Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks

Henrique Rodrigues[§], Jose Renato Santos[‡], Yoshio Turner[‡], Paolo Soares[§], Dorgival Guedes[§*]
[‡]Hewlett-Packard Laboratories (HP Labs) [§]Universidade Federal de Minas Gerais (UFMG)
[*] International Computer Science Institute (ICSI)

***Abstract*—Cloud environments should provide network performance isolation for co-located untrusted tenants in a virtualized datacenter. We present key properties that a performance isolation solution should satisfy, and present our progress on Gatekeeper, a system designed to meet these requirements. Experiments on our Xen-based implementation of Gatekeeper in a datacenter cluster demonstrate effective and flexible control of ingress/egress link bandwidth for tenant virtual machines under both TCP and greedy unresponsive UDP traffic.**

I. Introduction

In *cloud computing* [2] environments, mutually non-trusted *tenants* deploy their services in a shared datacenter infrastructure. Each tenant consists of a collection of one or more virtual machines (VMs) placed on one or more physical machines. Cloud environments have a strong requirement to enforce performance isolation among tenants that share a datacenter, but currently mechanisms are lacking to provide performance isolation for datacenter network I/O resources. Effective management of network bandwidth will be crucial to handle the growing range of service workloads that stress local area network resources in the datacenter. For example, data-intensive applications on scalable frameworks like MapReduce [5] can be highly network-intensive. Also, future datacenters will merge traditional messaging traffic with network storage traffic onto a single converged datacenter fabric, using new network standards [4], [6] and distributed storage and file systems [9].

This paper proposes properties that multi-tenant network performance isolation solutions should provide to meet the practical needs of both cloud users and cloud datacenter providers. We show that existing techniques fall short of meeting all of these requirements, and we report on our significant progress in building an I/O virtualization control system called Gatekeeper that is intended to fulfill these needs.

II. Requirements

We argue that solutions to the tenant network performance isolation problem must have the following properties to be practical.

Scalable. A cloud datacenter supports thousands of physical servers hosting 10s of thousands of tenants and 10s to 100s of thousands of VMs. The VMs and tenants in the datacenter come and go dynamically, with a high rate of churn. Each datacenter network link is thus potentially shared by a large and churning set of VMs. A solution for network performance isolation must work at these large scales. For example, techniques that require per-tenant or per-VM state to be maintained at each switch are impractical if the need to manage a large amount of state at high speed renders the switches prohibitively expensive for cloud computing infrastructure.

Simple service performance abstraction. The cloud infrastructure provider should clearly describe the performance level that users can expect when they deploy tenants. Typically, the cloud provider presents a menu of choices for the service level of VMs to deploy. For example, Amazon EC2 offers different "instance types" like small or medium instances. While each instance type offers a clear description of the CPU performance and memory and storage capacities, I/O performance is currently only vaguely specified. We argue that users should be offered more meaningful guidelines for expected network I/O performance, allowing users to better gauge the trade-off between service level and the monetary cost the users pay the cloud provider to host their tenants.

Robust to untrusted/malicious tenants. A key advantage of Infrastructure as a Service (IaaS) cloud computing is that it allows users to run arbitrary code as tenants, giving users great flexibility for innovation. But this flexibility has the downside of allowing tenants to execute malicious code that threatens to subvert the performance of other tenants or the datacenter infrastructure itself. A performance isolation solution should limit the performance impact that malicious tenants can inflict on others without restricting tenant code flexibility by, for example, mandating the use of TCP (versus UDP, etc.) or a particular implementation of the transport protocol.

Service level flexibility. Customers need deterministic guarantees ensuring predictable performance independent of VM placement and migration, and the traffic and churn of other tenants. However, deterministic guarantees can lead to overly conservative network resource allocation with severe underutilization of the physical resources. To achieve greater resource efficiency, the cloud provider should have the flexibility to offer service levels allowing tenants to exceed their minimum guarantees. The service level should specify both minimum and maximum bandwidth levels to trade-off determinism and resource efficiency. Supporting maximum rates is important for service providers that do not want their customers to get used to high high service levels and get disappointed if their services are later reduced to their minimum guarantees.

III. Existing mechanisms

TCP: TCP congestion control has been widely used to share network links across multiple flows. While TCP works well to provide best-effort service, it cannot enforce per tenant service guarantees. For example, if two tenants are sharing a single network link, and one tenant generates 99 TCP connections while the other generates only one, TCP will try to partition the bandwidth equally among the flows, giving 99% of the bandwidth to one tenant and only 1% to the other. Basically, TCP is designed to achieve fairness among flows and not among tenants.

Bandwidth Capping: Hypervisors such as VMware ESX and Xen have bandwidth capping mechanisms that enforce a maximum transmission rate for each virtual network interface (vNIC) associated with a virtual machine (VM). Bandwidth capping can be used to guarantee per vNIC transmission bandwidth using VM admission control to limit the total allocated bandwidth. More recent versions of hypervisors can also enforce receive bandwidth capping per vNIC. For well-behaved TCP connections, dropping packets that exceed the allocated bandwidth at the receiving vNIC causes TCP senders to reduce their rates and adapt to the available vNIC bandwidth. So using bandwidth capping could provide bandwidth guarantees at the server access links to the network in both TX and RX directions. However, this would require trusting that tenants run well-behaved TCP implementations. In essence, bandwidth capping is unable to control the ingress link bandwidth when tenants are not trusted. Another disadvantage of bandwidth capping is potential under-utilization of the link bandwidth. Using a more flexible traffic shaper such as Linux Hierarchical Token Bucket (HTB) can allow available bandwidth to be distributed to VMs with extra traffic and better utilize the link bandwidth. However, such schedulers can only be used in the transmit direction and cannot provide efficient use of receive bandwidth.

Secondnet: The Secondnet paper [11] describes a datacenter network allocation mechanism that provides bandwidth guarantees for traffic between each VM pair. We argue that providing end-to-end bandwidth guarantees for each pair of tenant VM is not the ideal model, from the tenant perspective. In general, tenants do not understand their applications' communication patterns well enough to specify their bandwidth requirements between each pair of VMs. Moreover, typical communication patterns are very dynamic, and the amount of data exchanged between any pair of VM will vary significantly over time. Creating bandwidth reservations in every network link in the path for every pair of communicating VMs is likely to be inefficient, since many reservations are expected to be unused at any time. In large datacenters, efficient use of network resources will only be possible in this model with statistical guarantees. However, this would require accurate statistical models of communication patterns in tenant applications which are very difficult to determine.

Seawall: The Seawall paper[19] describes a mechanism that allocates bandwidth on every link of a datacenter network by controlling rate limiters in the virtualization layer in each server at the edge of the network. Seawall's goal is to partition the bandwidth in each congested network link according to weights associated with each VMs sending traffic through that link. Congestion controlled tunnels between each pair of source and destination VM are created using sequence numbers added to each packet sent in the tunnel. Sequence numbers are stripped at the destination server and are used to detect packet losses due to network congestion. Upon receiving congestion notification messages from receivers, senders use network topology information to detect bottleneck links and adjust transmission rates at tunnels using that link. Rates are adjusted using weighted additive rate increase and multiplicative rate decrease functions, the goal of which is to partition the bandwidth in the bottleneck links according to the weights associated with VMs sending traffic to that link.

Seawall has several good design properties that are similar to our Gatekeeper design. First, since rates are enforced at the virtualization layer in the edge of the network, and tenant and rate state is distributed over the servers, the design is scalable to large datacenters. Second, the use of explicit feedback from receivers allows traffic to be throttled at the sources before they use network resources, and prevents a malicious VM to hog bandwidth in the network.

However, Seawall does not satisfy our predictable service level requirement. While Seawall can provide minimum guarantees if the maximum weight associated with each link is limited to a maximum value, it cannot enforce maximum rates to support deterministic behavior. More importantly, Seawall's bandwidth allocation does not divide the link bandwidth among tenants using the link, but among the total number of VMs sending traffic through that link. This favors tenants with a large number of VMs. For example, if a tenant has a single VM on a server but is receiving traffic from many senders it will use a significantly higher fraction of the server link receive bandwidth than a VM of a different tenant on the same server that has the same weight but is receiving traffic from just one sender. As we describe later, Gatekeeper will allocate the same bandwidth to each of the receiver VMs in this case because our service model jointly satisfies receiver and sender bandwidth guarantees.

AF-QCN: QCN (IEEE draft standard 802.1Qau) is a switch-based congestion control mechanism for datacenters. AF-QCN [12] proposes extensions to QCN for multi-tenancy. Like Seawall, AF-QCN divides link bandwidth among sending VMs without respect to receivers.

Netshare: Netshare [14] is the only mechanism that divides link bandwidth among tenants instead of sender VMs. However, it relies on a centralized bandwidth allocator which is difficult to scale to large datacenters and to deal with workload changes and the high rate of tenant and VM churn of cloud datacenters.

IV. Our approach

A. Service Model

A key design decision is to choose the form of network performance guarantees that should be provided to each tenant. We argue that tenants should be given a simple performance guarantee model that is easy for them to understand and specify. Figure 1 shows a simple model. In this model, all VMs of a tenant connect to a single logical non-blocking switch with guaranteed bandwidth on each access link. As it is common in real physical deployments to attach multiple servers directly to the same switch, this model should be familiar and easy to understand for users who deploy tenants in a datacenter. This model is similar to the hose model [7], [10] in which throughputs are constrained only by the guaranteed bandwidths of the access links of the VMs. The use of a single logical switch has also been proposed by others as a means of applying virtualization to the network [13], [3].

To obtain a better balance between determinism and efficiency, a tenant may be offered a variation of the above model in which a VM may exceed its guaranteed minimum bandwidth at times, if there is unused bandwidth on the physical links. The amount by which a VM may exceed its minimum guarantee can be limited to a specified maximum rate, and potentially could be assigned based on a dynamic pricing scheme like a spot market.

The model can be further extended to allow composition of multiple logical switches. That is, a VM can have multiple access links each attached to a different logical switch. For example, as shown in Figure 2, in a 3-tier web service one logical switch could be used to interconnect VMs of the web server and application server tiers, and a second logical switch could connect VMs of the application server and database tiers. Each application server VM would have two access links, one attached to each logical switch with an independently specified rate.

B. Reserving link bandwidth

Mapping the simple tenant performance model in Figure 1 to link bandwidth reservations of an arbitrary datacenter network topology is a dificult task. Tenant applications can generate many different communication patterns that could satisfy their access link bandwidth guarantees, but generate completely different demands on each network link.

We argue that it is useful and feasible to solve a subset of this general problem that is of particular importance in practice. In particular, several recent advances in datacenter networking research [10], [17], [1], [16], [18], commercial products [8], and Ethernet standards [20] promise to make it practical to cost-effectively scale the bisection bandwidth of large datacenter networks using multi-path switching. Even with traditional datacenter networks, network topology-aware placement of service workloads can provide full bisection bandwidth among the tenant VMs [15].

Our key observation is that using emerging scalable networks or placing tenants in bisection network regions shifts the bottleneck from the network fabric to the endpoint links that connect each physical server to the network fabric. This allows translating the problem of managing tenant network bandwidth into the more tractable problem of managing each server's network access links. Thus, tenant bandwidth management can focus on the endpoint server links, which are potentially shared by all VMs hosted on a server, instead of having to reason about network bottlenecks that could arise anywhere in the fabric which are difficult to predict without an accurate traffic pattern model.

V. Gatekeeper Architecture

Our Gatekeeper system provides network isolation for multi-tenant datacenters using a distributed mechanism implemented at the virtualization layer of each datacenter server. Gatekeeper achieves scalability using a simple point-to-point protocol and minimal datacenter-wide control state.

Gatekeeper controls the usage of each server's network access link. It provides per-vNIC link bandwidth guarantees in both directions of the network link at each physical server, i.e., for both ingress and egress traffic. Minimum bandwidth guarantees are achieved using an admission control mechanism that limits the sum of guarantees to the available physical link bandwidth. Each vNIC can exceed its guaranteed allocation when extra bandwidth is available at both transmitting and receiving endpoints. However, to provide deterministic behavior Gatekeeper limits each vNIC bandwidth to a maximum rate. By configuring the maximum rate, the system administrator can tradeoff determinism for efficiency. Complete determinism is provided by setting equal maximum rate and minimum guarantee. Maximum efficiency is provided by having no maximum rate limit. Operation between these extremes is provided by setting the maximum to a factor of the guaranteed rate.

For scheduling transmission bandwidth, Gatekeeper uses a traditional weighted fair scheduler that provides minimum bandwidth guarantees. For controlling receive bandwidth, Gatekeeper monitors the receive traffic rate at each vNIC and the physical link and determines the receive bandwidth allocation to each vNIC at periodic intervals (10 ms in our current implementation), taking into account the link usage and the minimum and maximum rates for each vNIC. If a vNIC receive bandwidth exceeds its computed allocation, Gatekeeper sends a feedback message to other remote Gatekeeper instances hosting VMs contributing to its traffic. The feedback message includes an explicit rate that is computed by distributing the desired vNIC receive rate to the senders.

Figure 3 shows an overview of the Gatekeeper architecture. Gatekeeper has a set of rate limiters associated with each vNIC interface. A root rate limiter enforces the maximum transmission rate of each vNIC. Additional rate limiters are dynamically created to reduce the rate of traffic sent to remote congested vNICs. A packet filter classifies outgoing packets based on their MAC address and direct then to the appropriate rate limiter.

In the absence of congestion notification messages the rate limit is increased in periodic intervals according to a linear function, and if it reaches the maximum rate the limiter is removed after a timeout interval.

Gatekeeper also keeps a dynamic set of counters associated with each vNIC. These counters measure the rate between every pair of communicating vNICs. The counters are created and deleted dynamically based on the active set of remote vNICs sending traffic to the corresponding local vNIC. Every counter also stores the MAC address of the corresponding remote vNIC which is used to send feedback messages[1]. Counters are created when packets from new sources are received and deleted after timeouts (we extended the Open vSwitch per flow packet counters to measure traffic rates). Periodically (every 10 ms in the current prototype) the measured rates are used to determine new allocated RX rates, and congestion feedback messages are generated if needed. A congestion message is generated if the aggregate rate on the physical link exceeds a given threshold (95% of the link bandwidth in the current prototype) or if a vNIC exceeds its maximum rate. If the aggregate rate exceeds the threshold, congestion feedback is generated for the vNIC that is exceeding its guaranteed receive rate by the largest relative amount. Gatekeeper sets the desired receive vNIC rate to its minimum guarantee and divides this rate among its active senders. A sender is considered active if its measured rate exceeds a threshold. A congestion feedback message is sent to each active sender with this explicit rate. The feedback message also includes the number of senders of the same tenant that are contributing to the receiver congestion. The sender uses this information to calibrate its rate increase function, such that the aggregate rate increase function at the receiving vNIC is independent of the number of senders.

Our current rate decrease function causes traffic exceeding its guarantee to be reduced to its minimum guarantee. This may be too aggressive and the link can become under-utilized for some time. An open question left for future work is to understand the tradeoff of different response functions that trade fast reaction to congestion versus fast recovery of available bandwidth.

VI. Evaluation

We implemented a Gatekeeper prototype on Xen 3.4.2 using the Open vSwitch 1.1.0pre2 (www.openvswitch.org) in Dom0 running Linux 2.6.34.6. Our prototype extends the Open vSwitch flow table to track flow rates. We use Linux hierarchical token bucket (HTB) scheduler to implement our link scheduler and rate limiters. Our current implementation does not yet support dynamic creation and deletion of rate limiters; we use a preconfigured set of limiters that matches our experimental setup.

We evaluate Gatekeeper for simple scenarios using a configuration with five servers, each with a one Gb/s Ethernet interface connected to a single switch, as shown in Figure 4. The system hosts two tenants. Tenant A has two VMs and tenant B has four VMs. One shared host runs one VM from each tenant, while the others run a single VM each. Each tenant runs a netperf (www.netperf.org) microbenchmark between its VM in the shared host and all its other VMs in the other hosts, i.e. tenant A runs one netperf flow and tenant B runs 3 netperf flows. We examine two scenarios: 1) transmit (TX) bottleneck where traffic is transmitted from each VM in the shared host to the other VMs of the same tenant, and 2) receive (RX) bottleneck where traffic is transmitted from the hosts with a single VM to the shared host. We allocate 70% of each server link bandwidth to tenant A and 30% to tenant B. Tenant A is a well behaved tenant running a single TCP connection. We consider three cases for tenant B traffic type: **a)** no traffic **b)** 3 well behaved TCP flows, **c)** 3 UDP flows representing a malicious tenant that does not use a well behaved TCP stack.

Figures 5 and 6 show the results. We consider four different bandwidth allocation mechanisms: **1)** No control, **2)** RX and TX bandwidth capping, **3)** Gatekeeper with equal maximum rate and minimum guarantee, **4)** Gatekeeper without maximum rate. The horizontal dotted lines show the ideal bandwidth shares for tenant A and tenant B given their minimum guarantees.

The results show that while bandwidth capping works well for well behaved tenants with TCP traffic, it cannot enforce bandwidth allocation for "misbehaving" tenants that generate unresponsive traffic. In addition, bandwidth capping cannot take advantage of unused bandwidth. Gatekeeper on the other hand can enforce the desired bandwidth allocation even for misbehaving tenants with unresponsive traffic for both the transmit and receive scenarios. Furthermore, Gatekeeper can take advantage of unused bandwidth both at the transmit and receive sides up to a maximum rate specified by the system administrator for each vNIC.

VII. Conclusion

There is a wide variation of network performance guarantees that can be offered to tenants in Cloud computing environments. We argue that current models are not satisfactory and propose a simple tenant performance model abstraction. We describe the Gatekeeper mechanism that supports this performance model in virtualized data centers. Our preliminary results show that Gatekeeper works well in simple scenarios. As part of future work, we plan to evaluate Gatekeeper behavior for larger configurations and dynamic workloads and explore alternative congestion response functions. In addition, our current implementation implements increase/decrease policies and congestion feedback generation in a user level daemon in Xen Dom0. While this approach facilitates policy experimentation, it adds overhead to the implementation because of kernel-user crossings. Currently, turning on Gatekeeper increases CPU load on Xen Dom0 by around 10% of a CPU core (Intel i7-930 2.8GHz) to manage a 1Gbps link under some traffic scenarios. We plan to migrate policy functions

[1]Our current prototype is based on the Open vSwitch (www.openvswitch.org).

from user space to kernel level to minimize the CPU cycles consumed by Gatekeeper.

ACKNOWLEDGEMENTS

This work was partially sponsored by Fapemig, CNPq, HP Brazil, UOL's Research Scholarship Program Proc. 20100214110700, and the Brazilian National Institute of Science and Technology for the Web, InWeb. Grants no. 573871/2008-6 and 141322/2009-8.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference*, pages 63–74, Seattle, WA, Aug. 2008.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb. 2009.

[3] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[4] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, and P. Grun. Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options. In *Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects (HOTI '09)*, New York, NY, Aug. 2009.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI '04)*, pages 137–150, San Francisco, CA, Dec. 2004.

[6] C. DeSanti and J. Jiang. Fcoe in perspective. In *Proceedings of the 2008 International Conference on Advanced Infocomm Technology (ICAIT '08)*, pages 1–8, Shenzhen, China, July 2008.

[7] 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 *Proceedings of the ACM SIGCOMM 1999 Conference*, pages 95–108, New York, NY, USA, 1999. ACM.

[8] Fulcrum Microsystems. FocalPoint in large-scale Clos switches, Oct. 2007.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 29–43, Bolton Landing, NY, Oct. 2003.

[10] 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 *Proceedings of the ACM SIGCOMM 2009 Conference*, pages 51–62, Barcelona, Spain, Aug. 2009.

[11] 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 *Proceedings of the 6th International Conference*, Co-NEXT '10, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

[12] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pages 58 –65, 2010.

[13] E. Keller and J. Rexford. The "platform as a service" model for networking. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 4–4, Berkeley, CA, USA, 2010. USENIX Association.

[14] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese. NetShare: Virtualizing data center networks across services. Technical Report CS2010-0957, University of California, San Diego, May 2010.

[15] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz. A case for topology-aware resource allocation for data-intensive applications in the cloud. Technical Report HPL-2009-335, HP Labs, Palo Alto, CA, 2009.

[16] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, Apr. 2010.

[17] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, S. R. Pardis Miri, V. Subramanya, and A. Vahdat. PortLand: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference*, Barcelona, Spain, Aug. 2009.

[18] M. Schlansker, J. Tourrilhes, Y. Turner, and J. R. Santos. Killer fabrics for scalable datacenters. In *IEEE International Conference on Communications (ICC)*, May 2010.

[19] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Sharing the data center network. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, March-April 2011.

[20] J. Touch. RFC 5556: Transparent interconnection of lots of links (trill): Problem and applicability statement, May 2009.

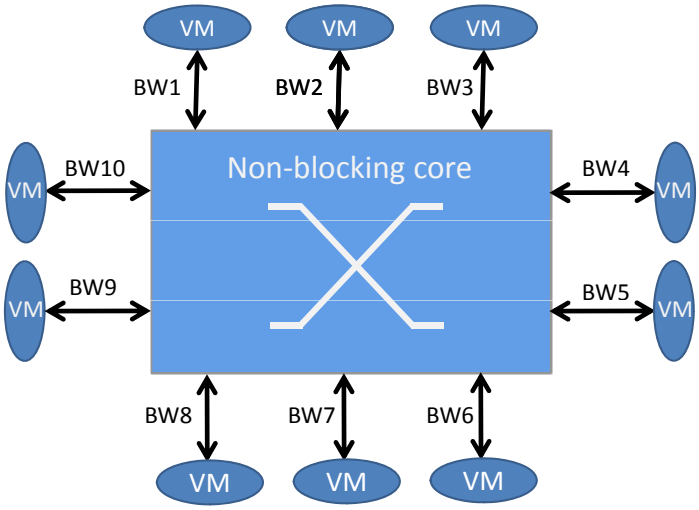

Fig. 1. Tenant Model

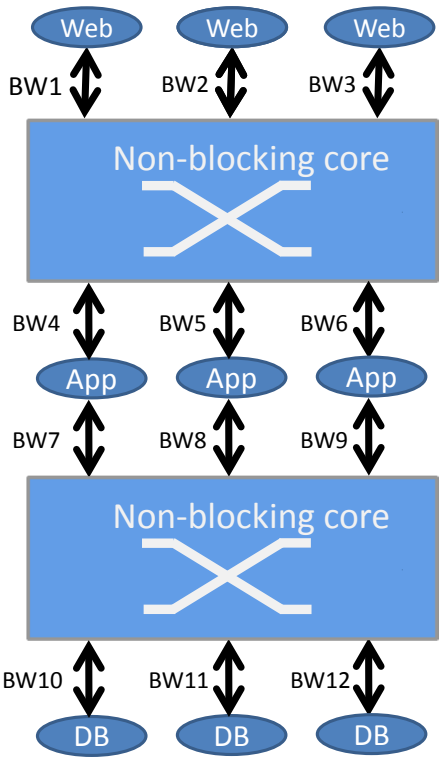

Fig. 2. Tenant Model with Logical Switch Composition

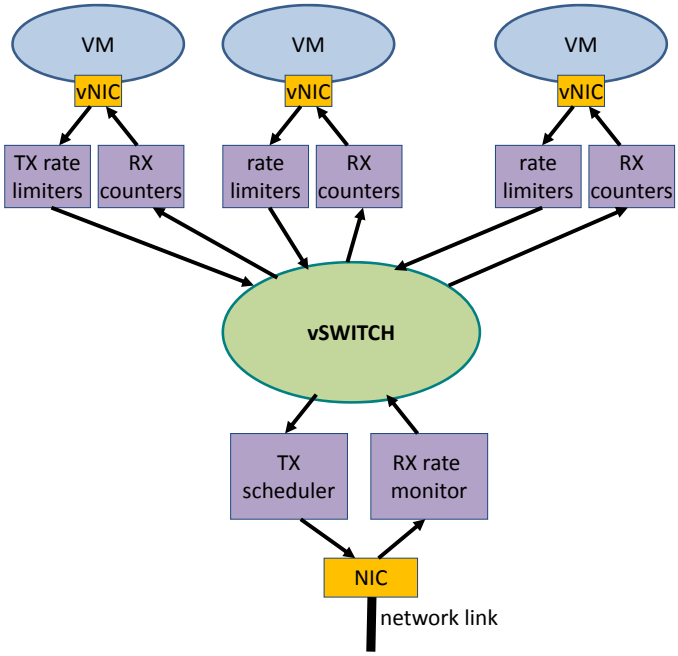

Fig. 3. Gatekeeper architecture

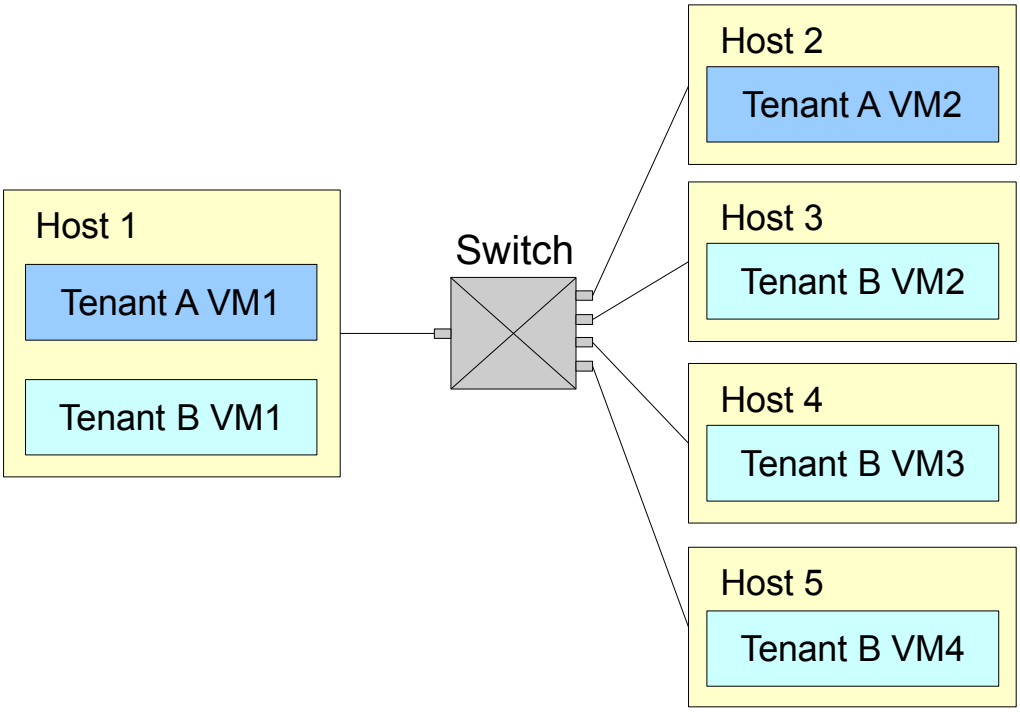

Fig. 4. Experimental setup

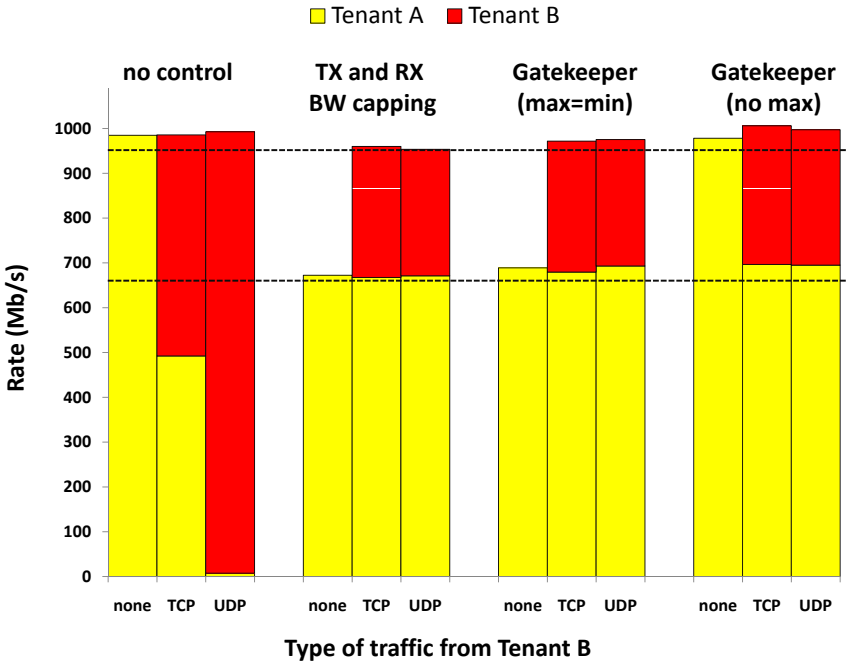

Fig. 5. Transmit Scenario Results

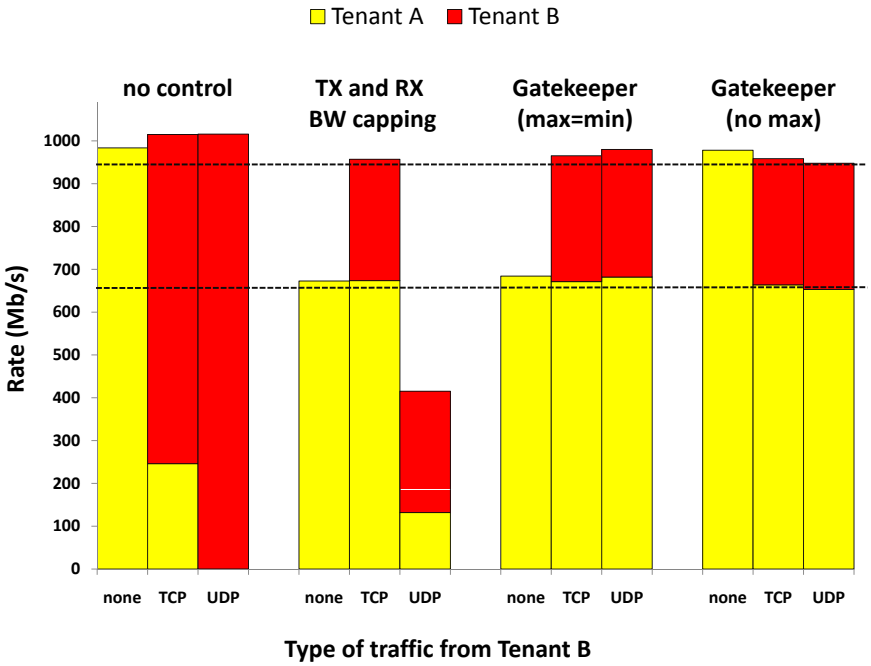

Fig. 6. Receive Scenario Results