applied sciences

MDPI

Article

Nxt-Freedom: Considering VDC-Based Fairness in Enforcing Bandwidth Guarantees in Cloud Datacenter †

Shuo Wang, Zhiqiang Zhou, Hongjie Zhang and Jing Li *

Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230052, China; shuowang@mail.ustc.edu.cn (S.W.); zzq0902@mail.ustc.edu.cn (Z.Z.); zhanghongjie@mail.ustc.edu.cn (H.Z.)

* Correspondence: lj@ustc.edu.cn

† This paper is an extended version of our paper published in Wang, S.; Li, J.; Zhang, H.; Wang, Q. Nxt-Freedom: Considering VDC-based Fairness in Enforcing Bandwidth Guarantees in Cloud Datacenter. In Proceedings of the 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), Chongqing, China, 11–14 October 2018; pp. 1–6..

Received: 28 September 2020; Accepted: 3 November 2020; Published: 6 November 2020

Abstract: In the cloud datacenter, for the multi-tenant model, network resources should be fairly allocated among VDCs (virtual datacenters). Conventionally, the allocation of cloud network resources is on a best-effort basis, so the specific information of network resource allocation is unclear. Previous research has either aimed to provide minimum bandwidth guarantee, or focused on realizing work conservation according to the VM-to-VM (virtual machine to virtual machine) flow policy or per-source policy, or both policies. However, they failed to consider allocating redundant bandwidth among VDCs in a fair way. This paper presents a bandwidth that guarantees enforcement framework NXT-Freedom, and this framework allocates the network resources on the basis of per-VDC fairness, which can achieve work conservation. In order to guarantee per-VDC fair allocation, a hierarchical max–min fairness algorithm is put forward in this paper. In order to ensure that the framework can be applied to non-congestion-free network core and achieve scalability, NXT-Freedom decouples the computation of per-VDC allocation from the execution of allocation, but it brings some CPU overheads resulting from bandwidth enforcement. We observe that there is no need to enforce the non-blocking virtual network. Leveraging this observation, we distinguish the virtual network type of VDC to eliminate part of the CPU overheads. The evaluation results of a prototype prove that NXT-Freedom can achieve the isolation of per-VDC performance, which also shows fast adaption to flow variation in cloud datacenter.

Keywords: cloud datacenter; cloud networks; per-VDC fairness; bandwidth guarantees

1. Introduction

As cloud computing has become more mature, many companies start to put their own applications in the cloud datacenters. These companies are called the VDCs, and they establish a multi-tenant environment. Whether the resources can be efficiently shared and multiplexed across these VDCs is critical to cloud computing [1–3]. However, the resources of cloud networks are generally shared in a best-effort way, as a result of which neither tenants nor cloud providers are clear as how the network resources are allocated. The development of network virtualization and commodities of scale shows the trend of building cloud datacenters consisting of millions of virtual end points [4,5]. If the VM scheduling is poor, diverse traffic from VDCs may converge on the same physical link, which will affect the application performance of VDCs with fixed bandwidth requirements, and further

reduce the utilization rate of entire underlying networks. Recent designs of the cloud datacenter network achieve horizontal scaling of hosts based on the path multiplicity [6–10]. There are many paths between all pairs of hosts. A simple and intuitive idea is to forward flows along these paths simultaneously and dynamically to increase network utilization [11] instead of per-flow static hashing [12], despite achieving effects, to some extent, it does not make performance isolation between VMs from different VDCs.

In the cloud datacenter, there is lack of coordination and mutual trust among tenants. They all wish to occupy more bandwidth, thus leading to contention for resources. Therefore, efforts should be made to insulate these tenants. For a VM X, the allocation of network resources not only relies on the VMs running on the same machine with X, but is also affected by other VMs that X communicates with and the cross-traffic on each link used by X. After the quantity of VMs in the cloud datacenter has increased to a certain level, it will be difficult to guarantee bandwidth in a centralized way. In some previous works [13–16] , the independently operating congestion control mechanisms based on hypervisor are employed. In such mechanisms, complex coordination between the hypervisors or with a central controller is not required, but in these works, the work-conserving bandwidth guarantee is realized based on the VM-to-VM flows, while VDC-based fairness is ignored. For example, as Figure 1 describes, on the same link, across which VDC-A has three VM-to-VM flows and VDC-B has one VM-to-VM flow, we assume all these flows are of TCP types. Eventually VDC-A gets 750 Mb/s and VDC-B gets 250 Mb/s, but this is not the ideal circumstance. It is rational that the link bandwidth is allocated among the VDCs in an even manner, that is to say, no matter how many flows that VDC-A has, it will get 500 Mb/s, and so does VDC-B.

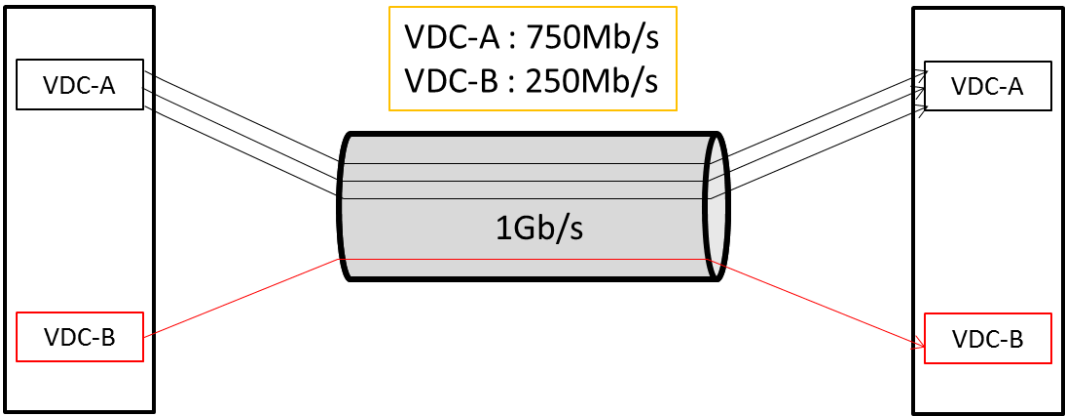

Figure 1. Unfair bandwidth allocation among VDCs.

One optimistic situation can be expected for the reasonable delivery of network resources in the cloud datacenter, where the tenants can be grouped into two categories. In the first category, the tenants buy bandwidth either for their VMs or for their services, which is used as the minimum guarantee of bandwidth, and it is called the QoS-VDC (similar to resource reservation). When there is contention for bandwidth, the minimum bandwidth can be guaranteed for this category. In addition, it is argued that, to ensure fair allocation of resources, the link should be used to allocate redundant link bandwidth allocated among VDCs evenly. In this way, it may take less time to implement the VDC tasks. As a result, more VDCs are accepted, and the profit can be increased. The second category involves the demands of tenants with no guarantees of bandwidth, which is called the NQoS-VDC. For this category, the minimum bandwidth guarantee has no value, and the allocated bandwidth will change with the number of tenants who share the redundant bandwidth.

To realize the optimistic situation mentioned above, one brand new pattern of bandwidth guarantees should be provided for cloud datacenters, which should achieve:

- Providing minimum bandwidth guarantees: A minimum absolute bandwidth is guaranteed for each VM, which is used to send/receive traffic.
- Work-conserving: Unused bandwidths can be re-used by other tenants in a way that does not hurt anyone's guaranteed bandwidths. For a given flow, if a link L is the bottleneck link, then L should be put into full utilization.
- Fair redundant bandwidth allocation: Tenants should share un-used bandwidth equally. To one link, minus the minimum bandwidth guarantees for VDCs, the remaining bandwidth should be allocated in a fair way.

In such an environment, to provide a bandwidth guarantee for VDCs brings some challenges, mainly consisting of two parts: One is for achieving scalability. The hypervisor-based congestion control mechanism is indispensable, which also brings certain CPU overheads. We argue that this cost should be as low as possible in that the CPU resources are rare in virtualization. The other is for providing the work-conserving bandwidth guarantees on the premise of VDC-based fairness. Every flow is tagged with its VDC label, and the flows belonging to the same VDC should be under the constraint such as the actual bandwidth allocated to the VDC. Then, the link redundant bandwidth is dynamically allocated among the VDCs on demand. To make all the flows perform in best efforts, it entails a complex and troublesome algorithm to control the bandwidth allocation for each active flow and each VDC in runtime. This means we should know not only the concrete bandwidth demand of every flow, but also that of every VDC. This is more difficult to achieve than the work-conserving bandwidth guarantees, merely according to the VM-to-VM flow.

This paper presents NXT-Freedom, which is a novel architecture used to enforce the fair allocation of bandwidth in cloud datacenters based on VDC. By decoupling its solutions, it can simultaneously achieve VDC-based fair bandwidth guarantees and work-conservation by a centralized bandwidth allocator (BA), and the per-VDC congestion is controlled by distributed bandwidth enforcers (BE) running in the hypervisor. When there is requirement for a virtual network (VN), BA will map it to the physical network (PN) according to its customized VN structure. In the course of mapping VN to PN, appropriate paths as the mapping results are chosen for carrying the flows generated from the VN, depending on which the incoming and outgoing bandwidth of the VMs are stipulated. However, traffic congestion at the destination VM still exists, even if the sending and receiving rates are restricted. If the VN itself is not of type non-blocking, we argue that congestion is inevitable. Hence, BE is employed to manage congestion control to avoid network performance degradation.

Our contributions are as follows:

1. Enforcing bandwidth guarantees of VDCs based on a more general and flexible VN model compared to the hose model.
2. An VDC-based fair bandwidth allocation algorithm, making that each flow of the VDC could get network resources on demand in best efforts. Eventually, it achieves high network utilization while ensuring VDC-based fairness.
3. By distinguishing the non-blocking VN from the blocking VN, the CPU overheads of distributed bandwidth enforcers could be reduced to some extent.
4. A distributed prototype design and implementation of NXT-Freedom with good scalability, making it practical in cloud datacenters.

The remainder of the paper is organized as follows. Section 2 presents related work. We describe our VDC network model and the main bandwidth guarantee methodology in Section 3, and Section 4 presents overview of the architecture of NXT-Freedom proposd in this paper. In Section 5, the hierarchical max-min fairness algorithm for guarantee of per-VDC fairness is introduced in detail, which also supports work-conservation. The mechanism of the distributed BE is presented in Section 6. Evaluation is conducted in Section 7 to analyze the experimental results of NXT-Freedom. Section 8 draws the conclusions of our work.

2. Related Work

Recently, there have been many works on how to enforce bandwidth guarantees in the cloud datacenter [17–19]. In accordance with the network characteristics of cloud datacenters, we classify these works into the following two categories: (1) In some studies such as Gatekeeper [20] and EyeQ [21], it is assumed that there is no congestion in the network core, and congestion only occurs in the endpoints of the server. Even though the path multiplicity technology (such as ECMP [22]) has been broadly used in the current datacenter, because of improper scheduling of VM flows, the problem of link congestion is still unsolved [11]. In addition, a central controller is employed for scheduling in [11], but scale-out is difficult to achieve. Thus, these solutions are impractical in reality. (2) In some other works, the authors do consider the possibility of congestion in the network core, such as SecondNet [13], Oktopus [14], Faircloud [15], Seawall [16], ElasticSwitch [23], NetShare [24,25] and BwShare [26]. In [13,14], the heuristic bandwidth allocation algorithms are designed and employed to map the virtual network to the physical network. In these two works, consideration was made to enforce the allocation of bandwidth, but they failed to consider work-conserving, and the utilization rate of underlying network is relatively low. In [15], the tradeoffs are achieved for sharing the resources of cloud networks based on various desirable properties and allocation policies. It manages to achieve both bandwidth guarantee and work conservation, but it also has some defects that cannot be ignored, and, for example, expensive hardware support and special network topology are required in order to implement bandwidth guarantee. In [16], by providing clear feedback from the receivers, traffic will be throttled at the sources before they use any network resources, which can prevent a malicious VM from hogging bandwidth in the network. This method is particularly efficient for the situation in which many VMs send traffic to one receiver, and compared to the case in which a VM of a different tenant on the same server with the same weight is receiving traffic from just one sender, it will occupy a significantly higher fraction of received bandwidth of the server link. This problem of unfairness is not solved in [23,26] either. In [23], redundant capacity is shared among active VM-to-VM flows in proportion, but minimum bandwidth guarantee is provided due to the hypothesis that the virtual network is built on the hose model. Based on that, the excess bandwidth is allocated in a dynamic manner. The authors in [26] also enable transparent work conservation across all VMs, which allows VMs to reuse idled bandwidths across the cloud efficiently without hurting bandwidth guarantees. The authors in [24] ensure per-tenant fair sharing of congested links, and as more tenants use the link, the per-tenant sharing can be arbitrarily reduced. Ali et al. [25] considered configurable fairness requirements of tenants, but the flow could be split into subflows.

Compared to works mentioned above, our work focuses on the enforcement of VDC-based fair bandwidth guarantees. It can provide the minimum bandwidth guarantee based on our Switch Centered Graph (SCG) model, compatible with the hose model. Our method not only emphasizes providing bandwidth guarantees with work-conserving property, but also ensures the VDC-based fair allocation of resources.

3. VN Allocation and Enforcement Methodology

Various companies generally carry their applications in a traditional system consisting of a fixed number of servers and network equipment connected to the cloud datacenters, so as to reduce the expenditure of hardware purchase and software upgrade. However, tenants may have different requirement for the CPU, memory and storage resources of VMs, and they may also need varying internal bandwidth resource of network. Tenants' unwillingness to reveal their network requirements does not help either tenants or providers. If the tenants continue to require performance isolation, the network is responsible of providing bandwidth guarantees for VDCs. Besides, tenants expect that their networking service can be customized in VDCs to achieve better application performance. Hence, we design a brand new VN model, which can be used by the tenants to specify their network requirements. The VN structure is introduced in detail in Sections 3.1 and 3.2. The VNs for tenant application are provided to the cloud first, and then the cloud deploys VMs onto servers and logic

switches onto physical switches. In this way, a cloud networking environment very similar to the actual enterprise environment of tenants can be provided for their application, which makes it more possible to preserve similar performance. In our work, we highlight the bandwidth guarantees enforcement based on this VN model, not the VN-to-PN mapping process.

3.1. The VN Model

In general, the cloud providers wish to provide a standardized template to tenants, which can be used by the tenants to describe their specific requirements for VN. Recently, different VN models have been proposed for bandwidth allocation, such as the traffic matrix [13], the hose model with invariable bandwidth [14], the hose model with time-varying bandwidth [27], and the TAG model [28] based on the structure of application communication. Specific VN-to-PN mapping algorithms can be designed using these models. In this paper, SCG is presented, which is a new model that can be used by the tenants to specify their fine-grained network requirements for VDCs, including the network structure, uplink bandwidth, and downlink bandwidth. This is different from the hose abstraction, in which the models are presented based on their resemblance with typical physical networks. It is not like the TAG abstraction either, because in the TAG abstraction, the actual communication patterns of applications are modeled, while the SCG abstraction can simulate the prior network environments of tenants where these applications are actually run. By utilizing the tenant's knowledge of network structure and bandwidth demand, the SCG model is able to represent the actual networks in a concise and flexible manner. This covers the common network topologies in practice.

As for most of applications, the multi-tier tree topology is the de-facto underlying network structure. Hence, we construct the SCG model, where each vertex stands for an entity (which could be a switch or a VM) to mimic the actual network structure of application. A simple VN modeled with SCG is presented in Figure 2, which has the oversubscribed multi-tier network architecture. In this VN, there are two bidirectional edges between the core layer and the aggregation layer, which are labeled as $\langle B1, B2\rangle$ and $\langle B3, B4\rangle$, respectively. The edges between the aggregation layer and the access layer are labeled in the same way. By controlling the uplink traffic and downlink traffic, the tenants can set appropriate sending rate and receiving rate for each VM in VDC, which can reduce the cost generated by excessive bandwidth.

SCG is a generalized model that can represent almost all of applications. Our work focuses on enforcing VDC-based bandwidth guarantees using this model.

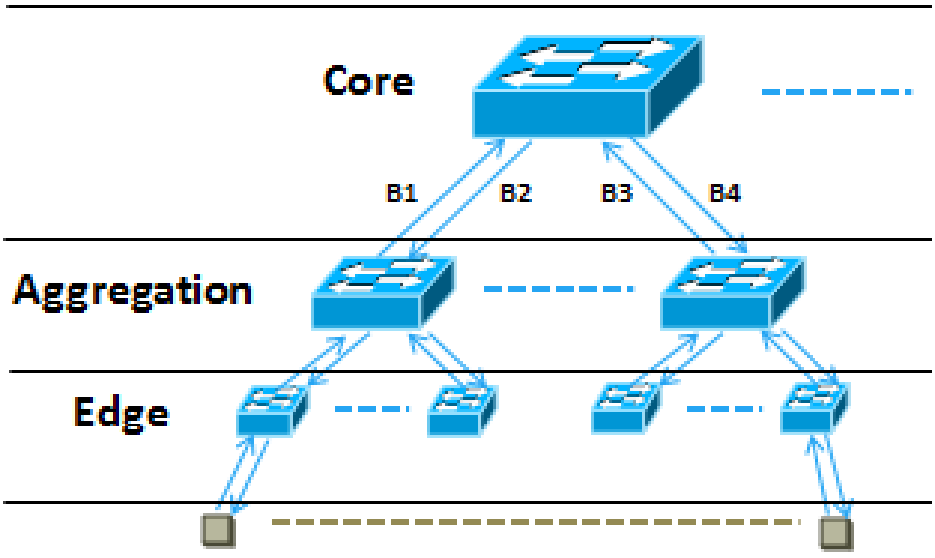

Figure 2. An SCG modeled VN.

3.2. Model Formulation

In this work, the physical network of datacenter is modeled in the form of a graph $PG = (S, M, E)$, in which S represents the set of all switches, M is the set of physical servers, and E stands for the set of all links in the network. A path from node u to node v is an ordered sequence $e_1, e_2, \ldots, e_l$ of distinct links where e1 and el have their respective end nodes at u and v and the adjacent links $\{e_{j-1}, e_j\}$ have a common distinct end node w where $w \neq u \neq v$. We denote a path between the nodes u, v as $p_{u,v}$ and the set of all paths between them as $P_{u,v}$. Each link $e(u, v) \in E$ has a capacity $C(e)$, representing the maximum bandwidth it can support.

Next, the VN will be modeled. Let $VN = (VS, \overline{VM}, VE)$ be a set of elements, which contains the logical switches, logical servers, and logical links. The VN should be mapped to PG for sharing the resources in the physical network of datacenter. Therefore, VS, $\overline{VM}$ and VE is mapped to S, M and E, respectively. The notations used in this paper are listed in Table 1. A switch S_i has k_i ($k_i \geq 1$) ports $\left\{ port_j^{S_i} \mid j \in [0, k_i - 1] \right\}$, k_i is constant. In the VDC, the switch in SCG model should be consistent with the real switch. Because the commodity switches are used as the underlying interconnecting elements of network infrastructure in the cloud datacenters, the native resources of switch in the SCG model should not exceed the resources of commodity switch in reality, such as the number of ports and port bandwidth. This means $interface(VS_i) < interface(S_j)$, $vib(VS_{i,m}) < ib(S_{j,n})$, and $veb(VS_{i,m}) < eb(S_{j,n})$. Tenants request bandwidth guarantees between entities by placing bidirectional edges between the corresponding vertices in the SCG model. Each edge $e = (VS_{i,m}, VS_{j,n})$ between the *m-th* port of VS_i and the *n-th* port of VS_j can be labeled with a heterogeneous weight pair $\langle u, v \rangle$ that represents per-link bandwidth guarantees. Specifically, VM connected to the *m-th* port of VS_i is guaranteed bandwidth $u = veb(VS_{i,m})$ for sending traffic to VM connected to the *n-th* port of VS_j, is another way to say, VM connected to the *n-th* port of VS_j is guaranteed bandwidth $u = vib(VS_{j,n})$ to receive traffic from VM connected to the $m - th$ port of VS_i. VM connected to the *n-th* port of VS_j is guaranteed bandwidth $v = veb(VS_{j,n})$ for sending traffic to VM connected to the *m-th* port of VS_i, is another way to say, VM connected to the *m-th* port of VS_i is guaranteed bandwidth $v = vib(VS_{i,m})$ to receive traffic from VM connected to the *n-th* port of VS_j. Obviously, we have $veb(VS_{i,m}) = vib(VS_{j,n})$ and $veb(VS_{j,n}) = vib(VS_{i,m})$.

Table 1. Notations.

Notations	Description
$S_{i,j}$	The *jth* port of *ith* physical switch S
$VS_{i,j}$	The *jthport* of *ith* virtual switch VS
$interface(S_i)$	The port number of S_i
$interface(VS_i)$	The port number of VS_i
$ib(S_{i,j})$	Physical ingress bandwidth of $S_{i,j}$
$eb(S_{i,j})$	Physical egress bandwidth of $S_{i,j}$
$rib(S_{i,j})$	Residual ingress bandwidth of $S_{i,j}$
$reb(S_{i,j})$	Residual egress bandwidth of $S_{i,j}$
$vib(VS_{i,j})$	Ingress bandwidth requirement of $VS_{i,j}$
$veb(VS_{i,j})$	Egress bandwidth requirement of $VS_{i,j}$
$rvib(VS_{i,j})$	Residual ingress bandwidth of $VS_{i,j}$
$rveb(VS_{i,j})$	Residual egress bandwidth of $VS_{i,j}$

3.3. Virtual Network Classification

Using the SCG model, the VN is constructed. The customized VN is classified as either blocking or non-blocking network. Let vm_i^r and vm_i^s denote the maximum receive rate and send rate of the i-th VM in the $\overline{VM}$ set. We say that the VN is non-blocking, including two aspects: endpoint link is non-blocking and network fabric is non-blocking. To VN, the endpoint link is the access link that

connects VM to the logic switch, and the network fabric is the network apart from the part that the access link connects. Otherwise this is the blocking network (blocking VN).

1. The endpoint link non-blocking: For arbitrary VM in the VN, it can accept the traffic from all other VMs at their permitted maximum sending rate simultaneously. We describe as follows:

$$\sum_{i \neq j} vm_j^s \leq vm_i^r, \forall vm_i \in \overline{VM} \tag{1}$$

2. The network fabric non-blocking:

$$\begin{cases} \sum\limits_{vm_i}^{\overline{VM}} iveb(p_{vm_j \in \overline{VM}, vm_i}, ve_1)[i \neq j] \leq iveb(ve_1) \\ \sum\limits_{vm_i}^{\overline{VM}} iveb(p_{vm_j \in \overline{VM}, vm_i}, ve_2)[i \neq j] \leq iveb(ve_2) \\ \qquad \cdots \\ \sum\limits_{vm_i}^{\overline{VM}} iveb(p_{vm_j \in \overline{VM}, vm_i}, ve_l)[i \neq j] \leq iveb(ve_l) \end{cases} \tag{2}$$

3.4. Network Blocking Observation

In an IaaS-based cloud, physical machines served as compute resources on which VMs run are connected through network fabric, as Figure 3 describes. While it is true that the VMs on the same physical server will compete for network resources due to the multiplexing feature of the virtualization resulting in the endpoint link congestion, it is also true that the network fabric may block when two VMs from different physical machines communicate due to improper flow scheduling. The bottleneck where congestion takes place is either the network fabric, or the endpoint links that connect each physical server to the network fabric. For any VN that has mapped onto PN, our key observation is that it is indispensable for blocking VN to have congestion control functioned to regulate VM behavior, whereas it is unnecessary for non-blocking VN as there will not be traffic congestion on VN.

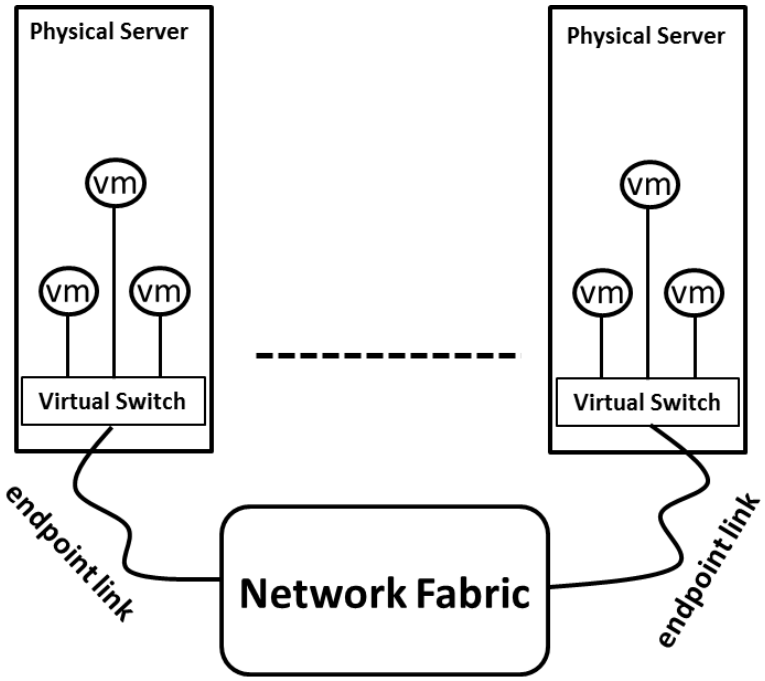

Figure 3. The IaaS-based cloud network.

3.5. Fair Allocation and Enforcement Methodology

3.5.1. Fair Allocation Methodology

A VN should be delivered and mapped to the underlying physical network first. The redundant bandwidth of a link should be allocated to one VDC according to the number of tenants using the link and the total excess bandwidth which this link can provide. For simplification, the influence of the flow direction is not considered. By using

$$map(ve, e) = \begin{cases} 1, \text{if } ve \text{ is mapped to } e \\ 0, \text{otherwise} \end{cases} \tag{3}$$

whether the logic link is mapped to the physical link can be determined. To obtain the number of tenants on the same link, we have:

$$tenantsOnEdge(e) = \sum_{VN} [\sum_{ve \in VE} [map(ve, e)] \geq 1] \tag{4}$$

The total redundant bandwidth on the link can be allocated according to:

$$R(e) = C(e) - \sum_{VN} \sum_{ve \in VE} C(ve)[map(ve, e)] \tag{5}$$

Thus, we obtain the per-VDC fair allocation of redundant bandwidth:

$$F(e) = \frac{R(e)}{tenantsOnEdge(e)} \tag{6}$$

To a path $p_{u,v} = \{e_1, e_2, \ldots, e_l\}$, the most effective blocking link determines whether the bandwidth should be allocated. Hence we have:

$$allocatedBandwidth(p_{u,v}) = \min_{p_{u,v}} \{F(e)[e \in p_{u,v}]\} \tag{7}$$

Our fairness embodies the even bandwidth allocation in the hierarchy of layers. The top layer is the even bandwidth allocation on each link among tenants, one situation should be noted, in which several VM-to-VM flows belonging to one VDC overlap on the same link, and we argue that the bandwidth allocated to these VM-to-VM flows should not be more than that of other VDCs with less VM-to-VM flows. The bottom layer is that the allocated bandwidth is evenly allocated among the VM-to-VM flows belong to one tenant. In Figure 4, tenant A and tenant B will get $0.5X$ of the bandwidth each—tenant A has three flows, thus each VM-to-VM flow of tenant A gets $0.13X$ of the bandwidth in average. Let $f_{u,v}$ be the flow from u to v, we define:

$$mapflow(f_{u,v}, e) = \begin{cases} 1, \text{ if } f_{u,v} \text{is mapped to } e \\ 0, \text{ otherwise} \end{cases} \tag{8}$$

Thus, we have $n_{VDC}(e)$ representing the total flows belong to one tenant on the link:

$$n_{VDC}(e) = \sum_{(u,v \in VM) \wedge (u \neq v)} mapflow(f_{u,v}, e) \tag{9}$$

Therefore, the bandwidth $b_{f_{u,v}}$ to each VM-to-VM flow gets:

$$b_{f_{u,v}} = \frac{allocatedBandwidth(p_{u,v})}{n_{VDC}(e)} \tag{10}$$

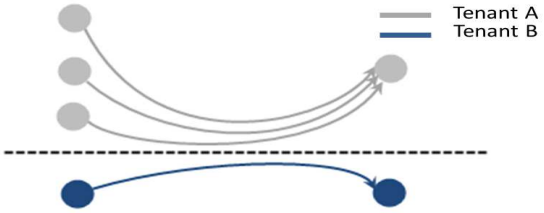

Figure 4. Fairness in the cloud based on tenants.

3.5.2. Fair Allocation Enforcement Methodology

After the VN is mapped to the underlying physical network and the redundant bandwidth is fairly allocated among tenants, we enforce these bandwidth guarantees. Non-blocking VN only requires the bandwidth guarantee enforcement at the senders, so congestion will not take place anywhere. For blocking VN, congestion will occur at the endpoint link or the network fabric—in this case, we take measures to control the flow rate of each sender according to the feedback of the receiver. At the receiver, we monitor and detect the congestion, then calculate the reasonable traffic sending rate for senders to avoid congestion, afterwards we send feedback to the senders, and senders limit the sending rate based on the received feedback.

4. System Architecture

We intend to employ NXT-Freedom to enforce bandwidth guarantees for VDCs fairly. It mainly has two objectives: one objective is to allocate the demanded bandwidth to each VDC dynamically and fairly; the other one is to ensure performance isolation among VDCs during enforcement of allocated bandwidth. To achieve those two objectives, we employ two modules, as shown in Figure 5. One module is a centralized bandwidth allocator module, and the other is a distributed bandwidth enforcer module running on the hypervisor of each compute server.

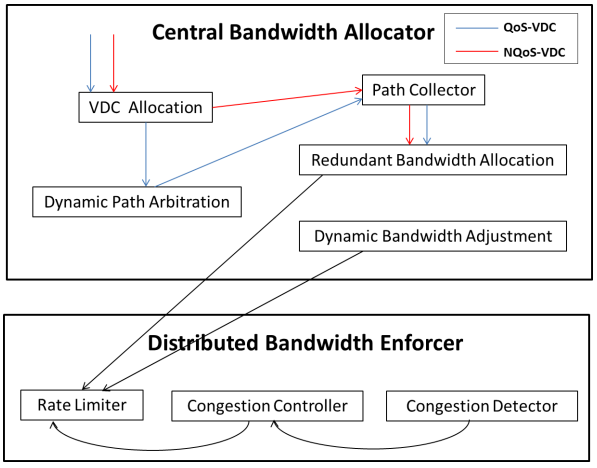

Figure 5. The system architecture of Nxt-Freedom.

4.1. The Workflow of Central Bandwidth Allocator

During the working process of central bandwidth allocator, a tenant submits his VDC requirement first. Then, this request is delivered to the Allocation component of VDC, and this component decides whether to map VN to PN based on the VN type. If the VDC requirement is of the QoS-VDC type, the VDC Allocation component will generate virtual links for mapping

to the physical links, and these virtual links are eventually sent to the Path Collector component. On the other hand, if the VDC requirement is of the NQoS-VDC type, it is unnecessary for the VDC Allocation module to generate the mapping results, because the minimum bandwidth guarantee of NQoS-VDC is zero. In this case, the Dynamic Path Arbitration component is used to generate these mapping results instead, and suitable paths are determined in accordance with the physical network's current workload status, which are then sent to the Path Collector component. In the final step, the Redundant Bandwidth Allocator component queries the mapping results from the Path Collector periodically, and decides how to allocate the redundant bandwidth. However, some VDCs may not use all the bandwidth they are allocated with. To address this problem, the independent Dynamic Bandwidth Adjustment (DBA) component is employed to detect idle tenants with spare bandwidth, and the spare bandwidth will be allocated among VDCs of the idle tenants first; if there is still redundant bandwidth, it will be allocated to other tenants based on the max–min fairness principle.

4.2. The Workflow of Distributed Bandwidth Enforcer

The distributed bandwidth enforcer resides in the hypervisor of every compute server, and it is composed of the following components: the Rate Limiter component, the Congestion Controller component and the Congestion Detector component. Together, these components provide the isolation of network performances among tenants and congestion control for each tenant. Furthermore, each component also has its own specific function. The Rate Limiter component can limit the sending rate of senders. The Congestion Detector component can be used to detect the packet losses, based on which, it can determine whether there is any congestion at the receivers. The Congestion Controller component decides a proper rate of every flow sent by the senders. For the non-blocking VN, the congestion can be prevented by limiting the sending rate of each sender as allocated, and in this case, the bandwidth guarantees are enforced. On the other hand, for blocking VN, congestion may occur anywhere in the network due to the unpredictable traffic behavior of the tenant. It is impossible to achieve performance isolation only by limiting the sending rate of each sender. For the purpose of preventing congestion and improving network efficiency, the Congestion Detector component at the receiver is used to detect packet loss first. Then, the Congestion Controller component analyzes and calculates the proper rate for all senders, and the result is used as the feedback. In the meantime, this feedback is delivered to the corresponding senders. At last, the senders adjust their sending rate according to the feedback.

5. Dynamic Bandwidth Allocation

In this work, the centralized DBA component is used to provide fair bandwidth guarantees based on VDC. The redundant bandwidth of a link onto which VDCs are scheduled should be allocated among these VDCs fairly. In order to achieve work-conservation, the bandwidth allocated for each VDC should be reallocated in a timely manner according to its actual bandwidth demands on the narrowest link. The allocation mechanism is introduced in detail in this section.

5.1. Initial VN to PN Allocation

For one VN modeled with SCG, the VDC Allocation component of NXT-Freedom searches for suitable positions to accommodate the VMs and links of this VN. After the VN is mapped onto PN, the upper limit of the sending/receiving rate for each VM will be set according to the redundant bandwidth of every access link that the VN is mapped to, and the intermediate redundant link bandwidth will be calculated. As shown in Figure 6a, two VDCs have their specific VN demands. Suppose they are mapped to the PN in Figure 6b, and each link has the capacity of 100. Because the communication behavior within the VDC is not sufficiently known, it is difficult to determine where the narrowest link is. To prevent VMs from sending more traffic, the bandwidth of access link is used

as the initial rate of each VM. Eventually, VM-1 gets 55, VM-3 gets 45, VM-2 and VM-5 get 50 each, and VM-4 gets 100.

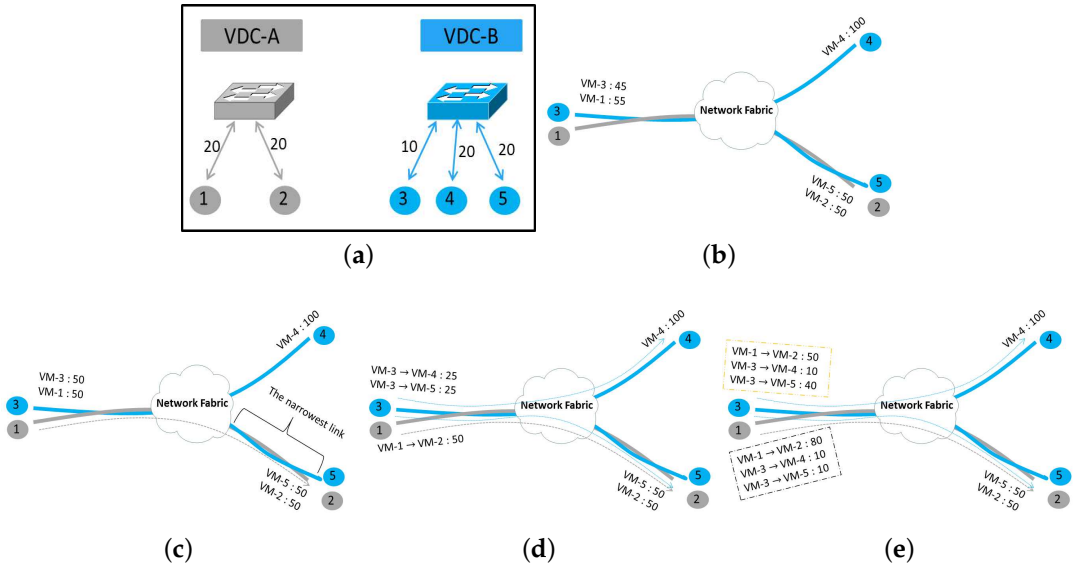

Figure 6. The bandwidth allocation mechanism. (**a**) The VN demands; (**b**) Initial allocation; (**c**) Rate stipulation (1 flow); (**d**) Rate stipulation (3 flows); (**e**) Dynamic flow rate adjustment.

5.2. Rate Stipulation for the Active Flows

When active VM-to-VM flows are generated by the VDCs, the traffic rates should be stipulated for active VM-to-VM flows. The narrowest link can be anywhere in PN, and the traffic rates of VM-to-VM flows from the same source could be different because of various bottlenecks existing in the network. To ensure fair competition and avoid congestion among these flows, first of all, we pick up the narrowest link by counting active VM-to-VM flows; then, we further stipulate the sending and receiving rates for each flow. Based on the assumptions that there is a flow from VM-1 to VM-2 and that the network bottlenecks can only occur at the access link, the initial rates of VM-1 and VM-2 are 55 and 50, respectively, and the narrowest link where congestion occurs is the access link, as shown in Figure 6c. Therefore, by setting the bandwidth of VM-1 as 50, the excess bandwidth is added to VM-3, thus VM-3 gets 50. The narrowest link is determined according to the number of VM-to-VM flows belonging to the same VDC. If there are two active flows from VM-3 to VM-4 and to VM-5, respectively, then the narrowest link bandwidth for each VM-to-VM flow is 25.

5.3. Dynamic Bandwidth Allocation Adjustment

A VDC may have more than one flow on any link. Conventionally, bandwidth is allocated among VDCs in a best-effort way, and more flows indicate more bandwidth. Thus, the VDC with less flows will not be allocated with its fair share of bandwidth. To prevent such unfair allocation of bandwidth, the dynamic bandwidth allocation should satisfy:

1. If all VDCs can fully use the bandwidth allocated to them, the redundant bandwidth of link should be allocated among these VDCs evenly. If all flows of a VDC can fully up the allocated bandwidth, the allocated bandwidth should be allocated among these flows evenly. As Figure 6d shows, one flow of VDC-A is allocated with half of the link bandwidth, while the two flows of VDC-B get another half in total.
2. If the bandwidth of any flow within a VDC is lower than its allocated bandwidth, the spare bandwidth should be allocated to other flows within the same VDC before being allocated to the flows in other VDCs. In the orange part, as shown in Figure 6e, within VDC-B, the flow

from VM-3 to VM-4 is reduced to 10, as a result of which, the flow from VM-3 to VM-5 can grow to 40. This kind of bandwidth allocation within the same VDC is called the intra-VDC bandwidth allocation.

3. If all flows within a VDC fail to fully use the allocated bandwidth, the spare bandwidth can be allocated to other VDCs evenly, and these VDCs will further allocate their share of spare bandwidth among the flows within them. This kind of allocation is called the inter-VDC bandwidth allocation. As shown by the gray part in Figure 6e, if two flows of VDC-B only require a flow of 20 of the bandwidth in total, the rest of the bandwidth can be assigned to VDC-A. As a result, the flow from VM-1 to VM-2 becomes 80.

By combining the situations discussed above, we present a hierarchical max–min fairness Algorithm 1 to ensure VDC-based fair allocation of redundant bandwidth. It mainly consists of two layers—the dynamic intra-VDC bandwidth allocation and the dynamic inter-VDC bandwidth allocation in a max–min fair way. It should be noted that intra-VDC bandwidth allocation takes precedence over the inter-VDC bandwidth allocation. In each stage, we collect the traffic of all VM-to-VM flows on each link first, and then calculate the traffic of the VM-to-VM flows of the same tenant and the number of tenants sharing this link, and we further get the available bandwidth that each tenant can be allocated with.

Algorithm 1: Hierarchical max–min fairness.

let $s_{t,p,f}$ denote the sending traffic rate of flow f should be limited at the sender for tenant t on path p;
let $r_{t,p,f}$ denote the receiving traffic rate of flow f should be limited at the receiver for tenant t on path p;
let $b_{t,l}$ denote the bandwidth allocated to tenant t on link l;
let C_l denote the capacity of link l;
$g_{t,l} \Leftarrow$ the bandwidth guarantee on demand on l of tenant t;
$p \Leftarrow$ get all the mapped paths from Path Collector;
collect(){
    collect VM-to-VM flows traversing on each link;
}
periodically(){ **for** *a path p in P* **do**
a link $l \in p$ collect()
$t_l \Leftarrow$ the tenants set on link l;
$nt_l \Leftarrow$ total tenants on link l;
$f_{t,l} \Leftarrow$ the VM-to-VM flows set belong to tenant t on link l;
$nf_{t,l} \Leftarrow$ total VM-to-VM flows be tenant on link l;
$b_{t,l} \Leftarrow \frac{C_l - \sum_{t\ on\ l} g_{t,l}}{nt_l} + g_{t,l}$;
$r_{t,p,f} \Leftarrow \frac{\min_{l \in p} b_{t,l}}{nf_{t,l}}$;
$s_{t,p,f} \Leftarrow r_{t,p,f}$;
}
$b_{t,l}^{shadow} \Leftarrow$ max-min-allocation$(f_{t,l}, b_{t,l}^{shadow})$;
for *tenant t in* t_l **do**
$r_{t,p,f}^{shadow} \Leftarrow$max-min-allocation$(f_{t,l}, b_{t,l}^{shadow})$;
set $r_{t,p,f} \Leftarrow r_{t,p,f}^{shadow}$;
if *flow f is untagged* **then**
set $s_{t,p,f} \Leftarrow r_{t,p,f}$;
else
$k \Leftarrow$ count the active flows having the same destination with the flow f;
set $s_{t,p,f} \Leftarrow \frac{r_{t,p,f}}{k}$;

5.4. VN Type Variation

In the former section, we classify the VN as either blocking or non-blocking network. Regarding non-blocking VN, its endpoints are unnecessary to be attached with functioning congestion control, thereby reducing the CPU overhead of the physical server where VMs reside. However, it is very likely that the type of VN can convert from non-blocking to blocking during the dynamic redundant bandwidth allocation. Though we can take measures to maintain the non-blocking characteristic when allocating redundant bandwidth, it would be contradictory to the VDC-based fairness.

Fortunately, it is also possible that some blocking VNs convert to non-blocking network as result of dynamic bandwidth adjustment. Hence, we make congestion control as a triggered manner, by inspecting the VN type in case of bandwidth adjustment, dynamically booting or shutting down the congestion control work at endpoints.

We use the maximum flow problem to determine the VN type. First, we select one VM in VN as dst node at the right side and add a source node src at the left side of the other VMs. Then, we add edges from src to these VMs, and assign the bandwidth demand of the corresponding VM as weight of an edge. Thus, it transforms to seeking the maximum flow from src to dst. We iterate this procedure for all other VMs, if the maximum flow is lower than the receiving bandwidth of the dst node, respectively, we consider that the VN type is non-blocking. Otherwise it is blocking. We take the Edmonds–Karp [29] algorithm as an implementation for computing the maximum flow in $O\left(VE^2\right)$ time. Hence, the total time complexity for determining the VN type is $O(V^2E^2)$.

6. Bandwidth Guarantees Enforcement

In Section 5, for the puporse of fully utilizing the network resources, efforts have been made to allocate link bandwidth among tenants in a work-conserving way. However, the tenants have a right to send more traffic than the bandwidth they are allocated with, leading to degradation of network performance. For achieving network performance isolation among tenants, we avoid bandwidth contention on the intermediate links by dynamically limiting the traffic sending rate at the source. Even so, there still exists inevitable congestions for blocking VN in that multiple VMs belonging to one tenant may communicate to the only VM of the same tenant at their full speeds. Eventually, this incurs bandwidth contention at the receiver side. Therefore, a congestion detector is introduced to detect the congestion on the destination VM in a timely manner. Meanwhile, the Congestion Controller component is responsible of regulating the sending rate of each source VM by using the rate limiters on every virtual interface of VM. These rate limiters are dynamically created to control the sending rate of each source VM since congestion is detected at the destination. If the VM-to-VM flows disappear, the corresponding rate limiters will be deleted.

Observation in T_{track}: The distributed bandwidth enforcer tracks the receiving rate at the endpoint periodically (in an epoch of 10 ms), and the enforcer can be further utilized to calculate the sending rate for every sender. In order to react to the congestion problem quickly, we try to achieve average rate guarantees [30] within a short time.

The key of bandwidth guarantee enforcement depends on how to identify congestion at the receiver side and determine an appropriate traffic rate for each VM-to-VM flow at the sender side. Traffic congestion in the network can be determined according to the packet loss at receiver. Congestion can be prevented by allocating the received bandwidth among all active VM-to-VM flows in a timely and even way, and then, the allocated bandwidth guarantees can be enforced at the senders. In the mean time, these senders are marked as tagged so that the DBA module cannot change their allocated bandwidth guarantees arbitrarily at the sender sides, until the receiving rate at the receiver side has become significantly smaller than the allocated rate in several epochs. If the bandwidth at receiver has increased, the increased part will be allocated to the senders equally, which can reduce the frequency of congestion in short time. The DBA module allocates the redundant bandwidth (m) by distinguishing the senders (n) from one receiver, and the receiver can get the total bandwidth m while

the senders obtain the bandwidth of m/n in average. However, the specific bandwidth demand of each VM-to-VM flow cannot be described accurately using this bandwidth allocation method. To address this problem, the demands of VM-to-VM flows are further estimated by measuring their throughput between epochs based on the principle of max–min fairness. The distributed control loop Algorithm 2 is designed to enforce per-tenant bandwidth guarantees, and its algorithm is as follows.

Algorithm 2: Distributed control loop.

Let boundedcollection BC denote the flows with stable traffic rate smaller than the allocated bandwidth;
Let unboundedcollection UNBC denote the flows with unstable traffic rate smaller than or equal to the allocated bandwidth;
$ratelimiterstatus \Leftarrow untagged$;
if *congestion takes place* **then**
set each source VM rate to average receive bandwidth;
$ratelimiterstatus \Leftarrow tagged$;
max–min-allocate() under the total network capacity of the receiver;
periodically( ){
set BC and UNBC according to the continuous several rate measurement;
for *flow f in BC* **do**
$f.ratelimiterstatus \Leftarrow$ tagged && $\frac{f.allocated - f.rate}{f.allocated} > 30\%$ $f.ratelimiterstatus \Leftarrow$ untagged;
allocate the redundant bandwidth from BC to the flows in the UNBC evenly;
}

Flow bounded assertion: One tenant can lend its spare bandwidth to other tenants if and only if one or more of the flows are asserted as bounded. If the flow traversing on the link stays stable and does not request any extra bandwidth, it is called the bounded flow. Here, the key is how to determine whether the flow is bounded or not. We try to decide whether the flow is bounded by monitoring the flow rate in several epochs. However, there is another issue, and if the epoch is too short, the rate monitored in an epoch is not very accurate, in which case, a longer epoch is required. Thus, to maintain one set of data for each epoch, the moving average technique is employed to smooth the rate observed in an epoch. To get a smoother plot, the moving average is calculated based on the data of previous $k_{bounded}$ epochs (these epochs are generally long, say 30). In this way, current flow rate can be represented in a relatively accurate way. If the demand of a VM-to-VM flow is estimated to be smaller than the allocated bandwidth, and the continuous traffic rates collected after the moving average in $m_{bounded}$ (say 30) epochs are close, this flow is considered bounded and the bandwidth allocated to this flow can be reduced.

Another problem we should mention is how the flow recovers the lent bandwidth if needed. We solve this by setting the throttled rate to be slightly higher somewhat more (say x%) than the regulated rate to allows for ramping up. Generally, the bounded flow sends packets at the rate regulated by the rate limiter. When ramp up arises, the flow is deemed as unbounded and the initial fairly allocated bandwidth is recovered.

Flow bounded to unbounded assertion: After one flow is deemed as bounded, it may increase or decrease its sending rate at some certain moment. This indicates that the status of the flow has changed from being bounded to unbounded. If the flow lowers its sending rate, it may look for the next bounded status, and the spare bandwidth can be once again lent to others. We still use the flow bounded assertion mechanism to find the next bounded status. However, if the flow increases its sending rate after lending all its excess bandwidth, the lent bandwidth should be retrieved in a timely manner. To increase transparency and minimize harm to the tenant who has lent its spare bandwidth, we should quickly find the flow tendency to decide whether to reclaim the lent bandwidth. Hence, we employ the continuous traffic rates collected after the moving average in $m_{unbounded} < m_{bounded}$ (say 15) epochs.

If the flow rates of these epochs are in ascending order, it is determined that the lent bandwidth should be recovered. To determine whether another bounded status is achieved, we also define the moving average using $k_{unbounded} < k_{bounded}$ (say 10) epochs to highlight the variation. If the continuous $m_{unbounded}$ are in ascending order, then it is considered as having reclaimed more bandwidth from the bounded status.

7. Evaluation

In this section, we evaluate the performances of algorithms used to provide VDC-based fair allocation of redundant bandwidth and enforce allocated bandwidth guarantees, respectively.

Experimental setup: Four servers are used in the experiment, connected by an Openflow-based Gigabit Ethernet switch, and these servers are used to construct a virtualized environment, consisting of three compute nodes and one controller node. Each server has two 3 GHz Pentium(R) Dual-Core E6600 CPUs and 4 GB of memory. Because there are limited computing resources available, only two tenants are considered in the experiment. VM is created using the KVM virtualization technology, and its virtual interface is connected to the port of virtual switch created by OpenvSwitch [31]. All virtual switches are connected to a centralized Openflow controller [32]. The open source controller Floodlight [33] is used in the experiment. By using OVSDB, a management protocol for configuration of OpenvSwitches, these ports can be configured for dynamic control of the sending/receiving rate of corresponding VMs. As shown in Figure 7, two VNs are mapped to PN, in which, tenant A has four VMs, while tenant B has two VMs. There are two VMs from each tenant, these VMs are running on the shared host compute-3, and they accept traffic from other VMs. There are four flows in total, one of which comes from tenant B, and the other three are from tenant A. Each VM is guaranteed with the minimum bandwidth of 300 Mbps. Because the network environment is not an enclosed system in the experiment, certain deviation exists between the sending rate of the VM and the actual receiving rate obtained from measurement. In order to address this problem, 40% slack is provided in enforcing guarantees on a 1 Gbps link, and the bandwidth of only 600 Mbps link in that open network environment is susceptible to other services. Iperf [34] is employed to generate these flows using different protocols at different rates.

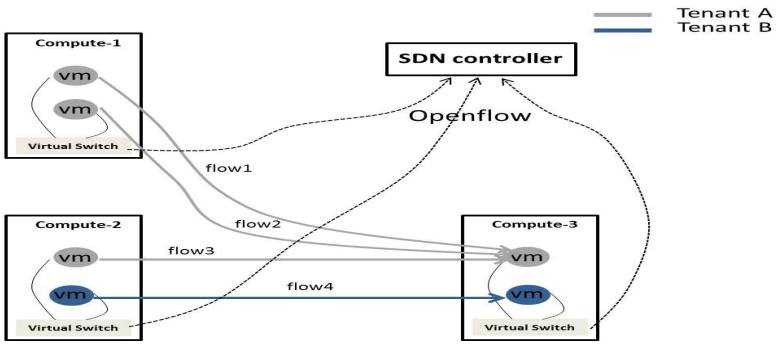

Figure 7. The flow status of tenant services.

7.1. The Bandwidth Guarantees Enforcement of VN

7.1.1. The Effect of Hypervisor-Based Congestion Control

The distributed bandwidth enforcer provides control of hypervisor-based congestion. To evaluate its performance, specific steps are designed for demonstration. The flow4 of tenant-B starts at its full rate first, followed by flow1, flow2 and flow3 at their full rates sequentially.

As shown in Figure 8, the monitored data of each flow are collected as one point every 10 ms, and the horizontal axis represents the number of points. At the time of t0, we start flow1, which occupies most of the allocated bandwidth (270 Mbps). At t1, flow2 is started, then the congestion at compute-3 is detected, and as a result, the sending rates of the source VMs are limited to 150 Mbps.

At t2, we start flow3, the bandwidth is evenly allocated among these flows. Based on the observation above, it can be seen that the distributed bandwidth enforcer is capable of preventing congestion at the destination in a timely manner, while making flow-based fair allocation of bandwidth at the same time.

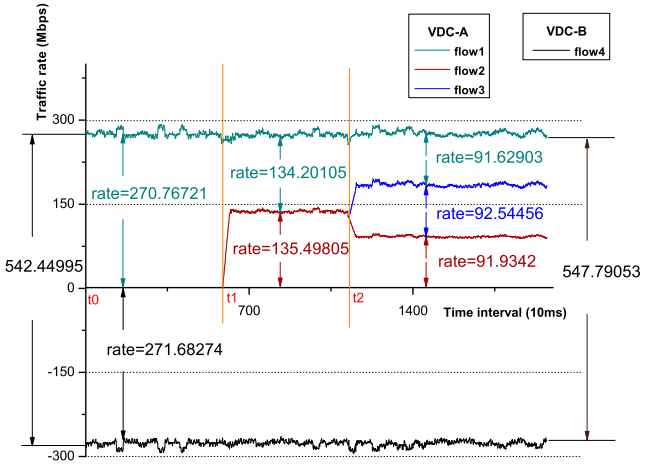

Figure 8. The variation of the flow rates.

7.1.2. CPU Overhead

The distributed bandwidth enforcer is within each hypervisor of the compute node. Generally, it is common that the CPU utilization of compute nodes in cloud datacenter maintains at high level (say 80%). We argue that the distributed bandwidth enforcer should not bring so much CPU overhead that it would compete with VMs. This impact is evaluated as Figure 9 shows. It is the mainstream that 30 or less VMs can be virtualized on one physical host. Every VM is attached one distributed bandwidth enforcer. Therefore, we use, at most, 30 threads (our implementation is multi-threaded) serving as distributed bandwidth enforcers. By varying the number of threads under the capacity of two CPU cores, the results of fine-grained profiling show that most CPU cycles are spent in detecting the number of active VM-to-VM flows, as well as packet drops. This overhead seemed to increase linearly with the increase in the number of threads. It almost used up all the CPU cores when the number of threads increased to 30. Evidently, the distributed bandwidth enforcer brings much CPU overhead while maintaining congestion control.

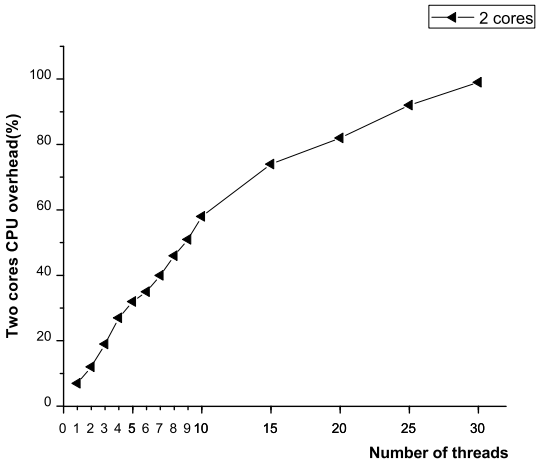

Figure 9. CPU variation with the number of threads.

Fortunately, we can eliminate the number of threads by inspecting the VN type. According to virtual network classification, only the VM belonging to the blocking VN can be equipped with an independent thread. We dynamically control the creation and termination of thread. In this way, to some extent, we can reduce the number of the threads, thereby decreasing CPU consumption. If the number of threads can be reduced to 10 when the type of certain number of VNs is non-blocking, it will lead to 60% CPU utilization, 40% of which is discounted.

7.2. The VDC-Based Fair Bandwidth Allocation

7.2.1. Tenant-Based Fair Redundant Bandwidth Allocation

In the above section, it is already shown that NXT-Freedom can provide minimum bandwidth guarantees and VDC-based fair allocation of redundant bandwidth. In this experiment, it is assumed that each VM has the minimum bandwidth guarantee of 100 Mbps. The performance of NXT-Freedom is compared with that of the following four approaches: (i) NoProtection—in this method, the traffic is sent directly, which has no bandwidth protection; (ii) Oktopus-like Reservation [14]—this is a reservation system with no work-conservation, in which, the flow rates are statically set based on the hose model; (iii) Seawall [16]—it is a work-conserving system with no minimum bandwidth guarantee, where the bandwidth of link is shared according to the number of active flows proportionally; (iv) ElasticSwitch [23]—this is a work-conserving system which can provide minimum bandwidth guarantee, but similar to Seawall, its link bandwidth is also shared according to the number of active flows in proportion, which fails to distinguish the flows of different tenants, and as a result, if one tenant has too many VM-to-VM flows traversing on the same link, excessive redundant bandwidth will be allocated.

Figure 10 shows the application-level TCP throughput of tenant B as the number of flows of tenant A changes with UDP traffic. The bar represents the throughput of each flow in respective setups. According to the results, when Seawall and ElasticSwitch are used, the bandwidth is allocated unfairly among tenants, and both methods are in favor of the number of senders. In comparison, NXT-Freedom is able to maintain VDC-based fairness by dividing the link bandwidth among tenants using this link, even if tenant B has more senders than tenant A.

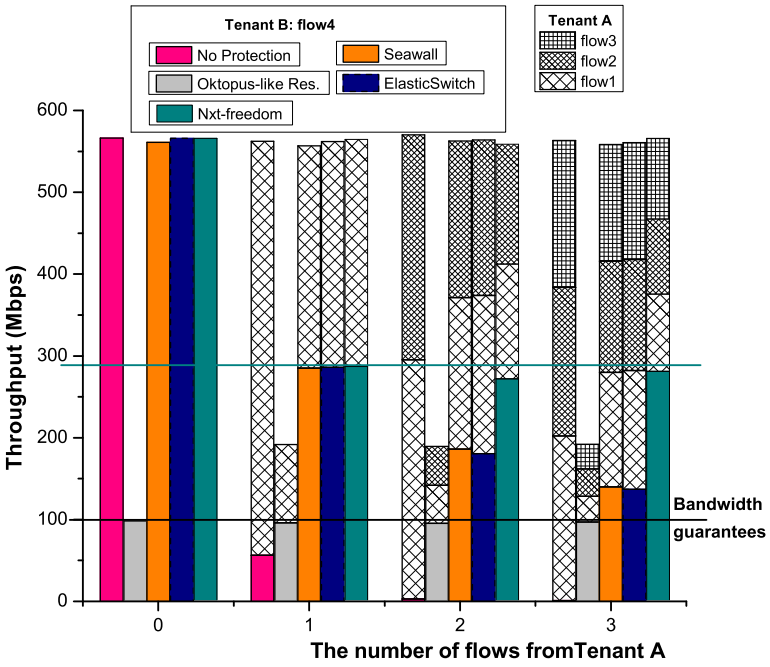

Figure 10. Comparison with state of the art.

7.2.2. The Precedence of Redundant Bandwidth Allocation

The dynamics of tenant flows may result in redundant bandwidth on certain links. The crux of redundant bandwidth allocation is that we should take intra-tenant allocation prior to inter-tenant allocation. The following procedures are going to exemplify this.

The procedures follow the experiment in Section 7.1 after all the flows become stable, and the result is as shown in Figure 11. At t0 time, we decrease the sending rate of flow1 from tenant A to 10 Mbps. At t1 time, we find that flow1 becomes bounded, and the spare bandwidth of flow1 has been evenly allocated to flow2 and flow3. This is the so-called intra-VDC allocation. At t2 time, all the flows are stable after the dynamic adjustment, and we reduce the flow rate of flow4 from tenant B to 50Mbps. Then, we can see that at t3 time, flow4 is deemed as bounded, and its spare bandwidth has been allocated between flow2 and flow3 from tenant A as flow1 is bounded. Thus, we prove that intra-VDC allocation takes precedence over inter-VDC allocation.

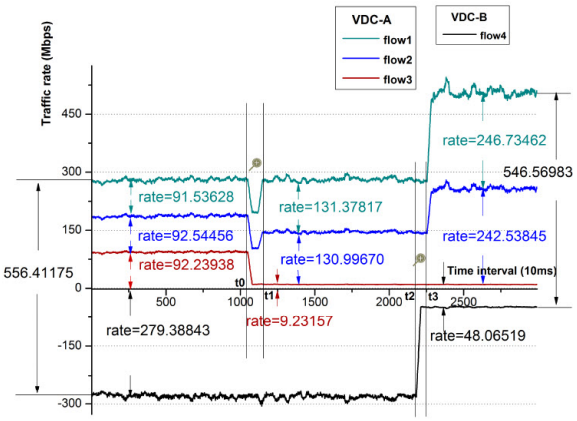

Figure 11. The variation of the flow rates.

We magnify the interval between t0 and t1 and, as Figure 12 shows, it costs 690 ms in total from flow3, reduced to 10 Mbps if flow3 is bounded. In the former 300 ms, the plots of flow1, flow2 and flow3 are moved down linearly in that the moving average technique (30 raw points) is utilized to smooth the plot. In the latter 390 ms, flow3 is deemed as bounded, though the accurate time is supposed to be 300 ms as we use 30 moving average points to determine whether a flow is bounded, it is within a tolerable range. We also magnify the interval between t2 and t3 and, as Figure 13 shows, it costs approximately 720 ms from flow4 reduced to 50 Mbps to the flow4 is bounded. The reason of the tendency of the plots is as that shown in Figure 12.

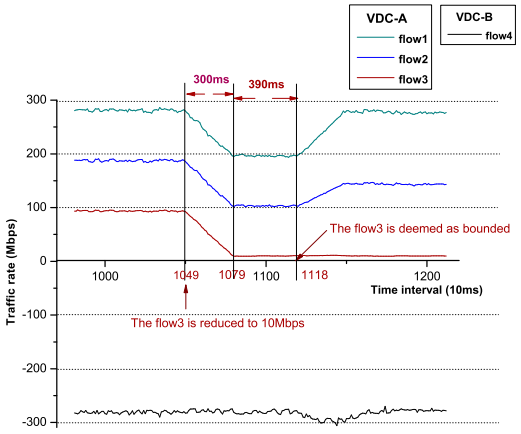

Figure 12. The detailed variation of flow rates in [t0, t1].

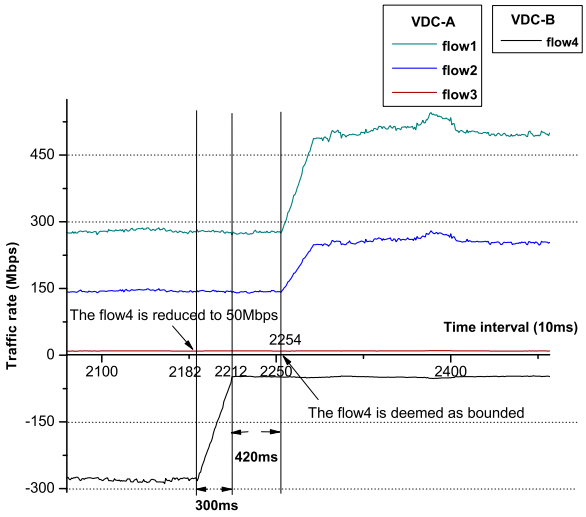

Figure 13. The detailed variation of flow rates in [t2, t3].

7.2.3. Rapid Recovery in $T_{recovery}$ Time

NXT-Freedom should rapidly recover the lent out bandwidth within $T_{recovery}$ time once lender claims. The prerequisite of bandwidth recovery is that the actual bandwidth allocated should be less than its minimum bandwidth guarantee. We set $T_{recovery}$~100 ms, which equals several periods of max–min allocation. It is expected that whether flow rate of the lender has an upward trend could be detected in $T_{recovery}$. Every period consists of query time of the receiving rate of active flows and calculation time of the suitable sending rate of each source VM as well as the short thread sleep time. To predict this upward trend, we set the throttle rate +20% of the regulated rate to allow ramp up, and we collect the continuous 10 flow rates. If these rates are in ascending order or some of them are equal to the throttled rate, then the lent out bandwidth will be reclaimed. As Figure 14 describes, we trace the stipulated bandwidth and the actual sending rate of each flow. At t0 time, we increase the rate of flow4 up to 150 Mbps. At t1 time, the upward trend of flow4 is detected, and then the rate of flow4 recovers to its original allocated bandwidth (300 Mbps). Besides, the bandwidth of flow1 and flow2 changes from 270 to 145 Mbps, respectively. At t2 time, flow4 is deemed as bounded, we evenly lend the redundant bandwidth of the flow4 to only the flow1 and flow2, as flow3 is bounded. Hence, we overlap the bandwidth of flow1 and flow2 with their corresponding traffic rate, respectively. Regarding flow3 and flow4, to reserve the possibility of taking back the lent bandwidth, they are allocated 20% more than the actual traffic rate each.

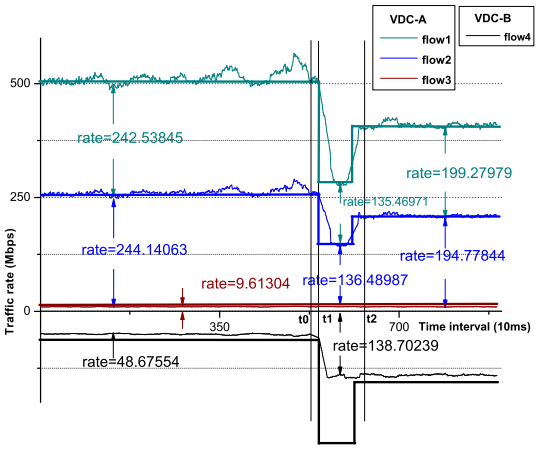

Figure 14. The rapid bandwidth recovery when increasing flow4.

We magnify the interval between t0 and t2 as Figure 15 shows, it costs 710 ms in total between flow4 increases to 50 Mbps and flow4 is bounded. In the former 180 ms, we detect the upward trend of flow4 rate. Obviously, this upward period is much shorter than the time of flow bounded assertion. In the latter 530 ms, flow4 becomes bounded. This time, the cost is much more than the ideal 300 ms, because it includes part of increasing time of flow4 as well.

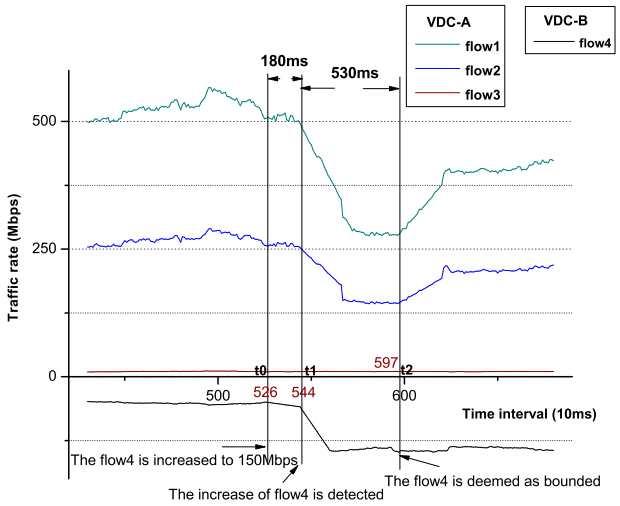

Figure 15. Detailed rapid bandwidth recovery in [t0, t2].

8. Conclusions

With the development of scale economy, more and more distributed applications are forced to co-exist with each other in shared infrastructure. The best-effort resource sharing mechanism of the network can cause unfair allocation of network bandwidth among VDCs. To address this problem, NXT-Freedom is an attempt made by us to achieve the enforcement of VDC-based fair bandwidth guarantees, while providing work-conserving allocations with minimum bandwidth demands at the same time. The minimum bandwidth demands abstracted by the SCG model in the VN are fine-grain than those of the hose model, further advancing the total network utilization. NXT-Freedom can be employed by the cloud service providers to achieve fair network bandwidth across tenants in a flexible and efficient manner, which can prevent accidental or malicious traffic interference. Through the classification of the VN type, to some degree, we can reduce the CPU overhead of NXT-Freedom, which makes NXT-Freedom more practical.

Author Contributions: Conceptualization, Shuo Wang and Zhiqiang Zhou; methodology, Shuo Wang; software, Hongjie Zhang; validation, Jing Li, Shuo Wang and Hongjie Zhang; formal analysis, Jing Li; writing–original draft preparation, Shuo Wang; writing–review and editing, Shuo Wang. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key Research and Development Program of China under Grant 2018YFB0204005.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shuja, J.; Bilal, K.; Madani, S.A.; Othman, M.; Ranjan, R.; Balaji, P.; Khan, S.U. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Syst. J.* **2016**, *10*, 507–519. [CrossRef]
2. Wang, S.; Li, J.; Zhang, H.; Wang, Q. Nxt-Freedom: Considering VDC-based Fairness in Enforcing Bandwidth Guarantees in Cloud Datacenter. In Proceedings of the 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), Chongqing, China, 11–14 October 2018; pp. 1–6.

3. Wang, S.; Li, J.; Wang, Q.; Zhang, H. Nxt-Max: for supporting VDC-based maximum redundant bandwidth allocation in cloud datacenter. *Clust. Comput.* **2018**, *22*, 12567–12580.
4. Noormohammadpour, M.; Raghavendra, C.S. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1492–1525. [CrossRef]
5. Zhang, J.; Yu, F.R.; Wang, S.; Huang, T.; Liu, Z.; Liu, Y. Load balancing in data center networks: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2324–2352. [CrossRef]
6. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 63–74. [CrossRef]
7. Guo, C.; Lu, G.; Li, D.; Wu, H.; Zhang, X.; Shi, Y.; Tian, C.; Zhang, Y.; Lu, S. BCube: A high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 63–74. [CrossRef]
8. Guo, C.; Wu, H.; Tan, K.; Shi, L.; Zhang, Y.; Lu, S. Dcell: A scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 75–86. [CrossRef]
9. Greenberg, A.; Hamilton, J.R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D.A.; Patel, P.; Sengupta, S. VL2: A scalable and flexible data center network. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 51–62. [CrossRef]
10. Greenberg, A.; Lahiri, P.; Maltz, D.A.; Patel, P.; Sengupta, S. Towards a next generation data center architecture: scalability and commoditization. In Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow, Seattle, WA, USA, 22 August 2008; pp. 57–62.
11. Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. *Nsdi* **2010**, *10*, 89–92.
12. Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. Technical Report RFC 2992. 2000. Available online: https://tools.ietf.org/html/rfc2992 (accessed on 5 March 2019).
13. Guo, C.; Lu, G.; Wang, H.J.; Yang, S.; Kong, C.; Sun, P.; Wu, W.; Zhang, Y. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In Proceedings of the 6th International Conference, Zurich, Switzerland, 14–17 September 2010; p. 15.
14. Ballani, H.; Costa, P.; Karagiannis, T.; Rowstron, A. Towards predictable datacenter networks. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 242–253. [CrossRef]
15. Popa, L.; Kumar, G.; Chowdhury, M.; Krishnamurthy, A.; Ratnasamy, S.; Stoica, I. FairCloud: Sharing the network in cloud computing. *ACM SIGCOMM Comput. Commun. Rev.* **2012**, *42*, 187–198. [CrossRef]
16. Shieh, A.; Kandula, S.; Greenberg, A.G.; Kim, C. Seawall: Performance Isolation for Cloud Datacenter Networks; HotCloud: 2010. Available online: https://www.usenix.org/event/hotcloud10/tech/full_papers/Shieh.pdf (accessed on 5 March 2019).
17. Hu, S.; Bai, W.; Chen, K.; Tian, C.; Zhang, Y.; Wu, H. Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud. *IEEE Trans. Cloud Comput.* **2018**. [CrossRef]
18. Kumar, R.; Hasan, M.; Padhy, S.; Evchenko, K.; Piramanayagam, L.; Mohan, S.; Bobba, R.B. End-to-end network delay guarantees for real-time systems using sdn. In Proceedings of the 2017 IEEE Real-Time Systems Symposium (RTSS), Paris, France, 5–8 December 2017; pp. 231–242.
19. Liu, F.; Guo, J.; Huang, X.; Lui, J.C. eBA: Efficient bandwidth guarantee under traffic variability in datacenters. *IEEE/ACM Trans. Netw.* **2017**, *25*, 506–519. [CrossRef]
20. Rodrigues, H.; Santos, J.R.; Turner, Y.; Soares, P.; Guedes, D.O. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. *WIOV* **2011**, *1*, 784–789.
21. Jeyakumar, V.; Alizadeh, M.; Mazières, D.; Prabhakar, B.; Greenberg, A.; Kim, C. EyeQ: Practical network performance isolation at the edge. In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, Lombard, IL, USA, 2–5 April 2013; pp. 297–311.
22. Cisco Data Center Infrastructure 2.5 Design Guide. Available online: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.html (accessed on 5 March 2019).
23. Popa, L.; Yalagandula, P.; Banerjee, S.; Mogul, J.C.; Turner, Y.; Santos, J.R. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 351–362. [CrossRef]
24. Lam, T.; Radhakrishnan, S.; Vahdat, A.; Varghese, G. *NetShare: Virtualizing Data Center Networks across Services*; University of California: Oakland, CA, USA, 2010.

25. Ali, B.S.; Chen, K.; Khan, I. Towards Efficient, Work-Conserving, and Fair Bandwidth Guarantee in Cloud Datacenters. *IEEE Access* **2019**, *7*, 109134–109150. [CrossRef]
26. Chen, K.; Yang, N. BwShare: Efficient bandwidth guarantee in cloud with transparent share adaptation. *Comput. Netw.* **2020**, *170*, 107095. [CrossRef]
27. Xie, D.; Ding, N.; Hu, Y.C.; Kompella, R. The only constant is change: Incorporating time-varying network reservations in data centers. *ACM SIGCOMM Comput. Commun. Rev.* **2012**, *42*, 199–210. [CrossRef]
28. Lee, J.; Turner, Y.; Lee, M.; Popa, L.; Banerjee, S.; Kang, J.M.; Sharma, P. Application-driven bandwidth guarantees in datacenters. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 467–478. [CrossRef]
29. Edmonds, J.; Karp, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* **1972**, *19*, 248–264. [CrossRef]
30. Pan, R.; Breslau, L.; Prabhakar, B.; Shenker, S. Approximate fairness through differential dropping. *ACM SIGCOMM Comput. Commun. Rev.* **2003**, *33*, 23–39. [CrossRef]
31. OpenvSwitch. Available online: http://openvswitch.org/ (accessed on 5 March 2019).
32. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
33. Floodlight Openflow Controller. Available online: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview. (accessed on 26 October 2020).
34. iPerf—The Network Bandwidth Measurement Tool. Available online: https://iperf.fr/ (accessed on 5 March 2019).