Equinox: Adaptive Network Reservation in the Cloud

Praveen Kumar*, Garvit Choudhary†, Dhruv Sharma*, Vijay Mann*
*IBM Research, India
{praveek7, dhsharm4, vijamann}@in.ibm.com
†IIT Roorkee, India
{garv7uec}@iitr.ac.in

***Abstract*—Most of today's public cloud services provide dedicated compute and memory resources but they do not provide any dedicated network resources. The shared network can be a major cause of the well known "noisy neighbor" problem, which is a growing concern in public cloud services like Amazon EC2. Network reservations, therefore, are of prime importance for the Cloud. However, a tenant's network demand would usually keep changing over time and thus, a static one-time reservation would either lead to poor performance or resource wastage (and higher cost). In this context, we present Equinox - a system that automatically reserves end-to-end bandwidth for a tenant based on the predicted demand and adapts this reservation with time. We leverage flow monitoring support in virtual switches to collect flow data that helps us predict demand at a future time. We use a combination of vswitch based rate-limiting and OpenFlow based flow rerouting to provision end-to-end bandwidth requirements. We have implemented Equinox in an OpenStack environment with OpenFlow based network control. Our experimental results, using traces based on Facebook's production data centers, show that Equinox can provide up to 47% reduction in bandwidth cost as compared to a static reservation scheme while providing the same efficiency in terms of flow completion times.**

I. Introduction

Sharing of resources such as CPU, memory and network are the foundation of any public cloud service. In such a scenario, security and performance become a prime concern for a cloud tenant. Given that today's data center networks [1] are highly oversubscribed, network guarantees gain prime importance when network is shared among multiple tenants. Recent works [2], [3], [1] highlight some of the key requirements for high performance in a shared network. According to [4], the top requirements include *minimum bandwidth guarantee* for a VM in the worst case traffic, *simplicity* for the tenant in terms of inputs, *high utilization* of the network resources and *scalability*.

Most of today's public cloud solutions (like Amazon EC2 [5] and Microsoft Azure [6]) provide dedicated compute and memory resources (EC2 offers dedicated instances), but they do not provide any dedicated network resources. At best, EC2 provides cluster networking, which ensures that all instances of a tenant are placed in a network that offers high-speed and low-latency. Noisy neighbor [7] is a growing concern in commercial cloud systems like EC2. Dedicated server resources ensure that there are no noisy neighbors sharing the same server hardware that can affect each other's performance. However, the network is still shared by different tenants' applications and in the absence of bandwidth reservations, a bandwidth hungry application belonging to a tenant may affect the performance of an application belonging to another tenant.

Given the shared nature of the data center network, it becomes important for a tenant to have some network guarantee to estimate the performance of the applications. Static reservations have been proposed before in literature [8], [2]. However, they have not been implemented in real cloud systems like EC2 for three main reasons:

1) End-to-end reservations are hard to enforce in a multi-tenant cloud - one needs to enforce the reservation at the host NIC level and then across the entire path that different flows originating from a tenant's VMs may take. Traditional networking equipment does not provide this type of fine grained flow-level reservations.
2) Making a static reservation may be non-trivial for tenants. It is rare that tenants have a precise idea about how much bandwidth their applications might need. If they oversubscribe, they will end up paying more, and if they under-subscribe their application will face performance problems.
3) Offering static reservation may be equally troublesome for cloud providers - how many bandwidth slabs should be offered to clients? Should tenants be charged at a flat rate for each slab or should it be "pay for what you use" (the general cloud philosophy)?

Furthermore, recent studies such as [9], [10], [11], [12] show that there is significant variation with time in bandwidth demands of applications. Therefore, a solution based on static network reservations may result in under-provisioning for peak periods leading to poor performance for the tenant while for non-peak periods, it may result in over-provisioning leading to wastage of resources [13]. There is a need for a network reservation mechanism that dynamically adjusts the bandwidth reservations based on the demand.

To overcome these problems, we present *Equinox* - a system for adaptive end-to-end network reservations for tenants in a cloud environment. *Equinox* implements adaptive end-to-end network reservations in an OpenStack [14] cloud provisioning system using network control provided by an OpenFlow controller. *Equinox* leverages end-host Open vSwitch [15] based flow monitoring to estimate network bandwidth demands for each VM and automatically provisions that bandwidth end-to-end. It implements reservations at the host level through vswitch based rate limits enforced through OpenStack. To ensure that flows get the required network bandwidth, *Equinox* routes flows through network paths with sufficient network bandwidth. We also describe practical ways in which *Equinox*

978-1-4799-3635-9/14/$31.00 ©2014 IEEE

can be implemented by a cloud provider. The cost model of *Equinox* makes use of a few bandwidth slabs (for example, Platinum: 10 Gbps, Gold: 1 Gbps, Silver: 100 Mbps, Bronze: 10 Mbps) only to set the maximum cap on how much bandwidth a tenant can use (similar to a max-rate setting). Each slab involves some upfront cost - however, this is meant to be small (at least for the Silver and Bronze categories) and most of the time a tenant would be billed as per its usage of the network. This provides a greater incentive for tenants to use this service than to use a service based on static reservations. In this cost model, a tenant need not really know his exact bandwidth requirements, and may gradually decide on the best category while paying for only the bandwidth actually used. At each point of time, the tenant applications have some bandwidth reserved based on historical usage, which ensures that bandwidth hungry applications of other tenants can not degrade their performance (noisy neighbor problem). We evaluate *Equinox* on a real hardware testbed and through simulations using traces based on Facebook production data centers. Our evaluation focuses on performance, cost to the tenant and benefit for the cloud provider ([16], [17], [18]). Our experimental results, show that *Equinox* can provide up to 47% reduction in bandwidth cost as compared to a static reservation scheme while providing the same efficiency in terms of flow completion times.

The rest of the paper is organized as follows. In Section II, we discuss some related work in the field and the background for this work. Section III explains the detailed design of the *Equinox* system. In Section IV, we evaluate the system with different bandwidth reservation schemes. Section V summarizes the outcome of this work.

II. Background and Related Work

Driven by the need for network performance isolation and guarantees, there has been a lot of work on the cloud network sharing problem recently.

Static and dynamic network reservations:

While [8] and [2] have proposed solutions based on static reservations, works like [19] and [20] have advocated and proposed time-varying bandwidth reservation mechanisms based on studies on the nature of intra-cloud traffic. [21], [22] and [4] try to ensure minimum bandwidth guarantees. Recent works such as SecondNet [8] and Oktopus [2] have analyzed the variation of network demands for workloads such as MapReduce [23]. They have proposed new abstractions to provide a network QoS guarantee. Both these works have only looked at assigning static network reservations for the tenant which is oblivious to the changes in network demand over time. In our recent work on dynamic fair sharing of network [20], we proposed an algorithm to ensure that the network resources are shared fairly among the tenants. *Equinox* builds on that earlier work and employs the same algorithm. Some recent works [19] have proposed a time-varying abstraction to model bandwidth provisioning. These time-varying abstractions are based on profiling the jobs that are run in the data center. Further, as discussed in [24], most of these works have focused on MapReduce type of applications in which the data traffic follows simple patterns and can be profiled. However, the applications deployed in a public data center can have very complex communication patterns which might not be easy to profile [25]. In such a scenario, it becomes important to have some mechanism to predict the demands based on real time traffic. In terms of fairness, various schemes have tried to handle different types of fairness criteria. While [3] tries to provide per-source fairness, [22] and [20] ensure fairness across tenants. [4] can ensure guarantees only at per-VM level. [20] also ensures minimum bandwidth guarantees based on input from the tenants.

Network demand prediction:

On the demand prediction aspect, there have been works such as [26], [27] and [28] that mainly focus on measuring flow statistics and estimating the bandwidth requirements. [26] proposes a flow level bandwidth provisioning at the switch level and provide an OpenFlow based implementation. However, it doesn't consider the fairness at the tenant level. [28] propose a Probability Hop Forecast Algorithm which is a variation of ARIMA forecasting to model and predict dynamic bandwidth demands. [27] also proposes using the flow measurements and estimate the bandwidth requirement at a future time.

However, none of the existing works have focused on an end-to-end system for network reservations in which bandwidth demand estimated from flow level statistics is actually provisioned for the VMs. *Equinox* uses the abstractions proposed in [2] and [20] to ensure fairness among tenants. It is deployed on an OpenStack Grizzly testbed with Opendaylight [29] as the OpenFlow network controller.

III. Design and Implementation

Equinox consists of two major components :

1) Flow Monitoring Service
2) Network Reservation Service

Fig.1 shows the design of the Flow Monitoring Service while Fig. 2 shows the Network Reservation Service. The components are described in detail in the following sections.

A. Flow Monitoring Service

The Flow Monitoring Service builds on our earlier work [30] and handles functions such as configuring monitoring at the end hosts, starting/stopping the flow collector, exposing various APIs, demand prediction etc. *Equinox* needs to monitor the traffic in the network to estimate the demands of the VMs. Since it is based on the egress demand rates of the VMs, it is sufficient to monitor traffic at only the compute node level. Our prototype deployment uses OpenStack (Grizzly) with KVM and Open vSwitch at the compute nodes. The Open vSwitch at the compute nodes is configured to monitor and export traffic statistics to a flow collector. NetFlow [31] and sFlow [32] are the two most commonly used techniques for flow monitoring. Open vSwitch and the *Equinox* flow collector support monitoring using both these methods. We extend OpenStack Quantum[1] to configure flow based monitoring at the Open vSwitch on

[1]recently renamed to Neutron. http://wiki.openstack.org/wiki/Neutron

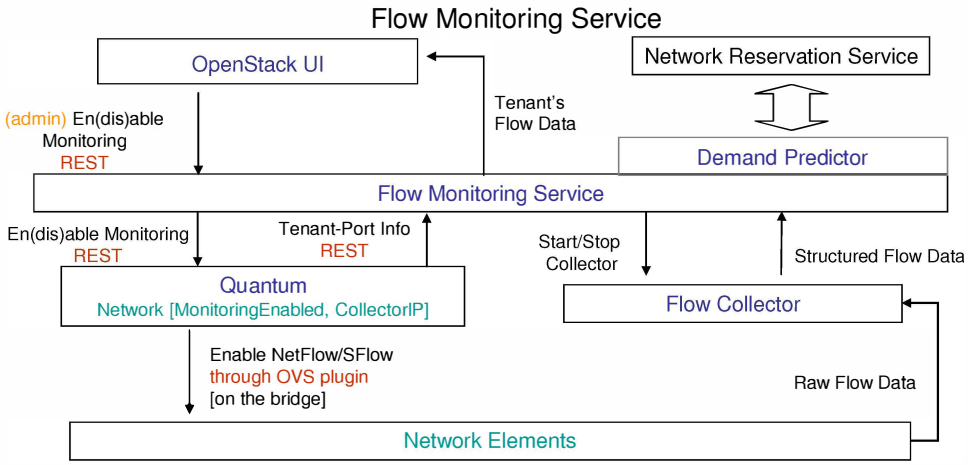

Fig. 1. *Equinox* Architecture: Flow Monitoring Service - Traffic Monitoring and Demand Prediction

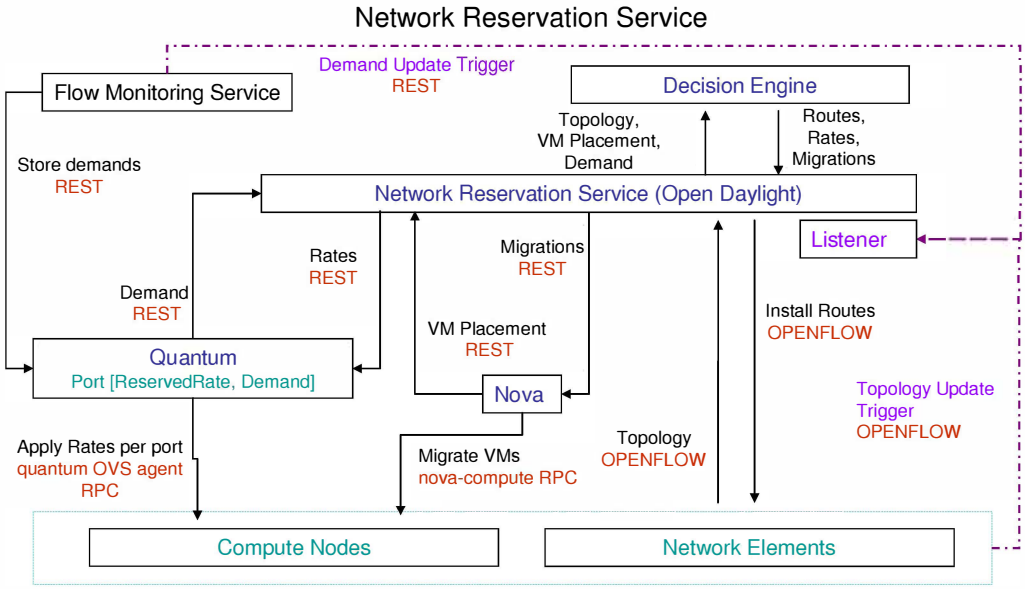

Fig. 2. *Equinox* Architecture: Network Reservation Service - Bandwidth Reservation, Rate-Limiting, Re-Routing and VM Migrations

all end hosts. Based on this, the Flow Monitoring Service also exposes an API through which the admin can configure the monitoring on the compute nodes. Once the compute nodes are configured to export the flow statistics and the service has started the flow collector at the specified location, the collector starts getting the monitoring data. The collector analyzes the raw data received from the Open vSwitch instances at the compute nodes and stores the monitored flow data in an appropriate format so that flow statistics can be calculated efficiently [30]. The Flow Monitoring Service can access this data and exposes APIs for various types of queries such as top-k flows for a tenant, average demand of a VM over the last time interval etc. The monitoring service also communicates with OpenStack for various tenant specific information such as identifying the VMs belonging to a tenant, setting the demand of a VM etc. OpenStack can query these APIs through REST and uses it to show information such as network usage of the VMs to a tenant.

In addition to monitoring flows and exposing APIs to query flow statistics, this service also implements demand prediction and triggers the Network Reservation Service with updated information. For each VM, the demands over the last n specified time-intervals can be queried and used to predict the demand for the next interval. Many numerical methods and techniques can be used to estimate the demand. The most commonly used ones include moving average, exponential smoothing, ARMA, ARIMA and GARCH models [28] [33]. The service predicts demands for each of the VMs based on the monitored data and updates the port [2] in the Quantum database with the estimated demand using the extended Quantum Update Port API. The service also periodically notifies the Network Reservation Service to update the network reservations.

[2]a *port* in Quantum refers to a VM's network interface

B. Network Reservation Service

The Network Reservation Service is responsible for computing and enforcing reservation through routing and end-host rate limits in the network. We have implemented this service as an Opendaylight [29] controller module. The service exposes a REST API to listen for triggers it will need to update the reservations. Whenever the service gets such a trigger, it will fetch the required information from various components and call the *Decision Engine* with the information. The Decision Engine is the core of the Network Reservation Service and is responsible for calculating the reservation. It needs the demands of all the VMs, the VM placement and the topology information to compute the reservation. The VM demands are fetched from Quantum, the VM placement information is gathered from Nova and the topology information of the compute nodes and switches is available from the Opendaylight topology manager. To map VM placement information from Nova to the network topology obtained from Opendaylight OpenFlow controller, the service processes packet-ins from the VMs. Based on the in-port of the packet-in, it identifies which compute nodes in the OpenStack world corresponds to which host in the OpenFlow topology. Once this mapping is done, the service has the entire network topology including the VMs.

The Decision Engine is based on the scheme proposed in our earlier work in [20]. For routing, it tries to provide the *Virtual Cluster (VC)* [2] abstraction to the tenant. In a VC abstraction, a tenant sees the logical abstraction that all the tenant's VMs are connected directly to a *Logical Switch* with dedicated links of given capacity. The logical switch is mapped to a switch in the physical topology and the required bandwidth is reserved on the path from each VM to the mapped switch.

The Decision Engine uses the information provided to calculate the new reservation, which is a set of end-host rate limits, routes and VM migrations. A reservation request is said to be satisfied if for all tenants: all the VMs of a tenant can be connected to the Logical Switch for that tenant and a bandwidth equal to the demand of each VM can be reserved on a path from the VM to the Logical Switch.

- The Decision Engine first tries to find if the request can be satisfied using the existing routing rules and placement. If it can be, then it just updates the rate-limits equal to the demands.
- Else, it tries to migrate the Logical Switches so that the demands can be satisfied. If the Logical Switch migration succeeds, in addition to the new rate-limits, the output will have the new set of OpenFlow based routing rules to update the routes for the VMs which were connected to the migrated Logical Switches.
- Else, in addition to Logical Switch migration, the Decision Engine also checks if migrating some of the VMs can satisfy the request. And if so, then it also suggests the VMs to be migrated and their destination hosts. Else, the request cannot be satisfied.

More details about Logical Switch migration and Virtual Cluster abstraction can be found in [20]. The Network Reservation Service, then, processes the output of the Decision Engine and enforces it on the network. To apply the end-host rate limits, it uses the Quantum REST API as mentioned in III-C. In order to handle Logical Switch migrations, re-routing needs to be performed as described in III-D. The VM migrations are handled by Nova as mentioned in III-E

C. End-Host Rate Control

Once the end-host rate limits are computed by the Decision Engine, the system needs to enforce these limits for the VMs on the compute nodes. The Opendaylight module uses Quantum REST APIs to notify the OpenStack compute nodes to update the rate-limits for the VMs. In our OpenStack setup, we use Open vSwitch based networking with Quantum. The current version of Quantum Open vSwitch plugin does not support specifying and setting QoS values for the Ports. In order to support QoS with Quantum, we have extended the Port entity in Quantum to store rate limits and demands. We have also extended the Quantum APIs to handle the QoS values (demand and rate limit). Whenever the update API is called with rate limit or demand, the database entry for the port gets updated. In addition, whenever there is an update on the rate limit of a port, the rate limit is enforced on that port. For the end-host rate-control [34], we leverage the QoS policing [35] which is supported by Open vSwitch. Open vSwitch uses the Linux traffic-control capability for rate-limiting. The *ovs-vsctl* utility allows one to specify the ingress policing rate and burst size as follows:

```
ovs-vsctl set Interface vnet1
     ingress_policing_rate=10000
     ingress_policing_burst=1000
```

The *ingress_policing_rate* is the maximum rate in kbps that the VM connected to the interface *vnet1* can send while *ingress_policing_burst* specifies the burst size which is the maximum data in kb that the VM can send in addition to the *ingress_policing_rate*. The Quantum Open vSwitch agent running at the end host, on getting an update request on the port rate-limit, applies the rate-limit on the interface.

D. Re-routing

In a VC abstraction, since every VM is assumed to be connected to a Logical Switch and sufficient bandwidth has been reserved along that path, appropriate routing rules need to be installed on switches so that a VM's traffic uses links on which bandwidth has been reserved for that VM. The Network Reservation Service installs bi-directional rules on all the switches on the path between the VM and the corresponding Logical Switch. The priority of the rule for a flow from a Logical Switch to a VM is set to be higher than the priority of the rule from a VM to the Logical Switch. This is done so that the traffic from one VM to another need not go all the way to the Logical Switch in case the reserved routes for the two VMs intersect at a switch other than the Logical Switch.

Whenever a Logical Switch migration happens, the routes need to be changed for all the VMs belonging to that tenant. However, this doesn't mean removing all the existing flow rules and installing new rules. It is possible that some of the flow rules required for the new Logical Switch location are the same as those used before the migration happened. In such a case, the Network Reservation Service uninstalls only the unneeded flow rules and installs only the new flow rules.

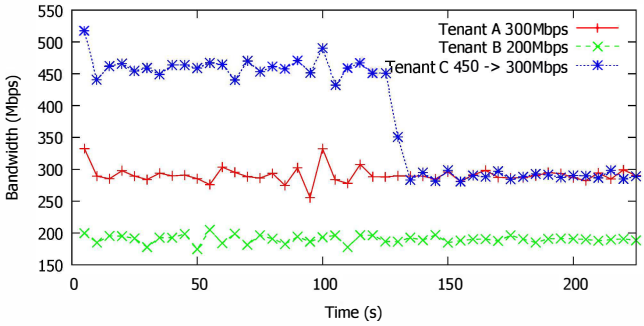

Fig. 3. Rate limiting: A rate limit of 300Mbps is applied to the Tenant C VM at $t = 130s$

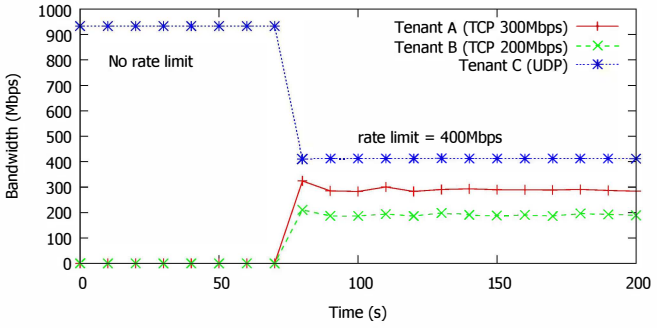

Fig. 4. Rate limiting in case of a bandwidth-hogging UDP traffic : A rate limit of 400Mbps is applied to the Tenant C VM at $t = 70s$

E. VM Migration

The Nova REST API for VM *live-migration* is used by the Network Reservation Service to migrate the VMs to a suitable compute node. On receiving a packet-in by a VM from the destination host, the service deduces a VM migration is complete and updates its topology information. To prevent the overhead of too many migrations, the number of allowed concurrent migrations can be configured on the Decision Engine.

F. Topology Updates

The Network Reservation Service registers with the topology service of Opendaylight and is notified whenever there is a change in the topology of the network. On receiving such notifications, the Network Reservation Service calls the Decision Engine with the updated information and applies the result.

IV. Evaluation

A. Experiment Model

Hardware Testbed: We run *Equinox* on an OpenStack setup as described in IV-B. The results in IV-D are based on this setup. We also ran Hadoop on a larger hardware testbed and collected flow level data using NetFlow. Remaining results in this section are based on simulations performed on this flow level data. Both these testbeds are described in more details in IV-B.

Simulations: In a real scenario, a cloud provider creates a set of categories or classes (such as Platinum, Gold, etc.) based on the QoS guarantees and varying rates. A tenant can opt for one of the categories which will decide the QoS that will be provided to the VMs. Each category here implies a specific value of the maximum bandwidth that a VM can use. Since we are evaluating only the network performance under various types of reservations, we do not take into account the other parameters such as CPU and memory that may be assigned for a category. Further, since we use the actual flow data that we get from the Hadoop testbed as described in IV-B and IV-C, we adjust the QoS guarantees for the different categories accordingly.

To ensure that the comparison between the reservation schemes is fair, we record all the flows every second in case of uncapped (or unreserved) bandwidths and use the same data to simulate the behavior of all the reservation schemes.

In case of the dynamic reservation schemes, there are many techniques that can be used to predict the demand. Since the objective of this paper is not to draw a comparison between various demand prediction methods, we show the results using just the exponential smoothing with varying α in case of dynamic reservations. The average bandwidth demand for the VMs is $\approx 10Mbps$. We consider two categories - I and II. For category I, the bandwidth is capped at $10Mbps$ and for category II, the bandwidth is capped at twice the average demand, i.e. $20Mbps$. The various reservation schemes that we compare are:

- ST_{Avg} : Static reservation - The average demand for each VM is reserved
- ST_{2Avg} : Static reservation - Twice the average demand for each VM is reserved
- $DY_{\alpha,cat}$: Dynamic reservation - Bandwidth reserved is based on exponential smoothing $\alpha \in \{0.25, 0.5, 0.75\}$ and $cat \in \{1, 2\}$ depending on the category (I or II).

$$rate_t = \alpha \times demand_{t-1} + (1-\alpha) \times rate_{t-1}, t > 0 \quad (1)$$

The results shown in IV-E, IV-F, IV-G and IV-H are based on these simulations.

B. Hardware Testbed Details

We use two different setups on our hardware to perform our experiments. Our testbed consists of 6 IBM x3650 M2 servers. Each server has a 12-core Intel Xenon CPU E5675 @ 3.07GHz processor and 128 GB of RAM. All the servers run 64-bit Ubuntu 12.04.1 server with Linux 3.2.0-29-generic kernel. These servers are connected through 1Gbps NICs to a network of three inter-connected IBM BNT RackSwitch G8264 switches with 10Gbps link capacities.

Hadoop Setup: We evaluate the performance of static and dynamic network reservations on traffic data collected from a testbed running various MapReduce jobs. For this, we run Hadoop 1.1.1 [36] on a cluster with 60-nodes and measure the traffic data. The 6 servers host 10 VMs each and each of the VMs is assigned 2GB of RAM and one CPU core. Every physical host has an Open vSwitch (OVS) running and all the nodes placed on the host are connected to it. The traffic is generated and measured as explained in Sec.IV-C.

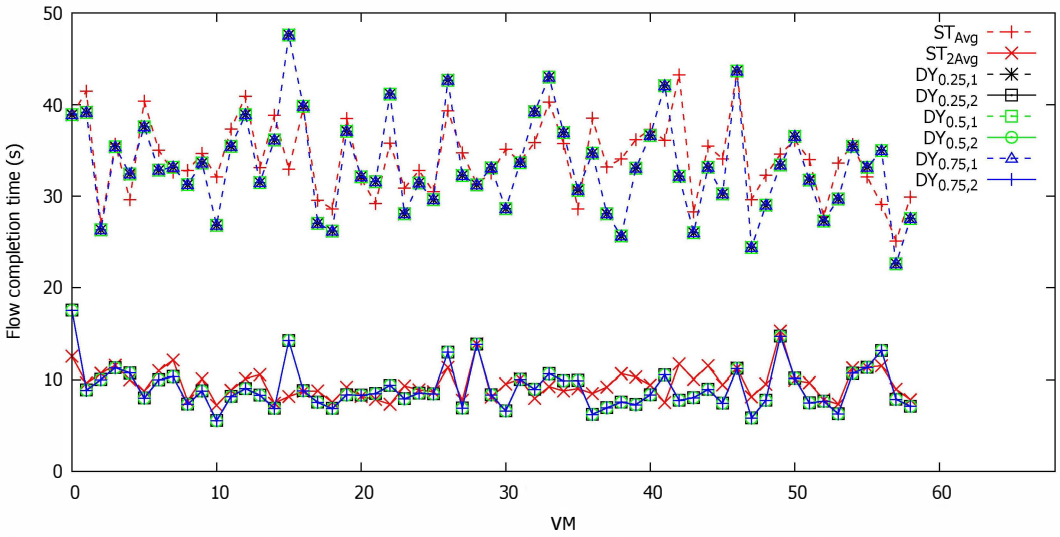

Fig. 5. Average flow completion times for all the 60 VMs over 24 hours when using different reservation schemes

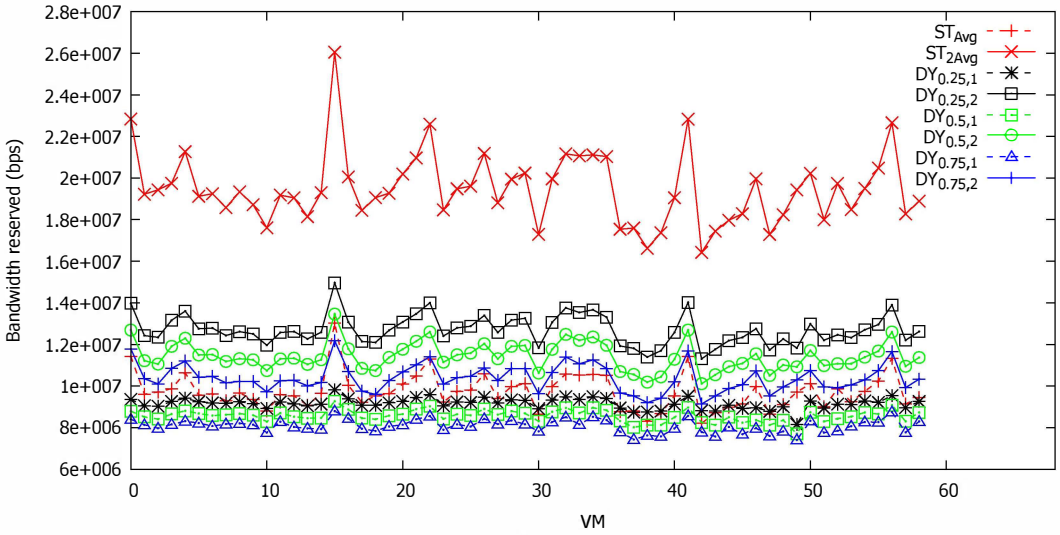

Fig. 6. Average bandwidth reserved for different VMs over a duration of 24 hours when using different reservation schemes

OpenStack Setup: To validate the working of *Equinox*, we deploy OpenStack Grizzly setup on our testbed. The setup consists of one control node and four compute nodes. The Flow Monitoring Service runs on a server with similar hardware while the Network Reservation Service which is built on the Opendaylight controller runs on a separate server which has a 2-core Intel Xenon CPU @ 3.40GHz processor and 4 GB of RAM.

C. Traffic generation

We deploy Statistical Workload Injector (SWIM) for MapReduce [37], [38] on our cluster to generate a real data center representative workload. SWIM provides a set of MapReduce workloads from real production systems and a tool to generate representative test workloads by sampling historical MapReduce cluster traces. These workloads are reproduced from the original trace that maintains the distribution of input, shuffle, and output data sizes, and the mix of job submission rates, job sequences and job types [38]. Using SWIM, we pre-populate HDFS (Hadoop Distributed File System) on our cluster using synthetic data, scaled to the 60-node cluster, and replay the workload using synthetic MapReduce jobs. Once the replay is started, the jobs keep getting submitted automatically to Hadoop at varying time intervals as calculated by SWIM and are handled by the default Hadoop scheduler. For our evaluation, we use the traces available in the SWIM workload repository which are based on Facebook's production data center. To measure the bandwidth utilization during the process, we enable NetFlow on the Open vSwitch bridges with an active timeout of 1 second and monitor the traffic using our NetFlow collector.

D. Network Reservation Validation on Hardware Testbed

Fig. 3 shows the bandwidth attained by three VMs belonging to different tenants when using the same 1 Gbps physical link. The VMs belonging to tenants A, B and C have demands of

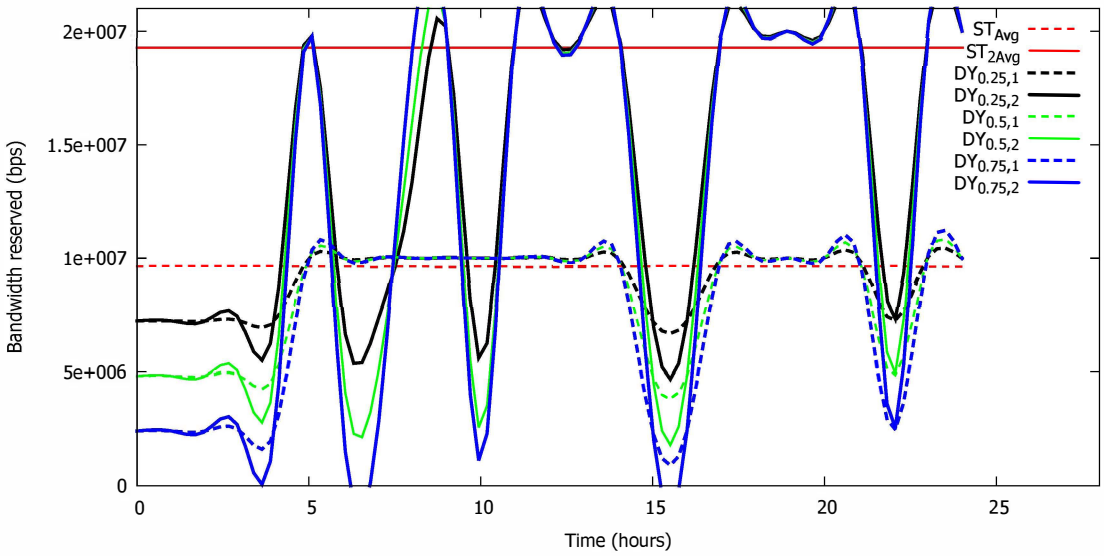

Fig. 7. Bandwidth reserved *(csplines smoothed)* for one VM with time during 24 hours when using different reservation schemes

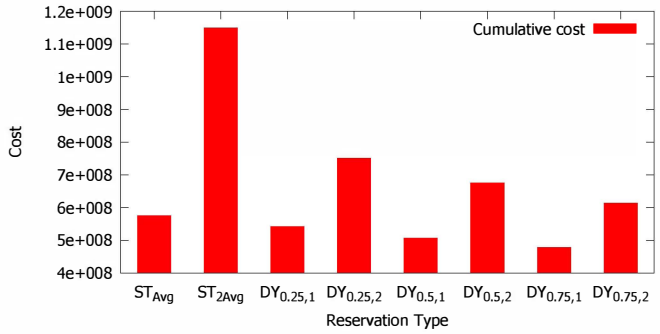

Fig. 8. Cumulative bandwidth cost for 24 hours when using different reservation schemes

300, 200 and 450 Mbps respectively and initially, they get the required throughput. At $t = 130s$, a network reservation of 300 Mbps is applied on the tenant C VM and its measured throughput reduces to the assigned bandwidth.

Fig. 4 shows how network reservation helps in isolating the performance for each tenant. The VMs belonging to A and B are using TCP and have a bandwidth demand of 300 and 200 Mbps respectively. The VM belonging to tenant C is using UDP (which is oblivious to congestion) and uses up the entire 1 Gbps bandwidth leading to starvation for A and B. At $t = 70s$, a network reservation of 400 Mbps is applied to the tenant C VM and we see that now the TCP flows get their required bandwidth.

E. Simulation: Flow Completion Time

Flow completion time is an important metric to judge the network performance. Fig. 5 shows the average flow completion times for different VMs in case of all the reservation schemes.

The average time taken for flow completion is almost the same in case of static and dynamic reservations with the same bandwidth caps. However, doubling the bandwidth cap decreased the average completion times by about 71%. ST_{2Avg}, $DY_{0.25,2}$, $DY_{0.5,2}$ and $DY_{0.75,2}$ result in best flow completion times.

F. Simulation: Actual Average Bandwidth Reserved

Fig. 6 shows the average bandwidth reserved for different VMs over a duration of 24 hours in case of all the reservation schemes.

The average bandwidth reserved in case of dynamic reservations belonging to category II is significantly less than that for ST_{2Avg}. Thus, the dynamic reservation schemes seem to utilize the reservation judiciously and save on unutilized reservations while ensuring the same flow completion times as that in case of over-provisioned static reservations.

G. Simulation: Bandwidth Reservation vs Time

Fig. 7 shows the variation of bandwidth reserved for one VM over a duration of 24 hours in case of all the reservation schemes.

While the static reservation schemes continue to reserve the same bandwidth throughout, the dynamic ones keep modulating the reserved bandwidth with time based on the demand. The dynamic reservation schemes with category II reserve a higher bandwidth at peak periods and thus are able to handle the peak demands quicker than the dynamic reservation schemes with category I. As a result, the duration for which the reserved bandwidth is equal to the cap value is longer for the dynamic reservation schemes with category I.

H. Simulation: Cumulative Bandwidth Cost

Fig. 8 shows the cumulative cost incurred for the 60 VMs over a 24 hour time period. The cost calculated here is basically the summation of the reserved bandwidth over time. This is based on the pay-as-you-use model and assuming that the cost varies linearly with the reserved bandwidth.

The cost of bandwidth when using ST_{2Avg} is the highest. Compared to ST_{2Avg}, $DY_{0.25,2}$, $DY_{0.5,2}$ and $DY_{0.75,2}$ lead

to a 34.8%, 40.8% and 46.9% reduction, respectively, in the cost while providing the same efficiency in terms of flow completion times.

V. Concluding Remarks

In this paper, we presented *Equinox*, a system for adaptive end-to-end network reservations for tenants in a cloud environment. *Equinox* builds on the abstractions from our earlier work [20], and ensures *performance guarantee* to the tenant and *high utilization* for the data center. Since *Equinox* automatically deduces and provisions bandwidth based on the monitored data, it ensures *ease of use and simplicity* for the tenant. Equinox has been implemented on top of OpenStack and Opendaylight. Our experimental results show that *Equinox* can provide up to 47% reduction in bandwidth cost as compared to a static reservation scheme while providing the same efficiency in terms of flow completion times. Our current and future efforts are directed mainly at three things. First, we are currently in the process of deploying a limited version of *Equinox* in a real cloud. Second, we intend to experiment with various bandwidth prediction models and evaluate what works the best. Third, through our real cloud deployment and other experiments, we are trying to understand the various overheads associated with a system like *Equinox*.

References

[1] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *ACM SIGCOMM*, August 2009.

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

[3] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *USENIX NSDI*, 2011.

[4] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and W. Azure, "EyeQ: Practical Network Performance Isolation at the Edge," in *USENIX NSDI*, 2013, pp. 297–312.

[5] "Amazon.com. Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: http://aws.amazon.com/ec2/

[6] "Microsoft Windows Azure." [Online]. Available: http://www.windowsazure.com/en-us/

[7] "VMWare - Getting rid of noisy neighbors: Enterprise class cloud performance and predictability." [Online]. Available: http://blogs.vmware.com/rethinkit/2010/09/getting-rid-of-noisy-cloud-neighbors.html

[8] 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 *ACM CoNEXT*, 2010.

[9] N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "A Flexible Model for Resource Management in Virtual Private Networks," in *ACM SIGCOMM*, 1999.

[10] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *IMC*. ACM, 2010, pp. 1–14.

[11] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[12] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *INFOCOM*. IEEE Press, 2010, pp. 1163–1171.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[14] "Openstack." [Online]. Available: http://www.openstack.org/

[15] "Open vSwitch: An Open Virtual Switch." [Online]. Available: http://openvswitch.org/

[16] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "The Price is Right: Towards Location-independent Costs in Datacenters," in *HotNets*. ACM, 2011, pp. 23:1–23:6.

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

[18] D. Niu, C. Feng, and B. Li, "Pricing cloud bandwidth reservations under demand uncertainty," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 151–162, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2318857.2254776

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

[20] M. Mishra, P. Dutta, P. Kumar, and V. Mann, "Managing Network Reservation for Tenants in Oversubscribed Clouds," in *MASCOTS*. IEEE, 2013.

[21] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks,," in *USENIX Workshop on I/O Virtualization*, 2011.

[22] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and Stochastic Netshare: Predictable Bandwidth Allocation for Data Centers," *SIGCOMM Computer Communication Review*, vol. 42, no. 3, 2012.

[23] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *USENIX OSDI*, 2004.

[24] J. Lee, M. Lee, L. Popa, Y. Turner, S. Banerjee, P. Sharma, and B. Stephenson, "CloudMirror: Application-Aware Bandwidth Reservations in the Cloud," 2013.

[25] 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 *ACM SIGCOMM*, 2010, pp. 243–254.

[26] R. De O. Schmidt and A. Pras, "Estimating Bandwidth Requirements Using Flow-level Measurements," in *AIMS*. Springer-Verlag, 2011, pp. 169–172.

[27] B. Krithikaivasan, K. Deka, and D. Medhi, "Adaptive bandwidth provisioning envelope based on discrete temporal network measurements," in *INFOCOM*, vol. 3. IEEE, 2004, pp. 1786–1796.

[28] H. Jin, D. Pan, J. Liu, and N. Pissinou, "OpenFlow-Based Flow-Level Bandwidth Provisioning for CICQ Switches," in *INFOCOM*. IEEE, 2011, pp. 476–480.

[29] "Opendaylight." [Online]. Available: http://opendaylight.org/

[30] V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," in *COMSNETS*, 2012.

[31] "NetFlow." [Online]. Available: www.cisco.com/go/netflow

[32] "sFlow." [Online]. Available: http://www.sflow.org/sFlowOverview.pdf

[33] D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-Assured Cloud Bandwidth Auto-Scaling for Video-on-Demand Applications," in *INFOCOM*. IEEE, 2012, pp. 460–468.

[34] S. Radhakrishnan, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "NicPic: Scalable and Accurate End-Host Rate Limiting," in *HotCloud*, 2013.

[35] "QoS Rate Limiting with Open vSwitch." [Online]. Available: http://openvswitch.org/support/config-cookbooks/qos-rate-limiting/

[36] "Hadoop 1.1.1 Documentation." [Online]. Available: http://hadoop.apache.org/docs/r1.1.1/

[37] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross Industry Study of MapReduce Workloads," in *VLDB*, 2012.

[38] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance Using Workload Suites," in *MASCOTS*. IEEE, 2011.