On Efficient Bandwidth Allocation for Traffic Variability in Datacenters

Jian Guo[1] Fangming Liu[1] Xiaomeng Huang[2] John C.S. Lui[3] Mi Hu[1] Qiao Gao[1] Hai Jin[1]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology.
[2]Ministry of Education Key Laboratory for Earth System Modeling, and Center for Earth System Science, Tsinghua University.
[3]Department of Computer Science & Engineering, The Chinese University of Hong Kong.

***Abstract*—Datacenter networks suffer unpredictable performance due to a lack of application level bandwidth guarantees. A lot of attentions have been drawn to solve this problem such as how to provide bandwidth guarantees for Virtualized Machines (VMs), proportional bandwidth share among tenants, and high network utilization under peak traffic. However, existing solutions fail to cope with highly dynamic traffic in datacenter networks. In this paper, we consider the effects of large numbers of short flows and massive bursty traffic in the datacenter, and design a novel distributed rate allocation algorithm based on the Logistic model under the control-theoretic framework. The theoretical analysis and experimental results using OpenFlow show that our algorithm not only guarantees the bandwidth for VMs, but also provides fast convergence to efficiency and fairness, and smooth response to bursty traffic.**

I. Introduction

Public cloud (e.g., [1]) has been increasingly popular since it provides economical resources for deploying today's business in a wide area. Using a simple pay-as-you-go charging model, cloud providers are able to lease the resources in the form of Virtualized Machines (VMs), with isolated performance on CPU and memory. However, to the best of our knowledge, current datacenters do not offer bandwidth guarantee for tenants. The network performance between two VMs can fluctuate significantly due to the interference of network intensive applications on other VMs [2]–[4]. This unpredictable performance leads to uncertainty in execution times of jobs, which increases the risk of revenue loss to tenants. As a result, providing performance guarantee for intra-datacenter communication has received significant interest in the networking research community. There is a general consensus among researchers about the need for *basic bandwidth allocation requirements* [5], such as bandwidth guarantee for VMs, proportionally sharing bandwidth resource among tenants and a high utilization of network resources.

However, previous measurement works (e.g., [6], [7]) reveal the characteristics of datacenter traffic. First, there are large numbers of short flows in datacenter networks. Second, short congestion periods are common across many links. Third, the datacenter traffic has significant variability. These unique characteristics make datacenter traffic remarkably different from any other network traffic. Hence, a practical bandwidth allocation should not only achieve the basic requirements, but also dynamically adapt to the traffic patterns in datacenters. Specifically, we consider the following *dynamic requirements*:

- *Handling of large nubmers of short flows:* Short flows may cause rate drops of other flows and reduce the link utilization. However, some of the short flows are too short to be rate-limited or controlled. The allocation policy should try to avoid sharp drops even at the presence of these short flows.
- *Smooth adaption in short congestion periods:* The allocation should change smoothly when congestions occur. If the rate-limit decreases the flows' bandwidth too sharply during congestions, the underlying TCP flows will fluctuate and the network performance will drop.
- *Fast convergence on bandwidth allocation:* The convergence of allocation process should be as fast as possible to keep pace with the significant variabilities in traffic. Otherwise, the allocation process can not converge to the result that satisfies the basic requirements.

However, designing a bandwidth allocation algorithm to handle the dynamic nature of datacenter traffic is far more difficult than just meeting the basic requirements. Previous work using rate-limit is either static, or based on TCP-like rate control, which deploys multiplicative decrease and makes the throughputs of VMs fluctuating. If there are large number of short flows or congestions, the rate control will become unstable and the network utilization may decrease sharply, thus making the bandwidth allocation hard to adapt to the highly variable traffic in datacenters. For example, [8], [9] use static or time-varying reservation for bandwidth guarantees. They can not achieve high utilization due to traffic variability in datacenters. [10]–[12] achieve work-conserving by using TCP-like rate control for VMs. The drawback is that the rate-limit fluctuates when congestions occur, leading to a remarkable decrease of the utilization performance degradation under lots of short flows.

In datacenters, the performance of cloud applications can be guaranteed through shaping traffic at the VM-level. Based on this method, the *objective* of this work is to design an efficient VM bandwidth allocation algorithm capable of handling highly dynamic traffic in datacenters. Unlike TCP-like algorithm, our solution uses explicit bandwidth information

The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). The research was supported in part by a grant from National Basic Research Program (973 program) under grant No.2014CB347800, by a grant from The National Natural Science Foundation of China (NSFC) under grant No.61370232.

978-1-4799-3360-0/14/$31.00

of physical links to enforce accurate rate control for traffic between VM-pairs, by taking advantage of the ability of global monitoring in datacenters. Such design enables a stable bandwidth allocation, and avoids the performance degradation caused by frequent decreases in network throughputs. Our rate control algorithm is inspired by the Logistic model (see Sec. II-B for details), which has fast convergence speed and smooth change rate. In the evaluation, it shows good adaptability to traffic variabilities caused by large number of short flows.

In summary, the contributions of this paper are as follows:

- By considering the characteristics of datacenter network traffic, we take the first attempt to present the dynamic requirements of designing an efficient bandwidth allocation algorithm in datacenters.
- We design a distributed rate control algorithm for traffic between VMs based on the Logistic model, and qualify various important properties of this algorithm under the control-theoretic framework, for both the basic requirements and the dynamic requirements.
- We implement a practical solution based on the OpenFlow. With experiments and trace-driven simulations, we show that the algorithm not only achieves bandwidth guarantees and work-conserving allocations, but also adapts to the highly variable traffic in datacenters.

II. Model and Design

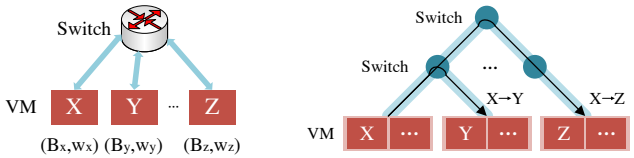
A. Datacenter Network Model

Datacenter network. We begin by introducing the datacenter network model. We consider a virtualized datacenter, where applications are running in VMs. Let $\mathbb{K} = \{1, 2, ..., K\}$ be the set of K VMs in the datacenter. Since we aim at guaranteeing bandwidth for applications at a VM-level, the traffic path between a pair of VMs is viewed as a *VM-flow*, where all traffic between this VM-pair counts as one flow. The active VM-flows in datacenter networks constitute a subset of directed VM-pairs, thus $\mathbb{N} \subseteq \{i_{x\to y} \mid x, y \in \mathbb{K}\}$ where VM-flow i is from VM x to y. We number them in order, $\mathbb{N} = \{1, 2, \ldots, N\}$, and denote the rate of VM-pair i as $v_i(t)$, $i \in \mathbb{N}$. The nodes (servers and switches) are connected by physical links and we use $\mathbb{M} = \{1, 2, ..., M\}$ to denote the set of M links across the datacenter network. Let C_l be the bandwidth of link l, $l \in \mathbb{M}$.

The rate allocation based mechanism and VM placement based mechanism are complementary to each other. Using the hose model [5] for bandwidth guarantees, we can accomplish VM placement through an existing approach (e.g., [8]). Hence, we focus on how to allocate the bandwidth to achieve work-conserving bandwidth sharing. Note that work-conserving in network means that if a link is the bottleneck, this link should be fully utilized.

Bandwidth requirements of VMs. Based on the hose model, we specify a throughput B and a weight w for each VM (Fig. 1(a)) to present the requirements of bandwidth guarantee and proportional bandwidth share, respectively. Both B and w are specified when a VM is created in the datacenter.

Bandwidth allocation model. As shown in Fig. 1(b), we allocate the bandwidth to VM-flows on each link. Let P_l denote



(a) Hose model. (b) Bandwidth allocation model.

Fig. 1. System model: use hose model for bandwidth requirements of VMs, and allocate bandwidth for VM-pairs on each link based on the requirements.

the set of VM-flows across link l, $\forall l \in \mathbb{M}$, and L_i denote the set of links passed by VM-flow i, $\forall i \in \mathbb{N}$. The guarantees and weights of VMs can be partitioned into VM-flows' guarantees and weights. For VM-flow i (from VM x to VM y), suppose X^r is the set of VMs receiving data from x, and Y^s is the set of VMs sending data to y. The weight can be set as

$$w_i = \frac{w_x}{|X^r|} + \frac{w_y}{|Y^s|}, \tag{1}$$

where $|X^r|(|Y^s|)$ is the number of elements in $X^r(Y^s)$. This way, the shared bandwidth of a tenant will be proportional to the total weights of its VMs [5]. The bandwidth guarantee of VM-flow i can be obtained by partitioning x's and y's bandwidth guarantees based on their weights [13], thus

$$B_i = \min\{B_x \frac{w_y}{\sum_{X^r} w}, B_y \frac{w_x}{\sum_{Y^s} w}\}. \tag{2}$$

We use the minimum in case that B_i exceeds the link capacity.

B. Logistic Model for Objectives

Considering the dynamic nature of datacenter networks and the shortcomings of previous solutions, our intuition is to use a *smooth* but *fast-growing* function as the rate, with gradual change rate at the beginning and around the equilibrium. This is exactly the situation in ecology where the dynamics of population $x(t)$ is modeled by the *Logistic model* [14] as:

$$\dot{x}(t) = rx(1 - \frac{x}{C}), \tag{3}$$

where r is the intrinsic rate of increase and C is the resources capacity denoted by the number of organisms. Note that the growth of population is proportion to the population itself, and has an inverse correlation to the resources.

As shown in Fig. 2, a Logistic-like rate controller can benefit the datacenter network from the following aspects: (i) The rates of VM-flows can be controlled according to the available bandwidth, and maintain at the equilibrium (maximum value) with no fluctuation. It can avoid severe decreases when congestions occur. (ii) The rate has an exponential convergence speed, which can quickly adapt to the highly variable traffic demands in datacenters. (iii) With a slow start speed, the bursty traffic caused by massive short flows can be flattened, and traffic fluctuation on bottleneck links can be reduced.

C. Design of Rate Allocation Functions

However, the Logistical model can not be directly applied to rate control of VM-flows, since it has a static upper bound (i.e., C) while we need dynamic rate limits for VM-flows. To achieve dynamic rate control in datacenters, the design should

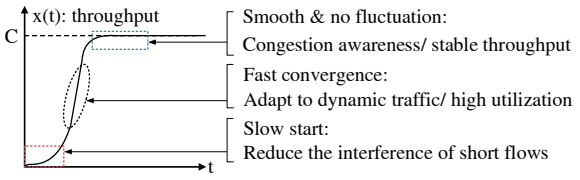


Fig. 2. Key mechanisms and benefits of the proposed datacenter rate control via the Logistic model.

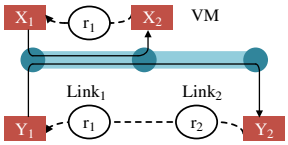


Fig. 3. The design uses explicit bandwidth information of links.

Notation	Meaning
$v_i(t)$	rate of VM-flow i
L_i	set of links on VM-flow i
$r_l(t)$	rate factor of link l
C_l	capacity of link l
P_l	set of VM-flows across link l
$\widetilde{r}_i(t)$	minimum rate factor in L_i

Fig. 4. Commonly used notations in this paper.

consist of two aspects: an algorithm on the source VM to control the rates of *VM-flows*, and an algorithm on the switch to feedback the bandwidth information of *links* to the source VM.

The design uses the explicit bandwidth information of physical links to accurately control the VM-flows' rates (Fig. 3). This is realizable because today's datacenters are of centralized architectures, and they are designed with the ability to perform global monitoring and management. Specifically, we first define a *rate factor* for each link, whose differential is:

$$\frac{\mathrm{d}r_l(t)}{\mathrm{d}t} = \beta r_l(t)\left(1 - \frac{\sum_{i\in P_l} v_i(t)}{C_l}\right), \tag{4}$$

where $\sum_{i\in P_l} v_i(t)$ represents the total throughput on link l.

Note that the rate factor increases if there is any available bandwidth, and decreases when the allocated bandwidth exceeds the link capacity C_l. It implies that there is a positive correlation between $r_l(t)$ and the available bandwidth. Hence, $r_l(t)$ can be used to dynamically limit the rates of VM-flows across link l. Therefore, the rate v_i of VM-flow i at the source can be controlled by the rate factor $\widetilde{r}_i$ from i's bottleneck link. We can express the differential of v_i as:

$$\frac{\mathrm{d}v_i(t)}{\mathrm{d}t} = \alpha v_i(t)\left(\ln \widetilde{r}_i(t) - \ln v_i(t)\right), \tag{5}$$

where $\widetilde{r}_i(t) = \min\{r_l(t) \mid l \in L_i\}$. Note that Eq. (5) is derived by taking the logarithm of the original $1 - \frac{x}{C}$ in the Logistic model. The reason is that we can quickly adapt the rate of VM-flow to the received rate factor within $O(\log\log)$ time (as we will prove in III-C) , thereby improving the convergence speed.

Since the actual utilization of links can not achieve 100% in real network based on TCP/UDP, we apply a ratio μ to the capacity C_l in our model. To achieve proportional bandwidth share among VMs, we multiply the rate factor with the corresponding weight. Thus, the *rate control model* in datacenter networks can be summarized as:

$$\begin{cases} \dfrac{\mathrm{d}v_i(t)}{\mathrm{d}t} = \alpha v_i(t)\left(\ln \widetilde{r}_i(t) - \ln v_i(t)\right), \forall i \in \mathbb{N} \\ \widetilde{r}_i(t) = w_i \min\{r_l(t) \mid l \in L_i\} \\ \dfrac{\mathrm{d}r_l(t)}{\mathrm{d}t} = \beta r_l(t)\left(1 - \dfrac{\sum_{i\in P_l} v_i(t)}{\mu C_l}\right), \forall l \in \mathbb{M}. \end{cases} \tag{6}$$

Remark: Note that v_i and r_l are coupled and form a feedback system. The rate factor is limited by the physical bandwidth and controlled by the throughput of VM-flows. The rate of each VM-flow is limited and controlled by the rate factor. Depending on the interaction between v_i and r_l, the system will converge to an equilibrium where the bandwidth is fully utilized. The convergence process corresponds to the population dynamic of the Logistic model.

To understand the above model, we consider the following simple example. When N VM-pairs share the same bottleneck link with bandwidth C, the equilibrium of the system can be derived from $\dot{v}(t) = \dot{r}(t) = 0$. Thus we have:

$$v_i = \widetilde{r}_i = \frac{w_i}{\sum_{i=1}^{N} w_i}\mu C_l = w_i r_l. \tag{7}$$

The rates of VM-flows are in proportion to their respective weights, and the bandwidth of this link is fully utilized. In the following section, we qualify the properties in detail.

III. Analysis via Control-Theoretic Framework

In this section, we analyze the rate control model via the control-theoretic framework, and provide insights on how its properties benefit datacenter network sharing.

A. Stability of Equilibrium

Since the differential equations in Eq.(6) form a nonlinear system, we use the Lyapunov stability theory to prove its stability. For the nonzero equilibrium of the system, let $\mathbb{M}' \subseteq \mathbb{M}$ be the set of bottleneck links. We have:

Theorem 1. For $v_i, r_l \in (0, \infty]$, where $i \in \mathbb{N}, l \in \mathbb{M}'$, the system formulated by Eq. (6) is *locally asymptotically stable* irrespective of capacities of bottleneck links and communication pattern of VMs.

Proof: We begin the proof with substitution. Let $\varphi_i(t) = \ln v_i(t)$ and $\phi_l(t) = \ln r_l(t)$, then Eq. (6) can be derived as

$$\begin{cases} \dfrac{\mathrm{d}\varphi_i(t)}{\mathrm{d}t} = \alpha(\widetilde{\phi}_i(t) - \varphi_i(t)), \\ \widetilde{\phi}_i(t) = \ln w_i + \min\{\phi_l(t) \mid l \in L_i\}, \\ \dfrac{\mathrm{d}\phi_l(t)}{\mathrm{d}t} = \beta\left(1 - \dfrac{\sum_{i\in P_l}\exp\{\varphi_i(t)\}}{\mu C_l}\right). \end{cases} \tag{8}$$

Let $f_l(\varphi(t))$ denote the function of $\varphi(t) = \{\varphi_i(t) \mid i \in P_l\}$, and $\frac{\mathrm{d}\phi_l(t)}{\mathrm{d}t} = \beta f_l(\varphi(t))$. Suppose $\frac{\partial F_l}{\partial \varphi_i} = f_l(\varphi)$. To apply the Lyapunov stability theory, we first construct a positive function. For all $i \in \mathbb{N}$ and $l \in \mathbb{M}$, we define

$$V(\varphi(t), \phi(t)) = \alpha \sum_{i\in\mathbb{N}} \frac{1}{2}\left(\widetilde{\phi}_i(t) - \varphi_i(t)\right)^2 + \beta \sum_{l\in\mathbb{M}'} F_l(\varphi(t)).$$

We have the partial differential of V on $\varphi_i(t)$ and $\phi_l(t)$.

$$\frac{\partial V}{\partial \varphi_i} = -\alpha(\widetilde{\phi}_i(t) - \varphi_i(t)) - \beta \sum_{l \in L_i} f_l(\varphi(t)),$$
$$\frac{\partial V}{\partial \phi_l} = \alpha \sum_{i \in P_l^*} (\widetilde{\phi}_i(t) - \varphi_i(t)),$$

where $P_l^* \subseteq P_l$ is set of VM-flows bottlenecked on link l, i.e., $\widetilde{\phi}_i(t) = \ln w_i + \widetilde{\phi}_l(t)$. Then we can derive the differential of V on time variable t.

$$\begin{aligned}\frac{\mathrm{d}V}{\mathrm{d}t} &= \sum_{i \in \mathbb{N}} \frac{\partial V}{\partial \varphi_i}\frac{\mathrm{d}\varphi_i}{\mathrm{d}t} + \sum_{l \in \mathbb{M}'} \frac{\partial V}{\partial \phi_l}\frac{\mathrm{d}\phi_l}{\mathrm{d}t} \\ &= \sum_{i \in \mathbb{N}} \left(-\alpha(\widetilde{\phi}_i - \varphi_i) - \beta \sum_{l \in L_i} f_l(\varphi) \right)(\alpha(\widetilde{\phi}_i - \varphi_i)) \\ &\quad + \sum_{l \in \mathbb{M}'} \alpha\beta f_l(\varphi) \sum_{i \in P_l^*} (\widetilde{\phi}_i - \varphi_i) \\ &= -N\alpha^2(\widetilde{\phi}_i(t) - \varphi_i(t))^2.\end{aligned}$$

Note that $\varphi_i'(t) = \phi_i'(t) = 0$ is the equilibrium of the system in Eq. (8). The function V is positive and the differential of V is negative except for the equilibrium. Hence, function V is a Lyapunov-candidate-function of the system and the equilibrium is proven to be locally asymptotically stable. ■

Insight: For VM-flows with given initial rates (> 0), they will converge to an equilibrium under the control of our model. The convergence depends on neither the link bandwidth in datacenters, nor the placement and communication patterns of VMs. The rate control for each VM-flow is completely independent, and can easily be scaled up via a distributed implementation. The following discussion is based on the fact that the system is asymptotically stable.

B. Achieving Efficiency and Fairness

Let $A_l = \sum_{i \in P_l} v_i$ be the aggregate rate of link l. Thus the utilization of a physical link can be represented by $u_l = \frac{A_l}{C_l}$. The link becomes a bottleneck when it reaches its maximum utilization, which is $u_l = \mu$ in our model. Eq. (5) indicates that the bottleneck of a VM-flow is the set of physical links in its path, whose rate factor is equal to the rate of this VM-flow. Let L_i^* be the set of bottleneck links of VM-flow i, then we have $L_i^* = \{l \mid v_i = w_i r_l\}$.

Theorem 2. In a multiple-bottleneck topology where N VM-flows share M links, if there exists a unique equilibrium, then the rates of VM-flows allocated by the algorithm in Eq. (6) achieve the *weighted max-min fairness*.

Proof: To decouple the weight from the rate for VM-flows, we first define a normalized rate x_i for each VM-flow, such that $v_i = w_i x_i$.

Suppose all rates of VM-flows have reached the equilibrium, then there should exist at least one bottleneck in the topology. Let $L_b^* \subseteq \mathbb{M}$ denote the non-empty subset of bottleneck links, where $L_b^* = \{l \mid u_l = \mu\}$. Thus, all physical links in L_b^* have reached the maximum utilization.

Since the rates of VM-flows are stable, each of them should pass through at least one bottleneck link, and by setting $x_i'(t) = 0$, we obtain the equilibrium rate of VM-flow i as $x_i = \min_{l \in L_i} r_l(t)$. Consider a bottleneck link $l \in L_b^*$. Let $P_l^* \subseteq P_l$ be the subset of VM-flows that has a bottleneck link l, thus

$$P_l^* = \{i \mid x_i(t) = r_l(t), L_i \cap P_l \neq \varnothing\}. \tag{9}$$

It means that such VM-flows not only pass through link l, but have a rate x_i equal to r_l.

For any bottleneck link $l \in L_b^*$, if there exists a path $j \in P_l - P_l^* \neq \varnothing$, whose bottleneck links do not consist of this link, thus $l \notin L_j^*$. There must be another bottleneck link $k \in L_j^*$, such that $x_j = r_k$. Since r_k is the minimum rate factor on link j, we have $r_k \leq r_m$ for any other link $m \in L_j$ along its path. Meanwhile, as link l is not the bottleneck of VM-flow j, the rate factor should satisfy $r_k \neq r_l$. Hence, we have $x_j < x_i$, which means that the VM-flows which are bottlenecked on link l must have the *same maximum* rate among all VM-flows across link l.

If we attempt to increase the rate of a VM-flow, we should decrease the rates of other VM-flows across its bottleneck link. However, this happens if and only if we decrease the rates of other VM-flows whose rates are *equal to or less than* this flow, which is exactly the definition of max-min fairness. In addition, if we organize the rates of all VM-flows in an increasing order $\{\widetilde{r_1}, \widetilde{r_2}, \ldots, \widetilde{r_n}\}$, we can remove the VM-flows with the lowest rates from the network and reduce the capacity of their corresponding links by the sum of these rates. The process can be repeated until there is no link/flow left. Hence, the normalized rates of VM-flows solve a hierarchy of optimization problems within each of them the minimal allocation is maximized, which is exactly the definition of max-min fairness. Finally, we conclude that the rate allocation $v_i = w_i x_i$ is weighted max-min fairness. ■

Insight: Our model can fully allocate the bandwidth of a bottleneck link to the VM-flows across it. This allocation result is a proportional sharing among VM-flows according to their weights. It indicates that the model meets the static requirements of work-conserving and network proportionality [5] in datacenter networks . For bandwidth guarantee, we will introduce a threshold based method in Sec. IV.

C. Rate of Convergence

Convergence to efficiency: We first propose a function to quantify the network utilization.

Definition 1. Given a constant $\lambda \in (0, 1]$. When n VM-flows are sharing a single bottleneck link l, the resource allocation is called λ efficiency if there exists time t_λ, for $t > t_\lambda$

$$h(t) = \frac{\sum_{i=1}^{n} v_i(t)}{\mu C_l} \geq \lambda. \tag{10}$$

Let $t = 0$ be the start time where $v_i(0) = v_0 \ll \mu C_l$ (v_0 is much less than μC_l), and the minimum t_λ when the system converges to λ efficiency is called the time of convergence to efficiency, thus $h(T_\lambda) = \lambda$.

The efficiency definition describes the utilization of the congested link when the system converges to the equilibrium.

To qualify the time of convergence to efficiency, we consider a scenario where n homogeneous VM-flows are sharing the same bottleneck link with C_l capacity. Since we aim at solving the upper bound of the convergence time, two cases should be considered, for v and r, respectively.

In the first case, we qualify the time for each VM-flow to converge to a constant feedback $r = \mu C_l/n$ (the value in equilibrium) for each VM-flow. Let t_v denote the convergence time of $v_i(t)$. Hence, the differential equation of $v_i(t)$ can be derived as

$$v_i'(t) = \alpha v_i(t)(\ln \widetilde{r}_i(t) - \ln v_i(t)) \tag{11}$$

where $\widetilde{r}_i(t) = \mu C_l/n \gg v_0$. Obviously, $v_i(t)$ is monotone increasing. With Definition 1, we have $v_i(t_v) = \lambda \mu C_l/n$ when the system converges to λ efficiency. Given the initial and convergent point, the convergence time t_v can be solved as

$$t_v = \frac{1}{\alpha} \ln \frac{\ln (nv_0/\mu C_l)}{\ln \lambda}. \tag{12}$$

In the second case, we assume that the rate of each VM-flow $v_i(t)$ is equal to the feedback $r_l(t)$ along its path. Let t_r denote the convergence time of $r_l(t)$ and the differential of $r_l(t)$ can be derived as

$$r_l'(t) = \beta r_l(t) \left(1 - \frac{\sum_{i \in P_l} v_i(t)}{\mu C_l}\right), \tag{13}$$

where $v_i(t) = r_l(t)$. Similarly, $r_l(t)$ is monotone increasing. With the initial and convergent values of $r_l(t)$, we obtain the convergence time t_r, such that

$$t_r = \frac{1}{\beta} \ln \left((\frac{\mu C_l}{nv_0} - 1) \frac{\lambda}{1-\lambda} \right). \tag{14}$$

Finally, we can derive the upper bound of the time of convergence to λ efficiency.

Proposition 1. Given n VM-flows across the same bottleneck link, the time of convergence to λ efficiency satisfies:

$$T_\lambda < t_r + t_v, \tag{15}$$

where t_v and t_r is given by Eq. (12) and Eq. (14), respectively.

Convergence to fairness: We start with defining a fairness function.

Definition 2. Given a constant $\rho \in (0, 1]$, when n VM-flows are sharing a single bottleneck l, the resource allocation is called ρ fairness if there exists time t_ρ, for $t > t_\rho$

$$g(t) = \frac{x_j(t)}{x_k(t)} \geq \rho, \tag{16}$$

where $x_j(t) = (\min_{i=1}^n v_i)/w_j$ and $x_k(t) = (\max_{i=1}^n v_i)/w_k$. Let $t = 0$ be the start time where $v_i(0) = \mu C_l/n$, and the minimum time t_ρ that the system converges to ρ efficiency, is called the time of convergence to fairness, thus $g(T_\rho) = \rho$.

The fairness definition describes the gap between the maximum and minimum normalized rates. To obtain the time of convergence to fairness, we assume that the rates of n homogeneous VM-flows have reached the equilibrium at the shared bottleneck with C_l capacity before another VM-flow v_j joins in. The new VM-flow has an initial rate such that $v_j(0) = v_0 \ll \mu C_l$. Hence, the fairness function $g(t)$ has an initial value $g(0) = nv_0/\mu C_l \ll 1$, which indicates that the system deviates from fairness after v_j joins in. Since $g(t) = v_j(t)/v_i(t)$ (based on Definition 2), the derivative of $g(t)$ can be derived as

$$g'(t) = -\alpha g(t) \ln g(t). \tag{17}$$

Thus $g(t)$ is monotone increasing as $g(t) \leq 1$. Formally, Eq. (17) can be derived as $(\ln \ln g(t))' = -\alpha$. With the initial and the convergence value of $g(t)$, we can solve Eq. (17) and obtain the time of convergence to fairness.

Proposition 2. Given n VM-flows across the same bottleneck link at a stable state, the time of convergence to ρ fairness with a new VM-flow satisfies the following equation

$$T_\rho = \frac{1}{\alpha} \ln \frac{\ln (nv_0/\mu C_l)}{\ln \rho}. \tag{18}$$

Insight: Definition 1 and 2 correspond to two common scenarios in datacenters: the utilization of a link suddenly peaks with the arrival of many flows, and a highly utilized link becomes congested as a new flow joins in. The burstiness of traffic and the short inter-arrival time of flows (average at 15ms [6]) require swift response of the rate control. Our model has $O(\log \log n)$ convergence time on the accuracy factors λ and ρ, for both convergence to efficiency and fairness. Hence, it is suitable for datacenter traffic.

D. Steady State Characteristics

The performance of the system in the neighborhood of steady state is characterized in this section. Similar to Sec. III, we consider an equilibrium where n VM-flows are sharing a single bottleneck l.

In the control theory, a nonlinear system can be considered linear about the equilibrium for small changes Δv and Δr [15]. By using the state space approach (Sec. 2.3 in [15]), we can obtain a linear approximation of the system in Eq. (6), which can be further analyzed by Laplace transform.

Let $\mathbf{v}(t)$ denote the rate vector of VM-flows, thus $\mathbf{v}(t) = [v_1(t), v_2(t), \ldots, v_n(t)]$, and $r(t)$ be the rate factor of the bottleneck link. Based on Eq. (6), we define $F_i(\mathbf{v}, r)(i = 1, 2, \ldots, n)$ and $H(\mathbf{v}, r)$ as the functions of $\mathbf{v}(t)$ and $r(t)$:

$$\begin{aligned} F_i(\mathbf{v}, r) &= \alpha v_i(t)(\ln w_i r(t) - \ln v_i(t)), \\ H(\mathbf{v}, r) &= \beta r(t)(1 - \frac{\sum_{i \in P_l} v_i(t)}{\mu C_l}). \end{aligned} \tag{19}$$

Suppose $O(\mathbf{v}^*, r^*)$ is the operating point, where $v_i^* = w_i r^*$ and $r^* = \mu C/\sum_{i=1}^n w_i$. Let $\Delta v_i = v_i - v_i^*$ and $\Delta r = r - r^*$ be the small changes of state. The linear differential equations for Δv_i and Δr should satisfy:

$$\Delta \dot{v}_i = \sum_{j=1}^n \left(\left.\frac{\partial F_i}{\partial v_j}\right|_{(\mathbf{v}^*, r^*)} \Delta v_j \right) + \left.\frac{\partial F_i}{\partial r}\right|_{(\mathbf{v}^*, r^*)} \Delta r, \tag{20}$$

$$\Delta \dot{r} = \sum_{j=1}^n \left(\left.\frac{\partial H}{\partial v_j}\right|_{(\mathbf{v}^*, r^*)} \Delta v_j \right) + \left.\frac{\partial H}{\partial r}\right|_{(\mathbf{v}^*, r^*)} \Delta r. \tag{21}$$

By solving the partial differential in the above equations at point O, we can obtain

$$\left.\frac{\partial F_i}{\partial v_j}\right|_{(\mathbf{v}^*,r^*)} = \begin{cases} 0, i \neq j \\ -\alpha, i = j \end{cases}, \left.\frac{\partial F_i}{\partial r}\right|_{(\mathbf{v}^*,r^*)} = \alpha w_i,$$
$$\left.\frac{\partial H}{\partial v_j}\right|_{(\mathbf{v}^*,r^*)} = \frac{\beta}{\sum_{j=1}^{n} w_j}, \left.\frac{\partial H}{\partial r}\right|_{(\mathbf{v}^*,r^*)} = 0.$$

Thus, the linear system can be derived as

$$\begin{cases} \Delta\dot{v}_i = -\alpha\Delta v_i + \alpha w_i \Delta r, \ i \in \mathbb{N}, \\ \Delta\dot{r} = \beta \sum_{i=1}^{n} \frac{\Delta v_i}{\sum_{j=1}^{n} w_j}. \end{cases} \tag{22}$$

Note that in the neighbourhood of O, each rate Δv_i can be denoted as $\Delta v_i = w_i \Delta v$, where Δv belongs to the same domain with Δv_i. By replacing v_i with $w_i \Delta v$ in Eq. (22), we find that all the equations for $\Delta\dot{v}_i$ can be derived as a group of linearly dependent equations. Hence, the linear system can be simplified as

$$\begin{cases} \Delta\dot{v} = -\alpha\Delta v + \alpha\Delta r, \\ \Delta\dot{r} = \beta\Delta v. \end{cases} \tag{23}$$

By using the Laplace transform for the system, the open loop transfer function can be obtained as follows

$$G(s) = \frac{\beta}{s} \cdot \frac{\alpha}{s+\alpha}, \tag{24}$$

which is a second-order system. Fig. 5 shows the block diagram of closed loop system, and the damping ratio can be derived as $\zeta = \frac{1}{2}\sqrt{\alpha/\beta}$.

To solve the relationship between α and β, we first consider the stability of the system. Note that when $s = j\omega$, the phase satisfies

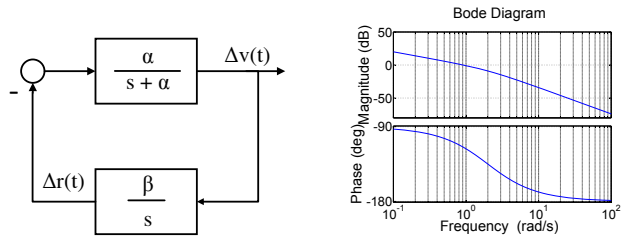
$$\angle G(j\omega) = -\pi + \arctan\frac{\alpha}{\omega} \subseteq (-\pi, -\frac{\pi}{2}), \tag{25}$$

thus the system is natively stable. We then consider the response of the system (Sec. 5.3 in [15]): (i) the swiftness of response, and (ii) the closeness of the response to the desired response. Generally, ζ is set as a value from 0.5 to 1. As ζ decreases, the system becomes more swift while the overshoot of rate becomes larger. Since they are contradictory requirements, a compromise must be obtained.

Insight: The characteristics of steady state imply the situation where steady long flows suffer interferences from large numbers of short flows, congestions, inaccurate rate enforcement, etc. When small changes occur in the system, we want the rate to swiftly respond to the changes and converge to another equilibrium. However, this will increase the overshoot of rate, and cause strenuous variation to the rate of VM-flow. To smooth the overshoot as well as to maintain the ability of quick recovery, we use $\zeta = 0.5$, thus $\alpha = \beta$.

IV. Algorithms Design in Datacenters

Let us now map our theory to a practical algorithm that can be implemented in datacenter networks.



(a) Block diagram of closed loop. (b) Bode diagram ($\alpha = 2$, $\beta = 1$).

Fig. 5. The block diagram and the Bode diagram of closed loop for second-order system.

A. Discretization on Time Series

As shown by the design goal in Sec. II, the rate control for each VM-pair is distributed and suitable for execution in VMs. The main issue is how to discretize the continuous functions into equivalent approximate recursions.

We use τ_v and τ_r as the time interval to update $v(t)$ and $r(t)$, respectively. This way, the time variables can be denoted as $k\tau_v$ and $k\tau_r$, where $k \in \{0, 1, 2, \ldots\}$ is the update rounds. For brevity, we use $v(k)$ and $r(k)$ to represent $v(k\tau_v)$ and $r(k\tau_r)$. The continuous functions are then transformed into time-discrete functions. The approximation can be derived as:

$$\frac{\mathrm{d}\ln v(t)}{\mathrm{d}t} \approx \frac{\ln v(k+1) - \ln v(k)}{\tau_v}, \tag{26}$$

and the same is true for $r(t)$.

Substituting Eq. (26) into Eq. (6) and replacing the time continuous functions with $v(k)$ and $r(k)$ yield recursion $\{v(k)\}$ and $\{r(k)\}$ as

$$\begin{cases} v(k+1) = v(k)^{1-\alpha\tau_v}\left(w_i r(\widetilde{k_r})\right)^{\alpha\tau_v}, \\ r(k+1) = r(k)\exp\left(\beta\tau_r(1 - \frac{\sum v(\widetilde{k_v})}{\mu C})\right). \end{cases} \tag{27}$$

Since the updates of $v(k)$ and $r(k)$ are asynchronous, each with their respective cycles, we use $\widetilde{k_r}$ to represent the corresponding rate factor when controlling the rates of VM-flows. Similarly, $\widetilde{k_v}$ is the exact rate that can be measured by a switch. Considering the delay τ_d of receiving rate factors for a VM-flow, we have $\widetilde{k_r} = \lfloor\frac{k\tau_v - \tau_d}{\tau_r}\rfloor$ and $\widetilde{k_v} = \lfloor\frac{k\tau_r}{\tau_v}\rfloor$.

In fact, in the implementation, $v(\widetilde{k_v})$ and $r(\widetilde{k_r})$ can be obtained without the knowledge of $\widetilde{k_v}$ and $\widetilde{k_r}$. The VM's rate controller can use the latest received rate factor, and the switch only needs to count the throughput on each link.

B. Algorithm Design

On providing bandwidth guarantees for VM-flows, we set the bandwidth guarantee as a lower bound for the rate limit of each VM-flow. Specifically, when the rates of VM-flows in the recursions (Eq. (27)) are lower than the bandwidth guarantees, their rate limits will still be the guaranteed bandwidth. Thus, the rate limit is represented as

$$v_i^{\max} = \max\{B_i, v_i(k)\}. \tag{28}$$

Algorithm 1 Update bandwidth guarantees on VM x

Input: Set of connected VMs: $\mathbb{V} = \varnothing$
Sum of weights for VMs in $\mathbb{V}$: $S = 0$
VM's bandwidth guarantee and weight: (B, w)
Output: Bandwidth guarantees for VM-flows on x: $B_{x\to i}$

1: function **UpdateGuarantee**(weight w)
2: **for** $i \in \mathbb{V}$ **do**
3: $B_{x\to i} = B_x \frac{w_i}{S}$
4: **end for**
5: end function
6: function **OnEstablishConnection**(VM i)
7: $\mathbb{V} = \mathbb{V} \cup i$
8: $S = S + w_i$
9: UpdateGuarantee(w)
10: end function
11: function **OnCloseConnection**(VM i)
12: $\mathbb{V} = \mathbb{V} - i$
13: $S = S - w_i$
14: UpdateGuarantee(w)
15: end function

Note that if the VM-flow's traffic demand is larger than the bandwidth guarantee, it will at least be ensured with the guaranteed bandwidth. If the VM-flow's traffic demand is lower than the bandwidth guarantee, the underutilized guarantee will be shared among VM-flows whose traffic demands exceed the guarantees. This way, the algorithm provides bandwidth guarantees for VMs, and maintains work-conserving.

The design of the Logistic based Rate Controller (**LRC**) consists of two major parts:

Bandwidth guarantee assignment. In datacenters, not only the allocated rate but also bandwidth guarantees for VM-flows can be variable. When a VM establishes a connection to another VM, or closes the connection to a VM, the bandwidth guarantees of VM-flows should be updated. The algorithm for updating bandwidth guarantees is shown in Algorithm 1. The bandwidth guarantee for VM-flow $x \to i$ is updated in function *UpdateGuarantee*. The update is triggered by two events: (i) connect to a new VM, (ii) disconnect from a VM. Specifically, when VM x establishes connections to VM i which has no connection with x, the *OnEstablishConnection* function will be called and i is then added to the set of connecting VMs. In the end, the algorithm updates the bandwidth guarantees for all VM-flows on VM x. The *OnCloseConnection* function, which is called when VM x closes the connection to a VM, has a similar process.

Rate control. According to Eq. (28), the rate control for each VM-flow relies on the output (bandwidth guarantee) of Algorithm 1 and the distributed recursions in Eq. (27). The update for rate of VM-flow $x \to i$ can be accomplished in a cooperative manner as shown in Algorithm 2. On one hand, the receiver (VM i) periodically sends a trace packet to the sender (VM x), with a time interval τ_v. The switch updates the link factors with the measured aggregate rate based on Eq. (27) every τ_r. On the other hand, after receiving a trace packet, the switch will update the rate factor if the packet passes through a link with a lower rate factor. This way, when the packet arrives at the sender, the rate factor will be the smallest one along its path. For the sender, the *OnReceivePkt* function will

Algorithm 2 Rate control for VM-flow $x \to i$

Input: Parameter settings: α, β, τ_v, τ_r, μ
Bandwidth guarantee for VM-flow $x \to i$: $B_{x\to i}$
Output: Rate limit on VM-flow: $v_{x\to i}$

1: function **OnReceivePkt**(Packet pkt, receiver i)
2: $v_{x\to i} = v_{x\to i}^{1-\alpha\tau_v} * (w_{x\to i} * pkt.r)^{\alpha\tau_v}$
3: $v_{\max} = \max\{B_{x\to i}, v_i(k)\}$
4: Set rate limit $v_{\max}$ to VM-flow $x \to i$
5: end function *//sender*
6: function **Feedback**(time τ_v, sender x)
7: Sleep for τ_v
8: Feedback Packet pkt to sender x
9: end function *//receiver*
10: function **UpdateLinkFactor**(time τ_r, link l)
11: Sleep for τ_r
12: Get aggregate rate V_l for link l
13: $r_l = r_l * \exp\left\{\beta\tau_r(1 - \frac{V_l}{\mu C_l})\right\}$
14: end function *//switch*
15: function **UpdatePacket**(Packet pkt, link l)
16: **if** $r_l < pkt.r$ **then**
17: $pkt.r = r_l$
18: **end if**
19: Send packet out
20: end function *//switch*

be called each time it receives a trace packet. The function updates the rate of VM-flow, and enforces the rate limit to VM-flow $x \to i$. Note that the algorithm involves operations in switches. However, as hardwares in datacenter are highly customized and the operation is simple (a switch only needs to attach the same information to few control messages), it can be easily achieved.

C. Parameters and Tradeoffs

In this section, we discuss how to balance the tradeoffs in implementation with the parameters.

VM parameters, B and w. These two parameters determine the bandwidth allocation result for each VM-pair. By tuning the guaranteed bandwidths and weights, cloud providers are able to flexibly balance the tradeoff between *bandwidth guarantee* and *proportional sharing*.

Time parameters, τ_v and τ_r. The time parameters are used to control the frequency of updating the rate of VM-flows and rate factor of links. One can improve the *precision* of bandwidth allocation by using very small τ_v and τ_r, however such improvement will certainly bring about large *overhead* to the servers and networks. As our algorithm has an exponential order of convergence speed, the interval of updating rate can be larger than the TCP RTT in datacenters, which has an order of magnitude of 1ms. In datacenters, the inter-arrivals of 70% TCP flows from/to servers are periodic at about 15ms [6]. To keep pace with the variation caused by new flows, we choose the same order of magnitude (tens of ms) to for τ_v. For the switches, we use a larger time period for τ_r (hundreds of ms) to reduce the overhead.

Convergence parameters, α and β. The convergence parameters have an effect on the rate of convergence. According to Proposition 1 and 2, the time of convergence

to fairness/efficiency is inversely proportional to α and β. However, whereas increasing α and β can significantly reduce the convergence time, it will also cause a large overshoot when the rate approaches the equilibrium. In the simulations, we find that there exists a tradeoff between the *convergence speed* and the *smoothness* of VM-flow's rate. The principle of choosing the numerical values for α and β is to use the maximum value while keeping the overshoot under an acceptable ratio, in case of fluctuations. In our evaluations, we use $\alpha = \beta = 2\text{s}^{-1}$.

V. Performance Evaluation

We aim at answering two questions in the evaluations: (i) Whether LRC can achieve the basic requirements as specified in Sec. I? (ii) How does LRC perform under dynamic traffic comparing with other rate control algorithms?

A. Achieving the Basic Requirements

To validate that LRC can achieve the basic requirements in datacenter networks, we implement an OpenFlow based prototype for LRC. The prototype updates the rate factors in an OpenFlow controller, and uses Traffic Control (TC) tool to limit the rate of each VM-flow in a Linux based OS.

The experiments focus on the bandwidth allocation on a bottleneck link, where VM X and Y compete for the limited bandwidth (like Fig. 3). In our testbed, the link capacity is 1Gbps, and the observed throughput can maximumly reach at about 900Mbps. Hence, we set $\mu C = 900$. For time parameters, we use $\tau_r = 100$ms and $\tau_v = 50$ms. For α and β, we both use 2s^{-1}. The experiments can be divided into three groups, each targeting at one aspect of the requirements.

Bandwidth guarantee: In this experiment, X and Y, both with 450Mbps bandwidth guarantee, are sending data to different remote VMs which have the same guarantee. Fig. 6 plots the throughput of X while increasing the number of TCP flows of Y. With no guarantee, the throughput of X decreases sharply since Y has more flows than X. With LRC, the rate of X stays around 450Mbps irrespective of the flow-level competition. This indicates that the traffic of Y is rate limited, and LRC can provide application layer bandwidth guarantees for VMs by limiting other aggressive VM-flows.

Work-conserving: As LRC guarantees bandwidth for VMs, another question is whether LRC will share the spare bandwidth from underutilized guarantees among unsatisfied VM-flows. The results are shown in Fig. 7. In this example, X with one TCP flow competes with TCP/UDP background traffic of Y. Both X and Y have 450Mbps guarantees, and we plot the throughput of X with an increasing traffic of Y from 0 to 900Mbps. As Fig. 7 shows, when the rate of Y is below 450Mpbs, the rate limit of X is above 450Mbps. The total throughput on the link, which is around 900Mbps, indicates that X utilizes the spared bandwidth and the allocation is work-conserving. Note that X's throughput maintains at about 450Mbps. It validates that LRC guarantees the bandwidth of X in spite of the interference of TCP/UDP background traffic.

Proportional share: We assign weights to VMs such that the weight ratio of X's VM-flow to Y's VM-flow varies from $1:2$ to $1:8$. Table I presents the throughputs of VM X and Y. When the weights of VM-flows are close to each other

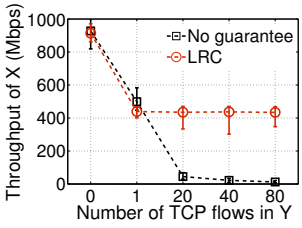


Fig. 6. Throughput of X (450Mbps guarantee) with different number of TCP flows of Y.

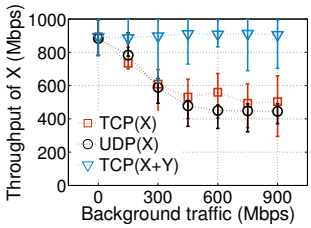


Fig. 7. Throughput of X with increasing throughput of Y (both 450Mbps guarantees).

TABLE I. Rate of X and Y with different weight ratios

Weight of $w_X : w_Y$	1:1	1:2	1:4	1:8
Throughput of X (Mbps)	441.0	306.9	190.5	112.9
Throughput of Y (Mbps)	444.1	580.2	687.9	747.9
Ratio of $v_X : v_Y$	1:1.01	1:1.89	1:3.61	1:6.62

(weight ratio < 2), the rates are proportional to their respective weights. However, the ratio of rates deviates from the weight ratio, when the latter ($w_X : w_Y$) grows larger. We trace the logs of received rate factors, and find that the cause is in the measurement error of link throughput in our testbed. The error brought by the received rate factor is non-uniformly magnified by the weights.

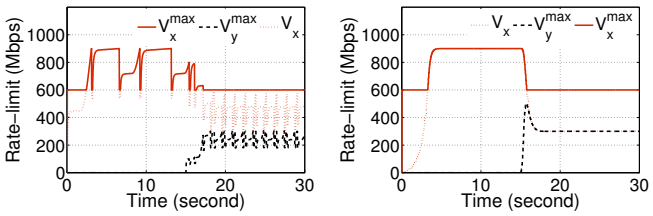
B. Performance on Dynamic Requirements

To precisely qualify the dynamic of rate limit in bandwidth allocation process, we simulate the algorithms and record the rates of VM-flows within each iteration. We compare LRC with a typical TCP-like rate control algorithm presented in [11], which is based on TPC-CUBIC [16]. Since [11] uses bandwidth guarantees as the weights of VMs, we decouple the guarantee from the weight so as to compare it with LRC. In the simulations, the link capacity is set as 900Mbps, and the rate allocation period is the same for TCP-like algorithm and LRC, both at 15ms. Particularly, we set $\alpha = \beta = 2\text{s}^{-1}$, $\tau_r = 100$ms, $\tau_v = 50$ms, similar to the experimental settings.

Convergence and congestions: Firstly, we characterize the convergence process under congestions on a single bottleneck.

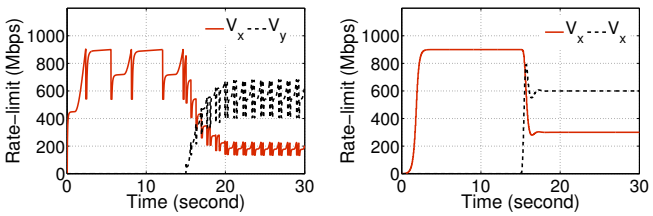
In the first case, VM X has a guarantee of 600Mbps. When $t = 15$s, a VM-flow (no guarantee) from Y joins in. Fig. 8 shows the dynamic of rate limit from 0 to 30s. Both LRC and the TCP-like algorithms have fast convergence speed to high utilization (4s) and fairness (3s). LRC is smooth during the whole period. However, in the TCP-like algorithm, when the VM-flow from Y joins in, the rate limit for X becomes fluctuating until it gets below the guarantee, and the rate limit for Y keeps fluctuating. The fluctuation leads to frequent changes of the rate-limit on underlying TCP flows, and may deteriorate the performance of the transport layer.

In the second case, both X and Y have no guarantees and the weight ratio w_X:w_Y is set as 1:2 (the same on the receiver). As Fig. 9 shows, LRC has a similar behavior as the previous simulation — fast convergence and smooth rate limit. However, the TCP-like algorithm not only becomes fluctuating when Y joins in, but also has an rate ratio (2.8:1).



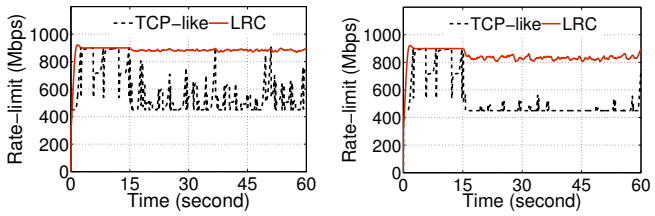
(a) TCP-like (b) LRC

Fig. 8. Rate-limit of VMs with $B_X = 600$Mbps, $B_Y = 0$ and the same weight. ($v_X^{max} = \max\{v_X, B_X\}$ and v_X is the rate-limit of the algorithms)



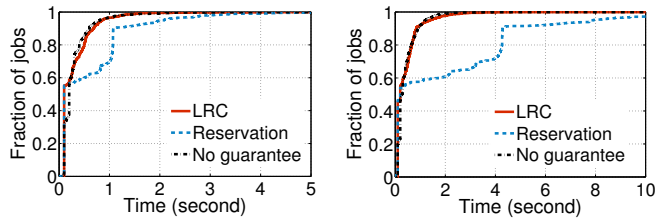
(a) TCP-like (b) LRC

Fig. 9. Rate-limit of VMs with $w_X : w_Y = 1 : 2$ and no guarantees. (v_X and v_Y are the calculated rate-limits of the algorithms)



(a) 5 short flows per sec. (b) 20 short flows per sec.

Fig. 10. Rate-limit of X (450Mbps guarantee) with arrival of short flows at 15^{th}s: 5 flows/s at average 17.4Mbps and 20 flows/s at average 66.2Mbps.



(a) Full bisection bandwidth. (b) $4\times$ oversubscription.

Fig. 11. The CDF of shuffle time for LRC, reservation and best effort manner.

One may question why the rate-limit in the TCP-like algorithm becomes more fluctuating when Y joins in. The reason is that TCP-like algorithms use multiplicative decrease when congestion occurs. In addition, in [11], the increment of rate has an inverse correlation with the rate (the authors call it as *rate-caution*). As a result, the VM-flow with small rate using rate-caution will be more aggressive than that using the original TPC-CUBIC protocol, thus fluctuations are more frequent when Y joins in. On the contrary, our algorithm applies a smooth rate limit to VMs, and will not decline the rate of TCP flows.

Frequent short flows: In this simulation, we evaluate the algorithms under frequent short flows. We consider the situation where VM X is sending long flows with a guarantee of 450Mbps, and has fully utilized the bottleneck link. We generate two groups of short flows randomly, one group with 5 flows/s, the other with 20 flows/s. The short flows pass through the bottleneck link of X from 15^{th}s to 60^{th}s. We test LRC and the TCP-like algorithm, each with 5 flows/s and 15 flows/s. Fig. 10 shows the rate limit of X. The TCP-like algorithm suffers severe drops in rate limit. As the average speed of short flows is very low (17.4Mbps for 5 flows/s and 66.2Mbps for 20 flows/s), the network bandwidth is wasted since the TCP-like algorithm enforces fluctuant limitations for the VM-flows. However, our algorithm is less sensitive to short flows. Even with 20 flows per seconds, the rate limit can stay flat and the decrement is about the average rate of all short flows. Such observations validate that LRC has advantages over the TCP-like algorithm under frequent short flows. Hence, we believe LRC is more suitable for datacenter environment.

Performance with Mapreduce workloads: To verify the performance of LRC under datacenter traffic, we simulate a cluster with 600 servers, and run the Mapreduce workload traces [7] collected from the same number of servers in Facebook production datacenter. The workload used in our simulation consists of up to 1000 jobs, with a maximum shuffle size of 667.1GB and a minimum of 0. We consider full bisection bandwidth and $4\times$ oversubscription network topologies with 40 racks and 5 aggregate switches. In the simulation, we compare LRC with static reservation and best-effort (no guarantee) bandwidth sharing. Since each server hosts 4 VMs and has 1Gbps bandwidth, the bandwidth guarantees for each VM in LRC and reservation policy are set as 250Mbps and 62.5Mbps, for full bisection bandwidth and $4\times$ oversubscription, respectively.

Fig. 11 shows the CDF of shuffling time. With equal bandwidth guarantee, LRC's shuffle times are the same as the shuffle times in the reservation policy for small jobs. For large jobs, the shuffle phase in LRC is faster than that in the reservation policy and the speedup is more obvious in $4\times$ oversubscription. This fits in with our expectation, since LRC ensures a lower bound bandwidth equivalent to the reserved bandwidth, and utilizes the spared bandwidth at the same time. In comparison with the best effort manner, LRC is faster for small jobs and a bit behind for large jobs. The reason is that when the rate of shuffle is less than the guaranteed bandwidth, LRC can guarantee the rate, whereas in the best effort manner, it needs to compete with the flows of other shuffle phase. This shows the effectiveness of bandwidth guarantees for small jobs in LRC. The drawback of LRC for large jobs comes from that LRC makes the underlying TCP flows less aggressive when the rate approaches the maximum. However, LRC still achieves 96% performance of the best effort manner, which validates the high network utilization of LRC.

VI. Related Work

Towards achieving predictable network performance for cloud applications, researchers have proposed numbers of approaches to share bandwidth in datacenter networks. The

main ideas in these works are two fold: The first idea focuses on VM allocation in datacenters, such as [8], [9]. Oktopus [8] uses VM placement to provide bandwidth guarantees. They both enforce static rate limits to reserve bandwidth for VMs. While these policies can provide predicable performance for VMs, they ignore the dynamic feature of datacenter traffic, thus they can hardly achieve high network utilization. Proteus [9] proposes a time varying reservation policy based on the bandwidth requirements of specific Mapreduce applications. However, the solution is limited to a few application types.

The other idea for sharing datacenter network is to allocate bandwidth for VMs after their placement by enforcing dynamic rate limit. Faircloud [5] presents the basic bandwidth requirements of bandwidth allocation problem and proposes three kinds of sharing policies. NetShare [17] achieves proportional bandwidth sharing among different VMs by using weighted fair queues. The policies in the above proposals need the support of per-VM queue in switches for rate control. This means they are hard to be scaled, due to the limited queues supported by each port at switches. The configuration is also complicated, as the communication patterns among VMs are changing. [10]–[12] leverage end-based rate limit to achieve work-conserving bandwidth allocation for VMs. Seawall [12] and ElasticSwitch [11] use TCP-CUBIC based rate control, and have fluctuations under bursty traffic when the rate limit is beyond the guarantee. ElasticSwitch also needs to number the packets for each destination in the switch, which is more complicated than our solution. EyeQ [10] uses a variant of RCP. EyeQ assumes a congestion free core and the rate control algorithm only has a linear convergence speed. [13], [18] develops a game theoretical allocation strategy that can flexibly balance the guarantee-proportionality tradeoff. The main drawback is its relying on precise traffic demand prediction. Finally, [19] applies the Logistic model in congestion control algorithms at the transport layer in the Internet, while our work use it for rate limits of VMs.

VII. Conclusion

Traffic in data centers are highly dynamic and bursty due to massive number of short flows. Previous work on bandwidth allocation for datacenters are not appropriate since rate limiting traffic is ineffective and causes system performance degradation. We propose using the Logistic model under the control theoretic framework, and present a practical distributed bandwidth allocation algorithm. Unlike previous proposals, our solution provides a stable and fast-convergent allocation process, which meets the basic and dynamic requirements of sharing datacenter networks. We demonstrate our algorithms in the OpenFlow based implementation and show its effectiveness in coping with traffic variability in datacenters, hence giving public cloud providers an additional performance guarantee features to users.

References

[1] Amazon elastic compute cloud. [Online]. http://aws.amazon.com

[2] F. Xu, F. Liu, L. Liu, B. Li, and B. Li, "iaware: Making live migration of virtual machines interference-aware in the cloud," *IEEE Transactions on Computers (TC)*, 2013.

[3] F. Xu, F. Liu, H. Jin, C. Wu, X. Liu, and B. He, "Managing performance overhead of virtual machines in cloud computing: A survey, state of art and future directions," *Proceedings of the IEEE*, 2013.

[4] Z. Zhou, F. Liu, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in saas clouds," in *IEEE INFOCOM*, 2013.

[5] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: Sharing the network in cloud computing," in *ACM SIGCOMM*, 2012.

[6] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM IMC*, 2009.

[7] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: a cross-industry study of mapreduce workloads," in *VLDB*, 2012.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[9] D. Xie, N. Ding, Y. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *ACM SIGCOMM*, 2012.

[10] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: practical network performance isolation at the edge," in *USENIX NSDI*, 2013.

[11] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: Practicalwork-conserving bandwidth guarantees for cloud computing," in *ACM SIGCOMM*, 2013.

[12] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[13] J. Guo, F. Liu, D. Zeng, J. C. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *IEEE INFOCOM*, 2013.

[14] J. D. Murray, *Mathematical biology*. springer, 2002.

[15] R. Dorf and R. H. Bishop, *Modern Control Systems*. Prentice-Hall, 2007.

[16] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, 2008.

[17] T. Lam and G. Varghese, "Netshare: Virtualizing bandwidth within the cloud," UCSD, Tech. Rep., 2009.

[18] J. Guo, F. Liu, H. Tang, Y. Lian, H. Jin, and J. C. Lui, "Falloc: Fair network bandwidth allocation in iaas datacenters via a bargaining game approach," in *IEEE ICNP*, 2013.

[19] X. Huang, C. Lin, F. Ren, G. Yang, P. D. Ungsunan, and Y. Wang, "Improving the convergence and stability of congestion control algorithm," in *IEEE ICNP*, 2007.