The Price of Virtualization: Performance Isolation in Multi–Tenants Networks

Roberto Riggio, Francesco De Pellegrini, and Domenico Siracusa
CREATE-NET, Via Alla Cascata 56/C, 38123 Trento, Italy
Emails: {*name.surname*}*@create-net.org*

Abstract—**Network virtualization sits firmly on the Internet evolutionary path allowing researchers to experiment with novel clean–slate designs over the production network and practitioners to manage multi–tenants infrastructures in a flexible and scalable manner. In such scenarios, isolation between virtual networks is often intended as purely logical: this is the case of address space isolation or flow space isolation. This approach neglects the effect that network virtualization has on resource allocation network–wide. In this work we investigate the price paid by a purely logical approach in terms of performance degradation. This performance loss is paid by the actual users of a multi–tenants datacenter network.**

We propose a solution to this problem leveraging on a new network virtualization primitive, namely an online link utilization feedback mechanism. It provides each tenant with the necessary information to make efficient use of network resources. We evaluate our solution trough a real implementation exploiting the OpenFlow protocol. Empirical results confirm that the proposed scheme is able to support tenants in exploiting virtualized network resources effectively.

I. Introduction

Due to the recent introduction of Software Defined Networking (SDN), network virtualization appears one of the rising stars in the area of networking technology innovations. From a practical standpoint, in fact, virtualization is a convenient technique to handle storage and computing resources in a flexible and scalable manner. Furthermore, virtualization configures more and more as the native approach to handle network resources associated to the switches interconnecting modern data centers. In that context, tenants purchase computing, storage, *and* networking resources from a cloud operator. As such, a tenant would expect a virtual network to provide the same usage *experience* as one would expect when operating the same resources on a dedicated physical network deployed at the customer's premises. With respect to the network fabric this corresponds to ensure the *isolation* of the network slices assigned to each tenant.

In practice, a request for a network slice is represented by a virtual network (VN) connecting a set of virtual machines, i.e., the nodes of the virtual network. Also, each virtual link requires a certain capacity. In literature, the problem of mapping VN requests coming from customers on top of the cloud operator infrastructure is know as Virtual Network Embedding (VNE) problem. VNE has been studied in several recent works [1], [2], [3], [4], [5], [6], [7], [8]. Throughout this paper, we will assume that virtual networks requested by customers have been already been embedded onto the substrate network, e.g., making use of available solutions in the literature such as [8].

978-1-4799-0913-1/14/$31.00 © 2014 IEEE

Conversely, our focus is the feasibility of VN isolation. In fact, we argue that, even if an optimal VNE solution is provided, and even if protocols ensure that virtual networks are indeed *logically* isolated, at the link level tenants still share outgoing buffers at each switch. Furthermore, according to the mainstream view of virtualization and network slicing, resources of alien slices are typically hidden by the virtualization layer. Thus, each tenant's controlling logic makes resource allocation decisions *without* complete information on the network status.

The combination of these factors may lead to sub–optimal usage of the networking resources, poor performance isolation between competing slices and, ultimately, fairness issues.

Clearly, one of the major roadblocks toward fair performance isolation in multi–tenants networks would be to enable coordination between the different slices. However, at present it is still unclear how this goal can be achieved. One feasible option could be the definition of a cross–platform/cross–controller interface between SDN–controllers (the so called eastbound interface). An alternative is the introduction of an orchestration layer above all the tenants. Finally, the virtualization layer could provide to the tenant's controlling logic a feedback about the actual network conditions. In this work we set to investigate the latter option, since it has the clear advantage of requiring lesser signaling and has feasible latency requirements compared to the coordination mechanisms mentioned before.

The main contribution of this work is a practical on–line feedback mechanism aiming at providing each tenant with the necessary information about per–link available bandwidth. The proposed solution relies on a hose model in order to specify the tenants' bandwidth requirements. Traffic shaping at the network edges is assumed in order to enforce bandwidth demands. Based on this model, tenants would need to specify just one additional parameter, i.e., the virtual network interface card (VNIC) bandwidth, alongside with the amount of memory and the number of CPU cores.

The rest of the paper is structured as follows. In Sec. II we expose the network slicing problem which is tackled in the rest of the paper. The high level description of our solution is introduced in Sec. III, whereas the system design is presented in Sec. IV. Results of a preliminary evaluation performed using the network emulation tool *mininet* [9] are reported in Sec. V. Finally, Sec. VI resumes our conclusions and draws a roadmap for future research.

II. Fairness and isolation

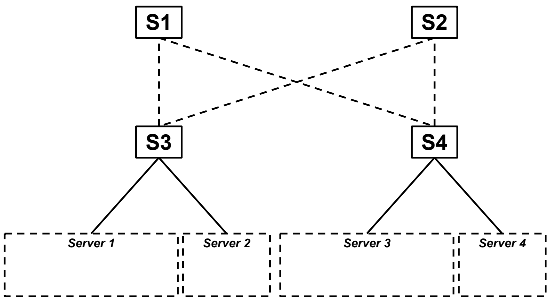
Hereafter, we first illustrate how network virtualization can lead to a sub–optimal utilization of networking resources in multi–tenants networks. In the following sections we will provide an high level description of the proposed solution.

In a network that is fully under the control of the cloud operator, solutions capable of exploiting full bisection bandwidth topologies available in modern data centers [10], [11], [12] do exist. Examples include the Equal–Cost Multi–Path protocol [13] and Hedera [14].

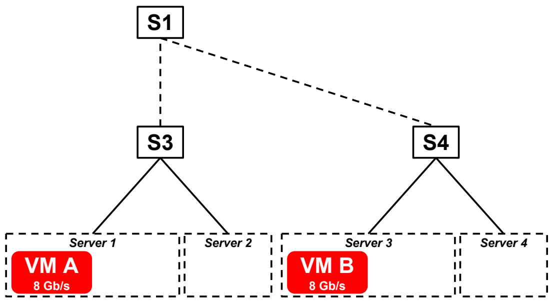
Nevertheless, such techniques alone do not guarantee efficient utilization of network resources. A counterexample is depicted in Fig. 1. In that scenario, two tenants (*Red* and *Blue*) demand one virtual network each. The VNs are composed of 2 and 4 virtual machines, respectively, a set of links and switches. More specifically, the *Red* tenant requires two VMs with a 8 Gb/s NIC while the *Blue* tenant requires two pairs of VMs; the first is equipped with 10 Gb/s NICs, while the second is equipped 2 Gb/s NICs. Notice that the bandwidths associated to each host are the VNIC speeds requested by each tenant during the booking phase (hose model). In this example we assume 10 Gb/s links across the entire infrastructure.

Figures 1b and 1c report a possible mapping between the two requests onto the cloud operator infrastructure depicted in Fig. 1a. For the sake of clarity we assume that both tenants made equal requests w.r.t. computing and storage. In such a scenario, each tenant expects full control of the resources assigned to it. E.g., each tenant may want to implement its own controlling logic in order to perform load balancing between different virtual machines.

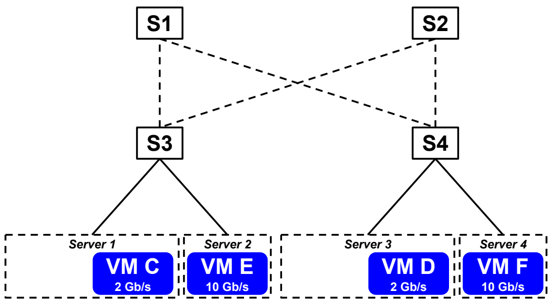
In Fig. 1d we observe that full bisection bandwidth is available at the substrate network and the two virtual topologies can thus be considered an optimal allocation with respect to the capacity allocation. However, because routing decisions are taken locally by the two controllers, flows belonging to *distinct* virtual networks may compete for bandwidth at links ($S1 \rightarrow S3$) of the substrate network (Fig. 1d). More in detail: in the above example the 10 Gb/s link between the switch $S1$ and the switch $S3$ is shared between the two flows originating at hosts A and E and terminating on, respectively, hosts B and F. If active traffic shaping is used at the network ingress points, then it could be reasonable to proportionally share the 10 Gb/s according to the host' demanded VNIC capacity. If no shaping is enforced at the edges, then the two flows would equally share the available bandwidth in the case of elastic (e.g., TCP) traffic and such reasoning still holds with different values of traffic, though.[1]



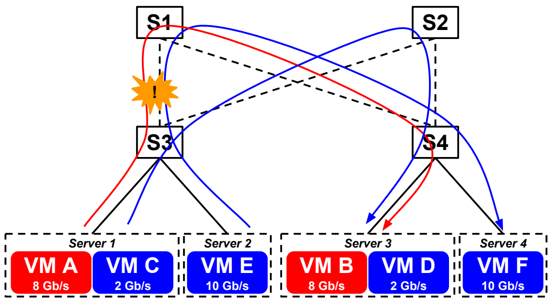
(a) Substrate network (all links are 10 Gb/s).



(b) Red tenant virtual network request.



(c) Blue tenant virtual network request.



(d) Effect on the substrate network.

Fig. 1: Albeit full bisection bandwidth is available in the substrate network, the partial network view exploited by the two tenants in order to make routing decision can lean to sub–optimal use of the available resources.

This will result in 4 Gb/s allocated to the flow $A \rightarrow B$ and 6 Gb/s allocated to the flow $E \rightarrow F$. Finally, the spare 6 Gb/s available at server 3 can be allocated to the flow $C \rightarrow D$. We can observe two effects:

1) *sub-optimality*: despite the full bisection bandwidth (20 Gb/s) is theoretically available (optimal VME), the total aggregated bandwidth accessible to the flows will be then 16 Gb/s.
2) *unfairness*: depite the *Blue* tenant requested a VMs pair with 2 Gb/s NICs and another one with 10 Gb/s NICs, the system eventually gives both VMs pairs same bandwidth. Even worse: the *Red* tenant's VM A bandwidth is even lower than *Blue* tenant's VM E.

[1]However, given the fact that host in a tenant's network can also use non–elastic and bursty traffic, the use of suitable shaping techniques is advised [15].

III. Enforcing performance isolation: concept

In order to guarantee performance isolation in multi-tenants networks our solution leverages on two technical features: (i) bandwidth requests made by the tenants must be enforced at the edges, and (ii) a feedback about the actual network utilization must be provided to each tenant's controlling logic. In this section we shall provide the reader with a conceptual view of our solution, leaving its design and implementation details to Sec. IV.

Enforcing bandwidth requests falls into the broad category of QoE provisioning. An effective and efficient mechanisms is presented in [15] where traffic shaping is used at the edges in order to make sure the endpoints, i.e., VMs, do not overload the network. Notice that albeit elastic traffic, i.e, TCP, can share proportionally the available bandwidth, such sharing is per-flow and not per-tenant. Moreover, TCP congestion control does not handle well bursty and/or UDP traffic, which may be present.

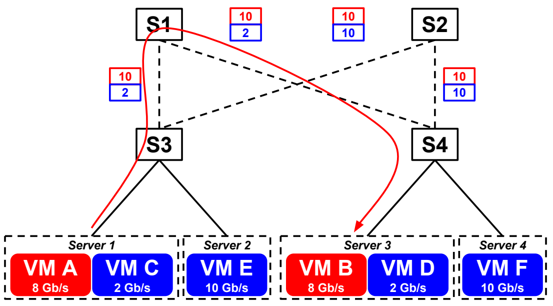
The second component (which is this work's contribution) is a feedback mechanism that provides each tenant with the information required in order to make efficient routing choices. We would like to stress that, today's algorithms commonly used in datacenters, such as ECMP [13], assume full view (and control) of the network resources. Such assumption does not hold anymore for tenants that are given control of the network slice corresponding to their VN.

The mechanism presented in this work aims precisely at restoring such view at the tenant-level in a scalable manner. From a logic standpoint this is equivalent to labeling each link in the network with a weight proportional to the fraction of the link bandwidth in use by *other* tenants. Note that such load information is aggregated over all the tenants and over all the flows sharing the link, i.e., customers ignore how many tenants share the network, nor the type or number of their flows.[2]

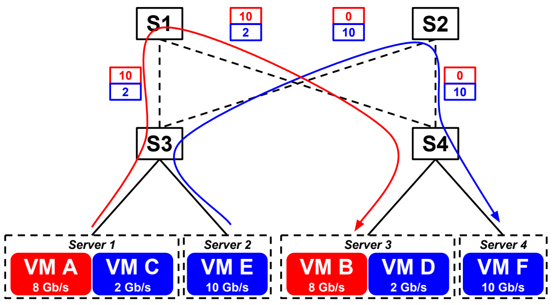
Intuitively, this means that each link in a virtual network is labeled with its residual capacity, computed considering the bandwidth of all other flows currently utilizing that link. We remark again that this formulation assumes that traffic shaping is used *before* such a labeling in order to ensure that the aggregated bandwidth exploited by a virtual machine does not exceed the VNIC's bandwidth assigned during the reservation phase to the tenant owning the VM.

We will now apply this link labeling concept to the previous example. Note that a different set of labels (indicated with different color in the example) is maintained for each tenant in the network, i.e., the *Blue* tenant will see only the *blue* labels while the *Red* tenant will see only the *red* ones.

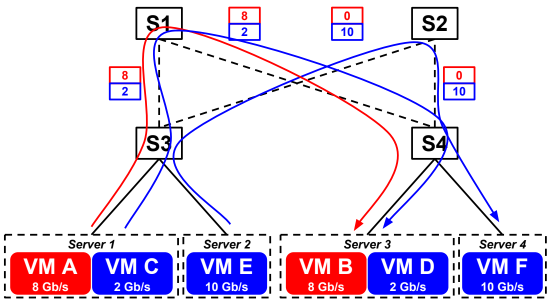
Assume that at the instant t_1 a saturated TCP connection between nodes A and B begins (Fig. 2a). Given the current status of the network, the *Red* tenant's controller decides to route this new flow through the path $S_3 \rightarrow S_1 \rightarrow S_4$ (any other path would have also been satisfactory). The connection can exploit the full line rate of 10 Gb/s. The *blue* labels associated to the link used by the new flow are then updated by subtracting the nominal capacity this flow is allocated. This is done in order to reflect the fact that this new flow shares a link with another virtual machine and will be throttled down if necessary.



(a) Instant t_1, flow $A \rightarrow B$ begins. Labels are updated reflecting the flow nominal capacity. Assuming work–conserving shaping, the flow is provisionally allocate full line rate.



(b) Instant t_2, flow $E \rightarrow F$ begins. An alternative path is selected by the *Blue* tenant's controlling logic. Labels are updated and the flow is assigned full line rate.



(c) Instant t_3, flow $C \rightarrow D$ begins. The *Blue* tenants routes the flow over the path with higher residual bandwidth. Network resource usage is maximized.

Fig. 2: Resource allocation with link labeling.

At instant t_2 (Fig. 2b), node E starts a new saturated TCP connection to node F; given the network status, the *Blue* tenant's controller decides to route this new flow trough the path $S_3 \rightarrow S_2 \rightarrow S_4$. The new flow can then exploit the 10 Gb/s path between nodes E and F.

Finally, at instant t_3 (Fig. 2c) a bursty UDP connection is generated between nodes C and D (Fig. 2b). The *Blue* tenant's controller can now either decide to share the bandwidth with its other flow (of which it has visibility) or load balance the new connection on the alternative path. Let's assume that the controller chooses to exploit the $S_3 \rightarrow S_1 \rightarrow S_4$ path for the new flow. The rate control algorithm detects the *Blue* tenant flow at node B and implements traffic shaping at both nodes A and B in such a way to avoid congestion at the receiver

[2]Privacy issues of malicious tenants inferring and exploiting the behavior of other tenants is out of the scope of the paper.

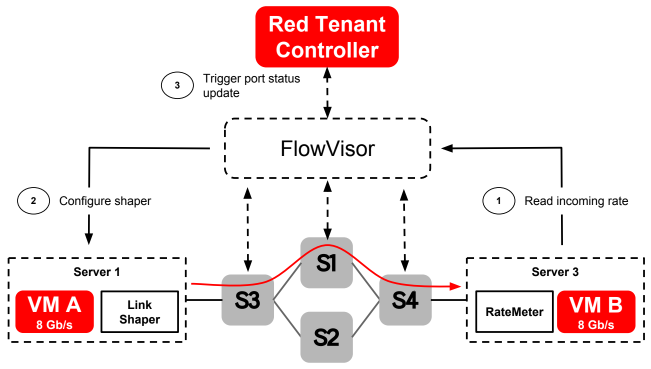


Fig. 3: The design of our performance isolation scheme relies on a distributed bandwidth shaping system combined with a reactive link labeling scheme and a label distribution mechanism exploiting existing OpenFlow messages (the *port status* message) although with minor modifications.

side. As a result the flow $A \rightarrow B$ is allocated 8 Gb/s while the flow $C \rightarrow D$ is allocated 2 Gb/s.

Observe that, by leveraging link labels, each tenant's controlling logic can actually choose routing paths that do not collide onto the substrate network *without* leveraging either a direct communication (impractical for a large number of tenants) or an orchestration mechanism (limiting the tenant's control on its virtual network).

IV. Solution Design

The system architecture implementing our solution is sketched in Fig. 3. As it can be seen, our performance isolation technique relies on three logical blocks:

- a traffic shaping mechanism scheduling each tenant flows according to the bandwidth demands made during the reservation phase;
- a link labeling block exposing each tenant with the amount of bandwidth it can exploit on a given link;
- a label distribution unit promptly updating such label upon detecting flow changes in a virtual network.

We would like to stress once again that in this work we assume that the VN request coming from the tenants have already been embedded onto the substrate network using one of the algorithms already available in the literature. The temporal relationship between the actions performed by the three modules is reported in Fig. 3. In particular, new flows are first detected at their termination point, triggering a rate update message. Such message is processed at the virtualization layer in order to configure the traffic shaper at the flow origination point. After end-to-end flow control has been achieved, the virtualization layer updates the link label and notifies each tenant controller of the changes. In this section we shall describe in detail the three components. Observe that, albeit the performance isolation challenges highlighted in the previous sections are independent from the particular networking virtualization stack employed by the data center operator, our system design and its evaluation will rely on the OpenFlow protocol [16].

OpenFlow is a flow–based forwarding plane abstraction which allows network developers to control how packets are forwarded in a switched network exploiting a vendor–agnostic interface, i.e. the OpenFlow protocol. The simplicity of the flow abstraction which lies at the heart of OpenFlow has paved the way to network slicing solutions such as FlowVisor [17]. FlowVisor allows infrastructure providers to partition network resources among different tenants while giving each of them full control of their virtual network (in the OpenFlow terminology, each tenant is allowed to run its own controller).

A. Traffic Shaping

Traffic shaping and in general QoE enforcement is a widely investigated topic: our design choice has been EyeQ [15]. EyeQ is a system capable of providing tenants with bandwidth guarantee in data centers networks. The main assumption of EyeQ is that full–bisection bandwidth is available in the switching fabric, which is also one of our assumption alongside with the hose model used by tenants to make bandwidth requests.

The EyeQ system exploits a distributed architecture leveraging rate meters and link shapers. The former (rate meters) detect possible congestion situation at the flow termination point and provides the later (link shaper) with the necessary information in order to enforce admission control at the flow origination point. Doing so, EyeQ performs congestion control at a time scale smaller than the one typically implemented by TCP, thus avoiding possible performance oscillations.

B. Link labeling

Residing between the switching fabric and the control layer, FlowVisor has complete knowledge of all network flows. More specifically, according to the OpenFlow model, switches are requested to contact the controller in order to learn how to treat new flows. Alternatively, the controller can push proactive flow processing instructions in order to reduce signaling overhead at least for a set of flows. In either case FlowVisor sits between the controller and the actual switches marshaling all the signaling traffic exchanged by the two entities. As a result, at any instant FlowVisor has a detailed view of all the flows currently active in the network for any tenant and thus can compute link labels.

C. Labels distribution

Finally, in order to implement the last piece of our architecture we need to modify the OpenFlow port status message. This message is used by OpenFlow switches to notify their controller about changes in the status of a port, e.g. link up/down event. The port status message is defined by the OpenFlow 1.0 standard as follows:

```
struct ofp_port_status {
  struct ofp_header header;
  uint8_t reason; /* One of OFPPR_*. */
  uint8_t pad[7]; /* Align to 64-bits. */
  struct ofp_phy_port desc;
};
```

where the `reason` can be *add*, *delete*, or *modify*. Each physical port is fully described using the following structure:

```
struct ofp_phy_port {
  uint16_t port_no;
```

```
    uint8_t hw_addr[OFP_ETH_ALEN];
    char name[OFP_MAX_PORT_NAME_LEN];
    uint32_t config;
    uint32_t state;
    uint32_t curr;
    uint32_t advertised;
    uint32_t supported;
    uint32_t peer;
    uint32_t label;
};
```

Observe the additional `label` field added at the end of the structure. Port status message are typically generated by the switch as physical ports are added, modified, and removed from the datapath and directed to the controller. In a virtualized environment FlowVisor will receive port status updates from the switches and will rely them to the suitable controller(s). Similarly, FlowVisor can trigger new port status messages in order to convey to the controller an updated set of labels.

Albeit such change may appear disruptive, it shall be also considered that OpenFlow currently accommodates for optional features. Examples are traffic counters and headers rewriting, both features are not mandatory for vendors to implement. Similarly, we can envision our label distribution protocol to be implemented using an optional OpenFlow message. Nevertheless, for ease of implementation we decide to modify the port description structure for the purposes of this work.

V. Performance Evaluation

We would like to point out that albeit no additional signaling is requested compared to standard operations, our solution mandates FlowVisor to track all the flows in the network, as opposed to its normal passive mode where only a policy compliance check is performed. Nevertheless, we decided not to report on the performance penalty incurred by applying our technique. Our implementation is hence to be considered a proof–of–concept with little or no code optimization because the main goal here is to prove the viability of the proposed approach.

A. Methodology

Mininet [18] has been used in order to evaluate the proposed link labeling solution. Mininet emulates a network of software–based virtual switches [19] supporting (among other things) the OpenFlow protocol. Mininet enables testing of OpenFlow–based applications before deploying them on the actual network. Given the fact that packets are exchanged between several Open vSwitches instances running on the emulating machine, Mininet is best suited for testing new functionalities rather the crude network performance. For example, emulating the network setup sketched in Fig. 1 allowed us to achieve up to about ≈ 120 Mb/s end–to–end TCP throughput between a pair of nodes. The testing machine was a laptop equipped with an Intel Core i7 (quad core) running at 2.7 GHz with 8 Gb of RAM. As result, we decided to emulate a network exploiting 10 Mb/s links.

In order to evaluate our solution, we replicated the scenario reported in Sec. II. More precisely, the first flow $A \rightarrow B$ arrives at the instant $t_1 = 0s$, while the second ($E \rightarrow F$) and third ($A \rightarrow B$) flows arrive respectively at instants $t_2 = 60s$ and $t_3 = 120s$. All flows are saturated TCP streams generated using the iperf [20] tool. Slicing is performed by exploiting the VLAN field. Each tenant runs its own controller instance. A modified version of the Pox [21] OpenFlow controller is used in this experiment. The additional feature we implemented supports load balancing across multiple paths exploiting link labels.

Both the link labels calculation and the setup of traffic shapers are implemented off–line: they have not been integrated into FlowVisor yet. Instead, they have been computed within the Mininet scripting environment and then sent to the controller over a dedicated signaling channel. All the results reported in the following section are the average of 10 runs. Confidence intervals are always smaller than 0.1 Mb/s.

B. Results

The results of emulation–based evaluation are reported in Fig. 4. In particular Fig. 4a shows the TCP throughput of each active flow in the network. As it can be seen, as long as the $A \rightarrow B$ is the only active flow in the network, full rate can be achieved. However, when then second and third flows start, the actual network utilization depends on the routing choices made by the *blue* tenant controller.

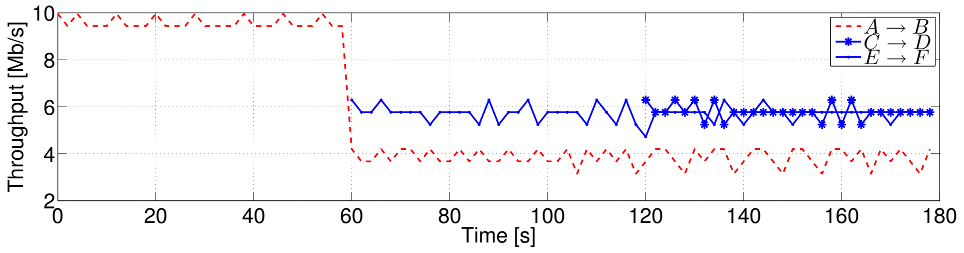
The final bandwidth allocated to the three flows are reported in Fig. 4a. We assumed that the controller randomly selects one of the two available paths with probability 0.5.

On the other hand when link labels are exploited by the controllers, it is always possible to converge to the optimal network bandwidth utilization reported in Fig. 4b. The result of this behavior is made clearer in Fig. 4c where the instantaneous aggregated network throughput is plotted. As it can be seen in that figure, without link labels the aggregated throughput in the worst case scenario (which, we remind, happens with probability 0.5) is just 10 Mb/s. When link labeling is used it approaches closely the nominal network capacity of 20 Mb/s.

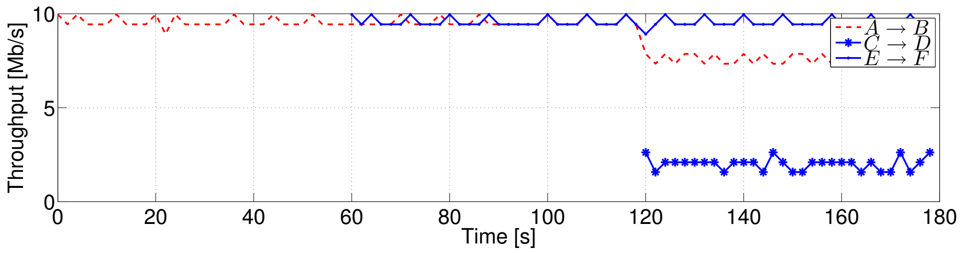
VI. Discussion

Virtualization of computing and storage resources has allowed cloud providers to improve spatial reuse by multiplexing tenants across a wide array of high volume servers. Also, tenants pay per resource usage only, thus cutting entrance costs for leveraging on advanced computing assets. Similar motivations are nowadays pushing toward a virtualization of the fabric of switches that interconnects data centers. In this context, tenants will expect to have full control of their virtual networks. However, albeit network virtualization has made significant strides forward, current solutions still leave each tenant blind to the resource allocation performed by other tenants. As a matter of fact, unlike in the computing domain where advanced hypervisor technologies provides effective resource partitioning, the SDN community is only now starting to investigate network virtualization abstractions, which deliver true performance isolation.

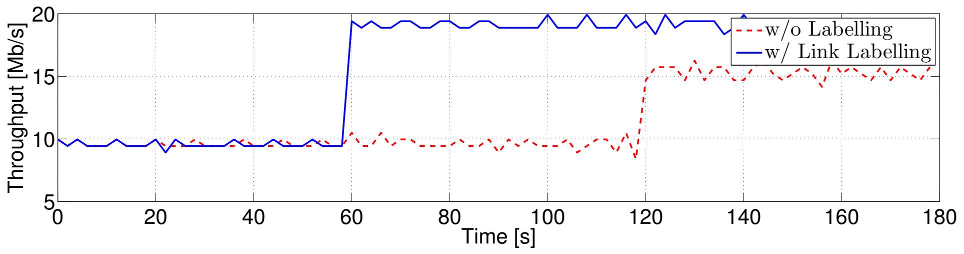
In this paper we provided a preliminary empirical evidence that sub–optimal resource allocation may result from the



(a) Per–flow throughput without labeling.



(b) Per–flow throughput with labeling.



(c) Aggregated throughput.

Fig. 4: Link labeling allows to achieve up to 100% performance increases in the worst case scenario. Flow rate samples are taken every 2s. Results are the average of 10 runs. Confidence intervals were smaller than 0.1 Mb/s.

utilization of state–of–the–art network virtualization technologies. We also designed, implemented, and tested a new virtualization primitive. It leverages an online feedback mechanism providing at runtime each tenant with minimal information to make sensible resource allocation choices. As part of future work we are developing a proof–of–concept implementation and we are investigating the effectiveness of the feedback mechanism in the case of malicious tenants.

References

[1] Y. Zhu. and M. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 23-29 2006.

[2] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 206 –219, February 2012.

[3] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," in *Proc. of ACM VISA*, 2010.

[4] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang, "Davinci: dynamically adaptive virtual networks for a customized internet," in *Proc. of ACM CoNEXT*, 2008.

[5] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," in *Proc. of IEEE ICC*, 2011.

[6] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. of ACM VISA*, 2009.

[7] Y. Minlan, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[8] R. Riggio and F. D. Pellegrini, "Progressive virtual topology embedding in openflow networks," in *Prof. of IEEE ManFi*, Ghent, Belgium, 2013.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. of ACM CoNEXT*, 2012.

[10] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, Aug. 2009. [Online]. Available: http://doi.acm.org/10.1145/1594977.1592575

[11] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers," in *Proc. of ACM SIGCOMM*, Seattle, WA, USA, 2008.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. of ACM SIGCOMM*, Barcelona, Spain, 2009.

[13] C. Hopps, "Analysis of an equal-cost multi-path algorithm," United States, 2000.

[14] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. of USENIX NSDI*, San Jose, CA, USA, 2010.

[15] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: Practical network performance isolation at the edge," in *Proc. of USENIX NSDI*, Lombard, IL, USA, 2013.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Can the production network be the testbed?” in *Proc. of USENIX OSDI*, Vancouver, BC, Canada, 2010.
[18] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proc. of ACM HotNets*, Monterey, CA, USA, 2010.
[19] “Open vSwitch.” [Online]. Available: http://openvswitch.org
[20] “Iperf.” [Online]. Available: http://iperf.sourceforge.net/
[21] “POX OpenFlow Controller.” [Online]. Available: http://www.noxrepo.org/pox/about-pox/