MPTCP under Virtual Machine Scheduling Impact

Phuong Ha, Lisong Xu
Department of Computer Science and Engineering, University of Nebraska-Lincoln
Lincoln, NE, USA 68588-0115, Email: {phuong.ha@huskers.unl.edu, xu@cse.unl.edu

GLOBECOM 2021 - 2021 IEEE Global Communications Conference | 978-1-7281-8104-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/GLOBECOM46510.2021.9685569

Abstract—**Multipath TCP (MPTCP) has captured the networking community's attention in recent years since it simultaneously transfers data over multiple network interfaces, thus increases the performance and stability. Existing works on MPTCP study its performance only in traditional wired and wireless networks. Meanwhile, cloud computing has been growing rapidly with lots of applications deployed in private and public clouds, where virtual machine (VM) scheduling techniques are often adopted to share physical CPUs among VMs. This motivates us to study MPTCP's performance under VM scheduling impact. For the first time, we show that VM scheduling negatively impacts all MPTCP subflows' throughput. Specifically, VM scheduling causes the inaccuracy in computing the overall aggressiveness parameter of MPTCP congestion control, which leads to the slow increment of the congestion windows of all MPTCP subflows instead of just a single subflow. This finally results in a poor overall performance of MPTCP in cloud networks. We propose a modified version for MPTCP, which considers VM scheduling noises when MPTCP computes its overall aggressiveness parameter and its congestion windows. Experimental results show that our modified MPTCP performs considerably better (with up to 80% throughput improvement) than the original MPTCP in cloud networks.**

Index Terms—**MPTCP; VM Scheduling; Clouds**

I. Introduction

Multipath TCP (MPTCP) [1] was introduced as an extension of the traditional single-path TCP to better utilize network resources by allowing simultaneous data transmission over multiple network interfaces. For example, mobile devices such as smart phones from Apple and Samsung have been enabled MPTCP [2] to transparently work over both WiFi and cellular networks for voice recognition applications since 2013. Another example is machines with multiple Ethernet interfaces in data centers transferring traffic over multiple interfaces simultaneously using MPTCP. With cloud computing's rapid development in recent years, MPTCP is also proposed to replace the single-path TCP in data centers [3] such as Amazon EC2 with significant throughput improvement.

MPTCP's main components including the path manager, congestion control (CC) algorithm, and scheduler. An MPTCP connection consists of multiple subflows where each subflow is a TCP session created on an available path between a pair of IP addresses of two MPTCP hosts. The path manager creates these subflows with their corresponding paths at the beginning of the MPTCP connection. Once the MPTCP connection is established, the CC algorithm controls the sending rate of each subflow, i.e., it updates each subflow's congestion window to achieve the design goals (i.e., improve throughput, do no harm, and balance congestion) [4]. Linked Increases Algorithm (LIA) [4] was first proposed to meet these goals and it became MPTCP's default CC algorithm. Then, the MPTCP scheduler decides how much data to send on each subflow with available congestion windows. The default scheduler [5] prefers the subflow with the smallest round-trip time (RTT) among all subflows with available congestion windows. Several algorithms have been proposed to improve MPTCP's CC algorithms and schedulers but only in traditional wired and wireless networks.

Deploying applications on clouds has become a trend because it reduces the setup and maintenance cost. Cloud providers often adopt virtual machine (VM) scheduling techniques to allocate physical CPUs among VMs. This allows them to provide customers with various options for different purposes to deploy their applications. Thus, we are motivated to study how MPTCP is affected under the VM scheduling impact. Although the VM scheduling impact on TCP has been experimentally studied [6], [7], [8], there has been no study on its impact on MPTCP. *For the first time, we analyze and find that VM scheduling has a negative impact on the throughput of ALL MPTCP subflows instead of just a single subflow, which is the fundamental reason for MPTCP's poor overall performance in cloud networks.* This is because MPTCP's CC algorithms are coupled. Specifically, all subflows increase their windows based on an important parameter called the overall aggressiveness parameter α. However, VM scheduling impacts negatively on the calculated α's accuracy, leading to the slow increment of all MPTCP subflows' windows instead of just a single subflow's window. This finally results in a poor overall performance of MPTCP in cloud networks.

The contributions of this paper are: (1) We are the first to analyze and identify that VM scheduling is the fundamental reason for MPTCP's poor performance in cloud networks. (2) Experimental results verify our analysis that VM scheduling negatively impacts *all* MPTCP's subflows. (3) From our analysis, we propose a modified version of MPTCP to improve its performance in cloud networks. Experimental results show that the modified MPTCP performs as well as the original MPTCP when there is no VM scheduling, and considerably better in terms of throughput (up to 80% improvement) and flow completion times when VM scheduling happens.

II. Related works

This section discusses existing works on MPTCP's main components in traditional wired and wireless networks, and the studies on the impact of VM scheduling on TCP.

MPTCP CC algorithms: MPTCP's performance metrics for a CC algorithm include fairness, responsiveness, and window-oscillation. [9] is proposed to further improve LIA's performance in terms of responsiveness and window-oscillation. Peng et al. [10] provide the fluid model of MPTCP's performance metrics to study the existing MPTCP

978-1-7281-8104-2/21/$31.00 ©2021 IEEE

CC algorithms, and propose Balanced Linked Adaptation (BALIA) to minimize the tradeoff among these metrics.

MPTCP Schedulers: MPTCP scheduler's main challenges are head-of-line blocking and receiver-window limitations [11]. [12], [13], have been proposed to improve the default scheduler's performance in wireless networks.

VM scheduling impact on TCP: [6], [14], [15] studied how VM scheduling affects the performance of TCP in cloud networks. Some prototypes, such as [7], [8], [16], were proposed to improve TCP's performance. However, [7], [8] require users to have certain hardware privilege controls, which are not feasible in public clouds. [16] does not require this type of control since it modifies TCP directly, but it focuses only on avoiding the wrong retransmission timeout of TCP. We show in Section IV that VM scheduling affects differently on MPTCP.

III. Overview of MPTCP

A. Design goals of MPTCP's congestion control

According to [4], a practical MPTCP CC algorithm has the following three design goals. *Goal 1 - Improve throughput:* MPTCP's throughput, the total throughput of all subflows, should be at least equal to the throughput of a TCP flow on the path with the highest available bandwidth. *Goal 2 - Do no harm (Fairness):* An MPTCP flow should not take up more capacity from any of the resources shared by its different paths than if it were a TCP flow using only one of these paths. *Goal 3 - Balance congestion:* An MPTCP flow should move traffic away from its most congested paths.

B. How LIA works

Since LIA is the default CC algorithm of MPTCP, we first summarize how it works in the case where there is no VM scheduling. All equations in this subsection are from [4], [9], and are necessary to understand our analysis presented in the next section. LIA is a CC algorithm that couples the additive increase function of all MPTCP subflows. That is, during the Congestion Avoidance state (CA), LIA incorporates how each subflow increases its congestion window so that MPTCP satisfies its three design goals, while it keeps the same algorithm of TCP for each subflow in other states including Slow Start, Fast Retransmit, and Fast Recovery, and the CA's multiplicative decrease phase.

Consider an MPTCP flow from a sender *SND* to a receiver *RCV* with a total of N available subflows. During CA, upon receiving the j^{th} ACK on subflow $r \in [1, N]$, *SND* updates the congestion window w_r of subflow r [4] using Eq. (1).

$$w_r = w_r + \min\left(\frac{\alpha}{w_{total}}, \frac{1}{w_r}\right) \tag{1}$$

where w_{total} is the total cwnd of all subflows when *SND* receives the j^{th} ACK, that is $w_{total} = \sum_{i=1}^{N} w_i$. The parameter α controlling the overall MPTCP aggressiveness, is computed using Eq. (2) as described in [4]:

$$\alpha = w_{total} \times \frac{\max_{i\in[1,N]}\left(w_i/sRTT_i^2\right)}{\left(\sum_{i=1}^{N} w_i/sRTT_i\right)^2} \tag{2}$$

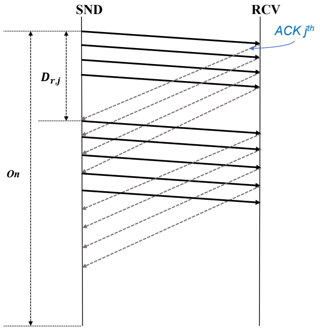


Fig. 1. The actual RTT $D_{r,j}$ for the j^{th} ACK of MPTCP subflow r in a case without VM scheduling.

where $sRTT_i$ is the estimated smoothed RTT of subflow i. Combining Eq. (1) and Eq. (2), we have:

$$w_r = w_r + \min\left(\frac{\max_{i\in[1,N]}\left(w_i/sRTT_i^2\right)}{\left(\sum_{i=1}^{N} w_i/sRTT_i\right)^2}, \frac{1}{w_r}\right) \tag{3}$$

The actual RTT $D_{r,j}$ and smoothed RTT $sRTT_r$ of subflow r after receiving the j^{th} ACK can be computed as follows as described in [17]. The actual RTT $D_{r,j}$ is illustrated in Fig. 1.

$$D_{r,j} = T_r + q_{r,j} \tag{4}$$

$$sRTT_r = \gamma \times sRTT_r + (1-\gamma) \times D_{r,j} \tag{5}$$

where $\gamma = 7/8$ [17], T_r is the propagation delay of subflow r in the network, and $q_{r,j}$ is the queuing delay of the j^{th} ACK of subflow r in the network.

MPTCP's first design goal is studied in [9] by considering a special case where all MPTCP subflows have same smoothed RTT values. In this special case, the throughput of MPTCP at the steady state x_{total} can be computed using Eq. (6) as described in [9]. Intuitively Eq. (6) means, when an MPTCP flow shares the link bandwidth with TCP flows at the steady state, the average total throughput of the MPTCP flow is equal to the throughput that a TCP flow can achieve on the best path (say k), specifically $\frac{\sqrt{2/p_k}}{sRTT_k}$ [9] where p_k is the packet loss probability of the best path k at the steady state.

$$x_{total} = \sum_{i=1}^{N} \frac{w_i}{sRTT_i} = \max_{i\in[1,N]} \frac{\sqrt{2/p_i}}{sRTT_i} = \frac{\sqrt{2/p_k}}{sRTT_k} \tag{6}$$

With this insight, MPTCP is expected to achieve its first design goal, i.e., it achieves at least the highest throughput that a TCP flow can achieve. However, we show in the next section that MPTCP could not achieve this goal in cloud networks when there is VM scheduling.

IV. MPTCP under VM scheduling impact

In this section, we analyze MPTCP's performance under VM scheduling using LIA - MPTCP's default CC algorithm.

In cloud networks, a VM can be scheduled on the physical CPU for some On periods. The VM is in a sleep time S when it is scheduled off, and has to wait until its turn to be scheduled on again. Depending on some factors such as the hypervisor type, the total number of VMs sharing the same

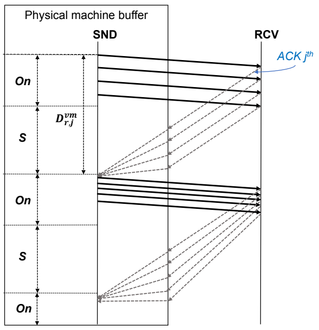


Fig. 2. The actual RTT $D_{r,j}^{vm}$ for the j^{th} ACK of MPTCP subflow r in a case with heavy VM scheduling.

CPU, and the VM scheduling algorithm, the sleep time S of a VM can reach tens of milliseconds [6], [8]. To simplify the analysis in this section, we assume that both On and S of a VM are constant. We consider a heavy VM scheduling case, where VM scheduling happens every round-trip propagation time for each subflow, i.e., $On < T_r^{vm} < On + S,\ S > 0$ for $\forall r \in [1..N]$. Note that, we make the constant On and S assumption and consider the heavy VM scheduling only in this analysis. Our experiments in Section VI have realistic random On and S values depending on several virtualization factors, such as the time slice and weight of a VM.

Below we analyze MPTCP's performance under heavy VM scheduling, and compare its performance with the case without VM scheduling as analyzed in the previous section. To distinguish between the notations without and with VM scheduling, we use notations with superscript "vm" for the analysis with VM scheduling. We consider the same network condition on each path $r \in [1, N]$ for the cases without and with VM scheduling, i.e., $p_r = p^{vm}$, $q_{r,j} = q_{r,j}^{vm}$, and $T_r = T_r^{vm}$ for $\forall r \in [1, N]$.

Fig. 2 illustrates the heavy VM scheduling impact on MPTCP. Intuitively, ACKs from *RCV* are stored at the physical machine's buffer when *SND* is scheduled off. After *SND* is scheduled on, the ACKs are then transferred to *SND*. Due to the sleep time S of *SND*, both the actual RTT $D_{r,j}^{vm}$ and smoothed RTT $sRTT_r^{vm}$ for the j^{th} ACK of subflow r are inaccurately estimated. The inaccurate $sRTT_r^{vm}$ leads to an inaccurate α, which in turn leads to a slower increment of w_r^{vm} for all subflows $r \in [1..N]$ and finally leads to a poor overall throughput of MPTCP under VM scheduling. Thus, the longer the sleep time S, the lower the overall MPTCP throughput. The formal analysis is given below.

The following theorem shows that given the same network condition and the same current congestion window of each subflow for cases without and with VM scheduling, the window increment without VM scheduling is higher than or equal to that with VM scheduling. As we compare the cases with and without VM scheduling when *SND* receives the j^{th} ACK on subflow $r \in [1, N]$, we will omit j for simplicity.

***Theorem** 1:* Given same network condition $p_r = p_r^{vm}$, $q_{r,j} = q_{r,j}^{vm}$, $T_r = T_r^{vm}$ for $\forall r \in [1..N]$, same initial window sizes $w_r = w_s = w_r^{vm} = w_s^{vm}$ for $\forall r, s \in [1..N]$, and same initial smoothed RTT values $sRTT_r = sRTT_r^{vm}$ for $\forall r \in [1..N]$, we have new $w_r \geq$ new w_r^{vm} for $\forall r \in [1..N]$.

Proof: With heavy VM scheduling (i.e., $On < T_r^{vm} < On + S,\ S > 0$), the actual RTT $D_{r,j}^{vm}$ for the j^{th} ACK of subflow r is inaccurately estimated as Eq. (7) after *SND* is scheduled on again, due to *SND*'s sleep time S.

$$D_{r,j}^{vm} = T_r^{vm} + q_{r,j}^{vm} + S \tag{7}$$

Similar to Eq. (5), *SND* updates $sRTT_r^{vm}$ as:

$$sRTT_r^{vm} = \gamma \times sRTT_r^{vm} + (1 - \gamma) \times D_{r,j}^{vm} \tag{8}$$

Given the same network condition for the cases with and without VM scheduling and because of Eqs. (4), (5), (7), and (8), we have

$$\begin{aligned} sRTT_r^{vm} &= \gamma \times sRTT_r^{vm} + (1 - \gamma) \times (D_{r,j} + S) \\ &= sRTT_r + (1 - \gamma) \times S \end{aligned} \tag{9}$$

The new congestion window w_r^{vm} of subflow r is updated as:

$$w_r^{vm} = w_r^{vm} + \min\left(\frac{\alpha^{vm}}{w_{total}^{vm}}, \frac{1}{w_r^{vm}}\right) \tag{10}$$

where

$$\alpha^{vm} = w_{total}^{vm} \times \frac{\max_{i \in [1,N]} \left(w_i^{vm}/(sRTT_i^{vm})^2\right)}{\left(\sum_{i=1}^{N} w_i^{vm}/sRTT_i^{vm}\right)^2} \tag{11}$$

Given the same initial window size for the cases with and without VM scheduling, we have $w_{total} = w_{total}^{vm}$. Below we prove that $\alpha^{vm} \leq \alpha$, where α is defined in Eq. (2).

Given the same network condition and same initial window size for the cases with and without VM scheduling, let k denote the best path in both cases. That is, $w_k^{vm}/(sRTT_k^{vm})^2 = \max_{i \in [1,N]} \left(w_i^{vm}/(sRTT_i^{vm})^2\right)$, $w_k/(sRTT_k)^2 = \max_{i \in [1,N]} \left(w_i/(sRTT_i)^2\right)$. Note that also because every subflow has the same initial window size, we have $sRTT_k^{vm} = \min_{i \in [1,N]} sRTT_i^{vm}$ and $sRTT_k = \min_{i \in [1,N]} sRTT_i$. We have

$$\begin{aligned} \alpha^{vm} &= w_{total}^{vm} \times \frac{w_k^{vm}/(sRTT_k^{vm})^2}{\left(\sum_{i=1}^{N} w_i^{vm}/sRTT_i^{vm}\right)^2} \\ &= \frac{w_{total}^{vm} \times w_k^{vm}}{\left(\sum_{i=1}^{N} \frac{w_i^{vm} \times sRTT_k^{vm}}{sRTT_i^{vm}}\right)^2} \\ &= \frac{w_{total}^{vm} \times w_k^{vm}}{\left(\sum_{i=1}^{N} \frac{w_i^{vm} \times (sRTT_k + (1-\gamma)S)}{sRTT_i + (1-\gamma)S}\right)^2} \end{aligned} \tag{12}$$

It can be proved that Eq. (12) is a non-increasing function of S using the derivative of α^{vm} with respect to S. That is, $\alpha^{vm} \leq \alpha$, where the equality holds when every subflow has the same smoothed RTT values $sRTT_r^{vm} = sRTT_s^{vm}$ for $\forall r, s \in [1, N]$.

Finally, combining with Eq. (3) and Eq. (10), we have new $w_r \geq w_r^{vm}$ for $\forall r \in [1, N]$. ■

Note that we only assume the same network condition, same initial window sizes, and same initial smoothed RTT values in the analysis. The experiments presented in the evaluation section consider more general cases.

The following corollary shows that MPTCP under heavy VM scheduling could not achieve the first design goal in the steady state in the same special case as discussed in [9] and in the previous section where all MPTCP subflows have the same smoothed RTT values.

***Corollary** 1:* Given $p_r = p_r^{vm}$, $q_{r,j} = q_{r,j}^{vm}$, $T_r = T_r^{vm}$, and initially $w_r = w_r^{vm}$ for $\forall r \in [1..N]$, if $sRTT_r^{vm} = sRTT_s^{vm}, \forall r, s \in [1, N]$, then $x_{total}^{vm} < \frac{\sqrt{2/p_k}}{sRTT_k}$.

Proof: Under the heavy VM scheduling condition, at the steady state, if all subflows have the same $sRTT^{vm}$, the average total throughput x_{total}^{vm} can be computed using Eq. (13), which is similar to Eq. (6).

$$x_{total}^{vm} = \sum_{i=1}^{N} \frac{w_i^{vm}}{sRTT_i^{vm}} = \max_{i \in [1,N]} \frac{\sqrt{2/p_i^{vm}}}{sRTT_i^{vm}} = \frac{\sqrt{2/p_k^{vm}}}{sRTT_k^{vm}} \tag{13}$$

Since $p_k^{vm} = p_k$ where k is the best path, we have: $x_{total}^{vm} = \frac{\sqrt{2/p_k}}{sRTT_k^{vm}}$. From Eq. (9), we also have: $sRTT_k^{vm} > sRTT_k$. Combining with Eq. (6), we then have:

$$x_{total}^{vm} < x_{total} = \frac{\sqrt{2/p_k}}{sRTT_k} \tag{14}$$

■

Intuitively, this corollary shows that MPTCP throughput under heavy VM scheduling is smaller than the maximum throughput that a TCP flow can achieve (i.e.,$\frac{\sqrt{2/p_k}}{sRTT_k}$ This is also verified in our experiments in Section VI.

V. Improve MPTCP's throughput

A. VM scheduling Detection

In [6], [14], [15], VM scheduling is showed to cause the measured RTT to be inflated and become larger than the propagation delay by tens of milliseconds. To detect VM scheduling, we consider *SND* is in the off period upon receiving the j^{th} ACK if $D_{r,j}^{vm}/minRTT_r > \theta$, where $minRTT_r$ is the minimum measured RTT of subflow r, and θ is a parameter for detecting VM scheduling. Currently we choose $\theta = 10$, i.e., when the measured RTT $D_{r,j}^{vm}$ is θ times greater than $minRTT_r$, *SND* is considered to be scheduled off during a sleep period. We plan to study the impact of θ on MPTCP's performance and choose a better value in our future work.

B. sRTT Estimation

Upon receiving the j^{th} ACK, the smoothed RTT $sRTT_r^{vm}$ is estimated using Eq. (15).

$$sRTT_r^{vm} = \begin{cases} \gamma \times sRTT_r^{m} + (1-\gamma) D_{r,j}^{vm}, \text{if } \frac{D_{r,j}^{vm}}{minRTT_r} < \theta \\ sRTT_r^{vm}, \text{ otherwise} \end{cases} \tag{15}$$

C. Modification in CA

SND can be scheduled off anytime during its transmission. From Section IV, we see that if MPTCP could not increase its cwnd in CA, it could not reach the best path's throughput. Thus, we modify the CA state to improve MPTCP's throughput. Specifically, upon receiving the j^{th} ACK, if VM scheduling is detected, α is updated to its past maximum values in a time interval of parameter δ. Since VM scheduling noises are in the order of tens to hundreds of milliseconds [6], [14], [15], currently we set δ to 1 second, which allows us to capture the maximum α with the least VM scheduling impact. By doing so, MPTCP can increase its windows and send more packets in the next *On* period that it could not send in previous periods because of the *SND's* sleep time.

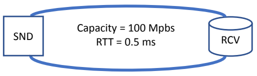


Fig. 3. Testbed topology: A VM as *SND* connecting to a physical machine as *RCV* via different paths. Each link has a capacity of 100 Mbps, and a Round-trip propagation delay of 0.5 ms. Since we change the cross-traffic according to each group of experiments, we do not show it in this figure.

VI. Evaluation and Discussion

A. Experiment setup

Our testbed includes a sender *SND* that is a VM sharing a physical CPU with other three VMs on a Xen server [18], and a receiver *RCV* that is a physical machine. *SND* and *RCV* are connected via two different paths as demonstrated in Fig. 3. Each path has a capacity of 100 Mbps and a round-trip propagation delay of 0.5 ms. In these experiments, we run lookbusy [19] first on the *SND* for several hours, and then on other VMs while *SND* is sending traffic to *RCV* in 30 s using Iperf [20]. This ensures that *SND* consumes a high number of its credits, and that VM scheduling happens as these VMs are competing for CPU cycles. Some methods to vary the *SND's* sleep time are: (1) increasing the number of VMs sharing the same physical CPU with *SND*; (2) increasing the credit scheduler's time slice; (3) decreasing *SND*'s weight so that it is given a less amount of CPU time compared to a VM with a higher weight. Amazon EC2 uses the credit scheduler with some modifications for their T2 VMs, of which in [6], [21] show that a VM on EC2 can be scheduled off a physical CPU from tens to hundreds of ms. We have emulated the effect of VM scheduling using all above three methods in our testbed and got similar patterns. Below we report our results for varying the time slice (Ts) and varying the weight (W) in these experiments. We design three groups of experiments to answer the following three questions, respectively.

- **Question 1**: Are the analysis of Theorem 1 and Corollary 1 valid for the original MPTCP?
- **Question 2**: Can the modified MPTCP achieve better throughput (i.e., MPTCP design goal 1) than the original MPTCP in cloud networks?
- **Question 3**: Does the modified MPTCP achieve the other two MPTCP design goals (i.e., Do no harm, balance congestion) as the original MPTCP?

B. Experiments for Question 1: Analysis Validation

To answer question 1, we vary time slices from 30 ms, 120 ms to 480 ms while keeping all VMs' weights at the default value. When the original MPTCP enters the CA state, we compare and expect that α, the best subflow's window, and MPTCP's total throughput in the case with VM scheduling are smaller than those of the case without VM scheduling.

From Theorem 1 and Corollary 1, we learn that the *SND's* sleep time inflates *SND*'s actual RTT resulting in the inaccuracy in computing $sRTT$ then α, and the window. Fig. 4 shows the decrease of α as *SND* is scheduled off for a longer time when the time slice increases during CA state. The windows under VM scheduling with different time slices, as shown in Fig. 5 labeled "Ts120" and "Ts480", could not increase as much as in the case without VM scheduling,

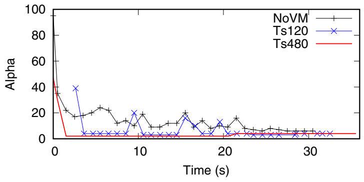


Fig. 4. α is inaccurately computed because the sRTT is inflated and inaccurately estimated due to *SND's* sleep time. This verifies Theorem 1.

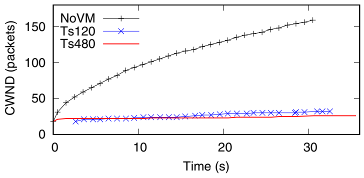


Fig. 5. The congestion window w_r^{vm} of a subflow under VM scheduling, labeled "Ts120" or "Ts480", is smaller than the congestion window w_r in the case without VM scheduling. This verifies Theorem 1.

labeled "NoVM". We also see from Fig. 6 that MPTCP's throughput reduces as the *SND*'s sleep time increases with the time slice.

Discussion: These results validate Theorem 1 and Corollary 1 that VM scheduling causes MPTCP to not achieve the throughput as a TCP flow can on the best path - the first design goal. This is because VM scheduling negatively affects the accuracy of calculated α, and of *All* MPTCP subflows' windows instead of a single subflow's.

C. Experiments for Question 2: MPTCP Design Goal 1

To answer question 2, we run the experiments with different time slices as in group A, and with different *SND*'s weights from 8, 32 to 128 while keeping the default value time slice of 30 ms. We also conduct the experiments studying the flow completion time of our modified MPTCP and the original MPTCP when transferring a 610 MB file under the VM scheduling 30 ms-time slice and default weights for all VMs.

The percentage throughput improvement of our modified MPTCP and the original MPTCP is shown in Fig. 7. Specifically, it is calculated as the throughput difference between the modified MPTCP and the original MPTCP normalized by the original MPTCP's throughput. As expected, Fig. 7(a) shows that without VM scheduling, our modified MPTCP achieves a similar throughput as the original MPTCP (i.e., percentage throughput improvement close to zero), and a higher throughput when VM scheduling happens, especially about 80% higher than the original MPTCP when the time slice is 480 ms. From Fig. 7(b), we can also see that as VM scheduling happens more often and for a longer time when *SND*'s weight decreases to 8, our modified MPTCP can achieve up to 60% higher throughput than the original MPTCP. Fig. 8 shows that our modified MPTCP completes transferring the file in 75 s with 78 Mbps throughput under VM scheduling, while it takes the original MPTCP more than 100 s with 50 Mbps throughput.

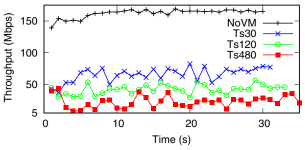
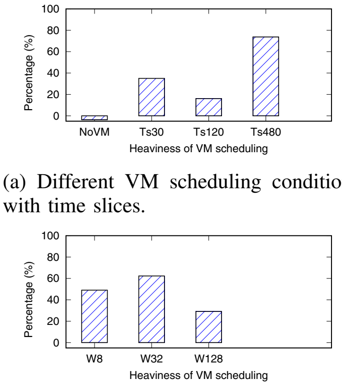


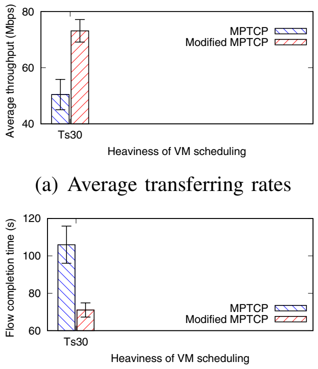
Fig. 6. MPTCP's throughput decreases significantly as *SND's* sleep time increases, i.e. when time slice increases from 30 ms to 120 ms, and 480 ms (labeled "Ts30", or "Ts120", or "Ts480"), MPTCP couldn't achieve the throughput as it does in the case without VM scheduling labeled "NoVm"). This verifies Corollary 1.



(a) Different VM scheduling conditions with time slices.

(b) Different VM scheduling conditions with weights.

Fig. 7. Percentage throughput improvement of modified TCP to original MPTCP: Without VM scheduling labeled "NoVM", modified MPTCP achieves throughput as high as original MPTCP; Different VM scheduling conditions with time slices labeled "Ts30", "Ts120", "Ts480", up to 80% higher throughput, and up to 60% higher throughput in cases with varying weights labeled "W8", "W32", "W128".



(a) Average transferring rates

(b) Flow completion time

Fig. 8. Modified MPTCP transfers a 610 MB file at a higher rate and in a shorter time than original MPTCP under 30 ms time sliced-VM scheduling.

Discussion: These experiments clearly show that our modified MPTCP can achieve significantly higher throughput than the original MPTCP in cloud networks with VM scheduling. Note that our modified MPTCP still could not reach the actual available bandwidth in cloud networks. This is because our modified MPTCP is designed to be as fair to other flows as its original MPTCP, and only being more aggressive when VM scheduling is detected to improve its performance.

D. Experiments for Question 3: MPTCP Design Goals 2 & 3

To answer question 3, we run two groups of experiments. In group 1, we study the fairness (design goal 2) by intro-

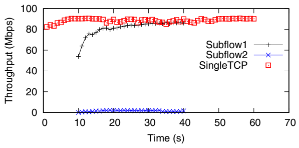


Fig. 9. Original MPTCP's fairness: MPTCP reduces its rate on subflow2 so that TCP single flow's rate is around 80 Mbps on the shared link.

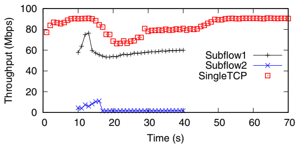


Fig. 10. Modified MPTCP's fairness: It also reduces its rate on subflow2, so TCP single flow's rate is around 70 Mbps on the shared link.

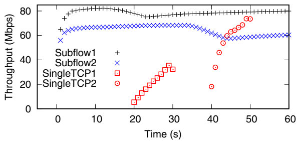


Fig. 11. Original MPTCP's load balancing: MPTCP sublfow 1's rate (black line) reduces as TCP single flow 1 starts from 20s, and subflow 2's rate (blue line) reduces as TCP single flow 2 starts from 40s.

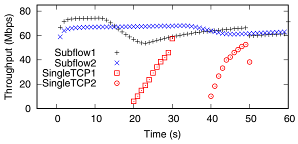


Fig. 12. Modified MPTCP's load balancing: Its sublfow 1's rate (black line) reduces as TCP single flow 1 starts from 20s, and subflow 2's rate (blue line) reduces as TCP single flow 2 starts from 40s.

ducing a single flow Cubic TCP from a different VM on the same Xen server on path 2 to *RCV* from 0-80 s, while MPTCP is sending traffic to *SND* from 15-45 s. In group 2, we study the balancing property (design goal 3) using experiments that include a single Cubic TCP flow from a different VM on the Xen server on each path to *RCV*: TCP flow 1 sends traffic from 10-20 s, TCP flow 2 sends traffic from 30-40 s, while an MPTCP flow sends traffic from 0-60 s.

For group 1, Figs. 9 and 10 show that both modified and original MPTCP can fairly share bandwidth with the single flow TCP on path 2 and thus achieve design goal 2. For group 2, Figs. 11 and 12 show that the original MPTCP and our modified MPTCP send more traffic on path 2 that is less congested than path 1 when the single flow starts on path 1, and shift traffic to path 1 when the single flow starts on path 2, thus, achieve design goal 3. Fig. 13 shows MPTCP's average smoothed RTTs from experiments of group 1 and group 2. We see that while our modified MPTCP achieves similar fairness and load balancing ability as the original MPTCP, it has smaller smoothed RTTs than the original MPTCP.

Discussion: These experiments show that our modified MPTCP achieves the same design goals 2 and 3 as the original MPTCP, while achieving significantly higher throughput (design goal 1) than the original MPTCP.

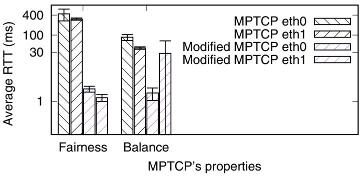


Fig. 13. Modified MPTCP has smaller average smoothed RTTs than original MPTCP in the experiments of groups 1 and 2 verifying its fairness and load balancing properties.

VII. Conclusion

In this paper, we study MPTCP's performance under VM scheduling impact and propose a modified version to improve its performance in cloud networks. Our testbed shows that without VM scheduling the modified MPTCP performs as well as MPTCP, and significantly better when VM scheduling happens. Our future works are to study the original MPTCP and our modified MPTCP in public clouds, and the modified MPTCP's parameters θ and δ based on VM scheduling levels.

Acknowledgment

The work presented in this paper was supported in part by NSF CNS-1616087.

References

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," *RFC 6824*, January 2013.

[2] "MultiPath TCP," https://www.multipath-tcp.org/.

[3] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 266–277, August 2011.

[4] C. Raiciu, M. Handly, and D. Wischik, "Coupled congestion control for multipath transport protocols," *IETF RFC 6356*, October 2011.

[5] "Why is the multipath tcp scheduler so important?" http://blog.multipath-tcp.org/blog/html/2014/03/30/why_is_the_multipath_tcp_scheduler_so_important.html.

[6] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon EC2 data center," in *Proceedings of IEEE INFOCOM*, San Diego, CA, March 2010.

[7] S. Gamage, A. Kangarlou, R. R. Kompella, and D. Xu, "Opportunistic flooding to improve tcp transmit performance in virtualized clouds," in *Proceedings of ACM Symposium on Cloud Computing*, 2011.

[8] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu, "vSnoop: improving TCP throughput in virtualized environments via acknowledgement offload," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2010.

[9] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J. Boudec, "MPTCP is not Pareto-optimal: Performance issues and a possible solution," in *Proceedings of ACM CoNEXT*, Nice, France, December 2012.

[10] Q. Peng, A. Walid, J. Hwang, and S. Low, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Transactions on Networking*, 2016.

[11] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of Multipath TCP schedulers," in *Proceedings of ACM SIGCOMM Workshop on Capacity Sharing Workshop*, 2014.

[12] Q. D. Coninck and O. Bonaventure, "Tuning multipath tcp for interactive applications on smartphones," in *IFIP Networking*, 2018.

[13] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "Daps: Intelligent delay-aware packet scheduling for multipath transport," in *Proceedings of ICC*, 2014, pp. 1222–1227.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of ACM SOSP*, New York, October 2003.

[15] Y. Xu, M. Bailey, B. Noble, and F. Jahanian, "Small is better: Avoiding latency traps in virtualized data centers," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, 2013.

[16] L. Cheng, C. Wang, and F. Lau, "PVTCP: Towards practical and effective congestion control in virtualized datacenters," in *Proceedings of IEEE ICNP*, Gottingen, Germany, October 2013.

[17] V. Paxson, "Computing TCP's retransmission timer," *IETF RFC 6298*, June 2011.

[18] the Linux Foundation, "Xen project," http://www.xenproject.org/.

[19] lookbusy - a synthetic load generator., https://www.devin.com/lookbusy/.

[20] Iperf, https://iperf.fr.

[21] P. Ha and L. Xu, "Available bandwidth estimation in public clouds," in *International Workshop on Big Data in Cloud Performance*, April 2018.