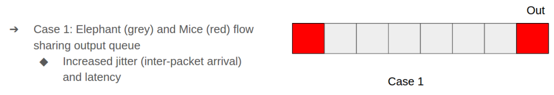
Predictive Queue Management in SDNs

Riccardo Mutschlechner, Stephen Sturdevant, Kausik Subramanian

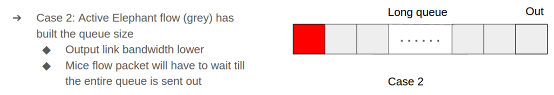
1 Motivation

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 shown in Figure 1 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.


➔ Case 1: Elephant (grey) and Mice (red) flow sharing output queue
- Increased jitter (inter-packet arrival) and latency

(a) Case 1


➔ Case 2: Active Elephant flow (grey) has built the queue size
- Output link bandwidth lower
- Mice flow packet will have to wait till the entire queue is sent out

(b) Case 2

Fig. 1: Cases of Queue Buildup

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 System Architecture

?? displays the system architecture of the scheduler used for predictive queue manage-

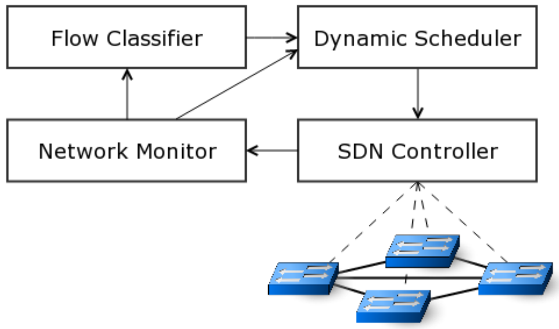


Fig. 2: System architecture

ment in software-defined networks. The four components of the system are as follows:

- **SDN Controller**: The centralized point of control of the network. The advantages of using SDNs for predictive queue management are two-fold: (1) OpenFlow has support for centralized monitoring of switches for different statistics, thus eliminating the need for specialized boxes for measurement and developing clever monitoring algorithms to minimize load on switches due to querying statistics (2) The controller can add/modify forwarding rules at switches, thus providing a centralized point to decide routing of flows, which is received as output from the dynamic scheduler to prevent queue buildups.

- **Network Monitor**: Based on the monitoring algorithm used at the controller, statistics from the entire network topology is collected and passed to the network monitor, which store the necessary statistics needed to predict queue buildups in switches.

- **Flow Classifier**: Using the characteristics gathered by the network monitor, the flow classifier will classify the flow as mice/elephant based on some threshold (can be user-defined)

- **Dynamic Scheduler**: The scheduler predicts queue buildups using the characteristics from the network monitor, and dynamically schedules paths for flows to prevent queue buildups affecting latency-sensitive mice flows using an earliest-deadline-first greedy scheduling algorithm to find new paths, which are sent to the controller to be deployed to the network.

We describe the components in detail in the following sections.

4 Flow Characteristics

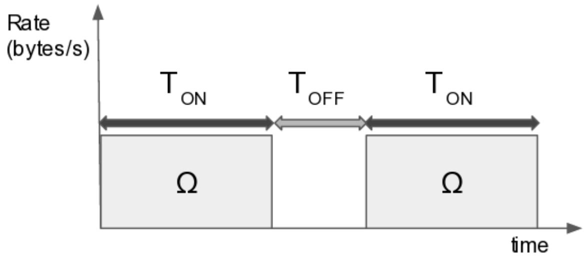


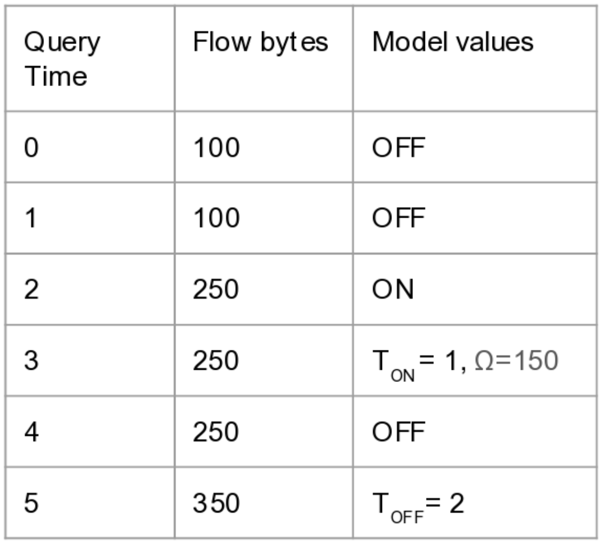
Fig. 3: Flow Characteristic Model

To predict how queue builds up, we need to develop a model of a flow such that we can predict the data it sends in any duration of time. We characterize each flow as following an repeating ON/OFF model [?] as shown in ??. The three main parameters are:

1. **Active time(T_{ON})**: The duration of time for which the flow is actively transmitting data
2. **Inactive time(T_{OFF})**: The duration of time for which the flow is inactive between two transmission cycles
3. **Transmission Bytes(Ω)**: The total number of bytes sent in a single transmission period (the rate is assumed to be constant in the period)

By maintaining a model for a flow, we can use this to easily estimate the number of bytes sent by this flow in a time interval. The queue buildup model can be built by using this information from all flows going through the queue. Development of sophisticated flow characteristic models based on specific distributions is a direction of future work.

4.1 Measurement



Query Time	Flow bytes	Model values
0	100	OFF
1	100	OFF
2	250	ON
3	250	T_{ON}= 1, Ω=150
4	250	OFF
5	350	T_{OFF}= 2

Fig. 4: Example measurement trace for a flow

We use OpenFlow's support for obtaining flow statistics from switches to estimate the flow characteristic parameters. An example measurement trace for a flow is shown in ??. In every t_{meas}, the edge switch where the flow enters the network, is queried for the statistics of the flow, which returns the cumulative bytes received at the switch. We maintain a expotential moving average(EWMA) for each parameter, and update the average from each reading.

The main advantages of this approach is that by measuring flow characteristics, we do not need cooperation from the tenants to provide information about flows (which may be difficult in public clouds and can be misused by malicious tenants). The major shortcoming of this approach is that the least count of T_{ON} and T_{OFF} is the query time interval t_{meas} (we cannot say anything about a time inside a query interval). Thus, it is a crucial trade-off between accuracy of the flow characteristics and the load on switches due to increased querying. We try to address these concerns in later sections.

5 Queue Buildup Model

6 Scheduling

7 Scalability

8 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.