PACE Your Network: Fair and Controllable Multi-Tenant Data Center Networks

Tiago Carvalho
Carnegie Mellon University and
Universidade de Lisboa

Hyong S. Kim
Carnegie Mellon University
Pittsburgh, PA, USA

Nuno Neves
Universidade de Lisboa
Lisbon, Portugal

Abstract—**Multi-tenant data centers host a high diversity of applications with continuously changing demands. Applications require response times ranging from a few microseconds to seconds. Therefore, network traffic within the data center needs to be managed in order to meet the requested SLAs. Current feedback congestion control protocols may be too slow to converge to a stable state under high congestion situations. Sudden bursts of traffic from heterogeneous sources may render any reactive control inefficient. In this paper, we propose PACE, a preventive explicit allocation congestion control protocol that controls resource allocations dynamically and efficiently. PACE specifically addresses Data Center requirements: efficient network usage, flow completion time guarantees, fairness in resource allocation, and scalability to hundreds of concurrent flows. PACE provides micro-allocation of network resources within dynamic periods, lossless communication, fine-grained prioritization of flows, and fast adaptation of allocations to the arrival of new flows. We simulate PACE and compare it with recent proposed protocols specially addressed to Data Centers. We demonstrate that PACE is fairer, in particular for short flows, flows with different RTTs and a higher number of concurrent flows. It also maintains high efficiency and controlled queue usage when exposed to sudden bursts.**

Keywords-Data Center; network; protocols; congestion control

I. Introduction

Commercial Data Centers (DC) host various applications like storage, financial services, social networking, e-mail, web hosting, cloud computing, and content delivery servers. These applications require very different Service Level Agreements (SLA). Some applications, like online searching require very low *response times*. Some other applications may require constant *bandwidth* during a variable amount of time. Video streaming applications have strict requirements in terms of bandwidth and jitter. On the other hand, *fairness* is important to Infrastructure-as-a-Service (IaaS) DCs like Windows Azure [1], Amazon EC2 [2] and Google Cloud [3]. Typically, these services are oversubscribed and require resources to be shared fairly among tenants.

Thus, a Congestion Control Protocol designed for DC networks should guarantee: ***efficiency***, the protocol should guarantee that links are fully utilized whenever possible; ***fairness***, all flows should be able to get access to an equal amount of resources, or proportional to their importance; ***controllability***, the protocol should provide an interface allowing prioritization and differentiation of flows; ***scalability***, the protocol should scale to a high number of flows without compromising the remaining properties. The protocol should achieve these properties independently of the traffic patterns or flow characteristics (e.g. flow size, RTT, etc.). Thus, it is very important to guarantee fast convergence to a stable, fair and efficient state.

Currently, no protocols are able to achieve all these goals for a wide range of traffic patterns. For instance, feedback-based protocols like TCP, RCP [4] and DCTCP [5] may be slow to converge to the fair state and very sensitive to different round-trip times (RTT). This leads to unfair treatment of short flows or to situations where fair stable state is not achieved. On the other hand, recently proposed D^3 [6] and PDQ [7] focus on specific traffic patterns and assume full knowledge of flow characteristics.

In this paper, we propose *PACE*, a preventive explicit allocation congestion control protocol. PACE provides micro-allocation of network resources within dynamic periods, lossless communication, fair allocations according to flows importance, fast adaptation to the arrival of new flows, and scalability to a large number of competing flows. PACE also provides administrators with the ability to control how resources in the network are allocated. PACE's main contributions are:

- Strictly lossless communication that combines credit and rate allocation to avoid congestion in the network core, and guarantees fast convergence to a fair state.
- PACE provides periodic reallocation and expiration of credits. This feature allows PACE to quickly adapt to transient and diverse traffic patterns.
- Dynamic periods of credit allocation that allow PACE to scale to a large number of competing, without compromising its fairness and efficiency guarantees.

II. PACE's Design Rationale

Congestion control protocols can be differentiated according to when they actuate, how they compute desirable transmission rates and how the transmission rate is mapped to

This work was partially supported by the Multiannual (LASIGE) and by FCT through the CMU-Portugal program.

the source. These design decisions influence how effectively they can meet the requirements of DC networks. We describe PACE's design rationale from these three perspectives. First, PACE is a *preventive control* protocol. It avoids congestion by controlling queue size and packet transmission from the source. Most protocols are based on *reactive control* and they only actuate when congestion is detected. TCP only reacts when packets are lost, while DCTCP [5] uses ECN [8] to signal that queue usage is above a certain threshold. Protocols like RCP [4] and D^3 [6] also adjust their allocations based on queue usage and link utilization. Preventive protocols have faster convergence to a stable state than reactive protocols. Second, PACE computes an *absolute* value for the transmission rate. It explicitly states the transmission rate for a source. Typically, setting an absolute rate allows more accurate variation of transmission rates and thus faster adjustment to new traffic scenarios. Third, congestion control protocols can also be classified into *credit-based* or *rate-based* protocols. Credit-based schemes explicit authorize sources to transmit a specific amount of data. On the other hand rate-based schemes authorize sources to transmit at a given speed. This authorization is valid until a new rate allocation is received. Credit-based schemes can guarantee lossless communication while rate-based schemes spread packets leading to lower queue peaks. PACE takes advantage of both credit and rate-based approaches by implementing a hybrid scheme. It authorizes endpoints to transmit a specified amount of data within a specified period.

A. *Key Design Ideas*

PACE's key idea is that one endpoint of a connection issues credits periodically. These credits allow the peer endpoint to send a proportional amount of data within a specified period of time. Credits have been used in prior works [9]. TCP's window-based solution is also implicitly based on credits. However, PACE uses credits in a very different way. First, PACE allocates credits periodically. This implies that credits are only valid until new credits are issued. As opposed to the credit-based solution proposed for ATM [9], PACE does not require per-flow queues. Second, PACE performs end-to-end credit allocation instead of hop-by-hop. Nodes along the path can control the amount of credits. Thus, PACE's allocation mechanism has two control variables: the *number of credits* and the *period of allocation*. Both of these variables are transmitted in control messages. Fig. 1 demonstrates how nodes can use these two variables to compute an allocation. Initially, the allocation is controlled by maintaining the size of the period constant. As congestion arises, the amount of allocated credits is reduced. Once it gets to a minimum value (e.g. equivalent to maximum size of an Ethernet packet), PACE starts increasing the period. The use of two control variables allows PACE to guarantee the important properties listed in Section I:

- **Credits** guarantee that no sender transmits more data than the core network can support. This ensures that we have no congestion in the network core, i.e. no data packets are lost. Rate-based methods rely on the arrival of a new rate that supersedes the old rate. In case of congestion, the sender may keep sending packets that can aggravate the problem. Thus, credits guarantee isolation and that flows only use the resources that are explicitly allocated. This improves *fairness*, especially for short-lived flows.
- The **allocation period** guarantees that the protocol operates *efficiently*. Intuitively, the period is short when there is no congestion. As the number of concurrent flows increases, PACE also increases the period of allocation. If an endpoint does not use its entire allocation, the allocations are quickly readjusted in the next period. Re-allocating credits periodically also guarantees *fast convergence* to a fair distribution of resources. Increasing the convergence to the fair state allows the protocol to be fair even for short flows. Dynamic adjustment of period allows PACE to *scale* while maintaining fairness. With longer periods, there are more credits to allocate and therefore more flows can be supported.

Another goal of PACE is to provide a mechanism to control the priority of flows. Thus, the number of credits allocated to a PACE flow is determined by the relative importance of the two endpoints. This differentiation is defined by assigning weights for both endpoints. *Controllability* is achieved by setting of *weights*.

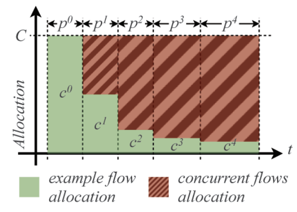


Fig. 1. Allocation by constraining the number of credits and/or the period as congestion grows.

III. PACE PROTOCOL

A. *Entities and Messages*

PACE involves three main entities. *Senders* initiate the connection and start sending data whenever they receive credits. *Receivers* allocate credits periodically to their peer senders. *Forwarders* are nodes located along the path traversed by a flow. These nodes can reduce the number of credits allocated by receivers. The goal is that forwarders only allocate credits that they can handle and will not overload their output queues.

PACE uses four types of messages: *request* - used by senders to initiate a connection. It carries the weight of the sender; *allocation* - sent by receivers to assign credits to senders. It carries the allocation fields, the weight of the receiver and the period of allocations; *data* - used to send data. The first data message sent in response to an allocation message provides feedback on the period and allocation; *terminate* - used to terminate the flow, remove state and release any resources allocated to the flow.

B. *Credits and Period of Allocation*

As mentioned before, PACE receivers periodically allocates credits to the senders. The *period of allocation* (p) plays an important role. First, it defines the granularity of credit allocation. The total amount of credits sent by a receiver or a forwarder corresponds to the link capacity R, multiplied by the period p. Second, the period also defines the time credits are valid. Credits cannot be used indefinitely. PACE defines a mechanism to expire credits based on logical clocks. The logical clock is incremented on every period. PACE requires p to be greater than one RTT. We define *validity of a credit* to be $2 \cdot p$ to allow for all the data packets for one period to arrive at the receiver with valid credits.

In our implementation, credit allocations are encoded in 2-byte fields, and 1 credit is equal to 8 bytes. For simplicity, from now on we refer to allocations in bytes (before conversion to credits). The period is encoded in a 1-byte field in control messages. The value is then multiplied by 20µs to obtain the real period.

C. *Receiver Logic*

The receiver is responsible for sending allocations for each flow periodically. The receivers maintain an estimation of how many credits each flow will be able to transmit in the next period, i.e. the flow's *demand*. This value is sent in the allocation message to inform forwarders about the flows resource requirements. Three fields in a credit allocation message are used to compute the current allocation and estimate future demand. All fields are initialized by the receiver and can be changed by forwarders. The allocation fields are:

- The field *potential demand (d_p)* carries the estimate of the demand of the flow for the next period. This value is initialized by the receiver with the maximum possible allocation (i.e., $p \cdot R$). The value can be decreased by forwarders and the sender returns the final value back to the receiver in the first data message.
- The field *demand (d)* carries the demand of the flow for the current interval. This value is initialized with the last potential demand that the receiver got from the sender. This field makes sure that all forwarders are informed of the demand of the flow.
- The field *allocation (a)* carries the actual allocation for the current period and it is initialized with the potential demand received from the sender. This field carries the actual number of credits that all the forwarders in the path can allocate for the current interval. For instance, at a given instant, a forwarder may have all credits allocated. In this case, the allocation value would be zero.

All nodes use the demand value to compute allocations instead of the allocation value. This ensures that the flow's correct demand will be taken into account when computing future allocations. Section III.E describes how forwarders compute the allocations for the flows.

In order to be able to differentiate flows, receivers can have different weights. The receivers can also combine their weight to the senders' weights to differentiate flows.

D. *Sender Logic*

When the sender receives an allocation message, it starts sending data. Using the period and the number of credits, the sender can determine how much data can be sent and at what rate. In the first data message, the sender includes the received allocation, potential demand and period back to the receiver.

E. *Forwarder Logic*

The forwarders need to verify if they have enough capacity to meet each allocation request. If they do not have enough capacity they can either reduce the number of credits or increase the period of allocation. An overview of the allocation process in forwarder i is depicted in Fig. 2. One of the concerns of our algorithm is to minimize processing delays of allocation packets. Thus, we split the allocation algorithm into an *offline* and an *online* component. PACE's offline component computes two variables periodically: *allocation per weight* (a_w) and *remaining allocation* (r). The demand computed at the previous forwarder, d_{i-1}, and the weight of the flow, w, are used to feed the offline processing data structure. The online processing uses the received allocation fields and the output of the offline processing, to compute the output allocation fields.

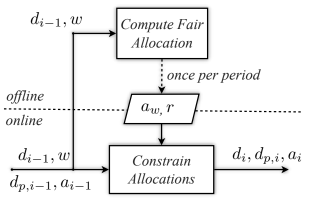


Fig. 2. The allocation algorithm at the forwarders

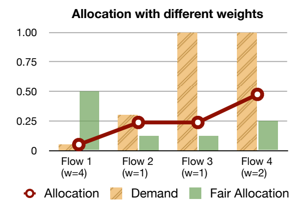


Fig. 3. Weighted Max-Min Fair Allocation given weight and demand.

1) *Computing Fair Allocations (Offline)*

Credit allocations are determined according to the weighted Max-Min fairness principle. In a weighted max-min fair allocation [10], the flows' fair shares are proportional to their weights. This policy is exemplified in Fig. 3. The figure represents four flows, each with a demand and a weight. The initial fair allocation (solid bar) is a share of the total possible allocation proportional to the flows weight. No flow receives more than its demand. If any flow does not use its entire fair allocation, the remaining credits are proportionally split among the other flows. Thus, in the figure, flow 1 receives only its demand despite its fair allocation being four times that of flows 2 and 3. The credits not used by flow 1 are distributed among the remaining flows according to their weight. Flow 4 receives twice the number of credits of the other flows' as its weight is twice the weight of the other two flows.

Typically, max-min fairness is achieved by using a progressive filling algorithm [11]. In each round all flows are assigned the fair share of the total available allocation. In subsequent rounds, the credits that were not used in the previous round are distributed among the other flows. This algorithm has a worst case processing complexity of $O(n^2)$ whenever a flow is updated. PACE uses a weighted max-min fairness algorithm based on a B-Tree that allows us to reduce the processing complexity to $O(\log n)$, the *Max-Min B-Tree*. PACE has a storage complexity of $O(n)$.

The rationale of the Max-Min B-Tree data structure is represented in Fig. 4. Given a set of flows $\{f_1, f_2, \dots, f_n\}$, each with demand and weight (d_i, w_i), we maintain a list of flows sorted according to the demand normalized by the flow's weight, $d_{w,i} = d_i/w_i$. We also maintain a cumulative sum of the demand and weight (D_i, W_i). To reduce complexity of computing the allocation, we maintain a set of L pointers $\{k_0, k_1, \dots, k_L\}$, which we call *allocation level pointers*. Each allocation pointer k_j, contains variables representing the total allocation for the level, C_j, a fair allocation per weight, $a_{w,j}$ and the non-allocated credits or remainder, r_j. Each allocation level pointers represents one round of allocation. For instance, the first pointer (k_0) represents the initial allocation obtained by dividing the maximum allocation by all the flows. In the example of Fig. 4, the total allocation $(C_0 = 1)$ is divided by the total weight, providing an allocation of 0.1 per weight unit. Thus k_0 points to f_1, which is the last flow requiring less than its allocation $(d_{w,f_1} = 0.08 < 0.1)$. The second pointer (k_1) distributes the remainder of the first level, r_0, among the rest of the flows. As shown in Fig. 4, the flows to the left of an allocation level pointer are flows whose demand (d_w) is fulfilled by the allocation of the pointer. Consider f_i to be the flow just before allocation level pointer k_j. Then the values of k_j are given by:

$$w_j = W_n - W_{i-1} \tag{1}$$

$$C_j = r_{j-1} \tag{2}$$

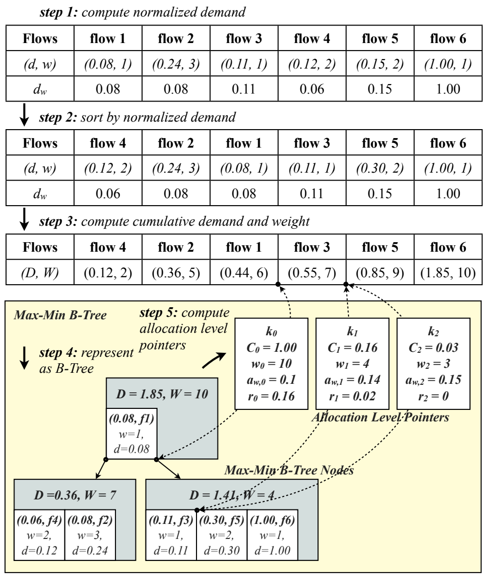
$$a_{w,j} = C_j/w_j + a_{w,j-1} \tag{3}$$

$$r_j = \min\{0, a_{w,j} \cdot (W_i - W_{i-1}) - (D_i - D_{i-1})\} \tag{4}$$

The last pointer has either no credits left $(r_L = 0)$ or no more flows requiring allocations $(w_L = 0)$. The allocation values used in the online processing are $a_w = a_{w,L}$ and $r = r_L$.

The sorted list is represented as a B-Tree. B-Trees are hierarchical structures that allow any operation (insert, delete, search and/or update) in $O(\log n)$ time. Each node in the B-Tree keeps track of the total demand and weight of its sub-tree. These variables are updated when inserting or deleting key without increasing processing complexity. The processing complexity of updating the levels is $O(L \cdot \log n)$, where L is the number of levels and n is the number of flows. Typically, three to four levels are enough to allocate all the credits.

In our implementation, all demand and allocation values in the B-Tree are normalized. Each flow has a given demand for a specified period. The normalized value corresponds to the demand divided by the total credits that the interface can transmit within the period of the flow. This normalization allows a node to vary the period of allocation without changing the Max-Min B-Tree values (see Section III.E.3). When the period of the forwarder is larger than the period of the flow, the forwarder update the period of the flow (in the allocation message) to match the local period.


step 1: *compute normalized demand*

Flows	flow 1	flow 2	flow 3	flow 4	flow 5	flow 6
(d, w)	*(0.08, 1)*	*(0.24, 3)*	*(0.11, 1)*	*(0.12, 2)*	*(0.15, 2)*	*(1.00, 1)*
d_w	0.08	0.08	0.11	0.06	0.15	1.00

step 2: *sort by normalized demand*

Flows	flow 4	flow 2	flow 1	flow 3	flow 5	flow 6
(d, w)	*(0.12, 2)*	*(0.24, 3)*	*(0.08, 1)*	*(0.11, 1)*	*(0.30, 2)*	*(1.00, 1)*
d_w	0.06	0.08	0.08	0.11	0.15	1.00

step 3: *compute cumulative demand and weight*

Flows	flow 4	flow 2	flow 1	flow 3	flow 5	flow 6
(D, W)	(0.12, 2)	(0.36, 5)	(0.44, 6)	(0.55, 7)	(0.85, 9)	(1.85, 10)

Max-Min B-Tree
step 5: compute allocation level pointers
step 4: represent as B-Tree
k_0 $C_0 = 1.00$ $w_0 = 10$ $a_{w,0} = 0.1$ $r_0 = 0.16$
k_1 $C_1 = 0.16$ $w_1 = 4$ $a_{w,1} = 0.14$ $r_1 = 0.02$
k_2 $C_2 = 0.03$ $w_2 = 3$ $a_{w,2} = 0.15$ $r_2 = 0$
D = 1.85, W = 10
(0.08, f1) w=1, d=0.08
Allocation Level Pointers
Max-Min B-Tree Nodes
D =0.36, W = 7
(0.06, f4) w=2, d=0.12
(0.08, f2) w=3, d=0.24
D = 1.41, W = 4
(0.11, f3) w=1, d=0.11
(0.30, f5) w=2, d=0.30
(1.00, f6) w=1, d=1.00

Fig. 4. Rationale of the Max-Min B-Tree Data Structure.

2) *Constrain Allocations (online)*

In the online procedure, the following expressions are used to compute the output values:

$$d_i = \min\{C(p) \cdot w \cdot a_w, d_{i-1}\} \tag{5}$$

$$d_{p,i} = \min\{C(p) \cdot w \cdot a_w + r, d_{p,i-1}\} \tag{6}$$

where $C(p)$ is the number of credits computed using the period in the allocation message $(p_{i-1} \cdot R)$.

As shown in Eq. (5), the output demand only depends of the fair allocation algorithm. However, the actual allocation for the current period, a_i depends on the forwarder having enough free credits to allocate when the allocation message arrives. The forwarder can allocate a total of credits equivalent to $2 \cdot C_0 = 2 \cdot p \cdot R$. The forwarder keeps a counter, $A(t)$, of the allocated credits. When the forwarder allocates new credits, it decreases the allocated credits from the counter. When a data message is received the counter is increased.

The allocation a_i is obtained using Eq. (7), where $Q_u(t)$ is the current usage of the output queue.

$$a_i = \min\{2 \cdot C_0 - A(t) - Q_u(t), d_i\} \tag{7}$$

In typical rate-based protocols like RCP [6], whenever an allocation message arrives, the flow immediately receives its fair allocation. As fair allocation changes with the arrival of new flows, this may lead to periods of congestion. D^3 [8] avoids this issue by pausing every non-deadline flow for one RTT. This solution leads to unnecessary latency and under-allocation. PACE avoids this over-allocation by filtering the credits based on queue usage and allocation not yet used.

3) *Dynamic Periods*

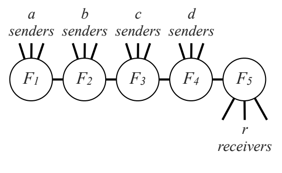
The period of allocation influences the performance of PACE. If we use small periods, the flows' rates adapt quickly when a flow starts or ends. However, as the number of concurrent flows increases, the allocation per flow gets smaller. At some point, smaller allocations mean shorter packets increasing the overhead per packet. In order to allow PACE to scale to a large number of concurrent flows, we increase the period of allocation.

PACE increases the period when the allocation for level pointer k_0 is not enough to transmit two maximum sized packets. The keys in the B-Tree are normalized with respect to the total possible allocation. When the period is increased, the demand of each flow would increase proportionally. Thus, the only value that needs to be updated is the total allocation, C_0. Also, flows in a forwarding node can have different allocation periods. For instance, a flow that is experiencing congestion in some other node will have a higher period. Thus, the allocation values in equations (5) and (6) are computed using the period values contained in the received allocation message.

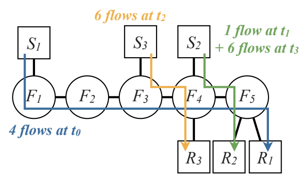
PACE maintains the allocation period as small as possible to maintain fast convergence. When the number of concurrent flows decreases and the base fair share increases, we decrease the period. When reducing the period we only need to decrease the total allocation, C_0. Equation (6) guarantees that no credits are allocated until old credits are used or expire. If the first term is negative due to high queue usage or high number of allocated credits, the filtered allocation will be zero.

IV. Experiments

We implemented PACE using NS-3 [12]. We also implement two recent proposals: DCTCP [5] and D^3 [6]. D^3 is designed having deadline-aware flows in mind. Here we evaluate it as a generic congestion control protocol without the deadline aware mode. In this case, D^3 is similar to an optimized RCP [4]. We use the parking-lot topology to study how the protocols behave under two different traffic patterns. To emulate a DC deployment the simulation network has a very small RTT (~150µs) and links operate at 1Gbps. An initial period of 200µs was used in the PACE implementation.



(a)



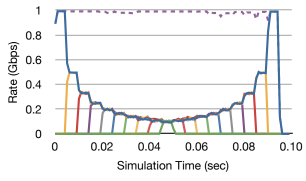
(b)

Fig. 5. Scenarios: (a) Sequential flows. (b) Composite traffic pattern.

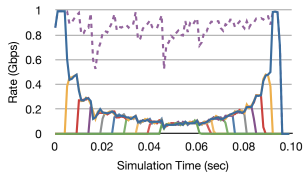
A. *Scenario 1: Fairness and RTT*

Consider the scenario in Fig. 5(a). We start 10 flows sequentially, each one separated by an interval of 5ms. Flows start from different senders, picking a sender from each of the four groups in a round-robin fashion (i.e. *a*, *b*, *c*, *d*, *a*, *b*, ...). We then end the flows sequentially in the inverse order using the same 5ms interval between each termination. The experiment is run for PACE, D^3, DCTCP and TCP.

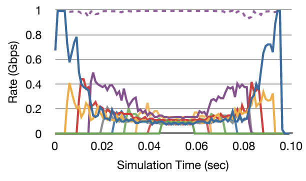
The rates for each flow are represented in Fig. 6(a-d). The dashed line represents the total rate at the bottleneck link (F_4-F_5). PACE obtains a fair distribution of bandwidth and converges quickly to a steady state. D^3 also distributes resources fairly but it does not use the channel efficiently. This is due to two reasons: the low granularity of resource allocation (units of 1bytes/µs); and convergence to full utilization is done by adjusting the total allocated capacity, which takes several RTTs to converge. DCTCP converges slowly to the fair state and, therefore, flows with lower RTT are unable to converge to the fair allocation. Finally, TCP does not converge to the fair state during the time frame of the experiment. TCP's additive increase and multiplicative decrease lead to drastic variations in the bitrate. Also, TCP adjusts at the cost of packet loss, which does not happen with the remaining protocols.



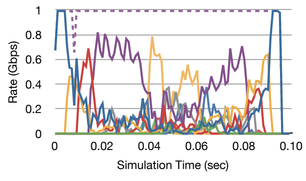
(a) PACE flows rate.



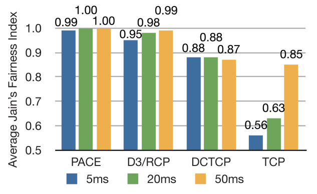
(b) D3 flows rate.



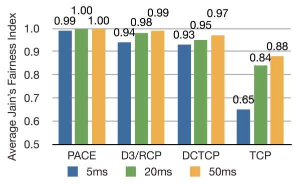
(c) DCTCP flows rate.



(d) TCP flows rate.



(e) Fairness (different RTT).



(f) Fairness (equal RTT).

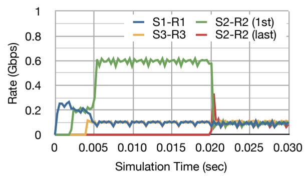
Fig. 6. Results for a sequence of flows sharing a common bottleneck. (a-d) Throughput of each flow for different protocols. (e) Average fairness with flows with the different RTT. (f) Average fairness for flows with same RTT.

We repeat the experiment but now all flows start from sender of group *a*, i.e. all flows have the same RTT. We also repeat the two experiments for longer periods between the start and end of the flow (20ms and 50ms). We then compute the average Jain's Fairness Index [13] throughout the experiment. Fig. 6 (e) and (f) show the average fairness index, for the scenarios with different RTT and equal RTTs, respectively. PACE maintains fairness for short flows and different RTTs. D3 also distributes resources fairly independently of RTT, but does not achieve the fair state as quickly (~5% drop in fairness for shorter flows). The results obtained for DCTCP and TCP demonstrate the sensitiveness of these protocols to the differences in RTTs.

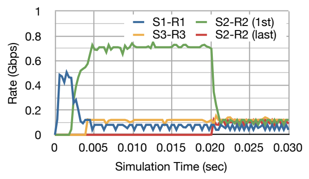
TABLE 1. FAIRNESS AND EFFICIENCY WITH A HIGH NUMBER OF CONCURRENT FLOWS

Properties	64 Flows			250 Flows	
	PACE	***D3***	***DCTCP***	***PACE***	***DCTCP***
Fairness (Jain)	99.2%	98.2%	98.9%	96.3%	39.9%
Efficiency	98.9%	93.1%	99%	98.9%	99.2%

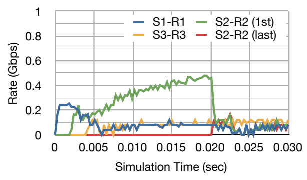
Scalability: We run one more experiment using the same scenario (Fig. 5(a)). We increase the number of concurrent flows (all starting from one of the *a* senders, i.e. same RTT) to 64 and 250 flows. The fairness results are presented in Table 1. D^3 has a theoretical limit of 125 flows for 1Gbps link due to the granularity of allocation. Thus, it is not possible to run the experiment for 250 flows using D^3. All protocols deal reasonably well with 64 simultaneous flows. D^3, however, has a 7% penalty in utilization. PACE maintains the fairness guarantees even when we increase the number of flows to 250. As a reactive protocol DCTCP uses timeouts once the delay increases. This leads to packet retransmissions, drastic window adjustments and consequent fairness degradation.



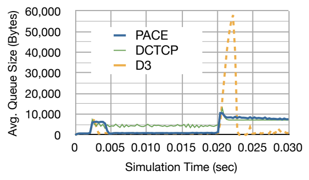
(a) PACE flows rate.



(b) DCTCP flows rate.



(c) D3 flows rate.



(d) Queue usage at F_4 (interface to F_5).

Fig. 7. Throughput and Queue usage for scenario 2.

B. Scenario 2: Composite Traffic Pattern

The second scenario is depicted in Fig. 5(b). There are 4 phases: at $t_0 = 0$, we start four flows from S1 to R1. At $t_1 = 2ms$, we start one flow from S2 to R2. At $t_2 = 4ms$, we start 6 more flows from S3 to R3. Finally, at $t_3 = 20ms$, we start 6 more flows from S2 to R2. In Fig. 7(a-c), we represent one flow from each group: one flow S1-R1, the first flow S2-R2, one flow from S3-R3 and one flow from the last group S2-R2. Once the flows S3-R3 start, S1-R1 throughput is reduced due to F3. The first flow S2-R2 takes advantage of the extra free bandwidth. DCTCP reacts quickly to traffic changes. However, throughput converges to an unfair distribution of resources. The represented flow from S1-R1 is using more than its fair share and, starting from t_2, the flow S2-R2 starts using almost 20% more than its fair share. D^3, however, presents more problematic results. In order for the first flow S2-R2 to use the available capacity, D^3 increases the allocation above the physical link capacity. When the new S2-R2 flows start, the total allocation largely surpasses the link capacity leading to queue build up. In Fig. 7(d), we can see the effect that D^3 has on the queue of forwarder F_4.

Controllability: To illustrate PACE's controllability we run scenario 2 with the following flows: one flow from S1-R1 with weight 2 at $t_0 = 0$; one flow from S2-R2 with weight 1 at $t_1 = 1.5ms$; and one flow from S3-R3 with weight 4 at $t_2 = 3ms$. All flows transmit 500KB. The results are shown in Fig. 8. When flow S2-R2 starts at t_1, it is competing with a flow that has twice its weight. Thus, it gets half of S1-R1 bandwidth. When the flow S3-R3 starts at t_2 it uses twice the bandwith of S1-R1. This in turn frees space for the flow S2-R2, which increases its throughput. As soon as flow S1-R1 finishes, both flows have the path freed and increase their throughtput to 100%.

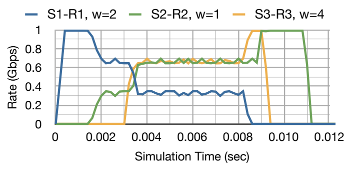


Fig. 8. Example of PACE's Controllability

V. CONCLUSION

In this paper we present PACE, a preventive explicit allocation congestion control protocol for Data Center networks. In multi-tenant Data Centers, as the number of applications rises, fast convergence to a fair distribution of resources becomes increasingly important. PACE allows nodes in the network to determine credit allocation and the period of allocation. These two variables allow PACE to adapt quickly to transient traffic patterns. The experiments demonstrate that PACE converges quickly to nearly perfect fairness with short flows, large number of concurrent flows, sudden traffic changes and flows with different weights.

REFERENCES.

[1] "Windows Azure", http://www.windowsazure.com/en-us/.

[2] "Amazon EC2", http://aws.amazon.com/ec2/.

[3] "Google Cloud Platform", http://cloud.google.com/.

[4] N. Dukkipati, "Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly,", PhD Thesis, 2007.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, and J. Padhye, "Data Center TCP (DCTCP)," in SIGCOMM '10, 2010.

[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in SIGCOMM '11, 2011.

[7] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," in SIGCOMM '12, 2012.

[8] K. Ramakrishnan, S. Floyd, and D. Black, "RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP," 2001.

[9] H. T. Kung and K. Chang, "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks," in IEEE INFOCOM 1995.

[10] S. Keshav, An Engineering Approach to Computer Networking. Addison-Wesley, 1997.

[11] D. Betsekas and R. Gallager, Data Networks. Prentice-Hall, 1992.

[12] Network Simulator 3: http://www.nsam.org.

[13] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation and shared computer system," Technical Report DEC-TR-301, Digital Equipment Corporation, 1984.