TCP BBR in Cloud Networks: Challenges, Analysis, and Solutions

Phuong Ha*, Minh Vu*, Tuan-Anh Le †, Lisong Xu*
* Department of Computer Science and Engineering
University of Nebraska-Lincoln
{pha, mvu, xu}@cse.unl.edu
† Faculty of Engineering and Technology
Thu Dau Mot University
letuanh@tdmu.edu.vn

***Abstract*—Google introduced BBR representing a new model-based TCP class in 2016, which improves throughput and latency of Google's backbone and services and is now the second most popular TCP on the Internet. As BBR is designed as a general-purpose congestion control to replace current widely deployed congestion control such as Reno and CUBIC, this raises the importance of studying its performance in different types of networks. In this paper, we study BBR's performance in cloud networks, which have grown rapidly but have not been studied in the existing BBR works. For the first time, we show both analytically and experimentally that due to the virtual machine (VM) scheduling in cloud networks, BBR underestimates the pacing rate, delivery rate, and estimated bandwidth, which are three key elements of its control loop. This underestimation can exacerbate iteratively and exponentially over time, and can cause BBR's throughput to reduce to almost zero. We propose a BBR patch that captures the VM scheduling impact on BBR's model and improves its throughput in cloud networks. Our evaluation of the modified BBR on the testbed and EC2 shows a significant improvement in the throughput and bandwidth estimation accuracy over the original BBR in cloud networks with heavy VM scheduling.**

***Index Terms*—TCP; BBR; VM Scheduling; Clouds**

I. Introduction

Bottleneck Bandwidth and Round-trip time propagation (BBR) [1] has been developed and tested by Google as "a general-purpose congestion control for LAN, WAN, data center, virtual machine (VM) guest" [2] and "as a drop-in replacement for Reno [3], Cubic [4], and DCTCP [5]". Representing a new model-based congestion control class, BBR demonstrates [1] improved throughput and latency on Google's backbone and services, and it is now the second most popular TCP on the Internet [6]. As BBR is designed as a general-purpose congestion control, this raises the importance of studying its performance in different types of networks.

However, existing works have studied BBR only in wired and wireless networks. Hock et al. [7] provide an extensive experimental evaluation of BBR's behavior in high-speed networks with various network settings. Atxutegi et al. [8] study performances of BBR, Cubic, and NewReno [9] in mobile networks. Scholz et al. [10] provide a framework for TCP congestion control measurements focusing on flexibility, portability, reproducibility, and automation. Zhang et al. [11] conduct simulation experiments to study performances of different TCPs including BBR in 5G cellular networks. Ware et al. [12] show that BBR degrades to window-based transmission when competing with traditional TCP flows in residential deep-buffered networks. Google researchers designed BBR version 2 (BBRv2) [2] to improve the performance of BBR version 1 (BBRv1) in terms of fairness and throughput but it is still in the developing stage.

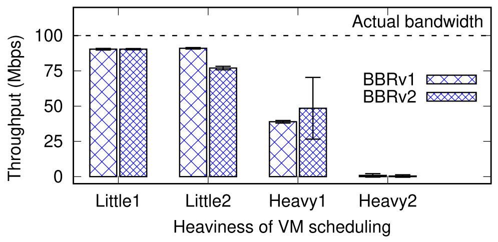


Fig. 1. EC2 experiments: "Little1" and "Little2" indicate little VM scheduling with credit balance of 25-30 and 12-15, respectively; "Heavy1" and "Heavy2" indicate heavy VM scheduling with credit balance of 0.4-1 and 0.15-0.2, respectively. Both BBRv1 and BBRv2 achieve poor performance with heavy VM scheduling.

Motivated by the rapid growth of cloud computing services and the lack of BBR study in cloud networks, we study the performance of BBR in cloud networks in this paper. Recent years have witnessed the rapid growth of cloud computing services that are predicted to even grow much bigger in the future. These services are provided by cloud providers such as Amazon and Google. To provide customers with various types of VMs, cloud providers often use VM scheduling techniques to virtualize physical CPUs and dynamically allocate CPU cycles among created VMs.

Existing works study the VM scheduling impact only on traditional TCPs. For example, VM scheduling in cloud networks is experimentally shown to cause traditional TCPs' throughput instability and abnormal Round-trip time (RTT) variation in [13], [14]. It is pointed out in [15] and [16] that delays in RTTs caused by VM scheduling noises result in a slower increase in the congestion window. Cheng et al. [17] show that VM scheduling also causes spurious retransmission timeouts which lead to the reduction of the congestion window.

Different from traditional TCPs, BBR uses a control loop of a pacing rate, a delivery rate, and an estimated bandwidth to adapt to changes in network conditions. While existing works on traditional TCPs show that VM scheduling only causes a certain decrease in the throughput, we find that, for the first time, VM scheduling impact on BBR is fundamentally different and even worse than on traditional TCPs because of BBR's control loop. An interesting result of our experiments conducted on Amazon EC2 with t2-nano VM as the senders using BBRv1 and BBRv2 is shown in Fig. 1. In these experiments, the senders connect to a physical machine as a receiver via a bandwidth of 100 Mbps and Round-trip propagation delay of 50 ms. We can see that the average throughputs of both BBRv1 and BBRv2 drop dramatically, and sometimes to almost zero, when heavy VM scheduling happens. Through rigorous analysis described in this paper, we have identified that BBR's control loop causes the underestimates of its elements to exacerbate iteratively and exponentially over time resulting in a throughput of almost zero. Note that our goal in this paper is not to design a new TCP specifically for cloud networks but to study the fundamental limitations of BBR (as representing model-based congestion control) in cloud networks and provide a solution to address the limitation.

The contributions of this paper are: (1) This is the first paper showing that VM scheduling has a fundamentally different and even worse impact on BBR's performance than on traditional TCPs: it only reduces traditional TCPs' throughput to a certain rate but it can bring BBR's throughput down to almost zero; (2) We identify and explain that, for the first time, the exponential underestimation of BBR's control loop over time is the main reason for the poor BBR performance in cloud networks; (3) We propose a patch that integrates BBR's model with a VM model to capture the VM scheduling impact on the control loop and improves its throughput in cloud networks; (4) Our evaluation on the testbed and EC2 shows that our modified BBR performs as well as the original BBR when there is little or no VM scheduling, and it achieves more than 50% increase in the throughput while maintaining similar RTTs as the original BBR under heavy VM scheduling.

II. Overview of BBR

A. The network path model of BBR

BBR uses a network path model to maintain and update its parameters across different time scales, in order to adapt to changes in network conditions of the path between the sender *SND* and the receiver *RCV*. The network path model of BBR [18] includes the following major components.

Available bandwidth β of a flow is the estimated maximum rate of a path from *SND* to *RCV* without causing any additional queuing delay in the network.

Minimum Round-trip time RTT_{min} is the minimum measured RTT for packets traversing along the path between *SND* and *RCV*.

B. The state machine of BBR: 4 states

BBR's state machine has four states: *Startup*, *Drain*, *ProbeBW*, and *ProbeRTT*. *Startup* is when its sending rate exponentially increases to estimate the maximum bandwidth. If its delivery rate could not gain more than 1.25 in three rounds, it removes the queue it has been building up till this point in *Drain*. These two states prepare BBR information to estimate the available bandwidth and minimum RTT of the link. BBR then moves to the *ProbeBW* with eight cycles. During either *Startup* or *ProbeBW*, BBR changes to *ProbeRTT* to estimate the new minimum RTT if its flow has not updated this value for more than 10 s, and then returns to the previous state. BBR spends most of its time in *ProbeBW* [1].

C. The control loop of BBR in ProbeBW

In *ProbeBW*, BBR starts pacing every data packet at a given *pacing rate*. Upon receiving ACKs for data packets sent over an interval, it computes the *delivery rate* and the *estimated bandwidth of a flow* during such an interval. Using this estimated bandwidth, BBR sets the *pacing rate* for next RTTs. *The pacing rate, delivery rate, and estimated bandwidth form a control loop* shown in Fig. 2 that BBR uses in its model to adjust its sending rate. This control loop continues to estimate and update these elements until the flow completes. How BBR computes these parameters and their relations and how the control loop under VM scheduling could cause BBR's throughput to reduce to zero are discussed in the next section.

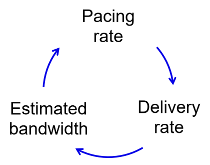


Fig. 2. BBR's control loop in *ProbeBW*. Section III shows that under VM scheduling, this control loop could cause its throughput reduce to zero.

TABLE I
Notation for cases (a) and (b)

Symbol	Description
β	Actual bandwidth of path from *SND* to *RCV*
Z	Size of a data packet
T	Round-trip propagation delay
RTT_{min}	Minimum measured RTT
RTT_n	measured RTT n^{th}
p	BBR's parameter: pacing gain
w	BBR's parameter: congestion window gain
Case (a), (b) with superscript 'a', 'b'	
P_n^a, P_n^b	Pacing rate in RTT_n
W_n^a, W_n^b	Congestion window in RTT_n
B_n^a, B_n^b	Estimated bandwidth in RTT_n
I_n^a, I_n^b	Delivery interval in RTT_n
D_n^a, D_n^b	Delivery rate in RTT_n
K_n^a, K_n^b	Number of data packets delivered in I_n^a or I_n^b
$sndus_n^a$, $sndus_n^b$	Sending interval at *SND* in RTT_n
$ackus_n^a$, $ackus_n^b$	Acknowledging interval at *SND* in RTT_n
G_n^a, G_n^b	Data packets' pre-defined gap at *SND* in RTT_n
A_n^a, A_n^b	Gap of ACKs at *SND* in RTT_n

III. Analysis of simplified BBR's performance

In this section, we consider the VM scheduling impact on a sender because from our experiments, the VM scheduling impact on a receiver is little. Thus, in our analysis, the sender is a VM sharing the same physical CPU with other VMs, and the receiver is a physical machine. Depending on several factors at the sender, such as virtualization technologies, VM schedulers, and the number of VMs sharing the same physical CPUs, the VM scheduling impact at a sender *SND* can be divided into three levels: case (a) when there is *little/no* VM scheduling, case (b) when *heavy* VM scheduling happens, and the medium case (or the mixed case of case (a) and (b)) when VM scheduling happens rarely for some periods and happens heavily for other periods. We are interested in analyzing the VM scheduling impact on BBR's control loop for cases (a) and (b). To do this, we firstly analyze each element of BBR's control loop - the pacing rate, delivery rate, and estimated bandwidth - for one measured RTT in case (a) and secondly in case (b). Then we extend the analysis of both cases for a whole flow duration. We also provide a discussion for the medium case and study it in the experiments because the complexity of BBR's control loop and the unpredictable VM scheduling noise cause a more complicated impact in this case.

We consider a network consisting of a sender *SND* (a VM) and a receiver *RCV* (a physical machine) interconnected by a link with an available bandwidth β and a round-trip propagation delay T. Table I shows the notation of cases (a) and (b) with superscripts "a" and "b", respectively. Our analysis uses the constant-rate fluid cross-traffic model [19] that means the cross-traffic rate is constant, and thus the available bandwidth β remains constant at all times.

A. Simplified BBR for analysis

Since the original algorithm of BBR is complicated, we use a simplified version of BBR to simplify the analysis. (1) It spends an infinite amount of time in *ProbeBW*; (2) It accurately estimates RTT_{min} when exiting from *Drain* and entering *ProbeBW*; (3) It sends multiple data packets in each RTT_n, which is the time interval between the send time of the first packet in one RTT and the time *SND* receives the ACK of this packet. Upon receiving ACKs at each each RTT_n, it estimates the delivery rate, current bandwidth and pacing rate based on the information of the first packet and its ACK. This simplified BBR algorithm is presented in Pseudocode 1. Please refer to [1] for a more detailed algorithm of BBR. Under the unpredictable noises created by VM scheduling, this simplified BBR allows us to capture the fundamental reason for the negative impact of VM scheduling on the BBR's control loop. Note that we only use the simplified BBR in the analysis, and we use the original versions of BBR for all the experiments in the paper.

B. BBR's control loop in case (a) for one measured RTT

Fig. 3 presents simplified BBR operating in case (a) where there is little/no VM scheduling impact on *SND*.

Pseudocode 1 Simplified BBR in *ProbeBW*

```
1: Upon receiving ACKs:
2: At each $RTT_n$
3:     Estimate delivery rate
4:     Estimate current bandwidth
5:     Estimate pacing rate for $RTT_{n+1}$
6:     Estimate cwnd for $RTT_{n+1}$
```

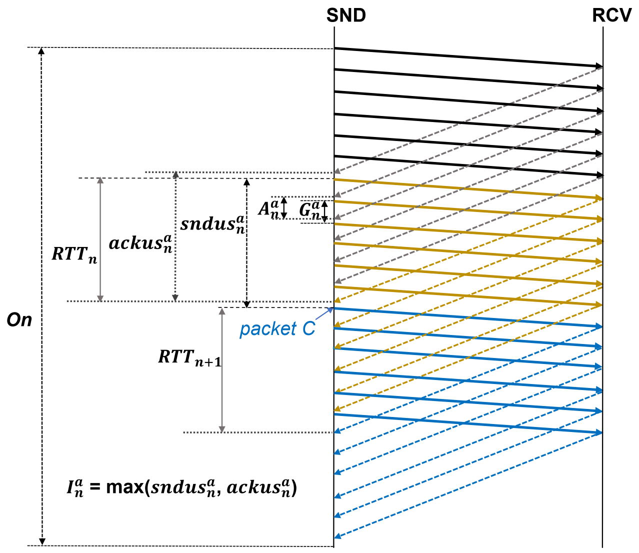


Fig. 3. One possibility of case (a): *SND* experiences little/no VM scheduling.

1) Pacing rate and pre-defined gap: BBR uses the pacing rate to control the rate at which data packets are sent to the network by spacing out two consecutive data packets with a pre-defined sending gap. Given a pacing rate P_n^a in RTT_n, *SND* sends data packets of size Z to *RCV* at the pre-defined sending gap G_n^a that is computed as:

$$G_n^a \equiv \frac{Z}{P_n^a} \tag{1}$$

2) Delivery interval: The delivery interval I_n^a in RTT_n is computed using Eq. (2), which is the the maximum of the sending interval $sndus_n^a$ and the acknowledging interval $ackus_n^a$ in order to filter out the noise caused by ACK compression [20].

$$I_n^a \equiv \max(sndus_n^a, ackus_n^a) \tag{2}$$

The sending interval is the difference between the send time of a packet C (e.g., first blue packet in Fig. 3) and the send time of the packet that is most recently delivered before the transmission of packet C (e.g., first yellow packet in Fig. 3). The sending interval $sndus_n^a$ in RTT_n is computed using Eq. (3), where K_n^a is the number of packets delivered in the interval.

$$sndus_n^a \equiv K_n^a \times G_n^a \tag{3}$$

The acknowledging interval is the difference between the time *SND* receives the ACK preceding the transmission of packet C and the time *SND* receives the ACK preceding the transmission of the packet that is most recently delivered before the transmission of packet *C*. The acknowledging interval $ackus^a_n$ in RTT_n is computed using Eq. (4), where acknowledging gap A^a_n is the gap between two consecutive ACKs.

$$ackus^a_n \equiv K^a_n \times A^a_n \tag{4}$$

$$A^a_n \equiv \frac{Z}{\beta} \tag{5}$$

3) Delivery rate: BBR estimates the delivery rate D^a_n in RTT_n as follows:

$$D^a_n \equiv \frac{K^a_n \times Z}{I^a_n} \tag{6}$$

***Theorem** 1:* Under the constant-rate fluid cross-traffic model, D^a_n can be computed as follows during *ProbeBW*.

$$D^a_n = \begin{cases} \beta & \text{if } P^a_n \geq \beta, \forall n \\ P^a_n & \text{otherwise} \end{cases} \tag{7}$$

Proof: During *ProbeBW*, under the constant-rate fluid cross-traffic model, if $P^a_n \geq \beta, \forall n$, which means $G^a_n \leq A^a_n$, then we have $sndus^a_n \leq ackus^a_n$. From Eq. (2), we have $I^a_n = ackus^a_n$. Combining with Eqs. (5), (4) and (6), we have $D^a_n = \frac{K^a_n \times Z}{K^a_n \times \frac{Z}{\beta}} = \beta$. If $P^a_n < \beta, \forall n$, which means $G^a_n > A^a_n$, then we have $sndus^a_n > ackus^a_n$. From Eq. (2), we have $I^a_n = sndus^a_n$. Combining with Eqs. (1), (3), and (6), we have $D^a_n = \frac{K^a_n \times Z}{K^a_n \times \frac{Z}{P^a_n}} = P^a_n$. ■

4) Estimated bandwidth: BBR then estimates the current *available bandwidth of the flow* B^a_n in RTT_n as the maximum of the *delivery rates* in a time period of 10 RTTs. B^a_n is computed using Eq. (8).

$$B^a_n \equiv \max(D^a_{n'}), n' \in [n-9, n] \tag{8}$$

5) Pacing rate and congestion window: BBR updates the next *pacing rate* P^a_{n+1} and congestion window W^a_{n+1} as follows:

$$P^a_{n+1} \equiv B^a_n \times p \tag{9}$$

$$W^a_{n+1} \equiv B^a_n \times RTT_{min} \times w \tag{10}$$

where p and w are BBR's parameters indicating the pacing gain and the congestion window gain, respectively.

C. BBR's control loop in case (b) for one measured RTT

1) Heavy VM scheduling condition: Under the VM scheduling impact, *SND* experiences *On* periods and *Off* periods in which it is scheduled on and off the physical CPU, respectively. The *On* and *Off* periods are assumed to be constant in the analysis. The *SND's* on percentage Θ is computed using Eq. (11).

$$\Theta \equiv \frac{On}{On + Off} \tag{11}$$

In case (b), we consider heavy VM scheduling condition, which means that VM scheduling happens in every round-trip propagation delay, specifically, $T > On$ and $On + Off > T$.

Fig. 4 demonstrates one possibility of case (b). Intuitively, while sending packets at a given pacing rate, *SND* is scheduled

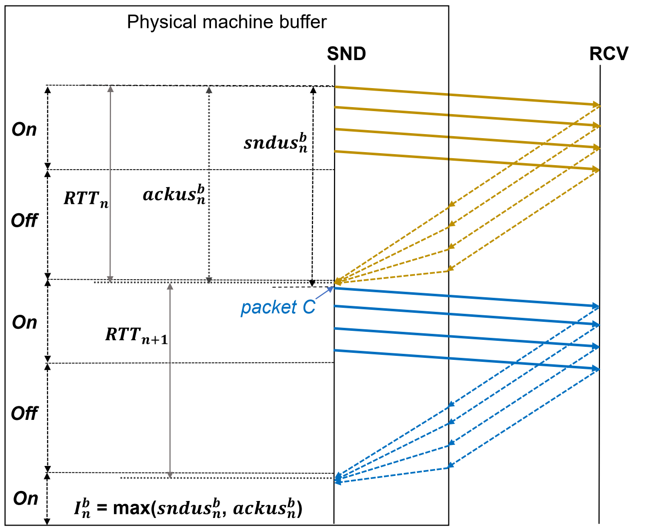


Fig. 4. One possibility of case (b): *SND* experiences heavy VM scheduling as VM scheduling occurs in every RTT.

off the physical CPU for an *Off* period. During the sleep time, *SND* cannot send any packets, and the ACKs from *RCV* are stored in the physical machine's buffer, which then are transmitted to *SND* later when it is scheduled on. Thus, the departure times of next data packets are delayed, BBR also mistakenly infers that ACKs are delayed because of network congestion while the actual cause is VM scheduling. This leads to the underestimates of the delivery rate, the current bandwidth, and the pacing rate for the next RTT.

2) Pacing rate and pre-defined sending gap: Given the pacing rate P^b_n in RTT_n, the pre-defined sending gap G^b_n between two consecutive packets of size Z in case (b) can be computed using Eq. (12).

$$G^b_n \equiv \frac{Z}{P^b_n} \tag{12}$$

3) Actual sending rate: *SND* can send at the pre-defined gap during *On* period. However, unlike case (a), under the heavy VM scheduling condition of case (b), due to an *Off* period, *SND* might send packets at the inflated gap rather than at the pre-defined gap, i.e., an actual sending gap might be inflated and then longer than the pre-defined gap. For example, we can see in Fig. 4 that black packets are sent at the same pre-defined sending gap during *On*, and the sending gap between the last black packet and the first yellow packet is inflated due to *SND's* sleep time. The actual sending rate S^b_n (i.e., the average rate in RTT_n) is then computed as:

$$S^b_n \equiv P^b_n \times \Theta \tag{13}$$

4) Delivery interval: I^b_n is computed as:

$$I^b_n \equiv max(sndus^b_n, ackus^b_n) \tag{14}$$

where $sndus^b_n$, and $ackus^b_n$ are the sending interval and acknowledging interval of case (b) in RTT_n.

5) Delivery rate: D_n^b is computed as:

$$D_n^b \equiv \frac{K_n^b \times Z}{I_n^b} \tag{15}$$

***Theorem** 2:* Under the constant-rate fluid cross-traffic model and the heavy VM scheduling condition, we have:

$$D_n^b = \begin{cases} \beta & \text{if } P_n^b \geq \frac{\beta}{O} \\ < \beta & \text{otherwise} \end{cases} \tag{16}$$

Proof: As both $sndus_n^b$ and $ackus_n^b$ might be inflated by *SND*'s sleep time which are *Off* periods in this case, BBR inaccurately estimates I_n^b as in Eq. (17).

$$I_n^b = On + Off \tag{17}$$

The actual acknowledging gap between two consecutive ACKs when they arrive at the physical machine A_n^{act} is computed as: $A_n^{act} = \frac{Z}{\beta}$.

If $P_n^b \geq \frac{\beta}{\Theta}$, then combining with Eq. (12), we have $G_n^b = \frac{Z}{P_n^b} \leq \frac{Z}{\beta} \times \Theta < A_n^{act}$ since $0 < \Theta < 1$. Then, the number of data packets K_n^b delivered during I_n^b can be computed as: $K_n^b = \frac{I_n^b}{A_n^{act}}$. Combining with Eq. (15), we have: $D_n^b = \frac{I_n^b}{A_n^{act}} \times \frac{Z}{I_n^b} = \beta$.

If $P_n^b < \frac{\beta}{\Theta}$, then the number of packets delivered during this interval is computed as: $K_n^b = \frac{P_n^b \times On}{Z}$. Combining with Eqs. (11) and (15), we have:

$$D_n^b = \frac{P_n^b \times On}{Z} \times \frac{Z}{I_n^b} = P_n^b \times \Theta < \beta \tag{18}$$

since $0 < \Theta < 1$.

As a result, we have $D_n^b = \beta$ if $P_n^b \geq \frac{\beta}{\Theta}$, and $D_n^b < \beta$ otherwise. ■

We can see that D_n^b *is sometimes lower than* β *because of two reasons related to VM scheduling*. First, because *SND* could not send the data packets in an *Off* period, K_n^b (numerator of D_n^b in Eq. (15)) is reduced and limited by the *On* period. Second, because *SND* could not receive the ACKs in an *Off* period, I_n^b (denominator of D_n^b) is inflated by the *Off* period.

6) Estimated bandwidth: The bandwidth is estimated as:

$$B_n^b \equiv \max(D_{n'}^b), n' \in [n-9, n] \tag{19}$$

7) Pacing rate and congestion window: The pacing rate and the congestion window of the next RTT are computed as:

$$P_{n+1}^b \equiv B_n^b \times p \tag{20}$$
$$W_{n+1}^b \equiv B_n^b \times RTT_{min} \times w \tag{21}$$

D. BBR's control loop in cases (a) and (b) for a flow duration

In this subsection, we extend the analysis in Section III-B and Section III-C from one RTT to a flow duration. That is, we compare BBR's performances of case (a) and case (b) over a long duration under the constant-rate fluid cross-traffic model. We are interested in BBR's behavior when $P_1^a = P_1^b = \beta$.

***Theorem** 3:* Under the constant-rate fluid cross-traffic model, if $P_1^a = \beta$ we have:

$$\lim_{n\to\infty} D_n^a = \beta \tag{22}$$
$$\lim_{n\to\infty} P_n^a = \beta \tag{23}$$

Proof: Under the constant-rate fluid cross-traffic model, if $P_1^a = \beta$, we have $D_n^a = \beta, \forall n$ from Theorem 1. Under the constant-rate fluid cross-traffic model, from Eq. (8) and Theorem 1, we have:

$$B_n^a = \begin{cases} \beta & \text{if } P_n^a \geq \beta, \forall n \\ P_n^a & \text{otherwise} \end{cases} \tag{24}$$

From Eqs. (24) and (9), we have:

$$P_{n+1}^a = \begin{cases} \beta \times p & \text{if } P_n^a \geq \beta, \forall n \\ P_n^a \times p & \text{otherwise} \end{cases} \tag{25}$$

Since the average pacing gain over a long duration is 1 [20], i.e., $\lim_{n\to\infty} p = 1$, we have $\lim_{n\to\infty} D_n^a = \beta$ and $\lim_{n\to\infty} P_n^a = \beta$. ■

Theorem 4: Under the constant-rate fluid cross-traffic model and the heavy VM scheduling condition, if $P_1^b = \beta$, we have

$$\lim_{n\to\infty} D_n^b = 0 \tag{26}$$
$$\lim_{n\to\infty} P_n^b = 0 \tag{27}$$

Proof: First, let's consider the relation between P_{n+1}^b and P_n^b when $p = 1$. The estimated bandwidth B_n^b can be obtained using Eq. (28) based on Theorem 2 and Eq. (19) under the constant-rate fluid cross-traffic model and the heavy VM scheduling condition.

$$B_n^b = D_n^b = \begin{cases} \beta & \text{if } P_n^b \geq \frac{\beta}{\Theta} \\ P_n^b \times \Theta < \beta & \text{otherwise} \end{cases} \tag{28}$$

Pacing rate P_{n+1}^b can then be obtained using Eq. (20). That is, $P_{n+1}^b = B_n^b \times p = B_n^b$.

Next, let's consider the limiting case $\lim_{n\to\infty} P_{n+1}^b$ with initially $P_1^b = \beta$. Because $\Theta < 1$ and $P_n^b < \frac{\beta}{\Theta}$ for all n, we have

$$\lim_{n\to\infty} P_{n+1}^b = \lim_{n\to\infty} P_n^b \times \Theta \tag{29}$$
$$= \lim_{n\to\infty} P_{n-1}^b \times \Theta^2 \tag{30}$$
$$= \dots \tag{31}$$
$$= \lim_{n\to\infty} P_1^b \times \Theta^n \tag{32}$$
$$= \beta \times \lim_{n\to\infty} \Theta^n \tag{33}$$
$$= 0 \tag{34}$$

Similarly, we have $\lim_{n\to\infty} D_{n+1}^b = 0$ with initially $P_1^b = \beta$. ■

Fig. 5 shows an example of this theorem, where initially $P_1^b = \beta = 150$ Mbps and $\Theta = 30\%$. After the first RTT, the pacing rate becomes $P_1^b \times \Theta = 45$ Mbps. The pacing rate reduces iteratively after every RTT, and finally, reduces to zero.

TABLE II

WHEN $P_1^a = P_1^b = \beta$: IN CASE (A), BBR ACCURATELY ESTIMATES PACING RATE, DELIVERY RATE. IN CASE (B), THE UNDERESTIMATE OF BBR'S CONTROL LOOP DUE TO VM SCHEDULING CAN CAUSE ITS THROUGHPUT TO ITERATIVELY REDUCE TO ZERO.

Case	Initially	Final pacing rate	Final delivery rate
(a)	$P_1^a = \beta$	$\lim_{n\to\infty} P_n^a = \beta$	$\lim_{n\to\infty} D_n^a = \beta$
(b)	$P_1^b = \beta$	$\lim_{n\to\infty} P_n^b = 0$	$\lim_{n\to\infty} D_n^b = 0$

Takeaway: Over a long duration, starting with the same pacing rate at the bandwidth of the link, while *SND*'s pacing rate and delivery rate in case (a) remain equal to this bandwidth, the pacing rate and delivery rate in case (b) drop to zero. Existing works studying traditional TCPs show that VM scheduling only reduces traditional TCPs' throughput to a certain rate. However, in this section, *we have shown that VM scheduling can bring BBR's throughput down to zero, and the fundamental reason is because of BBR's control loop*. Under heavy VM scheduling condition, the underestimation of BBR's control loop grows exponentially over the flow duration causing its throughput reduce to zero in cloud networks. Table II summarizes Theorems 3 and 4 discussed in Section III-D.

E. Discussion on Medium VM scheduling impact case

In this case, BBR can still deliver packets at a similar rate as in case (a) in the RTTs that are not affected by VM scheduling, and only in RTTs that are affected by VM scheduling BBR cannot deliver at such a rate. This means, sometimes the maximum delivery rate BBR can achieve in such unaffected RTTs is also β, given that BBR sends at the link's bandwidth when entering *ProbeBW*. Thus, BBR in this case sometimes estimates the current bandwidth accurately and thus sometimes sets the pacing rate accurately to the link's bandwidth, and as a result its throughput is affected by VM scheduling and it achieves a lower average throughput than case (a) and higher than case (b).

IV. EXPERIMENTAL VERIFICATION

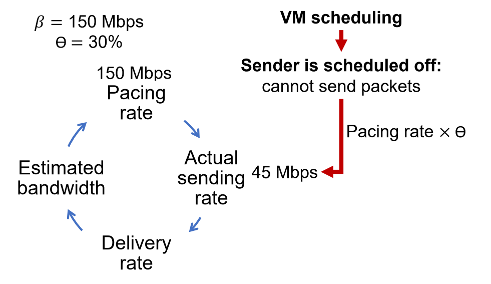
A. Verification goals

In this section, we design and run experiments to verify if BBR's actual behavior is consistent with our analysis. That is, to answer the following questions:

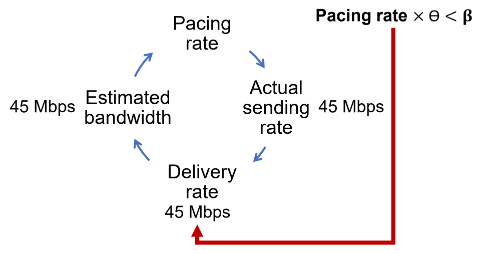
Question (1): *Given a pacing rate in RTT_n, how does BBR behave in the next RTT if SND is in case (a) and case (b)?*

Question (2): *Given that the pacing rate when BBR enters ProbeBW is at the link's bandwidth, how does BBR behave over the flow duration in case (a) and case (b)?*

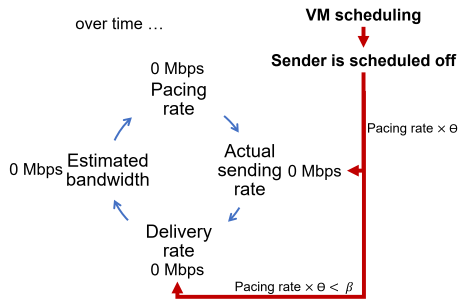
These experiments are conducted on NS3 [21] where we simulate the VM scheduling effect in a controlled environment, and on a testbed where our *SND* is a VM created on a server with Xen [22] 4.9 hypervisor so *SND* experiences the real VM scheduling effect with some unpredictable noises. We use Xen since Amazon EC2 also uses a highly customized Xen [23] with credit scheduler [10] for their VMs. The credit scheduler controls *On* and *Off* periods dynamically based on a VM's credit balance. This can create a combination of cases (a) and (b) with a mixed impact on *SND*. Different from the testbed experiments, running experiments on NS3 allows us to exactly control these periods, which ensures that *SND* in NS3 experiments is always under case (a) or (b). Therefore, we run experiments for question (1) on NS3 and run experiments for question (2) on the testbed. Experimental results for question (1) are to verify Theorems 1 and 2, and experimental results for question (2) are to verify Theorems 3 and 4.



(a) Under VM scheduling, *SND*'s actual sending rate is lower than pacing rate



(b) Under VM scheduling, BBR inaccurately estimates current bandwidth



(c) Under VM scheduling, over time, BBR's control loop causes its throughput to iteratively drop to zero.

Fig. 5. An example of Theorem 4: BBR starts *ProbeBw* at a pacing rate of the link bandwidth of 150 Mbps. Under heavy VM scheduling, *SND* cannot send and receive packets during the sleep time, resulting in a lower delivery rate than the link bandwidth. This causes the inaccuracy in estimating delivery rate, current bandwidth and the pacing rate of next packets. Over time, BBR's control loop causes its poor performance in cloud networks.

B. Experiment setup

The general setup of these experiments is as follows. We use the same topology shown in Fig. 6 for all of our experiments in this section. Between *N1* and *N2*, the link capacity is 200 Mbps with One-way delay (OWD) of 10 ms. The capacities and the OWDs between either *N1* or *N2* and other nodes are 200 Mbps

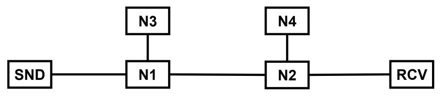


Fig. 6. Network topology for simulation and testbed experiments.

TABLE III
SETUP OF NS3 EXPERIMENTS.

Case	*On*	*Off*	Group 1 P_n^a=P_n^b=165 Mbps	Group 2 P_n^a=P_n^b=135 Mbps
(a)	30	10	1a	2a
(b)	10	70	1b	2b

and 2 ms, respectively. The UDP cross-traffic from N3 to N4 is 50 Mbps. So, the actual available bandwidth β is 150 Mbps. The duration of each experiment is 20 s.

NS3 experiments: We use the implementation [24] of BBRv1 on NS3 for these experiments. Since BBR's control loop can affect its performance, we fix its pacing rate at 165 Mbps in experiments for cases where $P_n^a \geq \beta$ and $P_n^b \geq \beta$, called *Group 1* of NS3 experiments, and at 135 Mbps in experiments for cases where $P_n^a < \beta$ and $P_n^b < \beta$, called *Group 2* of NS3 experiments. There are a total of 4 sub-groups of NS3 experiments as summarized in Table III: sub-groups 1a and 2a are when *SND* is in case (a) with *On*=30 ms and *Off*=10 ms, and sub-groups 1b and 2b are when *SND* is in case (b) with *On*=10 ms and *Off*=70 ms.

Testbed experiments: We run experiments with BBRv1 for Ubuntu 16.04. In order to avoid the impact of the Startup and Drain states on BBR's performance, when BBR first enters *ProbeBW* state, we fix its pacing rate at β. This ensures that we have $P_1^a = P_1^b = \beta$ as mentioned in Section III-D.

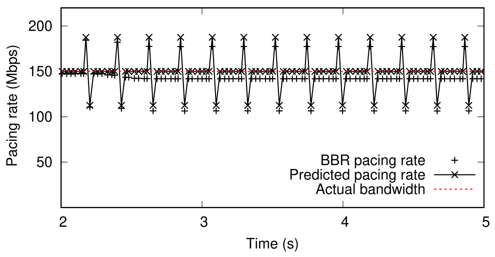
C. NS3 experiments

To simulate a VM sleep time, we modify the queuing model of NS3 such that node *N1* only forwards packets during *On*, and delays forwarding packets during *Off*.

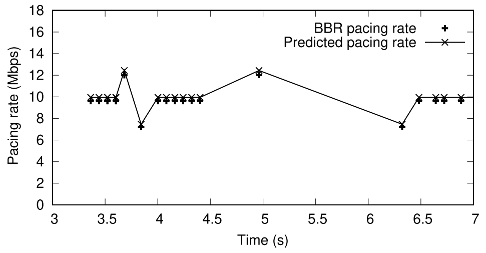
1) Group 1 with $P_n^a = P_n^b = 165$ Mbps $> \beta = 150$ Mbps: Fig. 7 presents the next-RTT pacing rates of sub-groups 1a and 1b computed by original BBR in comparison with the predicted next-RTT pacing rates computed by Eqs. (6), (8), and (9) for case (a), Eqs (18), (19) and (20) for case (b) when Θ is given. We can see that our model can reasonably accurately predict the trend of the actual next-RTT pacing rates.

Fig. 7(a) shows that the next-RTT pacing rates in sub-group 1a are around β. This is because it has little VM scheduling (with a high On percentage $\Theta = 30/(30+10) = 75\%$) and its given pacing rate P_n^a is higher than β according to Theorem 1.

Fig. 7(b) shows that the next-RTT pacing rates in sub-group 1b are much lower than β. This is because it has heavy VM scheduling (with a low On percentage $\Theta = 10/(10+70) = 12.5\%$) and its given pacing rate P_n^b is much lower than $\beta/\Theta = 150/0.125 = 1200$ according to Theorem 2.



(a) sub-group 1a: Little/No VM scheduling



(b) sub-group 1b: Heavy VM scheduling

Fig. 7. Group 1 of NS3 experiments: With $P_n^a = P_n^b = 165$ Mbps $\geq \beta = 150$ Mbps, $\forall n$, and given Θ, the actual next-RTT pacing rates of BBR are consistent with these calculated by Theorems 1 and 2 for cases (a) and (b), respectively. Note that with heavy VM scheduling in case (b), BBR's next-RTT pacing rates are much lower than β.

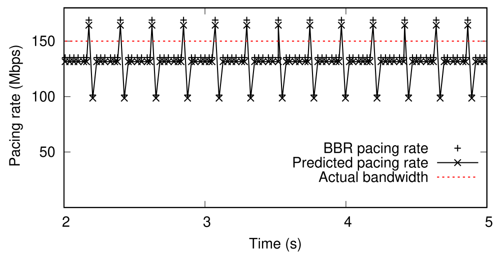
2) Group 2 with $P_n^a = P_n^b = 135$ Mbps $< \beta$: Fig. 8 shows the next-RTT pacing rates of sub-groups 2a and 2b computed by original BBR and the predicted next-RTT pacing rates computed by Eqs. (6), (8), and (9) for case (a), Eqs. (18), (19) and (20) for case (b) when Θ is given. We can see that our model can reasonably accurately predict the trend of actual next-RTT pacing rates.

Fig. 8(a) shows that the next-RTT pacing rates in sub-group 2a are lower than β. This is because its given pacing rate P_n^a is lower than β according to Theorem 1. Fig. 8(b) shows that the next-RTT pacing rates in sub-group 2b are much lower than β. This is because it has heavy VM scheduling (with a low On percentage $\Theta = 12.5\%$) and its given pacing rate P_n^b is much lower than $\beta/\Theta = 1200$ according to Theorem 2.

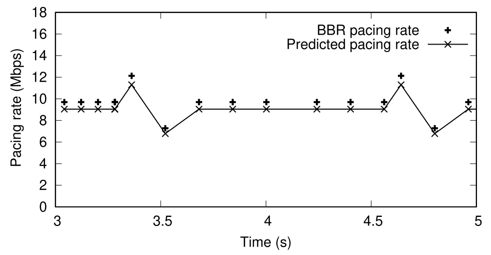
Takeaway: The experimental results in Groups 1 and 2 verify our analysis for one RTT described in Theorems 1 and 2 that BBR's control loop is negatively affected by VM scheduling. There is a slight difference between BBR's actual pacing rate and the predicted rate because we use the simplified BBR in our analysis, while we use the original BBRv1 in those experiments.

D. Testbed experiments

A Mininet [25] network including *N1*, *N2*, *N3*, *N4* and *RCV* is created on a physical server which is connecting to the Xen server that hosts our *SND*. N1 and N2 has OWD of 20 ms. *SND* shares a physical CPU with four other VMs. We first run lookbusy [26] on *SND* for several hours to consume a high number of its credits. Then we use Iperf [27] to start sending packets from *SND* to *RCV* while other VMs are running lookbusy. This ensures that all VMs are competing for CPU



(a) sub-group 2a: Little/No VM scheduling



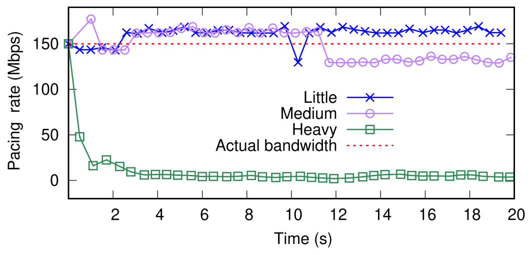
(b) sub-group 2b: Heavy VM scheduling

Fig. 8. Group 2 of NS3 experiments: With $P_n^a = P_n^b = 135$ Mbps $< \beta = 150$ Mbps, $\forall n$, and given Θ, the actual next-RTT pacing rates of BBR are consistent with these calculated by Theorems 1 and 2 for cases (a) and (b), respectively. Note that with heavy VM scheduling in case (b), BBR's next-RTT pacing rates are much lower than β.

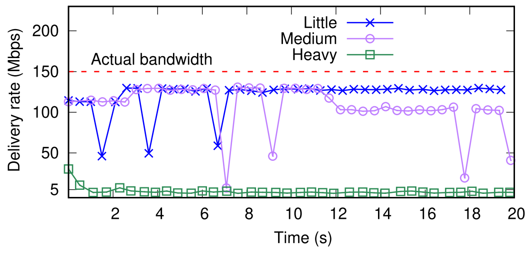
cycles, and that our *SND* is not given a higher priority than other VMs during the experiment time. Different heaviness levels of VM scheduling on *SND* can be implemented by varying the time slice and weight parameters of Xen's default scheduler, and the number of VMs sharing the same physical CPU on Xen server. The time slice is the scheduling quantum. That is, once a VM is scheduled off according to its credits, it has to wait for at least a time slice to be scheduled on again. A VM with a higher weight has a higher credit balance, and it is allocated with higher CPU time than a VM with a lower weight. In all Mininet experiments, we use the default time slice of 30 ms, and vary the weight of *SND* from the default value 256, to 128 and 64, while other VMs has the fixed weight of 256.

We can see from Fig. 9(a) that BBRv1's pacing rate is around β in case (a) (label "Little", implemented using a weight of 256), while its pacing rate greatly reduces in case (b) (label "Heavy", implemented using a weight of 64) because the average on percentage of *SND* is only about 1%. This is consistent with Theorems 3 and 4. The pacing rate of the medium case (label "Medium", implemented using a weight of 128) varies around the bandwidth β since VM scheduling occurs only in some RTTs, and is consistent with what we discussed in Section III-E. We also show in Figures 9(b) and 9(c) delivery rates and estimated bandwidths of these three cases. We can see that BBR's average delivery rate in the medium case is smaller than that of case (a) and greater than that of case (b), specifically, 2 Mbps $<$ 100 Mbps $<$ 120 Mbps).

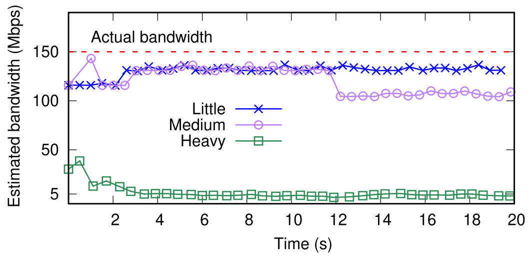
Takeaway: The experimental results verify our analysis in



(a) Pacing rates of different VM scheduling cases



(b) Delivery rates of different VM scheduling cases



(c) Estimated bandwidths of different VM scheduling cases

Fig. 9. Testbed experiments of three cases all starting with $P_1^a = P_1^b = \beta$: "Little" VM scheduling (i.e., case a), "Medium" VM scheduling, and "Heavy" VM scheduling (i.e., case b). The results are consistent with Theorems 3 and 4. That is, BBR under little VM scheduling can reasonably accurately estimate the bandwidth, delivery rate, and pacing rate. However, BBR under heavy VM scheduling iteratively underestimates the bandwidth, delivery rate, and pacing rate, and finally has very poor performance.

Theorems 3 and 4: over a flow duration, the underestimate of BBR's control loop due to heavy VM scheduling iteratively reduces its throughout and finally to almost zero in cloud networks.

V. Proposed patchedBBR

From the analysis in Section III, we propose a patch for BBR. It integrates BBR's model with *SND's* on percentage that captures the VM scheduling impact on the delivery interval to improve BBR's performance in cloud networks.

A. *Design options for the patch*

There are two possible design options to improve BBR's performance in cloud networks: *Option 1*: Achieve higher throughput than BBR and operate at similar measured minimum RTT as BBR; *Option 2*: Achieve the maximum throughput at the link's bandwidth β. We choose option 1 for our proposed BBR patch for the following reason. According to Theorem 2, the pacing rate for case (b) needs to be much higher than β in order to achieve the maximum throughput.

Pseudocode 2 patchedBBR in *ProbeBW*

```
1: Additional parameters: $\Theta$
2: Upon receiving ACKs:
3: At each $RTT_n$
4:     Estimate SND's on percentage using Eq. (11)
5:     Estimate deliver rate $D^*_n$ using Eq. (35)
6:     Estimate expected bandwidth $B^*_n$ using Eq. (36)
7:     Estimate pacing rate $P^*_{n+1}$ using Eq. (37)
8:     Estimate cwnd $W^*_{n+1}$ using Eq. (39)
```

However, a pacing rate much higher than β leads to an increased RTT. Because one of the main design goals of BBR is to operate around the minimum RTT, we choose option 1.

Thus, our proposed BBR patch is designed to achieve higher throughput than BBR and more accurately estimate β but does not always achieve the maximum throughput and does not always accurately estimate β. To implement this patch, we modify BBR's control loop by increasing the pacing rate as well as the window size during *On* periods, but only when BBR increases its pacing rate with pacing gain $p > 1$, and when *SND* is in the medium case or the heavy case. This ensures that *SND* can send more packets at a higher rate to the network, thus achieves a higher delivery rate while operating at a similar measured minimum RTT to BBR.

B. How patchedBBR works

patchedBBR operates similar to BBR during *Startup* and *Drain*. Once entering *ProbeBW*, patchedBBR estimates *SND's* on percentage over RTT_n. It then incorporates this information into the estimates of the current bandwidth, pacing rate and congestion window for the next RTT. The algorithm of patchedBBR is presented in Pseudocode 2, and its variables are presented with superscript '$*$'.

C. The model of patchedBBR in ProbeBW

1) Delivery rate: The actual delivery rate of patchedBBR is computed as::

$$D^*_n \equiv \frac{K^*_n \times Z}{I^*_n} \tag{35}$$

2) Estimated bandwidth: patchedBBR estimates the bandwidth during *ProbeBW* as:

$$B^*_n \equiv \max\left(\frac{D^*_{n'}}{\Theta}\right),\ n' \in [n-9, n] \tag{36}$$

Since BBR's implementation uses a timer to schedule departure times of next data packets, we add a function to detect and estimate when and for how long VM scheduling happens. More specifically, if the actual sending gap between two consecutive packets is 1 ms longer than the pre-defined gap, it is considered as *Off*; otherwise, it is considered as *On*. We use Eq. (11) to compute *SND's* on percentage Θ.

3) Pacing rate: With three VM scheduling condition levels, we consider *SND* in case (a) if its on percentage $\Theta > 2/3$, in case (b) if $\Theta \leq 1/3$, and in the medium case if $1/3 < \Theta \leq 2/3$. patchedBBR adjusts its next-RTT pacing rate as:

$$P^*_{n+1} \equiv \begin{cases} \frac{B^*_n \times p}{\Theta} & \text{if } p > 1 \ and \ \Theta \leq 2/3 \\ B^*_n \times p & \text{otherwise} \end{cases} \tag{37}$$

4) Congestion window: patchedBBR uses α to increase the congestion window when *SND* is in the medium case or the heavy case. α and the congestion window are currently computed using Eq. (38) and Eq. (39), respectively.

$$\alpha = \begin{cases} 2 & \text{if } \Theta \leq \frac{1}{3} \\ 0 & \text{if } \Theta > \frac{2}{3} \\ 1 & \text{otherwise} \end{cases} \tag{38}$$

$$W^*_{n+1} \equiv B^*_n \times RTT_{min} \times (w + \alpha) \tag{39}$$

Discussion: If Θ is close to 100%, patchedBBR has a similar delivery rate, estimated bandwidth, pacing rate, and congestion window as original BBR. If Θ is much lower than 100%, patchedBBR has a higher estimated bandwidth calculated by Eq. (36) than original BBR in order to more accurately estimate the available bandwidth β, and has a higher pacing rate calculated by Eq. (37) than original BBR but only when $p > 1$ in order to improve the throughput while maintaining similar RTT as original BBR.

D. Performances of patchedBBR and BBR in case (b) for a flow duration

In this subsection, we analytically compare delivery rates and estimated bandwidths of patchedBBR and BBR in cases (a) and (b) for a flow duration.

***Theorem** 5:* Under the constant-rate fluid cross traffic model, when patchedBBR and BBR start with the same pacing rate of β, patchedBBR performs as well as BBR in case (a) and better in case (b) in terms of bandwidth estimation and pacing rates.

Proof: First, let's consider case (a) with $P^{*a}_1 = P^a_1 = \beta$ and $\Theta \approx 1$. In this case, we have $P^{*a}_n \approx P^a_n = \beta$ and $B^{*a}_n \approx B^a_n = \beta$ for $\forall n$ from Eqs. (35), (37), (39), and Theorem 1. The approximation becomes more accurate when Θ is closer to 1. That is, patchedBBR achieves a similar pacing rate and estimates bandwidth as accurately as BBR in case (a).

Next, let's consider case (b) with $P^{*b}_1 = P^b_1 = \beta$ and $0 < \Theta \leq 1$. In the first RTT, we have $D^{*b}_1 = D^b_1 = \beta \times \Theta$ from Theorem 2. Thus, we have $B^{*b}_1 = \frac{\beta \times \Theta}{\Theta} = \beta \geq B^b_1 = \beta \times \Theta$ from Eqs. (19) and (36). Then, we have $P^{*b}_2 \geq P^b_2$ for any p from Eqs. (37) and (20), especially if $p > 1$, $P^{*b}_2 = p \times \frac{\beta}{\Theta} \geq P^b_2 = p \times \beta \times \Theta$. Starting from the second RTT, we can see that $P^{*b}_n \geq P^b_n$ and $B^{*b}_1 \geq B^b_1$ for $\forall n \geq 2$ based on Theorem 2. Therefore, compared to BBR, patchedBBR can send more packets at a higher rate to the network during *On* periods, which compensates for the number of packets that *SND* could not send during *Off* periods. As a result, patchedBBR achieves a higher throughput than BBR and more accurately estimate the bandwidth in case (b). ■

VI. Evaluation of patchedBBR

A. Experiment setup

We conduct experiments on both Mininet testbed and Amazon EC2. The testbed experiments are to verify our analysis in Section V-D so the pacing rates of patchedBBR and BBR are set such that $P^*_1 = P^a_1 = P^b_1 = \beta$. We use the same network topology shown in Fig. 6 and the same setting for the Mininet testbed mentioned in Section IV-D. The experiments on the public cloud are to compare the performance of patchedBBR with BBR in real-world environments. We use a t2-nano VM in EC2 as *SND* connecting to a physical server as *RCV* with a link of 100 Mbps capacity. T2 VMs are among the most popular VMs in Amazon EC2 since they have the lowest cost but can burst above their baselines when needed [28]. As EC2 adopts multi-hypervisor including Xen with a credit scheduler, we also use lookbusy [26] to consume *SND's* credit balance and to ensure that *SND* is competing for CPU cycles with other VMs. Each experiment is run 5 times and all results are shown in figures with 90% confidence interval bars, with labels “Little” for case (a), “Heavy” for case (b), and “Medium” for the medium case. In this section, we present only the result of patchedBBR implemented by modifying BBRv1, because BBRv2 is still in the developing and testing stage at Google.

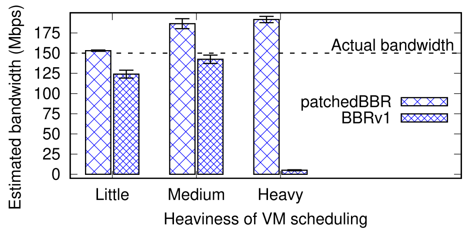
B. Experimental results

1) Mininet testbed: When $P_1^* = P_1^a = P_1^b = \beta$, Figs. 10(a) and 10(b) show patchedBBR's performance and BBR's in terms of the average estimated bandwidth and average throughput when *SND* is experiencing different levels of VM scheduling impact as the weight changes in the testbed. We can see that *patchedBBR reports more accurate estimated bandwidth than BBR in all cases*, especially in case (b) (i.e., “Heavy”). We also see that *patchedBBR achieves higher throughput than BBR in all cases*, especially in case (b) with 100 Mbps while that of BBR is less than 10 Mbps. This is consistent with Theorem 5. Note that patchedBBR does not achieve the maximum throughput in case (b). This is because patchedBBR is designed to achieve a similar RTT as BBR according to design option 1 as discussed in Section V-A, and we can see from Fig. 10(c) that patchedBBR has similar average RTTS. Note that Fig. 10(c) shows that both patchedBBR and BBR have longer average measured RTTs in case (b) and the medium case than in case (a). This is because these measured RTTs are inflated by *SND's* sleep time.

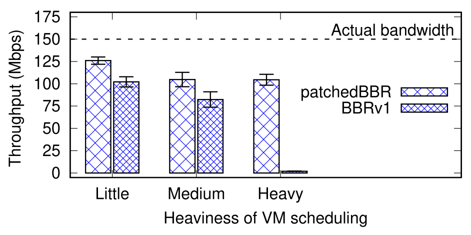
We also conduct experiments for cases without VM scheduling, and results show that patchedBBR performs as well as BBR in terms of estimated bandwidth, throughput, measured RTTs, and fairness among its flows. Due to space limit, we only present the experimental result of patchedBBR's fairness in Fig. 11 with five patchedBBR flows.

2) Public cloud: Fig. 12 shows the average estimated bandwidths, average throughputs and average measured RTTs of patchedBBR and BBR on EC2. We implement different levels of VM scheduling on *SND's* by varying its credit balance. In case (a), “Little1” and “Little2” are with credit balance of 25-30 and 12-15, respectively. In case (b), “Heavy1” and “Heavy2” are with credit balance of 0.4-1 and 0.15-0.2, respectively. We can see that *patchedBBR more accurately estimates the bandwidth and achieves higher throughput than BBR in all EC2 experiments*, especially in case (b). BBR performs very poorly in case (b) because VM scheduling noise not only causes BBR to inaccurately estimate the bandwidth in Startup but also affects its control loop in ProbeBW. Unlike BBR, patchedBBR can detect VM scheduling and increases its pacing rate according to *SND's* on percentage. This leads to more accurate estimates of the control loop's elements of patchedBBR. In Fig. 12(c), we see that patchedBBR has similar measured average RTT as BBR in all experiments.

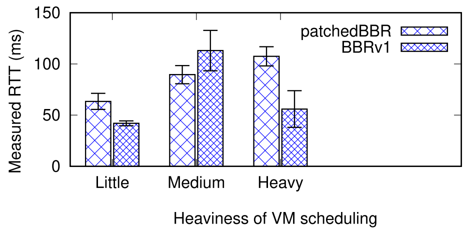
Takeaway: Both testbed and EC2 experiment results show that patchedBBR achieves our design goal that it achieves higher throughput than BBR while maintaining a similar RTT as BBR, especially with heavy VM scheduling.



(a) Estimated bandwidths



(b) Throughputs



(c) Average measured RTTs

Fig. 10. Testbed experiments with different levels of VM scheduling with labels “Little” for case (a), “Heavy” for case (b), and “Medium” for the medium case. In case (b), patchedBBR more accurately estimates bandwidth and achieves higher throughput than BBR. patchedBBR has similar average measured RTTs as BBR in all cases, and both have longer average RTTs in the medium case and case (b) as these are inflated by *SND's* sleep time.

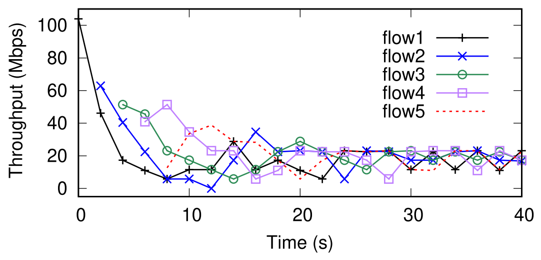
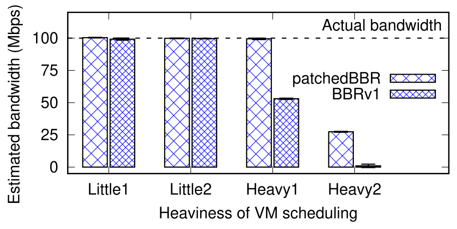
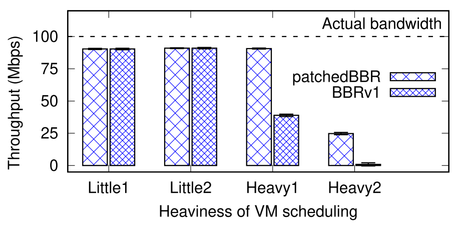


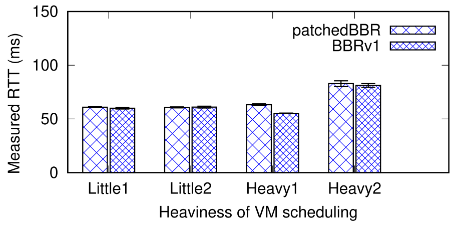
Fig. 11. patchedBBR maintains fairness between its flows as BBR: Five patchedBBR flows converge and share bandwidth fairly when competing over the same link.



(a) Estimated bandwidths



(b) Throughputs



(c) Average measured RTTs

Fig. 12. EC2 experiments with different credit balance as different levels of VM scheduling. Case (a): "Little1" and "Little2" with high credit balance. Case (b): "Heavy1" and "Heavy2" with low credit balance. In case (b), patchedBBR more accurately estimates bandwidth and achieves higher throughput than BBR, while having similar average RTTs in all cases.

VII. Conclusion

In this paper, we provide an analysis of BBR's performance as a case study for the model-based TCP class's performance in cloud networks. We prove analytically and experimentally that it performs poorly in terms of bandwidth estimation and throughput in cloud networks because of VM scheduling impact on its control loop. We also propose a patch for BBR, which integrates *SND's* on percentage into BBR's model. Experimental results show that patchedBBR performs as well as BBR when there is little/no VM scheduling, and achieves higher throughput than BBR while maintaining similar measured minimum RTT as BBR when heavy scheduling happens.

ACKNOWLEDGMENT

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

References

[1] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. pp. 58–66, Feb. 2017.

[2] N. Cardwell, Y. Cheng, Y. S. Hassas, J. Priyaranjan, S. Yousuk, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao, M. Mathis, and V. Jacobson, "BBRv2: A model-based congestion control performance optimizations," https://datatracker.ietf.org/meeting/106/materials/slides-106-iccrg-update-on-bbrv2.

[3] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," *RFC 5681*, September 2009.

[4] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating System Review*, vol. 42, no. 5, pp. 64–74, July 2008.

[5] M. Alizadeh, A. Greenberg, D. Maltz, and J. P. et al., "Data center TCP (DCTCP)," in *Proceedings of ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[6] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The Great Internet TCP Congestion Control Census," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019.

[7] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, 2017, pp. 1–10.

[8] E. Atxutegi, F. Liberal, H. K. Haile, K. Grinnemo, A. Brunstrom, and A. Arvidsson, "On the use of TCP BBR in cellular networks," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 172–179, March 2018.

[9] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," *RFC 3782*, April 2004.

[10] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of TCP BBR congestion control," in *Proceedings of IFIP Networking Conference and Workshops*, 2018, pp. 1–9.

[11] M. Zhang, M. Polese, M. Mezzavilla, J. Zhu, S. Rangan, S. Panwar, and M. Zorzi, "Will TCP work in mmWave 5G Cellular Networks?" *IEEE Communications Magazine*, vol. 57, no. 1, pp. 65–71, 2019.

[12] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Modeling BBR's interactions with loss-based congestion control," in *Proceedings of the Internet Measurement Conference*, New York, NY, 2019, p. 137–143.

[13] 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.

[14] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," in *Proceedings of IEEE INFOCOM*, 2014.

[15] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu, "vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload," in *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2010, pp. 1–11.

[16] 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*, New York, NY, 2011, pp. 1–24.

[17] 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.

[18] C. Neal, J. Van, Y. Cheng, Y. S. Hassas, V. Victor, S. Ian, J. Priyaranjan, S. Yousuk, and W. David, "Model-based network congestion control," *Technical Disclosure Commons*, Mar. 2019.

[19] X. Liu, K. Ravindran, and D. Loguinov, "A queueing-theoretic foundation of available bandwidth estimation: Single-hop analysis," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 918–931, 2007.

[20] Y. Cheng, N. Cardwell, S. H. Yeganeh, and V. Jacobson, "Delivery rate estimation, draft-cardwell-iccrg-delivery-rate-estimation-00," *IETF Internet Draft*, July 2017.

[21] Network Simulator 3, https://www.nsnam.org/.

[22] 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.

[23] Amazon Web Services: Overview of Security Processes., https://d0.awsstatic.com/whitepapers/aws-security-whitepaper.pdf.

[24] M. Claypool, J. W. Chung, and F. Li, "Bbr: An implementation of bottleneck bandwidth and round-trip time congestion control for NS-3," in *Proceedings of Workshop on NS-3*. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–8.

[25] Mininet, http://mininet.org//.

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

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

[28] Amazon Web Services, Inc., "Amazon EC2 Instances," http://aws.amazon.com/ec2/instance-types/.