Computer Networks 152 (2019) 154–166

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# Uranus: Congestion-proportionality among slices based on Weighted Virtual Congestion Control

Jiaqing Dong[a], Hao Yin[a,*], Chen Tian[b], Ahmed M. Abdelmoniem[c], Huaping Zhou[b], Bo Bai[c], Gong Zhang[c]

[a] Department of Computer Science, Tsinghua University, China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, China
[c] Future Network Theory Lab, Huawei, Hong Kong, China

ARTICLE INFO

*Article history:*
Received 21 May 2018
Revised 21 December 2018
Accepted 29 January 2019
Available online 1 February 2019

ABSTRACT

Modern data centers are the host for multitude of large-scale distributed applications. These applications generate tremendous amount of network flows to complete their tasks. At this scale, efficient network control manages the network traffic at the level of flow aggregates (or *slices*) who need to share the network with respect to operator's proportionality policy. Existing slice scheduling mechanisms can not meet this goal in multi-path data center networks. Hence, in this paper, we aim to fulfil this goal and satisfy the *congestion proportionality* policy for network sharing. The policy is applied to the traffic traversing congested links in the network. We propose Uranus, a novel slice scheduler based on a combination of flow-level control mechanisms. The scheduler implements two-tier weight allocation to individual flows. Then, relying on a non-blocking big switch abstraction, slice weights are allocated at the inter-rack level by aggregating the weights of rack-to-rack flows. Finally, Uranus can dynamically divide the rack-level weight to its constituent flows. We also implement Weighted Virtual Congestion Control (WVCC), an end-host shim-layer that enforces weighted bandwidth sharing among competing flows. Trace-driven NS3 simulations demonstrate that Uranus closely approximates the congestion-proportionality and is able to improve the proportional fairness by 31.49% compared to the state-of-the-art mechanisms. The results also prove Uranus's capability of intra-slice scheduling optimization. Moreover, Uranus's throughput in Clos fabrics outperforms the state-of-the-art mechanisms by 10%.



## 1. Introduction

Data center networks (DCN) are shared by multitude of large-scale distributed applications, such as search engine, advertising, video streaming, and e-Business. These applications generate tremendous amount of network flows to complete their tasks. Different flows may have different performance objectives because of their service requirements, even if they belong to the same application. For instance, deadline-sensitive flows need to be finished before deadline [19,42,45,48], while others are completion-sensitive and should be finished as soon as possible [3,5,14,19,25–27,33,47]. In this paper, we borrow the network slicing concept from 5G standards [4]. A *slice* consists of all flows in certain type of applications that share the same application-level performance objective, such as deadline-sensitive or latency-sensitive. It is more preferable for DCN administrators to manage network traffic with the granularity of *slices* instead of individual flows [41].

Congestion in data center networks is common since these networks are usually oversubscribed, especially for links between the rack switch level compared to the available bandwidth at the core switch level in that there is larger bandwidth fan-in. It is reported that packet drops due to congestion can be observed when the whole network utilization is around only 25% [40]. During congestion, slices compete with each other for available bandwidth on congested links. Hence, to achieve performance isolation, administrators can assign weights to different slices and a slice should get a share of the network proportional to its weight. This weight-based management policy should be enforced and respected.

Existing works fall short to meet this proportional-sharing requirement in data center networks rich in multipaths. For instance, the prior work Stacked Congestion Control (SCC) [41] schedules traffic at slice level and aggregates flows in a slice with the same source-destination pair into a tunnel. Firstly, SCC is not applicable to state-of-the-art fabrics for DCN, which usually adopts Clos-based

* Corresponding author.
*E-mail address:* h-yin@mail.tsinghua.edu.cn (H. Yin).

https://doi.org/10.1016/j.comnet.2019.01.039
1389-1286/

multipath topologies. Unless the whole network can be treated as a single bottleneck, it is hard for SCC to even approximate network proportionality. Secondly, a tunnel-based approach is unfriendly to load balancing schemes, which results in throughput decrease and instability (Section 2).

In this paper, we present Uranus, a traffic scheduling framework supporting proportional allocation of network resources among slices based on flow-level traffic management mechanism. Specifically, Uranus targets *congestion-proportionality*, which restricts the network proportionality requirement to the traffic traversing congested links [34]. Uranus achieves congestion-proportionality among slices through the *Proportional Sharing at Network-level* (PS-N) allocation scheme proposed by FairCloud [34].

We summarize contributions of this paper as follows:

1. **We propose a better and realistic topology abstraction for state-of-the-art DCN fabrics.** By exploring state-of-the-art load balancing schemes, the in/out traffic to/from each rack can be evenly spread to rack-to-core links. Thus, the whole core switch level can be abstracted as a non-blocking big switch. With the big-switch abstraction, proportional scheduling among slices can be achieved.
2. **We propose and enforce rack-level proportionality among slices.** Based on the topology abstraction, Uranus introduces a two-tier weight allocation scheme for allocating weights among individual flows. Uranus aggregates flows in a slice with the same source-destination rack-pair as a (virtual) rack-flow. At the inter-rack tier, a slice's weight is allocated to its rack-flows, following the PS-N scheme. As a result, Uranus can closely approximate the congestion-proportionality among slices.
3. **Uranus supports intra-slice optimization for flows of different objectives.** At the intra-rack tier, Uranus dynamically divides a rack-flow's weight to its constituent flows. This design enables the objective-oriented scheduling inside a slice.
4. **We present and analyze WVCC, a scalable per-flow weight enforcement mechanism.** In Uranus, WVCC works as a flow-level weight enforcement module, which is able to proportionally allocate bandwidth among competing flows with great scalability. Using fluid model analysis, the convergence characteristics of WVCC are mathematically proved (Section 4).

We have evaluated Uranus from several aspects. Concerning the enforcement, WVCC, we evaluate the model with Matlab emulation together with NS3 simuation. The results demonstrate that WVCC is work-conserving and can enforce proportional sharing at flow-level in congested links. We further test WVCC with Linux Kernel implementation and the results also prove WVCC's effectiveness and work-conserving feature. Then we evaluate the whole framework with trace-driven NS-3 simulations. In terms of network proportionality, Jain's-index [21] is adopted as the metric. Uranus closely approximates the congestion-proportionality and improves proportional fairness around 31.49% compared with the SCC design. In terms of intra-slice optimization, Uranus reduces the deadline-miss ration by around 25% on average and reduces AFCT by 20% under high load pressure. Uranus can further improve an individual slice's performance in scenarios where slices with different objectives coexist. Finally, from the perspective of network utilization, the throughput of Uranus outperforms SCC by 10% in Clos fabrics (Section 5).

## 2. Background and motivation

This section first gives a background of data center network topology, followed by a brief introduction of proportionality in data center networks. Then we discuss related works of network sharing in data center environment. Finally, we demonstrate the drawbacks of an existing approach via toy scenarios.

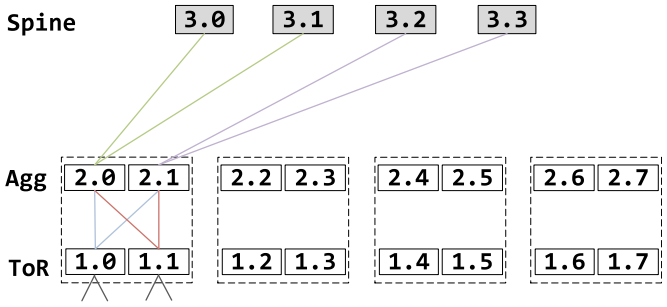



**Fig. 1.** CLOS-based network example: Fat-tree.

### 2.1. Topology of datacenter networks

***Typical data center network topology:*** State-of-the-art fabrics for data centers are usually Clos network alike multipath topologies. For instance, Google's data center network Jupiter and its ancestors [40] together with VL2 from Microsoft [15] all adopt clos-based topology. In a Clos network topology, switches connect with each other and are able to provide non-blocking connections between a large number of input and output ports far beyond the capability of a small-sized switch [11].

Fig. 1 presents a typical Clos-based network topology named Fat-tree in data center. A Fat-tree fabric has three switch layers called spine, Agg (Aggregation) and ToR (Top of Rack) from up to down respectively. Each ToR switch uses half of its ports to connect with servers, with the other half connected to aggregation switches. Similarly, each Agg switch uses half of its ports to connect with ToR switches, and the other half to connect to spine switches. For a K-ary Fat-tree, each Agg switch connects to k/2 ToR & k/2 spine switches while each spine switch connects to k Agg switches distributed in k pods. Fig. 1 gives a 4-ary Fat-tree network.

Bandwidth oversubscription usually happens between the ToR switch layer and upper layers because in reality, ToR switch usually connects to more than k/2 servers. For example, in Fig. 1, if each ToR is connected to 6 servers and 2 Agg switches with all links of the same capacity, then this topology has a 3:1 bandwidth fan-in. In this paper, we use core switch layer to refer to the spine switches of Clos topologies in general, and use rack layer to refer to the underlying layers.

In data centers with clos topology, congestion happens at rack level switch due to incast and outcast [40]. When equipped with state-of-the-art load balancing schemes such as flowlet [13,43] and flowcell [17], the incoming/outcoming traffic to/from each rack in Clos network will be evenly spread to rack-core links. As a result, the whole core switch level can be abstracted as a non-blocking big switch as in pHost [14].

### 2.2. Proportionality in data center networks

***Congestion proportionality:*** Strict network proportionality is inappropriate for data center networks. A slice will be disincentivized to use an uncongested intra-rack link, if it has to yield part of its inter-rack bandwidth to preserve strict network proportionality. FairCloud discusses general proportional network sharing problems and proposes congestion-proportionality [34].

As a trade-off, congestion-proportionality restricts the network proportionality requirement only to the traffic traversing congested links. In other words, if a link is occupied by only one slice then the usage of this link will not be accounted in the network proportionality for that slice. This feature removes the disincentive to use an uncongested path and encourages high utilization. However, even congestion-proportionality can incentivize a slice to ar-

tificially inflate or deflate its real traffic demand. For example, by reducing the traffic in a specific congested link, slice *A* can turn this link from a congested link to an uncongested link. The bandwidth used in this link is then not accounted for proportionality and *A* is able to acquire more bandwidth in other congested links under the same proportionality. By doing this, *A* could increase its allocation while hurting the whole network utilization.

Our key observation is as follows: the oversubscription usually happens to links between the rack level and the core level. These links exert higher impact than links inside a rack in terms of traffic. By always treating all these rack-core links as congested regardless of their real-time load conditions, every byte crossing these links is counted. In this scenario, the utilization incentive consideration of congestion-proportionality can be removed. As a result, congestion-proportionality fits well for these inter-rack links. Consequently, this paper focuses on preserving the congestion proportionality for inter-rack links.

***Proportional Sharing at Network-level(PS-N):*** The weight allocation scheme PS-N [34] aims to approximate network wide congestion proportionality irrespective of the communication patterns among virtual machines (VM). PS-N assigns the weight to a communication pair between VMs *X* and *Y* as:

$$W_{X-Y} = \frac{W_X}{N_X} + \frac{W_Y}{N_Y}, \tag{1}$$

where $W_X$(resp. $W_Y$) is the weight of *X*(resp. *Y*), $N_X$(resp. $N_Y$) is the number of other VMs that *X*(resp. *Y*) is communicating with across the network. Note that Eq. (1) assumes the weight value is attached to a specific VM.

PS-N can completely achieve *congestion proportionality* if one of the following conditions holds: (1) congested links have the same capacity as well as background flows, or (2) the summation of flow weights in any congested link is proportional to its capacity.

### 2.3. Related works of network sharing

Today's data centers are shared by many applications, while flows of different applications may have different performance objectives because of their service requirements. A lot of works have been proposed to regulate network sharing in data center.

***Proportional sharing:*** Seawall provides per-entity weight enforcement for each VM-to-VM tunnel [39], while it does not support bandwidth allocation at flow-level granularity. FairCloud discusses general proportional network sharing problems in [34].

***Bandwidth guarantee:*** SecondNet [16] and Oktopus [6] provide predictable guarantees based on the hose network model. However, they lack work-conservation property and do not utilize the shared network bandwidth efficiently. Gatekeeper [37] and EyeQ [23], built on hose model, can achieve work-conservation. However, these mechanisms are designed specifically for congestion-free core networks [15]. ElasticSwitch [35] transforms the hose model to multiple virtual VM-to-VM connections to provide minimum bandwidth guarantees for each connection. CloudMirror [24] proposes a placement algorithm and leverages application communication patterns to provide bandwidth guarantees. Bandwidth guarantee is proposed to meet the peak requirements, which is either inefficient or suboptimal in terms of flow objectives.

***Multi-tenant multi-objective sharing:*** pFabric [3] and Karuna [9] evaluate the coexistence of the deadline-sensitive(DS) and completion-sensitive(CS) flows by setting absolute priority to DS flows over CS flows. However, performance isolation among flows with the same objective but of different tenants is not considered. The previous work [41] first formulated the coexistence problem of multi-tenant-multi-objective and proposed Stacked Congestion Control (SCC). SCC proves that coexistence of flows with different objectives could cause severe interference due to different control signals and control laws. SCC adopts a tunnel-based solution and introduces several problems. We will discuss details of SCC in the following section.

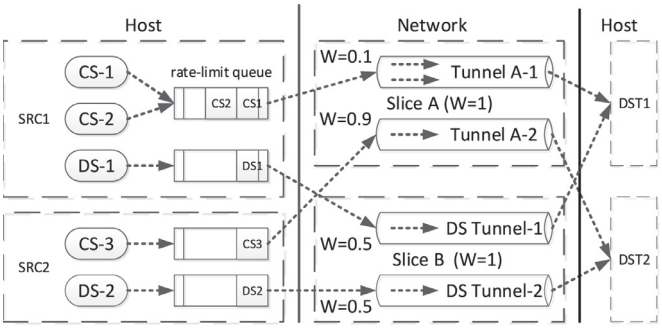



**Fig. 2.** SCC overview.

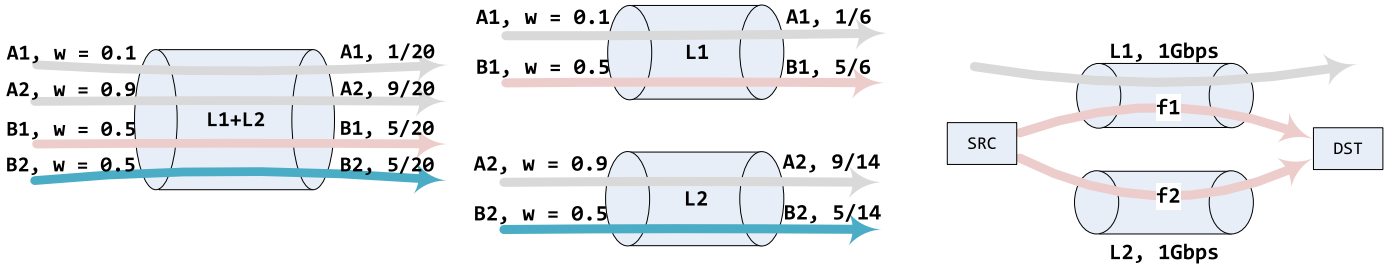
### 2.4. Problems of sharing the network via SCC

Fig. 2 illustrates an overview of SCC. SCC defines the term *division* to denote all flows of a tenant that share the same objective. The tenant-objective term *division* in SCC is semantically equivalent to our application-objective term *slice*, while slice is more consistent with the popular 5G standard definitions. The work in [41] intends to support both performance isolation among slices and objective scheduling inside each slice. The SCC mechanism centers on the *tunnel* design. A tunnel is the basic unit for running weighted congestion control, similar to Seawall [38]. In Seawall, flows of the same source-destination pair are aggregated into a physical tunnel. However in SCC, only flows *in a same slice* with the same source-destination pair are aggregated into a tunnel. SCC supports performance isolation among slices by dividing the weight of a slice to its constituent tunnels. To support objective-oriented scheduling, a tunnel's weight is devided to its contained flows. As the summation of the weight of all tunnels in a slice is fixed, a tunnel's weight can be scaled up/down according to the calculation of other tunnels. Fig. 2 is a simplified example to demonstrate the relation. Assume that two slices *A* and *B* are assigned with the same total weight 1.0. Objective oriented scheduling mechanism may allocate tunnel $A_1$ with weight 0.1 and $A_2$ with weight 0.9, while in slice *B* it gives weight 0.5 to both tunnels.

However, SCC has several drawbacks which will hurt its performance due to neglect of the network topology and inherent defects of the tunnel-based design.

***Proportionality violation:*** Slices should share networks according to the administrators' proportionality policy. SCC's weight allocation scheme can provide proportional sharing of the network only in the condition that the network has a single bottlenecked link. We present two scenarios for the case in Fig. 2. Fig. 3(a) is the single bottleneck scenario: the proportionality relation holds correctly between two slices, and between the tunnels in the same slice. The scenario in Fig. 3(b) has two bottleneck links, where $A_1/B_1$ share the same bottleneck and $A_2/B_2$ share the other. As slice *A* and *B* compete on different bottlenecks with the same capacity, proportionality relation does not hold anymore. Specifically, $A_1$ gets 1/6 of $L_1$ and $A_2$ gets 9/14 of $L_2$. Proportion between $A_1$ and $A_2$ becomes 7 : 27 rather than 1 : 9, while proportion between slice *A* and slice *B* changes to 17 : 25 instead of 1 : 1.

***Network utilization decrease:*** To combat the CPU overhead problem brought by *physical tunnel*, SCC introduces the concept of *virtual tunnel*, where flows in a tunnel are not encapsulated with an additional packet header. A tunnel in SCC uses the aggrated ECN feedback to estimate the network congestion level and adjust its sending rate accordingly. The key idea is to combine the conges-




(a) With a single bottleneck (b) With separate bottlenecks (c) Biased congestion signal of virtual tunnel

**Fig. 3.** Scenarios in which SCC does not work well.

tion signals (*i.e.*, CE-marked packets) from all contained flows to estimate the congestion level along the route, and perform tunnel-level congestion control. When tunnel-level congestion control is applied in a multipath environment, as a tunnel uses different flows' congestion signal, bias is introduced due to different congestion levels among different paths. For instance, as shown in Fig. 3(c), there are two bottlenecks $L_1$ and $L_2$. Two flows $f_1$ and $f_2$ are in the same virtual tunnel, while $f_1$ travels through $L_1$ and $f_2$ travels through $L_2$. $f_1$ shares $L_1$ with many other background flows, so that $L_1$ gets very congested and the congestion level $\alpha_1 = 1$. $f_2$ shares $L_2$ alone and $\alpha_2 = 0$. In this scenario, the tunnel gets non-zero overall $\alpha$ value and reduces the tunnel's sending rate which affects all contained flows. Hence, the throughput of $f_2$ is significantly decreased.

***Scalability problem:*** SCC enforces flow-level bandwidth allocation by assigning a software-based rate limiter for each flow, which will introduce high computational overhead and additional resource consumption [36].

## 3. Uranus design

This section provides an overview of Uranus's design. We first discuss the design requirements of the framework. Then we describe how Uranus fits the model and approximates network-level congestion-proportionality. Finally we discuss the intra-slice scheduling mechanisms of Uranus for optimizing different flow objectives.

### 3.1. Design requirements

(1) **Proportional Sharing at Network-level:** One of the main goals of Uranus is to respect the weight-based network management policy in the multi-tenant data center network scenario. Uranus should be designed to support proportional sharing schemes.
(2) **Fit for state-of-the-art DCN topology:** Uranus is required to work well for the majority of current data center network fabrics, which are usually Clos-alike multipath topologies. Characteristics of clos-based multipath topology should be taken into account when designing the overall scheduling framework.
(3) **Optimize for traffic of different objectives:** Modern data centers are the host for multitude of large-scale distributed applications, whose traffic flows with different objectives coexist in the same network. Another goal of Uranus is to make it possible to optimize scheduling traffic of different objectives.
(4) **Scalable for DCN environment:** In data center network, tremendous flows exist concurrently. Normal software-based rate limiter will not scale for this environment. Uranus requires a scalable and light-weight per-flow control mechanism to meet the harsh requirement in data center networks.
(5) **Work-conserving:** In terms of proportional sharing bandwidth among flows, it is preferable for the scheduling framework to be work-conserving. With work-conserving, utilization of links will be increased.

### 3.2. Uranus with PS-N model

**Why PS-N:** In the long term, traffic is approximately uniformly distributed across racks in data center networks. Hence, condition 1 of PS-N can be closely approximated. As a result, Uranus chooses to follow PS-N to approximate the *congestion proportionality* in data center networks. By following PS-N scheduling mechanism, Uranus can closely approximate congestion proportionality at network-level and meet requirement (1).

**Focus on racks:** Uranus uses a rack to replace the VM in the original PS-N design. Uranus focuses on congestion-proportional sharing across racks in data centers with Clos network, which is the typical topology for modern data centers as discussed in Section 2.2. For a Clos network, by exploring state-of-the-art load balancing schemes such as flowlet [13,43] and flowcell [17], the whole core switch layer can be abstracted as a non-blocking big switch [14]. With the non-blocking big switch abstraction, Uranus is able to focus on the incoming/outcoming traffic to/from each rack. The non-blocking big switch abstraction fits well for modern data center networks, and thus Uranus satisfies requirement (2).

At the rack level, flows from various VMs and servers are aggregated and abstracted into communication patterns among various racks. Since inter-rack communications contain more flows than at inter-host level, they are more consistent and stable than that among hosts [7]. The completion of one host's flows does not affect the established rack-to-rack communication patterns, as long as there are other hosts of the slice in the same rack.

Uranus adopts a two-tier weight allocation. It aggregates flows in a slice with the same source-destination rack pair as a (virtual) rack-flow. At the inter-rack tier, a slice's weight is allocated to its rack-flows, following the PS-N scheme. Then at the intra-rack tier, Uranus dynamically divides a rack-flow's weight to its current running flows. Eventually, each flow is assigned a global weight value. This two-tier design enables objective-oriented scheduling inside a slice, hence requirement (3) is satisfied by Uranus.

The last two requirements (4) and (5) are satisfied by the scalable per-flow weight enforcement mechanism WVCC, which we will discuss later in Section (4).

**Inter-rack weight allocation following PS-N:**

Calculation of weight allocation among slices is outside the scope of this paper. We assume weight of each slice is given as input either by an algorithm that enforces administrator's policy or by a utility maximization function as in [28,34].

It is possible to predict communication patterns of slices among racks [8,32]. Hence in Uranus, we do not require the administrator

**Table 1**
Symbols used.

| | |
|---|---|
| $d_i^j$ | Severity of deadline urgency for flow $f_j$ of rack $R_i$ |
| $r_i^j$ | Priority value for completion-sensitive flow $f_j$ of rack $R_i$ |
| $F_k$ | Number of flows for rack $R_k$ |
| $f_i^j$ | Flow $j$ in rack $R_i$ |
| $W_i^j$ | Weight for flow $j$ in rack $R_i$ |
| $Wr_i$ | Weight allocated to rack $R_i$ |
| $Tc(t)$ | Remaining time needed to complete a flow at time $t$ |
| $D(t)$ | Remaining time before deadline |

to know the exact traffic matrix, instead we assume that communication patterns of slices, i.e, among which racks a slice is communicating, are known by the scheduler and can be used to calculate weight allocations.

Weight of each slice is allocated to cross-rack traffic following PS-N policy. PS-N assigns weight to a rack-flow between rack X and rack Y following the equation:

$$Wr_{X-Y} = \frac{Wr_X}{Nr_X} + \frac{Wr_Y}{Nr_Y}, \tag{2}$$

where $Nr_X$(resp. $Nr_Y$) is the number of other racks with which this slice's hosts in rack $X$(resp. $Y$) are communicating across the core switches.

Assume that a slice $S$ with weight $W_S$ has communications among $n$ racks $R_1, R_2, \ldots, R_n$. Weight of each rack, $Wr_R$ should be calculated ensuring that:

$$W_S = \sum_{i=1,2,\ldots,n} Wr_{Ri}. \tag{3}$$

We discuss how to distribute $W_S$ to $Wr_{Ri}$ in the following section.

### 3.3. Intra-slice scheduling and optimization

In this section we propose the intra-slice scheduling algorithm. The intra-slice scheduling problem can be divided into two sub problems. The first one is how to divide a slice's total weight $W_S$ among communicating racks. The second is how to allocate a rack's weight $Wr_{Ri}$ to its constituent flows. With the ability to allocate bandwidth among flows proportional to their weights, the flow scheduling problem can be reformulated as the problem of weight allocation for flows. Flows inside a slice share the same objective. We discuss these two sub problems in scenarios of *deadline-sensitive (DS)* and *completion-sensitive (CS)* separately. We demonstrate later in evaluation that intra-slice scheduling can significantly improve performance, which accredits the effectiveness of Uranus.

The symbols used are listed in Table 1.

#### 3.3.1. Deadline-sensitive scheduling

The main idea behind deadline-sensitive scheduling is to allocate more resource to flows with earlier deadlines. We use a metric $d$, which will be defined formally later, to quantify the deadline urgency for a flow. Noting that each flow connects two racks, the resource demand of a flow actually indicates the demand of the two racks between which the communication happens. Inspired by this observation, we divide $W_S$ to $Wr_{Ri}$ with the following equation:

$$Wr_{Ri} = \frac{\sum_{l=1,2,\ldots,F_i} d_i^l}{\sum_{j=1,2,\ldots,n} \sum_{k=1,2,\ldots,F_j} d_j^k} \times W_S, \tag{4}$$

where $n$ is the number of racks, $F_j$ is the number of flows for rack $R_j$ and $d_i^j$ measures the deadline severity for flow $f_j$ in rack $R_i$. Eq. (4) ensures that Eq. (3) is satisfied so that the weight distribution of the DS slice among its constituent cross-rack traffic does obey the proportionality policy.

For the second sub-problem, we proposed a straightforward solution. In deadline-sensitive scheduling, we assume that flow sizes and deadlines are exposed to the scheduling algorithm as in [31,42]. We divide the weight of a rack to its constituent flows based on the flows' demand. Under the deadline-sensitive requirement, for a flow with remaining size $S(t)$ and deadline $D(t)$, the minimum bandwidth required is $\frac{S(t)}{D(t)}$. Assuming that current throughput is $B(t)$, then the scale factor for next bandwidth against current throughput should be $\frac{S(t)/D(t)}{B(t)}$. At the given time $t$, the expected completion time $T_c(t)$ for that flow can be calculated as $\frac{S(t)}{B(t)}$. Putting the expression into the previous one, we get the scale factor against current throughput with the new expression $\frac{T(t)}{D(t)}$. Then, we get $d(t) = \frac{T_c(t)}{D(t)}$ as a good scale factor for weight demand as it considers both previous throughput as well as deadline severity thus can be used as an indicator for describing the demand of a deadline-sensitive flow. The principle for deadline-sensitive flow scheduling in the paper is to reduce the deadline-miss ratio by allocating more weight to more likely-to-fail flows. By using d(t) as the competing factor for flows in each bandwidth reallocation round, the scheduler works in a greedy manner to reduce the number of likely-to-fail deadline-sensitive flows.

#### 3.3.2. Completion-sensitive scheduling

The scheduling mechanism for CS slices is similar to that of DS slices. The only difference is that the remaining flow size instead of the remaining time is considered. A straightforward strategy for CS scheduling is the Shortest-Flow-First (SFF) strategy. We use a metric $r$, which we will discuss later, to quantify the priority for a completion-sensitive flow. Then we can divide $W_S$ to $Wr_{Ri}$ with the following equation:

$$Wr_{Ri} = \frac{\sum_{l=1,2,\ldots,F_i} r_i^l}{\sum_{j=1,2,\ldots,n} \sum_{k=1,2,\ldots,F_j} r_j^k} \times W_S, \tag{5}$$

where $n$ is the number of racks, $F_j$ is the number of flows for rack $R_j$, $r_i^j$ measures the priority for flow $f_j$ in rack $R_i$. Similarly, Eq. (5) ensures that Eq. (3) is satisfied so that the weight distribution of the CS slice among its constituent flows accords with the proportionality policy.

When considering allocating the weight of a rack to its constituent flows in completion-sensitive manner, we should always allocate more weight to flows with smaller remaining size according to the SFF strategy. We assume that flow sizes are known for scheduling algorithm as in [31]. For Shortest-Flow-First strategy, the metric $r$ in Eq. (5) should satisfy that flows with smaller remaining size $S$ have larger $r$ value. A simple expression which satisfies this requirement is $r = \frac{1}{S}$. More compelicated expressions are also possible to approximate the SFF strategy, for instance starvation can be take into consideratin. We choose the simplest one in this paper to demonstrate the effectiveness of intra-slice scheduling. Consequently, in Step 1 and Step 2 of Algorithm 2, we replace the urgency value $d(t)$ with the remaining size priority $r(t)$.

Uranus periodically updates weight for each flow with Algorithms 1 and 2 according to the objective type respectively.

Note that Uranus works as a general architecture here, as long as requirement of Eq. (3) is satisfied, the congestion-proportionality requirement among slices will be satisfied. Researchers can developed algorithms for different objectives, or algorithms that also target DS/CS flows but with better performance.

## 4. Weighted Virtual Congestion Control (WVCC)

In this section, we provide details of WVCC, the per-flow weight enforcement mechanism for Uranus. Firstly, we briefly discuss the design requirements for the weight enforcement, inherited from

**Algorithm 1:** Weight allocation for DS slice.

**At the beginning of each decision slot, the slice**
**Step 1:** Update the flow information, $\{T_c(t), D(t)\}$
**Step 2:** Calculate the deadline urgency $d(t) = \frac{T_c(t)}{D(t)}$
**Step 3:** Calculate the total weight for rack $R_i$ with Eq. 4
**Step 3:** For each $f_i^j$ Let $W_i^{j*} = d_i^j(t) \times W_i^j$
**In-slice weight allocation:**
**for** *each flow* $f_i^j$ *in each rack* $R_i$**: do**
  Calculate the scale factor: $lambda \leftarrow \frac{\sum W_i^*}{Wr_i}$
  Update the new weight: $W_i^{jnew} = W_i^{j*}/\lambda$.

**Algorithm 2:** Weight allocation for CS slice.

**At the beginning of each decision slot, the slice**
**Step 1:** Update the flow information, $\{S(t)\}$
**Step 2:** Calculate the remaining size priority $r(t) = \frac{1}{S(t)}$
**Step 3:** Calculate total weight for rack $R_i$ with Eq. 5
**Step 3:** For each $f_i^j$ Let $W_i^{j*} = r_i^j(t) \times W_i^j$
**In-slice weight allocation:**
**for** *each flow* $f_i^j$ *in each rack* $R_i$**: do**
  Calculate the scale factor: $lambda \leftarrow \frac{\sum W_i^*}{Wr_i}$
  Update the new weight: $W_i^{jnew} = W_i^{j*}/\lambda$.

the requirements of Uranus. Secondly, we provide details of VCC, which works as a basis for our design. Then we present detailed design of our scalable per-flow weight enforcement mechanism. Finally, a thorough analysis and evaluation of the model is provided.

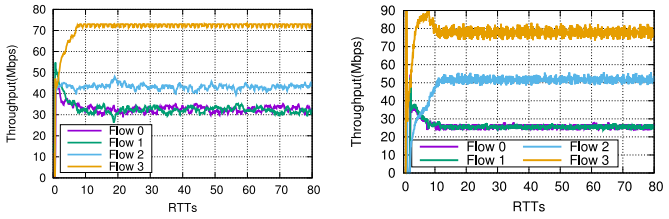
### 4.1. Design requirements

Weighted bandwidth allocation is critical to enforce administrator policy [41]. Uranus requires a per-flow control mechanism which supports proportionality enforcement. The mechanism should be *scalable and light-weight* to fit for data center network, where tremendous flows exist concurrently. In addition, it is more preferable if the enforcement is work-conserving so that the network utilization could be increased.

Traditional software-based rate limiters, such as hierarchical token bucket(HTB) in Linux, do not scale gracefully in data center scenarios in that they introduce high overhead due to frequent interrupts and contention [36].

### 4.2. Differentiated QoS over VCC

Enforcing virtualized congestion control is a new trend for data center networks [12,18]. Public cloud data centers are shared by various tenants running applications inside virtual machines (VM). Guest VMs have different TCP protocol stacks for different application objectives. These TCP versions could rely on different congestion signals (*e.g.*, ECN vs. packet drop) and exert objective-specific control laws. Nevertheless, they cannot share the same underlying physical data center network [41] peacefully. Virtual Congestion Control [12,18] adds a translation layer which "hijacks" the congestion control function in the datapath by modifying congestion signals and RWND of TCP header, allowing guest-VM applications continue to use their legacy TCP implementations without modification. Such *Virtual Congestion Control* mechanisms translate the legacy TCP versions in tenant VMs into a newer congestion control algorithm. Furthermore, the administrator can enforce a uniform




(a) Results of AC/DC (b) Results of WVCC

**Fig. 4.** Per-flow differentiation.

congestion control rule throughout the whole data center with this mechanism.

Virtual congestion control mechanism can also enforce differentiated Quality-of-Service (QoS) for flows. AC/DC emulates a DCTCP-like [1] congestion control algorithm at host virtual switch. AC/DC proposes a per-flow differentiation mechanism by assigning each flow with a priority $\beta$. The back-off phase of DCTCP has been revised as:

$$rwnd \leftarrow rwnd \times \left[1 - \alpha\left(1 - \frac{\beta}{2}\right)\right], \tag{6}$$

where $\alpha$ is a value indicating congestion level along the path calculated by measuring the fraction of CE-marked packets. With Eq. (6), flows with lower priority back-off more aggressively than higher-priority flows.

The per-flow differentiation algorithm based on Eq. (6) only provides qualitative rather than quantitative bandwidth allocation among flows. We use the NS3 simulation to illustrate this issue, where four flows with $\beta$ values [1, 1, 2, 3]/4.0 competing at a single link. As shown in Fig. 4(a), after convergence the flows with the same priority get similar throughput, while flows with higher priority obtain higher throughput.

However, the priority value $\beta$ cannot provide proportional bandwidth allocation among these flows accurately. In fact, $\beta$ in Eq. 6 is a priority value which only describes the back-off strength, without considering the increase phase. We further analyze the effect of $\beta$ in the next subsection with fluid model.

### 4.3. WVCC design

In Uranus, we develop WVCC enforcement for data center networks. Inherited from virtual congestion control mechanisms, WVCC does not require any modifications to legacy TCP stacks in tenant-side VMs. It is a novel per-flow differentiation mechanism capable of proportionally allocating bandwidth among flows.

**Overview:** WVCC is implemented in the datapath of the hypervisor, similar to AC/DC [18] and vCC [12]. With this architecture, Uranus is able to work without any TCP stack modifications in tenant-side environments. The sender and receiver modules work together to enforce the per-flow weighted congestion control with uniform congestion signal(i.e, ECN) through changing the receive window (RWND) field in incoming ACK packets. Specifically, WVCC algorithm calculates the weighted congestion window ($cwnd^*$) value. RWND is modified only when $cwnd^*$ is smaller than the original RWND set by the receiver.

**Algorithm:** WVCC's algorithm builds on top of DCTCP [1]. Switches are required to mark CE bits when packets in buffer exceed a threshold. Similar to DCTCP, we maintain the same variable $\alpha$, which can be used to measure the extent of congestion in the network. WVCC shares same features with TCP, such as slow start, congestion avoidance, and fast recovery. Upon receiving an

ACK, WVCC calculates new congestion window:[1]

$$cwnd^* \leftarrow \begin{cases} cwnd^* \times (1-\alpha/2), & \text{ECE bit of ACK set;} \\ cwnd^* + w/cwnd^*, & \text{otherwise.} \end{cases} \quad (7)$$

where $w$ ($w<1$) is the weight for that flow. When combined with slice scheduling in Section 3.3, $w$ is the per-flow weight updated by scheduling Algorithms 1 and 2. Algorithms 1 and 2 periodically distribute slices' weight to contained flows.

### 4.4. Fluid model

We develop a fluid model to analyze WVCC. The model assumes that all the flows compete on a single bottleneck link with capacity $C$, considering $N$ long-lived flows with different weights $w_i$ ($i=1,\ldots,N$). For each flow $i$, $W_i(t)$ denotes the window size and $\alpha_i(t)$ stands for congestion level along the path. $q(t)$ represents the queue size of the bottleneck switch port.

Firstly, we analyze dynamics of $W_i(t)$. In terms of the weighted additive increase part in Eq. (7), after each RTT, the window will increase by $w_i$. Consequently we have $\frac{dW_i}{dt} = \frac{w_i}{R(t)}$ describing the increase phase. Regarding the multiplicative decrease phase, the window size will be reduced by a factor of $\alpha(t)/2$ when packets are marked (i.e., $p(t-R^*)=1$), and this occurs once per RTT. Hence for this part, $\frac{dW_i}{dt} = -\frac{W_i(t)\alpha_i(t)}{2R(t)}p(t-R^*)$. With all these equations together, we have:

$$\frac{dW_i}{dt} = \frac{w_i}{R(t)} - \frac{W_i(t)\alpha_i(t)}{2R(t)}p(t-R^*). \quad (8)$$

Here, $p(t)=1_{\{q(t)>K\}}$ is a binary variable that indicates the packet marking process at the switch. Specifically, a new packet is marked only when the queue size exceeds the threshold $K$. The packet marking indicator $p(t)$ affects Eqs. (8) and (10) with a fixed approximate delay $R^* = d + K/C$.

Then we analyze the dynamics of $\alpha_i(t)$, which measures the congestion level of the link. $\alpha$ is updated once for every window of data (roughly one round-trip time) as follows:

$$\alpha \leftarrow (1-g)\alpha + gF, \quad (9)$$

where $F$ is the fraction of packets marked in the most recent window of data, and $0<g<1$ is a fixed update coefficient. So every round-trip time, $\alpha$ changes by $g(F-\alpha)$. Consequently, we have:

$$\frac{d\alpha_i}{dt} = \frac{g}{R(t)}(p(t-R^*) - \alpha_i(t)). \quad (10)$$

Finally, the dynamics of queue length $q(t)$ is relatively simple:

$$\frac{dq}{dt} = \frac{\sum_{i=1}^{N} W_i(t)}{R(t)} - C. \quad (11)$$

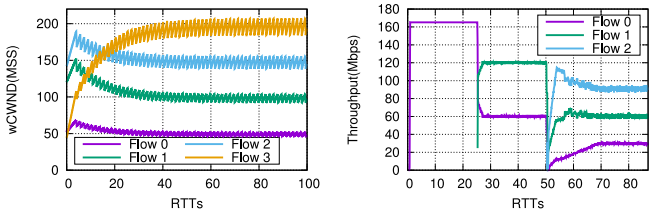
Eq. (11) models the queue evolvement: $\sum_{i=1}^{N} W_i(t)$ is the net input rate and $C$ is the link capacity.

To complete the formation, another equation $R(t) = d + q(t)/C$ is needed, denoting the round-trip time (RTT) where $d$ is the propagation delay for all the flows and $q(t)/C$ represents the queueing delay.

Now we have the complete fluid model:

$$\begin{cases} \frac{dW_i}{dt} = \frac{w_i}{R(t)} - \frac{W_i(t)\alpha_i(t)}{2R(t)}p(t-R^*), \\ \frac{d\alpha_i}{dt} = \frac{g}{R(t)}(p(t-R^*) - \alpha_i(t)), \\ \frac{dq}{dt} = \frac{\sum_{i=1}^{N} W_i(t)}{R(t)} - C, \\ R(t) = d + q(t)/C. \end{cases} \quad (12)$$

[1] Window cut happens at most once per window of data.




(a) Results of the fluid model (b) Flows arriving at intervals in WVCC

**Fig. 5.** Evaluation results of WVCC.

**Steady state analysis:** For simplicity of analysis in steady state, we change Eq. (8) into:

$$\frac{dW_i}{dt} = \frac{w_i}{R^*} - \alpha\frac{W_i(t)}{2R^*}, \quad (13)$$

where $R^*$ and $\alpha$ are constant. This simplification is reasonable in that at steady state, round-trip time and $\alpha$ are comparatively stable. At steady state, the additive increasement and multiplicative decreasement of the window size will be roughly equal in one RTT. So we have:

$$\int_0^{R^*} \frac{dW_i}{dt}dt = w_i - \alpha\int_0^{R^*}\frac{W_i(t)}{2R^*}dt = 0. \quad (14)$$

Assume the average window size at steady state is $W_i^*$, then we have:

$$\int_0^{R^*} W_i(t)dt = K * W_i^*, \quad (15)$$

where K is a constant. Combine this with Eq. (14), we conclude that:

$$W_i^* \propto w_i, \quad (16)$$

which means average windows size of each flow is proportional to its weight at steady state in WVCC. The difference between this model and the original DCTCP fluid model is that the additive increase coefficient $w_i$ takes the place of a unit additive increment. A flow with weight $w_i$ is equivalent to accumulating $w_i$ parallel identical unit flows. As a result, the throughput of a weighted flow, as well as its change in terms of throughput, is proportional to its weight.

Similarly, for AC/DC we have simplified equation derived from Eq. (6):

$$\frac{dW_i}{dt} = \frac{1}{R^*} - \frac{W_i(t)}{2R^*}\alpha\left(1-\frac{\beta}{2}\right), \quad (17)$$

from which we cannot derive a proportional relation between $W_i^*$ and $\beta$.

**Emulation:** According to (12), we develop the model with MATLAB. We translate the equations of (12) into forms of Delay Differential Equations and solve them with dde23 in MATLAB. The MATLAB emulation code is publicly available at [46]. We setup four flows with random initial window and each weights [1, 2, 3, 4]/5.0 respectively, competing on a single bottleneck. As Fig. 5(a) illustrates, these four flows get their throughput share proportionally in steady state.

## 5. Evaluation

This section evaluates the effectiveness of Uranus and determines if Uranus outperforms the existing solution. We first evaluate the weight enforcement component of Uranus, WVCC, to judge whether it can achieve flow-level proportional sharing in

congested links. Then we evaluate the performance of Uranus from three aspects: network proportionality, objective-oriented optimization and network utilization.

### 5.1. Evaluation of weight enforcement: WVCC

#### 5.1.1. NS3 simulation

We implement WVCC in NS3 and demonstrate the flow-level proportional bandwidth allocation capability. We test WVCC with a dumbbell topology in a dynamic scenario, where we setup three flows which start one after another, sending packets as fast as they can in different hosts with weights [1, 2, 3]/4.0, competing on the same bottleneck link. Fig. 5(b) shows that each time a new flow arrives, all flows converge to a new steady state with bandwidth share proportional to their weights. This simulation result accords with the fluid model analysis and also demonstrates that WVCC is work-conserving and can achieve proportional sharing in congested links.

#### 5.1.2. Linux Kernel implementation

In this section, we discuss the implementation details of WVCC as a loadable kernel module in Linux Kernel. Its performance is evaluated by reproducing similar scenario as shown previously. WVCC is a light shim-layer that sits between the TCP/IP stack (or guest VMs) and the link-layer (or Hypervisor). The module is built using the popular NetFilter [29] framework which is an integral part of Linux kernel. Netfilter uses hooks that attach to the data path in the Linux kernel just above the NIC driver. This system design dose not modify the TCP/IP stack of both the host and guest OSes. Since the implementation is realized as a loadable kernel module, it is easy to deploy in current data centers. The module intercepts all incoming TCP packets destined to the host or its guests right before it is pushed up to TCP/IP stack handling (i.e., at the post-routing hook). First, the 4 identifying tuples (e.g., source and destination IP address and Port numbers) are hashed and the associated flow is indexed into the Hash Table. The hash is calculated via Jenkins hash (JHash) [22]. Then, TCP packet headers are examined so that the flag bits are used to choose the right course of action (i.e., SYN-ACK, FIN or ACK). The module does not employ any packet queues to store the incoming packets. It only stores and updates flow entry states (i.e., ECN marking counts, arrival time and so on) on arrival of segments. WVCC is light-weight module and does not require fine-grained state collection and handling. It uses 4-tuples for hashing, uses kernel built-in hash functions [22] and performs minimal modifications on the TCP header. We collect various system load statistics during the various experiments with WVCC and they prove that there is no noticeable increase (1–3%) in the load due to the added processing of WVCC. For SCC, we replicate the experiments and the additional overhead increase is around 10–16%. Moreover, virtualized data centers run virtual switches on the end-hosts to manage tenants' virtual networks [44]. These operations are readily performed by the virtual switches as they maintain a hash-based flow tables and hence introduce little overhead to CPUs.

**Experimental setup:**

To put WVCC to the test, we use a small-scale testbed consisting of 4 servers interconnected via 1 server running a software switch (or OpenvSwitch [30]). As shown in Fig. 8, the testbed is organized into 3 senders and 1 receiver. Each server is connected to the OvS switch via 10 Gbps link which can run at speed of 1 Gbps or 10 Gbps. The average RTT in the network is $\approx$1 ms. The servers are loaded with Ubuntu Server 16.04 LTS with kernel version (4.4). The WVCC end-host module is installed and loaded on demand in the host OS whenever necessary. The OvS ports are configured using Linux Traffic Control to use RED AQM with marking threshold matching the DCTCP settings. We install the iperf program [20] to emulate long-lived background traffic (e.g., VM migrations, backups) in the experiments. The iperf traffic runs for long periods and in the meanwhile, the instantaneous, moving average and overall average throughput are reported. We use weights of (1/4, 2/4, 3/4) which translates to (1/6, 1/3, 1/2) of the bottleneck capacity.

**Results and discussion:**

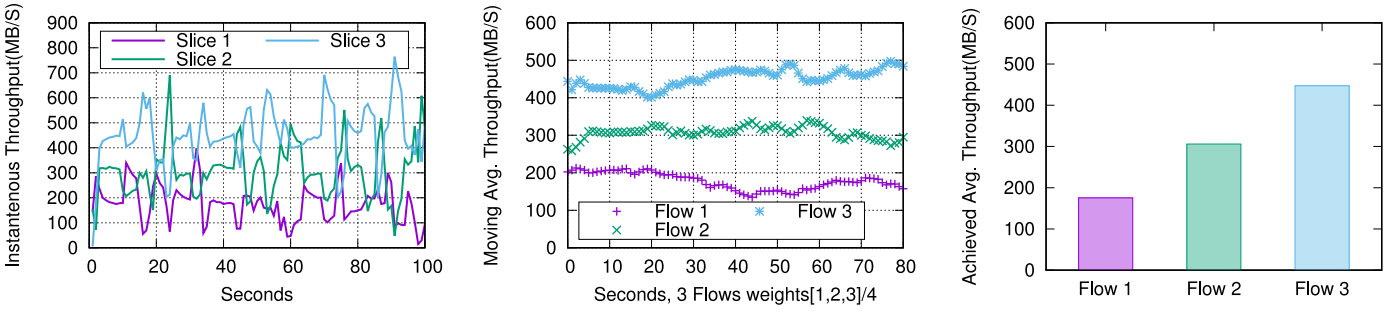
We run WVCC in two scenarios. In the first one, all flows start at the same time in the beginning (i.e., $t = 0$) and the experiments last for 100 s. In the second scenario, we evaluate the convergence speed of WVCC. Hence, we set the experiment to let two flows (1 and 2) start at $t = 0$ and last for 100 s, while the third flow starts at $t = 50$ and lasts until the end of the experiment at 150 s. Fig. 6(a)–(c) show the instantenous, moving average and overall average for the first scenario respectively. Fig. 7(a)–(c) show the same metrics for the second scenario. The results in both scenarios show that WVCC is able to achieve the goal of congestion proportionality among various flows according to their allocated weights. Specifically, Fig. 7(a) shows that as soon as two flows (1 and 2) finish, flow 3 consumes all available bandwidth quickly, which demonstrates WVCC is work-conserving. Moreover, the fast convergence speed of WVCC allows fast reaction to events such as flow arrivals/departures as well as any weight adjustments by the operator.

The convergence time of WVCC is similar to that of DCTCP, which is in the order of 20–30 ms at 1Gbps, and 80–150 ms at 10 Gbps [2]. In data centers, for nearly 70% of the ToR pairs, the traffic remains stable for several seconds on average [8], which is enough for our enforcement to take effect.

### 5.2. Evaluation of Uranus

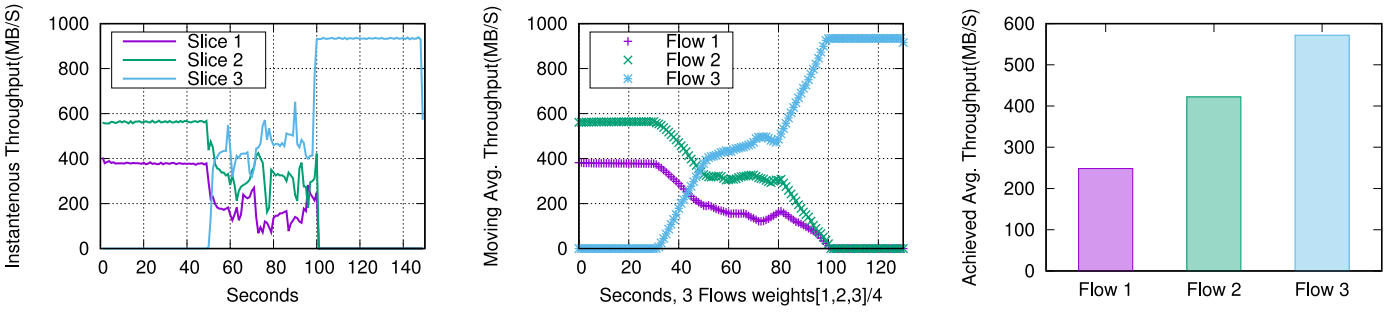
In this section we evaluate the performance of Uranus from several aspects. Specifically, the evaluation intends to answer the following questions:

- **Can Uranus achieve better network proportionality?** Network proportionality is Uranus's fundamental capability, with which Uranus will be able to isolate different slices in the same network and further to apply intra-slice flow scheduling. In order to evaluate Uranus in terms of network proportionality, we test Uranus under different proportion values and use Jain's fairness Index as a metric to quantify the proportional fairness. It is observed that Uranus improves proportional fairness by around 21.25%–31.49%.
- **How does the intra-slice scheduling of Uranus work?** Network proportionality enables Uranus to correctly allocate resources to different slices among congested links, while intra-slice scheduling enables Uranus to optimize in-slice objectives. We evaluate Uranus with scenarios where flows have different objectives. We first evaluate Uranus in isolation scenario, where all slices have the same objective. In isolation scenario, Uranus reduces the deadline-miss ratio by around 25% on average and reduces AFCT by 20% under high load pressure. Then we evaluate Uranus against the scenario where slices with different objectives coexist, in which both inter-sclice resource allocation and intra-slice optimization are both evaluated. With flows of different objectives coexist, Uranus also shows the ability to improve performance.
- **Can Uranus achieve better network utilization?** Section 2.4 reveals that tunnel-based weight enforcement method introduces decreased network utilization due to biased congestion signal. We evaluate Uranus from the perspective of network utilization against tunnel-based network sharing solution. We measure the aggregated bandwidth occupation, which illustrates that Uranus gets an additional 10% improvement in network utilization than tunnel-based scheduling mechanisms.




(a) The instantenous throughput (b) The moving average throughput (c) The average throughput

**Fig. 6.** Performance metrics of WVCC in the first experimental scenario.




(a) The instantenous throughput (b) The moving average throughput (c) The average throughput

**Fig. 7.** Performance metrics of WVCC in the second experimental scenario.

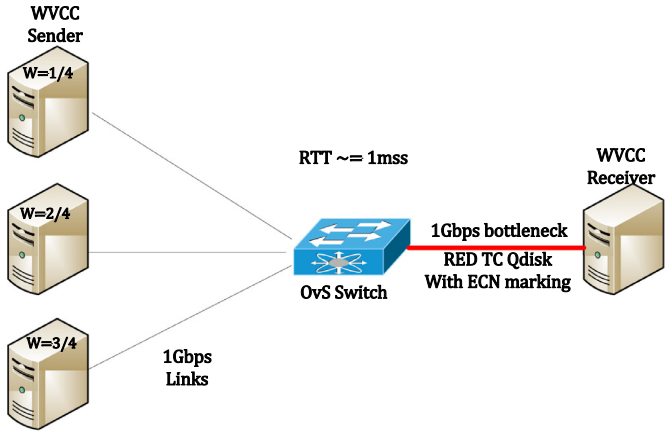



**Fig. 8.** A simple dumbbell topology testbed for WVCC evaluation.

**Experiment setup:** We use a three-layer Fat-Tree topology for NS3 simulations. The same topology is used in the simulations unless specified otherwise. The experiment adopts a 4-ary Fat-Tree similar with Fig. 1, with 4 core switches, 8 aggregator switches, 8 ToR switches, each ToR switch connected to 8 nodes. The capacity of edge links from server to ToR switch, from ToR switch to Agg switch is 1 Gbps, and core links is 10 Gbps. Switches in the network are ECN enabled with ECN marking threshold set to 65 packets. The maximum end-to-end RTT is 300 usec. The network is equipped with flowcell load balancing module for this simulation. We re-implement SCC and its tunnel-based scheduling algorithms described in [41] in NS3 with the same topology. SCC adopts virtual-tunnel in the evaluation experiments.

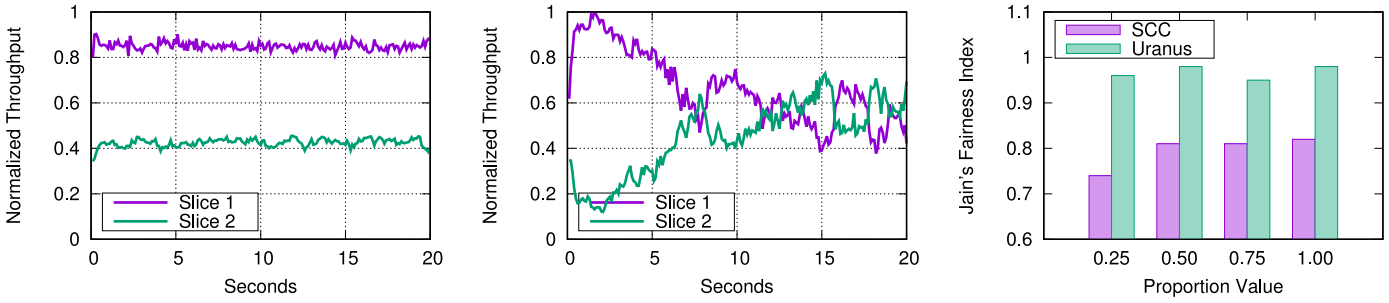
In our evaluation, we consider deadline-sensitive and completion-sensitive flows. Deadline-sensitive traffic uses D$^2$TCP [42] and completion-sensitive traffic uses L$^2$DCT [27] as legacy transmission protocols with defalut parameters from respective papers at the hosts. We replicate WebSearch [1], Data-mining [15] and MapReduce [10] workloads by generating traffic with corresponding flow-size distribution. Regarding deadline flows, the deadlines are exponentially distributed using guidelines from [27].

*5.2.1. Comparison of network proportionality*

In this section, we evaluate the performance of Uranus from the perspective of network proportionality. For comparison, we setup two slices in the network. Data center applications tend to adapt randomness to improve their performance[15]. For instance, distributed file system spreads data chunks randomly across servers for better load distribution and redundancy. Randomness used in data-center applications results in poor summarizability and unpredictability of traffic patterns[15]. So the traffics we setup in each slice are uniformly distributed across racks.

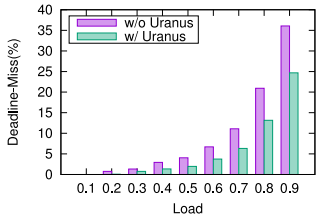
Before we generate traffic for these two slices, we generate background traffic randomly among racks, consuming around 10% bandwidth resource. Background traffic together with slice traffic make the whole network a little overloaded. Specifically, at each host sending packets, the client works as a *bulk-sender*, i.e, sending packets as fast as possible. We vary the weight among these two slices from 1 : 4 to 1 : 1, with proportion values of [0.25, 0.50, 0.75, 1.00] respectively. We compare the network proportionality achieved by Uranus and SCC for each proportion value. This work applies Jain's fairness index [21] as a metric to quantify network proportionality.

For clearness, we present the normalized throughput curve of Uranus and SCC under weight 1 : 2 in Fig. 9(a) and (b) respectively. As Uranus follows weight allocation algorithm of PS-N, it closely approximates congestion-proportionality. Fig. 9(a) shows that Uranus achieves network-wide proportionality almost ideally. In terms of SCC, as it allocates weight among tunnel pairs without considering network proportionality, it gradually loses network proportionality as flow scheduling happens. Fig. 9(c) com-

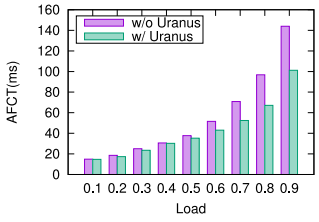



(a) Proportional Sharing of Uranus (b) Proportional Sharing of SCC (c) Jain's Fairness Index of different weight

**Fig. 9.** Comparison of network proportionality.

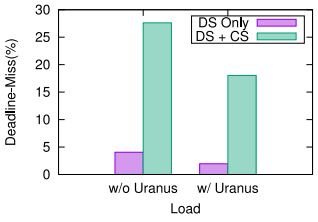



(a) Deadline Miss Ratio of DS flows

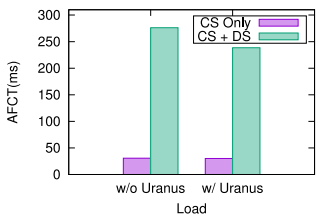



(b) AFCT of CS flows

**Fig. 10.** Flow scheduling in isolation scenario.




(a) Deadline Miss Ratio of DS flows




(b) AFCT of CS flows

**Fig. 11.** Interference of coexistence.

pares Jain's fairness index between Uranus and SCC under different proportion values. It is observed that Uranus always achieves much better network proportionality than SCC. The improvement is around 21.25%–31.49%.

*5.2.2. Effectiveness of intra-slice scheduling*

In this section, we evaluate the effectiveness of Uranus's intra-slice scheduling algorithms for flows with deadline-sensitive and completion-sensitive objectives. We use the fraction of missed deadlines, and average FCT (AFCT) as the performance metrics for evaluating the DS and CS flows respectively.

**Isolation scenario:**

We first evaluate the effectiveness of Uranus's intra-slice scheduling algorithms under the condition where no coexistence of flows with different objectives happens. Specifically, during each experiment, only one slice with a specific objective exists in the network. We compare Uranus against legacy transmission protocols, i.e. $D^2$TCP for DS flows while $L^2$DCT for CS flows, with default parameters set according to respective papers.

In the experiment, we use WebSearch workload and tune the network load from low to high $(0.1 - 0.9)$. The senders generate traffic according to a Poisson process, with workload CDF as input and $\lambda$ calculated according to $\lambda = \frac{linkrate * linkload}{mean\,flowsize}$ so that the average bandwidth requirement of the generated traffic accords with network load. For each network load, we generate traffic with the same parameters in the network for scenarios with and without Uranus scheduling.

*For deadline-sensitive traffics*, deadlines are exponentially distributed using guidelines from [27] and assigned to each flow. Deadline-miss ratio is collected at the receiver side. The experiment for each network load is replicated 3 times and average deadline-miss ratio is calculated. Fig. 10(a) shows that Uranus reduces the deadline-miss ratio by around 25% on average under different load pressure.

*For completion-sensitive traffics,* The completion-time of each flow is collected at the receiver side. Similarly, the experiment for each network load is replicated 3 times and average flow completion-time is calculated. Uranus reduces AFCT by 20% at high load pressure as shown in Fig. 10(b). When the network is less congested, i.e., under lower load pressure, Uranus is only slightly better than $L^2$DCT.

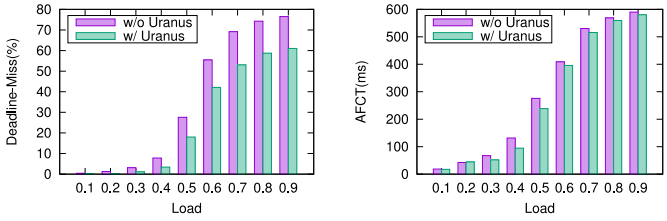
**Coexistence scenario:**

Then we examine whether intra-slice scheduling of Uranus works when flows of different objectives coexist in the network. In this experiment, we setup two slices, one is deadline-sensitive while the other is compeletion-sensitive. We also use WebSearch workload as the traffic pattern as in previous section and traffic is generated according to Poisson process. The parameters for the Poisson process are calculated according to given network capacity and network load. Specifically, the two slices share the same Poisson parameters so that the overall bandwidth requirement of the two slices are the same and they are expected to share the network equally. For deadline-sensitive flows, a deadline is assigned to each flow and deadlines are exponentially distributed using guidelines from [27].

We first consider the interference caused by coexistence of traffics with different objectives under medium network load. We start the deadline-sensitive slice first and collect the deadline-miss ratio as benchmark. Then we start the completion-sensitive slice and collect the deadline-miss ratio of deadline-sensitive slice under scenarios with and without Uranus. The results are illustrated in Fig. 11(a). Similarly, We start the completion-sensitive slice first and collect the average flow-comption time as benchmark. Then we start the deadline-sensitive slice and collect the average flow-completion time of completion-sensitive slice under scenarios with and without Uranus. The results are illustrated in Fig. 11(b).

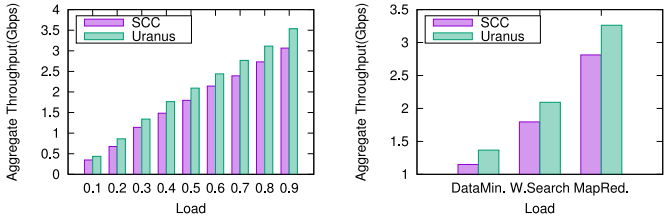
As illustrated in Fig. 11(a) and (b), coexistence of flows with different objectives does harm the overall performance. The deadline-miss ratio of DS traffic increases over 4X while AFCT of CS flows increases 5X on average. With Uranus, the deadline-missing ratio of DS is reduced by 30% and AFCT of CS flows reduces about 10% under medium load pressure.

Then we replicate the experiment with different network load. It is observed that intra-slice scheduling algorithm of Uranus functions well under different load pressure in coexistence scenarios. As shown in Fig. 12(a), Uranus reduces the deadline-miss ratio by




(a) Deadline Miss Ratio of DS flows (b) AFCT of CS flows

**Fig. 12.** Flow scheduling in coexistence scenario.




(a) Under different pressure (b) With different traffic pattern

**Fig. 13.** Comparison of network utilization.

about 18% on average. However, in terms of AFCT, though Uranus works under different network load, the effectiveness becomes less significant when the load gets high. We leave the improvement of scheduling algorithms as a future work. As the workloads change to Data-mining and MapReduce, we get similar results. We omit the results due to space limitation.

Traffics with different objectives are safe to coexist under conditions in which bottleneck bandwidth can be isolated according to objectives of the traffic. Uranus is equipped with the ability to enforce flow-level proportional bandwidth sharing, which can be used to apply performance isolation and thus reduce interference among traffics with different objectives. The results in this section demonstrate that Uranus can reduce the impact caused by interference and improve performce in situations where flows with different objective coexist.

### 5.2.3. Comparison of network utilization

In this experiment, we illustrate the benefits brought by the flow-level scheduling mechanism from the perspective of network utilization. We replicate WebSearch [1], Data-mining [15] and MapReduce [10] workloads. We set WebSearch traffics as deadline-sensitive and the other two as completion-sensitive. Aggregated throughput is used as the metric to quantify the network utilization.

We first consider the scenario in which only WebSearch traffics exist in the network. Traffics are generated with different load pressure. Under each network load, we run SCC and Uranus as the scheduler separately and collect aggregated throughput respectively. From Fig. 13(a) we can see that Uranus gets higher aggregated throughput than SCC under any load pressure. We further test all three traffic patterns under medium load pressure to compare the network utilization of Uranus against that of SCC. As shown in Fig. 13(b), the result further demonstrates that Uranus can achieve better network utilization than SCC irrespective of the traffic patterns. During this experiment, it is observed that packet-drop rate of SCC is higher than that of Uranus. Biased congestion signal of tunnel causes delayed response to congestions, which results in higher packet-drop rate and thus reduces the network utilization. In contrast, as Uranus applies flow-level weighted congestion control, it is capable of achieving finer-grained congestion signal processing and responding. In conclusion, Uranus improves overall network utilization by around 10%.

## 6. Conclusion

In this paper, we develop Uranus, a slice scheduling framework that can approximate congestion-proportionality in data center networks. With existing load balancing techniques, we can treat the core switch level of state-of-the-art Clos-based data center network as a non-blocking big switch. We use the *Proportional Sharing at Network-level* scheme in the rack level bandwidth weight allocation. We also develop Weighted Virtual Congestion Control (WVCC) mechanism to transparently enforce weight among flows. Extensive simulations show that Uranus closely approximates the congestion-proportionality and improves weighted fairness. Compared with state-of-the-art tunnel-based solution, Uranus also improves network utilization.

## Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. This work is partially supported by the National Key Research and Development Program of China under Grant number 2016YFB1000102, the National Natural Science Foundation of China under Grant numbers 61772265, 61672318, 61631013.

## References

[1] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), in: Proceedings of the ACM SIGCOMM 2010 Conference, in: SIGCOMM '10, ACM, New York, NY, USA, 2010, pp. 63–74, doi:10.1145/1851182.1851192.
[2] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (dctcp), in: in Proc. ACM SIGCOMM 2011, 2011.
[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pfabric: minimal near-optimal datacenter transport, in: ACM SIGCOMM Computer Communication Review, vol. 43, ACM, 2013, pp. 435–446.
[4] N. Alliance, 5G White Paper, Next generation mobile networks, 2015. white paper.
[5] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, H. Wang, Information-agnostic flow scheduling for commodity data centers, in: NSDI, USENIX, 2015.
[6] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, Towards predictable datacenter networks, in: ACM SIGCOMM, ACM, 2011, pp. 242–253.
[7] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: ACM IMC, ACM, 2010, pp. 267–280.
[8] T. Benson, A. Anand, A. Akella, M. Zhang, Microte: fine grained traffic engineering for data centers, in: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies, ACM, 2011, p. 8.
[9] L. Chen, K. Chen, W. Bai, M. Alizadeh, Scheduling mix-flows in commodity datacenters with Karuna, in: Proceedings of the 2016 ACM SIGCOMM Conference, ACM, 2016, pp. 174–187.
[10] Y. Chen, A. Ganapathi, R. Griffith, R. Katz, The case for evaluating mapreduce performance using workload suites, in: IEEE MASCOTS'11.
[11] C. Clos, A study of non-blocking switching networks, Bell Labs Tech. J. 32 (2) (1953) 406–424.
[12] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, I. Keslassy, Virtualized congestion control, in: Proceedings of the 2016 ACM SIGCOMM Conference, in: SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 230–243, doi:10.1145/2934872.2934889.
[13] D.E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, J.D. Hosein, Maglev: a fast and reliable software network load balancer., in: in: NSDI, 2016, pp. 523–535.
[14] P.X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, S. Shenker, pHost: distributed near-optimal datacenter transport over commodity network fabric, in: Proceedings of the CoNEXT, 2015.
[15] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Vl2: a scalable and flexible data center network, in: ACM SIGCOMM Computer Communication Review, vol. 39, ACM, 2009, pp. 51–62.
[16] C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, Secondnet: a data center network virtualization architecture with bandwidth guarantees, in: ACM CoNext 2010.
[17] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akella, Presto: edge-based load balancing for fast datacenter networks, ACM SIGCOMM Computer Communication Review, 2015.
[18] K. He, E. Rozner, K. Agarwal, Y.J. Gu, W. Felter, J. Carter, A. Akella, AC/DC TCP: virtual congestion control enforcement for datacenter networks, in: Proceedings of the 2016 ACM SIGCOMM Conference, in: SIGCOMM '16, ACM, New York, NY, USA, 2016, pp. 244–257, doi:10.1145/2934872.2934903.
[19] C.-Y. Hong, M. Caesar, P. Godfrey, Finishing flows quickly with preemptive scheduling, in: Proc. ACM SIGCOMM 2012.

[20] iperf, The TCP/UDP bandwidth measurement tool https://iperf.fr/.
[21] R. Jain, A. Durresi, G. Babic, Throughput Fairness Index: An Explanation, Technical Report, Tech. rep., Department of CIS, The Ohio State University, 1999.
[22] B. Jenkins, A hash function for hash table lookup, http://burtleburtle.net/bob/hash/doobs.html.
[23] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, A. Greenberg, Eyeq: practical network performance isolation at the edge, in: NSDI, USENIX, 2013.
[24] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, P. Sharma, Application-driven bandwidth guarantees in datacenters, in: ACM SIGCOMM, ACM, 2014, pp. 467–478.
[25] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al., Timely: Rtt-based congestion control for the datacenter, in: Proc. ACM SIGDC 2015.
[26] A. Munir, G. Baig, S.M. Irteza, I.A. Qazi, A.X. Liu, F.R. Dogar, Friends, not foes: synthesizing existing transport strategies for data center networks, in: ACM SIGCOMM 2014.
[27] A. Munir, I.A. Qazi, Z.A. Uzmi, A. Mushtaq, S.N. Ismail, M.S. Iqbal, B. Khan, Minimizing flow completion times in data centers, in: in Proc. IEEE INFOCOM, 2013.
[28] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, S. Katti, Numfabric: fast and flexible bandwidth allocation in datacenters, in: ACM SIGCOMM 2016.
[29] NetFilter.org, Netfilter packet filtering framework for linux, http://www.netfilter.org/.
[30] OpenvSwitch.org, Open virtual switch project, 2019 http://openvswitch.org/.
[31] S. Oueslati, J. Roberts, N. Sbihi, Flow-aware traffic control for a content-centric network, in: INFOCOM, 2012 Proceedings IEEE, IEEE, 2012, pp. 2417–2425.
[32] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, L. Gu, Hadoopwatch: a first step towards comprehensive traffic forecasting in cloud computing, in: INFOCOM, 2014 Proceedings IEEE, IEEE, 2014, pp. 19–27.
[33] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, H. Fugal, Fastpass: a centralized zero-queue datacenter network, in: ACM SIGCOMM.
[34] L. Popa, A. Krishnamurthy, S. Ratnasamy, I. Stoica, Faircloud: sharing the network in cloud computing, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, in: HotNets-X, ACM, New York, NY, USA, 2011, pp. 22:1–22:6, doi:10.1145/2070562.2070584.
[35] L. Popa, P. Yalagandula, S. Banerjee, J.C. Mogul, Y. Turner, J.R. Santos, Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing, in: ACM SIGCOMM 2013.
[36] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, A. Vahdat, Senic: scalable NIC for end-host rate limiting, in: NSDI, 14, 2014, pp. 475–488.
[37] H. Rodrigues, J.R. Santos, Y. Turner, P. Soares, D. Guedes, Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks, WIOV, 2011.
[38] A. Shieh, S. Kandula, A. Greenberg, C. Kim, Seawall: Performance isolation for cloud datacenter networks, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, in: HotCloud'10, USENIX Association, Berkeley, CA, USA, 2010. 1–1.
[39] A. Shieh, S. Kandula, A.G. Greenberg, C. Kim, B. Saha, Sharing the data center network, in: NSDI, 2011.
[40] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, et al., Jupiter rising: a decade of clos topologies and centralized control in Google's datacenter network, in: Proc. ACM SIGDC 2015.
[41] C. Tian, A. Munir, A.X. Liu, Y. Liu, Y. Li, J. Sun, F. Zhang, G. Zhang, Multi-tenant multi-objective bandwidth allocation in datacenters using stacked congestion control, in: INFOCOM, 2017 Proceedings IEEE, IEEE, 2017, pp. 3074–3082.
[42] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-aware datacenter tcp (d2tcp), in: ACM SIGCOMM, 2012, pp. 115–126.
[43] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, T. Edsall, Let it flow: Resilient asymmetric load balancing with flowlet switching., in: in: NSDI, 2017, pp. 407–420.
[44] A. Weeks, How the vswitch impacts the network in a virtualized data center, https://searchdatacenter.techtarget.com/tip/How-the-vSwitch-impacts-the-network-in-a-virtualized-data-center.
[45] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better never than late: meeting deadlines in datacenter networks, in: ACM SIGCOMM, 2011, pp. 50–61.
[46] wvcc, Matlab code for fluid model in wvcc, https://github.com/jiaqing-phd/wvcc.git.
[47] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, Detail: reducing the flow completion time tail in datacenter networks, in: ACM SIGCOMM, 42, 2012, pp. 139–150.
[48] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, M. Zhang, Guaranteeing deadlines for inter-datacenter transfers, in: ACM Eurosys, ACM, 2015, p. 20.

**Jiaqing Dong**, Ph.D candidate in Department of Computer Science, Tsinghua University, Beijing, China. Jiaqing Dong received the B.S. degree in computer science from Peking University, China in 2013. He is currently pursuing the Ph.D. degree in Department of Computer Science at Tsinghua University. His research interests include data center networks and distributed systems.

**Hao Yin**, Professor in the Research Institute of Information Technology (RIIT) at Tsinghua University. Hao Yin received the B.S., M.E., and Ph.D. degrees from Huazhong University of Science and Technology, China. He was elected as the New Century Excellent Talent of the Chinese Ministry of Education in 2009, and won the Chinese National Science Foundation for Excellent Young Scholars in 2012. His research interests span broad aspects of Multimedia Communication and Computer Networks.

**Chen Tian**, Associate Professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. Chen Tian received the B.S., M.S., and Ph.D. degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China. He was an Associate Professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology. From 2012 to 2013, he was a Postdoctoral Researcher with the Department of Computer Science, Yale University. He is currently an Associate Professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming, and urban computing.

**Ahmed M. Abdelmoniem**, Tenured Assistant Professor, Faculty of Computers and Information, Assiut University, Egypt. Ahmed M. Abdelmoniem received the B.S. and M.S. degrees in Computer Science Department, Faculty of Computers and Information, Assiut University, Egypt, in 2007 and 2012 respectively, and the Ph.D. degree in Computer Science and Engineering, Hong Kong University of Science and Engineering, Hong Kong, 2017. He worked as a Senior Researcher at Future Network Theory Lab, Huawei Technologies Co., Ltd., Hong Kong from 2017 to 2018. His research interests covers a range of topics from cloud/data center networks and systems, wireless networks, congestion control and traffic engineering.

**Huaping Zhou** is currently a second-year master student at the Department of Computer Science and Technology, Nanjing University, China. He received the B.S. degree from the School of Computer Science and Engineering, Beihang University, China. His research interests include data center networks and distributed systems.

**Bo Bai**, Senior Researcher at Future Network Theory Lab, Huawei Technologies Co., Ltd., Hong Kong. Bo Bai received the B.S. degree in School of Communication Engineering from Xidian University, Xi'an China, 2004, and the Ph.D. degree in Department of Electronic Engineering from Tsinghua University, Beijing China, 2010. He was a Research Assistant from April 2009 to September 2010 and a Research Associate from October 2010 to April 2012 with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. From July 2012 to January 2017, he was an Assistant Professor with the Department of Electronic Engineering, Tsinghua University. Currently, he is a Senior Researcher at Future Network Theory Lab, Huawei Technologies Co., Ltd., Hong Kong. He is leading a team to develop fundamental principles, algorithms, and systems for graph learning, cell-free mobile networking, bio-inspired networking, and quantum Internet.

**Gong Zhang**, Chief Architect Researcher Scientist, director of the Future Network Theory Lab. Gong Zhang's major research directions are network architecture and large-scale distributed systems. He has abundant R&D experience on system architect in networks, distributed system and communication system for more than 20 years. He has more than 90 global patents in which some play significant roles in the company. In 2000, he acted as a system engineer for L3+ switch product and became the PDT (Product development team) leader for smart device development, pioneering a new consumer business for the company since 2002. Since 2005, he was a senior researcher, leading future internet research and cooperative communication. In 2009, he was in charge of the advance network technology research department, leading researches of future network, distributed computing, Database system and data analysis. In 2012, he became the Principal Researcher and led the system group in data mining and machine learning.