TCPTokens: Introducing Currency into Datacenter Congestion Control

Anand Jayarajan
University of British Columbia
anandj@cs.ubc.ca

Michael Przystupa
University of British Columbia
michael.przystupa@gmail.com

Robert Reiss
University of British Columbia
rreiss@cs.ubc.ca

Fabian Ruffy
University of British Columbia
fruffy@cs.ubc.ca

ABSTRACT

Datacenter networks have become a hotbed for research in the recent past. Many works have been published on improving bisection bandwidth, cost, and latency. As a core contribution of this research, centralized scheduling has entered the stage as a dominant strategy to manage flows. Benefitting from a global view and full control over the network, centralized schedulers are able to enforce fine-grained traffic control, frequently achieving near-optimal bandwidth optimization. However, modern congestion control algorithms and schedulers are still fundamentally reactive. Many algorithms are designed to only respond to packet loss or significant increase in latency, not to actively prevent it. In this project, we plan to explore the opportunities of employing a preemptive and tightly controlling central network scheduler. Using tokens as a global tool to enforce traffic limits and admission control, the scheduler is able to proactively steer the flow of traffic. The notion of "knowledge" and predictive analysis in networks is a growing trend in research, which we intend to leverage in this system.

We investigate the potential in such a new form of interactive congestion control and analyze it against state-of-the-art solutions. Our tool of choice to model our system is Mininet, a rapid prototyping emulator for datacenter networks. We will compare our "Iroko" system against established TCP-congestion systems such as Hedera and DCTCP. Based on the findings and measurement results, we will reassess the feasibility and prospects of a predictive congestion control algorithm.

KEYWORDS

TCP, Congestion Control, SDN, Datacenter

1 INTRODUCTION / BACKGROUND

Developments in the past decade have changed the general networking environment. Data centers have emerged as an exciting new research frontier, posing novel design challenges and opportunities. Driven by minimization of costs and maximization of compute power, data centers must run at the highest possible utilization to achieve the ideal compute/cost ratio. Optimizing the distribution of traffic and simultaneously guaranteeing fairness is a perennial challenge for any network operator. Inefficient routing can quickly lead to bufferbloat [**?**] and the eventual collapse of a high-load network, requiring sophisticated approaches to solve congestion control. Despite the innovation potential of data centers, TCP has dominated as the congestion control protocol of choice. TCP is a *reactive protocol*, responding to indicators of congestion and latency in the network. However, the fact that packet loss and latency surges occur in the network, already portends a problem. Packet loss is incurred by overflowing queues in forwarding elements or mismatched hardware capabilities, implying that traffic has not been optimally distributed. Ideally, a network should always be "zero-queue", i.e., latency will merely be induced by propagation, and not queuing delay.

With the advent of Software-Defined Networking (SDN), operators now have the ability to freely control and adapt their network architecture, leading to highly customized systems and fine-grained optimization. [**?**]

Moving away from the principle of distributed communication and routing, SDN introduces the notion of "centralized management". A single controller with global knowledge is able to automatically modify and adapt the forwarding tables of all switches in the network, while notifying end hosts of changes in the network. These two new trends in system design; full architectural control and centralized management, facilitated new opportunities in the space of TCP congestion research. Traffic can now be managed in a *centralized* fashion based on *global knowledge* of the entire topology and traffic patterns.

A new line of centralized schedulers has emerged that can achieve close to optimal bandwidth utilization. [**? ? ? ? ?**] However, these schedulers are still *reactive* in nature. The central controller responds to changes in the network or requests by applications, which may cost valuable round-trip latency. Often, short-term flows or bursts are unaccounted for, which causes undesirable packet loss and back-propagating congestion.

The idea of admission control and service guarantees in networks is not new. [**? ?**]. However, such designs traditionally aim to assure quality and bandwidth guarantees in a contentious, decentralized, and untrusted environments such as the internet. In a datacenter, these conditions do not apply. End-hosts are generally considered reliable and restricted in behavior, which allows for great simplification of enforcement and prioritization policies.

A desirable solution is a global, centralized arbiter which is able to predict and fairly distribute flows in the network before bursts or congestion even occur. By treating the network's compute and forwarding power as a single finite resource, a controller acts akin to a OS scheduler distributing CPU time slices to processes. This design approach follows SDN's aspiration of introducing operating systems abstractions to the networking domain space.

In this project, we plan to explore the possibilities of a centralized, proactive flow scheduler. We ask ourselves the following research questions:

(1) What are the requirements for such a centralized, predictive scheduler to succeed?
(2) Is it possible to preemptively regulate a network by analyzing global traffic patterns?
(3) What latency, packet loss and utilization are we able to achieve?

In the scope of this course, we attempt to answer these questions and design a simple predictive scheduler in Mininet. If successful, we will benchmark our results and evaluate the level of utilization compared to contemporary scheduling systems.

2 DESIGN / PROPOSED APPROACH

2.1 Overview and Goals

In our initial simple system, "Iroko", a centralized controller regulates all node traffic by rate-limiting end-hosts. We have opted for a centralized manager in favor of a distributed protocol to leverage the advantages of a global network view. [1] By polling each switch individually for port and utilization statistics we are able to infer a global traffic matrix, which we can amalgamate with static routing and topology information.

Iroko guarantees minimal congestion and low average access latency. Reducing network-global packet loss and jitter is the priority and objective function of the arbiter, which will enforce these goals by restricting host bandwidth. In an ideal Iroko system, packet-loss will only rarely, if ever, occur. In respect to the free lunch theorem we aim to trade off maximum bandwidth utilization for optimal system stability and reliable latency.

It is important to note that routing decisions are not in the current scope of Iroko. While it may certainly beneficial to include dynamic and adaptive routing decisions in a predictive scheduler, we do not include these features in the current system due to time and complexity limitations. Iroko operates on static flow routes defined and generated by ECMP. It is assumed that these routes will not vary substantially and that the computed ECMP forwarding hash is consistent.

2.2 Controlling traffic flow

Rate-limiting is performed on the granularity of the IP protocol, allowing us to restrict traffic to specific hot regions and to package flows into groups. End-hosts are guaranteed a limited amount of bandwidth which they may send to each destination address. The traffic window size for any flow is calculated based on the host's current traffic availability. Total bandwidth of a group of flows may never exceed the imposed bandwidth limitation, but nodes are free to distribute the allocated resources on singular subflows.

The central arbiter enforces these bandwidth and access limitations by assigning "tokens" to nodes. These tokens are plain information packets specifying the affected destination address flow, its bandwidth guarantee, and the expiration time. Tokens act as the rate-limiter of the system and form a queue each end-host will cycle through.

[1] For now, we do not expect our design to scale up to thousands of switches. Iroko is intended for small to mid-tier size data centers, which may benefit from a simplistic, centralized scheduling model.

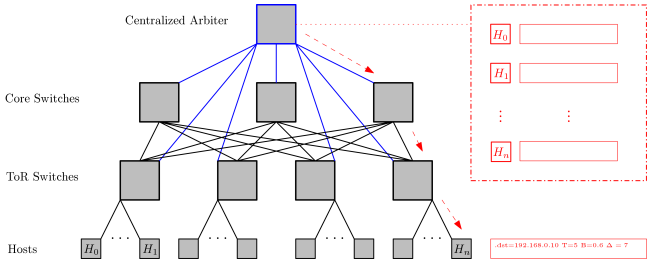

Figure 1: Simple sketch of the Iroko architecture. A central scheduler maintains a consistent view of the current network activity and computes the optimal end-host bandwidth distribution. Bandwidth is enforced at end-hosts on a per-IP basis.

When a node opens a flow, it will look up the current bandwidth restriction for the destination IP in its token database and calculate the maximum congestion window possible for the particular TCP connection. In our simple design, flows are always assigned a fair fraction.

Once a token has expired for a particular flow-group, the restrictions of the next token in the queue will be applied. This mechanism is the underlying basis for a predictive access control algorithm and does not require any application level modification. On end-hosts, only the transport layer services will have to be modified.

2.3 Adapting traffic flow

Initially, the controller will compute optimal route configuration based on the topology and link bandwidth using a simple heuristic bin-packing approach. End-hosts will be initialized with a fixed low-to-medium bandwidth guarantee that will be adjusted over time. This bandwidth guarantee will be below the host's proportional bi-section bandwidth that would be used to reach any other host. That is, if every end host transfers using the full bandwidth initially allocated to it, there would be no dropped packets due to congestion.

Of course, this would vastly underutilize network resources, thus the controller will dynamically adjust these allocations to ensure that hosts that require the bandwidth are allocated it, while host that are not currently utilizing all their allocated bandwidth have their allocated bandwidth reduced. To avoid starving hosts, some small amount of bandwidth must always be allocated to a given host even if it not utilizing network resources. Thus, the network, by design, will never reach 100% utilization.

Conversely however one should rarely see dropped packets due to network congestion. Furthermore, the controller must be able to react quickly to changing bandwidth needs for hosts. Ideally, using a predictive method.

2.4 Statistical System

One potential implementation for our arbiter is to use reinforcement learning. In reinforcement learning, an agent (in our case the arbiter) performs actions in the environment and receives a reward signal from the environment to adjust its decisions in the next epoch [?

]. The reward is generally used to represent the goal the agent is trying to achieve. The agent can freely choose whatever actions are necessary to achieve this goal [**?**].

The environment in this case is defined as the statistics collected from each switch. This decision is based on the fact that this data is easy to collect and provides a current snapshot of the performance of the network. We wish to optimize over the packet loss, and define the reward to be good if packet loss is decreasing (+1 reward); bad if it is increasing (-1 reward); and acceptable if packet loss remains the same within a threshold (0 reward). Using this representation, we take full advantage of all the information provided in our data center; This representation is replicable and can be applied beyond the scope of this project.

Given the data center assumption, the number of end-hosts is known in the topology. We can define our arbiter's actions as a discretized representation among an n-dimensional vector where each cell is one of three values: increase decrease, or maintain the allocated bandwidth. This representation offers sufficient complexity. There are 3^n possible actions to perform and to achieve optimal results the agent must explore the environment extensively, precluding a significantly larger action space.

A potential improvement to our representation is a hierarchical learning agent. In this case, our arbiter would manage agents for each host and these sub-agents would manage their respective host whether to increase, decrease, or maintain bandwidth for their particular host. This problem can be referred to as hierarchical reinforcement learning [**?**], but may prove to be beyond the scope of our current proposed solution, although has the added benefit of solving a set of sub-problems to address the global problem of packet loss and simplifies the representation spaces in the sub-problems.

Thus we frame our problem as one which can now be solved using techniques in reinforcement learning. A classic algorithm to consider is the use of SARSA [**?**], which is an online temporal difference algorithm used in classical reinforcement learning problems. One advantage of SARSA is its online nature [**?**] which promotes more conservative decisions. Given full representation of our environment and actions, we can store information in an artificial neural network which is another popular choice in reinforcement learning literature. This model would store the reward signal information that is used to make future action choices, and is a theoretically proven function approximator which has made intractable environmental representations manageable [**?**].

3 IMPLEMENTATION

We intend to emulate our system in Mininet [**?**] to observe traffic patterns and infer a suitable token algorithm. Mininet has proven itself to be a viable tool to model new congestion control algorithms [**?**], and will help us prototype our concept efficiently. We will build a custom SDN controller that interacts with traditional OpenFlow software switches as well as end-hosts. End-hosts will run a custom real-world traffic generation script which adjusts based on information packets sent by the controller. If time permits, we may expand our implementation to MaxiNet, which can emulate large-scale network stress tests on multiple physical hosts.

We initially considered a second implementation alternative in C/C++ based on the FastPass [**?**] source code. This would provide us with a fully deployable system which we could fork our implementation from. A major advantage of this approach is the ability to test scenarios and traffic algorithms using real software code. However, several concerns made us favour a Mininet emulation instead. Firstly, FastPass relies on DPDK integration, which requires actual hardware interfaces. The central arbiter in the FastPass design would need to run on a dedicated machine, which increases prototyping and development complexity substantially.

Secondly, the FastPass code is highly specialized and optimized research code with only little available documentation. Modifying and evolving the source code will require thorough understanding of kernel and networking development, a significant time-sink. For a class project, these may be major initial hurdles, taking away from the research aspect of the design concept. Consequently, we have decided to pursue an approach which allows us to quickly develop an understanding of the problem without being obstructed by engineering work.

4 EVALUATION

In the absence of a commercial data center, the implementation of our network design is going to be done on top of mininet with simulated FatTree topologies of varying size. We stress test with iPerf and simulate data center traffic using tcpreplay and packet traces provided by [**?**] .

To evaluate the general effectiveness of our system, we plan to measure against existing centralized as well as decentralized solutions. The centralized design will be based on Hedera [**?**], a common and influential datacenter scheduler. The decentralized congestion control mechanism will be DCTCP [**?**], a state-of-the-art TCP congestion algorithm.

Since we are primarily concerned about reducing the latency and packet drops while keeping utilisation at maximum, the measurements to get a good insight into how well the design can perform are as follows:

(1) Latency: We are aiming for a low latency network which means that 99th percentile latency in the network across all flows should as low as possible. Latency should be measured when all hosts are sending packets at the maximum limit and also with random traffic patterns. There should be low latency even during sudden bursts as the transmission rate is limited by a base value.
(2) Packet drop rate: Ideally this metric will approach zero, as the objective of Iroko is to minimize packet loss.
(3) Fairness: Fairness can be tested by introducing a new host to a completely saturated network and increasing the transmission rate to see if all the hosts gets fair share of the total bandwidth. We are assuming all flows should be at equal priority. Differential priority is out of scope.
(4) Responsiveness: This metric depends on the predictive power and efficiency of Iroko. It may be interesting to analyze the response time of the central scheduler compared to decentralized DCTCP and TCP. In addition, it is valuable to identify at what data center and flow size the central computing node of the network may become a bottleneck.

(5) Network utilization: We will measure the used bandwidth and the total load on the hosts. These metrics can be measured by varying the load from each host. Load to utilization ratio should be 1 until the saturation point and after that utilization should stay stable at the maximum bisection bandwidth. It is also important to check that there is no starvation happening in any host. This can be measured by simulating random traffic patterns and plotting residual bandwidth and excess load. We expect overall utilization to be lower compared to Hedera and DCTCP. Although Iroko will maximize goodput, we do not measure it on grounds of simplicity.

As baseline we will also compare to the default TCP congestion algorithm to estimate how our scheduler fares against the default case. As our initial simple design is very constrained in its ability to route and control traffic we expect to achieve less overall optimality against DCTCP and Hedera, but still gain a substantial advantage over common TCP.