Analysis of Virtualized Congestion Control in Applications based on Hadoop MapReduce*

Vilson Moro[000−0003−3680−06643],
Maurício Aronne Pillon[0000−0001−7634−6823],
Charles Christian Miers[0000−0002−1976−0478], and
Guilherme Piêgas Koslovski[0000−0003−4936−1619]

Graduate Program in Applied Computing, Santa Catarina State University, Brazil
vilson.moro@edu.udesc.br, {firstname.lastname}@udesc.br

Abstract. Among the existing applications for processing massive volumes of data, the Hadoop MapReduce (HMR) is widely used in clouds, having above all internal network flows of different volume and periodicity. In this regard, providers have the challenge of managing data centers with a wide range of operating systems and features. The diversity of algorithms and parameters related to TCP constitutes a heterogeneous communication scenario prone to degradation of communication-intensive applications. Due to total control in the data center, providers can apply the Virtualized Congestion Control (VCC) to generate optimized algorithms. From the tenant's perspective, virtualization is a transparently performed. Some technologies have made possible to develop such virtualization. Explicit Congestion Notification (ECN) is a technique for congestion identification which acts by monitoring the queues occupancy. Although promising, the specialized literature lacks on a deep analysis of the VCC impact on the applications. Our work characterizes the VCC impact on HMR on scenarios in which there are present applications competing for network resources using optimized and non-optimized TCP stacks. We identified the HMR has its performance substantially influenced by the data volume according to the employed TCP stack. Moreover, we highlight some VCC limitations.

Keywords: Virtualized Congestion Control · Hadoop MapReduce · TCP

1 Introduction

One of the key aspects of Infrastructure-as-a-Service (IaaS) clouds is the manageability and malleability provided to its tenants. Specifically, a virtual machine (VM) can be of several flavors, composed by different configurations of memory, storage, and CPU. Above all, the client has the control over the operating system (OS) installed on the VM, being able to install and update: applications, libraries, and OS kernel. Specifically, focusing on Transmission Control Protocol (TCP), two scenarios are relevant due to the heterogeneity

* Supported by UDESC / FAPESC, developed on LabP2D.

of its configurations [10]: *(i)* The maintenance complexity and dependencies of specific software versions are limiting factors for VMs updating. OSs with non-optimized TCP do not incorporate the latest advances on algorithms related to congestion control and avoidance; and *(ii)* Optimized settings related to temporary buffers, slow-start, selective acknowledgment, and so on may circumvent the fairness originally aimed by congestion control algorithms.

Algorithms for congestion control aid in the recovery of the performance in the occurrence of bottlenecks, while congestion avoidance algorithms act in the prediction of eventual losses [3]. Traditionally, TCP controls and avoids congestion at the endpoints of the network [9], without native support of the network core equipment. Several versions of TCP have been proposed to optimize traffic performance in data centers (DCs) [21, 1] by changing the interpretation of binary feedback originally conceived. In some cases, the core network participates by offering packet tags with alerts of possible bottlenecks [4]. In fact, it is in the interest of the provider to circumvent the problem imposed by the heterogeneous scenario of congestion control algorithms, qualifying the service offered to their customers. Possible solutions include the provision of communication resources (bandwidth and configuration of switches) [19, 22, 12, 20] and dynamic routing queue configuration [10]. However, the techniques listed tend to underuse communication resources or to have high management complexity [17]. In this context, Virtualized Congestion Control (VCC) was proposed as an alternative to standardize non-optimized TCP algorithms [5, 8]. In summary, the providers have administrative access to the virtual switches or VM hypervisors responsible for routing the packets between the tenants' VMs. Thus, a virtualization layer can intercept non-optimized traffic and handle, when necessary, to meet the requirements of the updated DC protocols. Although promising, the specialized literature lacks on a deep analysis of the VCC impact on the final applications.

Originally, VCC was analyzed with synthetic charges indicating the efficacy for scenarios aiming at equity of sharing and maximization of use in bottlenecks for mostly long flows [5, 8]. However, actual communication loads of applications traditionally run on clouds DCs have not been analyzed. Specifically, applications based on Hadoop MapReduce (HMR) have communication flows with characteristics different from those originally studied [1, 18]. Thus, the aim of the present work is to analyze the applicability of VCC to HMR applications. Our main contributions are: *(i)* Analysis of the execution time of HMR, using real execution traces, in heterogeneous scenarios using VCC; *(ii)* Discuss the perceived impact on HMR regarding the configuration of the marking on the switches queue; and *(iii)* Correlate the dropped packets in the switches with the total execution time. Our experimental analysis indicates applications based on HMR have a considerable impact when executed on VCC-based environments[1].

[1] This paper is a revised and expanded version of [14].

2 Motivation and Problem Definition

2.1 Data Flow in Hadoop MapReduce

HMR framework is widely used, both for structured and unstructured data processing. The optimization of HMR data communication is realized by scheduling processing on servers close to the data to be processed. The main server, called *Master*, takes a metadata view of the entire file tree of the system and manages the distribution of the tasks to the servers which will process the data (*Workers*). The *Master* monitors the execution in *Workers* and defines the tasks each one will execute, either the *map* or *reduce*. It is important to note that 33% of the execution time of HMR is attributed to the TCP-based communication tasks [4], motivating the accomplishment of this work. Above all, HMR clusters present a data locale dependency, overloading switches and generating congestion during different stages of execution [18]. Specifically regarding the TCP perspective, the HMR flows suffer with switches queues and packets dropped, which triggers retransmissions on TCP congestion control algorithms.

Applications based on partitioning, processing, and aggregation of information, such as HMR, carry control data sensitive to latency, as well as flows for data synchronization. Thus, latency-sensitive flows compete with background traffic. Analyzing the occupation of the switches queues, it is possible to note HMR flows larger than 1 MB have low multiplexing rate and consume a large portion of the available bandwidth [1], inducing the increased of the latency for the others flows. DCs are designed for high throughput and low latency. However, commonly used switches have shared memory-based architectures. Thus, when multiple flows converge to the same interface, the storage space in queues can be totally consumed, leading to drop packets.

2.2 TCP Congestion Control in DCs with Multiple Clients

Originally, because TCP is based on binary feedback [3], it only changes its mode of transmission in the occurrence of segment loss. In the absence of losses, the transmission window is increased, while in the occurrence, it is decreased [7, 9]. Specifically, congestion is perceived when a timer is terminated or when a set of duplicate ACKs are received by the sender. Although efficient in the scenarios originally proposed (mainly related to communication over networks with distinct configurations and capacities), the algorithms are unable to detect and control the previously listed scenario for HMR, common in DCs [1].

The Explicit Congestion Notification (ECN) technique, an extension of the TCP/IP stack, has modified the feedback received by the TCP sender, *i.e.*, packet loss is eventually replaced by a warning of the possible occurrence of congestion. Based on this warning, the sender can preventively reduce the volume of traffic data, mitigating queuing on the intermediate switches. As for the protocols, the implementation of this mechanism was accomplished through the introduction of 2 bits in the network layer: ECN-Capable Transport (ECT) and Congestion Experienced (ECE). The first one indicates the equipment is capable

of conveying congestion notification, while the second one signals a congestion situation is happening. In the transport layer of destination, when the notification is identified, the next segment is marked to inform the congestion situation. Upon receiving the notification, the sender reduces the congestion window to avoid loss of segments, and informs its action to the receiver via the bit Congestion Window Reduced (CWR).

The packets experiencing congestion are marked by the intermediate switches. For this, a constant monitoring of the switching queues is performed, triggering actions according to the parameters previously defined. In this context, configurations of Random Early Detection (RED) [21] can be applied by specifying: *(i)* The maximum bandwidth of the link (bytes/s); *(ii)* The the minimum, maximum and snapshot size (to meet bursts) of the queue (bytes); *(iii)* The average packet size; *(iv)* The tolerance for the snapshot size of the packet queue; and *(v)* The drop packet probability (from 0.0 to 1.0).

When the packet is received and remains below the minimum specified, no marking occurs. However, packets which are between the minimum and maximum thresholds are marked according to the reported probability. Finally, packets above the maximum threshold are dropped. It is a fact ECN allowed a better sharing of communication resources in DCs [2]. However, the effective application requires uniformity of the algorithms, *i.e.*, the servers must understand the meaning of the markings realized by the network core. Usually, this requirement is not take into account in cloud DCs, in which each tenant can execute different TCP algorithms. Also, several versions of OSs do not have support enabled by default for ECN [11]. Thus, while DC implements mechanisms to optimize traffic, the settings applied on VMs may be conflicting with the ideal scenario.

Although the TCP congestion control algorithms have been improved by the specialized literature, it is evident that a multi-tenant DC network carries flows originated from distinct and competitive versions. Moreover, as HMR is a network-based paradigm, the final performance of an HMR-based application is directly influenced by the TCP congestion control objective (*e.g.*, fairness, low latency, high throughput), even when executed by distinct tenants.

3 Virtualized Congestion Control (VCC)

The VCC consists in creating a translation layer for the TCP used by VMs in order to translate it to an optimized/recognized version used in the DC [8, 5]. In this way, the communication occurs using the TCP version selected by the provider. It is worthwhile to mention, no changes are needed on the endpoint hosts. Two approaches to VCC implementation have recently been proposed. AC/DC [8] implements VCC in virtual switches, obtaining a fine granularity in the control. Thus, algorithms can be selected for different types of flows, *e.g.*, Cubic for external flows to DC and Data Center TCP (DCTCP) for internal flows. Due to the control is implemented in the virtual switch datapath [16], all traffic can be monitored. [5] implemented VCC directly in the hypervisor of

VMs. In both, the perception of congestion is obtained through network core marked packets. Fig. 1 presents the canonical architecture for VCC.

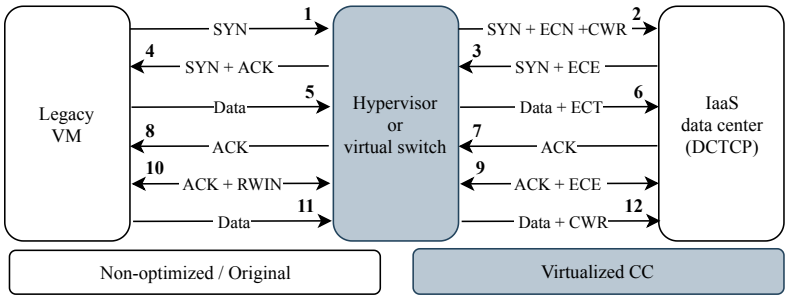


Fig. 1. Example of VCC usage.

When a non-optimized (or legacy) application establishes a connection (Fig. 1), the hypervisor (or virtual switch) monitors the exchange of packets and includes the information necessary for non-optimized traffic to be recognized by the network as traffic capable of supporting ECN. Initially, non-optimized TCP application sends a packet requesting connection (1), which is intercepted (2) to append the support information ECN. The recipient responds by confirming the establishment (3, ECE). Again, the packet is intercepted to notify the sender with a synchronization acknowledgment (4). It is important to note, the confirmation sent to the sender with non-optimized TCP algorithm does not have the information about ECN, which was removed by the virtualizer. The data sending starts (5), being intercepted to add the bit ECT, informing this flow is capable of conveying congestion information (6). Subsequently, the recipient acknowledges the received packet (7). The sender receives packet recognition from the hypervisor (8). In case of possible occurrence of congestion in the DC network, the *ECE* bit is activated (9). When congestion occurs, the sender with an optimized algorithm is forced to reduce the sending of data through the bottleneck of the receiving window (10), performed by the hypervisor. Finally, the sender continues to send data (11) to the hypervisor that transfers the data and the window size set to the recipient (12).

In order to induce the host transmission deceleration with non-optimized TCP without applying an intrusive technique, the congestion control is applied over the Receiver Window (RWND). The information internally measured by the algorithm in VM over the congestion window, congestion Window (CWND), remains unchanged. Natively, TCP checks $min(cwnd, rwnd)$ to identify the amount of data which ca be transmitted. On VCC, the value of RWND is changed to represent the correct value of CWND calculated by the virtualization algorithm, based on ECN.

AC/DC mechanism source code was not found available to the community. Thus, [5] was selected for our analysis (Section 4). VCC was implemented in Linux by a patch to the hypervisor core. In short, a set of intermediate buffers were created for each TCP connection, *i.e.*, the hypervisor monitors the communication of the VMs. In order to differentiate non-optimized flows and ECN

configurations, the capabilities of VCC and ECN are activated directly by manipulating the *sysctl*. It is important to emphasize the described implementation is a proof of concept. However, the authors have demonstrated the computational overhead does not impact on the final performance of the communications, *i.e.*, the implementation can be used for controlled experimental analysis [5].

4 Experimental Analysis

4.1 Testbed setup, Metrics and Execution Traces

The experiments aim to analyze the impact of VCC on the execution of HMR applications when executed on VM using non-optimized TCP. Data traffic in HMR was emulated by MRemu [15] using traces of HiBench benchmarking in a cluster consisting of 16 servers. MRemu only emulates the HMR traffic (using Mininet [13]), not performing the data processing. Our experimental testbed is composed of 16 VMs interconnected by a 1 Gbps Dumbbell topology (2 switches with 8 VMs each). Dumbbell topology was chosen to simplify the representation of the network bottlenecks. Background traffic was ingested using *iperf* tool, setting the number of communicating TCP pairs in order to represent multiple tenants disputing the communication capabilities. The *iperf* servers are connected to *Switch 1* while clients are connected to *Switch 2* (data source). As for the physical host, MRemu and Mininet were executed on a computer with OS GNU/Linux Ubuntu 14.04, AMD Phenom II X 4 core processor, and 8GB RAM. The VMs originally executed the TCP *New Reno*.

To analyze the results, four metrics were collected: *(i)* Elapsed time to run HMR; *(ii)* Number of dropped packets on switches; *(iii)* Switches queue occupation; and *(iv)* Amount of background traffic. The first metric represents the view of the tenant, while the other metrics provide data to do the analysis of the dropped packets and queue formation in switches from traffic of non-optimized TCP, virtualized, or optimized (IaaS provider perspective).

4.2 Experimental Scenarios

Three TCP configurations were used in our experimental scenario: *(i)* TCPVM: HMR using non-optimized TCP, without support for ECN; *(ii)* TCPDC: HMR using TCP optimized by the DC, support enable for ECN; and *(iii)* VCC: HMR using Virtualized Congestion Control (VCC). Background traffic was executed with TCPDC on all presented scenarios, and HMR was executed on TCPVM, TCPDC, and VCC. ECN and RED switches configurations were based on two configurations from the bibliography (Table 1) [1, 5]. The first configuration, RED1, the values of *min* and *max* are close, therefore, with no margin of adequacy. In addition, the *prob* parameter is set to mark 100% of packets, forcing the rapid reduction of traffic. The configuration identified as RED2, there is a suitability interval established between the values *min* and *max*. This interval allows the matching of the traffic according to the congestion notification by marking packets. Still, there is a difference in the maximum queue size.

Table 1. RED settings for marking packets in the *switch* queues.

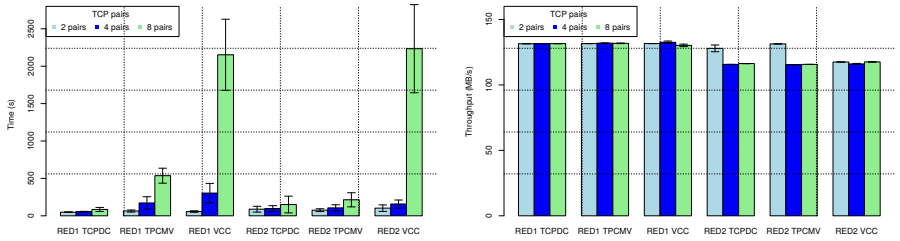
Configuration	*min*	*max*	*limit*	*burst*	*prob*
RED1	90000	90001	1M	61	1.0
RED2	30000	90000	400K	55	1.0

Regarding to the volume of data transferred between the pairs (Client/Server), responsible for the production of background TCPs traffic, two scenarios are analyzed. Scenario 1 was based on unlimited background load and there are 3 sets of experiments (2, 4, and 8 pairs), in which is performed a gradual increase in the number of pairs, and each communicating pair sent the maximum allowed by the application (and bottleneck). Scenario 2 was planned to have a controlled background traffic, using TCP load from 2MB to 32GB, and 4 pairs. We run each scenario 10 rounds, the presented graphs comprise mean, standard deviation, and variability of the data.

4.3 Result Analysis

The first analysis focuses on the impact of the chosen TCP, and the two configurations of RED, tenant's perspective. Thus, the elapsed time of HMR execution was observed, considering smaller the time than better is the result.

Scenario 1 Fig. 2 shows the HMR execution results. Fig. 2(a) indicates TCPDC has the smallest values, regardless of the number of concurrent pairs and RED configuration. TCPVM is most susceptible to the configuration difference of RED. For example, at 8 pairs, the mean time with RED1 was reduced from $535s$ to $213s$ using RED2. In this case, we identified the importance of proper RED configuration to the application behavior. The greatest peculiarity of the results was the behavior of VCC, which obtained results compatible with the other protocols when analyzed with 2 pairs, both RED1 and RED2. Using 4 pairs, it remained competitive only with RED1, and proved inefficient with 8 pairs. Using 8 pairs, while HMR / TCPDC using RED1 gets an average time of $84s$, VCC gets $2153s$. Worst results were obtained using RED2, reaching to $2234s$. Different from the results initially obtained with VCC [5, 8], the analyzed HMR applications undergo a considerable overhead at run time. Moreover, the application of VCC is influenced by the ECN and RED configurations, information that is abstracted from the tenants. On the other hand, background traffic (Fig. 2(b)) is practically the same, regardless of which TCP protocol was chosen (when the configuration is RED1). RED2 behavior differs only when 2 pairs are used. *Iperf* reaches values close to 130MB/s using TCPDC / TCPVM, and 120MB/s using VCC. The lower throughput identified using RED2 was from packet conservative marking, *i.e.*, due to the wide marking interval, the TCP emitters have a conservative behavior, reducing the data sending in order to avoid congestion. Finally, the background traffic is independent of the HMR execution time.



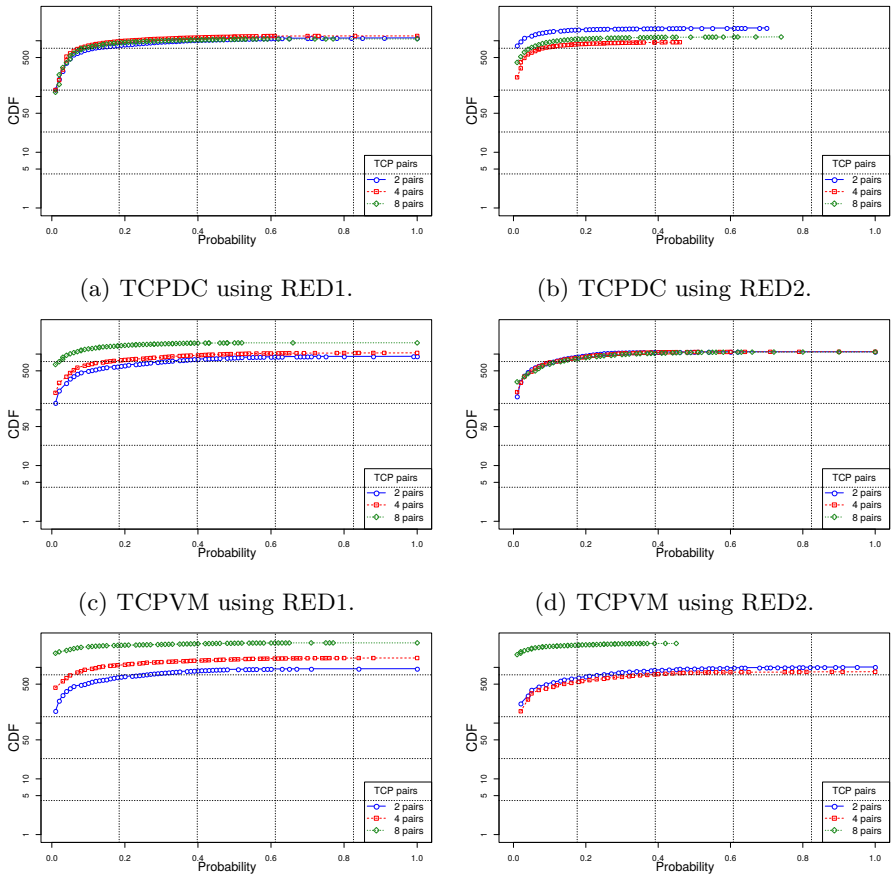
(a) HMR execution time. (b) Throughput of background traffic.

Fig. 2. HMR execution time and throughput of background.

The second approach is the analysis in the provider's perspective. In this case, the observation is in the traffic of the switches, through the occupation of the queues. Figs. 3 and 4 resume the results for *Switches* 1 and 2, respectively. Specifically, Figs. 3(a) to 4(f) axis x presents the probability of occurrence and, axis y the accumulated value of the queues (expressed as CDF - Cumulative Distribution Function). The tendency of queuing behavior in the switches is the higher the number of TCP pairs, than greater the number of queues. This behavior was noted in all the scenario variations for the TCPVM protocols and VCC, the only exception was Figs. 3(a) and 3(d) which are all similar. TCPDC and RED2 protocols, regardless of the switch, 2 pairs presents a larger queue occurrence values than 8 pairs, indicating the total occupancy (400000 bytes) and holding sharing equity (indicated by the throughput, Fig. 2(b)).

We also counted the number of dropped packets in each scenario (Figs. 5(a) and 5(b)). The smallest losses are with the TCPDC protocol, regardless of the number of pairs or switch. *Switch 1* (connecting HMR and *iperf* servers), the number of dropped packets using RED2 is greater than the number dropped by RED1, both for TCPVM and for VCC. *Switch 2* (connecting HMR and *iperf* clients), the highlight is the high number of packets dropped by VCC to 8 pairs, regardless of RED configuration adopted. Finally, it is important to note that approximately 75% of the data flows transmitted in the analyzed HMR applications carry a maximum of 7MB, being 10MB the largest volume transmitted. The volume is less than originally analyzed in VCC [5, 8], being susceptible to the impact of the ECN mark. The queues occupation in the switches indicated the RED configuration is directly related to the application. Even if we control the queue occupation, the volume of dropped packets (Figs. 5(a) and 5(b)), especially for *Switch 2*) is abusive to VCC when compared with TCPDC.

In summary, the analysis has shown VCC application is promising for pursuing fairness in a heterogeneous environment, such as clouds (presented by the background throughput on Fig. 2(b)). However, the use of VCC for HMR applications requires the proper configuration of the switches queues, a task commonly performed by the providers because it is needed administrative access to configure it [10, 17].



(a) TCPDC using RED1.

(b) TCPDC using RED2.

(c) TCPVM using RED1.

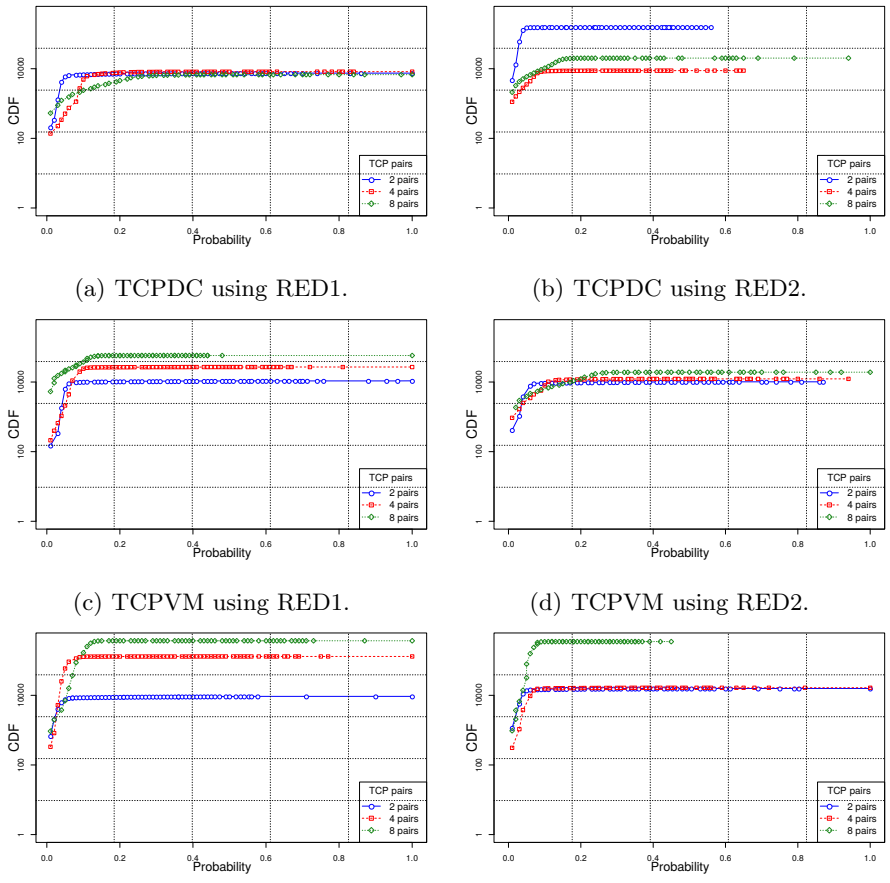
(d) TCPVM using RED2.

(e) VCC using RED1.

(f) VCC using RED2.

Fig. 3. Queue occupation on *Switch 1*.

Scenario 2 This scenario uses 4 ECN-aware TCP pairs to generate background traffic while HMR is being executed. Fig. 6 presents the HMR execution time for RED1 configuration, and Fig. 6(a) reveals that there is no direct correlation between background load and HMR execution time when ECN is used for all nodes (including HMR). In turn, Fig. 6(b) summarises results for HMR without ECN support. Background traffic with loads between 1GB and 32GB induced an overhead on HMR execution time. When applying the VCC (Fig. 6(c)), the problem is soften for background loads between 1MB and 256MB. However, a high variation is observed on all other scenarios, justified by the switches queue occupancy. Finally, the results for RED2 configuration are summarized in Figs. 6(d)-6(f). The minimum threshold for RED2 configuration is 1/3 of RED1 value, leading to an early packets marking, while the upper-bound limit is $3x$



(a) TCPDC using RED1.

(b) TCPDC using RED2.

(c) TCPVM using RED1.

(d) TCPVM using RED2.

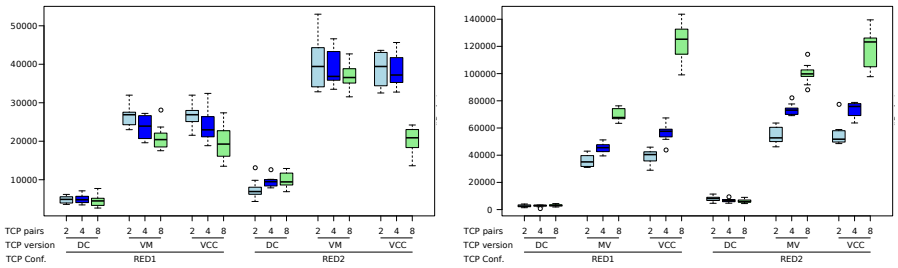
(e) VCC using RED1.

(f) VCC using RED2.

Fig. 4. Queue occupation on *Switch 2*.

RED1 configuration, increasing the marking interval. In this sense, the wide marking interval explains the variation observed with RED2 results.

Regarding the dropped packets originated by the introduction of background traffic, Figs. 7(a)-7(c) summarize results for RED1 configuration, while results for RED2 configuration are given by Figs. 7(d)-7(f). TCPDC with RED1 configuration has the values concentrated between 1900 and 4000 bytes, and smaller background traffic loads (2MB, 4MB, and 8MB) resulted on higher dropped packets. The loss in larger loads is less influenced by the stabilization of the congestion control window, while smaller background loads are more susceptible to aggressiveness in opening the window. TCPVM (Fig. 7(b)) significantly increased the amount of dropped packet on all background loads. Background loads which are greater than $4GB$, the values remained above 35000 bytes. RED1



(a) Dropped packets on the *Switch 1*. (b) Dropped packets on the *Switch 2*.

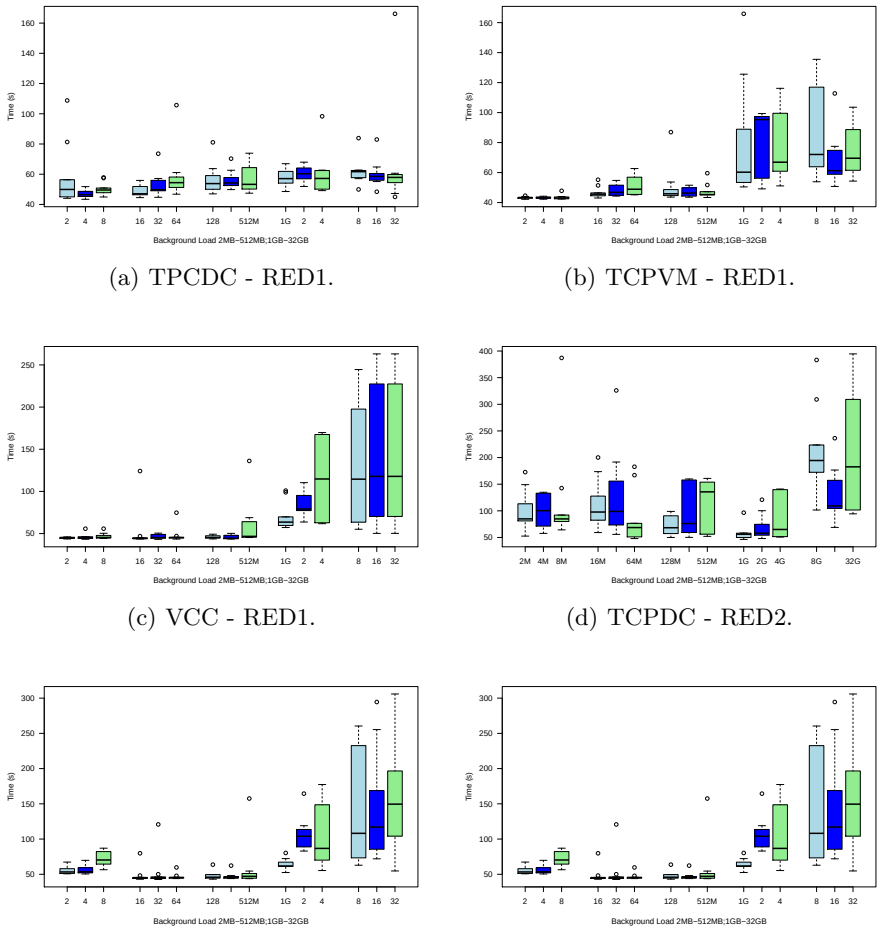
Fig. 5. Dropped packets on *Switches* 1 and 2.

configuration and the use of VCC (Fig. 7(c)) decreases the packet dropping occurrence on smaller background traffic (2MB to 512MB). On the other hand, it dramatically increases the packet dropping for background loads between 1GB at 32GB, when compared to TCPVM scenario. In turn, the dropped packets for TCPDC with RED2 configuration (Fig. 7(d)) oscillated following the increase in background traffic load, with an upper-bound of 10000 bytes, while for TCPVM (Fig. 7(e)) varies from 30000 to 70000 bytes. Using VCC enabled (Fig. 7(f)) the amount of dropped packets is decreased for small background loads.

The last set of data for Scenario 2 is the switches queue occupancy, summarized by Fig. 8. Initially, for RED1 configuration, TCPDC shows a gradual distribution of the queues in relation to the volume of background traffic (Fig. 8(a)), while the TCPVM scenario (Fig. 8(b)) the probability of queue occupancy increased 50% when compared to TCPDC. In addition, the VCC scenario with RED1 configuration (Fig. 8(c)) indicated the concurrent HMR execution time with background loads up to 256MB, VCC was able to keep the switches queue occupancy just over 5000 bytes, eventually outperforming the TCPDC scenario (for 8 and 16GB background traffic). Finally, results for RED2 configuration are summarized by Figs. 8(d), 8(e), and 8(f) for TCPDC, TCPVM, and VCC. This corroborates what has been previously concluded in Scenario 1; VCC is susceptible to the background traffic load and consequently impacts on the execution time of HMR.

5 Related Work

Related work comprises studies on explicit congestion notification, techniques for VCC, characterization and optimization of HMR traffic in DC. Initially, Sally Floyd [6] discusses the use of ECN in TCP. Simulations using New Reno and RED markers indicated the benefits of this feature in congestion control, avoiding losses by anticipating and predicting likely network saturation scenarios. Although disruptive, the implementation of RED on modern operating systems was recently largely adopted, motivating the present work. Specifically, Kuh-



(a) TPCDC - RED1.

(b) TCPVM - RED1.

(c) VCC - RED1.

(d) TCPDC - RED2.

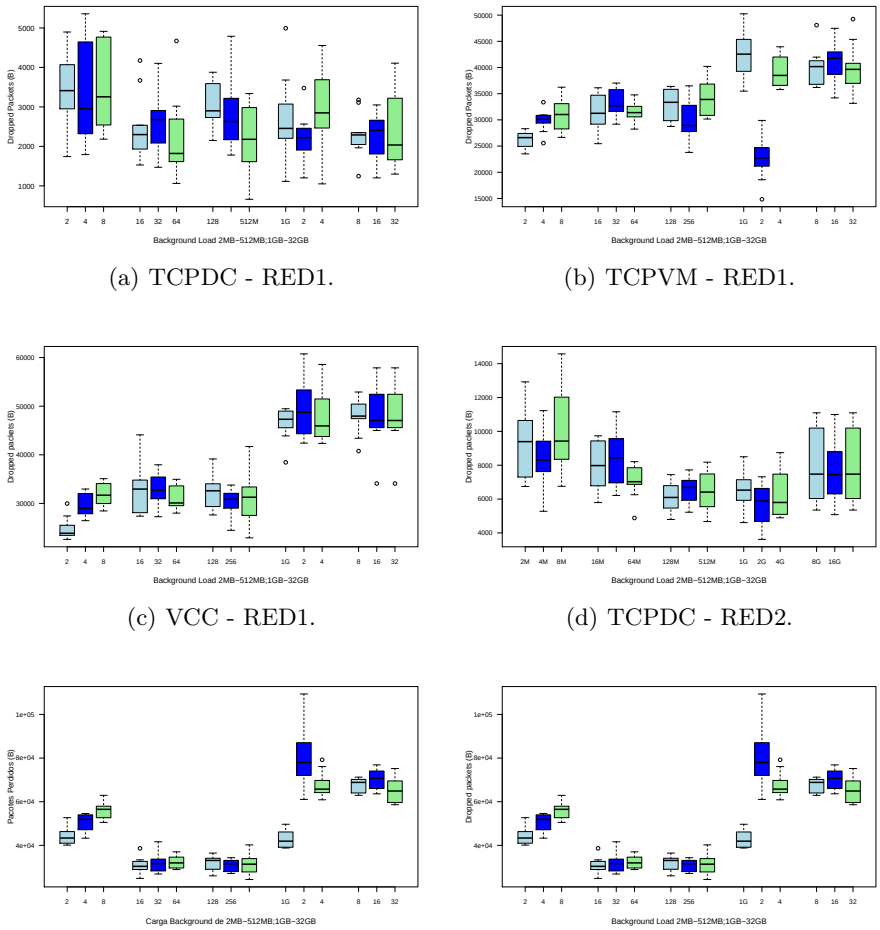
(e) TCPVM - RED2.

(f) VCC - RED2.

Fig. 6. HMR execution time with 4 pairs of background traffic.

lewind *et al.* [11] addresses the historical evolution of ECN in OSs, presenting a view of other ways to congestion prevention and mitigation. In summary, the work indicates the absence of a *de facto* solution to bypass the sharing disparity when multiple algorithms for congestion control are sharing bottlenecks. The present work is based on such assumption. Indeed, we envisage a multi-tenant DC scenario in which multiple users, each one with a private TCP congestion control algorithm, share the data center resources (processing and communication). For instance, the tenants virtual machines or containers can be configured with private TCP options, leading to an unfairness sharing of congested links.

Aware of the diversity of applications running on a cloud DC with outdated TCP/IP stacks, the specialized literature [5, 8] proposed flexibilization in



(a) TCPDC - RED1.

(b) TCPVM - RED1.

(c) VCC - RED1.

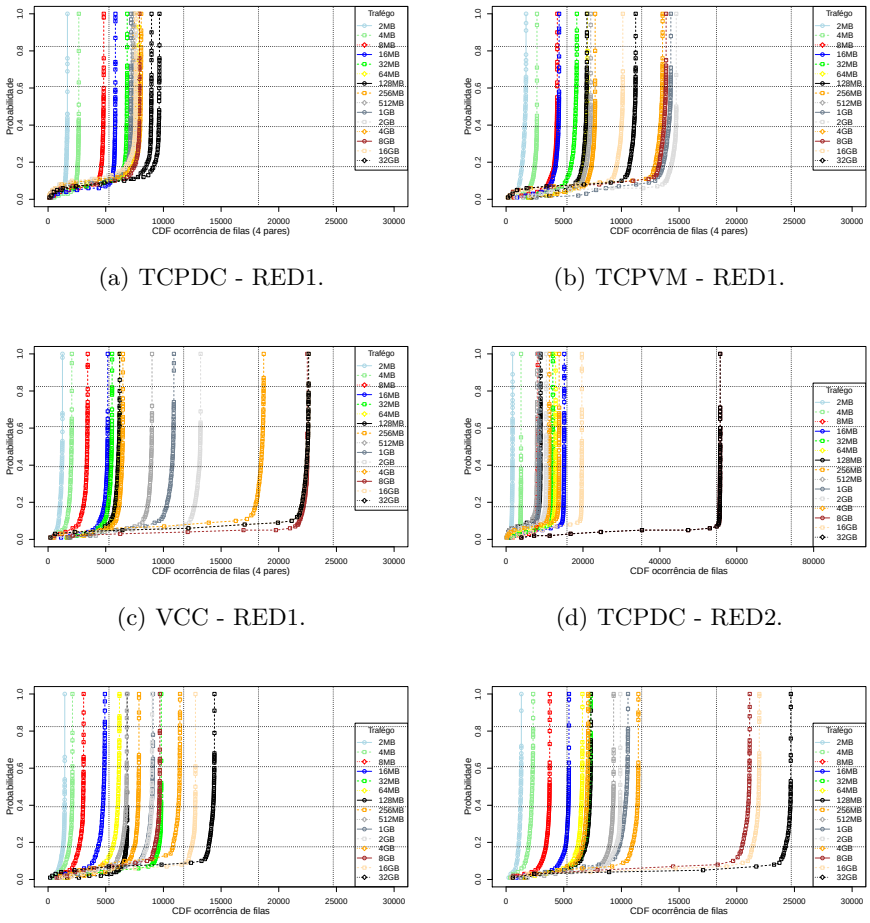
(d) TCPDC - RED2.

(e) TCPVM - RED2.

(f) VCC - RED2.

Fig. 7. Total dropped packets for Scenario 2.

configuration, defining the concept of VCC. This was a key motivator for our performance analysis using HMR applications. Alizadeh *et al.* [1] has identified communication patterns in DC that host HMR: query traffic, background traffic, competing flows of different sizes. Furthermore, Wu *et al.* [21] addresses the congestion traffic caused by incast traffic which occurs when multiple flows converge to the same receiver. We advanced the field by investigated the performance impact when running HMR, a real large-scale distributed application. Instead of applying synthetic loads, our experiments were based on traces of HMR execution performed on a real cluster.



(a) TCPDC - RED1.

(b) TCPVM - RED1.

(c) VCC - RED1.

(d) TCPDC - RED2.

(e) TCPVM - RED2.

(f) VCC - RED2.

Fig. 8. Switches queue occupancy for Scenario 2.

Regarding the management of IaaS DCs, Popa and colleagues [17] investigated the tough negotiation when establishing bandwidth allocation policies to ensure a proportionality of network usage, granting a minimum guarantee for the flow of the VM and at the same time to avoid idleness, inducing the high occupation of the network. In turn, Zahavi *et al.* [22] proposes a new abstraction of cloud services, called Link-as-a-Service (LaaS), with isolation between virtualized communication links. The client can choose to introduce in the link the congestion control algorithm that best meets the needs of their application, that is, the decision does not belong to the provider. Both works are examples of how VCC can be combined with bandwidth reservation techniques to provide Quality of Service (QoS) requirements. In our analysis we stressed the HMR application

with heavy competitive load causing the worst-case scenario for congestion control. A future work can combine bandwidth reservation techniques to soften the degradation of HMR performance when executed atop VCC-enabled DCs.

6 Final Considerations & Future Work

IaaS clouds allow tenants to install and configure their OS according to their hosted applications requirements. Such advent diffused the malleability of VMs, however, resulted in a heterogeneous scenario running on DC. Above all, several clients do not upgrade libraries, modules, and OSs due to technical constraints on legacy applications. Consequently, outdated versions of TCP algorithms for congestion control compete with updated versions in DCs. To mitigate disparity, virtualized congestion control was recently proposed.

The present work investigated the application of VCC in the execution of applications HMR, executed in VMs with non-optimized TCP. The experimental analysis was performed with traces of HMR execution on a dedicated cluster. Analyzing the results, was evidenced the virtualization is directly dependent on the configurations applied in the switches. Above all, experiments following the settings indicated in the literature (referred to as RED1 and RED2) resulted in an impact on the execution time of the HMR applications. The results advanced the TCP congestion control literature by empirically analysing the execution of a trace from a real communication-intensive application atop a VCC-based DC. As a continuity perspective, it is clear that VCC is a promising technology, however, as the results indicated, the definition of optimized RED and ECN configurations are essential and of course a continuum line.

References

1. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M.: Data center tcp (dctcp). SIGCOMM Comput. Commun. Rev. **40**(4), 63–74 (Aug 2010)
2. Alizadeh, M., Javanmard, A., Prabhakar, B.: Analysis of dctcp: Stability, convergence, and fairness. In: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems. pp. 73–84. SIGMETRICS '11, ACM (2011)
3. Chiu, D.M., Jain, R.: Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Comput. Net. ISDN Sys. **17**(1), 1–14 (1989)
4. Chowdhury, M., Zaharia, M., Ma, J., Jordan, M.I., Stoica, I.: Managing data transfers in computer clusters with orchestra. SIGCOMM Comput. Commun. Rev. **41**(4), 98–109 (Aug 2011)
5. Cronkite-Ratcliff, B., Bergman, A., Vargaftik, S., Ravi, M., McKeown, N., Abraham, I., Keslassy, I.: Virtualized congestion control. In: Proceedings of the 2016 ACM SIGCOMM Conf. pp. 230–243. SIGCOMM '16, ACM (2016)
6. Floyd, S.: Tcp and explicit congestion notification. SIGCOMM Comput. Commun. Rev. **24**(5), 8–23 (Oct 1994)
7. Ha, S., Rhee, I., Xu, L.: Cubic: A new tcp-friendly high-speed tcp variant. SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (Jul 2008)

8. He, K., Rozner, E., Agarwal, K., Gu, Y.J., Felter, W., Carter, J., Akella, A.: Ac/dc tcp: Virtual congestion control enforcement for datacenter networks. In: Proc. of the 2016 SIGCOMM Conference. pp. 244–257. SIGCOMM '16, ACM (2016)
9. Jacobson, V.: Congestion avoidance and control. SIGCOMM Comput. Commun. Rev. **18**(4), 314–329 (Aug 1988)
10. Judd, G.: Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation. pp. 145–157. NSDI'15, Berkeley, CA, USA (2015)
11. Kühlewind, M., Neuner, S., Trammell, B.: On the state of ecn and tcp options on the internet. In: Proceedings of the 14th International Conference on Passive and Active Measurement. pp. 135–144. PAM'13, Berlin, Heidelberg (2013)
12. Kumar, P., Dukkipati, N., Lewis, N., Cui, Y., Wang, Y., Li, C., Valancius, V., Adriaens, J., Gribble, S., Foster, N., Vahdat, A.: Picnic: Predictable virtualized nic. In: Proceedings of the ACM Special Interest Group on Data Communication. pp. 351–366. SIGCOMM '19, ACM (2019)
13. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: Rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. pp. 19:1–19:6. Hotnets-IX, ACM (2010)
14. Moro, V., Pillon, M.A., Miers, C., Koslovski, G.: Análise da virtualização do controle de congestionamento na execução de aplicações hadoop mapreduce. In: Simpósio de Sistemas Computacionais de Alto Desempenho - WSCAD (oct 2018)
15. Neves, M.V., De Rose, C.A.F., Katrinis, K.: Mremu: An emulation-based framework for datacenter network experimentation using realistic mapreduce traffic. In: Proc. of the 2015 IEEE 23rd Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. pp. 174–177. MASCOTS '15 (2015)
16. Pfaff, B., Pettit, J., Koponen, T., Jackson, E.J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M.: The design and implementation of open vswitch. In: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation. pp. 117–130. NSDI'15 (2015)
17. Popa, L., Kumar, G., Chowdhury, M., Krishnamurthy, A., Ratnasamy, S., Stoica, I.: Faircloud: Sharing the network in cloud computing. In: Proc. of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. pp. 187–198. SIGCOMM '12, ACM (2012)
18. Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the social network's (datacenter) network. SIGCOMM Comput. Commun. Rev. **45**(4), 123–137 (2015)
19. de Souza, F.R., Miers, C.C., Fiorese, A., Koslovski, G.P.: Qos-aware virtual infrastructures allocation on sdn-based clouds. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 120–129. CCGrid '17, IEEE Press, Piscataway, NJ, USA (2017)
20. Vicat-Blanc Primet, P., Anhalt, F., Koslovski, G.: Exploring the virtual infrastructure service concept in Grid'5000. In: 20th ITC Specialist Seminar on Network Virtualization. Hoi An, Vietnam (May 2009)
21. Wu, H., Ju, J., Lu, G., Guo, C., Xiong, Y., Zhang, Y.: Tuning ecn for data center networks. In: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. pp. 25–36. CoNEXT '12, ACM (2012)
22. Zahavi, E., Shpiner, A., Rottenstreich, O., Kolodny, A., Keslassy, I.: Links as a service (laas): Guaranteed tenant isolation in the shared cloud. In: Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems. pp. 87–98. ANCS '16, ACM (2016)