Coexistence of Congestion Control Mechanisms in Datacenter Networks

Chen Tian[†] Ali Munir[‡] Alex X. Liu[‡] Yingtong Liu[†] Hongqiang Liu[§]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*Department of Computer Science and Engineering, Michigan State University, USA*
[§]*Microsoft Research, Redmond, USA*
Email: {alexandretian, lampson0505}@gmail.com, {alexliu, munirali}@cse.msu.edu

Abstract

Cloud applications generate flows with different performance objectives according to their service level agreements. A large body of congestion control work exists to meet the performance objectives of different applications, and each protocol has its own version of congestion signal and control laws. Without surprise, such protocols can not live together peacefully and degrade application performance when sharing network with different protocols. The root cause of this conflict is that they try to control the congestion of the same underlying physical network with different congestion control mechanisms. In this paper, we identify the significant interference among various flow control approaches when they coexist in the same network. The key insight behind C3 is that: coexistence of flows can be achieved by exposing a vision to each control mechanism that it is the only congestion control protocol. We define the *virtual elastic pipe* abstraction, which achieves *Isolation* and enables *Coordination* among flows with different objectives. To realize the virtual pipe abstraction, we design a congestion control framework that inserts a dedicated shim layer between hosts' network and transport layers to efficiently realize network sharing, based on given bandwidth guarantee and weighted sharing parameters of each pipe. We also present a series of novel algorithms to derive the bandwidth guarantee and weighted sharing parameters for pipes. We evaluate C3 both on a small-scale testbed and with large-scale simulations. C3 reduces average flow completion time by up to $3\times$ for priority-sensitive flows, and reduces deadline misses by up to $4\times$ for deadline-sensitive flows, compared to coexistence without C3 support. More interestingly, C3 not only eliminates interference among flows of different objectives, it also improves the performance of flows in non-coexistence scenarios.

1 Introduction

Cloud datacenters are shared by various applications. These applications run many services, distributed across the datacenter network, that communicate with each other to complete various tasks. Therefore, intelligent network sharing mechanism plays an important role in improving application performance.

Applications generate flows with quite different performance objectives according to their service level agreements (SLA). We identify four main categories of flows as follows:

- ***Throughput-Sensitive (TS):*** The message completion time of large message applications [1, 2], and many user-facing and enterprise applications [3] depends on the throughput achieved by the flows in the network. Such applications have complex traffic interactions among multiple tiers of processing stages and predictable throughput guarantees among processing stages is desirable to deliver predictable response-times [4, 5].
- ***Latency-Sensitive (LS):*** The response time of web-search like applications, that generate query traffic, is quite sensitive to the per-packet latencies [6]. As another example, for queries configured with tied-request [7], a cancelation message should be sent to the counterpart server as soon as possible to reduce computation and network resource wastage.
- ***Deadline-Sensitive (DS):*** Online Data Intensive applications (OLDI) [8, 9] and real-time analytic [10, 11] operate with soft-deadline constraints (*e.g.*, 300 ms latency). The partition-aggregate architecture of such application enforces deadline semantics for every leaf-to-parent flow.
- ***Priority-Sensitive (PS):*** For some applications, throughput, latency or deadline may not be a constraint but it might be desirable to prioritize some flows over others to improve the overall application performance. For example, task-aware scheduling groups flows of a task and schedules them together, to reduce the flow completion times [12]. Similarly, scheduling policies (such as SRPT [13]) have been proved to optimize the performance of different applications [14, 15, 16].

The above flow performance objectives can be mapped to different requirements from the underlying datacenter networks. Table 1 summarizes these requirements: TS flows are sensitive to available bandwidth, but not to in-

	TS	LS	DS	PS
Latency	No	Yes	Yes	Minimal
Bandwidth	Yes	No	Yes	Yes

Table 1: Diverse network demands of applications.

network latency; LS flows are the opposite; DS flows are sensitive to both latency and bandwidth; PS flows are mostly sensitive to bandwidth, and low latency could only be valuable for short flows.

A large body of work exists to meet the performance objectives of different application flows, such as throughput [17, 18], latency [19], deadline guarantee [20, 21] and flow prioritization [14, 15, 16]). For these protocols, there are two key design factors that define its congestion control mechanism: (i) ***congestion signal*** that means what method is used to detect network congestion, and (ii) ***control laws*** that means what rate increase (decrease) law is used to grab (yield) network bandwidth.

A common inadequacy, of the existing works, is being *egocentric*. Most of the existing protocols are designed to achieve only one performance objective at a time. As a result, each protocol has its own version of congestion signal and control laws. For the congestion signal in datacenter networks, some protocols rely on packet drop [18], some rely on ECN bit marking [20, 6], and others rely on round trip time [22]. For the control laws, almost every protocol has its own bandwidth increase/back-off parameters, and some protocols even apply different rules to different endpoints according to the specific application performance objectives [20].

Without surprise, such protocols can not live together peacefully and could degrade application performance when sharing network with protocols having different congestion control mechanisms. Let us consider a simple example where throughput-sensitive flows (managed by ElasticSwitch [17]), and deadline-sensitive flows (managed by D^{2}TCP [20]) compete for a bottleneck link. Assume that the link capacity is large enough to accommodate all bandwidth guarantees and flow deadlines. For congestion signal, both protocols rely on ECN marking in switches to detect network congestion. However, ElasticSwitch assumes traditional single bit indication of congestion, while D^{2}TCP assumes the rich multi-bit congestion indication [6]. There is only one underlying physical network and the congestion signal can only be set in one way. We assume the method is multi-bit ECN. Therefore, ElasticSwitch misleadingly interprets the congestion signal. Also, being ignorant of other flows with a different performance objective, both ElasticSwitch and D^{2}TCP flows compete (*i.e.*, increases and back-offs) as if all other flows share the same control

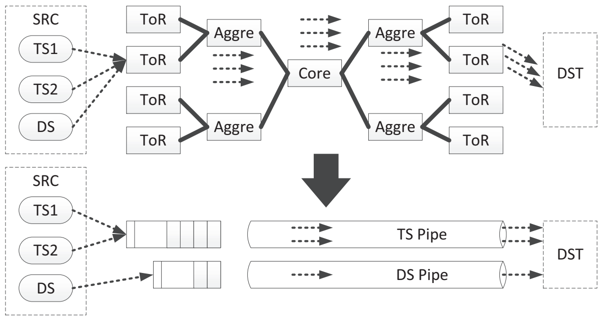


Figure 1: Virtual elastic pipe abstraction.

mechanism. As a result, both bandwidth and deadline guarantees can be compromised (details in Section 2).

The root cause of this conflict among different flows is that they are trying to control the congestion of the *same* underlying physical network with *different* congestion control mechanisms. An example is shown in Figure 1 upper figure: from a source host to a destination host, there are two TS flows and one DS flow. These flows could congest at, and get congestion signals from, any switch queue along the path.

We present C3 (Coexistence of Congestion Control), a system where flows with different performance objectives can coexist in the same datacenter network. The key insight behind C3 is that: coexistence of flows can be achieved by exposing a vision to each control mechanism that it is the only congestion control protocol in the network. Instead of exposing the underlying network congestion directly to different control mechanisms, we propose a *virtual elastic pipe* abstraction. A *virtual elastic pipe* is a point-to-point tunnel between source and destination managing the congestion along the path. In the example (Figure 1 bottom figure), there is one TS virtual pipe and one DS virtual pipe, both unidirectional. All TS flows enter the TS pipe, and the DS flows enter the DS pipe. Each pipe has a rate limiting queue attached to it's ingress port. A virtual pipe is elastic, since it reacts to the congestion in the underlying physical network, and correspondingly adjusts its queues' rate limit. Note that a pipe derives its congestion control parameters from its contained flows, guided by their common performance objectives. For each flow, the rate limit queue of its pipe is the only place it can experience congestion, and packets already inside the pipe are delivered in a congestion-free manner.

The benefits of this virtual elastic pipe abstraction are three-fold. First, it achieves *Isolation* among performance objectives. To flows, a virtual pipe is an abstraction of the allocated network resource of the underlying network. Flows with the same performance objective now have exclusive access to the pipe queue. The existing control mechanism can achieve the performance objective by managing the competition among these flows, without the concern of conflicting congestion signal and control laws. Second, it also enables *Coordination* among performance objectives. To networks, a virtual pipe is an abstraction of the network demands of all its contained flows. Each pipe competes to get a share that can meet the specific performance objective of its flows. Regardless of their objectives, we can enforce the same control mechanism for all pipes. With a single congestion control mechanism at pipe level, C3 can achieve small queuing at the physical switches, so that flow level control can enjoy either low or high queueing in their own pipe queues. With a central coordinator, C3 can also apply a scaling factor to each pipe's congestion control values, hence enforce a network-wide allocation policy. The third benefit is *Flexibility* to flow scheduling. With virtual pipes, flow scheduling to achieve performance objective is way more flexible. The main congestion point of a flow is the pipe queue at the sender end; without worrying about the congestion inside switches, it is expected that more efficient scheduling mechanism could be enforced at the host.

There are two challenges in achieving C3 goals. First, how to realize the virtual pipe abstraction in a practical way? We expect C3 to be easily deployable: it should not require any changes to switches, NICs, applications, existing transport protocols or operating systems. Second, what should be the congestion control mechanism for pipes? Datacenter network bandwidth is a precious resource and bandwidth allocation should be work-conserving. Besides meeting their respective performance objectives, flows should also compete to utilize available spare bandwidth. For each pipe, we can attach a minimum bandwidth guarantee and a weight for network-sharing. TS flows already have these parameters, we can simply sum them up. However, for DS, LS and PS flows, C3 needs to convert their network demands to minimum bandwidth guarantees and sharing weight without degrade their performance. For example, for a group of DS flows exclusively using a datacenter network, their D^{2}TCP control mechanism should achieve comparable performance with and without C3.

Overall, this paper makes following contributions:

- We show that various congestion control protocols, with different performance objectives, can experience significant interference when coexisting in the datacenter network.
- To realize the virtual pipe abstraction, we design a congestion control framework that inserts a dedicated shim layer between hosts' network and transport layers to efficiently realize network sharing, based on given bandwidth guarantee and weighted sharing parameters of each pipe.
- We present novel algorithms to derive the bandwidth guarantee and weight parameters, of a pipe, that depend on the performance objectives of the flows contained in a pipe.

We implement C3 as a Linux kernel module using NetFilter. We evaluate C3 on a small-scale testbed with 6 Dell servers and a commodity PICA-8 Gigabit Ethernet switch with ECN enabled. To complement our small-scale testbed experiments, we further conduct large-scale simulations using ns-2. We show that C3 is able to meet the performance objectives of difference flows at the same time. Compared with direct coexistence, C3 reduces average flow completion times (AFCT) by up to $3\times$ for priority-sensitive flows, and reduces deadline misses by up to $4\times$ for deadline-sensitive flows. More interestingly, C3 not only eliminates interference among flows of different objectives, it also improves the performance of flows in non-coexistence scenarios.

Rest of the paper proceeds as follows. We first review background and motivation in Section 2. The design of C3 is presented in Section 3. Then, we introduce details of the algorithms in Section 4. The implementation of the prototype system, together with preliminary results, is described in Section 5. Extensive evaluation results are demonstrated in Section 6. We conclude our paper in Section 7.

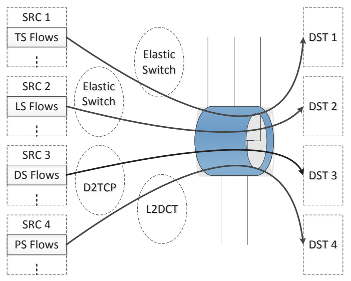
2 Background and Motivation

A lot of work has been done in recent years to improve the application performance in cloud datacenter networks. Most of these works optimize resource sharing for only one specific category of flows: TS, LS, DS or PS. Via an illustrative example, we demonstrate that existing protocols, when sharing the same network, can interfere with each other and can degrade the application performance. We also show that physical network segregation is not an answer.

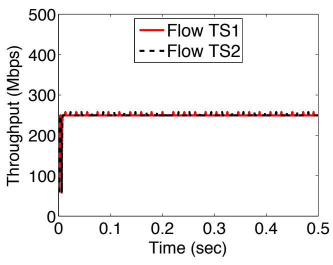
2.1 Related Work

Throughput Sensitive: Implementing bandwidth guarantees on a shared network infrastructure requires *a network abstraction model*, *an entity placement algorithm* and *a runtime mechanism* to enforce bandwidth guarantees. Several bandwidth guarantee approaches have been proposed recently, that provide network abstraction models and placement algorithms for tenants or applications to express their bandwidth requirements [23, 24, 25, 26, 17, 5].

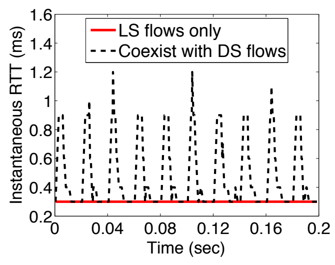
SecondNet [23] and Oktopus [27] provide predictable guarantees based on the hose network model; however, they lack work conservation property and as a result do not utilize the shared network bandwidth efficiently. Gatekeeper [25] and EyeQ [26], build on hose model and



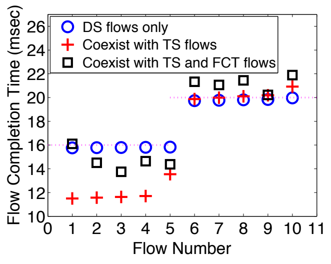
(a)



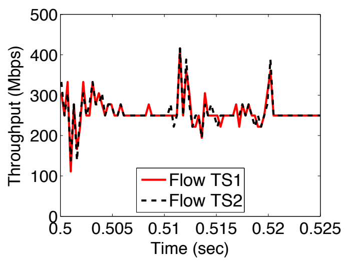
(b)



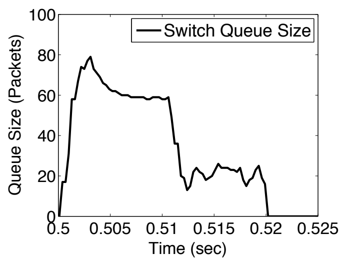
(c)



(d)



(e)



(f)

Figure 2: Interference demonstration: (a) example topology; (b) throughput-sensitive flows' achieved bandwidth at destination; (c) latency-sensitive flows' RTT, when alone and coexist; (d) deadline-sensitive flows' finish time, when alone and coexist;(e) throughput-sensitive flows' achieved bandwidth at destination, when coexist with deadline-sensitive flows; (f) zoom in queueing status.

achieve work conservation; however, these mechanisms are designed specifically for congestion-free core networks [28].

ElasticSwitch [17] transforms the hose model to multiple virtual VM-to-VM connections to provide minimum bandwidth guarantees for each connection. For work conservation, it employs a TCP-like mechanism to probe for additional bandwidth when there is no congestion. CloudMirror [5] proposes a placement algorithm and leverages application communication patterns to provide bandwidth guarantees. It uses ElasticSwitch as the underlying base protocol.

Latency Sensitive: Silo [19] also leverages the hose network model to provide bandwidth guarantees. It proposes a placement algorithm and a hypervisor-based packet pacing to provide per-packet latency guarantees. HULL [29] trades bandwidth for ultra-low latency. It requires modification to switch ASIC. Fastpass [30] uses a centralized arbiter to decide when and which path each packet should be send. Its potential in large scale datacenters is yet to be proven. QJUMP [31] uses multiple physical switch queues to provide different combinations of latency and throughput. It supports both latency-sensitive and throughput-sensitive flows, but with contraints.

Deadline Sensitive: D^3 pioneers the idea of incorporating deadline awareness into network scheduling [21]. However, D^3 and RACS [32] are not deployment friendly as they require modifications to the commodity switch hardware.

D^{2}TCP [20], is a distributed and practical deadline sensitive protocol. D^{2}TCP uses both ECN feedback and deadlines to modulate the congestion window; the key idea is that far-deadline flows back-off aggressively and near-deadline flows back-off conservatively on a congestion event.

Priority Sensitive: PDQ [15] uses preemptive flow scheduling to minimize average flow completion times (AFCT). pFabric [14] and PASE [16] assume flow size is known a priori, and attempt to approximate Shortest Job First (SJF), which is the optimal scheduling for minimizing AFCT in a single bottleneck scenario. L^{2}DCT [33] and PIAS [34], without prior knowledge of flow size information, reduce AFCT by approximating the Least Attained Service (LAS) scheduling discipline. Recently, the concept of *Coflow* has been introduced [12, 35], where a batch of parallel flows having application-specific correlations are scheduled together. The scheduling requirement is still either minimizing completion time or meeting deadlines, but at coflow level instead of individual flow level.

2.2 Motivation Example

We consider a simple example to understand the impact of interference, on application performance, when different protocols coexist in the same network.

Experimental Setup: To evaluate interference effects, we consider a single bottleneck topology as shown in Fig. 2(a). This represents a common datacenter scenario where edge to core links might be the bottleneck among flows of different applications. We enable multi-bit EC-N feedback support, and following DCTCP guidelines, set the switch ECN marking threshold to 20; the switch buffer size is 250 packets; the round trip time of each communication pair to 300 us. We use NS2 for performance evaluation.

For interference analysis, we consider traffic from a throughput-sensitive application (from $SRC1$ to $DST1$), a latency-sensitive application (from $SRC2$ to $DST2$), a deadline-sensitive application (from $SRC3$ to $DST3$), and a priority-sensitive application (from $SRC4$ to $DST4$). We use the bandwidth received at the destination, the instantaneous end-to-end packet RT-T, fraction of missed deadlines, and AFCT as the performance metrics for TS, LS, DS and PS applications respectively.

We consider state-of-the-art deployment friendly protocols for each of the performance goals and implement them in NS2. Specifically, we consider E-lasticSwitch [17] for bandwidth-sensitive and latency-sensitive applications; D^2TCP for deadline-sensitive and L^2DCT for priority-sensitive applications.

2.2.1 Protocols Performance in Isolation

To understand the impact of interference, we first evaluate the application performance in a scenario where each protocol has exclusive access to the underlying network. When used in isolation, we represent throughput-sensitive flows as Base-TS, latency sensitive flows as Base-LS, delay-sensitive flows as Base-DS, and priority sensitive flows as Base-PS. In these experiments the bottleneck link capacity is 500 Mbps.

Base-TS In this experiment, we setup two throughput-sensitive flows (Flow TS1/TS2) from $SRC1 \rightarrow DST1$. Each flow uses ElasticSwitch protocol and requires a bandwidth guarantee of 250 Mbps (total equal to the bottleneck bandwidth.). As shown in Fig. 2(b), both the flows achieve their bandwidth guarantees without hurting each other.

Base-LS In this experiment, we setup seven small (2 K-B) latency-sensitive flows from $SRC2 \rightarrow DST2$. All evenly spaced flows use ElasticSwitch protocol and require an aggregate bandwidth of 400 Mbps (over a shared 500 Mbps link). We repeat this experiment every 2 m-s and flows take 2 ms to finish. As shown in Fig. 2(c), the LS flow packets experience almost zero queuing delay and the round-trip latency is very close to the RT-T (300 us).

Base-DS In this experiment, we setup ten delay-sensitive (DS) flows from $SRC3 \rightarrow DST3$. Among these, five flows are of size 92 KB (16 ms deadline), and five flows are of size 138 KB (20 ms deadline). All the flows, use D^2TCP protocol, start simultaneously; thus, an aggregate average bandwidth of 500 Mbps is required to meet their deadlines. This process is repeated every 20 ms and lasts for 1 second. Fig. 2(d) demonstrates the flow completion time of ten DS flows and we can see that all the flows meet their deadlines.

Base-PS In this experiment, we consider the same set-up as deadline-sensitive flows, except that the flows use L^2DCT protocol, which is a size aware protocol aimed at minimizing AFCT, and achieve an AFCT of 17.64 ms.

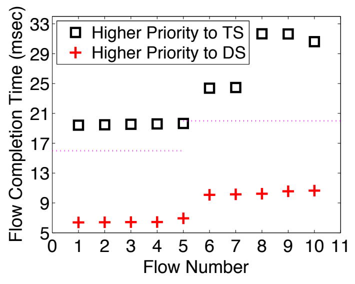
2.2.2 Protocols Performance in Coexistence

In this section, we evaluate the impact of interference, on application performance, when the flows with different performance goals coexist in the network.

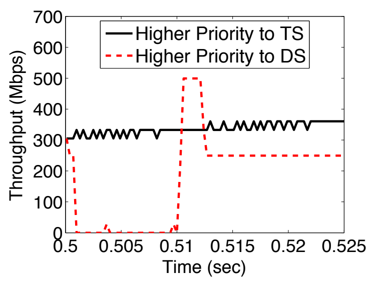
Coexist-TS/DS First, we consider the co-existence of throughput-sensitive and deadline-sensitive flows. We use flow settings same as Base-TS and Base-DS experiments; the two TS flows start at 0 sec and all the DS flows start at 0.5 sec. The bottleneck link capacity is set to 1000 Mbps, which is large enough to meet the requirements of both the TS and DS flows.

Throughput-sensitive flows grab the network bandwidth in the absence of delay-sensitive flows, however, their performance degrades when coexisting with D-S flows, as shown in Fig. 2(e). In the absence of DS flows, both the TS flows (Flow TS1/TS2) achieve nearly 400 Mbps throughput, which is comparable to the results in the original ElasticSwitch work [17]. However, when coexisting with DS flows (*e.g.*, 0.5 to 0.525 sec Fig. 2(e)), the bandwidth guarantees of both the TS flows are violated severely.

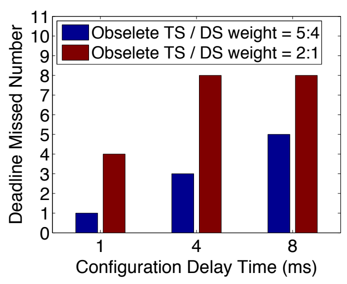
The reason behind performance degradation of TS and DS application lies in the difference in congestion signal and control laws used by ElasticSwitch and D^2TCP protocols. Upon arrival of DS flows, the queues in the switch starts building up, as shown in Fig. 2(f), and both the ElasticSwitch and D^2TCP flows react differently to network congestion. Each of the ElasticSwitch flow assumes that, using hose model, network bandwidth is shared fairly among all the flows and its guaranteed rate 250 Mbps is respected under all network conditions; as a consequence, a rate limiter at source never drops its rate lower than 250 Mbps. Similarly, each DS flow assumes that D^2TCP is the only protocol in the network and on observing congestion, five flows (with deadline 20 ms) back off aggressively and the remaining five flows (with deadline 16 ms) back off conservatively. As a result, the switch queue grows quickly (Fig. 2(f)), and flows experience large queuing delays. For TS flows, although



(a)



(b)



(c)

Figure 3: Multi-queue settings: (a) deadline-sensitive flows' finish time with SP; (b) throughput-sensitive flows' achieved bandwidth at destination with SP; (c) Number of deadline-sensitive flows missing deadlines with WFQ (assume new TS / DS weight = 1:1).

their packets are sent out at the guaranteed bandwidth, the realtime bandwidth achieved at the destination drops as low as 110 Mbps (and last for several milliseconds). Even when there are only 5 D^{2}TCP flows left (after 0.516 second), the achieved bandwidth still gets hurt. DS flows also experience poor performance because of the large queue buildup in the network. As shown in Fig. 2(d), 2 out of 10 deadline-sensitive flows miss their deadlines. Therefore, when co-existing, both the TS and DS flows can hurt each other.

Coexist-LS/DS Next, we consider the co-existence of latency-sensitive and deadline-sensitive applications. We use flow settings same as Base-LS and Base-DS experiments. The bottleneck link capacity is set to 1000 Mbps, which is large enough to meet the demands of both LS and DS flows.

In this scenario, the performance of latency-sensitive flows is severely hurt as shown in Fig. 2(c). LS flows experience large queuing delays and, as a result, their tail latency increases by more than 300%; the average latency increases by more than 80%; with the variance as large as 7.56. The reason is that, D^{2}TCP flows grab the spare bandwidth and occupy switch buffer space; now, when the packets of LS flows arrive at the switch, they get queued behind the packets of DS flows in the switch buffer. As a result, the latency of LS flows increases significantly.

Coexist-TS/DS/PS Lastly, we consider the coexistence of three kinds of flows (*i.e.*, throughput-sensitive, deadline-sensitive and priority-sensitive) simultaneously in the network. Without the loss of generality, we treat the scheduling as an optimization problem: TS and DS flow requirements are the constraints that should be met and minimizing AFCT of PS flows is the optimization objective function. We use same settings as Base-PS, Base-TS and Base-DS for each application and set the bottleneck link capacity to 1500 Mbps.

In this scenario, DS flows experience degraded performance, as shown in Fig. 2(d), six out of ten DS flows miss deadlines due to interference from flows of other applications. Whereas, the TS flows are only slightly hurt and the AFCT of PS flows (negligibly) improves to 17.62 ms.

2.3 Network Segregation is Not an Answer

An intuitive solution, to mitigate interference among congestion control mechanisms, is to segregate application flows with different performance objectives to separate physical switch queues. Although there are only four main performance objectives, there could exist multiple congestion control approaches for the same objective. Segregation solution may require a large number of traffic classes, which may not be supported by currently available CoS tags; or existing commodity switches may not have sufficient number of queues [24]. More importantly, network needs to schedule and map flows to separate queues and, as described below, none of the existing scheduling policies (*e.g.*, Strict Priority, Weighted Fair Queueing) can address our coexistence problem..

Strict Priority (SP) To highlight the limitations of SP scheduling, we consider the Coexist-TS/DS scenario (as above) and map TS and DS flows to two separate queues. We alternately assign higher priority to TS and DS flows.

The flows mapped to lower priority experience sever performance degradation as, due to the work conserving nature, the flows mapped to higher priority grab most of the network bandwidth. Therefore, both the TS and DS flows get hurt when mapped to lower priority as shown in Fig. 3(a-b).

Weighted Fair Queueing (WFQ) The limitations of strict priority (SP) can be addressed by assigning weights to different priority queues, which is WFQ scheduling discipline. To mitigate interference, in the above example, the queue weights, of the queues belonging to TS and DS flows, can be set according to the ratio (TS/DS) of the flow demands on every link in the network. However, datacenter traffic is highly dynamic [36, 37], and it is hard to keep an update-to-data stats of flows, with

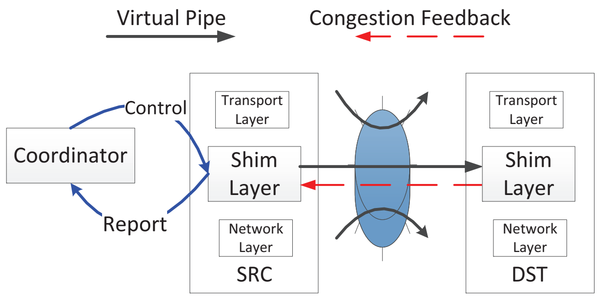


Figure 4: C3 Overview.

different performance goals, on every link in the network. More importantly, it is impossible to reconfigure the switches' setting throughout the whole network in a timely manner: usually such an network update takes at least tens of milliseconds [38, 39].

To highlight the detrimental effects of configuration delay on DS flows, we repeat the coexist-TS/DS experiment and configure queue weights such that the requirements of both the DS and TS flows are satisfied. Fig. 3(c) shows that: greater the difference in old and new weights is, and higher the configuration delay is, more the deadline-sensitive flows miss their deadlines.

3 C3 Design

The key insight behind C3 design is that the coexistence of flows can be achieved by exposing a vision to each control mechanism that it is the only congestion control protocol. By introducing *virtual elastic pipe* abstraction, C3 decouples the congestion control among flows with different performance objectives, from the congestion control among flows with the same objective.

Overview. C3 has two main system components: a *shim layer* at each endhost, and an embarrassingly-centralized *coordinator*, Figure 4. The shim layer adds a pipe for each of the performance objective at the endhosts, where a pipe is a congestion-control tunnel between a source and a destination, that contains all the flows with the same performance objective and control mechanism. Besides the four main performance objectives, C3 also sets up a fifth category for best-effort flows. As a result, for each source destination pair, there would be at most five pipes at each host. C3 is readily deployable as it only needs changes at the endhosts and does not require custom hardware changes or any centralized per-flow scheduling mechanism.

The *shim layer* enables coexistence of different objectives by adopting following mechanism: (1) the shim layer derives a *bandwidth guarantee* and a *network weight* value, based on the flows contained inside each pipe, and reports them to the coordinator periodically; (2) it enforces the virtual elastic pipe abstraction to all outgoing flows, and receives the congestion feedback from the receive end shim layer of each pipe; (3) based on the congestion feedback, bandwidth guarantee and network weight, it performs congestion control to adjust the rate limit of each pipe's ingress queue; (4) it enforces prioritized scheduling inside the queue of each pipe to further improve performance.

The *coordinator* monitors update-to-date bandwidth guarantee and network weight information from each pipe. The coordinator has a global view of the whole topology, so if necesssary it can update per-pipe guarantee and weight. For example, if a link is oversubscribed according to all its passing-through pipes' bandwidth guarantee, the coordinator indicates the respective shim layers to perform admission control for specific pipes: as a results some of the flows are demoted to the best-effort pipe, where all contained flows share a fixed minimum guarantee (*e.g.*, 1 Mbps in C3). To enforce network-level sharing policy, the coordinator determines a scale factor for each performance objective. The derived weight value of each pipe is multiplied by this scaling factor in the shim layer.

Below we discuss shim layer design in detail. We leave the algorithms of deriving per-pipe guarantee and weight to Section 4.

Virtual pipe abstraction. The virtual pipe abstraction is realized by encapsulation. For each pipe, the shim layer adds an additional IP/UDP encapsulation to all passing-through packets. The five-tuples in the outer IP/UDP header can uniquely identify a pipe. This design also facilitates pipe-level congestion control: ECN marking is done in the encapsulating IP headers and the inner IP headers are left untouched for flow-level mechanisms (such as, DCTCP, D^{2}TCP, L^{2}DCT etc.,). The inner IP headers are marked only at the shim layer queues.

Pipe-level congestion control. In the pipe level congestion control, C3 uses a DCTCP-like ECN congestion signal. For each pipe, it maintains two rates, minimum bandwidth guarantee rate (denoted as R_G) and weighted work conservation rate (denoted as R_w). For work conservation rate, R_w, C3 shim layer uses a weighted fair sharing based congestion control mechanism to maximally utilize any available spare bandwidth in the network. The work conservation rate is increased, when there is no congestion, using a weighted additive increase algorithm. On experiencing network congestion, each pipe reduces it's transmission rate based on the extent of congestion in the network and pipe weight. The work conservation rate is lower bounded by the minimum guarantee rate R_G.

The shim layer keeps track of all the pipes and if there is no congestion in the network, shim layer increases the work conserving rate of a pipe as:

$$R_w^{A\to B} = (1+w) * R_w^{A\to B}, \qquad (1)$$

where, w is the weight assigned to a flow, as discussed later. Similarly, upon detecting congestion, shim layer reduces the connection rate as:

$$R_w^{A \to B} = (1 - \alpha * w) * R_w^{A \to B}, \tag{2}$$

where α is the level of congestion in the network. C3 computes the level of congestion from the fraction of marked packets, similar to DCTCP [6].

In-queue scheduling. The ingress queue at the shim layer is the only congestion point for transport protocols. Therefore, without worrying about the congestion inside switches, more efficient scheduling mechanism could be enforced at the host. C3 uses priority sub-queues to support flow prioritization inside a pipe's ingress queue. By mimicking the well-known scheduling algorithm of the performance objective for each pipe, the application performance can be further improved.

More specifically, for DS flows, C3 emulates the Earliest-Dead-First (EDF) algorithm. There are two weighted sub-queues, one has absolute larger weight value over the other (*e.g.*, 95% v.s. 5%). The shim layer maps the flow with the earliest deadline to the top priority sub-queue, and rest of the flow of the same pipe to the lower priority sub-queue. It moves the flow with the next earliest deadline to the top sub-queue, when the highest priority flow completes. C3 favors large relative weight, instead of absolute priority, settings of sub-queues, to prevent the starvation phenomenon of flows in the lower priority sub-queue. Similarly, to improve flow completion times, C3 emulates the Shortest-Job-First (SJF) scheduling for PS flows. The flow with the smallest size is assigned to the high priority sub-queue and rest of the flows within the pipe are assigned to the lower priority sub-queue. This way, the shortest flow grabs most of the connection bandwidth and finishes quickly.

4 Pipe Guarantee and Weight Derivation

To achieve uniform control across all the objective pipes, C3 converts the network demands of all the flows in a pipe to two pipe-level congestion control parameters: minimum bandwidth guarantee and sharing weight. C3 needs a series of algorithms for different kinds of flow pipes, such as throughput-sensitive (S), deadline-sensitive (DS), and latency-sensitive (LS).

For flows, competition with and without C3 may be different. Take priority-sensitive flows for example, assume the priority is flow $f_1 > f_2 > f_3 > f_4$; f_1 and f_2 share the same source-destination pipe, and f_3 and f_4 share the same source-destination pipe. Prior to C3, and without the concern of coexistence, all PS flows may compete independently. With C3, now f_1 and f_2 are in the same pipe, which competes with another pipe that contains f_3 and f_4. The principal of a guarantee and weight derivation algorithm is that: for a group of flows with the same performance objective, using a datacenter network exclusively, the control mechanism should achieve comparable performance with and without C3.

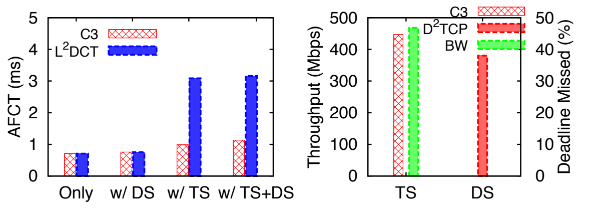
Throughput-sensitive applications usually define minimum bandwidth requirements as: for a groups of flows, guarantee a minimum bandwidth from source to destination. As a result, it is convenient to define bandwidth guarantee for TS pipes. Also, C3 assigns weight values to pipes proportional to their minimum guarantee, R_G. Below we present the algorithms for other pipes.

Deadline-sensitive pipe. Deadline-sensitive flows require to transfer a flow of certain size within a deadline. It is well known that for a single flow f with given size S_f and deadline TD_f, a guaranteed bandwidth no less than S_f/TD_f can meet it's deadline. While for a group of deadline-sensitive flows in the same connection, simply adding their minimum bandwidth requirement together may not be optimal, which can unnecessarily increase demanded bandwidth. Even worse, DS flows come and go; as a result, the derived bandwidth guarantee value should also be dynamic to keep up.

C3 dynamically update the required minimum bandwidth of each DS pipe based on the sizes and deadlines of all active flows in the pipe. The minimum bandwidth is lower bounded by the earliest deadline flow plus a small percentage of headroom reservation. The purpose of reservation is to match the in-queue priority scheduling design: a small headroom (*e.g.*, 5%) can keep low priority flows from starvation. An obvious upper bound for the minimum bandwidth is the sum of all the minimum bandwidths of all the flows in the pipe. We use a binary search between lower and upper bounds to find the minimum required pipe guarantee: for a target guarantee, we use EDF to test its applicability; the target value increases if the test failed, and decreases otherwise. To account for forthcoming DS flows, in the search test our algorithm also systematically under-utilize the target guarantee in the future timeslot. Similar to TS pipe, the weight is also proportional to the guarantee.

Latency-sensitive pipe. LS flows's network latency can be provided by a guarantee larger than their average bandwidth requirement, and a token bucket to absorb the burst. SILO [19] sets the LS guarantee to be 2.2 times of all LS flow's average bandwidth requirement, together with a 5 packet token depth, and reduces the latency to 0.2%. The challenge in this cases is how to derive the average bandwidth requirement.

To compute bandwidth guarantees, C3 leverages insights from how existing transport protocols work. For example, most of the existing protocols start with initial congestion window of two packets (such as DCTCP, L^2DCT, etc.,), and to avoid timeouts, they need at least one ack per RTT. Therefore, we set average bandwidth



(a) (b)

Figure 5: (a) PS flows' AFCT, (b) Throughput (TS) and deadline (DS) performance, w/ & w/o C3.

of a pipe to be at least $2 \times N_f$, where N_f is the number of LS flows in each pipe. There is no need to set weight for LS flows.

Priority-sensitive pipe. For each PS pipe, C3 only allocates a minimum value of bandwidth guarantee (*e.g.*, 1pkt/2RTT). The rational behind this choice is that: coexistence of flows with different performance objectives is more or less an optimization problem. Given the performance constraints of TS, DS and LS flows, PS flows' performance is the optimization objective.

In C3, each PS flow is attached with a global priority. Similar to pFabric [14], we use flow size as the priority, and let weight value be the inverse of a flow's priority. The weight of a pipe is the sum of all the weights of the contained flows.

Best-effort pipe. As mentioned above, C3 only allocate a minimum value of bandwidth guarantee (*e.g.*, 1 Mbps) for a BE pipe. To approximate the fair sharing nature, we set weight of a BE pipe to be proportional to its number of flows.

5 Testbed Verification

Implementation: We implement the C3 shim layer as a Netfilter kernel module in Linux. Two hash based flow tables, one for send and one for receive, are used for packet classification and for tracking per-flow state. To enforce accurate rates over short timescales, we use Linux high-resolution kernel timer, HRTIMER, for our rate limiters.

At the sender, we use the LOCAL_OUT hook to intercept all outgoing packets to mimic virtual pipe abstraction. Each outgoing packet is first matched against the send table and then encapsulated using pipe-level IP/UDP header encapsulation. To support ECN, the ECN-capable (ECT) codepoint is marked in every packet's IP header. The packet is then forwarded to a rate limited per-pipe queue.

At the receiver end, we use the LOCAL_IN hook to intercept all incoming packets. Each packet is matched against the receive table, and its ECN bit is checked. In each control interval (5ms), the receiver shim layer calculates the fraction of ECN marking packets and delivers this information back to the sender that uses it to perform pipe-level congestion control.

Experiments: For Testbed experiments, we use single rack with six DELL servers (with 1G NICs) and PICA-8 Gigabit switch with ECN support enabled.

For evaluation, we consider three kind of traffic, throughput-sensitive (ElasticSwitch), deadline-sensitive (D^2TCP) and priority-sensitive (L^2DCT). In this setting, the aggregate requirement of TS flows is 400 Mbps and that of DS flows is 250 Mbps. We assign same network weight values to all three objectives and all the flows have the same destination server. Within a 1 Gbps link and the same weight, we expect the PS pipe also get around 250 Mbps bandwidth share. We start two long lived bandwidth sensitive flows with 200 Mbps requirement and use ElasticSwitch protocol, $SRC1$ to DST. We also simultaneously start: i) 10 deadline-sensitive flows, periodically (0.5 sec) from $SRC2\&SRC3$ to DST and ii) 10 priority-sensitive flows from $SRC4\&SRC5$ to DST.

When these protocols coexist without C3 support, the PS and DS flows are hurt, whereas TS flows experience no interference. With C3 support enabled (Figure 5(a)), the AFCT of L^2DCT flows is improved by $4\times$ and is almost the same as PS flows do not experience any interference in the network. With C3, DS flows do not miss any deadlines thus improving deadline met performance by up to 40% (Figure 5(b)). C3 improves the performance of DS and PS flows, while meeting bandwidth requirement of TS flows, achieving an average throughput same as ElasticSwitch (Figure 5(b)).

6 Large-scale NS-2 Simulations

Our evaluation centers around the following questions:

- **Can C3 improve the performance of different flows when they coexist?** We evaluate the scenarios where flows with different performance objectives coexist in the data center network. C3 is able to support the performance goals of difference protocols like ElasticSwitch [17], L^2DCT [33], D^2TCP [20] simultaneously across a wide range of workloads such as Websearch [6], Data-mining [28] and MapReduce [40]. Our evaluation shows that C3 improves performance by up to $3\times$ in terms of AFCT and $4\times$ in terms of deadline met when throughput-sensitive, deadline-sensitive and priority-sensitive traffic coexist in the network. C3 is also able to achieve better average throughput for the throughput-sensitive applications.
- **Can C3 achieve the same or even better performance when only flows of the same objective exist in the network?** In our evaluation, we show that C3 can achieve similar or better performance for protocols when they exist in the network alone. At high

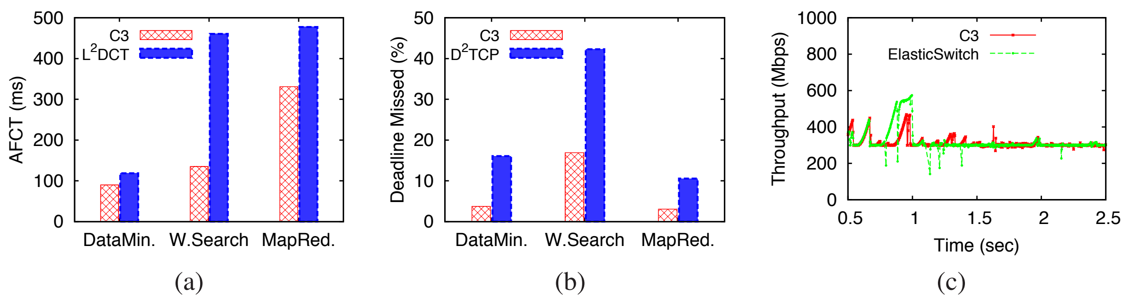


Figure 6: Coexistence performance (a) Priority-sensitive flows' AFCT, (b) deadline-sensitive flows' missed deadline, (c) throughput-sensitive (datamining workload) flows' bandwidth, with and without C3 support for different workloads

loads, C3 can reduce the deadline miss rate and improve AFCT by up to 2×.

- **Does C3 consistently perform well?** We test C3 performance in bursty and oversubscribed network settings. The results demonstrate that with C3 enabled, most protocols consistently deliver better performance.

Evaluation Setup: We use a tree topology for ns2 simulations, unless specified otherwise. The capacity of edge links is 1 Gbps and core links is 10 Gbps. We assume ECN capable switches with 250 KB buffering and maximum end-to-end RTT of 300 usec [16].

We model three traffic classes. Class-I traffic belongs to deadline-sensitive small message application and requires deadline guarantees. Class-II traffic is throughput-sensitive, which represents large data parallel applications and are only concerned with bandwidth guarantees. Class III traffic is similar to Class-I, but has different objective than deadline.We replicate Web-search [6], Data-mining [28] and MapReduce [40] like workloads for Class-I and Class-III traffic. This setting coarsely models the workload for OLDI applications and distributed storage.

In our experiments, throughput-sensitive traffic uses ElasticSwitch [17] protocol, deadline-sensitive traffic uses D^{2}TCP [20] and priority-sensitive traffic uses L^{2}DCT [33] transport protocol at the endhosts. We use default parameters, from respective papers, for each of the protocol. Class-I application generates traffic according to a poisson process, such that the average bandwidth requirement of this class is 300 Mbps to meet it's requirements. The deadlines are exponentially distributed using guidelines from [33]. For Class-II, we use a single application of throughput-sensitive flows, with 300 Mbps requirement, running on each server. The traffic distribution of Class-III is similar to Class-I, except that it does not have any deadline constraints and the goal is to minimize AFCT. By setting the same weight value, we also expect a Class-III pipe to achieve a target bandwidth of 300 Mbps. We use this setup for our experiments below, unless specified otherwise.

6.1 Comparison in Coexistence Scenario

In this section, we evaluate the impact of interference, on application performance, when the flows with different performance objectives coexist in the network.

Coexist-TS/DS/PS Performance: In the first setup, we assume that flows of the three objectives (i.e., throughput-sensitive, deadline-sensitive and priority-sensitive) coexist in the network. The aggregate workload generated by traffic from three objectives is 900 Mbps, thus generating a network load of 90%.

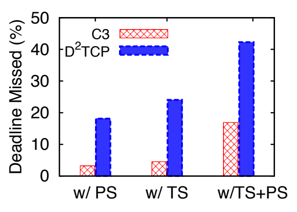
Figure 6 shows that when the three objectives coexist without C3 support, they hurt each others' performance. Data-mining workload is more skewed and more than 80% of the flows are less than 2 KB, therefore most of the flows finish quickly and C3 only improves it a little. Web-search workload has more diverse set of flows, which can be relatively long lived and therefore experience more network interference. C3 mitigates network interference and improves Web-search's AFCT by 3× compared to network without C3 support.

C3 reduces deadline missing ratio by 4×/4×/2× for the Web-search, Data-mining and MapReduce workloads respectively (Figure 6(b)). Without C3, throughput-sensitive flows' achieved bandwidth for Data-mining workload fluctuates a little and goes below it's bandwidth guarantees (Figure 6(c)). With C3 enabled, the achieved bandwidth meet almost perfectly with their guarantee. We observe similar trends for Web-search and MapReduce workloads.

Next, we further explore the performance of deadline-sensitive and priority-sensitive applications in different coexistence scenarios.

Deadline-sensitive applications' performance hurts in the presence of interference traffic without C3. To evaluate the impact of interference, we consider coexisting with Web-search workload and each application has network load of 300 Mbps.

When coexisting, DS flows get hurt in all the scenarios. Figure 7(a) shows that DS flows miss lots of deadlines because of interference.,In all three possible sce-


(a)

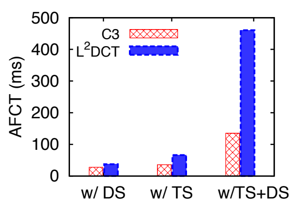
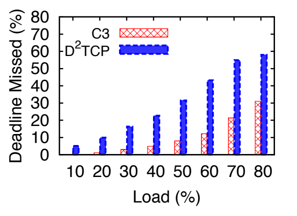

(b)

Figure 7: Performance of (a) Deadline-sensitive (b) Priority-sensitive, under Web-search workload.


(a)

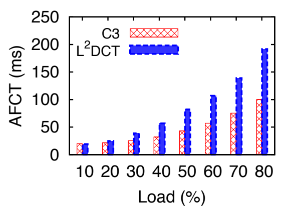
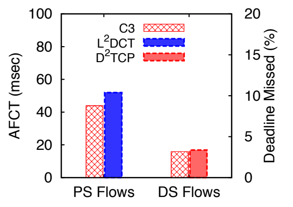

(b)

Figure 8: Isolation scenario (a) deadline-sensitive flows' missed deadline, (b) Priority-sensitive flows' AFCT.


(c)

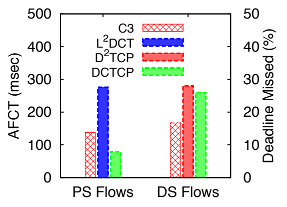

(d)

Figure 9: C3 Dynamics (a) Over-subscribed, (b) Different transport.

narios of coexistence with PS and TS flows, C3 is able to reduce the deadline miss ratio by up to 3×.

Priority-Sensitive applications' performance also hurts a lot in the presence of interference traffic. To evaluate the impact of interference, we consider Web-search workload and each application has network load of 300 Mbps.

When coexisting, PS performance hurts more in the presence of TS and TS+DS flows in the network. Throughput-sensitive flows sometimes get bandwidth less than their guarantees (not shown), whereas PS flows hurt severely. Throughput-sensitive flows grab the network bandwidth and fill switch buffer, and as a result PS flows experience large queuing delays, hence their AFCT increases in Figure 7(b). As a comparison, when C3 support is enabled, throughput-sensitive flows achieve bandwidth guarantee and the AFCT of PS flows reduces by 4×. However, when coexisting with only DS flows, the AFCT is not hurt. The reason is that there is sufficient available capacity in the network and flows do not experience significant queuing delays.

6.2 Comparison in Isolation Scenario

In this setup, we evaluate performance of C3 for deadline-sensitive and priority-sensitive objectives when they have exclusive access to the network and compare it to the performance of D^2TCP and L^2DCT protocols respectively. The goal is to evaluate if, in the absence of coexistence, C3 can achieve similar performance as the transport protocols across a range of network loads. We use Web-search workload for the DS & PS applications and use same network setup as in section 6.

For deadline-sensitive applications, C3 meets more deadlines across all the network loads and reduces the deadline missing by up to 2.5× (Figure 8(a)). At lower loads ($< 30\%$), C3 can meet almost all the deadlines, whereas, D^2TCP flows miss up to 18% deadlines. This shows that C3 not only eliminates interference among flows of different objectives, but also expose the new chance of performance improvement via the abstraction of virtual pipe congestion-free pipe.

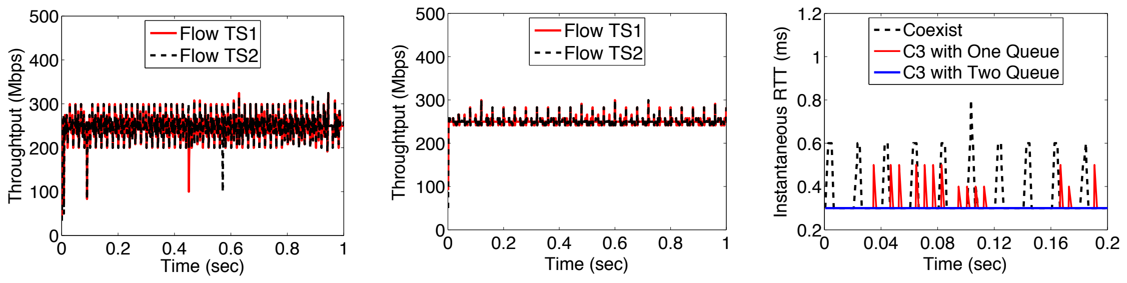
For priority-sensitive applications, C3 is able to minimize the AFCT and achieves performance similar to L^2DCT, as shown in Figure 8(b). C3 achieves similar performance as L^2DCT at low loads and improves AFCT at higher loads by eliminating the inter-flow interference via its priority sub-queues.

6.3 C3 Dynamics

In this section, we evaluate C3 performance under network oversubscription, using different transport protocols, and bursty traffic scenarios.

Over-subscribed Network Scenario: We evaluate C3 on a 3:1 oversubscribed network. In this topology, we have 4 ToR switches, each connecters to 30 hosts with 1 Gbps links and connects to core switches using 10 Gbps links. We generate east-to-west traffic: all traffic is inter-rack. Note that load is calculated based on network core load compared to previous scenarios, where the load was generated based on the edge links. We consider three applications, PS, DS and TS in the network. We have 10 TS flows with minimum guarantee of 100 Mbps. PS and DS flows follow the Web-search workload and have average network load of 300 Mbps each. The PS/DS flows follow exponential arrival rate. Figure 9(c) shows that C3 reduces AFCT of priority-sensitive flows by 20% and reduces deadline missing ratio of deadline-sensitive traffic only marginally.

Different Transport Protocol: Figure 9(d) shows that even with DCTCP as transport protocol, C3 improves the application performance compared to the original protocols. The AFCT of PS flows improves by 3× as these flows get equal share of the network bandwidth as D-



(a) (b) (c)

Figure 10: (a) throughput-sensitive flows' achieved bandwidth when coexist; (b) throughput-sensitive flows' achieved bandwidth with C3 support; (c) latency-sensitive flows RTT, with and without C3 support.

S flows. DCTCP is more aggressive in increasing rate than the L^{2}DCT, for large flows DCTCP increases congestion window by 1 pkt/RTT whereas L^{2}DCT increases by the factor k, where $k \in 0.5, 1$, which depends on the flow size. The performance of deadline-sensitive performance is also better than D^2TCP protocol, however, the deadline performance degrades compared to the scenario where C3 uses D^{2}TCP as the transport protocol. The reason is that now both PS and DS flows share network fairly and benefits of D^2TCP at the transport layer are lost.

Bursty traffic arrivals: We evaluate C3 performance under bursty traffic arrivals. We consider all four kinds of traffic and flows starts simultaneously in the experiments. In this setting every flow category has the same minimum bandwidth requirement of 500 Mbps, and the same network weight value. We use a dumbbell topology to evaluate interference effect. We start two bandwidth sensitive flows with 250Mbps requirement and use ElasticSwitch protocol. We also simultaneously start: i) 10 deadline-sensitive flows, using D^{2}TCP, periodically and ii) 10 latency-sensitive flows, using L^{2}DCT. The bottleneck link capacity is the sum of passing-through flows' minimum requirements.

When these protocols coexist without C3 support, they hurt each others' performance. Throughput-sensitive flows' achieved bandwidth, when coexist, shown in Fig. 10(a), fluctuates a lot and goes below it's bandwidth guarantees. About 9.6% D^{2}TCP flows miss their deadlines and the AFCT of L^{2}DCT flows increases by 2.05%. Latency-sensitive flows are also severely hurt when coexist with other flows, as shown in Fig. 10(c). However, with C3 support enabled, throughput-sensitive flows' achieved bandwidth meet almost perfectly with their guarantee (Fig. 10(b)); the deadline miss ratio reduces to 0.2%; and the AFCT of L^{2}DCT flows is 17.34 ms, almost the same as with an exclusive 500 Mbps link. With C3, the latency-sensitive flows improve, but still far from optimal (also in Fig. 10(c)) because LS flow's packets get queued behind other flows' packets in the switch buffer. We also explore a two-queue C3 version, where latency-sensitive flows enter a high priority switch queue. As shown in Fig. 10(c), now latency-sensitive flows' instantaneous RTT approximates minimum value. The spare bandwidth, left by latency-sensitive flows, is equally shared by other three kinds of flows in the lower queue, and their performance is improved.

7 Discussion and Conclusion

One concern is how to use C3 in a multi-path topology like VL2. In multi-path topologies such as VL2 [28], ECMP relies on the flow header information to hash flows to different paths for load balancing. C3 encapsulates flows to tunnels, which is significantly less in number and large in size. One work-around is that we can use a range of UDP source port number for each pipe, hence packets in the same pipe can be load balanced to multiple paths via ECMP hashing.

C3 enlights us to rethink the position of congestion control in datacenter networks. There exist two kinds of stakeholders in a datacenter network: the applications and the Cloud administrators. An individual application's congestion control only tries to achieve its own performance objective. With a global view of the network, the Cloud administrators need to maximize the chance of meet all applications' network demand, if possible. This paper paves a way to realize this vision and decouples the congestion control among different performance objectives, from the congestion control among flows with the same objective. This work shows that simple solutions can be adopted to eliminate the interference among different applications and therefore realize the peaceful coexistence of applications with difference performance objectives in the same datacenter network.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[3] Facebook future-proofs data center with revamped network.

[4] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. *ACM SIGCOMM CCR*, 41(4):243–254, 2011.

[5] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-driven bandwidth guarantees in datacenters. In *ACM SIGCOMM*, pages 467–478. ACM, 2014.

[6] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM CCR*, 41(4):63–74, 2011.

[7] Jeffrey Dean and Luiz André Barroso. The tail at scale. *ACM*, 56(2):74–80, 2013.

[8] Virajith Jalaparti, Peter Bodik, Srikanth Kandula, Ishai Menache, Mikhail Rybalkin, and Chenyu Yan. Speeding up distributed request-response workflows. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 219–230. ACM, 2013.

[9] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[10] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

[11] Cliff Engle, Antonio Lupher, Reynold Xin, Matei Zaharia, Michael J Franklin, Scott Shenker, and Ion Stoica. Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 689–692. ACM, 2012.

[12] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM*, pages 431–442. ACM, 2014.

[13] Nikhil Bansal and Mor Harchol-Balter. *Analysis of SRPT scheduling: Investigating unfairness*, volume 29. ACM, 2001.

[14] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM CCR*, 43(4):435–446, 2013.

[15] Chi-Yao Hong, Matthew Caesar, and P Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM CCR*, 42(4):127–138, 2012.

[16] Ali Munir, Ghufran Baig, Syed M Irteza, Ihsan A Qazi, Alex X Liu, and Fahad R Dogar. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM*, pages 491–502. ACM, 2014.

[17] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C Mogul, Yoshio Turner, and Jose Renato Santos. Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM CCR*. ACM, 2013.

[18] Alan Shieh, Srikanth Kandula, Albert Greenberg, and Changhoon Kim. Seawall: performance isolation for cloud datacenter networks. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 1–1. USENIX Association, 2010.

[19] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable message latency in the cloud. SIGCOMM, 2015.

[20] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM CCR*, pages 115–126, 2012.

[21] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM CCR*, pages 50–61. ACM, 2011.

[22] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *Proc. ACM Conf. on Special Interest Group on Data Communication*, pages 537–550. ACM, 2015.

[23] Chuanxiong Guo, Guohan Lu, Helen J Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proc. of the 6th Intl. Conf.*, page 15. ACM, 2010.

[24] Alan Shieh, Srikanth Kandula, Albert G Greenberg, Changhoon Kim, and Bikas Saha. Sharing the data center network. In *NSDI*, 2011.

[25] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival Guedes. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *WIOV*, 2011.

[26] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazieres, Balaji Prabhakar, Changhoon Kim, and Albert Greenberg. Eyeq: practical network performance isolation at the edge. *REM*, 1005(A1):A2, 2013.

[27] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM CCR*. ACM, 2011.

[28] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM CCR*, volume 39, pages 51–62. ACM, 2009.

[29] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *NSDI*, pages 19–19. USENIX Association, 2012.

[30] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized zero-queue datacenter network. In *ACM SIGCOMM*, pages 307–318. ACM, 2014.

[31] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. Queues dont matter when you can jump them! In *Proc. NSDI*, 2015.

[32] Ali Munir, Ihsan Ayyub Qazi, and S Bin Qaisar. On achieving low latency in data centers. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3721–3725. IEEE, 2013.

[33] Ali Munir, Ihsan Ayyub Qazi, Zartash Afzal Uzmi, Aisha Mushtaq, Saad N Ismail, M Safdar Iqbal, and Basma Khan. Minimizing flow completion times in data centers. In *INFOCOM, 2013 Proc. IEEE*, pages 2157–2165. IEEE, 2013.

[34] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-agnostic flow scheduling for commodity data centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015.

[35] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM*, pages 443–454. ACM, 2014.

[36] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[37] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[38] Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. zupdate: Updating data center networks with zero loss. *ACM SIGCOMM CCR*, 43(4):411–422, 2013.

[39] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 539–550. ACM, 2014.

[40] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The case for evaluating mapreduce performance using workload suites. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 390–399. IEEE, 2011.