Dynamic Scheduling of Latency-Sensitive Flows in Datacenters Using SDN

Riccardo Mutschlechner, Stephen Sturdevant, Kausik Subramanian

1 Motivation

Datacenters see diverse network workloads with different objectives. Primarily, traffic can be classified into two kinds of flows: high-throughput flows (elephant) and low-throughput, latency-sensitive (mouse) flows. Many applications have a multi-layer partition/aggregate pattern workflow where a task at each layer is partitioned into multiple tasks and provided to the workers at the next level. To provide some guarantees, workers will typically be assigned tight deadlines. Network delays can lead to missed deadlines, causing deteriorated performance. Thus, one of the key factors to application performance in datacenters is to provide low latency guarantees to the latency-sensitive flows.

When elephant and mice flows traverse the same queue, the major reason for increased latency is queue buildups. There are two primary cases in which queue buildup severely increases the latency of mice flows. In the first case, the mice flows experience increased latency as they are in queue behind the packets from the elephant flows that have built up the queue. In the second case, the mice flows are interleaved with elephant flows in queue, causing an unfair share of bandwidth and increased latency as well as jitter for the mice flows. Thus, there is a need to classify flows and monitor switch queues to detect queue buildups in real-time. When we detect queue buildups, we need to dynamically schedule flows on different routes based on their classification to ensure low latencies.

The prominence of software-defined networks has increased in recent times, and offers a centralized control of the network. The SDN controller has a global view of the network (such as topology and routing information) and can collect any relevant statistics (such as link utilization and traffic matrix). Using the resources available in a typical SDN setup, we believe there exist the means to schedule flows in real-time to avoid the problem of mice and elephant flow conflicts. Thus, we ask the following question: *How can we perform dynamic flow scheduling to minimize latency of the mice flows in a software-defined datacenter?*

2 Related Work

Alizadeh et. al [?] show that long-lived, greedy TCP flows cause the length of the bottleneck queue to grow until packets are dropped, resulting in the familiar sawtooth pattern in buffer-lengths. Queue buildup is also a major factor for increasing latencies observed in datacenters. To minimize queue buildup, they develop a modified TCP, DCTCP which uses the queue statistics at switches for congestion control, nearly achieving zero queue buildup.

Flow scheduling in datacenters has been an active area of research in recent times. Hedera [?] performs scheduling of flows by estimating TCP bandwidth requirements and dynamic flow scheduling using different placement algorithms (most importantly simulated annealing which is a probabilistic search technique for paths) to achieve full bisection bandwidth requirements. On the other hand, we focus on the problem of dynamically scheduling flows in terms of latency-sensitivity (mice flows, unlike elephant flows, are highly sensitive to latency), which also requires monitoring queue sizes at switches in real-time and a dynamic path placement with the objective of decreasing latency. Hedera does not focus on the aspect of latency in its flow-scheduling algorithm. This also reduces the timescales of scheduling decisions, and requires the need to devise real-time flow-placement algorithms which need not be optimal.

Fastpass [?] presents a datacenter network architecture where the time of packet transmissions and the path traversed by packets are determined by a centralized arbiter. This approach uses a timeslot allocation algorithm to determine when the endpoints packets should be sent and a path assignment algorithm to determine paths to eliminate congestion at switches. The centralized arbiter ensures zero queue buildup in the switches by globally scheduling each packet, and thus flows do not suffer queueing delays. However, there is a throughput penalty for achieving very low queueing delays, as some links would have to be idle.

Crovella et. al [?] examined a commonly used service, Apache; specifically, its routing policies. The paper points out that Apache does not prioritize shorter requests versus longer requests. The authors then examine the potential benefits of different queueing algorithms that prefer shortest-connection first, whose principles can be used in the context of flow-scheduling. The paper mentions that web server workloads are primarily tail-heavy, which may work to our benefit in the sense that elephant flows will not pay much of a penalty relative to the gains that mice flows will see.

3 Experimental approach

As far as our preliminary experimental approach, we will use the POX SDN controller and Mininet to emulate a datacenter network. One challenge we foresee is being able to generate a workload similar to a real-world data center. We plan to emulate a workload by sampling from a tail heavy distribution similar to the one described in [?]. We can use

the Fat-Tree topology [**?**] as a representative datacenter network with the hosts connected to the edges. We can also test our system in a real-world setting by setting up a small datacenter topology in Cloudlab and running it with network workloads to evaluate the performance of our system on a physical network. The different axes of evaluation are as follows: latency suffered by the mice flows, size of queue buildup and performance of the system in terms of overhead of measurement, classification and scheduling.