Network Sharing in Multi-tenant Datacenters

Hitesh Ballani, Dinan Gunawardena, Thomas Karagiannis
Microsoft Research, Cambridge

Technical Report
MSR-TR-2012-39

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1. INTRODUCTION

The on-demand access to computing resources offered by cloud datacenters has prompted a lot of applications to migrate to the cloud. These applications often involve virtual machines (VMs) belonging to a tenant communicating across the datacenter's *internal network*. The multi-tenant nature of the cloud has also led to both cloud providers and individual tenants offering services to other tenants [1]. For instance, Amazon EC2 offers services like SimpleDB, Simple Queue Service, EBS, while mapreduce++ and CloudBuddy are services run by tenants. The resulting tenant-provider and tenant-tenant network traffic further diversifies datacenter network communication.

This rise of intra-cloud network communication is at odds with today's cloud offerings. With infrastructure as a service (IaaS), tenants get VMs with dedicated CPU and memory but the underlying network is shared. Consequently, the network performance for tenants can vary significantly [2–4]. This, in turn, impacts the performance for a wide variety of applications; from user-facing web services [3,5] to data-parallel, HPC and scientific applications [3,6–9]. Ultimately, variable network performance impacts tenant cost, as tenants pay based on the time they occupy their VMs which is influenced by network performance. The shared nature of the network and the lack of any traffic isolation mechanisms also opens the door for DoS attacks aimed at disrupting specific services within the datacenter or increasing the cost for other tenants [1,10,11]. All these issues are often cited as key barriers to cloud adoption [12].

In view of these limitations, we identify three key objectives that the datacenter network should satisfy. First, the allocation of network bandwidth should ensure that the maximum network impact of a tenant is bounded (*bounded impact*). This prevents malicious and selfish behavior. Second, VMs should be coupled with *minimum network bandwidth guarantees* which would allow tenants to estimate worst-case bounds for application performance and costs [13,14]. Third, the bandwidth allocation should be *work conserving*. Cloud datacenters rely on multiplexing of resources for cost efficiency and the same should hold for the network.

While recent proposals target some of these objectives, none satisfy them all. For example, proposals to partition the datacenter network among tenants [15–17] ensure that network bandwidth between a tenant's VMs is guaranteed. This, however, causes network fragmentation and is not work conserving. Alternatively, weighted-sharing approaches allow for general communication patterns by assigning weights to tenants (or individual VMs) and allocating network bandwidth in a weighted fashion [10,13,18]. However, being agnostic to VM placement, they offer no or very weak minimum bandwidth guarantees. Further, tenants can gain an unbounded share of the network bandwidth by modifying their traffic patterns to/from other tenants.

The primary contribution of this paper is *the design of network sharing mechanisms that satisfy the complementary goals of minimum VM bandwidth and bounded impact*. We illustrate how careful VM placement and bandwidth allocation can be combined to satisfy all the above objectives, and present **Hadrian**, a system that implements the proposed mechanisms. With Hadrian, VMs are coupled with a minimum amount of network bandwidth. Tenants could specify the desired bandwidth guaranty or select from a set of bandwidth classes. A VM's minimum bandwidth influences its price and network flows are allocated bandwidth in a weighted fashion. The desired bandwidth for a tenant's VMs is used to guide their placement across the datacenter and to derive the weight for their network flows. These weights, when combined with smart VM placement, ensure that each VM achieves its guaranteed bandwidth. The weight selection also ensures bounded impact.

Hadrian's design includes three key features to improve the cloud provider's ability to support many concurrent tenants atop oversubscribed datacenter networks. First, instead of targeting arbitrary communication patterns, we rely on tenants expressing their communication dependencies, i.e., other tenants or *peers* they communicate with. A VM is only guaranteed network bandwidth for intra-tenant and peer communications. While specifying peer tenants is an obvious overhead for tenants, it offers benefits; it makes the datacenter network "default-off", thus protecting against malicious tenants [19]. Second, the minimum rate for communication between any pair of VMs is determined by the lower of the source and the destination guaranty. Finally, driven by typical application workloads where inter-tenant communication is sparse, Hadrian also offers hierarchical bandwidth guarantees (i.e., inter-tenant vs. intra-tenant).

Beyond the network sharing mechanisms, other contributions of the paper include–

- We present two abstractions, *multi-hose* and *hierarchical multi-hose*. These abstractions capture per-VM and per-tenant lower bounds for network bandwidth and allow tenants to reason about worst-case performance for their traffic.
- We present a novel formulation of the tenant bandwidth requirements as a max-flow network which, in turn, guides the placement of their VMs.

Our evaluation shows that minimum bandwidth guarantees yield better and predictable network performance for tenants. These guarantees also improve the datacenter throughput by preventing outliers with very poor network performance. This implies that providers can offer an improved service at a lower price, or choose to

retain today's prices and increase their revenue instead. Either way, there are benefits for both entities. Finally, by bounding the maximum impact a tenant can have, we help curb malicious behavior in shared settings.

In effect, we have co-opted the benefits of both groups of past proposals by combining bandwidth-aware VM placement [15,17] with weighted bandwidth allocation [10,13,18]. On the flip side, our sharing mechanisms require changes to network elements. While Hadrian's design minimizes these changes and our prototype shows their feasibility, they do present a barrier to adoption. We believe that, at the very least, the proposed mechanisms will inform a discussion of how a distributed resource like the network can be efficiently and robustly shared in multi-tenant settings.

2. NETWORK SHARING GOALS

Unlike other resources in the datacenter, the internal network is shared amongst tenants for both traffic between a tenant's VMs (*intra-tenant*), as well as between VMs of different tenants (*inter-tenant*). As in the Internet, bandwidth is allocated to network flows through end host mechanisms such as TCP congestion control which ensures per-flow fairness. While practical, this model has a number of harmful implications that guide design objectives for network sharing mechanisms in datacenters. Specifically:

Unfair sharing. The lack of proper mechanisms to isolate tenant traffic allows selfish tenants to easily obviate TCP's per-flow fairness by using multiple TCP flows [10] or simply UDP. Such abuse can occur across two dimensions: i) *unfair share to any other VM or a set of VMs*, whereby a selfish tenant can obtain higher bandwidth to VMs of other tenants running services. Taking this a step further, malicious tenants may even launch DoS attacks on specific tenants by sourcing a lot of traffic to their VMs. ii) *unfair share on any network link*, whereby malicious tenants can attack the infrastructure simply by generating a lot of traffic to arbitrary destinations, thus degrading performance for any tenants using common network links. Besides maliciousness, this can simply reflect application patterns. For example, a tenant shuffling a lot of data between its VMs or a tenant running a popular service can, by sending and receiving a lot of traffic, degrade performance for neighboring tenant VMs.

Objective 1: *Bounded impact.* The maximum impact a VM can have on VMs it communicates with *and* on VMs that share network links with it should be bounded. Precisely, the maximum bandwidth for a VM on any network link should be proportional to the VM's price and the traffic load on the link from other VMs. Further, it should be independent of the VM's traffic pattern.

Unpredictable performance and cost. From the discussion above, it follows that the network bandwidth achieved by a VM depends on where it is located, the network load imposed by neighboring VMs sharing network paths, the transport protocols being used, etc. Consequently, the network performance for a tenant can vary significantly in today's cloud [2–4,20] and production datacenters [9] with an order of magnitude variation not being uncommon [17]. This variation is a leading cause for unpredictable application performance in the cloud [3], impacts a wide-variety of applications [3,5,6,21] and hinders the ability of the cloud to support applications classes, like HPC and scientific computing, that rely on predictable performance [7,8]. Such unpredictability extends to actual cost as tenants are charged based on the time they occupy their VMs and this time is influenced by the network.

Objective 2: *Minimum bandwidth guarantees.* The cloud's internal network should be elevated to a first-class resource by ensuring that tenant VMs are guaranteed a minimum network bandwidth. This allows tenants to estimate *worst-case* performance and cost for their applications.

To tackle the lack of network isolation across tenants, recent proposals resort to enforcing hard guarantees by virtually partitioning the network across tenants [15–17]. While effective, such solutions result in network fragmentation and underutilization, thus hurting efficiency.

Objective 3: *Work conserving.* Cloud datacenters multiplex physical resources across tenants to amortize costs and the same should hold for the network. Hence, bandwidth allocation mechanisms should be work conserving and any unused network bandwidth should be available for VMs with network demand, irrespective of their bandwidth guaranty.

3. BANDWIDTH LOWER BOUNDS

To satisfy the second network sharing objective, we propose tenants be offered VMs with a lower bound on their network bandwidth. In today's multi-tenant datacenters, tenants request VMs with varying amount of CPU, memory and storage resources. By abstracting away details of non-network resources, each tenant request today can thus be summarized by $<V>$, the number of VMs requested. We extend this interface by providing *minimum* guarantees. A tenant P requesting V_P VMs with a minimum bandwidth of B_P^{min} is characterized by $<V_P, B_P^{min}>$. In this section, we explicitly describe the semantics of the guaranty offered to tenants. The goal is to balance the competing needs of tenants and providers; the guarantees should be reasonable for tenants yet provider friendly. They should not limit the provider's flexibility in accepting tenant requests.

Our guaranty ensures that the *worst-case* network performance achieved by any VM in the datacenter is the

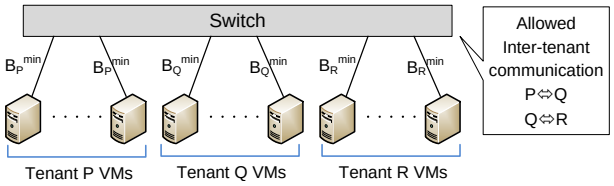

Figure 1: Multi-hose model for a datacenter with three tenants. Tenant communication dependencies are used to determine the inter-tenant communication allowed.

performance it would achieve if it were connected by a link, whose capacity equals the VM's minimum bandwidth, to a central switch. Figure 1 shows this representation for a datacenter with 3 tenants. This extends the hose model [22] for capturing tenant bandwidth demands to a multi-tenant setting and is thus referred to as the **multi-hose model**.

The multi-hose model implies that *the minimum bandwidth for a network flow in the datacenter is the same as the bandwidth the flow would achieve on the multi-hose topology.* Here, a flow refers to all transport connections between a pair of VMs. For example, consider two VMs p and q belonging to tenants P and Q respectively. Assuming that these VMs have a single flow between them and no flows to any other VMs, the bandwidth for the flow on the multi-hose topology is $\min(B_P^{min}, B_Q^{min})$. Since the multi-hose topology specifies *worst-case* performance, the actual bandwidth for the flow in the datacenter is no less than $\min(B_P^{min}, B_Q^{min})$. Generalizing this, say p and q communicate with N_p and N_q VMs each, then the bandwidth for a flow between p and q should be at least $\min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$. **Note that the actual rate for the flow can exceed this value**. In effect, ensuring the minimum bandwidth for VMs can be used to determine lower bounds for the bandwidth that individual flows receive, thus allowing tenants to estimate worst-case flow completion time and possibly even application performance.

However, given the oversubscribed nature of typical datacenter topologies, offering any non-trivial minimum bandwidth guarantees to tenants severely limits the provider's ability to accommodate many concurrent tenants on their infrastructure. To improve the provider's flexibility in terms of accepting many tenants, we modify the semantics of the bandwidth guarantees offered to tenants by introducing i) tenant "peer" relationships, and ii) hierarchical guarantees. We elaborate on these below.

3.1 Communication dependencies

Allowing arbitrary VMs to communicate under guaranteed performance is impractical. Instead, our guarantees apply only for "expected" communication. To achieve this, tenants expose their communication dependencies to the provider when requesting for virtual machines.[1] A tenant's communication dependency is a list of other tenants or *peers* that the tenant expects to communicate with. Examples of such dependencies are listed below.

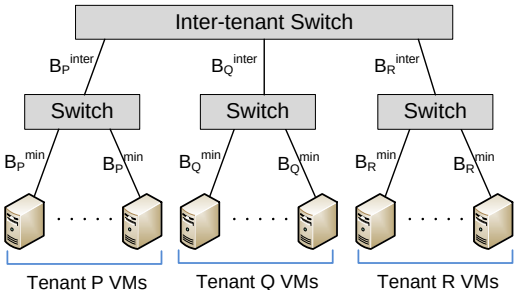

Figure 2: Hierarchical multi-hose model gives per-VM bandwidth lower bound for intra-tenant traffic and per-tenant bound for inter-tenant traffic.

$$1)\ P:\{Q\}, \qquad 2)\ Q:\{P,R\}, \qquad 3)\ R:\{*\}$$

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 used by other tenants may not know its peers apriori, we allow for wildcard dependencies. Thus, the last dependency implies that tenant R is an *open tenant* and can communicate with any tenant that explicitly declares a peering with R (in this example, tenant Q). Note however that since tenant P has not declared a peering with R, communication between VMs of P and R is not allowed.

As shown in Figure 1, the provider can use the communication dependencies to determine the allowed inter-tenant communication and as we show later, is thus better positioned to offer bandwidth guarantees to tenants. While expecting tenants to express their communication dependencies is an overhead for them, it also offers benefits. It makes the datacenter network "default-off" since traffic can only flow between a pair of tenants if both have declared a peering with each other.

3.2 Hierarchical bandwidth lower bounds

The multi-hose model offers tenant VMs the same minimum bandwidth guaranty for traffic to all VMs, irrespective of whether the destination VMs belong to the same tenant or to other tenants. As a contrast, we envision typical cloud applications will involve more communication between VMs of the same tenant than

[1]We assume tenants know that the tenants they depend on are running in the same datacenter. This is typically true for services as tenants advertize where the service is run to attract customers.

across tenants. Such structure already applies to common enterprise applications moving to the cloud [23]. For instance, consider a tenant serving web content to end users and relying on an ad service run by another tenant [1]. There would be more intra-tenant traffic between VMs running the web service than inter-tenant traffic to VMs of the ad service.

To capitalize on the skewed traffic patterns of such applications, we allow for hierarchical guarantees. This is the **hierarchical multi-hose model** and is shown in Figure 2. Each VM for tenant P is guaranteed a bandwidth no less than B_P^{min} for traffic between its VMs. Beyond this, the tenant also gets a minimum bandwidth guaranty for its aggregate inter-tenant traffic, B_P^{inter}. The ratio of this inter-tenant bandwidth and the aggregate per-VM bandwidth ($V_P * B_P^{min}$) allows a tenant to capture the relative sparseness of their inter-tenant traffic as compared to intra-tenant traffic. Thus, with the hierarchical multi-hose model, a tenant requesting V VMs is characterized by the four tuple $<V, B^{min}, B^{inter}, dependencies>$.[2]

The combination of the hierarchical multi-hose model with tenant dependencies improves the provider's ability to pack tenants on their infrastructure without limiting tenant value.

4. Hadrian

To illustrate our network sharing mechanisms, we design Hadrian, a system for sharing network bandwidth in multi-tenant datacenters. Hadrian relies on bandwidth-aware VM placement and weighted bandwidth allocation to satisfy the three objectives identified in Section 2 with the guaranty semantics as described in the previous section. This is achieved through 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 machines. This allocation of VMs to physical machines accounts for the minimum bandwidth requirements of the VMs and for their communication dependencies.

- *Bandwidth Allocation.* Flows are assigned network bandwidth in a weighted fashion. The flow weights are chosen such that the bounded impact and minimum bandwidth requirements are assured.

4.1 VM Placement

The placement manager takes a tenant request and places the tenant's VMs at empty slots on datacenter physical machines. VM placement problems are often mapped to multi-dimensional packing with constraints regarding various physical resources [24]. In our case, each tenant requires slots on physical machines and minimum network bandwidth on the links connecting them. However, determining the bandwidth required on any given link is not trivial. For all accepted tenants, their VMs should be assured of the desired minimum bandwidth for traffic to other VMs of the same tenant and aggregate (per-tenant) minimum bandwidth for traffic to other tenants.

[2]When $B^{inter} = V * B^{min}$, the hierarchical multi-hose is the same as the multi-hose and tenants simply get per-VM minimum guarantees for all traffic.

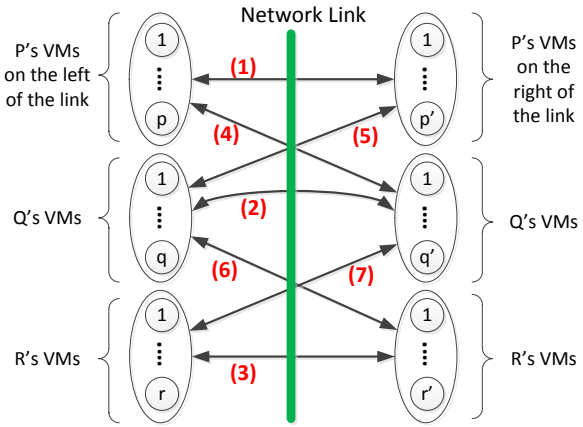

Figure 3: Sets of flows that can traverse a network link.

We begin by quantifying the bandwidth required on individual network links to support tenant bandwidth guarantees. This, in turn, allows us to define what a valid placement looks like and design a placement algorithm. In doing so, we focus on tree-like physical network topologies; examples include the multi-rooted tree topologies used in today's datacenters, and richer topologies like VL2 [25] and FatTree [26]. Such topologies are hierarchical, recursively made up of sub-trees at each level. For instance, in a three-level topology, physical hosts are arranged in racks, racks make up pods and pods make up the complete datacenter.

4.1.1 *Characterizing bandwidth requirements*

We use the three tenant scenario described earlier to both provide intuition regarding the challenge of quantifying the bandwidth required on network links to support tenant requirements and to explain our approach. Let's consider any network link in the datacenter. Say the link has p VMs for tenant P to the left and p' VMs to the right such that $p + p' = V_P$, where V_P is the number of VMs belong to tenant P.[3] Similarly, the link has q and r VMs for tenants Q and R on the left, and

[3]This assumes that the datacenter network has a simple tree topology, so the link separates P's VMs into two groups. "Left" corresponds to the sub-tree under the link while "Right" corresponds to the rest of the datacenter. We relax this simple tree assumption in Section 4.3.

q' and r' VMs on the right.

We first enumerate the sets of flows that can possibly traverse the link. These are shown in Figure 3, labeled (1) to (7). This includes intra-tenant flows between VMs of P, Q and R (labeled (1), (2), (3) respectively), and inter-tenant flows that conform to tenant dependencies, i.e., between VMs of $P \Leftrightarrow Q$ (labeled (4) and (5)) and VMs of $Q \Leftrightarrow R$ (labeled (6) and (7)).

The minimum bandwidth guaranty for VMs entails that each of these sets of flows has a minimum rate associated with it. Further, irrespective of the traffic pattern across the physical link, it should be able to ensure that the combined minimum rates for all flows across it are met. The minimum rate for any given set of flows, when running in isolation, is easy to determine. For example, consider the flow set (1) between VMs of P on the left and the right of the physical link. As explained earlier, the minimum rate for these flows is the rate they would achieve on the multi-hose topology in Figure 2. In the topology, each of P's VMs has a minimum bandwidth of B_P^{min}. Hence, the combined bandwidth for the p VMs on the left of the link is $p{*}B_P^{min}$ and the combined bandwidth for the p' VMs on the right is $p'{*}B_P^{min}$. Thus, the total rate for this set of flows is $\min(pB_P^{min}, p'B_P^{min})$. This is the minimum rate the set of flows should receive across the physical link in question, assuming P has no inter-tenant flows across it (i.e., if flow sets (4) and (5) did not exist).

The same analysis can be extended to determine, in isolation, the minimum rate for other sets of flows across the link. The physical link should have enough capacity to satisfy the combined minimum rate for all these sets of flows. However, simply summing the minimum rates for all sets of flows over-estimates the total bandwidth needed. Instead, 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.

Figure 4 shows the flow network corresponding to the link being considered and is explained below. To avoid confusion, "link" refers to physical network links while"edge" corresponds to the flow network. 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 per-VM bandwidth constraint. The two intra-tenant nodes for each tenant are connected by an edge of infinite capacity (long-dashed edge). These intra-tenant nodes effectively represent the intra-tenant

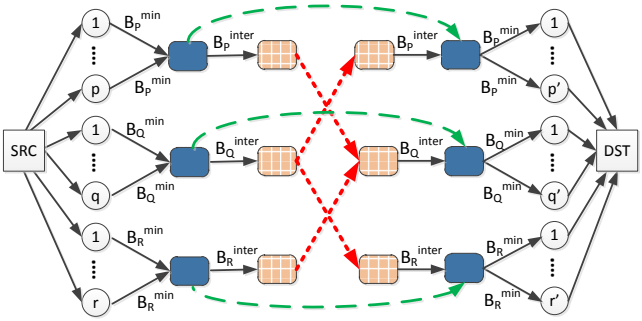

Figure 4: Flow network to capture the bandwidth needed on a link that connects p VMs of tenant P on the left to p' VMs on the right, and so on for tenants Q and R. Circles represent VMs, solid rectangles are intra-tenant nodes and shaded rectangles are inter-tenant nodes.

switch in the hierarchical multi-hose model. 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 represents the bandwidth constraint for communication between tenants. 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 maximum rate that the sets of real flows (1)–(7) can achieve on the multi-hose topology. This, in turn, is the minimum *bandwidth required* on the physical link to ensure that the minimum bandwidth guarantees of VMs are assured.

While the description above involves already existing tenants, the flow network formulation can also be used to determine the bandwidth required when placing a new tenant request. Specifically, for each link, the placement manager maintains an associated flow network that includes VMs of existing tenants on either side of the link. To determine whether the link will be able to support the communication demands for newly requested VMs, the flow network for the link is modified by adding the appropriate nodes and edges. The max flow for the resulting flow network is the bandwidth required on the link to accommodate the new VMs, and should be less than the link's capacity.

4.1.2 *Characterizing 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 be less than or equal to the link's available capacity. Thus, the *VM Placement*

problem boils down to finding a valid placement for the tenant's VMs.

However, 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. Precisely, for any link l, we can define a set for the number of VMs that can be placed in its sub-tree while ensuring that the link has sufficient bandwidth. We term this the *VMs_Allowed* set for a link and is defined as

$$VMs\_Allowed_l \quad = \quad \{i \mid max\_flow(l,i) \leq C_l\}$$

where, *max_flow(l,i)* is the max-flow for link l's flow network after placing i VMs of the new tenant in the sub-tree rooted at the link (and the rest of the tenant's VMs outside the sub-tree) and C_l is the link capacity. In effect, the *VMs_Allowed* set abstracts away the bandwidth requirements and the communication dependencies of tenants, expressing them as constraints regarding the number of VMs that can be placed at any level of the datacenter hierarchy.

Given this, we formally define a valid placement below. Consider a tenant request $<V, B^{min}, B^{inter}, dependencies>$ and a datacenter whose topology is represented by tree $T(L, H)$, where L is the set of network links and H is the set of physical hosts. For each link l, $host_l$ is the set of physical hosts in the sub-tree under l. Further, for each host h, $slots_h$ is the number of empty VM slots on it. A placement $[a_h]$ determines the number of VMs allocated at each host. A placement is said to be valid if it satisfies the following constraints–

$$\sum_{h \in H} a_h = V \tag{1}$$

$$a_h \leq slots_h, \qquad \forall\ h \in H \tag{2}$$

$$\sum_{i \in host_l} a_i \in VMs\_Allowed(l) \qquad \forall\ l \in L \tag{3}$$

The first constraint ensures that all V requested VMs are allocated while the second constraint ensures that VMs are placed only on empty slots. The final constraint ensures that the number of VMs placed in the sub-tree under any link should be part of the link's *VMs_Allowed* set, so that the minimum bandwidth guarantees are satisfied.

Note that the *VMs_Allowed* set does not have any structure which makes it non-trivial to find a valid placement. We illustrate this with a simple example. As shown in Figure 5, consider a physical host with six VM slots, two of which are occupied by a tenant Q. Also, say a new tenant P requests 5 VMs with a minimum bandwidth of 1000 Mbps between its VMs and an aggregate bandwidth of 1100 Mbps to Q's VMs. The VMs_Allowed set

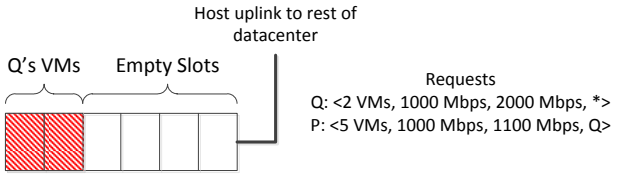

Figure 5: Placement of tenant P. The host's uplink has capacity 1000 Mbps and the number of VMs allowed to be placed on the host is {1, 4}.

for this host is {1, 4} i.e., only 1 or 4 of P's VMs can be placed on the host. Placing zero VMs is not valid since the capacity of the host uplink will be less than the 1100 Mbps needed to ensure the minimum rate for traffic between VMs of P and Q. Similarly, two (or three) VMs cannot be placed as the uplink's capacity will not be sufficient for intra-tenant traffic between P's VMs. This illustrates that the number of VMs to be placed in any sub-tree of the datacenter is not a simple range set. Further, a minimum number of VMs may need to be placed in the sub-tree (i.e., 0 is not always a member of VMs_Allowed).

4.1.3 VM Placement Algorithm

The VM Placement problem requires placing a tenant's VMs while satisfying constraints regarding the number of VMs that can be placed at any level of the datacenter hierarchy. We designed a greedy, first-fit placement algorithm. This is shown in Figure 6. The basic idea behind the placement is simple– the number of VMs placed on any physical host is determined by both the VMs_Allowed set for the host and by constraints for higher level sub-trees that include the host; for instance, in a three-level topology, these higher-level sub-trees reflect the rack and pod that include the host. Given this, the placement algorithm uses the recursive "Alloc" function to traverse the topology in a depth-first fashion. Constraints for each level of the topology are used to determine the maximum number of VMs that can be placed at sub-trees below the level (line 9), and so on till we determine the number of VMs that can be placed on any given host. VMs are greedily placed on the first available host (line 12). We then verify whether constraints at higher levels of the topology are met (line 21).

The placement algorithm described above ensures that if a placement is returned, it is a valid placement. Our actual implementation includes several enhancements to improve both the speed and scalability of the algorithm, as well as the kind of placement. We briefly describe a few of these–

Flow network decomposition. We use Edmonds-Karp algorithm to calculate the max-flow for the flow network associated with network links, which works particularly well for sparse graphs. For each link, instead of maintaining a single flow network that includes all ex-

```
Require: Topology tree T(L) with root_link as root
Ensure: Placement for request <V, B^min, B^inter, dep>
 1: for each l ∈ L do
 2:     Calculate VMs_Allowed_l
    //Place V VMs under the root_link
 3: if Alloc(V, root_link) = V then
 4:     return True
 5: else
 6:     return False


    //Allocate at most hi VMs in sub-tree under l
 7: //Returns number of VMs allocated, -1 on error
 8: function Alloc(hi, l)
 9: hi = min(hi, max(VMs_Allowed(l)))
10: if l is a host uplink then
11:     //Base case - h is the corresponding physical host
12:     total = min(slots_h, hi)
13: else
14:     //Iterate over sub-trees
15:     for i ∈ children_l do
16:         allocated = Alloc(hi, i)
17:         if allocated != -1 then
18:             hi = hi− allocated; total = total + allocated
19:         else
20:             return -1
21: if total ∉ VMs_Allowed(l) then
22:     return -1
23: return allocated
```

Figure 6: VM Placement algorithm.

isting tenants in the datacenter, we decompose the flow network based on tenant communication dependencies. This is based on the insight that an average tenant will not communicate with all other tenants. With this decomposition, two tenants are part of the same flow network if they either depend on each other directly or have a sequence of dependencies that connects them. Consequently, when a new tenant request arrives, we only need to modify the flow network(s) containing peers of the tenant. These flow networks are much smaller than the original flow network. This significantly improves the speed of placement decisions. Further, we also use the communication dependencies to prune the list of network links that may be impacted by the incoming tenant and whose VMs_Allowed set needs to be recalculated.

Minimum VM constraints. As explained earlier, there may be constraints regarding the minimum number of VMs that must be placed inside a datacenter sub-tree. Consequently, the placement algorithm proceeds in two stages. We first traverse the datacenter and place VMs so as to satisfy minimum VM constraints. We then use the algorithm described above to place any remaining VMs.

Locality. Inspired by the fact that datacenter network topologies are typically oversubscribed with less bandwidth towards the root than at the leaves, the optimization goal for our 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 subtree possible. This heuristic reduces the number and the height of network links that may carry the tenant's traffic and hence, preserves network bandwidth for future tenants.

4.2 Bandwidth Allocation

The VM Placement algorithm ensures that a tenant request, if admitted, is placed such that the underlying links have sufficient capacity to support the minimum rates for the tenant's flows. This is the rate a flow would achieve on the hierarchical multi-hose topology. The allocation of network bandwidth to flows should achieve three goals: i) a flow's actual rate is at least its minimum rate, ii) the allocation should be work conserving, and iii) tenant impact is bounded, i.e., tenants should not be able to increase their allocation by modifying application traffic patterns.

To achieve this, Hadrian allocates network bandwidth to flows *in proportion to their minimum rates through weighted max-min fairness.* Specifically, network bandwidth is shared in a weighted fashion and the weight for each flow is its minimum rate. This allows providers to connect a tenant's payment with the tenant's resulting bandwidth allocation: a VM's price influences the VM's minimum bandwidth which dictates the minimum rate its flows should achieve. The allocation scheme ensures that the actual flow rate is proportional to (and no less than) this minimum rate.

To explain Hadrian's bandwidth allocation, we focus on two tenants P and Q whose VMs are guaranteed a minimum bandwidth of B_P^{min} and B_Q^{min}. This is the multi-hose model, although the analysis generalizes to the hierarchical model too. Consider a flow between two VMs p and q for these tenants. These VMs are communicating with a total of N_p and N_q VMs respectively. As explained earlier, the minimum rate for this flow is $\min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q})$. This is also going to be the flow's weight $w_{p,q}$, and implies that the rate for the flow, as determined by the bottleneck link along its path, is given by

$$B_{p,q} = \frac{w_{p,q}}{w_T} * C, \tag{4}$$

where C is the capacity of the bottleneck link and w_T is the sum of the weights for all flows across the link.

Below we show that such weighted sharing ensures that flows achieve their minimum rate and the bandwidth allocation has a bounded impact.

4.2.1 Lower bound on flow rate

We use the bandwidth allocated by the bottleneck link to the flow between VMs p and q as an example to sketch

a proof showing that Hadrian ensures a minimum flow rate.

To do this, let's superpose all VM-to-VM "real flows" across the link as a single "aggregate flow" on top of the flow-network associated with the link. Say each real flow contributes a volume equal to its weight to the aggregate flow in the flow-network. The weights assigned to the real flows are such that the volume across any edge in the flow network will be less than the edge's capacity. Hence, the aggregate flow is a valid flow for the flow network. The aggregate weight w_T for all the real flows represents the total volume of the single aggregate flow, and cannot exceed the max-flow. This is true irrespective of the traffic pattern across the link. Further, the placement algorithm ensures that the max-flow for the flow network cannot exceed the link capacity. Thus, $w_T \leq C$. Using this in (4), we get

$$B_{p,q} \geq w_{p,q} = min(\frac{B_P^{min}}{N_p}, \frac{B_Q^{min}}{N_q}).$$

This shows that the combination of flow weights and careful VM placement ensures that each flow is guaranteed the desired minimum rate. *Since allocation is based on weighted max-min fairness, any unused capacity is available for flows with demand, and the actual flow rates can exceed their minimum rate.* Below we show that our allocation scheme ensures fair and robust sharing of any unused network capacity.

4.2.2 Upper bound on VM impact

For robust network sharing, tenant VMs should not be able to benefit unfairly by modifying their traffic pattern, either by using more transport connections or by communicating with a lot of other VMs. A tenant VM can "benefit" in two ways– (i). get an unfair fraction of bandwidth to a specific set of VMs, and (ii). get an unfair fraction of the bandwidth on any given link. We focus on ther latter since (i) is simply a special case of (ii); achieving an unfair bandwidth allocation to a set of VMs entails achieving an unfair allocation on the physical uplinks for those VMs.

With Hadrian, the bandwidth that a VM can achieve on a link is capped and this upper bound depends only on the desired minimum bandwidth for the VM (and hence, the VM price) and the weights for other VMs using the link. We prove this in the Appendix and show that the maximum possible bandwidth (B_p^l) a VM p can achieve on link l is

$$B_p^l \leq \frac{B_P^{min}}{B_P^{min} + k} * C_l,$$

where C_l is the link capacity and k is the sum of the weights for other VMs. This upper bound has two key properties. First, *it is independent of the number of flows for VM p, i.e., the number of other VMs it communicates with.* Second, *it is work conserving as p can use the entire link capacity if no other VMs are using it.* We note that the first property does not hold for the status quo or even recent weighted sharing proposals like per endpoint weights [13] where a VM can always grab more of a link's bandwidth by communicating with more destinations. Even with Seawall's source-based weights [10], the impact of a VM running a popular service on a link increases as more VMs initiate flows to it. We show this experimentally in § 5.3 by comparing Hadrian with such proposals. More generally, allowing a VM to gain an unbounded share of a link's bandwidth by modifying its traffic pattern opens the door for selfish and malicious behavior in shared settings.

The key intuition behind Hadrian's bounded impact property is the following. Flow weights are designed such that, irrespective of a VM's traffic pattern, the aggregate weight for its flows can never exceed the VM's desired minimum bandwidth. We explain this with an example. Consider a VM in a setup where all VMs have a 50 Mbps minimum bandwidth. When the VM has one flow and assuming this is the only flow for the destination too, the flow weight is $\min(\frac{50}{1}, \frac{50}{1}) = 50$. When the VM communicates with two destinations, the sum of the flow weights is $\min(\frac{50}{2}, \frac{50}{1}) + \min(\frac{50}{2}, \frac{50}{1}) = 50$. And so on for more destination VMs. Thus, by bounding the aggregate weight for a VM's traffic, we ensure an upper bound for the impact a VM can have on any network link and hence, on the datacenter network.

4.2.3 Implementing Bandwidth Allocation

With the weighted bandwidth allocation described above, the weight for a flow depends on both the desired minimum bandwidth and the number of flows for the VMs at its endpoints. Further, with the hierarchical multi-hose model, the weight for a inter-tenant flow also depends on the total number of inter-tenant flows for the source and destination tenants who own the endpoints. Thus, neither the source nor the destination VM has all the information to compute a flow's weight.

An obvious approach is to use a centralized controller that monitors flow arrivals and departures to calculate the weight and hence, the rate for individual flows. These rates can then be enforced by the hypervisors on physical machines. However, making such a solution work conserving requires the controller to also monitor the utilization of individual links and update the rates of all flows periodically. This is necessary as sources might not be able to match their given rates, ergo underutilizing the network. The bursty nature of datacenter traffic and the resulting communication overhead with this centralized approach renders it practically infeasible in realistic scenarios.

We adopt an alternative tact and distribute the rate calculation logic amongst the network switches. The overarching principle here is to minimize the amount

of network support required by moving functionality to the trusted hypervisors. Our design is based on explicit control protocols like RCP [27] and XCP [28] that share bandwidth equally and convey flow rates to end hosts. With Hadrian, we require weighted, instead of an equal, allocation of bandwidth. We provide a design sketch of our bandwidth allocation architecture below. Our implementation is detailed in Section 5.4.

With IaaS, VMs can use any and all transport protocols. We aim to retain this feature. Hence, VM network traffic is tunneled inside hypervisor-to-hypervisor flows such that all traffic between a pair of VMs count as one flow. Traffic is only allowed to other VMs of the same tenant and to VMs of peers. The hypervisor embeds the `VM-id` and the `tenant-id`, a count of the total number of flows, B^{min}, B^{inter}, and a count of the total number of inter-tenant flows for the VM sourcing the traffic in the packet header. The destination hypervisor embeds the same information for the destination VM in the packets in the reverse direction. Network switches along the path only maintain a count of the number of flows for each VM and the number of inter-tenant flows for each tenant traversing it. Based on this information, the switch can determine the weight for any flow.

Instead of using per-flow queuing to enforce these weights at the switches, we push this functionality to the hypervisors. Given a flow's weight, the switch allocates a weighted share of the outbound link's capacity to the flow. This rate allocation is conveyed to the destination and is piggybacked to the source hypervisor where it is enforced. As rate allocations depend on traffic patterns, the rate allocation occurs periodically and the source hypervisor adjusts its sending rate.

4.3 Design discussion

Hadrian's reliance on careful VM placement implies that we inherit the assumptions and concerns afflicting bandwidth-aware VM placement proposals [15,17]. These include–

Placement assumptions. Our placement algorithm assumes knowledge of an up-to-date network topology. Further, it assumes that traffic between all VMs for tenants belonging to the same flow-network is routed along a tree. Note that our flow-network decomposition ensures that two tenants are part of the same flow-network if they directly or indirectly depend on each other. This assumption allows us to precisely identify the physical links along which a given VM's traffic will be routed and account for the bandwidth required on only these links, instead of all links. This assumption holds trivially if the datacenter has a tree network topology. However, today's datacenter typically use multi-rooted tree topologies, while even richer fat-tree topologies have been proposed [25,26]. Such topologies offer multiple paths between any VM pair. This poses a challenge for the placement since ensuring that each path has sufficient bandwidth for the traffic across it which would make it hard for the provider to accommodate many concurrent tenants. Two approaches can be used to address this.

The placement algorithm can treat multiple physical links as a single aggregate link if traffic is distributed across them evenly. Popular multi-pathing mechanisms like ECMP and VLB achieve this by splitting traffic across links using a hash function on a per-flow basis. While variation in flow lengths and hash collisions can cause traffic imbalances, a centralized controller can be used to reassign flows and ensure even traffic distribution across multiple links [29].

Alternatively, the datacenter routing can be controlled explicitly to satisfy our assumption atop a multi-path topology. Specifically, instead of distributing traffic from all VMs across all paths, traffic from a given VM can be routed along one (or a few) paths such that we only need to account for the VM's guaranteed bandwidth on the links along these paths. Proposals like SPAIN [30] and SecondNet [15] offer backwards compatible techniques to do just this.

Bandwidth allocation assumption. The bandwidth allocation for flows assumes they traverse a single path and routing is symmetric. Since multi-pathing mechanisms like ECMP and VLB operate at the granularity of flows, a flow is already routed along a specific path today.

Network failures. Failures of physical links and switches impact both the guarantees for existing tenants and the placement of subsequent tenants. Hence, Hadrian's placement manager requires fast updates concerning network failure events. We note that timely failure information is necessary for network management and VM placement even today, and production datacenters use automated tools for this [31]. The placement manager can use such failure information to determine the VMs whose minimum bandwidth may not be assured and migrate them appropriately.

Overall, while these are important practical issues, the impetus of this paper is on the notion of sharing a distributed resource like the network and the benefits of the properties offered.

5. EVALUATION

Our evaluation covers three main aspects. First, we use large scale simulations to show that Hadrian achieves the three network sharing objectives. Second, we highlight that satisfying these objectives improves the overall datacenter performance. Finally, we benchmark our implementation on a small testbed, and show that the placement manager can deal with the scale and churn posed by datacenters.

5.1 Simulation setup

To model the operation of Hadrian at scale, we developed a simulator that coarsely models a multi-tenant datacenter. The simulated datacenter has a three-level topology. Forty hosts with 1 Gbps links are assigned to racks and a Top of Rack switch connects each rack to an aggregation switch. The aggregation switches, in turn, are connected to the core switch. The network topology has no path diversity. Consequently, we vary the oversubscription of the physical topology by varying the bandwidth of the links between the switches. The 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 10:1.

Tenants. We model two kinds of tenants. First, open tenants that offer services for other tenants and hence, have wildcard (*) communication dependencies. Second, client tenants that may use the services of zero or more open tenants. Each client tenant runs a job requiring some VMs. Given our focus on network performance, each job involves network flows and the job finishes when its flows finish. More precisely, a tenant requiring V VMs involves V flows of uniform length, one for each VM. With Hadrian, tenants also ask for a minimum bandwidth (B^{min}) for their VMs. Given this bandwidth value and the length of a job's flows, tenants can estimate their worst case job completion time. A job is said to be an *outlier* if it takes longer than the worst-case completion time.

For each job, some of its flows are intra-tenant, i.e., to other VMs for the same job, while others are to VMs of open tenants the job depends upon. The fraction F 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*$>$.

This naive workload was deliberately chosen; by abstracting away any non-network resources, we are able to directly compare the impact of various network sharing techniques. Our job model though is still broad and realistic since, as discussed in § 2, network performance does have a significant impact on the performance of a large class of cloud applications.

Cloud Provider. To model the operation of cloud datacenters, we simulate tenant requests arriving over time. The provider uses a placement algorithm 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 occupancy* of the datacenter. This is the fraction of datacenter VMs that, on average, are expected to be occupied. Regarding bandwidth guarantees, we consider a setup where 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* requested by the tenants.

Placement → B/w Allocation	Greedy	Dependency -aware	Hadrian's placement
Per-flow	*Baseline*	*Baseline+*	*Hadrian-P*
Per-endpoint	*FairCloud* [13]	–	–
Per-source	*Seawall* [10]	–	–
Hadrian's weights	–	–	*Hadrian*

Table 1: Simulated approaches.

5.1.1 Simulation breadth

Given our focus on VM placement and bandwidth allocation, we consider various approaches for both.

VM Placement. We consider three placement approaches. (i) With *Greedy* placement, a tenant's VMs are greedily placed close to each other. This reflects the most representative placement policy for today's datacenters. (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 communication dependencies and the minimum bandwidth requirements of tenants.

Bandwidth allocation. We consider four bandwidth allocation approaches. (i) With Per-flow sharing, flows are assigned their max-min fair share. This results in the per-flow fairness that TCP provides and is indicative of today's setup. (ii) With Per-endpoint sharing, bandwidth is allocated to flows in a weighted fashion. The weight for a flow between VMs p (tenant P) and q (tenant Q) with N_p and N_q flows each is $\frac{B_P^{min}}{N_p} + \frac{B_Q^{min}}{N_q}$ (*FairCloud* [13]). (iii) With Per-source sharing, the combined weight for the flows sourced by a VM p is B_P^{min} (*Seawall* [10]). (iv) With Hadrian, bandwidth is allocated as explained in §4.2.

Table 1 summarizes all the possible combinations we simulated. For brevity, we focus on the six most relevant approaches marked in the table. Note that allocating tenant VMs greedily combined with the max-min sharing of the network reflects the operation of today's datacenters. Hence, we use it as the *Baseline* for comparison.

Evaluation metrics We use two primary metrics to compare the aforementioned approaches.

(1). *Outlier requests* reflect tenants whose jobs take longer to complete that the tenant's worst-case estimate.

(2). *Rejected requests* are the fraction of the requests that cannot be admitted. This captures the providers ability to accommodate tenants on their infrastructure. Note that, compared to other approaches, Hadrian satisfies additional constraints during admission. With Hadrian, requests may be rejected if not enough VM slots or network bandwidth remains, while the other placement

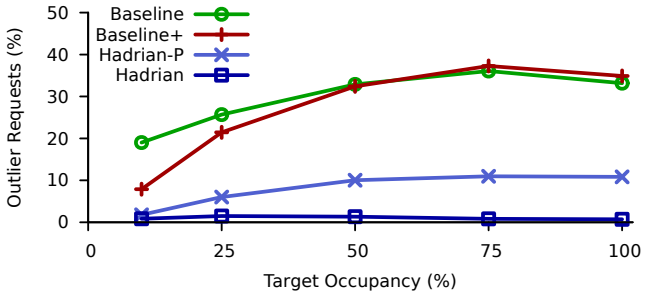

Figure 7: Outlier requests with average minimum bandwidth of 200 Mbps.

approaches reject requests only if enough slots are not available.

Simulation parameters. To characterize the performance of various approaches across the spectrum, we present results while varying most parameters of interest. This includes the target occupancy, the average minimum bandwidth requested by tenants, etc. To begin, we present results with the following setup– the average simulated tenant requests 15 VMs, 10% of the tenants are open tenants, 25% of requests have dependencies on open tenants and 25% of the traffic for such tenants is to VMs of open tenants.

5.2 Cloud performance with Hadrian

We simulated the arrival and execution of 25K tenant jobs. As explained earlier, each tenant request asks for a minimum bandwidth which, in turn, can be used to derive an upper bound estimate for the completion time. Figure 7 shows the percentage of outlier requests with varying target occupancy. As the target occupancy increases and requests start arriving faster, more of them achieve poor network performance and become outliers. As the occupancy exceeds 25%, 30-35% of requests are outliers with *Baseline*. The results for *FairCloud* and *Seawall* are similar and we omit them for figure clarity. Making the VM placement dependency aware with *Baseline+* does not help much either. *As a contrast, with Hadrian-P, 10% of requests are outliers while only 1% of requests are outliers with Hadrian.* This illustrates the benefits resulting first from smart placement alone (Hadrian-P), and second by combining smart placement with weighted bandwidth allocation (Hadrian).

We note that, despite the minimum VM bandwidth guaranty, Hadrian may have some outlier requests because the minimum rate for a flow is determined by the lesser of the guarantees for its endpoints. Thus, a flow for a VM with 150 Mbps minimum bandwidth may achieve a lower rate when the destination VM's minimum bandwidth is the bottleneck. In our setup, this can happen for tenants with a lot of flows to popular open tenants. We verified that none of the tenants are outliers because of insufficient network capacity.

Since operators like Amazon EC2 target an average

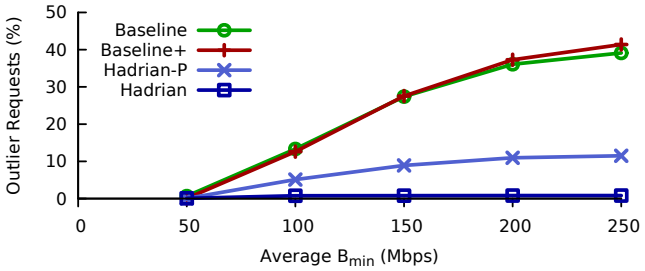

Figure 8: Outlier requests with target occupancy of 75%.

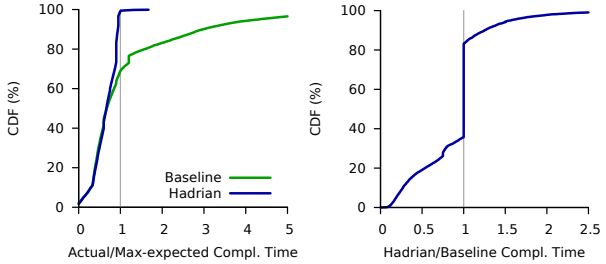

Figure 9: Tenant requests finish faster with Hadrian as compared to *Baseline*. (Average B^{min} = 200 Mbps, Occupancy = 75%)

occupancy of 70-80% [32], we now focus on outlier requests with 75% target occupancy and vary the average minimum bandwidth (i.e., varying fraction of tenants in each bandwidth class). This is shown in Figure 8. As Average B^{min} increases and tenants demand more bandwidth, more of them receive poor network performance and hence, the fraction of outliers increases. However, with Hadrian, the percentage of outliers is very low (<1%) throughout. With *Baseline*, not only are there a lot of outliers, but also some of them receive extremely poor network performance. This is shown in Figure 9 (left) which plots a CDF for the ratio of a job's actual completion time to the maximum expected time. With *Baseline*, 17% of the requests last more than twice as long as their maximum expected completion time. *Instead, Hadrian offers significant reduction in the worst-case completion time by satisfying objectives 2 and 3*; tenants are provided with minimum guarantees but are free to use under-utilized resources due to Hadrian's work-conserving bandwidth sharing.

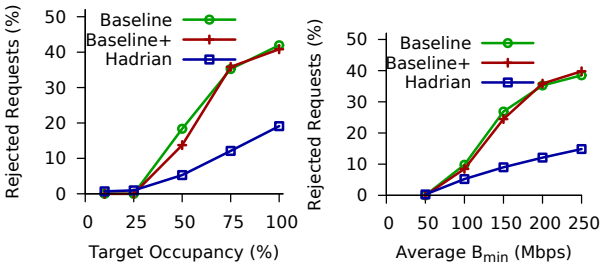

Figure 10: Rejected requests with varying target occupancy and average minimum bandwidth.

Pricing strategy	Tenant cost				Provider revenue
	Average	25^{th}%ile	Median	75^{th}%ile	
Today's pricing	0.64	0.71	1	1	0.83
Revenue neutral	0.77	0.86	1.19	1.19	1

Table 2: Cost analysis for Hadrian- numbers are relative to *Baseline*. Average B^{min} = 200 Mbps and Occupancy = 75%

This improvement in tenant performance does not come at the provider's expense; actually, it is to the contrary. Figure 10 shows the percentage of tenants requests that are rejected. As expected, for all approaches, as target occupancy increases with tenant requests arriving faster, or as average B^{min} increases and tenant jobs are more network intensive, more requests are rejected. The figures show that Hadrian rejects far fewer requests than *Baseline*; for occupancy greater than 50%, it rejects 25-30% fewer requests. The performance of *Baseline+*, relative to Hadrian, is poor too. Hadrian-P, not shown in the figure for clarity, is able to accept just as many requests as Hadrian. However, our earlier experiments show that it results in more outliers.

Hadrian is able to accept more requests because it, apart from reducing outliers, improves average tenant performance too. To show this, we look at the CDF for a tenant's completion time with Hadrian relative to *Baseline* (figure 9, right). While the median tenant performance is the same, more requests finish faster with Hadrian than with *Baseline*. The average request completion time with Hadrian is lower than *Baseline* by 7%. Further, 25% of the requests are more than 29% faster. This results in increased datacenter throughput which, in turn, allows the provider to accept more requests.

Varying workloads. To examine Hadrian over a wide-range of scenarios, we repeated the experiments above while varying other simulation parameters: the physical network oversubscription, the fraction of open tenants, the fraction of client tenants with dependencies and the amount of their inter-tenant traffic. The results follow similar trends and are summarized as follows. With Hadrian, less than 1.5% requests are outliers compared to 20-45% for *Baseline*. Further, Hadrian is able to accept 22-30% more requests.

A particularly interesting scenario is when the datacenter network has no oversubscription. We found that while Hadrian accepts the same number of requests as *Baseline*, 22% of requests are outliers with *Baseline*. Note that with 4 VM slots and 1 Gbps link capacity per machine and no network oversubscription, each VM's fair share of the network is 250 Mbps ("hidden" oversubscription due to VM-to-machine packing). With *Baseline*, tenants demanding a minimum bandwidth of 300 Mbps do receive lower than expected performance due to this hidden oversubscription.

Cost analysis. Today's cloud providers charge tenants a fixed amount per hour for each VM; for instance, Amazon EC2 charges $0.085/hr for small VMs. Hence, the improved tenant performance with Hadrian has implications for their cost too. As shown in Table 2, while the median tenant cost with Hadrian is the same as with *Baseline*, on average, tenants pay 36% less than today. This is due to the reduction in outliers receiving very poor network performance. The reduced tenant cost, however, also reduces the provider's revenue. This is somewhat offset by the fact that the provider can now accept more tenants. Overall, we find the provider's revenue with Hadrian is 83% of that with *Baseline*.

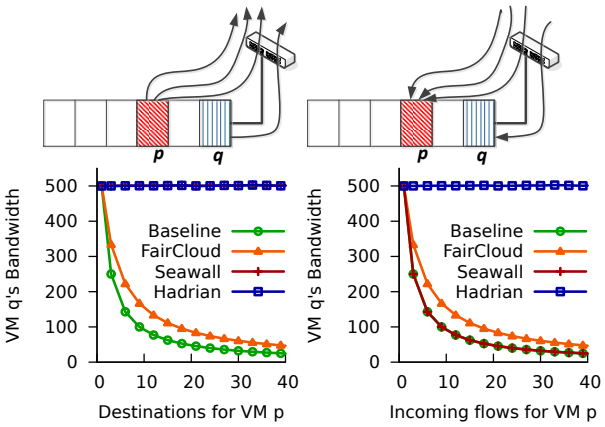

Figure 11: Malicious and selfish behavior involving two VMs co-located on the same host.

This provider loss in revenue can be overcome by new pricing models that account for the added value Hadrian offers. In the simplest case, since Hadrian provides tenants with VMs that have minimum bandwidth, the provider can increase the VM price as compared to today. We repeat the cost analysis to determine how much tenants would have to pay so that the provider remains revenue neutral. Table 2 shows that in the average case, tenants would pay 23% less. The median cost increases since there are no tenants with very poor performance subsidizing other tenants.

Overall, the results above show that Hadrian provides the desired minimum bandwidth for tenants and hence, significantly reduces outliers while allowing the provider to accept more tenants. Consequently, the provider can offer network guarantees while reducing the average tenant cost.

5.3 Objective 1: Bounded Impact

To evaluate the benefits of bounding tenant impact, we focus on two simple scenarios that capture malicious and selfish behavior in shared settings.

Scenario 1. We simulate two VMs for different tenants co-located on the same physical machine (Figure 11, top left). VM q has one flow while VM p is trying to degrade q's performance by initiating flows to a lot of

destinations. All VMs, including destination VMs, have a minimum bandwidth of 300 Mbps. Figure 11 (bottom left) shows the average bandwidth for q's flow for this scenario.[4] With the *Baseline* and *FairCloud*, VM p grabs an increasing fraction of the link's bandwidth as it communicates with more destinations. As a contrast, its bandwidth allocation with Hadrian (and with *Seawall* whose line overlaps with Hadrian) does not change. This is because the aggregate weight for p's flows with Hadrian cannot exceed 300 which is also the weight for VM q's flow. Hence, the link is shared equally.

Scenario 2. The right part of figure 11 (top and bottom) simulates the reverse scenario where VM p is running a popular service and receives a lot of flows while VM q has just one incoming flow. The figure shows that *Seawall*, with its source-based weights, performs the same as *Baseline* (the lines overlap in the figure) and the bandwidth for q's flow declines sharply. With Hadrian, q's bandwidth remains the same throughout.

Albeit very simple, these experiments show that all existing approaches allow tenants to abuse the network at the expense of others. Bounding tenant impact addresses this and can curb malicious network behavior in shared settings. Beyond this, Hadrian only allows communication between tenants that have explicitly declared dependencies to each other, an added countermeasure against internal attacks.

5.4 Implementation and deployment

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

(1). A placement manager that implements the algorithm in § 4.1 to make online placement decisions about tenants.

(2). For bandwidth allocation, we 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 (see § 4.2.3). This involves an endhost component (to be run inside the hypervisor) and a router component. Application packets are tunneled inside RCP_w flows with a custom header. The router component inspects this header, maintains the relevant per-VM and per-tenant counters and encodes the rate allocated to the flow in the packet header. The endhost component ensures that application traffic is sent at the allocated rate. In our implementation, the rate allocation happens once every round trip time (RTT). Every RTT, one of the packets sent by the source contains information about its flows so that switches can update their counters while one of the packets received has the rate allocated to the flow for the next RTT.

Both components run in user space. A kernel driver, bound to the physical network interface, marshals packets between the NIC and these components. The average packet processing time at the router was less than 1μs and was indistinguishable from normal user-space packet forwarding.

Scalability. To evaluate the scalability of the placement mechanisms, we measure the time to allocate tenants requests in a datacenter with 100K machines. Over 100K requests, the median allocation time is 4.13ms with a 99^{th} percentile of 2.72 seconds. Such placement only needs to be run when a tenant is admitted.

Deployment. We deployed Hadrian across a small testbed structured like the multi-tier tree topologies used in today's datacenters. The testbed includes twelve endhosts arranged across four racks. Each rack has a top-of-rack (ToR) switch, and the ToR switches are connected through a root switch. All endhosts and switches are Dell T3500 servers with a quad core Intel Xeon 2.27GHz processor, 4GB RAM and 1 Gbps interfaces, running Windows Server 2008 R2.

To validate our simulation results, we repeat the experiments from § 5.2 on the testbed. Given our focus on network performance, the tenants are not actually allocated VMs but simply run as a user process. With 4 process/VM slots per host, the testbed has a total of 48 slots. Given the small testbed size, we scaled down the mean tenant size to 4 VMs, there is one open tenant, and the average minimum bandwidth is 100 Mbps. Each VM for a client tenant generates network traffic that is tunneled inside a RCP_w flow by Hadrian's endhost component. We vary the weight for the flows to model various approaches; for instance, flows get uniform weights for *Baseline*. Our experiments involve the arrival and execution of 100 tenant requests. Figure 12 shows that roughly 10% of the requests are rejected for all approaches. This is expected due to the size of the testbed. The most important observation for this exercise is that results between the testbed and the simulator experiments were consistent, with the rejected requests on the testbed and the simulator being within 3% for all scenarios. Further, the performance for requests that are accepted is similar. This *cross-validates our simulator and gives us confidence regarding the correctness of our large-scale simulation results.*

6. RELATED WORK

Network sharing proposals have been discussed throughout the paper. To achieve the desired network sharing objectives, we have borrowed and extended ideas from many of these and we elaborate on this. Duffield et al. introduced the hose model [22], and its use to capture minimum bandwidth guarantees has been proposed

[4]This experiment evaluates the efficacy of the bandwidth allocation scheme used and is orthogonal to VM placement. Since *Baseline+* and Hadrian-P achieve per-flow fairness, they perform the same as *Baseline*.

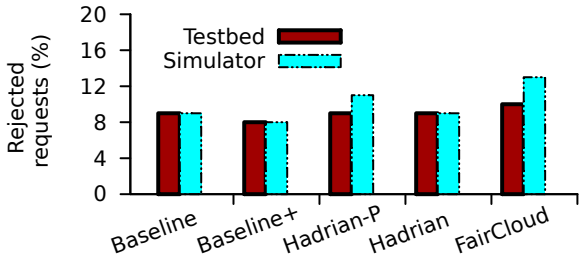

Figure 12: Testbed vs. simulator results.

in [13,14]. We extend this by adding communication dependencies and hierarchy. The latter implies per-tenant minimum guarantees for inter-tenant traffic. In a similar vein, Oktopus [17] and NetShare [18] offer per-tenant reservations and weights respectively.

FairCloud [13] describes a comprehensive set of properties for cloud network sharing. Of these, the "strategy proofness" property requires that a VM communicating with a set of VMs cannot increase its bandwidth allocation by modifying its traffic pattern. However, given this definition, a tenant can still strategize to gain an unfair share of any network link by increasing the set of VMs it communicates with. The examples in §5.3 show that for network sharing to be truly strategy proof, it should ensure bounded impact. FairCloud also proposes an allocation mechanism (OSPES) that yields minimum bandwidth guarantees. However, these guarantees depend on the network bisection bandwidth and on all other admitted VMs. Thus, without admission control, the guaranty can be arbitrarily small and as today, does not help tenants in estimating worst-case performance. Instead, Hadrian provides explicit lower bound on network performance.

There is a large body of work focusing on bandwidth guarantees (e.g., IntServ, ATM) and differentiated services (e.g., DiffServ) in the Internet. Key differences include the fact that the datacenter has a single operator who controls VM placement, controls the hypervisors and is offering minimum instead of hard guarantees. The problem of placing tenants, with its multi-dimensional constraints, is similar to testbed mapping [33] and virtual network embedding [34]. However, these efforts focus on mapping arbitrary virtual networks onto physical networks which restricts them to small physical networks.

7. CONCLUDING REMARKS

This paper looks at the problem of sharing the network in multi-tenant datacenters. We show that through careful VM placement and weighted bandwidth allocation, the dual goals of minimum and bounded VM bandwidth can be achieved. Apart from benefiting both tenants and providers in terms of their performance, this also ensures robust sharing of the network.

An alternative tact to tackle unfair sharing and malicious behavior is to change the pricing model to account for the network. However, even such multi-resource pricing requires VMs to be coupled with a lower-bound on network performance to be fair [14], and Hadrian offers this. It opens the door for differentiated pricing whereby a provider can charge based on a VM's minimum bandwidth, just as they do today based on the VM's processing capacity. This would elevate the network to a shared yet first-class datacenter resource.

8. REFERENCES

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

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

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

[4] "Measuring EC2 performance," http://bit.ly/48Wui.

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

[6] M. Zaharia, A. Konwinski, A. D. Joseph, Y. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *OSDI*, 2008.

[7] E. Walker, "Benchmarking Amazon EC2 for high performance scientific computing," *Login*, 2008.

[8] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *HPDC*, 2010.

[9] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the Outliers in Map-Reduce Clusters using Mantri," in *OSDI*, 2010.

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

[11] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS*, 2009.

[12] B. Craybrook, "Comparing cloud risks and virtualization risks for data center apps," 2011, http://bit.ly/fKjwzW.

[13] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing The Network In Cloud Computing," in *HotNets*, 2011.

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

[15] 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 *CoNext*, 2010.

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

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

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

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

[20] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *IEEE Infocom*, 2010.

[21] D. Kossmann, T. Kraska, and S. Loesing, "An Evaluation of Alternative Architectures for Transaction Processing in the Cloud," in *(SIGMOD)*, 2010.

[22] 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 *SIGCOMM*, 1999.

[23] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," in *SIGCOMM*, 2010.

[24] 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.

[25] 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 *SIGCOMM*, 2009.

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

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

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

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

[30] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul, "SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies." in *NSDI*, 2010.

[31] M. Issard, "Autopilot: Automatic Data Center Management," *ACM OSR*, vol. 41, 2007.

[32] "Amazon's EC2 Generating 220M," http://bit.ly/8rZdu.

[33] R. Ricci, C. Alfeld, and J. Lepreau, "A Solver for the Network Testbed Mapping problem," *SIGCOMM CCR*, vol. 33, 2003.

[34] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: substrate support for path splitting and migration," *SIGCOMM CCR*, vol. 38, 2008.

APPENDIX

LEMMA 1. *The aggregate weight for a VM p's flows cannot exceed its desired minimum bandwidth (B_p^{min}).*

PROOF. We use the terminology defined in the paper. Further, say $dst(p)$ is the set of destination VMs for p's flows. The aggregate weight for p's flows is

$$w_{\text{aggregate}} = \sum_{q \in dst(p)} w_{p,q} = \sum_{q \in dst(p)} min(\frac{B_p^{min}}{N_p}, \frac{B_q^{min}}{N_q})$$
$$\leq \sum \frac{B_p^{min}}{N_p} = \frac{B_p^{min}}{N_p} * N_p = B_p^{min}$$

□

THEOREM 1. *The maximum bandwidth for a VM p on link l is independent of the destination VMs p communicates with.*

PROOF. Say VM p communicates with VMs in $dst(p)$ through link l. B_p^l is aggregate bandwidth for all these flows. Since some of these flows may be bottlenecked elsewhere, B_p^l cannot exceed the sum of the bandwidth the flows would achieve on the link ($\sum B_{p,q}^l$). Hence,

$$B_p^l = \sum_{q \in dst(p)} B_{p,q} \leq \sum_{q \in dst(p)} B_{p,q}^l = \frac{\sum w_{p,q}}{w_T} * C_l$$
$$= \frac{\sum w_{p,q}}{\sum w_{p,q} + w_{T'}} * C_l \quad (5)$$

where $w_{T'}$ is the sum of weights for all non-p flows.

From Lemma 1, $\sum w_{p,q} \leq B_p^{min}$. Combining this with (5)

$$B_p^l \leq \frac{B_p^{min}}{B_p^{min} + w_{T'}} * C_l$$

□