Improving Fairness of Network Bandwidth Allocation for Virtual Machines in Cloud Environment

Zhiyuan Shao, Kai Zhang, and Hai Jin
Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: {zyshao, hjin}@hust.edu.cn

***Abstract*—With the rapid development of cloud computing, data center's resource management has become a critical component in its business model. CPU, memory and network in a data center have many effective partitioning methods among the virtual machines that could offer high-efficiency, enhanced fairness and performance guarantee. However, in existing software based virtualization architecture (e.g., Xen), the network bandwidth is always decided by the number of connections, and no effective proportional share mechanism exists for the concurrently running virtual machines (VMs). Therefore, existing mechanism of network sharing will cause unfairness between VMs. In case that VMs should be given the same network bandwidth, there will be no such guarantee, especially when they establish different number of connections to outside world, even when the VMs are assigned with equal weight. In this paper, we present a mechanism that can improve the fairness of network bandwidth allocation in Xen para-virtualization in dynamic network environment. The experiment results demonstrate that every VM could share the network bandwidth almost according to its share even under real-world fluctuating network conditions.**

***Index Terms*—Xen, Network Virtualization, Bandwidth Unfairness, Dynamic Network Environment**

I. Introduction

Cloud computing [1] and big data [2] have brought new opportunities and challenges for researchers and IT companies. Amazon's EC2 [3] and GoogleApps [4] offer excellent cloud services that user could rent a virtual machine or a virtual cluster for service applications (e.g., web server) or even scientific computation without affording the expensive investment on hardware equipments. Data centers [5] that provide the cloud computing services employ virtualization technique to manage CPU, memory and storage resources [6]. Customers can get a measurable CPU performance, memory and disk capacity for their instances. For example, Amazon EC2 offers different kinds of instances: user can obtain an instance with one processor (i.e., VCPU), 1 EC2 Compute Units (ECU), 1.7 GB memory and 160 GB disk storage (i.e., the small currency instance) for the start-up application deployment, and replace it with better configured instance later.

Many approaches and theories have been introduced to efficiently manage virtual CPU, memory and I/O resource. Xen's Credit scheduler [7] is a widely used *Proportional Share* (PS) [8] CPU scheduler, which allocates CPU time slots to the virtual CPUs (VCPUs) of multiple VMs proportionally to create the illusion of multiple VMs running simultaneously on one physical machine. Every VM that coexists in one physical machine will be allocated CPU resources according to the assigned *weight* value. PS schedulers mainly focus on CPU fairness based on the predefined weight value.

Cloud environment usually needs performance isolation among its tenants, but TCP connections only achieve fairness among the streams. Figure 1 illustrates an example: by using one physical server, two virtual machines are invoked, and they are equally important (given exactly same portion of resources from the physical server). One of these two VMs is used for on-line video server and the other is used for company network file server. Ordinary users browse the video web site and watch the on-line videos, while employees of the company only launch one TCP connection to download a file from the file server. In such scenario, the video server could obtain larger percent of the available network resources, since it is more popular, and attracts large number of users. Conversely, far fewer simultaneous TCP connections are established for the file server, which means a rather poor share of network bandwidth. Therefore, as the virtual machine allocates network bandwidth on the basis of network connections, it cannot guarantee that the bandwidth resources are fairly distributed to these two virtual servers.

Although many methods and theories [9][10][11][12] were proposed to resolve this problem and expected to achieve fair allocation of network bandwidth resource in the virtualized environment, there are still many aspects to be resolved. Seawall [9] and Gatekeeper [10] only consider the problem in the static cloud network environment, however, real cloud environments are far more complex from the considered static environment for the continuously changing workload. In this paper, we present an approach that can allocate bandwidth for the VMs under fluctuating network conditions. This approach

978-1-5090-1925-0/16/$31.00 ©2016 IEEE

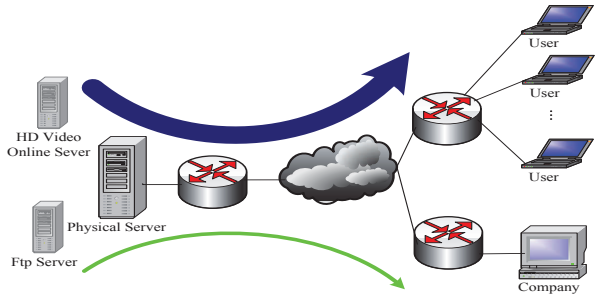


Fig. 1. An example of network bandwidth unfairness

assumes the available bandwidth value is an unknown variable. By observing the changes of bandwidth consumption of the virtual machines, our approach speculates the external utmost bandwidth value, and adjusts the bandwidth allocation of virtual machines to improve the bandwidth fairness at appropriate time. The experiment results show that this model can improve fair sharing of bandwidth allocation among virtual machines even when the network workload fluctuates abruptly. The degree of bandwidth unfairness can be improved from 81% to 8.7% by our approach in static network environment, and from 64% to 11% in real-world fluctuating network environment. The rest of this paper is organized as follows. Section II gives a brief survey on related works. Section III introduces the system model. Section IV presents our approach. Section V discusses our implementation. Section VI evaluates our approach. Section VII concludes the paper.

II. Related Works

In order to achieve fair share of network bandwidth among the VMs, Seawall [9] employs end hosts rate controllers to restrict the malicious bandwidth occupation and provide bandwidth isolation and proportional share among tenants. Seawall's rate limiters were implemented in hypervisor by using multi-queue *vNICs* (virtual Network Interface Card) to classify incoming packets into per-(sourceVM, destinationVM, path) queues. This mechanism allows Seawall to use any rate controlling algorithm. Furthermore, rate controller could send both positive and negative feedback message, so as to adjust window size or *ECN* (explicit congestion notification) [13] to correspond with other hypervisors.

Gatekeeper [10] employs similar mechanism as Seawall to achieve performance isolation for networking, i.e., also deploying rate limiter on the edge and distributing rate state to other servers. Gatekeeper allocates bandwidth among links not only simply according to the VMs number, which improves bandwidth fairness if one user purchases several VMs and another user buys only one VM. Every vNIC being assigned minimum and maximum rates, Gatekeeper uses weighted fair scheduler to provide minimum bandwidth guarantee and vNIC could reach maximum rate if more bandwidth is available. A more complex feedback communication mechanism than Seawall is introduced to cope with VMs traffic congestion.

NetShare [11] provides a statistical multiplexing mechanism to solve network unfairness allocation problem. In NetShare, bandwidth is divided into slices, tenants could share the bandwidth in a proportional way. It could share the network among services, applications and groups, rather than connections. NetShare employs *Deficit Round Robin* (DDR) [14] algorithm to realize group allocation to handle TCP flows and centralized bandwidth allocation mechanism to manage unused bandwidth for specific flows.

Although these methods try to solve one or more specific network allocation problems in virtualization environment, none of them provides a good mechanism to make themselves suitable in a real-world network environment, where the network situation may vary during fly.

III. System Model

We denote the multiple (N of them) VMs hosted in the same physical server $\mathcal{M}$ as set $S = \{VM_i | i \in [0, N-1]\}$, and denote the *weight* of the VMs on network bandwidth share as W_0, W_1, ..., W_{N-1}, where $\sum_{i=0}^{N-1} W_i = 1$. We denote the *demand* for network bandwidth of VM_i at moment T as D_i, and its *actual network bandwidth* at that moment as RBW_i. Note that RBW_i is measurable, and can be regarded as an known variable of the system, while D_i is decided by the applications running inside VM_i, which is not technically measurable at the hypervisor, and therefore considered as an unknown variable in this paper.

By considering the relationship between VM_i's network bandwidth demand D_i and actual network bandwidth RBW_i, there will be two possibilities: 1) $D_i \geq RBW_i$, and 2) $D_i < RBW_i$. In the former case, the applications in VM_i demands more network resources, and in the latter case, the VM voluntarily gives up its network resources. We define the set of VMs in former case as $\dot{S}$:

$$\dot{S} = \{VM_i | D_i \geq RBW_i, i \in [0, N-1]\}$$

and define the set of VMs of latter case as $\ddot{S}$:

$$\ddot{S} = \{VM_i | D_i < RBW_i, i \in [0, N-1]\}$$

We denote the *normalized network bandwidth* of VM_i as NBW_i, and compute it by Equ. 1:

$$NBW_i = \frac{RBW_i}{W_i} \tag{1}$$

For N VMs, the *average normalized network bandwidth*, denoted as NBW_{AVE}, is computed by Equ. 2:

$$NBW_{AVE} = \frac{\sum_{i=0}^{N-1} NBW_i}{N} \tag{2}$$

After defining the average normalized network bandwidth, we can further classify the VMs into two sets: the "strong" VMs and the "weak" VMs. Denote these two sets as S_{strong} and S_{weak} respectively, we have:

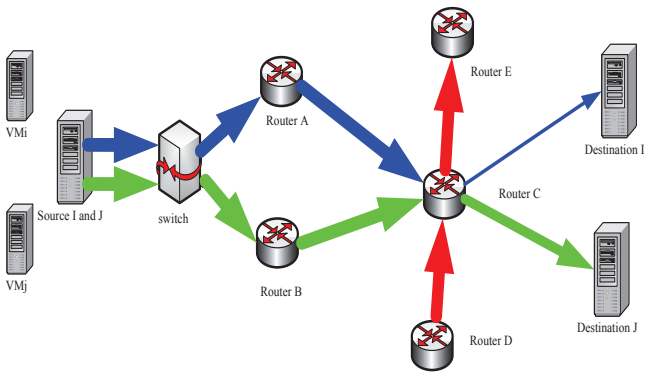


Fig. 2. A scenario of two VMs with unfair network bandwidth occupation

$$S_{strong} = \{VM_i | NBW_i > NBW_{AVE}, i \in [0, N-1]\} \quad (3)$$

and

$$S_{weak} = \{VM_i | NBW_i \leq NBW_{AVE}, i \in [0, N-1]\} \quad (4)$$

We denote a TCP connection established between one of the VMs (say VM_i) and a client machine $\mathcal{C}$ as a *link* of devices that the connection passes by, and represent it as $L = \{\mathcal{M}, R_1, ..., R_m, \mathcal{C}\}$, where each R_t, $t \in [1, m]$ denotes a switch or router device.

A. Problem statement

Suppose there are two same-weighted VMs, VM_i and VM_j, residing in $\mathcal{M}$, and during the fly, each of them communicate with client machine $\mathcal{C}$ and $\mathcal{C}'$ respectively. $L_i = \{\mathcal{M}, R_1, ..., R_m, \mathcal{C}\}$ and $L_j = \{\mathcal{M}, R_1', ..., R_k', \mathcal{C}'\}$ are used to denote the link of devices that the TCP connections are passed by. Now, let us suppose $L_i \cap L_j = \{\mathcal{M}, R_k, R_l\}$. That is, besides the physical server $\mathcal{M}$, there are two switch or router devices, i.e., R_k and R_l, in common for L_i and L_j. Fig. 2 illustrates this scenario.

During the fly, both $\mathcal{M}$ and R_k have abundant network bandwidth (say 100Mb/s) to support the communication. However, the router device R_l has only limited bandwidth (say 10Mb/s), since it has to allocate the majority of its bandwidth to the network flows that are from Router D to Router C. In order to occupy more bandwidth, VM_j invokes multiple (say 9) connections via link L_j. Since it is engineeringly impossible to let all devices (including R_l) to know the weights of the VMs, R_l will allocate its remaining bandwidth according to the number of connections. VM_j will thus occupy 9Mb/s from R_l, and VM_i occupies the remaining 1Mb/s, i.e., RBW_i = 1Mb/s, and RBW_j = 9Mb/s.

Obviously, from the perspective of both $\mathcal{M}$ and R_k, such network bandwidth allocations are "reasonable", since both RBW_i and RBW_j does not exceed the bandwidth limit that they are expected to occupy (50Mb/s as they are same-weighted). However, the unfairness on the network bandwidth of these two VMs in this scenario is obvious. The per se of such unfairness is because there is a bottleneck (i.e., R_l in above example) in the common path of the communication between VMs to outside world. In real-world networks, the fluctuation is likely to happen more frequently, which further implies that such communication bottlenecks will also appear and disappear frequently. The work presented in this paper is to achieve as fair as possible network bandwidth distribution among the VMs in the same physical server, by considering the fluctuating nature of real-world networks.

B. Objective function

Since there are two possible relationships between a VM's network bandwidth demand and its actual network bandwidth, considering the VMs that demand less network bandwidth than they actually have, we have the first objective:

$$Objective1 : \forall VM_i \in \ddot{S} \cap S_{weak}, MAX\{RBW_i\} \quad (5)$$

Considering the VMs that demand more network bandwidth than they are actually provided, we have the second objective:

$$\begin{aligned} &Objective2 : \forall VM_i \in \dot{S}, \\ &MIN(\sum_{i=0}^{N-1} |NBW_i - NBW_{AVE}|/NBW_{AVE}) \end{aligned} \quad (6)$$

That is, the objectives of our bandwidth adjustments is to guarantee the bandwidth provision for the VMs that demand less bandwidth than they can have, and minimize the standard deviation of bandwidth provision for the VMs that need more bandwidth than they actually have.

IV. Our Approach

In order to achieve fair share of network bandwidth, we take the "suppress the strong and help the weak" philosophy. Although it is easy for us to find the strong VMs (remember we put all VMs whose normalized network bandwidths are beyond the average into S_{strong}), suppressing a VM that belongs to S_{strong} does not necessarily mean a help to VMs that belong to S_{weak}. Come back to the example presented in subsection III-A. Suppressing the strong VM, i.e., VM_j, in that example will definitely help the weak VM, i.e., VM_i. This is because all communications of these two VMs have the same bottleneck, i.e., R_l. However, if the communications initiated from VM_i change their paths, and no longer pass by R_l, reducing the bandwidth of VM_j will no longer help improving the network bandwidth of VM_i.

Therefore, before trying to achieve fair share of network bandwidth by adjusting the bandwidth provision, we have to decide whether an adjustment is effective or not. Denote the bandwidth reduction of a strong VM as ΔBW_{strong}, and denote the bandwidth gain of the weak VMs on machine $\mathcal{M}$ as ΔBW_{weak}, we say an adjustment is *effective*, if following inequity satisfies:

$$|\Delta BW_{weak} - \Delta BW_{strong}| < \xi \quad (7)$$

where ξ is a small constant parameter, and is set to an empirical value of 0.5Mb/s in our implementation. In order to achieve fair share of network bandwidth for VMs that are hosted on $\mathcal{M}$, a bandwidth adjustment procedure listed in Algorithm 1 is invoked.

Note that in Algorithm 1, the effectiveness of an adjustment is justified by first summarizing all bandwidth gains of all "weak" VMs, and comparing the summation with the reduction on bandwidth of a selected strong VM. In order to maintain a dynamic balance, the bandwidth adjustment procedure will be invoked periodically with an interval of 5 seconds. During experiments, we observe that the role of a VM can switch from "strong" and "weak" from time to time.

Algorithm 1 Bandwidth adjustment procedure

1: **procedure** BANDWIDTHADJUSTMENT
2: compute RBW_i and NBW_i for all VM_i where $i \in [0, N-1]$
3: compute NBW_{AVE} according to Equ. 2
4: classify VMs into S_{strong} and S_{weak} by Equ. 3 and 4
5: $\overrightarrow{S_{strong}} \leftarrow$ sort S_{strong} by descending order.
6: **for all** $VM_i \in \overrightarrow{S_{strong}}$ **do**
7: reduce VM_i's bandwidth by ΔBW;
8: sleep for 0.5 seconds
9: re-compute actual network bandwidth for all VMs in S_{weak}
10: $\Delta BW_{weak} \leftarrow$ bandwidth gain for all VMs in S_{weak}
11: **if** $|\Delta BW_{weak} - \Delta BW| < \xi$ **then**
12: re-compute RBW_i
13: **if** $RBW_i > (NBW_{AVE} + \Delta BW)$ **then**
14: goto line 7
15: **end if**
16: **else**
17: return ΔBW to VM_i
18: **end if**
19: **end for**
20: **end procedure**

V. IMPLEMENTATION

In order to use the I/O resource efficiently, Xen proposes Split Device Driver Model. As shown in Figure 3, the device drivers for networking include the front-end driver in guest domain and back-end driver in privilege domain. Xen provides virtual firewall-router (VFR) for the network virtualization, by which, each guest domain could use virtual network interfaces (VIFs) to utilize network resource. VIF has two I/O rings for packets sending and receiving. To send a packet, guest OS enqueues a packet onto the transmission ring and then notifies the privilege domain. The back-end driver in the privilege domain will process the packet and send it out using the native driver, which resides also in the privilege domain. Similar as the sending direction, when a packet is received, Xen will check the rules for receiving and pass it to the right VIF, then put the packet buffer to the I/O ring for receiving.

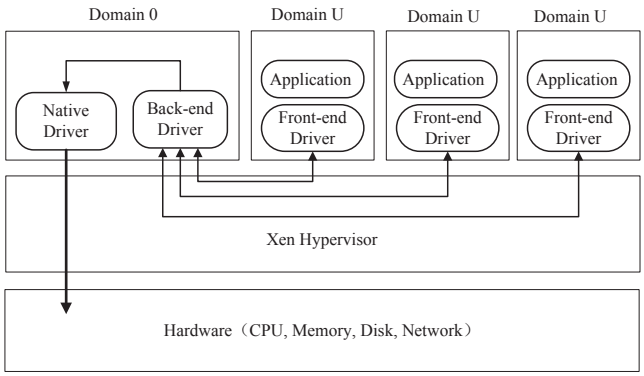


Fig. 3. Xen Split Driver Model

VIF has a credit mechanism, but it is not the same as the credit scheduler that allocates CPU resources. The VIF credit is used for bandwidth limit and traffic shaping. Each guest domain can be configured with a *credit* value through its configuration file at the time of invocation. The following line shows an example:

vif=['rate=10Mb/s, time window=50ms, bridge=xenbr0, mac=XX:XX:XX:XX:XX:XX'].

As the VIF credit value will be computed with the equation

$$VIF\_credit = rate \times (time\ window),$$

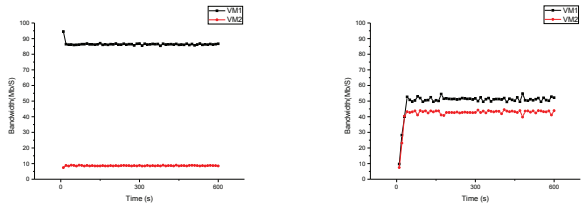
the VIF with above configuration example will have the credit value of $10Mb/s \times 50ms \approx 500 \times 10^3 bit = 62500 Bytes$, which means the size of packets sent via this VIF in any 50ms interval cannot exceed 62500 bytes. VIF's remain credit will be subtracted by the size of the packet actually sent, and will be put in sleep queue and wait next credit replenishment period if the credit value is less than zero. Pending VIF in sleep queue will be woken up when the timer expires. The time window in configure line represents the credit recruiting interval, and small time window leads to the good bandwidth shaping and flatness, but increased latency will be introduced.

Our approach discussed in Section IV relies on this VIF credit mechanism to implement the network bandwidth adjustment. In order to suppress a VM's network bandwidth provision, we reduce its VIF credit, while for the VMs that we want to help, we simply give its VIF enough credits.

VI. PERFORMANCE EVALUATION

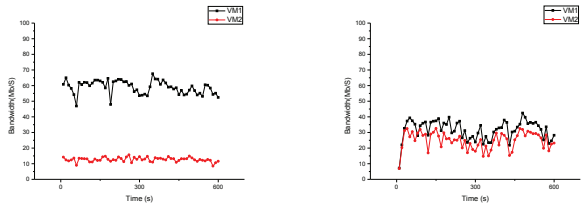
We implement our approach on Xen 4.2.1 by modifying or adding no more than 200 lines of code, Dom0 running Linux kernel 3.1.10. We create two same-weighted VMs on Xen and use another server as the client. One VM will launch multiple TCP connections and represent the "strong" VM which abuses bandwidth, and another VM only launches one TCP connection representing the "weak" VM in this bandwidth competition situation.

We employ *Iperf* benchmark to simulate the behavior of user's bandwidth usage. VM1 launches 10 TCP connections and VM2 only launches one connection, and every connection



(a) Before (b) After

Fig. 4. Bandwidth allocation before and after applying our approach in static network



(a) Before (b) After

Fig. 5. Bandwidth allocation before and after applying our approach in fluctuating network

would try its best to occupy bandwidth utmost. The other *Iperf* parameters use default value, e.g. TCP windows. Our experiment environments include cable direct connect and an actual cloud platform network. *Iperf* transmits 600 seconds and records every 10 seconds for the average bandwidth.

The direct cable connect situation before applying our approach is shown as Fig 4(a). As expected, VM1 occupies bandwidth at the same proportion as its launched TCP connection instances. At the average of the whole 600 seconds, VM1 is 86.4Mb/s and VM2 is 8.63Mb/s. This is an unfair bandwidth allocation, since the two VMs are same-weighted, which means that their owners pay the same price for renting this resource in a cloud environment. Fig 4(b) illustrates the network bandwidth share of these two VMs after employing our approach. It can be observed that our approach is effective in maintaining the fairness on bandwidth share of these two VMs.

In order to validate our algorithm could achieve fairness guarantee even in the fluctuating real-world network environment, we place the clients that connect the VMs at the public network and thus simulate the fluctuating network environment. Fig 5 illustrates the bandwidth of the VMs under experiment before and after employing our approach. It can be observed that our approach can effectively improve the fairness share of network bandwidth in such scenario.

VII. Conclusion

In this paper, we present an approach that improves fairness of network bandwidth allocation in cloud environment. Our approach can handle the situation that happens in the fluctuating network environment and thus has its merits for real usage cases. Experimental results indicate our method could distinctly improve the virtual domain bandwidth fairness, not only the static network but also the dynamic network environment. However, from the experiment results, it can be observed that there are still jitters that exist in the bandwidth curves, although they are relatively small. We believe a better algorithm or more appropriate parameters can reduce the jitters, and place this in our future work.

Acknowledgments

This work is supported by National Science Foundation of China under grant No.61232008.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Institute, Tech. Rep., June 2011.

[3] Amazon, "AWS—Amazon Elastic Compute Cloud (EC2)," Available at: http://aws.amazon.com/ec2/.

[4] Google, "Google apps for business," Available at: http://www.google.com/enterprise/apps/business/.

[5] M. F. Bari, R. Boutaba, R. P. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey." *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[6] X. Liao, H. Li, H. Jin, H. Hou, Y. Jiang, and H. Liu, "Vmstore: Distributed storage system for multiple virtual machines," *Science China Information Sciences*, vol. 54, no. 6, pp. 1104–1118, 2011.

[7] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 2, pp. 42–51, 2007.

[8] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94, 1994, pp. 1–1.

[9] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10, 2010, pp. 1–1.

[10] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proceedings of the 3rd Conference on I/O Virtualization*, ser. WIOV'11, 2011, pp. 6–6.

[11] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, *NetShare: Virtualizing data center networks across services*. Department of Computer Science and Engineering, University of California, San Diego, 2010.

[12] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (CoNEXT 10)*, November 2010.

[13] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.

[14] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.