Chatty Tenants and the Cloud Network Sharing Problem

Hitesh Ballani[†] Keon Jang[†] Thomas Karagiannis[†]
Changhoon Kim[‡] Dinan Gunawardena[†] Greg O'Shea[†]

[†]*Microsoft Research* [‡]*Windows Azure*
Cambridge, UK *USA*

Abstract

The emerging ecosystem of cloud applications leads to significant inter-tenant communication across a datacenter's internal network. This poses new challenges for cloud network sharing. Richer inter-tenant traffic patterns make it hard to offer minimum bandwidth guarantees to tenants. Further, for communication between economically distinct entities, it is not clear whose payment should dictate the network allocation.

Motivated by this, we study how a cloud network that carries both intra- and inter-tenant traffic should be shared. We argue for network allocations to be dictated by the least-paying of communication partners. This, when combined with careful VM placement, achieves the complementary goals of providing tenants with minimum bandwidth guarantees while bounding their maximum network impact. Through a prototype deployment and large-scale simulations, we show that minimum bandwidth guarantees, apart from helping tenants achieve predictable performance, also improve overall datacenter throughput. Further, bounding a tenant's maximum impact mitigates malicious behavior.

1 Introduction

As cloud platforms mature, applications running in cloud datacenters increasingly use other cloud-based applications and services. Some of these services are offered by the cloud provider; for instance, Amazon EC2 offers services like S3, EBS, DynamoDB, RDS, SQS, CloudSearch, etc. that tenants can use as application building blocks [1]. Other services like CloudArray and Alfresco are run by tenants themselves [2]. The resulting ecosystem of applications and services means that, apart from communication between virtual machines of the same tenant, there is an increasing amount of tenant-tenant and tenant-provider communication [3]. Indeed, examining several datacenters of a major cloud provider, we find that such inter-tenant traffic can amount up to 35% of the total datacenter traffic. Looking ahead, we expect this ecosystem to become richer, further diversifying network traffic in the cloud.

The increasing importance and diversity of network traffic in cloud datacenters is at odds with today's cloud platforms. While tenants can rent virtual machines (VMs) with dedicated cores and memory, the underlying network is shared. Consequently, tenants experience variable and unpredictable network performance [4–6] which, in turn, impacts both application performance and tenant costs [5,7,8]. To tackle this, many network sharing policies have been proposed [9–14]. In recent work, FairCloud [14] presented a set of requirements for network sharing: i). associate VMs with minimum bandwidth guarantees, ii). ensure high network utilization, and iii). divide network resources in proportion to tenant payments. However, the proposals above focus only on intra-tenant communication, and naively extending these requirements to inter-tenant settings is problematic.

Inter-tenant traffic changes the network sharing problem both quantitatively and qualitatively, and leads to two main challenges. First, offering (non-trivial) minimum bandwidth guarantees for inter-tenant traffic is harder as the set of VMs that can possibly communicate with each other is significantly larger. Second, traffic flowing between distinct economic entities begs the question– whose payment should the bandwidth allocation be proportional to? Extending intra-tenant proportionality [14] to inter-tenant scenarios entails that bandwidth should be allocated in proportion to the combined payment of communicating partners. However, this is inadequate as tenants can increase their network allocation, beyond what their payment dictates, by communicating with more VMs of other tenants. Thus, the challenge is how to achieve proportional yet robust network sharing in inter-tenant scenarios.

Motivated by these challenges, we revisit the requirements for sharing a cloud network. To address the first

challenge of providing minimum bandwidth guarantees, we propose relaxing the semantics of the guarantees offered. For example, instead of targeting arbitrary communication patterns, a VM is only guaranteed bandwidth for intra-tenant traffic and for traffic to tenants it depends upon. Such *communication dependencies* could be explicitly declared or inferred. To address the second challenge, we identify a new network sharing requirement, *upper-bound proportionality*, which requires that the maximum bandwidth a tenant can acquire is in proportion to its payment. This mitigates aggressive tenant behavior and ensures robust network sharing. We show this requirement can be met by allocating bandwidth to flows in proportion to the *least-paying* communication partner. We call this "Hose-compliant" allocation.

To illustrate these ideas, we design *Hadrian*, a network sharing framework for multi-tenant datacenters. With Hadrian, VMs are associated with a minimum bandwidth. This minimum guarantee and tenant dependencies guide the placement of VMs across the datacenter. Network bandwidth is allocated using the hose-compliant allocation policy. This, when combined with careful VM placement, achieves the complementary goals of providing tenants with minimum bandwidth while achieving upper-bound proportionality.

As a proof of concept, we have implemented a Hadrian prototype comprising an end-host and a switch component. Through testbed deployment and cross-validated simulation experiments, we show that Hadrian benefits both tenants and providers. Minimum VM bandwidth yields predictable and better network performance for tenants (at the 95^{th} percentile, flows finish 3.6x faster). For the provider, such guarantees improve datacenter throughput up to 20% by avoiding outliers with very poor network performance. Thus, providers can offer an improved service at a lower price while remaining revenue neutral.

Overall, our main contributions are–

- We provide evidence of the prevalence of inter-tenant traffic through measurements from eight datacenters of a major public cloud provider.
- We present a new definition of payment proportionality that ensures robust network sharing in inter-tenant scenarios. We also devise a bandwidth allocation policy that meets this proportionality.
- We present relaxed bandwidth guarantee semantics to improve the multiplexing a provider is able to achieve. Further, we design a novel VM placement algorithm that uses a max-flow network formulation to satisfy such guarantees.
- To illustrate the feasibility of the mechanisms above, we present the design and implementation of Hadrian.

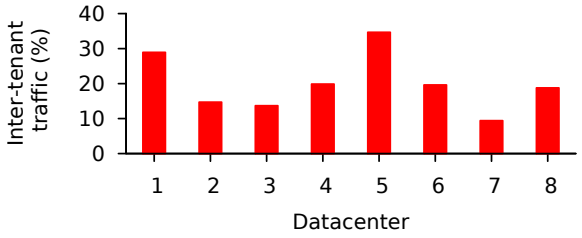


Figure 1: Inter-tenant traffic, as a % of the datacenter's total traffic, for eight production datacenters.

2 Motivation and challenges

With Infrastructure-as-a-Service, tenants rent VMs with varying amount of CPU, memory and storage, and pay a fixed flat rate per-hour for each VM [15,16]. However, the cloud's internal network is shared amongst the tenants. The cloud network carries *external* traffic to and from the Internet, *intra-tenant* traffic between a tenant's VMs and *inter-tenant* traffic. The latter includes traffic between different customer tenants, and between customer tenants and provider services. In the rest of this paper, we use the term "tenant" to refer to both customer tenants and provider services.

In this section, we show the prevalence of inter-tenant traffic in datacenters, and use this to drive our design.

2.1 Inter-tenant communication

Today's cloud providers offer many services that tenants can use to compose their cloud applications. For example, Amazon AWS offers sixteen services that result in network traffic between tenant VMs and service instances [1]. These services provide diverse functionality, ranging from storage to load-balancing and monitoring. Over a hundred cloud applications using such services are listed here [17]. Beyond provider services, many tenants run cloud applications that provide services to other tenants. AWS marketplace [2] lists many such tenant services, ranging from web analytics to identity and content management. These result in tenant-tenant traffic too.

Chatty tenants. To quantify the prevalence of inter-tenant traffic in today's datacenters, we analyze aggregate traffic data collected from eight geographically distributed datacenters operated by a major public cloud provider. Each datacenter has thousands of physical servers and tens of thousands of customer VMs. The data was collected from August 1-7, 2012 and includes all traffic between customer VMs. This does not include traffic to and from provider services. The total volume of such traffic in each datacenter was a few hundred terabytes to a few petabytes. Figure 1 shows that the percentage of inter-tenant traffic varies from 9 to 35%. When external traffic to or from the Internet is excluded, inter-tenant traffic varies from 10 to 40%.

We also studied traffic between customer tenants and

a specific provider service, the storage service. Since the local disk on a VM only provides ephemeral storage, for persistent storage, all tenants rely on the storage service which needs to be accessed across the network. The resulting network traffic is inter-tenant too. By analyzing application-level logs of the storage service, we found that read and write traffic for the storage service is, on average, equal to 36% and 70% of the total traffic between customer VMs respectively.

We complement these public cloud statistics with data from a private cloud with ∼300 servers. This represents a mid-size enterprise datacenter. The servers run over a hundred applications and production services, most serving external users. In summary, we find that 20% of the servers are involved in inter-tenant communication. For these servers, the fraction of inter-tenant to total flows is 6% at the median and 20% at the 95^{th} percentile with a maximum of 56% (3% at the median and 10% at the 95^{th} percentile in terms of volume). A tenant communicates with two other tenants in the median case and six at the 95^{th} percentile. Further, this inter-tenant traffic is not sporadic as it is present even at fine timescales.

Overall we find that tenants are indeed chatty with a significant fraction of traffic between tenants.

Impact of network performance variability. Several studies have commented on variable network performance in datacenters [4–6]. This impacts both provider and tenant services. For example, the performance of the cloud storage service varies both over time and across datacenters [18,19]. Any of the resources involved can cause this: processing on VMs, processing or disks at the storage tier, or the cloud network. For large reads and writes, the network is often the bottleneck resource. For example, Ghosal et al. [18] observed a performance variability of >2x when accessing Amazon EBS from large VMs (and far greater variability for small VMs), a lot of which can be attributed to cloud network contention.

In summary, these findings highlight the significance of inter-tenant traffic in today's datacenters. As cloud service marketplaces grow, we expect an even richer ecosystem of inter-tenant relationships. Offering service-level agreements in such settings requires network guarantees for inter-tenant communication.

2.2 Cloud network sharing

The distributed nature of the cloud network makes it tricky to apportion network bandwidth fairly amongst tenants. Today, network bandwidth is allocated through end host mechanisms such as TCP congestion control which ensure per-connection fairness. This has a number of drawbacks. For instance, misbehaving tenants can unfairly improve their network performance by using multiple TCP connections [12] or simply using UDP. Consequently, the cloud network sharing problem has received a lot of attention [9–14]. This section describes how inter-tenant traffic adds a new dimension to this problem.

2.2.1 Network sharing requirements

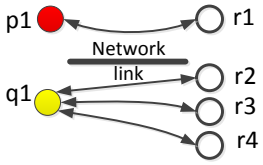
We first discuss how a cloud network carrying only intra-tenant traffic should be shared. FairCloud [14] presented the following broad set of requirements for fairly sharing the cloud network with a focus on intra-tenant traffic.
(1). **Min-Guarantee**: Each VM should be guaranteed a minimum bandwidth. This allows tenants to estimate worst-case performance and cost for their applications.
(2). **High Utilization**: Cloud datacenters multiplex physical resources across tenants to amortize costs and the same should hold for the network. So, spare network resources should be allocated to tenants with demand (work conservation). Further, tenants should be incentivised to use spare resources.
(3). **Proportionality**: Just like CPU and memory, the network bandwidth allocated to a tenant should be proportional to its payment.

However, inter-tenant traffic has important implications for these sharing requirements. As explained below, such traffic makes it harder to offer minimum bandwidth guarantees and necessitates a different kind of proportionality. Overall, *we embrace the first and second requirements, and propose a new proportionality requirement suitable for inter-tenant settings*.

2.2.2 Implications of inter-tenant traffic

Min-guarantee. Guaranteeing the minimum bandwidth for a VM requires ensuring sufficient capacity on all network links the VM's traffic can traverse. For intra-tenant traffic, this is the set of network links connecting a tenant's VMs. However, for inter-tenant traffic, the set expands to network links between VMs of all tenants that may communicate with each other. If we assume no information about a tenant's communication partners, the minimum bandwidth for each VM needs to be carved on all network links, and is thus strictly limited by the capacity of the underlying physical network. For instance, consider a datacenter with a typical three-tier tree topology with a 1:4 oversubscription at each tier (i.e., core links are oversubscribed by 1:64). If each physical server has 4 VMs and 1 Gbps NICs, such naive provisioning would provide each VM with a minimum guarantee of a mere 4 Mbps (≈ 1000/(4*64))! Hence, *richer traffic patterns resulting from inter-tenant communication make it harder to guarantee minimum bandwidth for VMs*.

Payment proportionality. Defining payment proportionality for inter-tenant settings is tricky. Since traffic can flow between different tenants, a key question is whose payment should dictate the bandwidth it gets. Intra-tenant proportionality requires that a tenant be allocated bandwidth in proportion to its payment. A simple



(a)

Allocation	P	Q	R
per-flow	250	750	1000
per-src (→)	500	500	1000
per-src (←)	250	750	1000
PS-L	333	666	1000
PS-P	250	750	1000

(b)

Figure 2: Inter-tenant traffic: Tenants P, Q and R have one ($p1$), one ($q1$) and four VMs respectively.

extension to inter-tenant settings entails that if a set of tenants are communicating with each other, their combined bandwidth allocation should be proportional to their combined payment. Assuming tenants pay a fixed uniform price for each VM, this means that a tenant's bandwidth should be in proportion to the total number of VMs involved in its communication (its own VMs and VMs of other tenants too).

As an example, consider the inter-tenant scenario shown in Figure 2. Since the traffic for tenants P and Q involves 2 and 4 VMs respectively, such proportionality requires that tenant Q's bandwidth be twice that of P, i.e., $\frac{B_Q}{B_P} = \frac{4}{2}$. Similarly, $\frac{B_R}{B_P} = \frac{6}{2}$ and $\frac{B_R}{B_Q} = \frac{6}{4}$. Further, the high utilization requirement entails that the link's capacity (1000 Mbps) should be fully utilized, i.e., $B_P + B_Q = B_R = 1000$. An allocation where $B_P = 333$, $B_Q = 666$ and $B_R = 1000$ satisfies these requirements.

While past proposals for network sharing have all focused on intra-tenant traffic, we consider how they would fare in this inter-tenant scenario. Figure 2b shows the bandwidth for tenants P, Q and R with different allocation strategies. Per-flow allocation ensures each flow gets an equal rate. Here, "flow" refers to the set of connections between a given pair of VMs. Hence tenant P, with its one flow, gets a quarter of the link's capacity, i.e., $B_P = 250$. Per-source allocation [12] gives an equal rate to each source, so the bandwidth allocated depends on the direction of traffic. PS-L and PS-P are allocation strategies that assign bandwidth in a weighted fashion with carefully devised weights for individual flows [14]. As the table shows, only PS-L, which was designed to achieve intra-tenant proportionality, satisfies the extended proportionality definition too.

However, *such proportionality means that a tenant can acquire a disproportionate network share by communicating with more VMs of other tenants*. For example, in Figure 2b, all approaches result in a higher bandwidth for tenant Q than P because Q is communicating with more VMs, even though both P and Q pay for a single VM and are using the same service R. Q could be doing this maliciously or just because its workload involves more communication partners. Further, Q can increase its share of the link's bandwidth by communicating with even more VMs. This key property leads to two problems. First, this makes it infeasible to guarantee a minimum bandwidth for any VM. In effect, such proportionality is incompatible with the min-guarantee requirement. Second, it allows for network abuse. An aggressive tenant with even a single VM can, simply by generating traffic to VMs of other tenants, degrade the performance for any tenants using common network links. The presence of many cloud services that are open to all tenants makes this a real possibility.

The root cause for these problems is that, with current network sharing approaches in inter-tenant settings, there is no limit on the fraction of a link's bandwidth a VM can legitimately acquire. To avoid such unbounded impact, we propose a new network sharing requirement.

Requirement 3. Upper bound proportionality: The maximum bandwidth each tenant and each VM can acquire should be a function of their payment. Further, this upper bound should be independent of the VM's communication patterns. Later we show that, apart from mitigating tenant misbehavior, *this new proportionality definition is compatible with the min-guarantee requirement*. It actually facilitates offering minimum VM bandwidth.

3 Revisiting network sharing

Guided by the observations above, we study how a cloud network that carries both intra- and inter-tenant traffic should be shared. The sharing should meet three requirements: (i). Minimum bandwidth guarantees, (ii). High Utilization, and (iii). Upper bound proportionality.

3.1 Minimum bandwidth guarantee

The cloud provider may allow tenants to specify the minimum bandwidth for individual VMs. So each VM p is associated with a minimum bandwidth B_p^{min}. Alternatively, the provider may offer a set of VM classes with varying guarantees (small, medium, and large, as is done for other resources today). In either case, a VM's minimum guarantee should dictate its price.

Like past proposals [10,11,14,20,21], we use the hose model to capture the semantics of the bandwidth guarantees being offered. As shown in Figure 3, with this model a tenant can imagine each of its VMs is connected to an imaginary, non-blocking switch by a link whose capacity is equal to the VM's minimum bandwidth. For simplicity, we assume all VMs for a tenant have the same minimum bandwidth.

However, as described in §2.2.2, richer inter-tenant communication patterns severely limit that the provider's ability to accommodate many concurrent tenants with minimum bandwidth guarantees atop today's oversubscribed networks. We show this in our experiments too. To better balance the competing needs of tenants and providers, we propose relaxing the bandwidth guarantees offered to tenants; they should be reasonable for tenants yet provider friendly. To achieve this, we rely on: i).

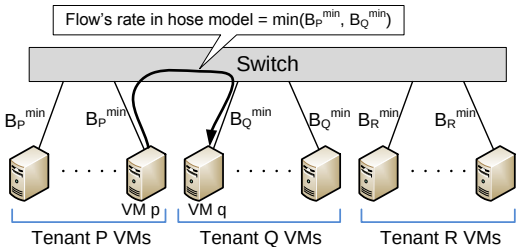


Figure 3: Hose model with three tenants.

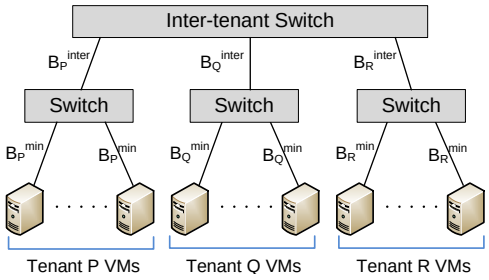


Figure 4: Hierarchical hose model gives per-VM minimum bandwidth for intra-tenant traffic and per-tenant minimum for inter-tenant traffic.

communication dependencies, and ii). hierarchical guarantees. We elaborate on these below.

3.1.1 Communication dependencies

Allowing arbitrary VMs to communicate under guaranteed performance is impractical. Instead, our guarantees apply only for "expected" communication. To this end, we rely on communication dependencies. A tenant's communication dependency is a list of other tenants or peers that the tenant expects to communicate with. Examples of such dependencies include: i) P: {Q}, ii) Q: {P, R}, iii) R: {*}.

The first dependency is declared by tenant P and implies that VMs of P, apart from sending traffic to each other, should be able to communicate with VMs of tenant Q. The second dependency is for Q and declares its peering with tenants P and R. Since a tenant running a service may not know its peers a priori, we allow for wildcard dependencies. Thus, the last dependency implies that tenant R is a service tenant and can communicate with any tenant that explicitly declares a peering with R (in this example, tenant Q). Note however that since P has not declared a peering with R, communication between VMs of P and R is not allowed.

The provider can use these dependencies to determine what inter-tenant communication is allowed and is thus better positioned to offer bandwidth guarantees to tenants. In the example above, the communication allowed is $P \leftrightarrow Q$ and $Q \leftrightarrow R$. An additional benefit is that this makes the cloud network "default-off" [22] since traffic can only flow between a pair of tenants if both have declared a peering with each other. This is in contrast to the "default-on" nature of today's cloud network.

We admit that discovering communication dependencies is challenging. While tenants could be expected to declare their dependencies when asking for VMs, a better option may be to infer them automatically. For example, today tenants need to sign up for provider services, so such tenant-provider dependencies are known trivially. The same mechanism could be extended for third-party services.

3.1.2 Hierarchical guarantees

With the hose model, each VM gets a minimum guarantee for all its traffic, irrespective of whether the traffic is destined to the same tenant or not. However, when accessing other tenants and services, tenants may find it easier to reason about an aggregate guarantee for all their VMs. Further, typical cloud applications involve more communication between VMs of the same tenant than across tenants. This was observed in our traces too. To account for these, we introduce hierarchical guarantees. Figure 4 shows the hierarchical hose model that captures such guarantees. Each VM for tenant P is guaranteed a bandwidth no less than B_P^{min} for traffic to P's VMs. Beyond this, the tenant also gets a minimum bandwidth guarantee for its aggregate inter-tenant traffic, B_P^{inter}.

Putting these modifications together, we propose offering tenants hose-style guarantees combined with communication dependencies and hierarchy. Hence, a tenant requesting V VMs is characterized by the four tuple $<V, B^{min}, B^{inter}, dependencies>$.[1] We show in §5.1 how this allows the provider to achieve good multiplexing while still offering minimum bandwidth to tenants.

3.2 Upper bound proportionality

Upper bound proportionality seeks to tie the maximum bandwidth a tenant can acquire to its payment. The same applies for individual VMs. Since a VM's minimum bandwidth dictates its price, it can be used as a proxy for payment. Thus, such proportionality requires that the upper bound on the aggregate bandwidth allocated to a VM should be a function of its minimum bandwidth. In this section, we describe a bandwidth allocation policy that achieves such proportionality.

3.2.1 Hose-compliant bandwidth allocation

Allocating bandwidth to flows in proportion to the combined payment of communicating partners results in tenants being able to get a disproportionate share of the network. To avoid this, we argue for bandwidth to be allocated in proportion to the least paying of communication partners. In other words, the bandwidth allocated to a flow should be limited by both source and destination

[1] Typically $B^{inter} < V * B^{min}$. If $B^{inter} = V * B^{min}$, no hierarchy is used and VMs simply get the same minimum bandwidth for all their traffic.

payment. For example, consider a flow between VMs p and q belonging to tenants P and Q. The minimum bandwidth for these VMs is B_P^{min} and B_Q^{min}, and they have a total of N_p and N_q flows respectively. Note that B_P^{min} reflects the payment for VM p. Assuming a VM's payment is distributed evenly amongst its flows, p's payment for the p–q flow is B_P^{min}/N_p. Similarly, q's payment for the flow is B_Q^{min}/N_q. Hence, this allocation policy says the flow should be allocated bandwidth in proportion to the smaller of these values, i.e., $\min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$.[2]

To achieve this, we assign appropriate weights to flows and allocate them network bandwidth based on weighted max-min fairness. So the bandwidth for a flow between VMs p and q, as determined by the bottleneck link along its path, is given by

$$B_{p,q} = \frac{w_{p,q}}{w_T} * C, \quad \text{where } w_{p,q} = min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q}) \quad (1)$$

Here, $w_{p,q}$ is the weight for this flow, C is the capacity of the bottleneck link and w_T is the sum of the weights for all flows across the link. Note that the weight for a flow is equal to the rate the flow would achieve on the hose model. Hence, we call this allocation "*Hose-compliant*".

Below we discuss how hose-compliance leads to upper bound proportionality and can also meet the other two network sharing requirements.

Req 3. Upper-bound Proportionality. Hose-compliant bandwidth allocation satisfies upper bound proportionality. The intuition here is that since the weight for each flow is limited by both the source and destination payments, the aggregate weight for a VM's flows and hence, its aggregate bandwidth has an upper bound. Formally, the aggregate weight for a VM p's flows–

$$w_p^{aggregate} = \sum_{q \in dst(p)} w_{p,q} = \sum_q min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$$

$$\Rightarrow w_p^{aggregate} \leq \sum_q \frac{B_P^{min}}{N_p} = \frac{B_P^{min}}{N_p} * N_p = B_P^{min} \quad (2)$$

So the aggregate weight for a VM's flows cannot exceed its minimum bandwidth. This aggregate weight, in turn, dictates the VM's aggregate bandwidth. This means that a tenant cannot acquire bandwidth disproportionate to its payment. More precisely, this yields the following constraint for a VM's total bandwidth on any link–

$$B_p = \sum_{q \in dst(p)} \frac{w_{p,q}}{w_T} * C \; = \; \frac{w_p^{aggregate}}{w_T} * C \; \leq \; \frac{B_P^{min}}{B_P^{min} + w_{T'}} * C$$

where B_p is the total bandwidth for VM p on the link, $w_{T'}$ is the sum of weights for all non-p flows and C is the link capacity. *Hence, hose-compliant allocation results in an upper bound for a VM's bandwidth on any link. This upper bound depends on the VM's minimum bandwidth (and hence, its payment).*

To understand this, let's revisit the inter-tenant scenario in Figure 2. Assume a minimum bandwidth of 100 Mbps for all VMs. With hose-compliant allocation, the weight for the $p1$–$r1$ flow is $\min(\frac{100}{1}, \frac{100}{1}) = 100$ while the weight for $q1$–$r2$ flow is $\min(\frac{100}{3}, \frac{100}{1}) = \frac{100}{3}$. Similarly, the weight for the $q1$–$r3$ and $q1$–$r4$ flow is $\frac{100}{3}$ too. Hence, the actual bandwidth for the $p1$–$r1$ flow is 500, while the other three flows get $\frac{500}{3}$ each. Note that even though tenant Q has three flows, their aggregate weight is the same as the weight for P's single flow. Hence, both tenants get the same bandwidth. This is desirable as both of them pay for a single VM. Even if tenant Q were to communicate with more VMs, the aggregate weight of its flows will never exceed 100. Thus, by bounding the aggregate weight for a VM's traffic, we ensure an upper bound for the impact it can have on any network link and hence, on the datacenter network.

Req 1. Min-guarantee. Minimum guarantees for VMs can be used to determine the minimum bandwidth that their flows should achieve. For example, in Figure 3, if VMs p and q communicate with N_p and N_q VMs each, then the bandwidth for a p–q flow should be at least $\min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$. With hose-compliant allocation, this is also the flow's weight $w_{p,q}$. This observation simplifies ensuring that flows do get their minimum bandwidth.

To ensure a flow's actual bandwidth always exceeds its guarantee, the total weight for all traffic that can traverse a link should not exceed its capacity. Formally, to ensure $B_{p,q} \geq w_{p,q}$, we need $w_T \leq C$ (see equation 1). This condition can be used to design a VM placement algorithm that ensures the condition holds across all links in the datacenter. §4.1 presents such an algorithm.

Req 2. High utilization. Hose-compliant allocation is work conserving. Since flows are assigned bandwidth in a weighted fashion, any VM with network demand is allowed to use spare capacity on network links. Beyond work conservation, high utilization also requires that tenants not be disincentivised to use spare bandwidth. This can be achieved by making the flow weights vary from link-to-link, as proposed in [14]. However, for brevity, we omit this extension in the rest of the paper.

4 Hadrian

Apart from a policy for bandwidth allocation, a complete network sharing solution has to include an admission control and VM placement mechanism to achieve proper network sharing. Guided by this, we design Hadrian, a

[2] A VM may favor some flows over others and choose to distribute its payment unevenly across its flows. This can be used by service providers to offer differentiated services to their clients. The allocation policy can accommodate such scenarios.

network sharing framework for multi-tenant datacenters that caters to both intra- and inter-tenant communication. Hadrian relies on the following two components.

• *VM Placement*. A logically centralized placement manager, upon receiving a tenant request, performs admission control and maps the request to datacenter servers. This allocation of VMs to physical servers accounts for the minimum bandwidth requirements and communication dependencies of tenants.

• *Bandwidth Allocation*. Hose-compliant allocation is used to assign network bandwidth to flows.

4.1 VM placement

VM placement problems are often mapped to multi-dimensional packing with constraints regarding various physical resources [23]. Our setting involves two resources– each tenant requires empty VM slots on physical servers and minimum bandwidth on the network links connecting them. *The key novelty in our approach is that we model minimum bandwidth requirements and communication dependencies of tenants as a max-flow network*. This allows us to convert our two-dimensional placement constraints into a simple set of constraints regarding the number of VMs that can be placed in a given part of the datacenter.

The placement discussion below focuses on tree-like physical network topologies like the multi-rooted tree topologies used today. Such topologies are hierarchical, made up of sub-trees at each level. Also, it assumes that if the topology offers multiple paths between VMs, the underlying routing protocol load balances traffic across them. This assumption holds for fat-tree topologies [24,25] that use multi-pathing mechanisms like ECMP, VLB [24] and Hedera [26].

4.1.1 Characterizing bandwidth requirements

Hose-compliant bandwidth allocation simplifies the problem of ensuring minimum VM bandwidth. As explained in §3.2.1, to satisfy the minimum guarantees of VMs, the provider needs to ensure the total weight for all traffic that can traverse any network link should not exceed the link's capacity. Thus, we need to quantify the total weight for traffic across any given link. To explain our approach, we use an example scenario involving three tenants, P, Q and R across any network link. The link has p VMs for tenant P to the left and the remaining p' VMs to the right. Similarly, there are q and r VMs for Q and R on the left, and q' and r' VMs on the right.

With hose-compliant bandwidth allocation, the aggregate weight for any VM's traffic cannot exceed its minimum guarantee (equation 2). So the total weight for traffic from all VMs on the left of the link cannot exceed the sum of their minimum bandwidths, i.e., $\sum(pB_P^{min} + qB_Q^{min} + rB_R^{min})$. The same holds for VMs on the right of the link. Further, since the weight for any given flow is limited by both its source and destination, the total weight for all traffic across the link is limited by both the total weight for VMs on the left and for VMs on the right of the link.

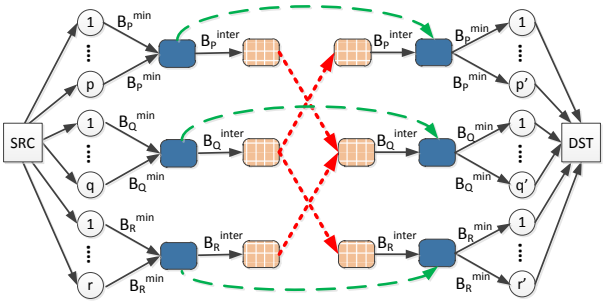


Figure 5: Flow network to capture the bandwidth needed on a link that connects p VMs of tenant P on the left to p' VMs on the right, and so on for tenants Q and R. Circles represent VMs, solid rectangles are intra-tenant nodes and shaded rectangles are inter-tenant nodes.

However, this analysis assumes all VMs can talk to each other and the same guarantees apply to both intra- and inter-tenant traffic. Accounting for communication dependencies and hierarchical guarantees leads to even more constraints regarding the total weight for the link's traffic. To combine these constraints, we express them as a **flow network**. A flow network is a directed graph where each edge has a capacity and can carry a flow not exceeding the capacity of the edge. Note that this flow is different from "real" flows across the datacenter network. Hereon, we use "link" to refer to physical network links while "edge" corresponds to the flow network.

Figure 5 shows the flow network for the link in our example and is explained below. All unlabeled edges have an infinite capacity. Each VM to the left of the physical link is represented by a node connected to the source node, while each VM to the right of the link is represented by a node connected to the destination. The VM nodes for any given tenant are connected to "intra-tenant" nodes (solid rectangles) by edges whose capacity is equal to the minimum bandwidth for the VM. These edges represent the constraint that the weight for a VM's traffic cannot exceed its minimum bandwidth. The two intra-tenant nodes for each tenant are connected by an edge of infinite capacity (long-dashed edge). Further, the two intra-tenant nodes for each tenant are connected to "inter-tenant" nodes (shaded rectangles) by edges whose capacity is equal to the tenant's minimum inter-tenant bandwidth. This constrains the total weight for inter-tenant traffic that each tenant can generate. Finally, based on the tenant communication dependencies, the appropriate inter-tenant nodes are connected to each

other (short-dashed edges). For our example, tenant Q can communicate with P and R, so inter-tenant nodes of Q are connected to those of P and R.

The max-flow for this flow network gives the total weight for all traffic across the link. This, in turn, is the *bandwidth required* on the physical link to ensure that the bandwidth guarantees of VMs are met.

4.1.2 Finding Valid Placements

Given a tenant request, a valid placement of its VMs should satisfy two constraints. First, VMs should only be placed on empty slots on physical hosts. Second, after the placement, the bandwidth required across each link in the datacenter should not exceed the link's capacity. The *VM Placement problem* thus involves finding a valid placement for a tenant's request. We designed a greedy, first-fit placement algorithm that we briefly sketch here. A formal problem definition, algorithm details and pseudo code are available in [27].

Instead of trying to place VMs while satisfying constraints across two dimensions (slots and bandwidth), we use the flow-network formulation to convert the bandwidth requirements on each physical link to constraints regarding the number of VMs that can be placed inside the sub-tree under the link, i.e., in the host, rack or pod under the link. Hence, given a tenant request, its placement proceeds as follows. We traverse the network topology in a depth-first fashion. Constraints for each level of the topology are used to recursively determine the maximum number of VMs that can be placed at sub-trees below the level, and so on till we determine the number of VMs that can be placed on any given host. VMs are then greedily placed on the first available host, and so on until all requested VMs are placed. The request is accepted only if all VMs can be placed.

A request can have many valid placements. Since datacenter topologies typically have less bandwidth towards the root than at the leaves, the optimization goal for the placement algorithm is to choose placements that reduce the bandwidth needed at higher levels of the datacenter hierarchy. To achieve this, we aim for *placement locality* which comprises two parts. First, a tenant's VMs are placed close to VMs of existing tenants that it has communication dependencies with. Second, the VMs are placed in the smallest sub-tree possible. This heuristic reduces the number and the height of network links that may carry the tenant's traffic. It preserves network bandwidth for future tenants, thus improving the provider's ability to accommodate them.

4.2 Bandwidth allocation

Hadrian uses hose-compliant bandwidth allocation. This can be achieved in various ways. At one end of the spectrum is an end-host only approach where a centralized controller monitors flow arrivals and departures to calculate the weight and hence, the rate for individual flows. These rates can then be enforced on physical servers. At the other end is a switch-only approach where switches know the VM bandwidth guarantees and use weighted fair queuing to achieve weighted network sharing. Both these approaches have drawbacks. The former is hard to scale since the controller needs to track the utilization of all links and all flow rates. The bursty nature of cloud traffic makes this particularly hard. The latter requires switches to implement per-flow fair queuing.

We adopt a hybrid approach involving end-hosts and switches. The goal here is to minimize the amount of network support required by moving functionality to trusted hypervisors at ends. Our design is based on explicit control protocols like RCP [28] and XCP [29] that share bandwidth equally and explicitly convey flow rates to end hosts. Hose-compliance requires weighted, instead of an equal, allocation of bandwidth. We provide a design sketch of our bandwidth allocation below.

To allow VMs to use any and all transport protocols, their traffic is tunneled inside hypervisor-to-hypervisor flows such that all traffic between a pair of VMs counts as one flow. Traffic is only allowed to other VMs of the same tenant and to VMs of peers. The main challenge is determining the rate for a flow which, in turn, is dictated by the flow's weight. The weight for a p–q flow is $\min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$. The source hypervisor hosting VM p knows B_P^{min} and N_p while the destination hypervisor knows B_Q^{min} and N_q. Thus, the source and destination hypervisor together have all the information to determine a flow's weight. For each flow, the hypervisor embeds B^{min} and N into the packet header, and over the course of the first round trip, the hypervisors at both ends have all the information to calculate the weights.

Packet headers also contain the flow weight. For the first round trip, hypervisors set the weight to a default value. Switches along the path only track the sum of the weights, S, for all flows through them. For a flow with weight w, its rate allocation on a link of capacity C is $\frac{w}{S} * C$. Each switch adds this rate allocation to the packet header. This reaches the destination and is piggybacked to the source on the reverse path. The source hypervisor enforces the rate it is allocated, which is the minimum of the rates given by switches along the path. To account for queuing or under-utilization due to insufficient demand, switches adjust C using the RCP control equation.

This basic design minimizes switch overhead; they do not need to maintain per-flow state. However, it does not support hierarchical guarantees. With such guarantees, the weight for a flow between different tenants depends on the total number of inter-tenant flows each of them have. Hence, hypervisors at the end of a flow do

not have enough information to determine its weight. Instead, switches themselves have to calculate the flow weight. In this extended design, the hypervisor also embeds the `VM-id`, the `tenant-id`, the inter-tenant bandwidth guarantee for the source and destination VM in the packet header. Each switch maintains a count of the number of inter-tenant flows for each tenant traversing it. Based on this, the switch can determine the weight for any flow and hence, its rate allocation. Thus switches maintain per-tenant state. The details for this extended design are available in [27].

4.3 Implementation

Our proof-of-concept Hadrian implementation comprises two parts.

(1). A placement manager that implements the placement algorithm. To evaluate its scalability, we measured the time to place tenant requests in a datacenter with 100K machines. Over 100K representative requests, the median placement time is 4.13ms with a 99^{th} percentile of 2.72 seconds. Note that such placement only needs to be run when a tenant is admitted.

(2). For bandwidth allocation, we have implemented an extended version of RCP (RCP_w) that distributes network bandwidth in a weighted fashion, and is used for the hypervisor-to-hypervisor flows. This involves an end-host component and a switch component.

Ideally, the end-host component should run inside the hypervisor. For ease of prototyping, our implementation resides in user space. Application packets are intercepted and tunneled inside RCP_w flows with a custom header. We have a kernel driver that binds to the Ethernet interface and efficiently marshals packets between the NIC and the user space RCP_w stack. The switch is implemented on a server-grade PC and implements a store and forward architecture. It uses the same kernel driver to pass all incoming packets to a user space process.

In our implementation, switches allocate rate to flows once every round trip time. To keep switch overhead low, we use integer arithmetic for all rate calculations. Although each packet traverses the user-kernel space boundary, we can sustain four 1Gbps links at full duplex line rate. Futher, experiments in the next section show that we can achieve a link utilization of 96%. Overall, we find that our prototype imposes minimal overhead on the forwarding path.

4.4 Design discussion

Other placement goals. Today, placement of VMs in datacenters is subject to many constraints like their CPU and memory requirements, ensuring fault tolerance, energy efficiency and even reducing VM migrations [30]. Production placement managers like SCVMM [31] use heuristics to meet these constraints. Our flow network formulation maps tenant network requirements to constraints regarding VM placement which can be added to the set of input constraints used by existing placement managers. We defer an exploration of such extensions to future work. We do note that our constraints can be at odds with existing requirements. For example, while bandwidth guarantees entail placement locality, fault tolerance requires VMs be placed in different fault domains.

Hose-compliant allocation. Allocating bandwidth in proportion to the least paying of communication partners has implications for provisioning of cloud services. A service provider willing to pay for VMs with higher minimum bandwidth (and higher weight) will only improve the performance of its flows that are bottlenecked by the weight contributed by the service VMs. Performance for flows bottlenecked by client VMs will not improve. This is akin to network performance across the Internet today. When clients with poor last mile connectivity access a well-provisioned Internet service, their network performance is limited by their own capacity. Allocating bandwidth based on the sum of payments of communicating partners avoids this but at the expense of allowing network abuse.

5 Evaluation

We deployed our prototype implementation across a small testbed comprising twelve end-hosts arranged across four racks. Each rack has a top-of-rack (ToR) switch, and the ToR switches are connected through a root switch. All switches and end-hosts are Dell T3500 servers with a quad core Intel Xeon 2.27GHz processor, 4GB RAM and 1 Gbps interfaces, running Windows Server 2008 R2. Given our focus on network performance, the tenants are not actually allocated VMs but simply run as a user process. With 8 VM slots per host, the testbed has a total of 96 slots. We complement the testbed experiments with large-scale simulations. For this, we developed a simulator that models a multi-tenant datacenter with a three tier network topology. All simulation results here are based on a datacenter with 16K hosts and 4 VMs per host, resulting in 64K VMs. The network has an oversubscription of 1:10.

Overall, our evaluation covers three main aspects: (i) We combine testbed and simulation experiments to illustrate that Hadrian, by ensuring minimum VM bandwidth, benefits both tenants and providers, (ii) We use simulations to quantify the benefits of relaxing bandwidth guarantee semantics, and (iii) We use testbed experiments to show hose-compliance mitigates aggressive behavior in inter-tenant settings.

5.1 Cloud emulation experiments

We emulate the operation of a cloud datacenter on our testbed (and in the simulator) as follows. We generate

Placement → B/w Allocation	Greedy	Dependency -aware	**Hadrian's placement**
Per-flow	*Baseline*	*Baseline+*	–
Hose-compliant	–	–	**Hadrian**
Reservations	–	–	Oktopus [11]
Per-source	*Seawall* [12]	–	*Seawall*
PS-L	*FairCloud* [14]	–	*FairCloud*

Table 1: Solution space for cloud network sharing

a synthetic workload with tenant requests arriving over time. A placement algorithm is used to allocate the requested VMs and if the request cannot be placed, it is rejected. The arrival of tenants is a Poisson process. By varying the rate at which tenants arrive, we control the *target VM occupancy* of the datacenter. This is the fraction of datacenter VMs that, on average, are expected to be occupied. As for bandwidth guarantees, tenants can choose from three classes for their minimum bandwidth– 50, 150 and 300 Mbps. By varying the fraction of tenant requests in each class, we control the *average minimum bandwidth* for tenants.

Tenants. We model two kinds of tenants– service tenants that have a wildcard (*) communication dependency and client tenants that depend on zero or more service tenants. Tenants request both VMs (V) and a minimum bandwidth (B^{min}). Each tenant runs a job involving network flows; some of these flows are intra-tenant while others are to VMs of service tenants. A job finishes when its flows finish. The fraction $F \in [0,1]$ of a tenant's flows that are inter-tenant allows us to determine the minimum bandwidth required by the tenant for inter-tenant communication. Overall, each tenant request is characterized by $<V, B^{min}, V*B^{min}*F$, *dependencies*$>$.

By abstracting away non-network resources, this simple workload model allows us to directly compare various network sharing approaches. While the workload is synthetic, we use our datacenter measurements to ensure it is representative of today's datacenters. For instance, the fraction of client tenants with dependencies (20%), the average number of dependencies (2), the fraction of inter-tenant flows (10-40%), and other workload parameters are as detailed in §2.1.

In the following sections, we compare Hadrian against alternate network sharing solutions. Since a complete network sharing framework ought to include both careful VM placement and bandwidth allocation, we consider various state of the art solutions for both–

VM Placement. We experiment with three placement approaches. (i) With *Greedy* placement, a tenant's VMs are greedily placed close to each other. (ii) With *Dependency-aware placement*, a tenant's VMs are placed close to each other and to VMs of existing tenants that the tenant has a dependency on. (iii) Hadrian's placement, described in §4.1, which is aware of tenant minimum bandwidths and their dependencies.

Bandwidth allocation. Apart from Per-flow, Per-source and PS-L sharing, we evaluate two other policies. (i) With hose-compliant allocation, bandwidth is allocated as described in §3.2. (ii) With "Reservations", VMs get to use their guaranteed bandwidth but no more. Hence, this allocation policy is not work conserving.

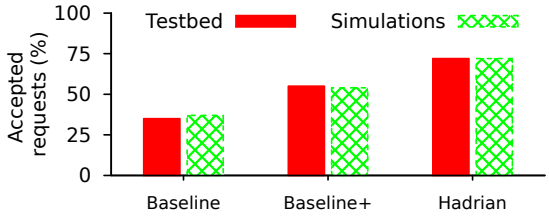


Figure 6: Accepted requests in testbed and simulator.

Table 1 summarizes the solution space for cloud network sharing. Note that by combining Hadrian's placement with Reservations, we can extend Oktopus [11] to inter-tenant settings. We begin by focussing on the approaches in the first two rows. The approach of placing tenant VMs greedily combined with the Per-flow sharing of the network reflects the operation of today's datacenters, and is thus used as a *Baseline* for comparison.

5.1.1 Testbed experiments

The experiment involves the arrival and execution of 100 tenant jobs on our testbed deployment. The average minimum bandwidth for tenants is 200 Mbps and requests arrive such that target VM occupancy is 75%. Note that operators like Amazon EC2 target an average occupancy of 70-80% [32]. Since our prototype uses weighted RCP, we emulate various bandwidth allocation policies by setting flow weights appropriately; for example, for Per-flow allocation, all flows have the same weight. Figure 6 shows that *Baseline* only accepts 35% of the requests, *Baseline+* accepts 55%, while Hadrian accepts 72%. This is despite the fact that Hadrian will reject a request if there is insufficient bandwidth while the other approaches will not. To show that Hadrian leads to comparable benefits at datacenter scale, we rely on large-scale simulations.

However, we first validate the accuracy of our simulator. To this end, we replayed the same set of jobs in the simulator. The figure also shows that the percentage of accepted requests in the testbed is similar to those accepted in the simulator; the difference ranges from 0-2%. Further, the completion time for 87% of the requests is the same across the testbed and simulator; at the 95^{th} percentile, requests are 17% faster in the simulator. This is because the simulator achieves perfect network sharing (e.g., no overheads). This gives us confidence in the fidelity of the simulation results below.

5.1.2 Large-scale simulations

We simulate a stream of 25K jobs on a datacenter with 16K servers. Figure 7(left) shows that, with Hadrian, the

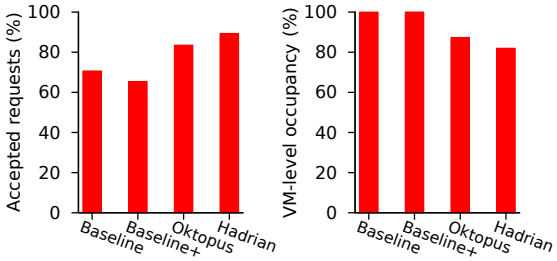


Figure 7: Provider can accept more requests with Hadrian. (average B^{min} = 200 Mbps)

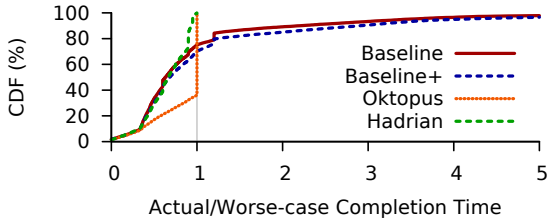


Figure 8: With *Baseline* (and *Baseline+*), many tenants receive poor network performance and finish past the worst-case completion time estimate.

provider is able to accept 20% more requests than both *Baseline* and *Baseline+*. We also simulate the use of hard reservations for allocating bandwidth (Oktopus), and find that Hadrian can still accept 6% more requests. Further, figure 7(right) shows the average VM occupancy during the experiment. With Hadrian, the average VM utilization is 87% as compared to 99.9% with *Baseline* and 90% with Oktopus. This is because jobs finish earlier with Hadrian. Thus, Hadrian allows the provider to accommodate more requests while reducing the VM-level utilization which, in turn, allows more future requests to be accepted.

To understand this result, we examine the performance of individual requests. Since tenants are associated with minimum bandwidths, each tenant can estimate the worst-case completion time for its flows and hence, its job. Figure 8 shows the CDF for the ratio of a job's actual completion time to the worst-case estimate. With Hadrian, all requests finish before the worst-case estimate. With Oktopus, tenants get their requested bandwidth but no more, so most requests finish at the worst-case estimate.[3] As a contrast, with *Baseline*, many tenants get very poor network performance. The completion time for 15% tenants is 1.25x the worst-case estimate and for 5% tenants, it is 3.4x the worst-case. These outliers occupy VMs longer, thus driving up utilization but reducing actual throughput. By ensuring minimum bandwidth for VMs and thus avoiding such outliers, Hadrian allows the provider to accept more requests.

[3]Requests that finish earlier with Oktopus have all their flows between co-located VMs in the same physical machine, hence achieving bandwidth greater than their reservation, so the job can finish early.

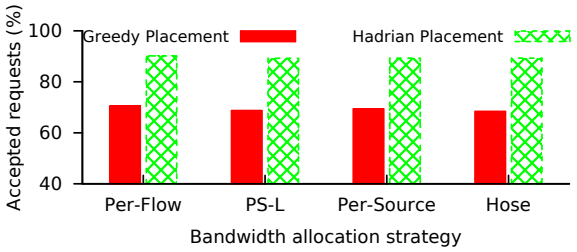
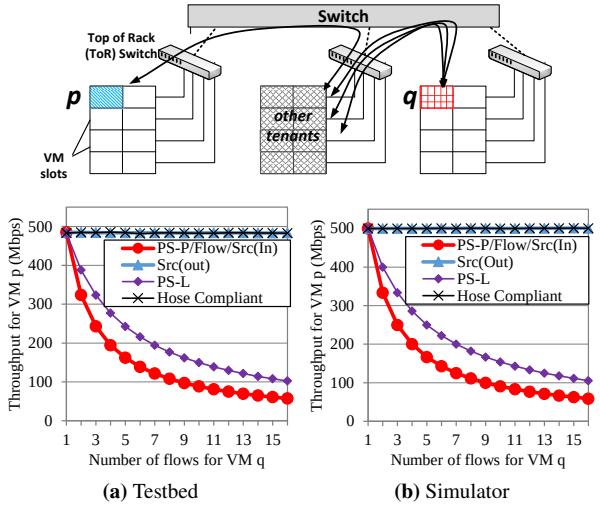


Figure 9: With non-aggressive tenants, Hadrian's placement provides most of the gains.

Beyond this, we also experimented with other values for the simulation parameters– the average minimum bandwidth, the target occupancy, the network oversubscription and the percentage of inter-tenant traffic. The results are along expected lines so we omit them for brevity but they are available in [27]. For example, Hadrian's gains increase as inter-tenant traffic increases and network oversubscription increases. Further, Hadrian can offer benefits even when there is no oversubscription. While it accepts the same number of requests as *Baseline*, 22% of requests are outliers with *Baseline*.

Cost analysis. Today's cloud providers charge tenants a fixed amount per hour for each VM; for instance, Amazon EC2 charges $0.08/hr for small VMs. Hence, the improved tenant performance with Hadrian has implications for their cost too. For the experiment above, the average tenant would pay 34% less with Hadrian than *Baseline*. This is because there are no outliers receiving very poor network performance. From the provider's perspective though, there are two competing factors. Hadrian allows them to accommodate more tenants but tenants finish faster and hence, pay less. We find the provider's revenue with Hadrian is 82% of that with *Baseline*. This reduction in revenue can be overcome by new pricing models that account for the added value Hadrian offers. Since Hadrian provides tenants with VMs that have minimum bandwidth, the provider can increase the price of VMs. We repeat the cost analysis to determine how much tenants would have to pay so that the provider remains revenue neutral and find that the average tenant would still pay 19% less. Overall, the results above show that Hadrian allows the provider to offer network guarantees while reducing the average tenant cost.

Importance of relaxed guarantee semantics. In the experiments above, we find that with simple hose guarantees where all tenants are assumed to speak to each other, the provider is only able to accept 1% of the requests! However, when the provider is aware of tenant dependencies, it can accept 85% of the requests. This is because bandwidth constraints need to enforced on far fewer links. Hierarchical guarantees allow the provider to accept a further 5% requests. These results highlight the importance of relaxed guarantee semantics.



(a) Testbed **(b)** Simulator

Figure 10: By sending and receiving more flows, q can degrade p's network performance.

Importance of placement. We now also consider the Per-source and PS-L allocation, and compare their performance, when coupled with Greedy and Hadrian's placement. Figure 9 shows that Hadrian's placement provides gains irrespective of how bandwidth is being allocated. Here, the allocation policy does not have much impact because *all tenants have the same traffic pattern*. Hence, under scenarios with well-behaved tenants, VM placement dictates the datacenter throughput. However, as we show in the next section, when some tenants are aggressive, the allocation policy does matter.

5.1.3 Benefits of hose-compliant allocation

To illustrate the benefits of hose-compliance relative to alternate bandwidth allocation policies, we focus on a simple scenario that captures aggressive tenant behavior. The experiment involves two tenants, each with one VM (p and q). As shown in Figure 10 (top), VM p has one flow while VM q has a lot of flows to VMs of other tenants. All flows are bottlenecked at the same link. VM q could be acting maliciously and initiating multiple flows to intentionally degrade p's network performance. Alternatively, it could just be running a popular service that sends or receives a lot of traffic which, in turn, can hurt p's performance. All VMs, including destination VMs, have a minimum bandwidth of 300 Mbps.

Figures 10a and 10b show the average bandwidth for VM p on the testbed and in the simulator respectively. With hose-compliant allocation, VM p's bandwidth remains the same throughout. As a contrast, with other policies, p's bandwidth degrades as q has more flows. This is because, with these approaches, there is no limit on the fraction of a link's bandwidth a VM can grab. This allows tenants to abuse the network at the expense of others. By bounding tenant impact, hose-compliant allocation addresses this. Note that with Per-source allocation, p's bandwidth does not degrade if both VMs are sending traffic (labelled as "out") but it does if they are receiving traffic (labelled as "in"). Instead, hose-compliant allocation is not impacted by the traffic direction.

The figures also show that the testbed results closely match simulation results for all policies. With hose-compliance, VM p's bandwidth on the testbed is 480 Mbps as compared to the ideal 500 Mbps. This shows our weighted RCP implementation is performant and can achieve a link utilization of 96%.

6 Related work

Many recent efforts tackle the cloud network sharing problem. They propose different sharing policies, including reservations [9,11], time-varying reservations [33], minimum bandwidth reservations [10,14,34], per-source fairness [12] and per-tenant fairness [13]. Mogul et al. [35] present a useful survey of these proposals. However, as detailed in this paper, none of these proposals explicitly target inter-tenant communication which poses its own challenges.

To achieve desirable network sharing, we have borrowed and extended ideas from many past proposals. The hose model [20] has been used to capture both reservations [11] and minimum bandwidth guarantees [14,21]. We extend it by adding communication dependencies and hierarchy. The use of hierarchy means that Hadrian offers aggregate, "per-tenant" minimum guarantees for inter-tenant traffic. This is inspired by Oktopus [11] and NetShare [13] that offer per-tenant reservations and weights respectively. CloudPolice [3] argues for network access control in inter-tenant settings. However, many cloud services today are open to tenants. Hence, network access control needs to be coupled with a robust network sharing mechanism like Hadrian.

7 Concluding remarks

Inter-tenant communication plays a critical role in today's datacenters. In this paper, we show this necessitates a rethink of how the cloud network is shared. To ensure provider flexibility, we modify the kind of bandwidth guarantees offered to tenants. To ensure robust yet proportional network sharing, we argue for coupling the maximum bandwidth allocated to tenants to their payment. Tying these ideas together, we propose Hadrian, a network sharing framework that uses hose-compliant allocation and bandwidth-aware VM placement to achieve desirable network sharing properties for both intra- and inter-tenant communication. Our evaluation shows that Hadrian's mechanisms are practical. Further, apart from improving the performance of both tenants and providers, it ensures robust network sharing.

References

[1] "Amazon AWS Products," http://aws.amazon.com/products/.

[2] "Amazon AWS Marketplace," http://aws.amazon.com/marketplace/.

[3] L. Popa, M. Yu, S. Ko, S. Ratnasamy, and I. Stoica, "CloudPolice: Taking access control out of the network," in *Proc. of ACM HotNets*, 2010.

[4] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proc. of ACM IMC*, 2010.

[5] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," in *Proc. of VLDB*, 2010.

[6] "Measuring EC2 performance," http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed.

[7] A. Iosup, N. Yigitbasi, and D. Epema, "On the Performance Variability of Cloud Services," Delft University, Tech. Rep. PDS-2010-002, 2010.

[8] E. Walker, "Benchmarking Amazon EC2 for high performance scientific computing," *Usenix Login*, vol. 33, October 2008.

[9] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. of ACM CoNEXT*, 2010.

[10] P. Soares, J. Santos, N. Tolia, and D. Guedes, "Gatekeeper: Distributed Rate Control for Virtualized Datacenters," HP Labs, Tech. Rep. HP-2010-151, 2010.

[11] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proc. of ACM SIGCOMM*, 2011.

[12] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Sharing the Datacenter Network," in *Proc. of Usenix NSDI*, 2011.

[13] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," University of California, San Diego, Tech. Rep. CS2010-0957, May 2010.

[14] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network In Cloud Computing," in *Proc of ACM SIGCOMM*, 2012.

[15] "Windows Azure Pricing," http://www.windowsazure.com/en-us/pricing/details.

[16] "Amazon EC2 Pricing," http://aws.amazon.com/ec2/pricing/.

[17] "Amazon Case Studies," http://aws.amazon.com/solutions/case-studies/.

[18] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/O performance of virtualized cloud environments," in *Proc. of DataCloud-SC*, 2011.

[19] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of Windows Azure," in *Proc. of HPDC*, 2010.

[20] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," in *Proc. of ACM SIGCOMM*, 1999.

[21] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "The Price Is Right: Towards Location-independent Costs in Datacenters," in *Proc. of ACM HotNets*, 2011.

[22] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, "Off by Default!" in *Proc. of ACM HotNets*, 2005.

[23] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating Heuristics for Virtual Machines Consolidation," MSR, Tech. Rep. MSR-TR-2011-9, 2011.

[24] 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," in *Proc. of ACM SIGCOMM*, 2009.

[25] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of ACM SIGCOMM*, 2008.

[26] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of Usenix NSDI*, 2010.

[27] H. Ballani, D. Gunawardena, and T. Karagiannis, "Network Sharing in Multi-tenant Datacenters," MSR, Tech. Rep. MSR-TR-2012-39, 2012.

[28] N. Dukkipati, "Rate Control Protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Stanford University, 2007.

[29] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proc. of ACM SIGCOMM*, Aug. 2002.

[30] A. Rai, R. Bhagwan, and S. Guha, "Generalized Resource Allocation for the Cloud," in *Proc of ACM SOCC*, 2012.

[31] "Microsoft System Center Virtual Machine Manager," http://www.microsoft.com/en-us/server-cloud/system-center/default.aspx.

[32] "Amazon's EC2 Generating 220M," http://cloudscaling.com/blog/cloud-computing/amazons-ec2-generating-220m-annually.

[33] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *Proc of ACM SIGCOMM*, 2012.

[34] V. Jeyakumar, M. Alizadeh, D. Mazires, B. Prabhakar, and C. Kim, "EyeQ: Practical Network Performance Isolation for the Multi-tenant Cloud," in *Proc. of Usenix HotCloud*, 2012.

[35] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM CCR*, Oct. 2012.