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 `rteTbl` module to use the hash table for more efficient access to the routing table. Be sure to implement the `removeEntry` method, in addition to `lookup` and `addEntry`. Verify that your new version of `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 (`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.

   See the Appendix for the source and logs.

   The new configuration duplicates vnet 1 on vnets 3-7, with routes identical except for the vnet field.

   We know that the routing lookup must be working because the router logs show that packets are being delivered to directly-connected hosts. Note that packets being forwarded along the R1-R2 link do not demonstrate successful routing lookups, because these are normally sent via flooding after unsuccessful routing lookups.

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.
Absent collisions, the Hash Table search time is independent of the size of the routing table. As expected, the curve is level over different entry counts.

Contrast this with the Linear Table scan, which is \( O(N) \) in the number of entries. As expected, the curve is approximately proportional to \( 1/N \).

3. On the web site, you will find alternate versions of `wuRouter` and `ioProc` in files named `wuRouterX` and `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 `wuRouter` code. In particular, `ioProcX` creates input packets artificially (rather than reading them from a socket), while `wuRouterX` suppresses most of the `gettimeofday` calls. Repeat the comparison of the two routing table implementations using `ioProcX` and `wuRouterX`. Produce a chart like the one you did before and comment on the differences. Do you find anything about the results surprising?
In Question 2, the curves had constant time $K$ per packet in addition to routing time, consisting of packet reception processing and system calls to recvfrom() and gettimeofday(). For the Hash Table search, the time per packet is $K+O(1)$; for the Linear Search, the time per packet is $K+O(N)$.

In Question 3, we have minimized the constant processing time $K$. The Hash Table search stays level, as expected, but throughput is improved by reducing $K$. The Linear Search approximates a throughput of $1/N$ even more closely, as expected.

We can also see that the total throughput has increased dramatically for the hash table search and for small linear searches. This tells us that for the hash table search, the operating system accounts for more than 90% of the packet processing time.

We can roughly quantify the operating system overhead, $K$. For the Hash Table search with real packets, we obtain about $1.3 \times 10^5$ pps, or $7.7 \mu s$ per packet. When not receiving real traffic and reducing the system calls to the minimum, we see about $1.6 \times 10^6$ pps, or $.625 \mu s$. From this we can conclude that the operating system overhead is roughly $7 \mu s$ per packet.