In this project, you will modify the routing table of the wunet router to use a hash table instead of performing linear scan. Then, you will evaluate the impact this has on the performance of the router.

1. You will find a hash table data structure on the web site. Modify the \texttt{rteTbl} module to use the hash table for more efficient access to the routing table. Be sure to implement the \texttt{removeEntry} method, in addition to \texttt{lookup} and \texttt{addEntry}. Verify that your new version of \texttt{rteTbl} works correctly, by modifying the basic ONL demonstration configuration used in the previous project. Modify the initial route table files for R1 and R2 to include additional routes (at least 5 per router), and modify the packet generation files to generate packets that exercise all the routes. You should run the packet generator (\texttt{wuHost}) with the repeat flag turned off.

Turn in copies of all the source code you modified, with notations highlighting your changes. Also turn in copies of your route table files and your packet generation files, along with the log files for R1 and R2, with notations that explain how the log files demonstrate that the routing lookup is working correctly.

2. Repeat the throughput test you did in the last part of the previous project using routing tables with 100, 1K, 5K, 10K, 20K and 30K entries. Note that you will need to make these entries all distinct from one another to avoid hash table collisions. I suggest writing a small program to generate the entries using random values for the vnet and address fields.

Produce a chart showing the maximum router throughput for each table size (use minimum length packets). Your chart should have two curves, one for the original router implementation (using a linear scan of the routing table) and one for your modified implementation. Turn in a copy of your chart and comment on the results. Is anything about the results unexpected? If so, try to explain it.

3. On the web site, you will find alternate versions of \texttt{wuRouter} and \texttt{ioProc} in files named \texttt{wuRouterX} and \texttt{ioProcX}. These versions eliminate most of the system calls that the standard versions make, which is useful for separating the time spent on the system calls from the time spent in the core \texttt{wuRouter} code. In particular, \texttt{ioProcX} creates input packets artificially (rather than reading them from a socket), while \texttt{wuRouterX} suppresses most of the \texttt{gettimeofday} calls. Repeat the comparison of the two routing table implementations using \texttt{ioProcX} and \texttt{wuRouterX}. Produce a chart like the one you did before and comment on the differences. Do you find anything about the results surprising?