Sharing Cloud Networks with FairCloud

Gautam Kumar and Mosharaf Chowdhury *
University of California, Berkeley

ABSTRACT

The network is a crucial resource in cloud computing, but in contrast to other resources such as CPU or memory, the network is currently shared in a best effort manner. However, sharing the network in a datacenter is more challenging than sharing other resources. The key difficulty is that the network allocation for a VM `X` depends not only on the VMs running on the same machine with `X`, but also on the other VMs that `X` communicates with, as well as on the cross-traffic on each link used by `X`. In this paper, we first propose a set of desirable properties for allocating cloud networks with fairness and show that there exist multiple tradeoffs between some of these properties. Second, we show that the existing allocation models violate one or more of these properties, and propose a flexible mechanism that can be tuned to select different points in this tradeoff space.

1. INTRODUCTION

Cloud computing is becoming increasingly popular for deploying and running many of today's businesses. Core to cloud computing is the ability to share and multiplex resources across users. While there has been significant work on sharing CPU, memory or storage, cloud networks are shared in a best-effort manner making it hard for both users and cloud operators to reason about how network resources are allocated.

The traditional networking literature is rich with strategies for sharing individual links. These strategies, however, were generally aimed toward sharing access links that connected end hosts to the Internet, because access links are typically the bottlenecks. Unlike the Internet, a user can rent multiple virtual machines (VMs) in the cloud; consequently, sharing the network is not limited to individual links any more. It becomes challenging because it differs from independent resources like CPU, memory or storage in several key ways: network usage attribution involves at least two parties (the source and the destination), it is a distributed resource and thus cannot easily be divided, and network shares of a user on individual links depend on the communication patterns and VM placements of other users. Given these characteristics, the traditional notion of fairness on a single link cannot be used in cloud networks.

In this paper, we explore the notion of fairness in cloud networks. We define fairness across network shares to have several primary attributes: *work conservation* to completely utilize available resources, *weight proportionality* to give a notion of prioritization across different users with different payments, *strategy-proofness* to disallow cheating, and *envy-freeness* to ensure user satisfaction. In addition, given the distributed nature of the network, we consider *symmetry* to be able to calculate shares without explicit knowledge of the applications, *independence* between shares in individual links to maximize utilization and limit cheating, and finally, the amount of *guaranteed minimum bandwidth* to provide users with an upper bound on their communication time. Section 3 discusses these properties in detail.

We show that there are inherent tradeoffs between different sets of these attributes (*e.g.,* envy-freeness vs work conservation, work conservation vs strategy - proofness vs independence, and weight proportionality vs guaranteed bandwidth). We also explain how each of the existing network sharing mechanisms (*e.g.,* TCP-based per flow sharing, per source [17] or per destination [19] sharing, static allocation schemes [4, 11]) have implicitly chosen their corners.

Given the knowledge of various tradeoff spaces, we propose *Per Endpoint Sharing*(*PES*) that primarily aims toward high resource utilization (by picking work conservation) and prioritization (by choosing weight proportionality). We also propose two variants of *PES*, *LinkPES* that calculates weights based on information on individual links and favors independence, and *NetworkPES* that calculates weights in the network level and chooses strategy-proofness.

We compare our proposed schemes against the existing ones through flow-level simulation and click-based implementation to find that GlobalPES hits a sweet spot in the solution space with minimal CPU overheads and throughout loss.

*This is a joint work with Lucian Popa, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica.

Property	Definition	Motivation
B1. Work Conservation	If the traffic between VMs X and Y is bottlenecked at link L, then it should not be possible to increase the allocation for $X - Y$ without decreasing the allocation of another source-destination pair using the same link.	If this property is not satisfied, the network is not fully utilized even when there is unsatisfied demand.
B2. Weight Proportionality	On any congested link `L` actively used by a set T of VMs, any subset $Q \subset T$ that communicates only to other VMs in Q (*i.e.,* Q does not communicate with $T \setminus Q$) is allocated *at least* a total share of W_Q/W_T of the bandwidth, where W_Q is the total weight of the VMs in set Q, assuming all VMs in Q have unsatisfied demand. This allocation should occur regardless of the distribution of the VMs in the set Q between the two ends of the link and of the communication pattern over L.	This property can be seen as providing network shares that are proportional to payment. For example, if all VMs have equal weight and we have one tenant with k_1 VMs and another tenant with k_2 VMs that compete over L, then the ratio of the bandwidths allocated to them is k_1/k_2.
B3. Strategy Proofness	A set of VMs Q should not be able to increase its bandwidth allocation to another set of VMs P by modifying its behavior at the application level (*e.g.,* using multiple flows or adopting a different traffic pattern).	This property prevents VMs from obtaining an unfair bandwidth allocation with respect to competing VMs.
B4. Envy Freeness	A tenant should not favour the allocation of a different tenant with the same payment.	This property entails that a tenant may not want to swap its position with another.

Table 1: Fairness Properties of a Resource Allocation Mechanism

Before proceeding, we clarify some of the assumptions our work builds on. We assume an Infrastructure-as-a-Service (IaaS) model where users pay per VM [1]. Our discussion is *agnostic* to VM placement and routing algorithms which we assume are implemented independently, and we only consider network sharing in a single datacenter. Finally, our discussion is largely orthogonal to work on network topologies to improve bisection bandwidth [2, 10, 12], as the possibility of congestion (and hence the need for sharing policies) remains even in full bisection bandwidth networks—*e.g.,* many-to-many communication, as in map-reduce, can congest the links of source or destination VMs.

2. CHALLENGES SHARING NETWORKS

The network differs from other resources in terms of usage attribution, divisibility, and the impacts of communication patterns and VM placements on its divisibility.

1. Usage Attribution: One can clearly attribute the consumption of a unit of CPU or memory to a single VM. In contrast, network communication involves at least two VMs—a source and a destination. Which of the two VMs should consumption be attributed to?

At one extreme, one can attribute consumption entirely to the source and consequently make bandwidth allocation decisions with respect to sources; at the other extreme one might do the same with respect to destinations only. Prior models for network sharing have often *implicitly* assumed a particular design point; *e.g.,* RSVP [19] is destination driven, while Diffserv [13] and Seawall [17] adopt a source-based approach. We argue that the appropriate approach for cloud networks would be to consider *both* sources and destinations in making allocation decisions, because both endpoints are benefitted.

2. Distributed Resource: For resources such as CPU and memory, it is relatively easy to take the total aggregate resource and divide it into smaller units (cores, cycles, or pages) that can be *independently* allocated to different VMs. The network however is formed by a topology of interconnected links and thus is not easily divisible into independent units for allocation. The network allocation for a VM A depends on *any* VM whose traffic shares a link with traffic to/from A.

3. Communication Patterns: A further complication is that a source (destination) VM may simultaneously communicate with multiple destination (source) VMs; hence, one has to consider the broader communication pattern of a VM. The network allocation for a VM depends on the *other* VMs it communicates with.

4. VM Placements: Finally, each host can contain more than one VMs that share the access link bandwidth of that host. As a result, the network share of a VM can very well be determined by the `other` VMs collocated with it.

3. PROPERTIES FOR NETWORK SHARING

Existing cloud providers price VMs based on different shares of CPU, memory and storage they provide but leave network sharing to the underlying transport mechanisms. We extend this model to associate a (positive) *network weight* to each VM and factor this weight in their prices alongside other resources.[1] Using this

[1]We consider the problem of appropriate placement of VMs based on their desired shares/application level requirements to be an orthogonal issue because private clouds already

Property	Definition	Motivation
E1. Symmetry	Assume all links in the network have the same capacity in both directions. If we switch the directions of all flows in the network, then the reverse allocation of each flow should match the original (forward) allocation of that flow.	Existing allocation models make an implicit assumption as to whether the allocation is receiver or sender centric; however, in general, it is difficult to anticipate application-level preferences. For example, server applications might value outgoing traffic while client applications might value incoming traffic. In the absence of application-specific information, we prefer allocations that provide equal weight to both incoming and outgoing traffic.
E2. Independence	The bandwidth allocations for a VM along two paths that share no congested links should be independent. In particular, if a VM sends traffic on an uncongested path, this should not affect its traffic on other congested paths.	This is a property that is satisfied in today's Internet. Lack of this property would lead to inefficient utilization; for example, an endpoint might refrain from sending on an uncongested path in order to get a larger traffic share on a different congested path. Note: This property also comes into play between network-level allocation and link-level allocation, any link-level allocation has this property, while network level allocations tend not to have it.
E3. Guaranteed Bandwidth	Each VM X is guaranteed a bandwidth allocation of B_{minX}, as if X were connected by a link of capacity B_{minX} to a central switch with infinite capacity to which all other VMs are also connected (see Fig. 1). This is also known as the hose model [7].	There is a lower bound on the bandwidth allocated to X regardless of the traffic demands and the communication patterns of the other VMs. This property enables predictability in tenant applications. For example, if one knows the communication pattern between her VMs, she can select the weights accordingly and predict the application performance. Higher guaranteed bandwidths provide stronger incentives for tenants to rent VMs with higher weights.

Table 2: Desirable properties for sharing Network Bandwidth

model as a starting point for sharing cloud networks, we consider its design implications without making any claims about its superiority.

With the notion of explicit payment for the network in place, we seek desirable properties that a mechanism to share the network should provide. We classify such properties into two categories. The first set, Table 1, enlists the properties that are common for fair allocation of any resource. *Work conservation* illustrates the efficiency of a resource allocation mechanism to not waste resources if there exists a demand for them. *Weight proportionality* is the ability of a mechanism to bind resource allocations to weights (payments) and provide *at least* a share that conforms with its payment. Such a mechanism should also be *strategy-proof* against application-level tricks played by users to increase their allocations. Finally, *envy-freeness* implies that between two tenants paying equally, the allocations of one is not better than that of the other. It is important to note that while weight proportionality compares different points in the weight spectrum and compares the relationships between resource allocations and corresponding weights, envy-freeness compares resource entitlements for different entities at the same weight.

However, for a distributed resource like the network, where (i) a network-wide allocation is a conglomerative result of allocations in multiple links, and (ii) which cannot be attributed to a single entity (source or destination), any network sharing mechanism must satisfy some additional properties (Table 2) as well.

For resources like CPU or memory, *weighted max-min fairness* satisfies all of the basic properties; however, there is no clear way to achieve the same fairness attributes for the network. Many of these properties possess inherent tradeoffs, and it comes down to selecting the properties that are more desirable than the others. To this end, we try to formalize the tradeoffs between some of these properties in the remainder of this section. For simplicity, assume the *network weight* of each VM to be 1.

Tradeoff 1. Envy-Freeness vs. Work Conservation: We argue that considering VM placements to be outside the scope of a network sharing mechanism creates a ***strict*** tradeoff between *envy-freeness* and *work conservation*. Consider a tree topology for the data center network, and two tenants, T_1 and T_2 each having x VMs, with T_1 having all its VMs on the left sub-tree and T_2 having all its VMs on the right subtree. If another tenant, T_3 is also placed along with T_1, it competes for T_1's network share thereby decreasing it, whereas T_2's share does not change if we want to be work conserving. In such a scenario, T_1 *envies* T_2.

Tradeoff 2. Work Conservation vs. Strategy-Proofness vs. Independence: There exists a three-way tradeoff between the desired properties of *work-conservation*, *strategy-proofness* and *independence* properties. A mechanism that tries to be resilient to the different communication patterns exhibited by different

allow users to specify custom placement constraints, and public cloud providers might support that as well in the future.

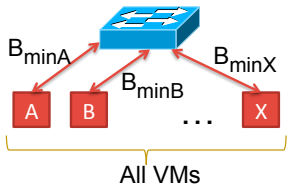


Figure 1: Model for Guaranteed Bandwidth.

tenants will need to perform a global optimization by allocating bandwidth at a network level rather than at the level of each congested link, and thus violate the independence property. In such a scenario, the entitlement of a VM for one of its communications depends on its usage across other communications. Both the properties can be satisfied if a pre-determined share is specified for every communication. However, this violates the work conservation property.

Tradeoff 3. Weight Proportionality vs. Bandwidth guarantee: We observe another tradeoff between weight proportionality and minimum bandwidth guarantee. Payment describes the degree of correlation of the bandwidth allocation with weights. Bandwidth guarantee, on the other hand, enables predictability for the applications deployed in clouds. For example, if a user rents two VMs (`A` and `B`), she receives a lower-bound on the bandwidth that will be allocated for the traffic between the two VMs, irrespective of the communication demands of the other VMs in the network. The guaranteed bandwidth in this case is $\min(B_{minA}, B_{minB})$. As shown in Figure 1, this is equivalent to network shares provided by a star-topology network where each VM is connected to a central switch using a link with capacity equal to that of the VM's minimum guaranteed bandwidth. If allocations were to be strictly bound to the weight, then the network share of one particular VM can become arbitrarily small with the increasing weights of its collocated VMs.

The example in Figure 2 helps illustrate this tradeoff. `A` and `B` are two VMs collocated on the same server, and `A` communicates with one other VM while `B` communicates with 10 others. Let all VMs have unit weights. If we consider a proportional allocation, `A` gets only 2/13 of the access link (as there are 13 VMs in total competing for the access link). It is easy to see that if `B` communicates with more VMs, `A`'s share of its access link would further decrease. A strictly monotonic allocation might provide a larger share of the access link capacity to `A` compared to a proportional allocation, but it would also reduce `A`'s share when any of the remote hosts communicating with `B` increase their weight.

However, in a different model that satisfies only monotonicity, `A` could be guaranteed a sizable fraction of its access link capacity, regardless of the communication pattern of `B`. For example, in a network with full bisection bandwidth, `A` could be guaranteed half of the access link capacity to communicate with other VMs in the network. In this case, the flows of `A` and `B` would each get $\frac{1}{2}$ of the access link, which is not achievable while having strict monotonicity, let alone proportionality. Thus, one can achieve higher bandwidth guarantees but be less sensitive to weights, or respect weights strictly on each link but provide very small bandwidth guarantees.

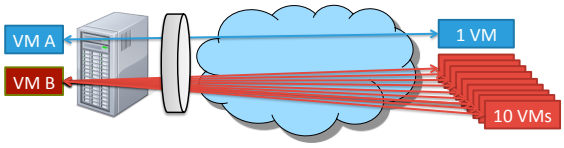


Figure 2: Sharing the access link of one machine.

Strictly speaking, the guaranteed bandwidth property is achieved by any allocation, since the number of contenders is always bounded in practice (*e.g.,* even with a per flow allocation, there is a practical limit to the number of flows used). Ideally, we would like the minimum bandwidth guarantee to be comparable to the bisection bandwidth of the network divided by the number of VMs in the network, *i.e.,* if we increase the network size and scale the bisection bandwidth by a similar factor, then the guaranteed bandwidth would remain the same. We also note that preserving the value of the guaranteed bandwidth during one VM's lifetime relies on a form of admission control of VMs and weights into the network.[2].

4. PER ENDPOINT SHARING (PES)

Given the desirable properties and tradeoffs between them, we seek to explore some points in this design space. First and foremost, we select *work conservation* against *envy-freeness* with the belief that efficient utilization of network bandwidth is of utmost importance. With this preference, we present *Per Endpoint Sharing* (*PES*), a flexible mechanism to share cloud networks that enables us to achieve different points in the other two tradeoffs. We build on the premise that a communication between two VMs should be attributed to both the sender and the receiver, and thus a natural approach is to assign each source-destination pair a weight based on the weights of *both* the source and the destination, *i.e.,* $W_{S-D} = f(W_S, W_D)$, where W_S is the weight of endpoint `S`. To satisfy the symmetry property, the weight of the allocation should be the same in both directions, *i.e.,* $W_{A-B} = W_{B-A}$.

***Per Endpoint Sharing* (*PES*)** is a mechanism that assigns to a communication between VMs `A` and

[2]The number of VMs per server is anyway limited by the number of CPUs, memory, etc.

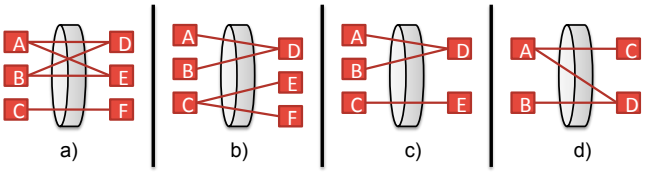


Figure 3: Sharing a single link, examples. A square represents a VM while a line connecting them represents a communication. Each VM can belong to any tenant.

B on link L a weight of:

$$W_{A-B} = \frac{W_A}{N_A} + \frac{W_B}{N_B} \quad (1)$$

where N_A is the number of other VMs A is communicating with (similarly N_B). On a particular link, the weights are normalized by dividing them with W_T, the total weight of all communications on that link. The share of a particular tenant with VMs $\{\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_n\}$ is then,

$$\sum_{i=1}^{n} \sum_{j=1}^{N_{A_i}} \frac{W_{A_i}}{N_{A_i}} = \sum_{i=1}^{n} W_{A_i} \quad (2)$$

In this way, *PES* achieves *weight proportionality* as described in Table 1. For example, consider Figure 3 (c) and assume that all VMs have unit weight. *PES* would assign weights of 1.5, 1.5, and 2 to A-D, B-D, and C-E, respectively. So the flows between A-D and B-D would receive $\frac{1.5}{5}$ of the link capacity, since D's weight is split across its two flows. In Figure 3 (d), *PES* would assign weights of 1.5, 1.5, and 1 to A-C, B-D, and A-D, respectively. The flow between A and D would then receive $\frac{1}{4}$ of the link capacity.

Note that a drawback of the *PES* mechanism, common to all mechanisms that compute a static weight for each source destination pair, is that the properties exhibited by the allocation are different for different demands. For example, consider Figure 3 (c) and assume that all VMs have unit weight. If the flow between A and D has a very small demand ϵ the allocation between B-D and C-E will respect the ratio of 1.5/2 instead of a more desirable 1/1 ratio. Clearly, this could be addressed if we also take into account the actual demands, however, such a mechanism will in practice be more difficult to apply, and we leave its exploration to future work.

Table 3 illustrates how *PES* satisfies the properties discussed in the previous section. Note that the theoretical bandwidth guarantee offered by *PES* is small since, in the worst case, one VM might have to divide its access link fairly with all the other VMs in the network. However, if the congestion is in the center of the network (as it typically occurs) and the routing can balance the traffic across all available paths, a VM would in fact get its fair share of the bisection bandwidth.

We return to the Equation (1), and specify two different strategies for the computation of N_A and N_B that lets us achieve two points in the tradeoff space. In the first strategy, *LinkPES*, we chose N_A (similarly N_B), to be the number of VMs A talks to on a link to link basis. This enables the links to compute N_A (and N_B) locally for each of the communications and do *weighted fair queueing* to give appropriate shares. However, this also means that the weight of a communication can arbitrary vary from one link to the other.

In the second strategy, *NetworkPES*, we choose N_A to be the number of VMs A talks to in the entire network. This implies that switches need to know the weight for a particular communication and cannot locally compute it, however this ensures that a communication has the same *weight* across all links, and can thus be approximated using *Core-stateless fair queueing*.

We observe that since the weights in *LinkPES* only depend on the communicating VMs on a single link and is oblivious to the allocations at other links, it is able to satisfy the independence property. However since in *NetworkPES*, the weight of a communication depends on parameters that are global in nature, it voids independence across paths *i.e.,* , a VM talking to N other VMs can increase its weight on remaining $N-1$ communications by ceasing to communicate with one of the VMs. In this way, these two policies align themselves at the opposite ends of the *strategy-proofness vs. independence* tradeoff.

We observe that *NetworkPES* can provide a worst-case bandwidth guarantee of $\frac{C}{W_T}$, where W_T is the total weights of the VMs in the network. If we consider a particular VM-VM communication, A-B, and another VM C, is co-located with A and is communicating with all the VMs in the network, then the share of A-B can go down to $\frac{C}{W_T}$. Intuitively, to offer higher worst case bandwidth guarantees, we would like to give higher importance to some VMs compared to others based on the "importance" of the link with respect to the VMs. For example, on the access link of one host, we would like to divide the link in a proportion closer to the VMs on that host rather than to the remote VMs.

To this end, we generalize *PES* to $W_{A-B} = W_{B-A} = \alpha\frac{W_A}{N_A} + \beta\frac{W_B}{N_B}$. The coefficients α and β provide the ability to weight differently the VMs located on the two sides of the link: α is applied to all the VMs on one side of L while β to the VMs on the other side. In this way, the weights of the VMs on one side of L can be scaled up/down or even completely disregarded by using different values for α and β.

By setting specific values for α and β at different links in the network, one can use the generalized *PES* mechanism to achieve different design points along the the described tradeoff, trading bandwidth guarantee with

Property \ Mechanism	PerFlow	PerSource	Static Allocation	LinkPES	NetworkPES
B1. Work Conservation	√	√	×	√	√
B2. Weight Proportionality	×	×	√	×	√
B3. Strategy-Proofness	×	×	√	×	√
B4. Envy-Freeness	×	×	√	×	×
E1. Symmetry	√	×	√	√	√
E2. Independence	√	√	√	√	×
E3. Guaranteed Bandwidth	None	Very small	Exact	Very small	Small

Table 3: Properties achieved by different network sharing mechanisms. The guarantees are discussed in the context of a tree-based topology.

weight proportionality. For example, consider the following mechanism, called *OSPES* henceforth, applicable to tree-based topologies (*e.g.,* traditional datacenter architectures, VL2 [9], fat-trees [2]). *OSPES* sets $\alpha = 1$ and $\beta = 0$ for all links in the tree that are closer to `A` than `B` and $\alpha = 0$ and $\beta = 1$ for all links closer to `B` than `A`. Essentially, *OSPES* translates into applying per source fair sharing for the traffic towards the tree root and per destination fair sharing for the traffic from the root. In a full bisection bandwidth network, each VM is guaranteed a bandwidth that represents its fair share of the access link when competing *only* with the other VMs collocated at the same host and not on the entire network with the assumption that the total weight of the VMs collocated on each host is the same throughout the network.

Lastly, we note that one might consider as alternative to *PES* to apply max-min [5] at the granularity of VMs (max-min is the most common mechanism for allocating bandwidth between flows). Max-min fairness could be applied at the granularity of VMs by maximizing the minimum allocation for the traffic sent or received by any VM. For example, assuming equal weight VMs and infinite traffic demands in Figure 3 (b), VMs `A`, `B`, `E` and `F` would each get an equal allocation to send/receive 1/4 of the link capacity. While this approach is fair at the VM level and it satisfies the symmetry property, it may lead to some flows not getting any traffic at all, which would violate the non-zero flow allocation property. For example, in Figure 3 (d), per VM max-min fairness would allocate one half of the capacity to the flows between `A` and `C` and one half to the flows between `B` and `D`. This is a perfect distribution of the capacity between the VMs (each would send/receive an equal amount), but there is no traffic between `A` and `D`. This allocation is also not strictly monotonic and does not provide adequate bandwidth guarantees.

5. ALTERNATIVES TO PES

Traditional approaches to network sharing range from using TCP-based fairness across multiple flows in one extreme to static allocation across the whole network in the other. In this section, we compare *LinkPES* and *NetworkPES* with their counterparts and compare all of them (Table 3) in the light of the properties and tradeoffs defined in Section 3.

5.1 Per Flow Sharing and Variants

The traditional approach to sharing network resources is to perform *per flow* fairness, where a flow is characterized by the standard five-tuple in packet headers. However, per flow allocation could lead to unfair bandwidth allocation at the VM (endpoint) granularity [6]. Indeed, two VMs can increase the traffic allocation between them at the expense of other VMs by simply instantiating more flows. Thus, a fundamental weakness of the per-flow model is that a user has a *strategy* to increase the traffic allocation between a pair of his VMs by instantiating multiple flows in order to compete unfairly with other users.

A natural workaround for the per-flow allocation unfairness would be to use a *per source-destination pair* (per S-D pair) allocation model, where each source-destination pair is allocated an equal share of a link's bandwidth regardless of the number of flows between the pair of VMs. However, this model is still arguably unfair, as a VM that communicates with many VMs gets more bandwidth than a VM that communicates with fewer VMs. For example, a user that employs an all-to-all communication pattern between N VMs will get a bandwidth share of $O(N^2)$, while a user that performs one-to-one communication between the same number of N VMs will get a share of only $O(N)$. Figure 3 (a) shows one such example, where the allocation of hosts `A,B,E,F` is twice that of hosts `C,D,G,H`.

5.2 Per Source/Per Destination Sharing

To address this problem, previous works have proposed using a *per source* allocation model, *e.g.,* Seawall [17]. With such a model, sources communicating over a given link are assigned equal weights, and the traffic is divided fairly between sources. While this model is fair to sources, it might not be fair to destinations. For example, in Figure 3 (b) `E` and `F` receive four times less traffic than `D`, and for traffic flowing in the opposite direction, `A` and `B` receive four times less traffic than `C`.

Similarly, while a *per destination* allocation model has some desirable properties, such as providing protec-

Table 4: Compared Strategies

Notation	Strategy Description
PerFlow	TCP fair sharing between flows
PerSource	Share calculated by the source VM (*i.e.,* Seawall [17])
LinkPES	PES using local knowledge
NetworkPES	PES using global knowledge

Table 5: Compared Communication Patterns

Notation	Pattern Description
11	Each VM talking to one other VM
$1N$	One VM talking to $(N-1)$ VMs
NN	All VMs talking between each other
MR	M VMs talking to R VMs (similar to MapReduce shuffle)

tion against DoS attacks [18], it is not fair to sources. Thus, these allocation models are *asymmetric* in that they can only be fair with either sources or destinations, but not both. Such allocation asymmetry is undesirable, as it implicitly assumes that the network has knowledge about whether a VM values more the incoming or the outgoing traffic, which is not the case in practice. All examples in Figure 3 (b), (c) and (d) suffer from asymmetry, as the allocation for each host and each S-D pair will be different along the two directions of the link.

5.3 Static Allocation Model

Static allocation models [4, 11] take an extreme approach, where individual S-D pairs have exact bandwidth reservations. These models satisfy all the desirable properties except for work conservation, because unused bandwidth is never redistributed. Unfortunately, this results in underutilization of resources. Reservation schemes result in fragmentation as well.

6. EVALUATION

While link-level behavior of different network sharing strategies can be calculated analytically, how they actually perform across the network cannot. We performed flow-level simulations and experimental evaluation using a kernel-level Click implementation to measure the overheads of the proposed strategies as well as to compare them against the status quo.

6.1 Simulation

To better understand the network-wide behavior of different strategies, we have performed flow-level simulation by varying weights of each tenant and their communication patterns on three-level eight-node fat tree (Figure 5) and oversubscribed tree (Figure 6) topologies. Table 4 summarizes the strategies we have compared in these simulations, and Table 5 describes the communication patterns we have considered. Result discussed below are averaged over 500 simulation runs.

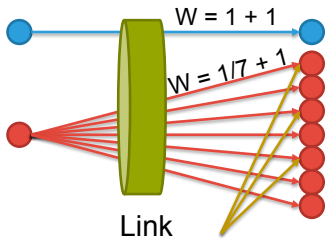


Figure 4: Links in *LinkPES* do not get to see yellow flows and compute weight of each of red flows as $\frac{1}{7}+1$, giving the red tenant 4 times as much share as the blue tenant on this link.

We summarize the highlights of the simulation results shown in Figure 5 and Figure 6 in the following:

TCP is not an option: Without any notion of tenant weights, TCP cannot reflect the payment in network shares. For example, in Figure 5(e) and Figure 6(e), PerFlow is giving equal shares to each tenant even though the second tenant has three times more weight; this is true even when both the tenants have the same communication pattern.

***LinkPES* fails in the presence of differing communication patterns:** Consider the scenario where two tenants having the same number of VMs paying the same amount exhibit different communication patterns. Consider our 8-node tree toplogy with three layers and tenants, T_1 and T_2 with T_1 having a workload in which each VM talks to another VM, whereas T_2 having a workload which constitutes all VMs talking to all other VMs. Assuming a unit payment for every VM for both the tenants, the scenario at the outward access link is depicted in Figure 4, with T_1 in red and T_2 in blue. Since, the link lacks the knowledge that each of the seven red destinations are also receiving from other sources, *LinkPES* assigns a weight of $\frac{8}{7}$ to each of T_1's communications giving T_1 a total weight, 8 compared to T_2's total weight, 2 (Figure 5. This implies that *LinkPES* is not strategy-proof with respect to the communication pattern of tenant.

***NetworkPES* has maximum resilience:** By incorporating the most amount of network knowledge from both sources and destinations, *NetworkPES* allocates shares that respect to tenant weights (*e.g.,* Figure 5(b) and Figure 6(f)). However, this strategy-proofness comes at the cost of independence, since a VM may decrease the value of N_A by talking to only a few other VMs, thereby increasing its weight on all links.

For similar communication patterns both, *LinkPES* and *NetworkPES* exhibit better weight propor-

Table 6: Experimental vs Simulated Shares

Strategy	T1 (Mbps)	T2	Ratio
PerFlow (Sim)	40	759	18.98
PerFlow (Exp)	35	662	18.91
NetworkPES (Sim)	127	672	5.29
NetworkPES (Exp)	110	583	5.30

tionality: In the same topology, consider 5 tenants each having 8 VMs, 4 of them sending data to the to other 4 VMs (similar to MapReduce shuffle), with the payments made by the tenants for their VMs in the proportion 1:2:3:4:5. Averaged over 500 runs, we observe that both *LinkPES* and *NetworkPES* give better proportionality compared to PerFlow or PerSource strategies (Figure 7).

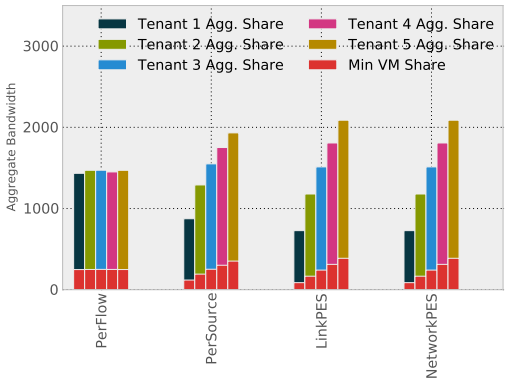


Figure 7: Weight Proportionality in the presence of 5 tenants, each having 8 VMs and performing a Map-Reduce style shuffle with 4 Mappers and 4 Reducers.

6.2 Experimental Results

To validate the simulation results, we have implemented *NetworkPES* in the kernel mode using the Click modular router (Figure 8) and evaluated it in the DETERlab testbed using the same oversubscribed tree topology used in our simulations. Each router implements weighted fair queueing (WFQ) across all the active flows using weights calculated at end host hypervisors by *NetworkPES*. We encode the weight of a flow, computed using *NetworkPES*, in the packet header.

6.2.1 Simulation Validation

We have recreated the simulation settings for Figure 6(a), Figure 6(b), and Figure 6(d) in DETERlab and compared the experimental results with that from simulation. For situations where both the tenants have 11 or NN communication patterns, experimental results were the same as simulation (*i.e.,* both tenants got equal shares). Table 6 compares the shares between two tenants when the first has a 11 communication pattern, while the second has NN. Note that the ratios match perfectly, even though the aggregate bandwidth do not since even without click, the measured maximum throughput in this testbed is 89Mbps on 100Mbps advertised links. We discuss the overheads next.

6.2.2 Throughput and Overheads

While performing the aforementioned experiments, we also collected data to compute the overhead and throughput of our implementation on individual links between VMs. We found CPU overhead to be less than 3% and throughput decrease due to our Click implementation less than 1 Mbps.

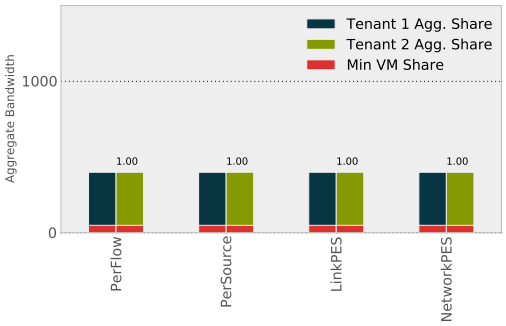
7. RELATED WORK

Recently, there have been a few proposals on how to share the network within a datacenter. Seawall [17] proposes to enforce fairness in hypervisors based on ECN feedback from switches, however it uses per source sharing. Gatekeeper [16] proposes a per-VM hose model similar to our guaranteed bandwidth model for full bisection - bandwidth networks. Gatekeeper uses a hypervisor - based approach and provides guarantees for sharing access links. We are currently investigating whether some variants of our more general sharing mechanism can be implemented in hypervisors as well.

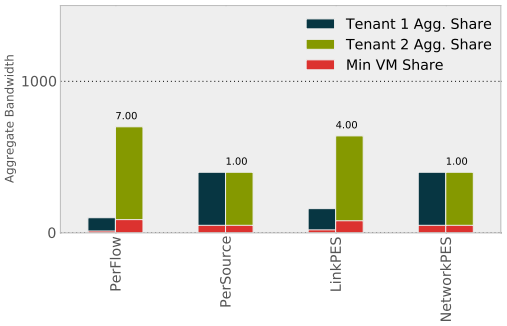
Other related work propose bandwidth allocations at the granularity of tenants rather than VMs. We divide these approaches into two broad categories: (a) reserving virtual networks for each tenant [4, 11] and (b) network multiplexing along with per tenant weights [14]. We discuss these schemes below.

Reserving virtual networks per tenant as proposed by Oktopus [4] and SecondNet [11] does provide bandwidth guarantees, but only when communicating with other VMs of the same tenant. More importantly, a reservation system does not achieve the Pareto Efficiency property, since the unused bandwidth is not shared between tenants. For small tenants in oversubscribed networks, per tenant reservation can offer higher bandwidth guarantees than per VM sharing, by reserving clusters of nodes that are collocated on the same or nearby racks. However, per VM sharing discussed in this paper could also be extended to offer different bandwidth guarantees to different sets of VMs (*e.g.,* VMs rented by the same tenant vs. other VMs), when it is integrated with VM placement. And while specifying a virtual topology could provide fine grain control to users, the allocation at a VM level is simpler. Users do not need to also specify network topologies and to possibly update them when adding new VMs (especially since adding machines on a daily/hourly basis is common).

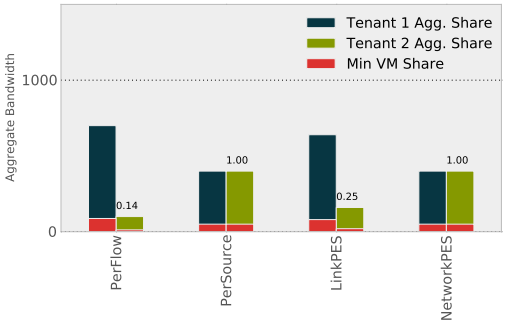
Schemes that advocate network multiplexing through the use of per tenant weights (*e.g.,* NetShare [14]) pro-



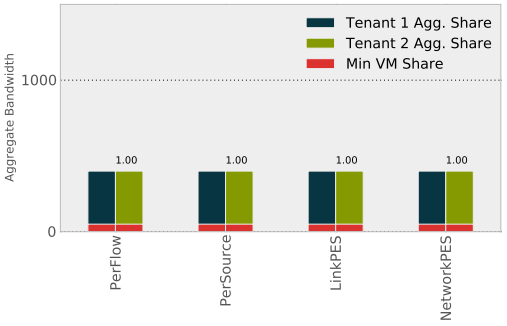
(a) $(P_1 : P_2 = NN : NN)$, $(W_1 : W_2 = 1 : 1)$



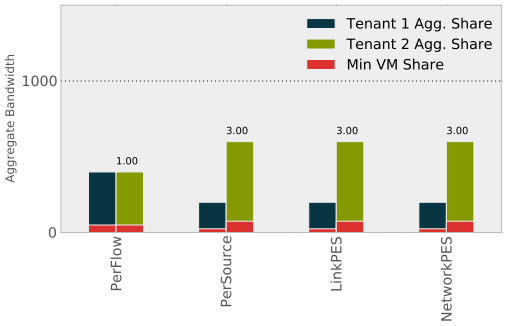
(b) $(P_1 : P_2 = 11 : NN)$, $(W_1 : W_2 = 1 : 1)$



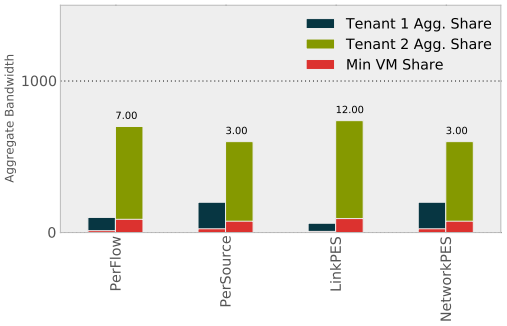
(c) $(P_1 : P_2 = NN : 11)$, $(W_1 : W_2 = 1 : 1)$



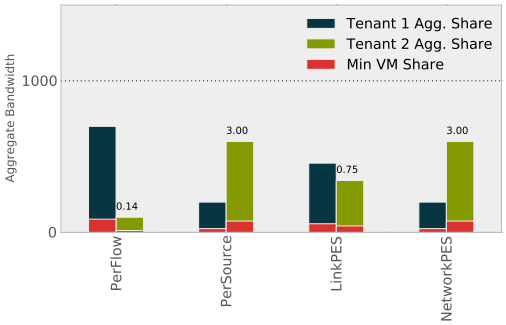
(d) $(P_1 : P_2 = 11 : 11)$, $(W_1 : W_2 = 1 : 1)$



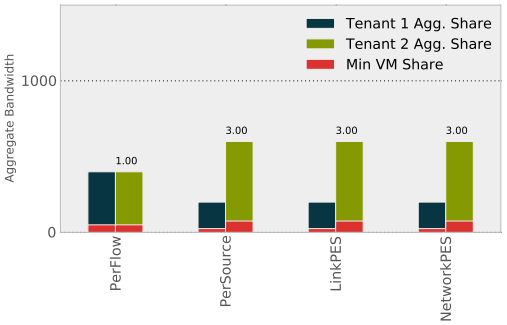
(e) $(P_1 : P_2 = NN : NN)$, $(W_1 : W_2 = 1 : 3)$



(f) $(P_1 : P_2 = 11 : NN)$, $(W_1 : W_2 = 1 : 3)$

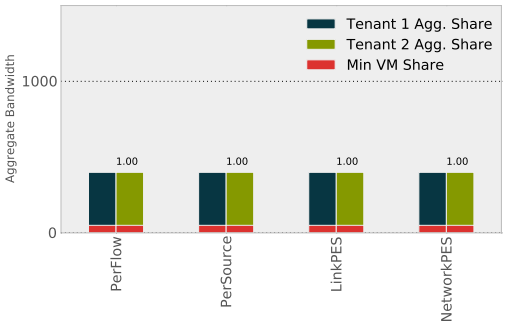


(g) $(P_1 : P_2 = NN : 11)$, $(W_1 : W_2 = 1 : 3)$

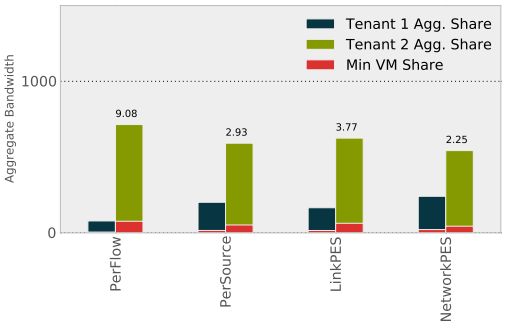


(h) $(P_1 : P_2 = 11 : 11)$, $(W_1 : W_2 = 1 : 3)$

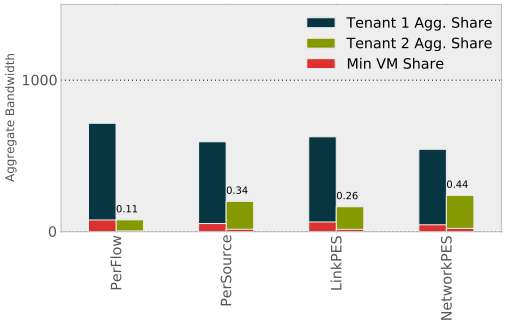
Figure 5: Simulated aggregate bandwidths of two tenants on a eight-node three-level full bisection bandwidth tree with 100 Mbps duplex links to end hosts. Each tenant has one VM in each host. P_i and W_i denote the communication pattern and weight of Tenant i, respectively. Numbers on top of the second tenant's bars denote the ratio of aggregate network shares.



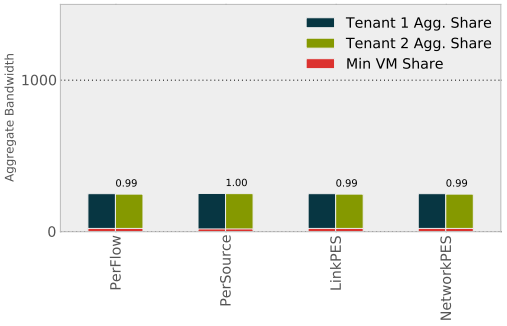
(a) $(P_1 : P_2 = NN : NN)$, $(W_1 : W_2 = 1 : 1)$



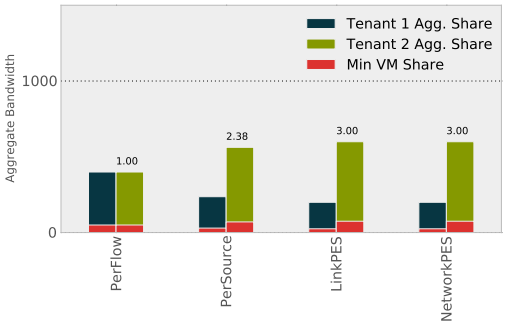
(b) $(P_1 : P_2 = 11 : NN)$, $(W_1 : W_2 = 1 : 1)$



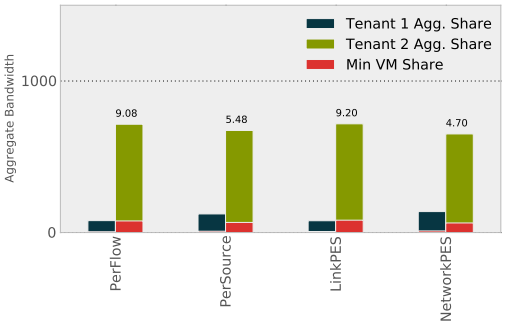
(c) $(P_1 : P_2 = NN : 11)$, $(W_1 : W_2 = 1 : 1)$



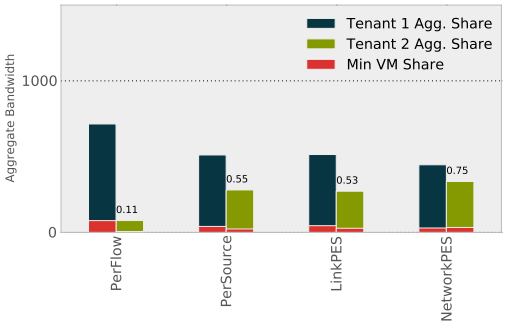
(d) $(P_1 : P_2 = 11 : 11)$, $(W_1 : W_2 = 1 : 1)$



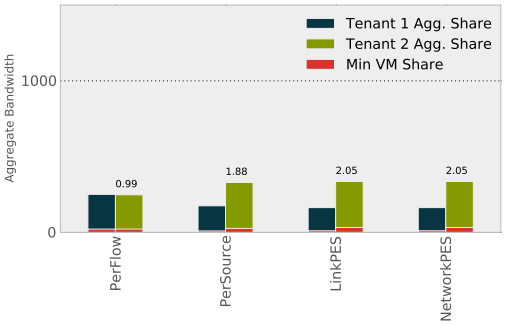
(e) $(P_1 : P_2 = NN : NN)$, $(W_1 : W_2 = 1 : 3)$



(f) $(P_1 : P_2 = 11 : NN)$, $(W_1 : W_2 = 1 : 3)$

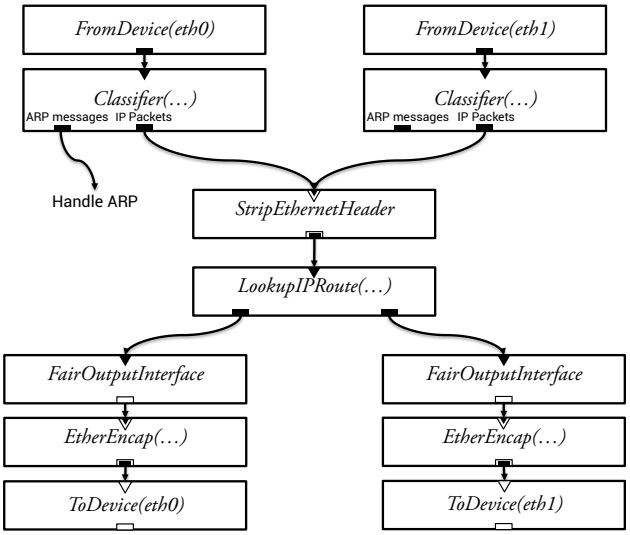


(g) $(P_1 : P_2 = NN : 11)$, $(W_1 : W_2 = 1 : 3)$

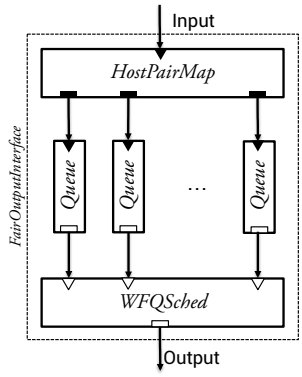


(h) $(P_1 : P_2 = 11 : 11)$, $(W_1 : W_2 = 1 : 3)$

Figure 6: Simulated aggregate bandwidths of two tenants on a eight-node three-level tree with oversubscription ratios $1 : 2$ in the aggregation level and another $1 : 2$ in the core level (*i.e.*, all links are 100 Mbps duplex). Each tenant has one VM in each host. P_i and W_i denote the communication pattern and weight of Tenant i, respectively.



(a) Click implementation of WFQ-enabled router



(b) Internal details of *FairOutputInterface* with per-flow queues

Figure 8: Click implementation of the WFQ router used to evaluate *NetworkPES* in the DETERlab testbed.

vide a different set of properties than those that use per VM weights. The number of tenant VMs that communicate over a congested link varies across links and is not necessarily reflected by the per tenant weight, which is a network wide constant. Thus, the properties achieved by a placement agnostic network sharing mechanism are more difficult to understand.

8. CONCLUSION AND FUTURE WORK

In this paper we addressed the problem of sharing cloud networks in a fair manner. To this end, we enumerated a set of desirable properties for achieving fairness and identified several tradeoffs between some of these properties. We observed that the tradeoff between *envy-freeness* and *work conservation* is strict and proposed *Per Endpoint Sharing*, a flexible mechanism that allows us to select different points in the solution space considering the tradeoffs between conflicting properties.

We want to naturally extend this model to MapReduce workloads, considering each job to be a separate tenant. Assuming a unit weight for each of the MapReduce slots, today, a job with M map tasks and R reduce tasks is given total share proportional to $M \times R$ during the shuffle phase since it has $M \times R$ flows; however, the total weight of the job's resources is $M + R$. This implies that network allocations are quadratic, and thus bigger jobs (with more slots) get unfair advantage over small jobs (with less slots). We plan to study the implications of our mechanism in this setting, and whether it can help penalize small jobs linearly instead of by a quadratic factor that exists today.

9. REFERENCES

[1] Amazon web services. `http://aws.amazon.com`.
[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*. ACM, 2008.
[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.
[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *ACM SIGCOMM*, 2011.
[5] D. P. Bertsekas and R. Gallager. *Data networks (2. ed.)*. Prentice Hall, 1992.
[6] B. Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 2007.
[7] N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *SIGCOMM*, 1999.
[8] A. Ghodsi, M. Zaharia, B. Hindman, et al. Dominant resource fairness: fair allocation of multiple resource types. In *USENIX NSDI*, 2011.
[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. *ACM SIGCOMM*, August 17 - 21 2009.
[10] C. Guo and G. Lu *et al.* BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. *ACM SIGCOMM*, 2009.
[11] C. Guo and G. Lu *et al.* Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *CoNEXT*. ACM, 2010.
[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. In *SIGCOMM*, 2008.

[13] Internet Draft. RFC2475 - An Architecture for Differentiated Services, 1998.

[14] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese. NetShare: Virtualizing Data Center Networks across Services. *Technical Report, UCSD*, 2010.

[15] C. Raiciu and S. Barre *et al.* Improving Datacenter Performance and Robustness with Multipath TCP. In *ACM SIGCOMM*, 2011.

[16] H. Rodrigues, J. R. Santos, Y. Turner, et al. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *USENIX WIOV*, 2011.

[17] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the Data Center Network. In *Usenix NSDI*, 2011.

[18] X. Yang, D. J. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *ACM SIGCOMM*, 2005.

[19] L. Zhang, S. E. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: A new resource reservation protocol. *IEEE Network*, 1993.