

EyeQ: Practical Network Performance Isolation for the Multi-tenant Cloud

Vimalkumar Jeyakumar
Mohammad Alizadeh
David Mazières
Balaji Prabhakar
Stanford University

Changhoon Kim
Windows Azure

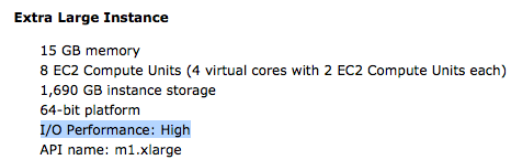
Abstract

The shared multi-tenant nature of the cloud has raised serious concerns about its security and performance for high valued services. Of many shared resources like CPU, memory, etc., the network is pivotal for distributed applications. Benign, or perhaps malicious traffic interference between tenants can cause significant performance degradation that hurts performance of applications, and hence, impacts their revenue. Network performance isolation is particularly hard because of the distributed nature of the problem, and the short (few RTT) timescales at which they manifest themselves. This problem is further exacerbated by the large number of competing entities in the cloud, and their volatile traffic patterns.

In this paper, we motivate the design of our system called EyeQ, with the goal of providing predictable network performance to tenants. The enabler for EyeQ is the availability of high bisection bandwidth in data centers. The key insight is that by leaving a headroom of (say) 10% of access link bandwidth, EyeQ simplifies dealing with potentially a global contention problem into one that is *mostly* local, at the sender and receiver. This allows EyeQ to enforce predictable network sharing completely at the end hosts, with minimum support from the physical network.

1 Introduction

The shared, multi-tenant nature of cloud providers has raised concerns about their security and performance. Many cloud customers have reported the “noisy-neighbour” [2, 4] problem, where performance of the system is unpredictable if a colocated tenant tries to grab resources (CPU, disk, IO) disproportionately. While hypervisors are equipped with mechanisms to deal with CPU, memory and disk isolation, network isolation has attracted attention only recently. To date, commercial



```
Extra Large Instance
15 GB memory
8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each)
1,690 GB instance storage
64-bit platform
I/O Performance: High
API name: m1.xlarge
```

Figure 1: Screenshot of an offering from Amazon EC2. CPU, disk and memory are allocated in familiar units. In contrast, the units of “IO,” is unclear.

offerings of network performance document only *reachability* isolation. But, network *performance* isolation is particularly vital for scale-out distributed services which have a demanding network component, unlike CPU intensive jobs.

Many cloud service providers today have some means of allocating CPU, memory and disk resources, in *familiar units* such as “virtual” cores, GB of memory and disk capacity (Figure 1). When it comes to IO, which includes the network, the units are either absent, or nebulous: “low, moderate and high.” Customers do not get a clear picture of their network resource guarantees, and are unable to cope with bad performance. Some of them either give up [3] on the cloud citing bad performance, or significantly rearchitect their applications. For example, Netflix reported that their own data center networks offered good performance, which afforded them to build chatty applications; but on Amazon Web Services, they redesigned their infrastructure to deal with performance variability [1]. Our experience in talking to customers suggests that they would like to have *predictable* performance, as if the network allocated to them were *dedicated*.

Conventionally, network sharing has been enforced by making the network aware of competing classes of traffic, i.e., by configuring Class-Of-Service queues exposed by many commodity switches. However, as noted by Shieh et. al [18], the number of queues in switches has

not evolved beyond 8 or 16 queues per port. Hence, we need a scalable mechanism to cope up with number of tenants in the cloud. In the rest of this paper, we focus on the problem of sharing the network at large scale: $\sim 10\text{K}$ tenants, $\sim 100\text{K}$ servers and $\sim 1\text{M}$ VMs. Providing predictable network performance—i.e. guarantees on bandwidth, latency, jitter—is hard because, (a) unlike CPU, memory and disk, network contention is a distributed problem: contention can occur anywhere in the network; (b) the number of contending entities and their traffic characteristics are diverse: tens of thousands of tenants with different flavours of TCP, UDP; (c) very short timescale (few milliseconds) contention can affect long term performance (as shown in Figure 2).

To enable a customer to set clear expectations of network performance, we envision that a cloud provider should be able to provision network resources for its tenants, starting with bandwidth, in familiar units: *bits per second*. The provider assures each instance (VM) a guaranteed *minimum* bandwidth, so that the VM is able to transmit and receive at least the chosen capacity. The provider then supports a simple performance abstraction where the customer’s ‘virtual’ network of VMs, in aggregate, perform as if they were connected to a *single switch* with full bisection bandwidth. We asked ourselves, “What are the requirements of a mechanism that tries to enforce such guaranteed performance?” We arrived at the following requirements by talking to developers and operators of a cloud provider; the requirements are by no means complete, but some of these are highlighted by other proposals [6, 18] as well.

1. **Performance guarantee** The mechanism should work in the worst case: it should guarantee bandwidth to VMs, irrespective of the application demands, traffic communication patterns, and network activity of VMs belonging to other tenants.
2. **Simplicity** The mechanism should be simple and should co-exist with network stacks of customers’ VMs. It should not rely on any kind of extensive signalling from the application about its bandwidth demands, and should not require customers to overhaul their applications.
3. **High Utilization** The mechanism should retain the statistical multiplexing benefits of the cloud, and not statically allocate resources.
4. **Scalability** The mechanism should scale efficiently with the number of tenants, VMs, servers; the size and speed of the network, i.e. 1Gbps, 10Gbps, 40Gbps, and beyond.

Related Work: Recently, hypervisor based mechanisms have been proposed that mitigate the effect of

many (possibly) malicious traffic patterns. Seawall [18] tackled the problem by tunneling packets through end-to-end congestion-controlled tunnels. On a link, this achieves weighted max-min fairness between multiple source VMs. The inherent problem of weighted sharing is that a *tenant’s* bandwidth share depends on the unknown activity (i.e., number of senders) of other active tenants; this violates our first requirement. Oktopus [6] also made the case for predictability in data center networks, and used an end-to-end mechanism combined with intelligent placement to provide strict bandwidth guarantees. However, Oktopus operated at fairly large timescales (2 seconds). As we will see in §2, it is important for performance isolation mechanisms to react quickly, within a few round trip times. Moreover, Oktopus (and SecondNet [13]) *statically* reserve bandwidth, which violates our third requirement.

The rest of the paper is organized as follows. Section 2 further elaborates the difficulties of sharing the network. Section 3 lays down some design choices that greatly simplify mechanisms for sharing network bandwidth, and Section 3.2 presents a preliminary design for enforcing predictable network performance. Finally, Section 4 concludes with some directions for future work.

2 Why is network sharing hard?

In this section, we will first see why network sharing is a hard problem, and then discuss various scenarios in which performance interference can occur.

Distributed Nature: In a general network topology, contention (or congestion) for bandwidth and latency can happen at any hop inside the network. To cope with increasing application bandwidth demands, modern data center networks have very high bisection bandwidth. Even in such network topologies with large bisection bandwidth, congestion can happen *anywhere* if the traffic does not conform to the hose model [11] (i.e., if the traffic matrix is non-admissible). This situation arises when any link in the network receives traffic at a sustained rate greater than its capacity. If the traffic matrix is admissible, then it is difficult to congest the network core, but it is easy to congest any single “port,” i.e., an end-host. This happens if more than one host sends data simultaneously to a receiver (N to 1 traffic pattern).

Traffic diversity: Many cloud environments host tens of thousands of tenants, allowing them to bring their own code and OS. This allows customers to tweak settings in their network stack; use different TCP implementations that have better performance; use multiple TCP connections to boost throughput; disable congestion control, or even use an unreactive UDP session to transmit data.

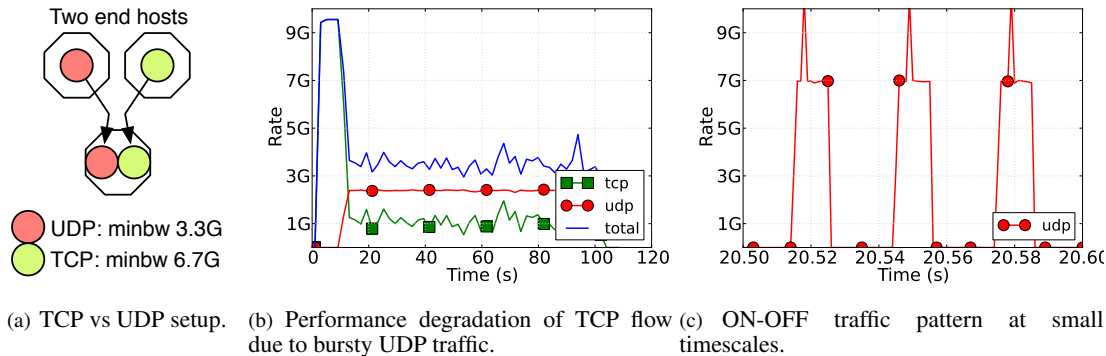


Figure 2: At large timescales (2 second intervals), the UDP tenant’s demand is seemingly small. But, at a finer timescale, its bursty nature has an adverse effect on the dynamics of TCP. These finer timescale interactions put a stringent performance requirement on the reaction times of any isolation mechanism, and mechanisms that react at large timescales may not see the big picture.

This diverse nature of application traffic and unknown communication patterns present a significant challenge.

Fine timescales: The maximum end-to-end latency within a data center is becoming smaller, and is currently of the order of few $100\mu s$. Since a majority of applications use TCP—which operates at RTT timescales—interactions at small timescales could affect performance. To understand why timescales matter, consider an experiment where there are two tenants TCP (green) and UDP (red), and say we wish to share bandwidth between TCP and UDP in the ratio 2:1. Both tenants send traffic to their servers, that are colocated on an end-host. The TCP tenant starts one long lived flow, but the UDP tenant sends traffic in an ON-OFF bursty fashion so that its average rate is about 2.3Gbps (ON for 10ms, OFF for 20ms, at 7Gbps maximum). Figure 2(b) shows the per-tenant *receive* throughput for both TCP and UDP, aggregated over a 2 second period. The traffic looks benign at such large timescales (Figure 2(b)), when it is actually very bursty at smaller timescales (Figure 2(c)). Such bursty traffic patterns are not uncommon for many multi-tiered applications [7]. The fine timescales of interactions places stringent requirement on the reaction times of any mechanism.

3 Consequences on Design

3.1 Network Design

Such stringent requirements of providing predictable performance guarantees to tenants do exist for cloud providers. The network design plays a crucial role in dealing with network resource contention within the data center. Since servers’ capital and operational costs account for a large fraction ($\sim 85\%$ [12]) of the total costs

of a data center, it is important that the network does not prevent the servers from achieving their full potential [17].

If one is to design a network that offers predictable performance in the presence of unknown and random traffic patterns, the network necessarily needs to have a high, if not full-bisection bandwidth. In hindsight, it is not surprising that many recent research efforts [11, 14] have focused on how to build such high bandwidth interconnects from cheap commodity hardware. Further, to ensure that the available bisection bandwidth is well utilized, there is a need for intelligent routing mechanisms; this is an area of active research [5, 16]. These mechanisms bring us one step closer to realizing the data center network as one *giant switching fabric* [19].

3.2 Mechanism design

If one were to view the data center network as a giant switch, how would a bandwidth arbitration mechanism look like? While previous work such as MPTCP [16] and Hedera [5] focus on maximizing the total throughput of the data center fabric, we now discuss a possible answer to the above question, where the goal is enforcing fairness across multiple traffic classes.

Addressing distributed contention: Algorithms for bandwidth sharing (such as weighted round robin), and their implementations (such as deficit round robin), have been designed with the model of an output queued switch. In this model, bandwidth contention is visible locally, at the output port of a switch, where it can be resolved using packet scheduling algorithms. Hence, switch designs that try to provide Quality of Service guarantees strive to emulate an output queued switch, using a switching fabric that has some *speedup* over the

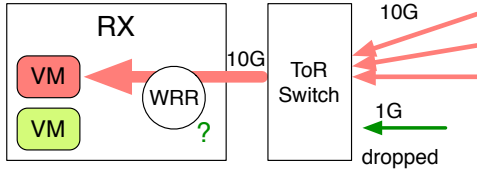


Figure 3: The difficulty of detecting access link contention on the receive path.

edge link capacity. For any arrival traffic pattern to the switch, it has been shown that a speedup of about 2 is necessary [8], to perfectly emulate an Output Queued Switch. In practice, it has been found that a smaller speedup (between 1.1 and 1.2) usually suffices [10] to have the same benefits of an Output Queued Switch. The purpose of speedup is to simultaneously ensure that (a) the fabric is not a bottleneck; and (b) contention is moved to the edge (i.e., the output queue), where it can be detected and resolved locally.

The above observation guides the design of our mechanism. Viewing the network as a giant switching fabric greatly simplifies a global network contention problem to a local one: contention for transmit and receive bandwidth of the access link. Contention on the transmit side happens first within the end host, which can be resolved by packet scheduling mechanisms at the VSwitch. Unfortunately, contention at the receiver first occurs at the access link, which is inside the network. As shown in Figure 3, in-network congestion can cause network demands of a VM to go undetected. It is important that the access link contention be quickly detected. To understand this, consider the example shown in Figure 2. When UDP bursts at 7Gbps, the access link saturates (at short-timescales) leading to packet drops. Since the switch is not tenant aware, it drops both TCP and UDP packets, to which TCP aggressively reacts by exponentially backing-off. This ‘elasticity’ of TCP hides the true demand for bandwidth, and hence, mechanisms that react at timescales larger than a few RTTs cannot differentiate between two cases: one where there is a genuine lack of demand, and the other, where TCP has backed-off.

Introducing a small speedup to the network fabric (over the edge links) mitigates contention from the network, moving it completely to the edge, i.e., the VSwitch port inside the hypervisor. This allows the VSwitch to detect impending network congestion, and accurately account for congestion on a per-VM basis. The speedup effectively gives headroom to prevent packet drops in the network, and allows the VSwitch to tease out the true network demand from the underlying TCP flow, without requiring any interaction with the VM’s TCP stack. By

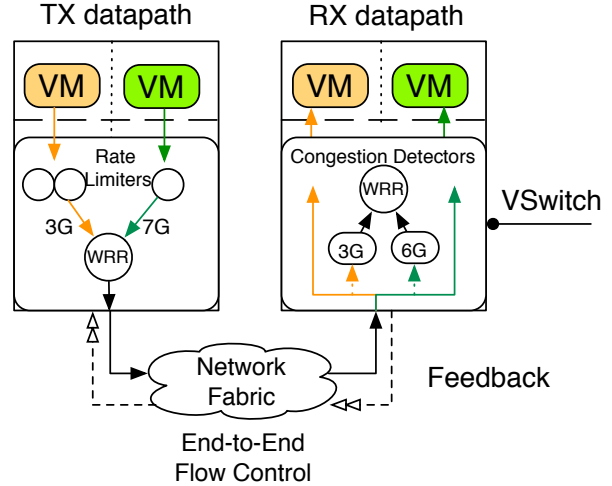


Figure 4: Overall system architecture.

measuring the rate at which each VM is receiving traffic at the congestion detectors, the VSwitch signals sources that exceed their fair share. Thus, the VSwitch can enforce rich bandwidth sharing policies—such as minimum, maximum, or weighted bandwidth guarantees—in an end-to-end fashion.

To achieve the effect of speedup, we slow down the end-hosts by detect congestion when the access link utilization exceeds some fraction γ (less than, but close to 1). This bandwidth headroom of $(1 - \gamma)$ may seem like a big price to pay, but in some cases, it actually leads to better link utilization (as shown in Figure 5).

Taming traffic diversity: Our notion of performance isolation translates to providing minimum end-to-end bandwidth guarantees to tenants. Thus, to be agnostic to the type of traffic, we treat packets between source-destination pairs as a single *meta-flow* whose aggregate rate is controlled through end-to-end congestion control, irrespective of the tenants’ network stack.

Embracing timescales: The example shown in Figure 2(a) illustrates the need to react quickly, at timescales of the order of round-trip times. Implementing the end-to-end rate control mechanism in a distributed fashion, in the datapath makes it possible to react to congestion in a timely manner, within a few RTTs.

3.3 EyeQ Architecture

Figure 4 shows the high level overview of EyeQ’s transmit (TX) and receive (RX) datapaths. The TX datapath consists of rate limiters to enforce admission control. Contention at the transmit side is resolved by using a TX-weighted round robin (WRR) scheduler that as-

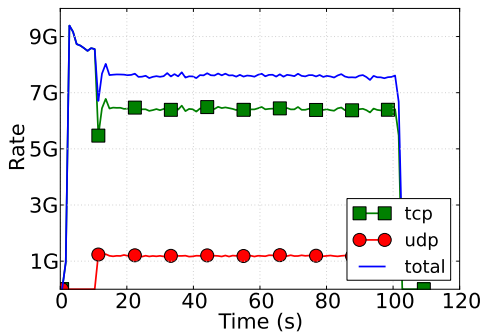


Figure 5: A preliminary prototype of our design helps mitigate the harmful bursty nature of UDP traffic, and let TCP traffic attain its minimum bandwidth guarantee of 6.6Gbps. Figure 2(b) shows the harmful effect of UDP traffic without our isolation mechanism.

sure each VM its (egress) minimum bandwidth guarantee. To ensure that traffic does not congest any receiver, there are multiple per-destination rate limiters. This is analogous to ‘Virtual-Output-Queues’ in switches; multiple per-destination rate limiters prevent packets to uncongested destinations from head-of-line blocking other packets. These rate limiters vary their sending rate periodically using a control loop similar to TCP’s AIMD process, using feedback generated by the RX datapath.

The RX datapath consists of a number of congestion detectors, one per VM. Each detector is assigned a “fair rate” by the RX-WRR scheduler, and generates feedback whenever the VM exceeds its allotted rate. The congestion detector is clocked by the arrival of packets; if a packet arrives to a VM, and the VM’s rate exceeds its share, the congestion detector sends a feedback to the source of the packet. The feedback can be anything: a single bit (such as ECN), or an explicit rate (such as RCP [9]). The RX-WRR scheduler enforces speedup by splitting a maximum of γC between the VMs, where C is the physical NIC capacity. The RX datapath works in tandem with the TX datapath to enforce end-to-end flow control.

4 Discussion and Future Work

In this paper, we discussed a mechanism for enforcing network performance isolation in a large multi-tenant environment. We have implemented this design as a Linux Kernel Module, and tested it against many adversarial traffic patterns similar to the scenario discussed in Figure 2(a). We found that EyeQ is able to mitigate the harmful effects of malicious traffic. In particular, Figure 5 shows how EyeQ protects the TCP tenant from bursty UDP traffic, while simultaneously improving the

total network utilization from about 4Gbps (Figure 2(b)) to about 8Gbps.

In-network contention: While high bandwidth network designs present lesser opportunity for in-network contention, it does not eliminate its possibility. EyeQ does not ignore this possibility, but gracefully falls back to per-sender max-min fairness, using tenant agnostic congestion notification mechanisms such as ECN. If the network gets congested on ‘large’ timescales (such as a few hours), it strongly indicates an unbalanced system design. We believe that the right approach is to invest more on the network, so that it does not “get in the way” of providing customer satisfaction.

Other Scenarios: The mechanism presented in this paper focuses on a particular kind of traffic pattern, which we call “intra-tenant;” communication between VMs of the same tenant happens over a high speed network interconnect, interfering with similar communication patterns of other tenants. However, cloud networks also host services like memcached clusters, storage, load balancers. These services are usually implemented as tenants, and hence “inter-tenant” communication can also result in performance interference. While we demonstrated EyeQ’s ability to enforce minimum bandwidth guarantees on a per-VM basis, EyeQ can directly benefit from techniques such as Distributed Rate Limiting [15] that enforce a limit on *aggregate* bandwidth consumption.

Also, a data center has a lot of network capacity, but typically only has ~ 100 Gbps uplink bandwidth to the Internet. Since tenants share this uplink, it is important to have automated defense mechanisms to protect this bandwidth that is crucial for the infrastructure. And finally, it is equally important to protect the downlink, and defend against attacks originating from the Internet; this is the holy grail of Internet Quality of Service.

5 Conclusion

We presented EyeQ, a platform to enforce network sharing policies at large scale. By viewing the data center network as a giant switch, and by trading off a small fraction of the access link bandwidth, EyeQ is able to assure a guaranteed minimum bandwidth guarantees to VMs in a timely fashion, completely at the edge, with minimum support from the network.

References

- [1] 5 lessons we’ve learned using AWS. <http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html>.

- [2] Has Amazon EC2 become over subscribed? http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm.
- [3] Mixpanel—Why we moved off the cloud. <http://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud/>.
- [4] VMWare: Rethink IT: Getting rid of noisy neighbours. <http://blogs.vmware.com/rethinkit/2010/09/getting-rid-of-noisy-cloud-neighbors.html>.
- [5] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation* (2010), USENIX Association.
- [6] BALLANI, H., COSTA, P., KARAGIANNIS, T., AND ROWSTRON, A. Towards predictable data-center networks. In *ACM SIGCOMM* (2011).
- [7] BENSON, T., ANAND, A., AKELLA, A., AND ZHANG, M. Understanding data center traffic characteristics. *ACM SIGCOMM CCR* (2010).
- [8] CHUANG, S., GOEL, A., MCKEOWN, N., AND PRABHAKAR, B. Matching output queueing with a combined input/output-queued switch. *Selected Areas in Communications, IEEE Journal on* (1999).
- [9] DUKKIPATI, N., MCKEOWN, N., AND FRASER, A. Rcp-ac: Congestion control to make flows complete quickly in any environment. In *INFOCOM 2006*.
- [10] FIROOZSHAHIAN, A., MANSHADI, V., GOEL, A., AND PRABHAKAR, B. Efficient, fully local algorithms for cioq switches. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE* (2007), IEEE, pp. 2491–2495.
- [11] GREENBERG, A., HAMILTON, J., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D., PATEL, P., AND SENGUPTA, S. VL2: a scalable and flexible data center network. *ACM SIGCOMM* (2009).
- [12] GREENBERG, A., HAMILTON, J., MALTZ, D., AND PATEL, P. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review* (2008).
- [13] GUO, C., LU, G., WANG, H., YANG, S., KONG, C., SUN, P., WU, W., AND ZHANG, Y. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International Conference* (2010), ACM, p. 15.
- [14] NIRANJAN MYSORE, R., PAMBORIS, A., FARRINGTON, N., HUANG, N., MIRI, P., RADHAKRISHNAN, S., SUBRAMANYA, V., AND VAHDAT, A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM* (2009), ACM.
- [15] RAGHAVAN, B., VISHWANATH, K., RAMABHADRAN, S., YOCUM, K., AND SNOEREN, A. Cloud control with distributed rate limiting. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (2007), ACM.
- [16] RAICIU, C., BARRE, S., PLUNTKE, C., GREENHALGH, A., WISCHIK, D., AND HANDLEY, M. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM* (2011).
- [17] RASMUSSEN, A., PORTER, G., CONLEY, M., MADHYASTHA, H., MYSORE, R., PUCHER, A., AND VAHDAT, A. Tritonsort: A balanced large-scale sorting system. In *Proceedings of NSDI* (2011).
- [18] SHIEH, A., KANDULA, S., GREENBERG, A., KIM, C., AND SAHA, B. Sharing the data center network. In *NSDI, USENIX* (2011).
- [19] WANG, G., ANDERSEN, D., KAMINSKY, M., KOZUCH, M., NG, T., PAPAGIANNAKI, K., GLICK, M., AND MUMMERT, L. Your data center is a router: The case for reconfigurable optical circuit switched paths.