Revisiting Multipath Congestion Control for Virtualized Cloud Environments

Chi Xu[*†], Jia Zhao[†], Jiangchuan Liu[†], and Fei Chen[‡]

[*]College of Natural Resources and Environment, South China Agricultural University, China
[†]School of Computing Science, Simon Fraser University, Canada
[‡]College of Computer Science and Technology, Qingdao University, China

***Abstract*—Virtualized datacenters are often designed from scratch with multiple, redundant paths. Yet the majority of the existing congestion control schemes for virtual machines or containers are variants based on single-path TCP design. Lacking the flexibility of leveraging underlying paths, these schemes cannot further improve the utilization of datacenter networks, or mitigate hotspot links. In this paper, we examine the performance of multipath congestion control design on typical hypervisor and container virtualization platforms. We observe that, the involvement of virtual switch, together with the multi-tenancy nature on these platforms, poses new challenges when handling multipath traffic. Through realworld experiments with production-grade applications, we further reveal that, while multipath congestion control increases per-connection throughput and achieves better traffic balancing, it experiences performance degradation when the number of connections is abruptly increased or there exist path-sharing subflows. These issues are due to the *enforced QoS policies* and *interface mapping schemes* applied by virtual switch. To this end, we present vMCC, a practical solution which incorporates explicit congestion notification (ECN) support on virtual switches and ECN-aware multipath congestion control algorithms. We show by comprehensive evaluations that vMCC improves throughput, round trip time, fairness, and energy efficiency for cloud datacenter traffic, and subsequently benefits typical cloud workloads.**

***Index Terms*—computer networks, platform virtualization, network interfaces, data communication**

I. Introduction

Virtualization technologies allow datacenters to be managed with reduced costs through server consolidation and elastic resource provisioning. The everlasting effort devoted by giant companies such as Amazon and Microsoft has actuated service migration towards fully virtualized environments. Major cloud providers also host their service inside virtualized datacenters around the world. According to Gartner's report [1], about 80% of x86 server workloads are virtualized nowadays, and virtualization technologies are becoming more lightweight and adaptive to support different kinds of cloud workloads.

As for the network infrastructure in virtualized datacenters, redundant paths are commonly provisioned to ensure quality of service (QoS) and scalability. However, even if endpoints are connected by multiple paths, without transport layer support, communications are still restricted to a single path per connection. The majority of the congestion control schemes for virtual machines (VMs) and containers are based on single-path TCP design and does not natively support multipathing. This can often lead to congestion on specific paths and under-utilization on other ones [2].

Recently, multipath TCP (MPTCP) [3] is introduced for end machines to leverage path diversity in datacenters. However, the performance and challenges of employing multipath TCP in virtualized environments have yet been fully investigated. In contrast to traditional bare-metal servers and physical switches, the networking constitution and the multi-tenancy nature in virtualized environments make key differences.

First, these platforms introduce virtual switches to connect the virtual network interface cards (NICs) of VMs/containers and establish connections to the outer network. The implementation of a virtual switch is drastically different from its hardware counterpart. Virtual switches use multiple levels of caches and flow caching instead of ternary content-addressable memory for packet classification and forwarding. A typical example is Open vSwitch [4]. The data forwarding path, wiring state, and QoS settings are all handled by software, providing great flexibility for datacenter management.

Second, the multi-tenancy nature and elastic service provisioning are other key factors in virtualized environments. The multi-tenancy nature refers to the fact that the virtual machines or containers hosted on the same physical server may belong to different tenants. Such autonomy leads to different TCP congestion control behaviors. Commonly, to ensure traffic fairness and provide bandwidth guarantees, administrators enforce *non-work conserving QoS policy* for each VM/container. This leads to limited buffer space for each virtual NIC, causing buffer pressure and queueing delays. As bursty fan-in communication pattern is often seen in datacenter workloads, the incast problem can bring even worse impacts.

Third, the mapping scheme between virtual NIC and physical NIC can often be changed with VM/container migrations and other traffic management actions. Such mapping schemes are invisible inside a VM/container itself, and may cause multipath traffic to inexplicitly share the same path. The overall network performance is greatly affected in such circumstances.

In this work, we for the first time examine the performance and practical concerns of multipath TCP deployment in typical virtualized environments. Our measurement reveals a series of distinct observations that are not seen on bare-metal servers. To

978-1-7281-6887-6/20/$31.00 © 2020 IEEE

begin with, we confirmed the superiority of multipath congestion control deployment with comprehensive experiments. The experiments are conducted with production-grade applications across different network configurations.

In particular, multipath TCP traffic exhibits throughput increase and background traffic adaptivity in all these experiments. Typically, multipath TCP connections increase the throughput by $22.7 - 31.4\%$ when compared with single-path TCP connections. When there exists background traffic, the percentage goes up to 42.1%. We then show two potential caveats of deploying multipath TCP on virtual machines and containers: the first is the handling of bursty fan-in traffic on virtual switches: the traffic from multiple co-located VMs/containers overwhelms the buffer space associated with the virtual NICs, causing performance degradations. Furthermore, We uncover a subflow path-sharing bottleneck. This issue is caused by the mapping scheme between virtual NIC and physical NIC. For example, multiple virtual NICs of one VM/container are mapped onto the same physical NIC, and thus the multiple subflows of one multipath TCP connection share the same path. In this scenario, the subflows of the same connection compete with each other. And the overall transmission performance is also affected by such competition.

Based on our in-depth analysis, we propose vMCC[1], a comprehensive solution which comprises enhanced congestion control algorithms and virtual switch plug-ins. vMCC leverages explicit congestion notification (ECN) to enhance multipath TCP congestion control. The proposed solution ensures switch buffer occupancy to be persistently at a low level, and it also maintains high throughput for long flows. We then theoretically prove the fairness between vMCC traffic and conventional single-path TCP traffic. Taking a step further, we also propose to leverage similar ECN and round trip time (RTT) patterns to infer and terminate suspicious path-sharing subflows. To provide a self-contained solution, we developed loadable kernel modules for hypervisors and container engines. It is noted that the vMCC implementation is not confined to any specific virtualization platform and is completely transparent to upper-layer applications. Our extensive evaluations show that vMCC achieves up to 4.5x improvement on round trip time without impeding network throughput. The goodput of long flows is increased by up to 11.8% and the flow completion time of short flows is reduced by up to 22.1% in our experiments. Meanwhile, vMCC shows significant improvements in flow fairness and energy efficiency. vMCC also effectively detects and terminates path-sharing subflows, which is validated with high F-score values.

The rest of this paper is organized as follows: In Section II, we present related works, which cover recently proposed congestion control schemes for virtualized datacenter and recent multipath congestion control research. Section III introduces our motivations and detailed measurement configurations. In Section IV, we present and analyze the performance and challenges of multipath TCP deployments in virtualized environments. We then propose congestion control enhancements and discuss the implementation details in Section V. The evaluation results are shown in Section VI, and Section VII concludes this paper.

[1]Virtualized Multipath Congestion Control

II. Related Work

There have been several works on examining and improving multipath transport performance. To begin with, Wischik *et al*. [5] proposed a TCP-friendly link increased algorithm (Lia) for multipath congestion control. Raiciu *et al*. [2] evaluated the performance of multipath TCP deployment in datacenters and showed that multipath TCP can greatly improve datacenter network utilization. Khalili *et al*. [6] further investigated the optimized resource allocation among a large number of simultaneous multipath TCP connections. Later, Cao *et al*. [7] proposed a delay-based multipath congestion control algorithm. Peng *et al*. [8] designed a multipath congestion control scheme that balances algorithmic responsiveness and TCP-friendliness. Agache *et al.* used multipath TCP in their solution GRIN [9] to improve datacenter network capacity.

Following these pioneer studies, various multipath TCP designs have been proposed for latency-sensitive flows in datacenters. Cao *et al*. [10] proposed a multipath congestion control scheme (XMP) to achieve high performance for both long and short flows in datacenters. Kheirkhah *et al*. [11] proposed a random packet scheduler to exploit good-quality paths for short multipath flows and reduce their completion time. Chen *et al*. [12] further designed a fast loss recovery approach for multipath transmission. Lately, AMP [13] discussed the coexistence of DCTCP traffic and multipath TCP traffic. DCMPTCP [14], [15] also sought to solve similar problems with switch-side support.

On the other hand, there is also much effort on tuning congestion control performance for virtualized datacenters. To our best knowledge, these works mainly target single-path TCP performance. For example, Wu *et al*. [16] proposed a proactive congestion control approach for TCP incast problem. Cheng *et al*. [17] improved the calculation of RTO and RTT in TCP to filter out the negative impact posed by hypervisors' scheduling delays. He *et al*. [18] proposed AC/DC TCP that employs the DCTCP algorithm in the virtual switch to enforce per-flow congestion control. Sharing a similar idea, Cronkite-Ratcliff *et al*. [19] proposed vCC to enable an ECN-based congestion control algorithm with hypervisor support.

Our work differs from the previous multipath transport designs [2], [5]–[15] in that we focus on the performance of multipath transport in virtualized environments. Employing such transport in virtualized environments brings unique challenges. Therefore, we first thoroughly evaluate the performance of multipath TCP via realworld measurements on typical virtualization platforms. This is performed with both benchmark tools and production-grade cloud applications. We also vary the network configurations to fully examine the results. Based on the findings, we design a practical and efficient multipath congestion control scheme to deal with the caveats. Our work also differs from the congestion control

TABLE I: Platforms and Tools

Platform/Tool	Description	Reference
KVM	Kernel-based virtual machine	https://www.linux-kvm.org/page/Main_Page
Docker	a popular container virtualization engine	https://www.docker.com
Open vSwitch	a production quality, multilayer virtual switch	https://www.openvswitch.org/
Intel DPDK	Data Plane Development Kit to accelerate packet processing workloads	https://www.dpdk.org/
virtio	paravirtualized drivers for KVM/Linux	https://www.linux-kvm.org/page/virtio
QEMU	a generic and open source machine emulator and virtualizer	https://qemu.org

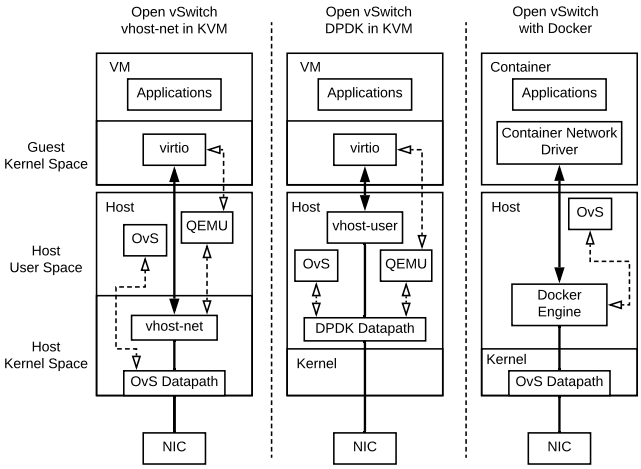


Fig. 1: Network I/O subsystem breakdown in different virtualization environments. From left to right: Open vSwitch vhost-net in KVM, Open vSwitch with DPDK in KVM, and Open vSwitch with Docker container engine.

design for virtualized cloud datacenters [16]–[19], since we have to tackle with the performance degradation of multiple subflows and the impact of path diversity.

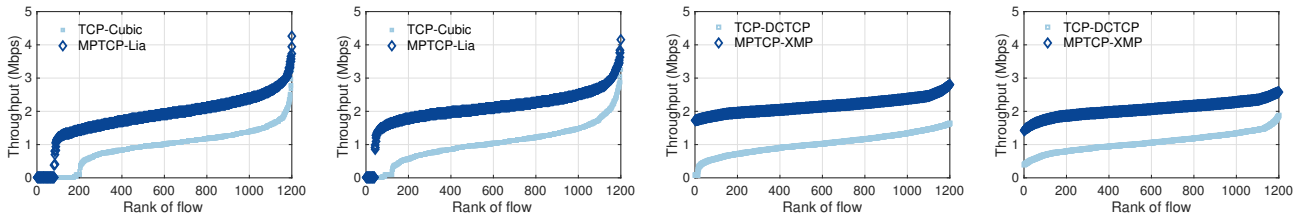
III. Preliminaries

Striving to understand multipath TCP performance in virtualized environments, we have conducted extensive experiments with varied hardware and software configurations. We first briefly introduce the virtualized environments and network configurations of these experiments.

We measure multipath TCP performance on both hypervisor and container virtualization platforms. For hypervisor virtualization platform, we mainly work with KVM, also known as Kernel-based Virtual Machine. It is a full virtualization solution for Linux on x86 hardware containing virtualization extensions. In the KVM environment, multiple virtual machines can run with unmodified images. Each virtual machine has its private virtualized hardware, including network interface cards, disk, graphics adapter. For container virtualization platform, we work with Docker, a popular production-grade container virtualization engine. Docker containers isolate software from the surroundings and help reduce conflicts when tenants running different software on the same cloud infrastructure. As for the selection of virtual switch, Open vSwitch (OvS) is a software switch tailored for virtualized systems. OvS features advanced flow caching, fast packet classification, and compatibility with the OpenFlow protocol. In KVM environments, we built OvS from the source with the native forwarding engine vhost-net and also tested the polling mode forwarding engine with Intel DPDK and vhost-user drivers. Across all setups, we configured virtio as the datapath between host and KVM virtual machines. We performed similar procedures for the Docker engines. And the detailed breakdown of the network I/O subsystem can be referred to in Fig. 1. For the ease of reference, we also list the platforms and tools in Table I. Another option for setting up a virtual switch is using the default Linux bridge (LB). Linux bridge is an in-kernel L2 switch shipped with mainline Linux distributions. It features fast, reliable packet transmissions and easy setup. Linux bridge is often used as an out-of-box-solution for setting up networks for VMs and containers.

The hardware configurations are listed as follows. The rack servers employed in our experiments are 10 Dell PowerEdge R430 servers. Each one is equipped with two Intel Xeon E5-2630 v3 2.4GHz CPU nodes. Supported by Intel hyper-threading technology, a CPU node has 8 physical cores and 16 logical CPUs. The rack server has 256GB RAM with a typical non-uniform memory access (NUMA) architecture. It also has two Intel Ethernet Controller 10 Gigabit X540-AT2 NICs, supporting Dual 10 Gbps networking connections. We have two types of Top-of-Rack (ToR) switches, including Netgear ProSAFE Smart Managed Switch XS712T and Dell Networking N4032 switch. The Dell Networking N4032 switch supports advanced switching functions, e.g., random early detection (RED) and ECN marking. These functions enable that sending sources are quickly notified of the queue overshoot before excessive packet loss events happen.

The operating system installed in both hosts and guest VMs is Ubuntu 16.04.2 LTS. We then installed multipath TCP release version 0.92. The corresponding Linux kernel version is 4.4.83. For the KVM environments, we installed QEMU version 4.0.0. The Docker version is 18.09. The virtual switches installed at the hypervisor or host are Open vSwitch and Linux bridging utilities. We compiled Open vSwitch 2.10.0 and DPDK 18.08 from its source tree. To quantify the networking performance, our experiments are conducted with production-grade applications such as Linux network file system (NFS), Apache webserver, and Hadoop. All these applications transfer large-/medium-size file chunk. We also tested the platforms with Memcached for generating short messages. We then used common profiling tools integrated with mainline Linux distribution for benchmarking, which include `ping`, and `netstat`. For an in-depth analysis on RTT variations and retransmissions, we used tools such as `tcpdump/tcptrace` to extract the information.



(a) OvS vhost-net in KVM (b) OvS DPDK in KVM (c) Linux bridge in KVM (d) OvS with Docker containers

Fig. 2: A comparison of throughput increase with different switch settings.

IV. Measurement Results and Analysis

The measurement results and analysis are presented in this section. To begin with, our experiments quantified the improvements brought by enabling multipath transmissions in typical virtualized environments.

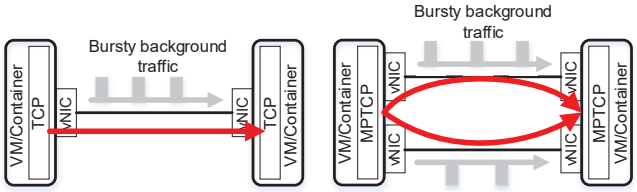
A. Throughput Increase

One of the compelling arguments of using multipath transmission is to achieve better aggregated throughput for each connection. This is because exploring more paths and load balancing them properly will reduce the number of underutilized links. We used general traffic sending experiments to quantify such throughput improvements in virtualized environments. In this experiment, we configured 12 virtual machines to serve as NFS servers on the target physical machine, and then set up 100 connections for each virtual machine to transfer file chunks. The number of VMs and connections is set to avoid exceeding the resource capacity [20]. Each of the virtual machines is configured with two virtual NICs, and then leverages two disjoint paths. To mimic the non-work conserving policy applied in virtualized datacenters, we set the bandwidth guarantee for each virtual machine to be 600 Mbps, which is divided among the available virtual NICs.

We show the rank of throughput[2] of these 12×100 flows in Fig. 2. We first compared the throughput of TCP Cubic and MPTCP Lia connections since Cubic and Lia are the default congestion control schemes for TCP and multipath TCP traffic, respectively. The experiments are conducted in KVM environments with two different virtual switch settings. In particular, Fig. 2a shows the result with Open vSwitch and vhost-net drivers. In this experiment, only 1032 TCP Cubic connections are completed, with the average throughput being 1.45 Mbps per connection. As for MPTCP Lia, 1121 connections completed the transmission. The average throughput is 2.08 Mbps for each connection. As shown in Fig. 2b, we also measured the throughput with the polling mode driver DPDK, and the results remain quite similar.

Note that the transmission in the previous set of experiments does not yet leverage ECN feedback. As a next step, we also quantified the throughput increase with ECN-aware congestion control schemes, e.g., by comparing DCTCP [21] and XMP [10] performance in virtualized environments. Fig. 2c presents the experimental results with the default Linux bridge

[2]The throughput of the parallel connections is sorted ascendingly.



(a) Single path (b) Multipath

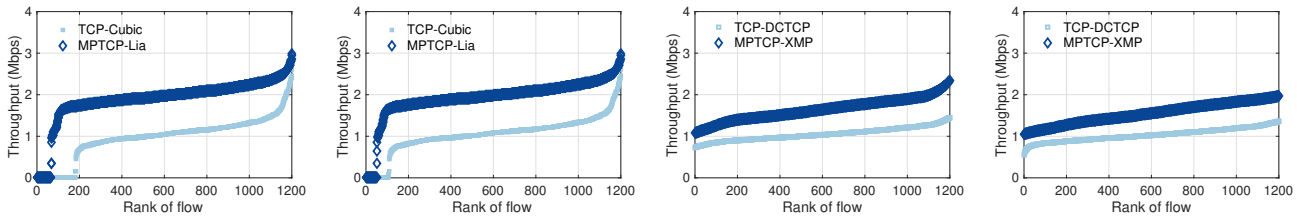
Fig. 3: TCP and multipath TCP flows share paths with bursty background UDP traffic.

and Fig. 2d shows the results on a Docker platform. A comparison between Fig. 2a and Fig. 2c indicates that, in all the experiments without ECN awareness, the number of incomplete connections increases due to the congestion at the beginning stage of setting up connections. With multiple connection requests issued at nearly the same time, a significant number of packets were dropped and a small number of connections were never established. In contract, with an ECN-aware design, DCTCP and XMP have improved the aggregated throughput and nearly all connections were established.

In all the cases, the observed throughput increase of multipath TCP connections is from that the multipath traffic is capable of utilizing both of the two virtual NICs to send traffic. Due to the TCP friendly design and RTT based traffic shifting, multipath TCP connections do not consume all the available bandwidth. Typically, MPTCP Lia and XMP have increased the throughput by $22.7 - 31.4\%$ when compared with the corresponding single-path case. Moreover, flow fairness can also be observed by comparing the deviations of the throughput. In all these cases, DCTCP and XMP show better fairness among these 1200 connections with reduced throughput deviations. For example, in Fig. 2c, the standard deviations of DCTCP per-connection throughput and XMP per-connection throughput are 0.44 Mbps and 0.22 Mbps, respectively.

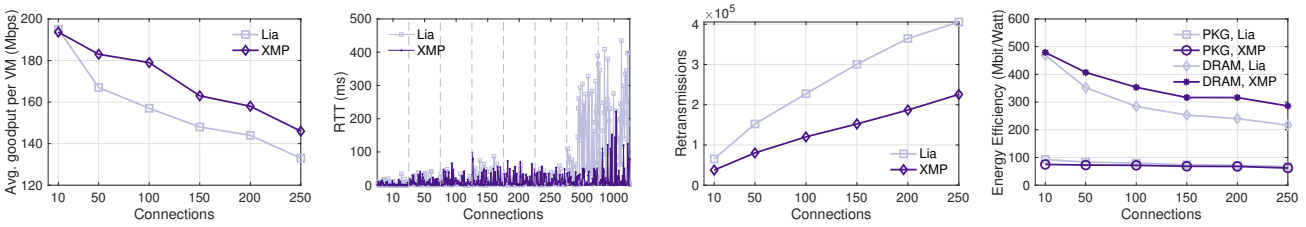
B. Traffic Shift

Another benefit brought by multipath transmission is the capability of traffic shifting between paths, since multipath TCP variants tend to send traffic on a less congested path. Multipath TCP variants such as Lia detect good paths with low delay in realtime, and shift traffic to such paths. Further, they also adapt to different topologies (e.g. hierarchical) and traffic characteristics (e.g. bursts). To quantify the improvements brought by the multipath TCP design in virtualized environments, we compared the performance of single-path



(a) OvS vhost-net in KVM (b) OvS DPDK in KVM (c) Linux bridge in KVM (d) OvS with Docker Containers

Fig. 4: Throughput comparison when running with bursty background traffic.



(a) Goodput (b) Round trip time (c) Total retransmission (d) Energy efficiency

Fig. 5: Impact of incast traffic in virtualized environments.

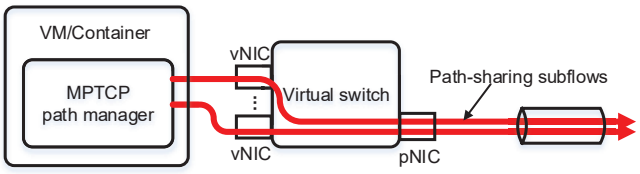


Fig. 6: Path-sharing subflows of a multipath TCP flow.

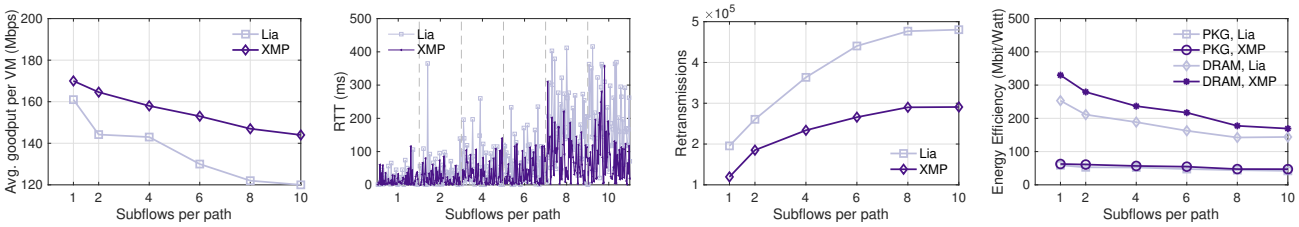
TCP and multipath TCP when VMs/containers experience the interference of bursty background traffic. The experiment illustration is shown in Fig. 3. Each of the VMs/containers first uses single-path TCP for data transmission, and the congestion control schemes are set to be the same as in the previous experiment, e.g., Cubic and DCTCP. The transmission lasted for 60 seconds. During the experiments, we initiated background UDP traffic with the sending rate set at 1 Gbps on one path every 10 seconds. The UDP traffic lasted for 5 seconds. We then compared multipath TCP performance when running with the same background traffic. Herein, the congestion control schemes are set to be Lia and XMP. Similar to the previous experiment, Fig. 4 presents the throughput of all these 1200 flows in ascending order. In this experiment, multipath TCP variants achieved up to 42.1% and 21.3% throughput increase when running with Open vSwitch and Linux bridge in KVM, respectively. And the percentage of throughput increase is 18.9% on a Docker platform. All the comparisons are made with the single-path TCP connections.

C. Incast Traffic

Despite the enhanced throughput and traffic adaptability brought by multipath TCP, there also exist caveats when deploying the protocols in virtualized environments. We first consider the network sharing issues of multipath TCP traffic. When a VM/container demands a specific amount of bandwidth, the administrator needs to have mechanisms to achieve bandwidth allocation and guarantees [22]. Typically in virtualized datacenter environments, providing bandwidth guarantees means that each VM shares a minimum guaranteed bandwidth to send and receive traffic whenever needed. This is crucial for achieving predictable application performance, especially for data-intensive applications whose completion time mainly relies on the available network bandwidth.

For example, hypervisors such as KVM, Xen, and VMware ESX, as well as the Docker Engine, have bandwidth capping (or rate limiting) mechanisms that enforce the desired transmission rate for each virtual NIC associated with a virtual machine. More recent versions of hypervisors can also enforce receiving bandwidth cap per virtual NIC. The disadvantage of bandwidth capping is the potential under-utilization of link bandwidth. Using a more flexible traffic shaper such as Linux Hierarchical Token Bucket (HTB) can allow available bandwidth to be distributed to VMs with extra traffic demands. We have also seen recent efforts on enabling bandwidth guarantee, work-conserving, and low latency simultaneously [22]–[24] in virtualized cloud datacenters.

Despite the different schemes applied by datacenter operators, the subdivided traffic shaping policies require to maintain buffers on virtual switches and use tail drop mechanisms for traffic management. This potentially leads to a traffic incast problem on virtual switch. We conducted a stress test with an increased number of parallel connections on the 12 virtual machines. Fig. 5 presents the performance when the virtual machines are experiencing bursty fan-in traffic, e.g., object search queries and replies. Fig. 5a shows the average goodput of each VM. With both MPTCP Lia and XMP congestion control schemes, the goodput drops when the number of parallel connections increases. And the RTT and retransmissions are both increased, which are shown in Fig. 5b and Fig. 5c, respectively. We can see that, with ECN awareness, XMP performs better than MPTCP Lia. Fig. 5d shows the energy efficiency of the underlying server when experiencing such incast traffic.



(a) Goodput (b) Round trip time (c) Total retransmission (d) Energy efficiency

Fig. 7: Impact of path-sharing subflows.

The energy efficiency metric is defined as: how much data can be transmitted with the unit watt power consumption. The indicators in Fig. 5d are calculated via dividing the throughput by the instant power consumption of processor package (PKG) or memory (DRAM) [25], respectively. In this scenario, the energy efficiency indicators also decrease by up to 54.1%.

D. Path-sharing Subflows

In virtualized environments, multiple subflows of one multipath TCP connection can actually share the same path even before leaving the end host [26]. For example, multiple virtual NICs can be mapped onto the same physical NIC with the traffic management of virtual switches, as shown in Fig. 6. In this case, the traffic goes out of the same physical interface and shares the same datapath. However, allowing multiple subflows to share the same path is not ideal in any situation. The competition between subflows will lead to performance degradations and cause even severer congestions.

To provide an intuition, Fig. 7 shows the performance when multiple subflows of Hadoop shuffling traffic share the same path. In particular, Fig. 7a presents the goodput comparison when we increased the number of subflow on one path. The experiment is conducted when we benchmarked 12 Hadoop workers with the Terasort job sorting 1TB randomly generated data. We can see a monotonically decreasing trend in the results: as the number of subflows per path increases, the average goodput of MPTCP Lia connections eventually dropped from 160.2 Mbps to 121.2 Mbps. Fig. 7b shows the corresponding RTT measurements. With the competition on the same path, we can see the increased instability of the RTT with both MPTCP Lia and XMP. The total count of TCP retransmission, shown in Fig. 7c, is significantly increased as well. And Fig. 7d shows the decreasing trend of the energy efficiency indicators. Similar negative impacts can also be observed in short flows during our extensive experiments. Since short flows usually have strict completion deadlines ranging from tens of milliseconds to hundreds of milliseconds, it can be inferred that the related application performance will also be greatly damaged if the subflows share the same path.

E. Summary

We now summarize the measurement results and the opportunities for further enhancing multipath TCP performance in virtualized cloud environments:

- Multipath TCP exhibits throughput increase and background traffic adaptivity in all the experiments. Multipath TCP connections increase the throughput by 22.7-31.4% when compared with single-path TCP connections. When there exists background traffic, the percentage goes up to 42.1% in our experiments.
- Multipath TCP performance significantly decreases when there exists incast traffic. Despite enabling ECN marking on physical switches, the typical bandwidth regulations with tail drop mechanisms on virtual switches also cause such performance impairment.
- Due to the virtual NIC mapping scheme, multiple subflows of a multipath TCP connection could share the same path before leaving the physical host. Unfortunately, such path-sharing subflows not only cause goodput dropping, but also increase the values of RTT, retransmission, and instant power consumption.

V. vMCC : A COMPREHENSIVE SOLUTION FOR VIRTUALIZED ENVIRONMENTS

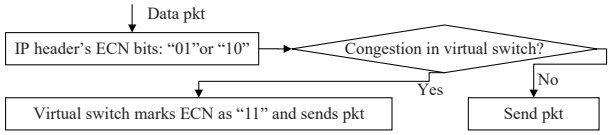
Based on the observations, we design vMCC to enhance multipath TCP performance in virtualized environments. The design consists of three parts: 1) enabling advanced ECN marking on virtual switches, 2) enabling ECN support for multipath congestion control and maintaining fairness to other congestion control schemes, and 3) ECN-aware path-sharing subflow termination.

A. Enabling advanced ECN marking on virtual switches

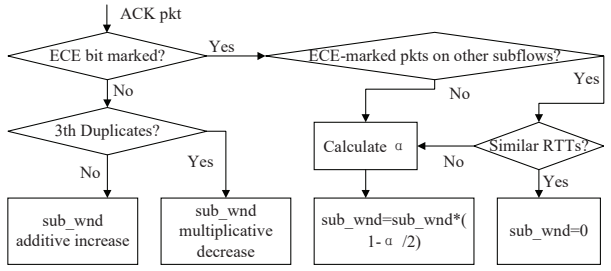
To start with, it is required to enable ECN marking on both physical and virtual switches. In fact, many commodity physical switches have already supported RED/ECN marking. Datacenter operators can simply enable such schemes with updated software configurations. On the other hand, we found that little work has been done on virtual switches.

The general ECN marking scheme works as follows, when the queue of a specific switch port piles up, the switch will set the congestion experienced (CE) bit in the IP header of a packet. The standard ECN marking threshold is based on a single queue model. Consider several synchronized flows with identical round trip times sharing the only queue of a switch port, to fully utilize the link bandwidth while achieving low latency, the ECN marking threshold T should be set as follows:

$$T = C \times RTT \times \lambda \tag{1}$$



(a) ECN marking on virtual switch



(b) ECN support for multipath congestion control

Fig. 8: vMCC design.

where RTT is average round trip time, C is link capacity, and λ is an empirical parameter closely related to congestion control algorithms. In virtualized datacenter networks, round trip time is relatively stable and can be measured through passive monitoring to compute the ECN marking threshold.

It is noted that nowadays commodity physical switches support multiple classes of service queues per port. Current virtualized datacenter management practice is to leverage queues to segregate traffic from different services (based on the urgency) and enforce weighted fair sharing among different queues. And we expect similar behaviors of virtual switches to cooperate with the operations of physical switches. As such, we enable multiple service queues on virtual switch ports and then adjust the ECN marking parameter for each service queue with a generalized processor sharing scheduler. This design adopts a similar idea to that presented in [27] for achieving high throughput, low latency, and weighted fair sharing. Then, to provide an intuition, the ECN marking scheme on a virtual switch is shown in Fig. 8a. We have currently implemented the workflow as a Linux queueing discipline (`qdisc`) kernel module. This helps eliminate the overhead of frequent data copy and context switch between user and kernel space.

B. ECN support and maintaining fairness

The main processes of our ECN-aware multipath congestion control are shown as the flow chart in Fig. 8b. For any ACK packet with ECN-Echo (ECE) bit unmarked, vMCC comes to congestion avoid phase and executes the multipath congestion control schemes similar to Lia. This design choice comes from that often there exist legacy virtual machines or containers running inefficient, out-dated, or misconfigured TCP/IP stacks [18]. And it is required to maintain flow fairness between tenants, especially when running with work-conserving traffic management policies. As shown in Algorithm 1, this algorithm couples the subflow window increase in order to maintain both efficiency and fairness. If vMCC receives ECE-marked ACK packets, it transfers to a DCTCP-like window decrease algorithm. Each subflow maintains the ECN-based congestion

Algorithm 1: Subflow Window Evolution

```
in_ACK_event():
for each ACK do
    /* extract TCP header ECE bit from subflow socket */;
    if TCP_flags_ECE=0 then
        /* increase subflow window w_r with Lia */;
        w_r ← w_r + min( max_{k∈s} w_k/RTT_k^2 / (Σ_{k∈s} w_k/RTT_k)^2 , 1/w_r );
    else
        update ECE count for this flow C_E = C_E + 1 ;
        /* counting threshold γ used to initiate the detection*/;
        if C_E ≥ γ && Path-sharing detection not performed
        then
            Calculate the referenced RTT R_RTT;
            Set the referenced ECE count to be R_E = C_E;
            Trigger ps_subflow_detection();
        else
            update α_r;
            /* execute DCTCP-like window decrease */;
            w_r = (1 − 1/2 α_r) w_r;
            update parameter α_r = (1 − g) · α_r + g · F;
        end
    end
end
```

Algorithm 2: Path-sharing Detection and Termination

```
ps_subflow_detection():
//Detect if subflows are on the same path;
if Number of subflows > 1 then
    Keep track of the referenced ECE count and referenced
    RTT in a hashtable;
    for Each pair of established subflow s_1 and s_2 do
        if |R_E(s_1) − R_E(s_2)| < θ &&
        |R_RTT(s_1) − R_RTT(s_2)| <
        δ × max{R_RTT(s_1), R_RTT(s_2)} then
            tcp_close(s_2);
        else
            Clear the references if no need for further
            detection and termination;
        end
    end
end
```

state α and the marked packet fraction F. The fraction F is updated after the last window of data is sent. Next, we will show that the congestion control algorithm of vMCC is fair to both conventional TCP congestion control and ECN-enabled TCP congestion control. Note that these control schemes may be adopted by legacy virtual machines or containers, and the fairness is ensured by the following theorem.

Theorem 1. *The congestion control algorithm of vMCC satisfies the fairness design goal suggested by the RFC 6356, i.e. the aggregated throughput of all the subflows of a vMCC connection at the stable state is no more than the throughput that it can achieve on the best path if it was a conventional TCP (e.g., NewReno or its variants) connection; it is also fair to ECN-enabled TCP connections (e.g., DCTCP).*

Proof. Consider the network model that is similar to [6]. Let L be the set of links in the network and $l \in L$ be of finite

capacity c_l. Let S be the set of MPTCP connections in the network. Let each connection $s \in S$ represent the set of available paths between its source end and destination end. Let $r \in s$ be a path that consists of multiple links. If a link l is on the path r, then we denote $l \in r$. For each path r, $RTT_r(t)$ and $w_r(t)$ denote the round trip time and congestion window, respectively, and $x_r(t) = w_r(t)/RTT_r(t)$ represents the instantaneous throughput at time t. For each connection $s \in S$, the vector $\mathbf{x}_s(t) = (x_r(t), r \in s)$.

$[R^l_{sr}]$ denotes the routing matrix where $R^l_{sr} = 1$ if link l is on the path $r \in s$ and 0 otherwise. Let $y_l = \sum_{s \in S} \sum_{r \in s} R^l_{sr} x_r$ be the aggregate traffic on link l. Let $p_l(y_l)$ be the packet loss probability at link l, which is an increasing function of y_l with the constraint $y_l \leq c_l$. The packet loss probability on path r is $\lambda_r = 1 - \prod_{l \in r}(1 - p_l) \approx \sum_{l \in r} p_l$. For simplicity, we omit the time t in the functions w_r, RTT_r, x_r and $\mathbf{x}_s$.

Instantaneous throughput increase and decrease by using vMCC can be modelled with the following differential equation:

$$\frac{dx_r}{dt} = \frac{\psi_r x_r^2}{RTT_r^2 (\sum_{k \in s} x_k)^2} - \frac{1}{2}\lambda_r x_r^2 - (1 - \frac{\alpha}{2})x_r^2 \qquad (2)$$

where $\alpha_{new} = (1 - g) \cdot \alpha_{previous} + g \cdot F$, and the smoothing factor g and the ECE-marked proportion F are defined in [21].

At the equilibrium, $\mathbf{x}^*_s = (x^*_k, k \in s)$, and $\frac{dx_r}{dt} = 0$. Let $h = \underset{k \in s}{\operatorname{argmax}}\, x^*_k$ be the best path of the vMCC connection. On the best path h we have $\frac{\psi_h x_h^2}{RTT_h^2 (\sum_{k \in s} x^*_k)^2} = \frac{1}{2}\lambda_h x_h^2 + (1 - \frac{\alpha}{2})x_r^2$. We use the loss-based congestion window increase algorithms in the MPTCP Linux kernel. For the parameter ψ_r, at the equilibrium the existing algorithms Lia [5] ($\psi_r = \frac{RTT_r^2 (\sum_k x_k)^2}{(\sum_{k \in s} w_k)^2}$), Olia [6] ($\psi_r = 1$) and Balia [8] ($\psi_r = \frac{2}{5} + \frac{1}{2}\frac{\max_k x_k}{x_r} + \frac{1}{10}(\frac{\max_k x_k}{x_r})^2$) all satisfy $\psi_r \leq 1$. Then we have $\sqrt{\frac{\psi_h}{0.5\lambda_h + (1 - \frac{\alpha}{2})}}/RTT_h \leq \sqrt{\frac{1}{0.5\lambda_h + (1 - \frac{\alpha}{2})}}/RTT_h \leq \sqrt{\frac{2}{\lambda_h}}/RTT_h$. $\sqrt{\frac{\psi_h}{0.5\lambda_h + (1 - \frac{\alpha}{2})}}/RTT_h$ is the aggregated throughput vMCC can achieve at the stable state. On the best path h, conventional TCP can achieve the throughput $\sqrt{\frac{2}{\lambda_h}}/RTT_h$, and ECN-enabled TCP can achieve the throughput $\sqrt{\frac{1}{0.5\lambda_h + (1 - \frac{\alpha}{2})}}/RTT_h$. □

C. ECN-aware path-sharing subflow termination

Our experiment results in Fig. 7 show that path-sharing subflows may lead to severe performance degradation: the average goodput per VM decreases with the increasing number of subflows per path; the values of RTT, total retransmissions, and instant energy consumption also increase in that case. This motivates us to design a detection mechanism that can identify and terminate suspicious path-sharing subflows. The goal is to maintain one subflow per path so that other path-sharing subflows are no longer active. For latency-sensitive short flows in virtualized datacenters, the path-sharing subflow detection and termination must be quick enough to handle the flows that have strict deadlines. As we have discussed, ECN has already been exploited in our design to reply to the sender whether congestions happen in the intermediate nodes. Accordingly, we can also use ECE counts as the signal to help identify path-sharing subflows.

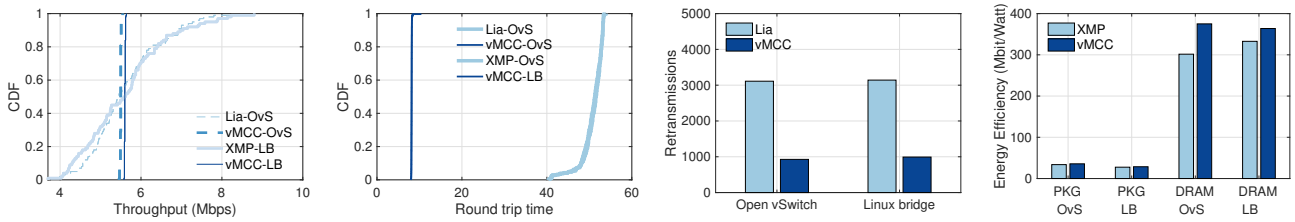
The design of path-sharing subflow detection is shown in Algorithm 2. When a sender receives ACK packets with ECE-marked on one subflow, a counter starts to keep the record. When the number of ECE-marked packets is greater than the threshold, the detection algorithm is triggered. It examines whether the just received ECE-marked subflow has similar RTTs and similar counts of ECE notifications when compared with other subflows. If two subflows have similar RTTs and counts of ECE notifications, the recently marked subflow will be terminated. On the contrary, if the two subflows do not have similar RTTs, the ECE-marked packet will be processed as a normal congestion notification and vMCC will go to execute the parameter update for a DCTCP-like window decrease. The references of RTTs and ECE counts stored for each subflow are cleared when there is no need for further detection. And the path-sharing detection is only executed during the beginning stage of the transmission.

VI. Evaluations

In this section, we provide evaluations on the vMCC design. Similar to the preliminary measurement study, we use throughput, RTT, retransmission, and energy efficiency as metrics to better understand the performance. The hardware and software settings are similar to those introduced in Section III.

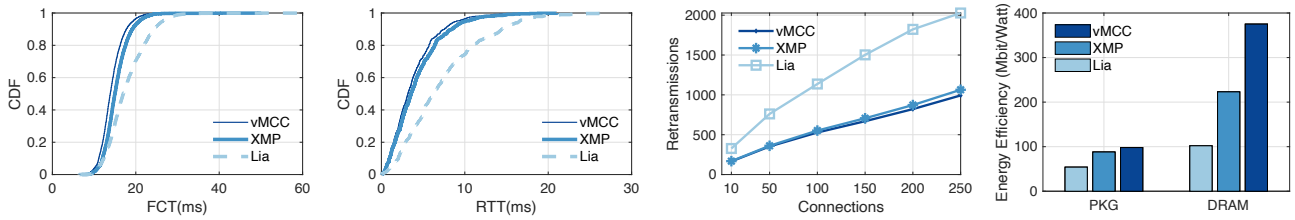
A. ECN-enabled Multipath Congestion Control

To begin with, we examined the congestion control performance by comparing vMCC, MPTCP Lia, and XMP with different virtual switch settings. The target virtual machine, running an NFS server, set up 100 parallel connections to send large file trunks. The connections all leverage two disjoint paths in this scenario. Fig. 9 presents the detailed comparisons. Fig. 9a first shows the throughput of these connections, and we use its CDF to better understand the results. The aggregate throughput of vMCC is 585 Mbps with the Open vSwitch settings, having a slight advantage over Lia (523.1 Mbps) and XMP (562 Mbps). A notable difference lies in the flow fairness of the 100 parallel connections. When enabling vMCC in both the Open vSwitch and Linux Bridge settings, the throughput of each connection is almost perfectly balanced around a specific value. It clearly shows that vMCC provides better fairness than other schemes. Next, we compare the RTT statistics in Fig. 9b. When using Lia, the network congestion can only be detected when the queue piles up and packet drop event happens, it is quite intuitive that its RTT is 4.5x worse than using vMCC. Since XMP also reacts to ECN when performing congestion control, the RTT statistics are quite similar to vMCC. The curves of the two therefore almost overlap with each other. As a next step, we compare the number of retransmissions of vMCC and Lia in Fig 9c. We measured the number of retransmission per connection in both the Open vSwitch and Linux Bridge settings. With the results presented in Fig. 9c,



(a) Throughput (b) Round trip time (c) Retransmissions per connection (d) Energy efficiency

Fig. 9: Long flow performance.



(a) Flow completion time (b) Round trip time (c) Retransmissions per connection (d) Energy efficiency

Fig. 10: Short flow performance.

we can see that vMCC achieves retransmission reduction by up to 62.3% in this figure. As for the comparison of energy efficiency, Fig. 9d presents the energy measurement of processor packages (PKG) and DRAM, and it clearly shows a $5.2 - 18.6\%$ improvement achieved by the vMCC design.

We also examined the performance when multiple VMs are sending traffic with short flows. This is performed by issuing small object queries on Memcached servers. The detailed settings are as follows: we developed a client application to generate traffic for fetching small objects based on a Poisson process. The distribution of the object size is similar to the cache workload in [27]. Our experiment is then conducted with 12 co-located VMs. The Memcached applications, running on all the 12 virtual machines, responded with the requested data. The experimental results are shown in Fig. 10. In particular, Fig. 10a shows the CDF of the short flow completion time. The results indicate a 5.1% reduction of average flow completion time achieved by vMCC (compared with XMP), and a 22.1% reduction when compared with Lia. The RTT statistics, shown in Fig. 10b, also indicate a 7.2% reduction (compared with XMP) when running such workloads. The improvements lie in the fact that the virtual switch also enables ECN markings and provides fast congestion feedback in time. Next, we present the retransmission statistics and energy efficiency indicators in Fig. 10c and Fig. 10d, respectively. As shown in Fig. 10c, vMCC also reduced the number of retransmissions per connection by up to 48.7% in a comparison with Lia. To pinpoint the energy savings, in Fig. 10d, the processor packages (PKG) and DRAM indicators of vMCC are significantly improved when compared with both XMP and Lia.

B. Path-sharing subflow termination

We then tested the path-sharing subflow termination performance during the experiments. In this test, we calculate the F-score as the metric. To make it clear to readers, We present the definitions of the positive/negative samples in Table II. The

TABLE II: F-Measure Defination
(T-True, F-False, P-Positive, N-Negaitve)

Type	Has path-sharing subflows?	Terminated?
T&P	Yes	Yes
T&N	No	No
F&N	Yes	No
F&P	No	Yes

proposed subflow termination algorithm is also tested with different virtualization platforms and virtual switch settings. Drawn from the preliminary measurements, we selected γ (ECE-marked packet count) to be 10 for Algorithm 1. We then varied the parameter θ (the difference of ECE-marked packet count between subflows) and δ (ratio of the difference in RTT) for Algorithm 2 to tune the overall subflow termination performance. The F-score statistics with two parameter combinations are then presented in Table III and Table IV, respectively. The results in Table III are with parameter set ($\theta = 3, \beta = 0.10$), and we tested the path sharing detection performance with different number of parallel connections to examine the scalability. The F-score values in all the cases are between 0.7068 and 0.8138. The best results with this parameter set are achieved in KVM environments with the OvS vhost-net configuration. The results with another parameter set ($\theta = 5, \beta = 0.14$) are presented in Table IV and remain stable across different network configurations. The F-score values are between 0.6963 and 0.8146 in the results.

VII. Conclusion

In this paper, we examined the performance of multipath TCP deployment in typical virtualized environments. With realworld experiments conducted, we not only quantified the improvement brought by multipath TCP deployment, but also pinpointed the caveats of deploying multipath TCP on virtual machines and containers. We proposed and implemented vM-

TABLE III: Subflow termination statistics ($\theta = 3, \beta = 0.10$)

Settings	No. of Conns.	Precision	Recall	F-score
OvS vhost-net in KVM	1	0.8419	0.7875	0.8138
	2	0.8295	0.8022	0.8156
	5	0.8019	0.7779	0.7908
	10	0.7455	0.7571	0.7562
OvS DPDK in KVM	1	0.8125	0.7959	0.8041
	2	0.7905	0.7753	0.7828
	5	0.7850	0.7592	0.7719
	10	0.7115	0.7127	0.7121
Linux bridge in KVM	1	0.7236	0.6984	0.7108
	2	0.7277	0.6905	0.7123
	5	0.7120	0.6913	0.7015
	10	0.7010	0.7127	0.7068
OvS with Docker	1	0.7792	0.8004	0.7897
	2	0.7652	0.7831	0.7741
	5	0.7581	0.7653	0.7617
	10	0.7417	0.7591	0.7503

TABLE IV: Subflow termination statistics ($\theta = 5, \beta = 0.14$)

Settings	No. of Conns.	Precision	Recall	F-score
OvS vhost-net in KVM	1	0.7515	0.8611	0.8026
	2	0.7657	0.8701	0.8146
	5	0.7500	0.8684	0.8049
	10	0.7317	0.8276	0.7767
OvS DPDK in KVM	1	0.7351	0.8286	0.7774
	2	0.7034	0.7451	0.7237
	5	0.7067	0.6696	0.6877
	10	0.6895	0.6172	0.6514
Linux bridge in KVM	1	0.7566	0.7063	0.7306
	2	0.7666	0.7606	0.7636
	5	0.7373	0.7470	0.7421
	10	0.7132	0.7145	0.7138
OvS with Docker	1	0.8132	0.7591	0.7852
	2	0.6980	0.6922	0.6951
	5	0.6911	0.6899	0.6905
	10	0.7069	0.6125	0.6563

CC, which comprises enhanced congestion control algorithms and virtual switch plug-ins. The solution provides enhanced congestion control performance for different virtualization platforms. We have also conducted comprehensive experiments to quantify the performance gain. As a future direction, we will continuously explore the scalability-accuracy tradeoff and enhance the robustness of the vMCC design.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers of IWQoS 2020 for their insightful comments and feedback on the paper. This work is supported by the National Natural Science Foundation of China, under Grant U1901601 and Grant 61602214.

REFERENCES

[1] "Magic Quadrant for x86 Server Virtualization Infrastructure," https://www.gartner.com/doc/reprints?ct=160707&id=1-3B9FAM0&st=sb.

[2] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. of ACM SIGCOMM*, 2011.

[3] C. Raiciu, C. Paasch, S. Barre, A. Ford *et al.*, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proc. of USENIX NSDI*, 2012.

[4] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson *et al.*, "The Design and Implementation of Open vSwitch," in *Proc. of USENIX NSDI*, 2015.

[5] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. of USENIX NSDI*, 2011.

[6] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J. Y. L. Boudec, "MPTCP is not Pareto-optimal: Performance Issues and a Possible Solution," in *Proc. of ACM CoNEXT*, 2012.

[7] Y. Cao, M. Xu, and X. Fu, "Delay-based Congestion Control for Multipath TCP," in *Proc. of IEEE ICNP*, 2012.

[8] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, Design, and Implementation," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 596–609, 2016.

[9] A. Agache, R. Deaconescu, and C. Raiciu, "Increasing Datacenter Network Utilisation with GRIN," in *Proc. of USENIX NSDI*, 2015.

[10] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit Multipath Congestion Control for Data Center Networks," in *Proc. of ACM CoNEXT*, 2013.

[11] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A Multipath Transport Protocol for Data Centers," in *Proc. of IEEE INFOCOM*, 2016.

[12] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan *et al.*, "Fast and Cautious: Leveraging Multi-path Diversity for Transport Loss Recovery in Data Centers," in *Proc. of USENIX ATC*, 2016.

[13] M. Kheirkhah and M. Lee, "AMP: An Adaptive Multipath TCP for Data Center Networks," in *Proc. of IFIP Networking*, 2019.

[14] E. Dong, X. Fu, M. Xu, and Y. Yang, "DCMPTCP: Host-Based Load Balancing for Datacenters," in *Proc. of IEEE ICDCS*, 2018.

[15] ——, "Low-Cost Datacenter Load Balancing with Multipath Transport and Top-of-Rack Switches," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[16] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in Data-center Networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 345–358, 2013.

[17] L. Cheng, C. Wang, and F. C. Lau, "PVTCP: Towards Practical and Effective Congestion Control in Virtualized Datacenters," in *Proc. of IEEE ICNP*, 2013.

[18] K. He, E. Rozner, K. Agarwal, Y. G. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks," in *Proc. of ACM SIGCOMM*, 2016.

[19] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy, "Virtualized Congestion Control," in *Proc. of ACM SIGCOMM*, 2016.

[20] "How many VMs per host is too many?" http://searchdatacenter.techtarget.com/feature/How-many-VMs-per-host-is-too-many.

[21] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. of ACM SIGCOMM*, 2010.

[22] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: Practical Work-conserving Bandwidth Guarantees for Cloud Computing," in *Proc. of ACM SIGCOMM*, 2013.

[23] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing Bandwidth Guarantees, Work Conservation and Low Latency Simultaneously in the Cloud," in *Proc. of IEEE INFOCOM*, 2016.

[24] Z. Liu, K. Chen, H. Wu, S. Hu, Y.-C. Hu, Y. Wang, and G. Zhang, "Enabling Work-conserving Bandwidth Guarantees for Multi-tenant Datacenters via Dynamic Queue Binding," in *Proc. of IEEE INFOCOM*, 2018.

[25] C. Xu, Z. Zhao, H. Wang, and J. Liu, "On the interplay between network traffic and energy consumption in virtualized environment: An empirical study," in *Proc. of IEEE CLOUD*, 2014.

[26] J. Zhao, J. Liu, H. Wang, C. Xu, W. Gong, and C. Xu, "Measurement, Analysis, and Enhancement of Multipath TCP Energy Efficiency for Datacenters," *IEEE/ACM Transactions on Networking*, 2019.

[27] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in Multi-Service Multi-Queue Data Centers," in *Proc. of USENIX NSDI*, 2016.