omniCluster - Virtual Network Allocation in Data Centers using Software-Defined Networking

Daniel Gomes Fonseca Caixinha
daniel.caixinha@tecnico.ulisboa.pt

Instituto Superior Técnico, INESC-ID
Av. Prof. Doutor Aníbal Cavaco Silva - 2744-016 Porto Salvo, Portugal

Abstract. The creation of data centers allowed global access to huge computational resources, previously only available to large companies or governments. By renting the desired computational power, small companies (or even an individual person) do not need to worry about large investments. In current data center environments, a client can ask for a computational instance of various sizes, and the service provider assures levels of guaranteed performance (through an SLA) for that computational instance.
However, these guarantees are not extended to the networking layer. Cloud providers do not offer network performance guarantees to their tenants. Since communication is carried over a network shared by all tenants, the performance that a tenant's application can achieve is unpredictable and depedent on several factors - some outside the tenant's control.
In this work we propose *omniCluster*, which solves these problems by using the abstraction of virtual networks. Virtual networks are isolated from each other, providing performance guarantees. We designed a scalable OpenFlow controller, that is able to allocate virtual networks (with bandwidth guarantees) in a work-conservative system, and achieves both high consolidation on the allocation of virtual networks and high resource utilization of the Data Center's resources.
Our assessments shows that the above mentioned properties are achieved, being carried out in two common data center network topologies: Tree and Fat-tree.

Keywords: Data Center, Virtual Networks, Bandwidth Guarantees, Work-Conservation, Software-Defined Networking.

1 Introduction

The creation of data centers allowed global access to huge computational resources, previously only available to large companies or governments. By renting the desired computational power, small companies (or even an individual person) do not need to worry about large investments. In current data center environments, a client can ask for a computational instance of various sizes, and the service provider assures levels of guaranteed performance (through an SLA) for that computational instance. This guarantee is possible due to the huge evolution of virtualization technologies. Nowadays, hypervisors can control the behaviour of the virtual machines (VMs) it possesses, ensuring that a VM cannot use more CPU than what it was requested (except when the hypervisor allows). In this way, clients (or tenants) are not harmed by the misbehaviour of others.

This simplicity of computational resources on demand has generated a lot of interest around the world. However, there are still a lot of things to improve in this area. One of them is the lack of network accounting in this renting of resources. Cloud providers do not offer network performance guarantees to their tenants. In fact, a tenant's compute instances (i.e. VMs) communicate over the network, that is shared by all tenants.

Thus, the network performance that a certain VM can get is dependent on several factors, most of them outside the tenant's control, such as the network load on a given moment or the placement of that VM in the network. Moreover, this is aggravated by the oversubscribed nature of a Data Center network.

The lack of guarantees in a shared communication medium leads to unpredictable application performance (as well as tenant cost). This is well documented in [7]. This work shows the impact of machine virtualization on network performance, and conclude that virtualized machines often present abnormally large packet delay variations, which can be a hundred times larger than the propagation delay between the two hosts they used for measuring. Another interesting finding by this work is that TCP and UDP throughput can fluctuate rapidly (in the order of tens of milliseconds) between 1 Gb/s and zero, which shows that applications will have a very unpredictable performance. This can be very important, since many applications running in the cloud are data intensive, such as video processing, scientific computing or distributed data analysis. This may severely degrade the performance achieved by an application. For instance, with intermittent network performance, MapReduce[2] applications will experience harsh issues when the data to be shuffled amongst mappers and reducers is quite large.

Solving this problem is hard for many reasons. Nonetheless, the major one is the difficulty of obtaining the current state of the Data Center network. In turn, this is hard to do because the network state is distributed: both physically among network devices that are running distributed protocols; and logically, since each network device has its own convoluted way to configure and monitor. Software-Defined Networking (SDN) [6] is an emerging networking concept, that decouples the control and the data planes.

This abstraction thereby decouples the forwarding hardware from the control software, which means that the control mechanism can be extracted from the network elements and logically centralized in the SDN controller. The controller creates an abstraction of the underlying network, and thereby provides an interface to the higher-layer applications. By introducing layers and using standardized interfaces, SDN brings a modular concept that enables the network to be developed independently on each one of the three layers. Most notably, it creates the opportunity for others than networking hardware vendors to innovate and develop control software and applications for the network.

In this project, we use SDN to have a global view of the state of the network. This way, we are able to give *virtual networks* to tenants. Such as a VM, a virtual network gives performance isolation, but at the network level. This allows tenants to express their requirements in terms of bandwidth, which is then enforced through virtual networks.

As expected, the objectives are focused on improving the state-of-the-art, providing properties that are lacking in current systems. Thus, we want to build a system that: is scalable to Data Center environments; achieves high consolidation (within the placement of virtual networks); achieves high resource utilization of the Data Center's physical resources (namely servers and network); and provides bandwidth guarantees in a work-conservative system.

2 Related Work

We now give a brief description of our key enabling technology - OpenFlow. Then, we will categorize the works that are most related to ours, describing the properties achieve by each category of system.

The OpenFlow architecture consists of three concepts: the network is made of OpenFlow-compliant switches, that form the data plane; the control plane is made by at least one OpenFlow controller; and there is a secure channel between each switch and the control plane.

Each switch is a "dumb" forwarding device, that simply uses its flow table to determine a packet's next hop. A flow table is composed by a series of flow entries. Each flow entry has the **header fields** on which it will try to match incoming packets (e.g. Ethernet destination address), **actions** to perform when an incoming packet matches this table entry (e.g. forward to a specified port or flood to all ports), and **counters** that hold statistical information about each flow (e.g. number of packets and bytes transmitted in this flow). Each OpenFlow switch uses Ternary Content Addressable Memory (TCAM) to allow fast lookup of wild-card matches, and thus fast packet forwarding. When an incoming packet does not match any entry on the flow table, it is sent to the controller using the secure channel, which will decide what to do with this packet.

We now present the description and classification of the works related to ours. At the highest level, the studied works can be divided in two categories. Thus, our taxonomy for network allocation techniques in data centers is: dynamic allocation based on network monitoring; and static allocation based on virtual network embedding. The former aims at maximizing resource utilization, fair bandwidth sharing of the physical network and sometimes give Quality of Service (QoS) guarantees for end hosts, while the latter provides support for virtual networks with QoS guarantees but with small (or even none) maximization of network usage.

The first type of works, classified as doing dynamic allocation based on network monitoring, rely on a constant monitoring of the network performance to keep an up-to-date view of the data center network. For that purpose, it is periodically gathered statistical information from the network elements (and/or end hosts), in order to infer the bandwidth usage for each VM or physical server. With this updated view of the network, this type of systems can use traffic engineering techniques to maximize resource utilization, leading to an economical gain for the provider, since it can for instance aggregate requests and shut down unused resources. The maximization of resource usage is possible since these dynamic approaches can react to the changing demands from different tenants, and manage them to get the most of the providers' resources. The solutions we have surveyed can be categorized as based on: centralized traffic matrix estimation or bandwidth regulation in each end host.

The second type is composed by works that make static allocations based on virtual network embedding. Whereas the algorithms from the previous paragraph aim at maximizing the resource usage and QoS guarantees for the tenants, in virtual network embedding the aim is to completely virtualize the network, providing performance isolation among tenants at the network level. So, virtual network embedding consists in the mapping of virtual networks (consisting of virtual nodes and links) onto the substrate network (consisting of physical nodes and links). Some works do not call this embedding, but are doing the same thing (mapping virtual networks in one or more physical networks).

As we can see, we have some works that are focused on maximizing resource utilization (which are favourable to the service provider), and other works that aim to provide bandwidth guarantees by making allocations of virtual networks (which is advantageous for the customer). None of the surveyed works try to achieve the two at the same time, giving the best of both worls, in a solution that satisfies everybody involved.

3 *omniCluster*

We now present and describe our solution - *omniCluster*. We begin by providing a generic use case for our system. In Figure 1 we can see the high level architecture of the solution, in this case using a simple tree topology with depth equal to 3 and fanout equal to 2. Although many topologies are being proposed by the network research community, we will focus on the traditional tree-like topologies (Tree and Fat-tree) since they are still the most used data center topologies, as described in [1]. The OpenFlow controller (in the top-right corner) is

the central component of the solution, as it is responsible for running the virtual network embedding algorithm in order to map the requests on the substrate network, and then program the switches to deploy the requested virtual networks. As previously stated, each switch acts merely as a packet forwarder, according to the rules dictated by the controller. The controller has a connection to every switch in the network, represented by the dashed red lines that form the control plane. They are dashed to represent the logical (and not physical) separation between the data and the control planes. The control plane does not necessarily need dedicated connections, it can use the same physical links of the data plane.

Our network embedding algorithm tries to allocate the requests on the smallest available subset of the substrate network (i.e. on the same physical server, then on the same rack, and so on). Thus, we aim to maximize the proximity of VMs belonging to the same tenant, which results in minimizing the number of hops between those VMs. This is advantageous for two reasons: first, with less hops the delay is normally reduced; and second, keeping the VMs close (e.g. in the same rack) relieves the bandwidth usage in the upper links of the tree, which in the data center is where the bandwidth is scarcer[1]. With this approach, we will be able to accept more virtual network requests, since the core links will not be so likely to become the bottleneck of the data center.

In Figure 1, we can see an example of the placement of three virtual networks according to this algorithm. The virtual network of tenant A represents the best case possible where all the VMs of the virtual network can be mapped on the same physical server. In this case, there is no usage of the network (which saves bandwidth for future requests), and the bottleneck of the virtual network is only the speed within the server. In the virtual network of tenant B the request could not be mapped to a single server, and so it uses another server belonging to the same rack to accommodate the entire request. The virtual network of tenant C shows a case where the virtual network could not be mapped in the same rack, and has to use a server on the adjacent rack. As we can see, the bandwidth of the links on the top of the tree is only used in the worst cases (i.e. when the request is large or the data center is operating near saturation).

Besides the network embedding algorithm, that ensures the network can provide the bandwidth guarantees requested by each tenant, we make use of the centralized information in the controller to provide two properties not seen in the surveyed network embedding systems: fair bandwidth sharing (i.e. work-conservation) among tenants (non-existent in network embedding systems) and incremental consolidation of virtual network requests. The first is achieved by instructing every switch used by a virtual network (which is determined by the embedding algorithm) to create a new queue for that virtual network. The queues in OpenFlow are used to provide QoS guarantees (in this case bandwidth). The second property is enforced in the network embedding algorithm itself, choosing the location of a virtual network according to a best-fit heuristic on the VM placement. To do this, the algorithm needs the current state of the network and the physical servers (which is kept by the OpenFlow controller). By managing all this information centrally, the controller could become itself the bottleneck of the network. However, the controller does not need to be a single machine. For instance, it could be a cluster of powerful machines.

3.1 Detailed Architecture

We will now describe in greater detail the architecture shown in the previous section. Figure 2 is the result of "zooming in" in each component of Figure 1. Thus, Figure 2 contains the software that will run in each component as well as the most important interactions between those components.

To simulate our data center network, we will use Mininet [4], both because it is open-source (which is very useful in research) and also because it is the *de facto* standard of OpenFlow emulators [5]. It is also important to refer that all the source code that will be

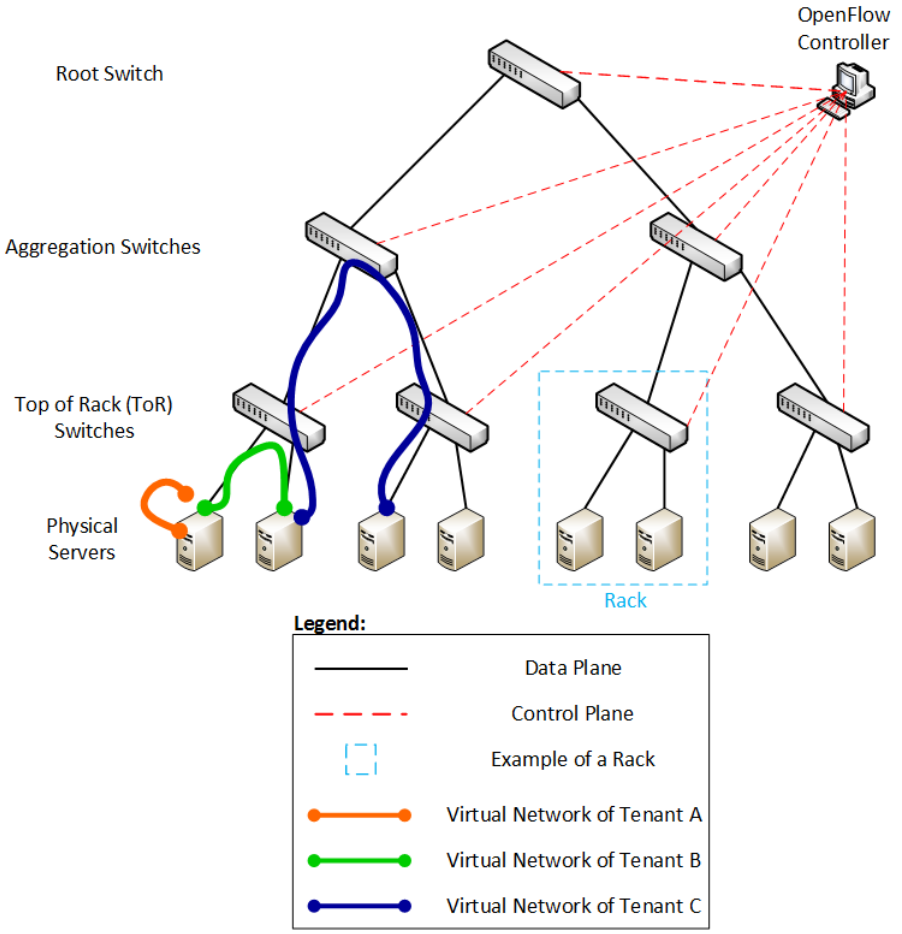

Fig. 1. High level architecture of the solution with a tree topology.

developed could be ported to a physical data center with few (or no) changes. The only exception is in the Linux Process, which in a real scenario would be running an hypervisor software to manage the VMs inside that host. In this work this is simplified to an operating system managing processes, where each process will simulate a VM. Another important detail is that we assume all the physical servers have equal CPU, so that in a virtual network request a tenant asks for a percentage of a CPU (instead of a CPU with a certain frequency, as in real data centers).

We now describe the typical flow of information when the system is in operation. First of all, the tenant expresses its demands in a virtual network request (which consists in a XML file). This consists in defining two things: the number of VMs required (and also the percentage of CPU of each one) as well as the bandwidth required between the VMs that will be connected (expressed in MBit/s). This request is fed into the virtual network embedding algorithm, that is running in the OpenFlow controller. Upon receiving the request, the embedding algorithm contacts the network information manager to get the current state of the network. Based on this state, the algorithm determines (if the request is accepted) where this virtual network will be allocated. It is important to note that as all the requests are processed by the controller, this updated view of the network involves zero control messages

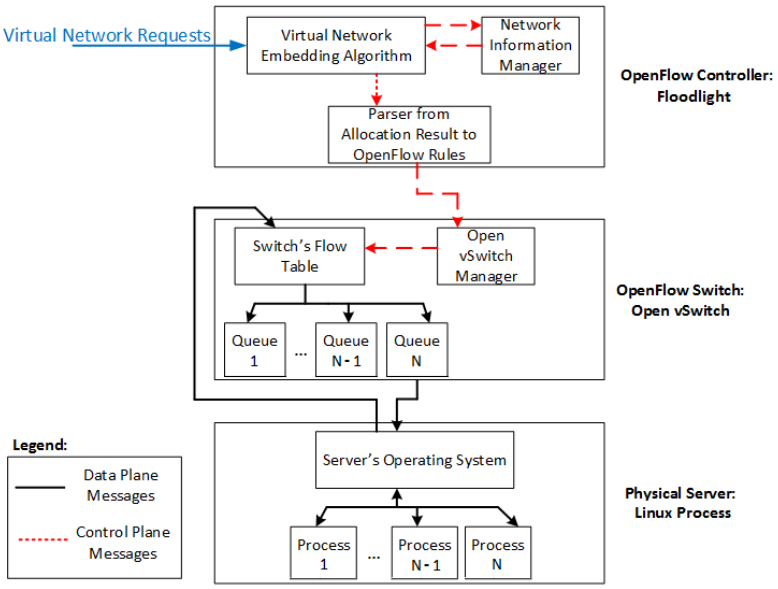

Fig. 2. Software architecture of the components present in the solution.

over the network (both to switches and hosts), since the controller just has to update this information when it processes a new virtual network request.

The controller then translates the resulting decision of the algorithm (i.e. the affected switches and hosts) to OpenFlow rule(s) to reprogram the switch(es). Upon receiving this message, the Open vSwitch manager takes two actions: creates a new queue for this virtual network, with the assured bandwidth present in the received message; and installs a new rule in the switch's flow table to forward packets from a certain VM to the newly created queue. This means that there will be one queue for each pair of linked VMs in a virtual network. In this way, a VM can use more bandwidth than its minimum when the link is not throttled. Moreover, the sharing of bandwidth between queues is made fairly according to the minimum bandwidth a queue has (a queue with a higher minimum bandwidth will use more spare bandwidth). Thus, the resource usage is maximized, since the tenants share unused bandwidth fairly, but at the same time get their minimum bandwidth guarantee when the network is saturated.

As already mentioned, the VMs will represented by and implemented as Linux processes. To simulate tenants' workloads, each process will be running a traffic generator. As depicted in Figure 2, upon the necessary configurations, each process (i.e. VM) can communicate with other processes on the same virtual network, using either the operating system (in case the processes are on the same server), or contacting its adjacent switch, which will use the flow table to check to which queue it should forward this solicitation (in case the processes are on different servers). In this work we will only focus on communication inter-server (as the intra-server communication would be a responsability of the hypervisor in a real deployment).

4 Evaluation

In order to assess the goal fulfillment of our solution, we have implemented what was described in the previous section on top of the Floodlight controller, developing the necessary modules to achieve the desired behavior.

Then, our experiments were run as follows: the controller processes virtual network requests, and we stop an experiment when the controller returns `False` to an allocation, meaning it can not allocate more virtual networks. Then, each experiment is run a thousand times, in order to get meaningful results. To generate our dataset, we have produced virtual network requests (XML files) where: a VM asks for a CPU that is generated randomly (using an uniform distribution) between 0.1 and 5 %; and the connections between VMs are also randomly generated (with an uniform distribution as well) between 0 and 10 Mbit/s. For each size of the virtual network requests (i.e. number of VMs in it), which we defined as going from 2 to 40.

Regarding the network topologies, we ran our experiments in Mininet using Tree and Fat-tree topologies. However, due to space constraints, here we only show the results achieved with a tree topology. We now show the evaluation made according to each goal that was defined in section 1.

4.1 Goal I - Scalable to Data Center Environments

In this goal we want to assess if our solution is scalable to environment (and size) particularly found in Data Centers. To this end, we have measured the time it takes to process each virtual network request. This is calculated using the method `currentTimeMillis` from Java API. In Figure 3, we can see the results of obtained with a Tree topology:

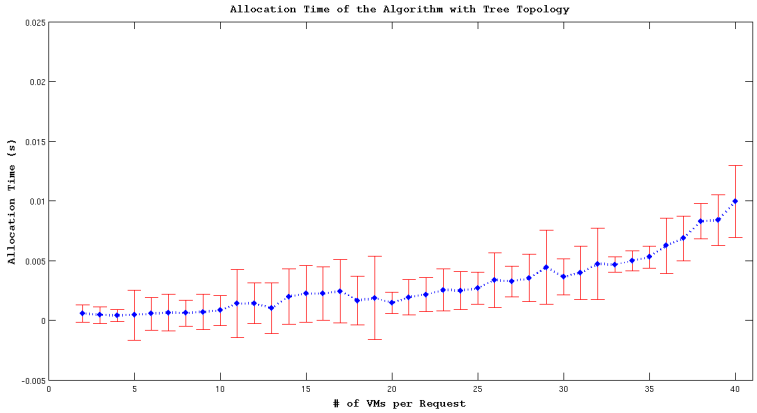

Fig. 3. Allocation time for a virtual network request using a Tree topology.

By analyzing Figure 3, we can see that up to about 25 VMs in a request, we have a processing time under 5 ms, which we think are great results. As a comparison, SecondNet[3] achieves 10 ms in requests with 10 VMs, which is twice the processing time in requests with less than half the VMs. We can see that our system takes about 10 ms to process requests with 40 VMs. Although the emulated network is not close to the size of a Data Center, we want to point out that a request with 40 VMs is almost one third of the number of physical servers in the network. Even in these conditions, our processing time did not grow abruptly, which we think is a good indicator of the scalability of our algorithm.

4.2 Goal II - High Consolidation

The evaluation around this goal aims to measure how consolidated virtual networks are, since our algorithm strives to take server locality into account, allocating the VMs of a

virtual network as close as possible. This is important since a low number of hops leads to a low latency in the communication between the VMs of a virtual network. This metric is calculated by counting the numbers of physical servers there are in a virtual network allocation. In Figure 4 the results of using a Tree topology are depicted.

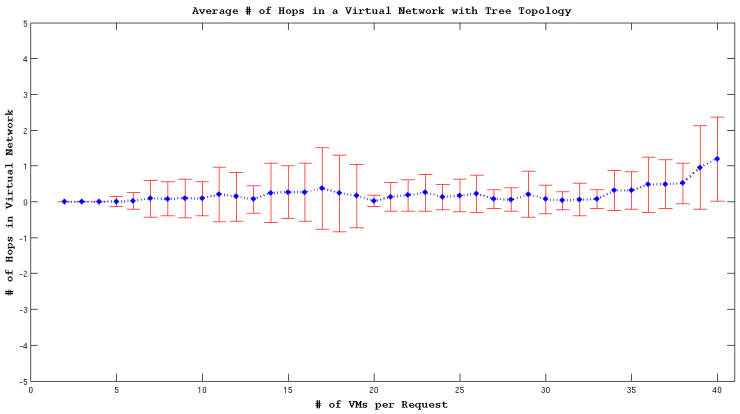

Fig. 4. Average number of hops in a virtual network using a Tree topology.

Analyzing Figure 4 we can see that we achieve high consolidation of the virtual networks, since the average is almost always near zero. It starts out equal to zero up to 5 VMs per request, then the average is almost the same but the variance increases, meaning most of the virtual networks do not have any hop, but some do. It keeps this behavior along the line, with the average increasing less than linearly. On 40 VMs per request we get an average number of hops close to 1. Again, we want to point out that 40 VMs in a request is big since our data center network has 125 servers, and even in that case the virtual networks only need, on average, one hop.

4.3 Goal III - High Resource Utilization

By evaluating this goal we want to measure the resource utilization within the Data Center: this means calculating the server and link utilization. This is computed by calculating the utilization of these resources when the experiment stops, and dividing it by its full capacity. The resource utilization results using a Tree topology are portrayed in Figure 5.

Analyzing Figure 5 we can see that most of the time our system achieves high server utilization. This makes sense since one of the main concerns of our algorithm is doing a best-fit placement of the VMs within the servers. This also shows that our decision of not having a consolidation algorithm running periodically in the controller is correct, since our algorithm already does an incremental consolidation. The server utilization starts to drop when the number of VMs in a virtual network is around 20. This happens because with a request of this size (and larger), some of the VMs have to be placed on different servers, which causes fragmentation of the CPU utilization by a server, which results in a lower server utilization. Obviously, when this happens the network utilization starts to grow, since we have more and more utilized links across the network. We also want to point out that our low network utilization is a result of getting all the servers full before we get some virtual networks that require link usage, since this is just a matter of which resource is exhausted first. Since this is doing what we would expect by looking at the algorithm, we do not think this is relevant.

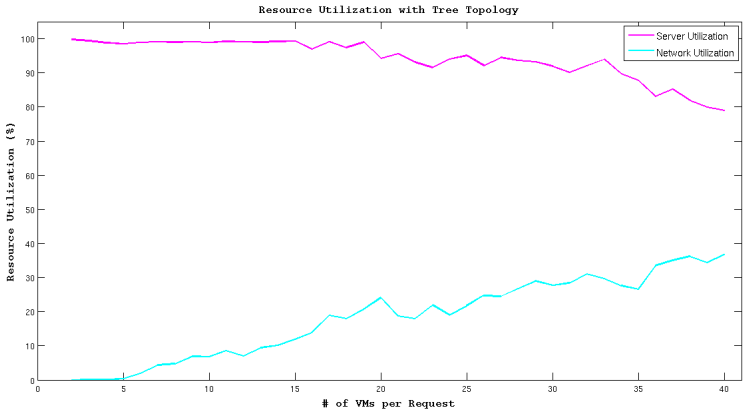

Fig. 5. Resource utilization using a Tree topology.

4.4 Goal IV - Bandwidth Guarantees in a Work-conservative System

We now intend to demonstrate that a server can get more bandwidth than what was requested when the link is free, and gets at least what was requested when the link is saturated. However, we were not able to do a full-fledged evaluation of this goal, beucase of a software bug between Mininet's links and Open vSwitch's queues (described in the full thesis).

To work around this limitation and to prove the concept of our idea, we have devised a different way to test this. We have created a topology with 8 hosts, were 4 hosts generate traffic towards the other 4. Each host generates 50 Mbit/s, and we simulate a 100 Mbit/s link using a queue with the `max-rate` parameter set to this value.

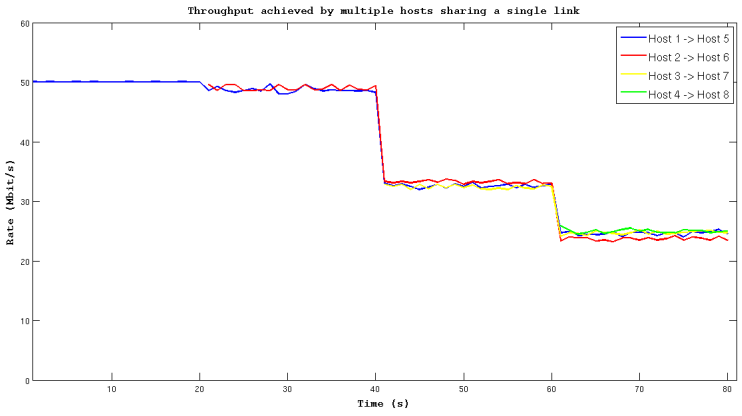

Fig. 6. Rate achieved by each host when all of them are sharing a single link.

In the beginning ($t = 0s$), only *Host 1* is generating traffic with the bit-rate stated above, and with *Host 5* as destination. As he is the only one doing so, he gets the full bandwidth that he is requesting - 100 Mbit/s. This goes on until $t = 20s$, when *Host 2* starts to generate traffic towards *Host 6*, and there are two hosts generating traffic at 50 Mbit/s, which is the

"link capacity". Each host on the right gets pratically the bandwidth that its correspondent is generating. When we reach $t = 40s$, *Host 3* starts to generate traffic to *Host 7*. Now, the sum of the traffic generated by the hosts exceeds the "link capacity", which will cause dropped packets. However, each host gets more than its assured bandwidth (25 Mbit/s), as the three of them divide the link, each one getting about 33 Mbit/s. Finally, at $t = 60s$, *Host 4* begins to generate packets towards *Host 8*. Now, the 100 Mbit/s link is divided by the four hosts, and each one gets about its assured bandwidth.

As we can see by this example, our solution allows hosts to get guarantees about bandwidth, while using more when there are spare resources. This remains to be tested at a large scale, where several hosts generate traffic at the same time.

5 Conclusions

Current Data Centers lack network performance guarantees, since all tenants interchangeably share the network. This makes the performance of a tenant's application unpredictable, since it is depedent on factors outside of its control. This unpredictability severely prevents a wider cloud adoption, since there are many use cases that require network performance guarantees.

These problems are solved using the abstraction of virtual networks. Virtual networks are isolated from each other, providing performance guarantees. We have undertook this challenge using Software-Defined Netwoking. We designed an OpenFlow controller that is able to allocate virtual networks (with bandwidth guarantees) in a work-conservative system, achieving high consolidation on the allocation of virtual networks and high resource utilization of the Data Center's resources.

Our evaluation uses Tree and Fat-tree topologies (although only the first is shown in this document), and shows that we have pretty much accomplished the goals we have set out in the first section. Throughout the evaluation, we can see that our system has: low execution time, high consolidation of virtual networks and high resource utilization. Regarding the last goal (providing bandwidth guarantees in a work-conservative system), due to a conflict in our software stack, we were not able to make the full-fledged evaluation we would like. With this conflict solved, we could undoubtedly achieve all goals, and with it make some more interesting research work.

References

1. Kashif Bilal, Samee Ullah Khan, Joanna Kolodziej, Limin Zhang, Khizar Hayat, Sajjad Ahmad Madani, Nasro Min-Allah, Lizhe Wang, and Dan Chen. A comparative study of data center network architectures. In *ECMS*, pages 526–532, 2012.
2. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
3. Chuanxiong Guo, Guohan Lu, Helen J Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, page 15. ACM, 2010.
4. Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.
5. Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. 2013.
6. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
7. Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.