecards. Each time it is called, it sends a message to a different linecard, rotating through all the other linecards in round-robin fashion. The VOQ status message sent to linecard \textit{i} has three pieces of information: the number of bytes waiting to be sent to linecard \textit{i}, the number of bytes that linecard myLcn has to send on its own external link and the total input side backlog that other linecards have to send to linecard myLcn. This last quantity is obtained by summing the values sent to myLcn by the other linecards. The status packets should be placed in a separate queue from the normal data traffic, so that they don’t get delayed behind waiting data traffic. The \texttt{egress()} method must also be extended to process VOQ status messages sent from other linecards. The data received in these status messages should be stored in the linecard table, using fields and methods that have been provided for that purpose.

The second new method in the \texttt{wuLinecard} class is \texttt{voqUpdate(x)}. This method is called right after a VOQ status message from linecard \textit{x} has been received and processed. Its job is to adjust the VOQ sending rates that are maintained within the \texttt{InkTbl} class. This is done using a variant of the LOOFA scheduling algorithm. To implement the scheduling algorithm \texttt{voqUpdate} maintains a list of linecards that are arranged in increasing order of their output-side backlogs. When assigning VOQ rates, it goes through this list in order, assigning both a bit rate and a packet rate to each VOQ. The update process for linecard \textit{x} has two parts. First, the sorted list of VOQ’s needs to be adjusted to reflect the change’s in \textit{x}’s status, since the last update. Since only \textit{x}’s status has changed, we need not do a full sort. It’s sufficient to shift \textit{x} to the “left” or “right” in the list, as appropriate. Next, we need to update the VOQ rates for \textit{x} and those VOQs that come after \textit{x} in the list. The rate assigned to each linecard \textit{i} should be computed in accordance with the following pseudo-code.

\begin{verbatim}
rate = (myLcn’s share of the input-side backlog going to i)* (max rate to linecard i)
rate = min(rate, the rate that myLcn needs to clear its current backlog to i before the next update for i)
rate = max(rate, minRate)
rate = min(rate, remaining unallocated input bandwidth at myLcn - (number of VOQs still left to go)*minRate)
\end{verbatim}

This should be repeated to compute both the bit rate and packet rate to linecard \textit{i}. The pseudo-code variable \texttt{minRate} corresponds to the two program variables \texttt{minBitRate} and \texttt{minPktRate} that you will find already defined in \texttt{wuLinecard}. The first line in the above pseudo-code initializes the rate to myLcn’s share of the output bandwidth at linecard \textit{i}, based on myLcn’s share of the input-side backlog going to \textit{i}. If the first line allocates more bandwidth to linecard \textit{i} than myLcn can use during the period between the current time and the next time it
does an update for \( i \), the rate allocated in the first line is reduced to avoid this over-allocation. This adjustment is handled by the second line above. (Note that the time between successive updates for a particular linecard is given by the program variable \texttt{schedPeriod}, that is defined in \texttt{wuLinecard}.) The third line above ensures that each VOQ gets a minimum rate that is high enough to ensure that VOQ status messages get delivered in a timely fashion. Finally, the last line ensures that there is no over-allocation of the input-side bandwidth at \texttt{myLcn}.

Changes have been made to the \texttt{lnkTbl}, \texttt{lcTbl} and \texttt{statsMod} classes. You should not have to make changes to any of the other components, but you will probably find it worthwhile to review the code and make sure you understand what’s new. The \texttt{wuHost} class has also been changed to allow it to handle packet specifications that include “repeat specifications”. These are used in the experimental setup you will be using. See the code for details.

You will find a new source tarfile (\texttt{srcTarFile7}) on the web site, which has everything you will need for the project, including source files, a wunet configuration files (\texttt{sp7-1.onl}) plus a directory (\texttt{sp7-1} with scripts and other support files to test your router). More details below.

1. Once you have written your new code, you can test it in ONL using \texttt{sp7-1.onl} and the setup in the \texttt{sp7-1} directory. This setup is for a wunet router with seven linecards, which run on the seven hosts at the top of the RLI window. The five hosts at the bottom run \texttt{wuHost}, to generate traffic. One of them (HST0) is not actually used to generate traffic, but serves as a traffic sink for the linecards that do not receive traffic directly from hosts. The packet specification files implement a version of the “stress test” traffic pattern. The monitoring displays show the total input side backlogs to each output and the output side backlogs.

   For this part, you should turn in screenshots of the monitoring displays and selected portions of the linecard and host log files. Include a zoomed-in display of the input backlogs towards the end of the run. Use this information to demonstrate that your scheduling code is working correctly.

   Near the start of the second phase of the stress test, you should observe a small output-side backlog at linecard 2. Explain why this backlog arises and why it goes away when it does.

2. For this part, you will vary the speedup provided by the router and produce a chart showing how the “overshoot” varies as a function of the speedup. Your chart should be similar to the one on page 11 of the section of the lecture notes on coarse-grained scheduling (yours will have just one curve however). Plot values of the overshoot for speedups ranging from 1.1 up to 1.8 (in increments of .1). Compare your curve to the one in the notes for distributed batch LOOFA. Comment on any differences you observe.

3. In \texttt{wuLinecard}, you will notice the program variable \texttt{schedInterval}, which sets the interval between the sending of successive VOQ status packets. In the provided version of \texttt{wuLinecard}, this is set to 2000 \( \mu \text{s} \). Change this value to 500 and run the stress test using a speedup of 1.2. Compare the overshoot to what you observed earlier. Explain the difference. Now, change \texttt{schedInterval} to 10,000 \( ms \) and observe what happens. Discuss what you observe.