**QJump**

**CS740 Assignment 2**

**Samuel Kruse**

**Summary**

Some data-center applications require low-latency while others may need high-throughput. High-throughput applications can cause network interference which may create delays for low-latency applications. QJump is a module that bounds application latency requirements while supporting line-rate throughput. This is done by leveraging commodity hardware quality of service levels as priority levels, allowing low-latency applications to jump ahead. The trade-off is that the priority level of data from a particular application is inversely proportional to the number of packets allowed into the network from that application. This means that high-priority applications are forced to sacrifice throughput for lower latency. The exact priority levels and how these factors scale are parameters configured at the network level.

**My Code**

I have done a small implementation of the core parts of QJump: the rate-limiting mechanism that slows down high-priority packets from entering the network and the so I could run a small simulation of network traffic. I implemented the queue as a minimum priority queue and made three different priority levels (10, 100, 1000) where the priority was also the rate-limit: i.e. an application with priority 10 could only send 10 packets an epoch. I made several simplifications to make the implementation easier: all packets are the same size, all packets arrive at the same time, the epoch is one iteration of a loop, and there are only three applications. The model supposes that these applications run on separate hosts and only send packets to one host. The network contains one switch with three ingress ports and one egress port.

A picture containing chart

Description automatically generated

I simulated traffic by generating random packets from each application (between 70-100% of the rate limit for each application), and then recorded all the packets that were issued from the switch during the epoch. The rate limit, the range of number of packets generated by each application and the size of the queue heavily affected the outcome of the simulation. The simulation could have been improved by randomly drawing which priority would be sent first as once the queue is filled up, more packets, even high priority ones, cannot enter.

**Can qJump be reproduced?**

While my simulation demonstrates some key ideas from QJump, it is quite far away from an actual implementation. Fortunately, the authors of QJump made their experimental procedure and set-up abundantly [clear](https://www.cl.cam.ac.uk/research/srg/netos/qjump/repro.html). While setting up the experiment and setting it up is clear, a significant amount of hardware setup and software configuration is required. For instance, you would need to install and run Hadoop, Memcached, and PTPd; have 10 hosts running different parts of these applications; configure and route three separate Arista 7124fx switches; and reproduce their architecture on all [hosts](https://www.cl.cam.ac.uk/research/srg/netos/qjump/nsdi2015/network.html).

Though there is code to [help applications](https://github.com/camsas/qjump-app-util) interface with QJump he actual code of the QJump implementation is relatively straight [forward](https://github.com/camsas/qjump-tc/blob/master/sch_qjump.c). This code handles the rate-limiting and the queue control. In total, it is less than 600 lines of code.