

7-25-2019

Towards Efficient, Work-Conserving, and Fair Bandwidth Guarantee in Cloud Datacenters

Baraa Saeed Ali

Wayne State University, baraa@wayne.edu

Kang Chen

Southern Illinois University Carbondale, kchen@siu.edu

Imran Khan

Southern Illinois University Carbondale, imran.khan@siu.edu

Follow this and additional works at: https://opensiuc.lib.siu.edu/ece_articles

This work is licensed under a Creative Commons Attribution 4.0 License.

Recommended Citation


Ali, Baraa S., Chen, Kang and Khan, Imran. "Towards Efficient, Work-Conserving, and Fair Bandwidth Guarantee in Cloud Datacenters." *IEEE Access* 7 (Jul 2019): 109134 - 109150. doi:10.1109/ACCESS.2019.2930888.

This Article is brought to you for free and open access by the Department of Electrical and Computer Engineering at OpenSIUC. It has been accepted for inclusion in Articles by an authorized administrator of OpenSIUC. For more information, please contact opensiuc@lib.siu.edu.

Received June 11, 2019, accepted July 6, 2019, date of publication July 25, 2019, date of current version August 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2930888

Towards Efficient, Work-Conserving, and Fair Bandwidth Guarantee in Cloud Datacenters

BARAA SAEED ALI^{1*}, KANG CHEN¹², (Member, IEEE), AND IMRAN KHAN²

¹Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, USA

²Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901, USA

*Baraa Saeed Ali contributed to this paper only during his study at Southern Illinois University Carbondale (SIUC)

Corresponding author: Kang Chen (kchen@siu.edu)

This work was supported by the startup fund and the COPE fund from SIUC. All contributions are from SIUC.

ABSTRACT Bandwidth guarantee is a critical feature to enable performance predictability in cloud datacenters. This process is expected to achieve three requirements: work conservation, fairness, and simplicity. However, the distributed nature of datacenters raises significant challenges to attaining those requirements at the same time. In this paper, we propose an efficient approach that can satisfy the three requirements simultaneously. Our scheme takes advantage of multipath TCP (MPTCP) to generate explicit bandwidth guarantee (BG) traffic and work conservation (WC) traffic. We further prioritize the BG traffic over the WC traffic in the network fabric. Due to the priority setting, WC cannot harm bandwidth guarantees and thus is effectively supported. We show that the MPTCP fits this direction well but presents some new issues when the WC subflows own a low priority. We thus adapt the MPTCP to handle these issues through a customized scheduler (which strictly prioritizes BG subflow during packet scheduling) and adopting a large receive buffer. In addition, we enable tenants to share unused bandwidth fairly by managing the overall aggressiveness of the WC traffic. The proposed system can be easily implemented with commercial off-the-shelf servers and switches. We have implemented with the Linux kernel MPTCP for experiments. The extensive experiments in a small cluster (including one MapReduce experiment) and trace-driven simulations show that our scheme achieves the design goals effectively.

INDEX TERMS Datacenter network, bandwidth guarantee, work conservation, MPTCP, fairness.

I. INTRODUCTION

Cloud computing has become an emerging computing paradigm that can efficiently, scalably, and flexibly provide computing resources to tenants. It handles many problems faced by installing and maintaining dedicated computing infrastructures [1]. However, while computing and storage resources of different tenants are clearly isolated in current clouds, network bandwidth is generally shared among tenants in a best-effort manner [2]. Such a sharing model causes unpredictable network performance, which in turn affects the performance of tenant applications [3]. Consequently, tenants may have to rent the computing and storage resources for a longer period of time and thus pay more. This not only lowers the overall resource utilization efficiency of the cloud but also imposes a high hidden cost to tenants, which may discourage the migration to the cloud [4], [5]. As a result, it is desirable to provide isolated and guaranteed network bandwidth to tenants, just like the storage and computing resources.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Maaz Rehan.

Due to the limitation of current cloud network architecture, bandwidth guarantees are implemented by rate limiting at VMs. Particularly, when VMs sharing a link all send up to the allocated share (note that the sum of shares is smaller than the link capacity), everyone's guarantee is respected. However, such a static reservation strategy makes idled bandwidth not accessible to others, which would greatly lower the utilization efficiency of network bandwidths. To this end, we believe that the following three requirements need to be satisfied for bandwidth guarantee in datacenters.

- Work conservation: unused bandwidths can be re-used by other tenants in a way that does not hurt any one's guaranteed bandwidths.
- Fairness: tenants should be share un-used bandwidth by following a certain fairness policy. The policy can be easily specified by the cloud administrator (e.g., for preventing free riding).
- Simplicity: the proposed system should be easily implementable over commercial off-the-shelf hardware without adding additional software or protocol layers to the cloud stack.

It is true that there are already extensive study on enabling efficient bandwidth management in datacenters [2], [4], [6]–[19]. However, current schemes are struggling on achieving bandwidth guarantee and work conservation simultaneously. First, a large group of methods can only provide one of them. For example, the works in Oktopus [4], SecondNet [8], Proteus [9], and CloudMirror [10] provide only bandwidth guarantee, while the works in FairCloud PS-L/N [2], NetShare [6], and Seawall [7] only support work conservation. There are also works that only enable non-strict bandwidth guarantee (e.g., Cicada [16] and SoftBW [18]). Second, existing works that can achieve both bandwidth guarantee and work conservation (e.g., FairCloud PS-P [2], GateKeeper [11], EyeQ [12], and ElasticSwitch [13]) suffer from either impractical assumptions or low efficiency.

The shortcomings of existing methods are caused by two facts. First, the bandwidth resource is distributed across the datacenter (i.e., not centralized). Second, the bandwidth needs of tenants change dynamically. This indicates that it is basically impossible to effectively and timely monitor the bandwidth usage in datacenters for adapting the bandwidth shares of tenants. Consequently, in order to achieve work conservation, a large portion of existing approaches either “predict” network usage status by monitoring at end hosts (e.g., relying on TCP feedback) or require the core network to be congestion free. It is not hard to see that such a direction can hardly be accurate and responsive. It also adds additional complexity to end hosts or routers.

The above limitations drive the development of another strategy that is more efficient. Particularly, first, we let end hosts generate two types of network traffic explicitly: bandwidth guarantee (BG) traffic and work conservation (WC) traffic. The two types of traffic are used to achieve bandwidth guarantee and work conservation, respectively. Second, we assign the BG traffic a higher priority than WC traffic in datacenter routers, i.e., in-network separation. Such a configuration allows WC traffic to take unused bandwidth (i.e., achieve work conservation) aggressively without hurting the bandwidth of BG traffic. Consequently, bandwidth guarantee and work conservation are attained simultaneously and efficiently. We thus name this strategy as priority-based bandwidth guarantee and work conservation.

The advantages of this strategy have been demonstrated in two research studies (i.e., Trinity [14] and Harmonia [15]). However, such a direction still is not worry-free. For example, the WC traffic may easily get trapped or dropped due to the low priority, which limits the WC traffic’s ability to take idled bandwidth. Trinity simply split packets of a single TCP flow to BG and WC traffic. Therefore, it is easier to suffer from the above issue, as whatever happens on the WC traffic applies to the TCP flow directly. For example, a packet loss in the WC traffic would directly reduce the throughput of the TCP flow. Harmonia adopts multipath TCP (MPTCP) [20] to generate BG and WC traffic as two different subflows. This alleviates the above issue. However, it distributes each TCP flow to all paths, which increases the possibility of packet loss on

the WC traffic. Furthermore, how to achieve fair WC under this strategy is not discussed in both works. Such a feature is necessary for cloud providers to prevent tenants from abusing work conservation.

Therefore, in this paper, we aim to develop a solution system that can achieve all three requirements simultaneously. Our solution also exploits MPTCP. Each TCP flow generates two subflows: one is used for bandwidth guarantee (i.e., BG subflow) and the other one is used for work conservation (i.e., WC subflow). We adopt MPTCP because it presents a good synergy with the goal of this paper. It has built-in designs to handle the issues incurred by generating two subflows from one TCP flow (i.e., retransmission, reordering, and packet schedule). MPTCP has been employed to exploit the path redundancy in datacenters [21], [22]. This work differs from them by assigning different roles to MPTCP subflows for bandwidth guarantee and work conservation.

We further adapt MPTCP to handle potential issues due to the low priority of WC subflows. We modify the MPTCP default scheduler to prioritize the BG subflow over the WC subflow. We also find that a small receive buffer may throttle the throughput of the WC traffic due to out-of-order packets and propose to adopt a large receive buffer.

It can be seen that the priority-based work conservation approach can easily motivate a VM to generate more WC subflows for more idled bandwidth. Therefore, we further develop a fair WC scheme by adapting the congestion control of WC subflows. Our scheme manage the overall aggressiveness of all WC subflows on a VM by following a fairness policy, thus ensuring fair sharing of idled bandwidth.

The proposed scheme can be easily and practically implemented. It does not modify current cloud software stack nor need any hardware support that is not available in current commodities. Particularly, it can be built with three easily-implementable techniques: MPTCP, priority queues on switches, and rate control on hosts. MPTCP has already been implemented in the Linux kernel [23]. Our scheme only requires two priority queues per switch port, which can be easily provided on current commodity switches [24], [25]. The Linux `tc` commands can implement the rate control on end hosts [26].

Moreover, though our scheme is built on a customized version of MPTCP, it can be deployed in real clouds easily.

First, thanks to the backward compatibility of MPTCP, users can choose to use MPTCP or just the legacy TCP (by enabling or disabling MPTCP). When a user chooses to not use MPTCP, they just cannot enjoy the benefit of work conservation enabled by our system. They can even use a different WC scheme that achieves WC by dynamically adjusting bandwidth guarantee upon congestion monitoring, e.g., ElasticSwitch [13]. This is because the WC traffic in our scheme has a low priority and will not interrupt BG traffic. Second, cloud providers may be interested in adopting our system due to its safe work conservation. The work conservation traffic in our scheme cannot hurt the bandwidth guarantee because of the low priority. Thus, our system can be regarded

as a safe step to offer BG and WC at the same time in cloud datacenters.

In summary, the contributions of this paper include:

- 1) We novelly exploit MPTCP to provide transparent work conservation upon rate-limiting based bandwidth guarantee in cloud datacenters.
- 2) We have proposed the concept of fair work conservation and proposed a scheme to support this function.
- 3) We have studied how to further improve MPTCP to better support the design goal of this paper, which includes a customized scheduler.
- 4) The proposed system is easily implementable since it only requires protocols and techniques that are available on current commodities.

We organize the rest of the paper as follows. Related work is presented in Section II. The system design is introduced in Section III. Section IV introduces the implementation of the proposed system on a small testbed. Section V presents the performance evaluation. Finally, we conclude the paper in Section VI.

II. RELATED WORK

There are extensive studies regarding efficient bandwidth management in datacenters. However, to the best of our knowledge, current studies fail to achieve the three aforementioned requirements (i.e., bandwidth guarantee, fair work conservation, and design simplicity) simultaneously.

Many early studies only enable bandwidth guarantee in datacenters without work conservation [4], [8]–[10], [27]. SecondNet [8] and Oktopus [4] allows tenants to reserve bandwidth guarantee through the concept of virtual networking abstraction. A central network manager thus is proposed to coordinate VM placement and bandwidth guarantee allocation. Proteus [9] improves the efficiency of bandwidth guarantee by modeling and exploiting tenants' time-varying bandwidth demands (i.e., through a model called temporarily-interleaved virtual clusters or TIVC). However, the model cannot dynamically re-allocate idled bandwidths. CloudMirror [10] also improves the modeling of VMs' bandwidth needs (i.e., for the purpose of bandwidth guarantee) through a novel abstraction named tenant application graph (TAG). SpongeNet [27] proposes a network abstraction model called Fine-grained Virtual Cluster (FGVC) to allow tenants to specify refined bandwidth demands. We can see that these methods only provide static bandwidth reservation without work conservation.

Some other works, e.g., Seawall [7], NetShare [6] and FairCloud (PS-L/N) [2], only support work conservation. Specifically, these methods allocate bandwidths to VMs proportionally. Thus, all bandwidths are allocated as long as there are demands, which improves network utilization efficiency. However, since there is no guarantee regarding the minimal bandwidth, tenants cannot achieve predictable network performance.

It is not hard to see that both bandwidth guarantee and work conservation are necessary for efficient bandwidth management in datacenters. Thus, many works have been devoted to enabling the two functions simultaneously. Gatekeeper [11] and EyeQ [12] allocates bandwidth guarantees to VM pairs through the hose model. However, in order to provide work conservation, they assume that the datacenter core network is congestion free. This assumption may not hold in real datacenters [28], [29]. FairCloud (PS-P) [2] achieves both functions by requiring an unrealistic amount of priority queues on each switch (i.e., one queue per tenant/VM). Hadrian [30] provides per VM bandwidth guarantee for intra-tenant communication and an aggregate bandwidth guarantee per tenant for inter-tenant communications using a hierarchical hose model based framework. However, it requires switches to be able to track flow weights and add rate allocations to packets, which are not readily available now. ElasticSwitch [13] and eBA [31] achieves work conservation by adjusting bandwidth guarantees dynamically. They use the feedbacks of TCP connections to deduce whether there are idled bandwidths in the network. However, since the network usage status changes dynamically, the observation on end hosts can hardly be responsive and accurate.

The above discussion shows that it is a challenging task to provide work conservation on top of bandwidth guarantee. Two methods [14], [15] thus propose to handle this issue by generating explicit low-priority WC traffic, which can take idled bandwidth without hurting bandwidth guarantees. While this direction handles the battle between BG and WC, it still suffers from some issues. The low priority WC traffic may be easily trapped or dropped in the network fabric, particularly when there is less idled bandwidth. This may backfire to the throughput of the WC traffic. Trinity just logically labels packets from a TCP flow as BG traffic or WC traffic. As a result, all packet losses (including those on the WC traffic) will lower the throughput of the TCP flow. This makes the WC traffic (which is prone to have packet loss due to the low priority) easily affect the performance of bandwidth guarantee. Harmonia, on the other side, adopts MPTCP to generate explicit BG subflow and WC subflow for each TCP flow. Since each subflow in MPTCP handles packet loss separately, the aforementioned problem is effectively alleviated. However, MPTCP still suffers some issues such as the choice of the scheduler and out-of-order packets at the receiver, which are not addressed in Harmonia.

In a brief summary, most current works cannot efficiently provide bandwidth guarantee and work conservation simultaneously. The idea of priority-based BG and WC is promising but still suffers from some issues that are not completely handled. Furthermore, how to achieve fair work conservation is largely missing in current works. Those facts motivate us to design a scheme that can achieve the three critical requirements simultaneously.

III. SYSTEM DESIGN

A. DESIGN OVERVIEW

The high-level idea of our approach is to 1) let VMs generate explicit BG traffic and WC traffic and 2) physically isolate them by assigning a higher priority to the BG traffic. Consequently, the BG traffic fulfills the bandwidth guarantee, and the WC traffic takes idled bandwidths without affecting bandwidth guarantees. In addition, the aggressiveness of WC traffic is dynamically adapted to achieve the goal of fairness. Further, as shown later, these functions can be implemented with components that are readily available on commodity servers/switches without additional software or hardware supports. This makes the proposed scheme satisfy the requirement of simplicity.

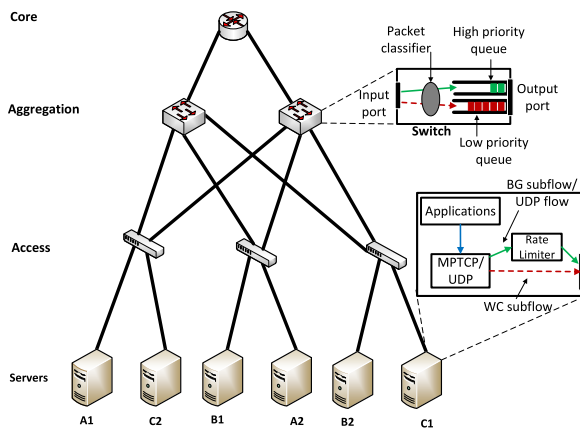


FIGURE 1. System overview (note that details of switch and VM in the figure apply to every switch and VM in the datacenter).

The architecture of our method is plotted in Figure 1. The detailed components in the figure are introduced in the following subsections. The proposed system does not depend on any particular network topology. Therefore, it can work with all state-of-art datacenter topologies, e.g., VL2 [32], fat tree [33], and BCube [34]. Since the WC traffic is transparent to BG traffic in the proposed scheme, a VM can achieve effective and fair work conservation regardless of how bandwidth guarantee is implemented. Therefore, the proposed scheme supports both pipe model and hose model bandwidth guarantee.

B. BANDWIDTH GUARANTEE AND WORK CONSERVATION

Our scheme achieves the design goals by exploiting MPTCP, priority queues on switches, and rate limiter on VMs. Particularly, we use MPTCP to make every TCP flow generate two subflows. We use the two subflows for bandwidth guarantee (i.e., BG subflow) and work conservation (i.e., WC subflow), respectively. In other words, we treat BG subflows as BG traffic and WC subflows as WC traffic. Meanwhile, since UDP flows do not generate two subflows, they are regarded as BG traffic directly. Consequently, each VM generates two types of traffic explicitly. Furthermore, we isolate the two

types of traffic by assigning BG traffic a higher priority than WC traffic on every switch.

Following the same idea as in almost all bandwidth guarantee works, the bandwidth guarantee is achieved through limiting the rate of VMs. Therefore, even clients that only transfer UDP flows cannot send at a rate higher than the guaranteed bandwidth. We assume that there is already an admission mechanism that accepts tenants and places VMs according to their bandwidth requests and bandwidth availability in the datacenter. After the placement of VMs, the number of rate limiter needed on each VM depends on the bandwidth guarantee model. If the hose model is adopted, each VM only needs one rate limiter. However, if we adopt the pipe model, each VM needs to configure a rate limiter for each pair of VMs with a bandwidth guarantee (i.e., each pipe). In this paper, we present our system design with the hose model for its simplicity in illustration.

Our novel handling of WC traffic enables work conservation in a safe and efficient manner without any tradeoff needed. First, all switches strictly prioritize the BG traffic over WC traffic (note that only two priority levels are needed). As a result, each WC subflow can only obtain unused bandwidth on its path and has no chance to affect the bandwidth guarantees allocated over the path. Second, with the safety in hand, there is no need to limit the rate for WC subflows. Thus, work conservation can be efficiently achieved. As a result, a WC subflow can take idled bandwidth as efficient as a normal TCP flow. In this paper, we adopt the ToS value inside the IP header to differentiate BG traffic and WC traffic, i.e., assigning BG subflows and UDP flows a ToS value that is different from that for WC subflows. Note that this differentiation method is mandatory. Other labeling methods may serve this function too as long as they are supported by switches and VMs.

Since all UDP flows are regarded as BG traffic in the current design, UDP applications can only obtain bandwidth through bandwidth guarantee, i.e., cannot get extra bandwidth through work conservation. However, the WC function for UDP applications can be supported similarly by adopting the multi-path UDP protocol, which is left to future work.

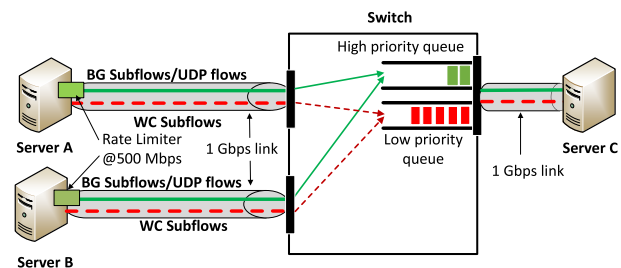


FIGURE 2. Illustration of the bandwidth guarantee and work conservation.

Figure 2 illustrates an example of the proposed scheme in a simplified scenario. In this example, both server A and server B have a bandwidth guarantee of 500 Mbps to server C

(note that every link has a capacity of 1 Gbps in the figure). This guarantee is implemented by limiting the rate of BG traffic on servers A and B to 500 Mbps. Consequently, when both server A and server B demand at least 500 Mbps bandwidth when sending data to server C, no bandwidth is left on the link connecting server C to the switch. In this case, the WC subflows from both servers cannot gain any bandwidth. However, if one of them, say server A, has a bandwidth demand smaller than 500 Mbps, server B's WC subflow will grab the idled bandwidth immediately.

C. ADAPTING MPTCP FOR BG AND WC

While the high-level idea mentioned in the previous subsection is not complex, the usage of MPTCP leads to many unique questions and challenges that need thorough studies, which include 1) why use MPTCP to generate two types of traffic; 2) what is the appropriate number of WC subflows for each TCP flow; 3) how to choose the MPTCP congestion control algorithm; 4) how to improve the MPTCP scheduler; and 5) preventing the throughput degradation resulted from out-of-order packets. We explain our thoughts and solutions for these problems in the following.

1) WHY MPTCP

Multipath TCP (MPTCP) is an extension to the TCP proposed by the Internet Engineering Task Force (IETF). It enables a single TCP connection to transparently send over multiple available interfaces/paths through parallel subflows. Moreover, it is designed in a way that is transparent to upper-layer applications [35].

There are two major reasons that motivate us to use MPTCP in this paper. First, MPTCP allows a TCP flow to generate multiple subflows transparently. Such a characteristic directly satisfies the goal of generating two types of traffic in our design. More importantly, current MPTCP design comprehensively handles issues that could be caused by the split of traffic. The issues include the scheduling of packets to subflows, packet retransmission, packet reordering, and the management of buffers. Second, MPTCP is already supported in the Linux kernel [23]. The current implementation provides a rich set of configuration options to assist our system. Particularly, we can configure the number of subflows of each TCP flow, select the congestion control algorithm, select the scheduler to distribute traffic over active subflows. We discuss these options and how to adapt MPTCP to achieve efficient BG and WC in the following.

2) NUMBER OF WC SUBFLOWS

Our approach only requires one network interface on each VM, which can be easily satisfied in practice. We can still generate two subflows over the single network interface since current MPTCP allows a TCP flow to generate multiple subflows over the same pair of source and destination IPs [23]. These subflows have different TCP ports.

When a TCP flow owns the ability to generate multiple subflows, the next question is what should be the appropriate

number of WC subflows for a TCP flow. Intuitively, generating more WC subflows for a TCP flow would enhance its ability to gain idled bandwidth. We claim that the number of WC subflows should be decided cautiously. The reasons lie in two aspects.

First, on one hand, allowing every TCP flow to generate WC subflows excessively over different paths can easily become destructive. This is because generating too many WC subflows would lead to severe competition among WC subflows on any path. What's worse, WC subflows own a lower priority than BG subflows in our design and can only bandwidth left by BG traffic. Consequently, in this case, WC subflows easily get starved (i.e., packets get dropped or trapped) in the network and experience a larger delay than BG subflows. However, in MPTCP, packets of a flow are sequentially scheduled to its subflows at the sender. Thus, for a particular flow, packets may frequently arrive at the receiver out of order due to the delay difference between the BG subflow and the WC subflow. This would lower the efficiency of both the WC and the BW in the cloud. For the former, WC subflows would be punished by the current design of MPTCP to avoid the throttling from the receive buffer, which makes them unable to take all idled bandwidth. Section III-C5 presents the details of this issue. For the latter, when the receive buffer becomes full due to out-of-order packets, the BG subflow will be held from sending packets too. This would lower the throughput of the BG traffic, which has been proved in the literature [20], [36]–[38].

Second, on the other hand, the number of WC subflows also affects fairness in sharing unused bandwidth at the flow level. Kindly note that we discuss the fairness at the VM level later in Section III-D. For easy illustration, we assume that a TCP flow owns M disjoint paths between its source VM and destination VM. Then, there are four cases regarding the number of WC subflows, denoted N , for the flow. We discuss them one by one in the following.

- $N = 1$. This means that every TCP flow only generates one WC subflow. Naturally, we want the path of the WC subflow to be the same as that of the BG subflow. This makes a TCP flow takes unused bandwidth on the path of its BG subflow, which is consistent with legacy work conservation schemes such as ElasticSwitch [13]. Further, putting the WC subflow on the same path as the BG subflow also makes it easy to achieve fair sharing of unused bandwidth between flows over the same path. In detail, since each subflow essentially is a TCP flow in MPTCP, WC subflows on the same path can fairly share the unused by just adopting the standard TCP congestion control that achieves fairness among flows without the need of extra components.
- $1 < N < M$. In this case, TCP flows between the same pair of hosts (i.e., with the same source and destination IPs) can hardly share unused bandwidth over the M paths fairly. For example, the TCP flow that spreads its subflows over paths that are less congested would gain more advantages than others. Meanwhile, in this case,

the definition of fairness becomes obscured, as TCP flows may partially share their paths. Then, it is hard to appropriately define fairness. Moreover, when this configuration is adopted, tenants may be motivated to be a “free rider” that hopes to find paths with more idled bandwidth by repetitively generating and terminating WC subflows. This would make the overall WC scheme, i.e., the sharing of unused bandwidth, turn into chaos.

- $N = M$. In this case, it is not hard to see that it is better to scatter the N WC subflows to the N disjoint paths. This would solve the issues in the previous case. First, since TCP flows between the same pair of hosts own a WC subflow over every path between the two hosts, these flows can share the unused bandwidths over these paths evenly by following the TCP fairness. Second, since repetitively starting new WC subflows cannot bring a new path to a TCP flow, the aforementioned “free rider” behavior does not work either. However, the problem is that on average, the competition for unused bandwidth on each path is M times severer than that when $N = 1$. Thus, as mentioned above, this could easily starve WC subflows and hurt the efficiency of both BG and WC, especially when the amount of unused bandwidth is limited.
- $N > M$. It is not hard to find that making the number of WC subflows (i.e., N) larger than the number of disjoint paths (i.e., M) is meaningless. This does not make a TCP flow to gain unused bandwidth from more paths. Instead, in this case, some WC subflows of a TCP flow would overlap over the same path and compete for unused bandwidth with each other. As a result, some TCP flows may own more WC subflows over a particular path than others. This makes it hard to achieve fair sharing of unused bandwidth, as it requires to recognize which WC subflows belong to which TCP flow.

Based on the above analysis, we prefer $N = 1$ and $N = M$. This is because they both make it easier to achieve fair sharing of unused bandwidth among flows. When comparing the two options, making N equal to 1 provides the safest WC conservation but limits the range of WC to a single path. However, on the other side, setting N to M could enhance the efficiency of WC (i.e., allow a flow to take unused bandwidth from multiple paths) but risk making WC subflows starved due to intensified competition. Such starvation could reduce the efficiency of WC and BG, as explained later in Section III-C5. Therefore, the selection depends on the context. For example, a cloud provider may leave a sufficient amount of headroom in reserving bandwidth guarantees on each path. In this case, letting $N = M$ is better as there is always sufficient unused bandwidth for WC subflows. However, if the provider does not reserve bandwidth for WC subflows, directly making $N = M$ may not be appropriate due to its potential destructive effect on WC and BG.

Therefore, in summary, we think there is no definite answer regarding the appropriate number of WC subflows for a TCP flow. The cloud provider can choose the value of N based

its needs. We recommend $N = 1$ by default and $N = M$ if the provider leaves enough headroom for WC subflows. We further propose to integrate explicit path control schemes such as [39] to control the path for WC subflows. The WC subflow(s) will be on the same path as the BG subflow if $N = 1$ and be distributed to all disjoint paths between the source and destination if $N = M$.

3) MPTCP CONGESTION CONTROL ALGORITHM SELECTION

In this subsection, we analyze the selection of MPTCP congestion control algorithm. MPTCP currently has two classes of congestion control (CC): uncorrelated CC and correlated CC. Uncorrelated CC algorithms let each subflow handle congestion control independently with a legacy TCP congestion algorithm such as TCP Reno, TCP Vegas, etc. Correlated CC algorithms, on the other hand, correlate the congestion control of subflows of the same TCP flow. The goal of the correlation is to provide fairness and friendliness to TCP flows over the shared bottleneck link, while ensuring the fairness to MPTCP. Representative algorithms include LIA [40], OLIA [41] and BALIA [42].

We found that uncorrelated CC is more suitable for supporting BG and WC in datacenters for two major reasons. Firstly, BG subflows and WC subflows are conceptually uncorrelated in our design. As introduced in Section III-B, BG subflows are responsible for receiving the guaranteed bandwidth, while WC subflows are designed to take idled bandwidth (i.e., work conservation). Meanwhile, BG subflows own a higher priority than WC subflows. Therefore, the two types of subflows should not be correlated. Otherwise, they will not be able to achieve the functions of bandwidth guarantee and work conservation effectively.

We have conducted an experiment to demonstrate this point. Specifically, we let three servers, denoted S1, S3, and S4, share a link and configure different bandwidth guarantees for them (i.e., 100 Mbps, 200 Mbps, and 300 Mbps). Meanwhile, after allocating the bandwidth guarantee, there is about 300 Mbps idled bandwidth left over the link. We let every server generate one TCP flow with MPTCP over the link. In other words, each of the three servers has one BG subflow and one WC subflow running over the link. We then measured the throughput of each server’s WC subflow under both correlated and uncorrelated congestion control algorithms. We have plotted the results in Figure 3. We see from the figure that when LIA, OLIA, and BALIA are adopted, the idled bandwidth is shared in a chaos manner among the three WC subflows. They all fail to share the idled bandwidth evenly among the three servers. This is because, due to the correlated congestion control, the increase/decrease of the congestion window of the WC subflow is correlated with the bandwidth of the BG subflows. Thus, when BG subflows of the three servers take different guaranteed bandwidths, their WC subflows present different levels of aggressiveness in competing for unused bandwidth.

Secondly, the rate limit based bandwidth guarantee and the strict prioritization of BG subflows over WC subflows can

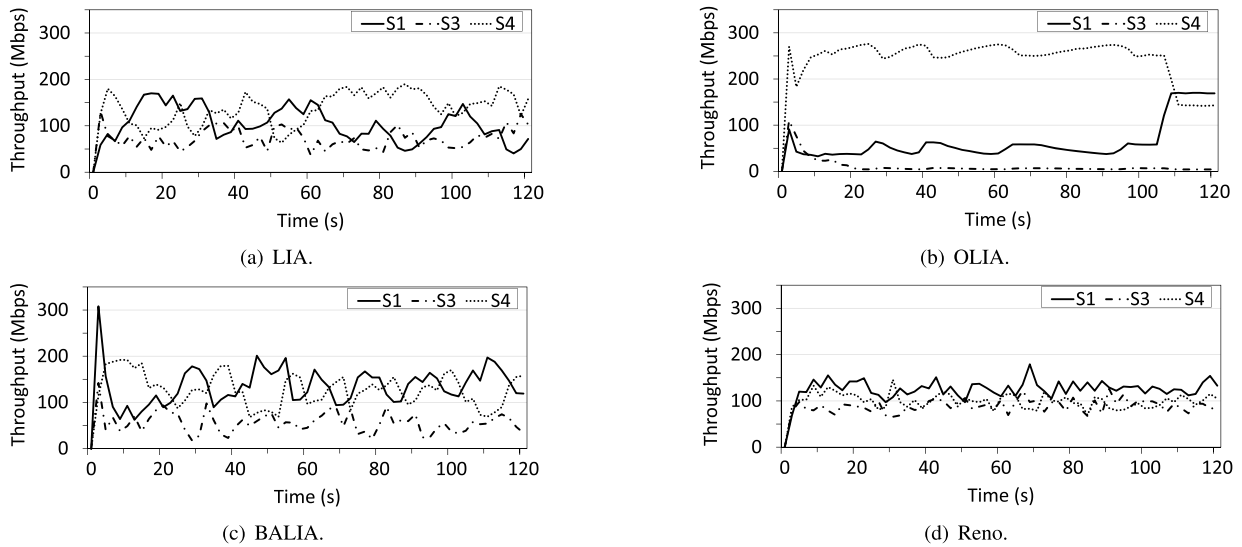


FIGURE 3. Throughput of WC subflows under different congestion control algorithms.

handle the unfairness to single-path TCP flows. An MPTCP flow takes the guaranteed bandwidth through its BG subflow and only obtains idled bandwidth through its low-priority WC subflow. Therefore, there is no unfairness issue to normal TCP flows in the datacenter with our scheme deployed. Due to these reasons, we choose an uncorrelated congestion control algorithm, i.e., TCP Reno, in this paper. Other state-of-art TCP congestion control algorithms such as Vegas and Cubic can be adopted directly too.

4) IMPROVING THE MPTCP SCHEDULER

MPTCP internally relies on a scheduler to distribute data from the upper layer application to different subflows in a transparent manner. Current MPTCP implementation offers three schedulers: “LeastRTT”, “RoundRobin” and “Redundant”. “LeastRTT”, which is the default scheduler, prioritizes the subflow based on their RTTs. The remaining two distribute packets to subflows in sequentially (i.e., round-robin) and redundantly, respectively. As different subflows in this paper carry different roles, it is necessary to examine the suitability of scheduling strategies.

Problem: It is not hard to find that the BG subflow, which is supposed to take the guaranteed bandwidth and owns a high priority, should be prioritized over the WC subflow in the scheduling. Thus, the “RoundRobin” scheduler and the “Redundant” scheduler do not suit our design. The default “LeastRTT” scheduler appears to indirectly satisfy this goal. This is because WC subflows own a low priority in switches and thereby tend to have a longer RTT than BG subflows. Thus, the BG subflows are prioritized. However, this is not always the case even in our small scale tests. We find that, with the “LeastRTT” scheduler, the BG subflow often cannot achieve the guaranteed bandwidth, and the WC subflow may obtain more bandwidth than the idled bandwidth, especially when there is substantial idled bandwidth.

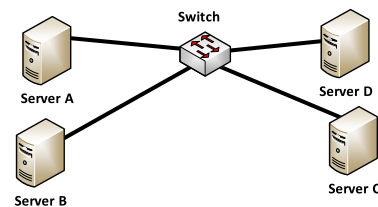


FIGURE 4. The setup for evaluating the influence of MPTCP scheduler.

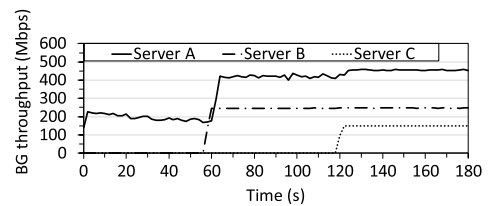


FIGURE 5. Default scheduler: 1) server A’s BG subflow fails to take the guaranteed bandwidth initially; 2) server A takes 2 seconds to shift its traffic from WC subflow to BG subflow after server B starts transmitting.

Our experiments in a simple scenario shown in Figure 4 illustrates this point. In this configuration, all links are 1 Gbps. Servers A, B and C own a bandwidth guarantee of 500 Mbps, 250 Mbps and 150 Mbps on the link to server D, respectively. In the test, servers A, B, and C start sending traffic to server D (through iperf) at 0s, 60s, and 120s, respectively. Figure 5 illustrates the throughput of each server’s BG subflow. We see that server A’s BG subflow initially only achieves around 200 Mbps, while the guaranteed share is 500 Mbps. At this moment, there is much idled bandwidth available. As a result, the WC subflow’s packets can get through switches easily, though they own a low priority due to our configuration. This makes the WC subflow do not always

present a higher RTT than the BG subflow. Consequently, the BG subflow is not strictly prioritized at the scheduler. Such an explanation is further confirmed by the fact that when servers B and C start sending packets, their BG subflows grow to the guaranteed share directly. This is because the amount of idled bandwidth is quite small at this time.

Kindly note that such an effect does not break the bandwidth guarantee. When the amount of idled bandwidth reduces, BG subflows would take back their guaranteed shares due to the priority setting, as shown for server A in the previous example. However, this causes instability to data transfer may also cause packet drops on WC subflows.

Solution: In order to handle the issue illustrated above, we modified the default scheduler to strictly prioritize the BG subflow whenever it is available regardless of the RTT. Only when the BG subflow is not available, i.e., its congestion window is full, and the WC subflow is available, the scheduler would dispatch a segment over the WC subflow. Algorithm 1 presents the pseudo-code of the BG-prioritizing scheduler. Its performance will be evaluated in Section V-D1. Furthermore, replacing the “LeastRTT” scheduler with the BG-prioritizing scheduler does not make the receiver receive more out-of-order packets in the context of this work. The major reason is that the BG subflow strictly owns a higher priority than the WC subflow. Thus, packets dispatched to the WC subflow suffer from a higher chance of loss, though the WC subflow may temporarily present a smaller RTT. This indicates a higher chance of causing out-of-order packets. Consequently, the WC subflow is always an inferior option for controlling out-of-order packets due to the low priority. This means that the BG-prioritizing scheduler would not perform worse than the default “LeastRTT” scheduler in terms of out-of-order packets.

Algorithm 1 BG Subflow Prioritization

```

1: procedure check_to_send(sk_buff* skb)
2:   if the master subflow is available then
3:     send it on master subflow
4:   else if the slave subflow is available then
5:     send it on slave subflow
6:   else
7:     return NULL
8:   end if
9: end procedure

```

5) INFLUENCE OF RECEIVE BUFFER

MPTCP schedules packets of a TCP flow to different subflows at the sender side [20]. When different subflows go through paths with different delays, packets can easily arrive the receiver out of order. We thus analyze how this effect affects the proposed bandwidth guarantee scheme.

Problem: When the path heterogeneity is not significant, the receive buffer can effectively handle out-of-order packets and allow all subflows to achieve at its attainable bandwidth.

However, this is not assured all the time. When the receive buffer is filled with out-of-order packets that can not be delivered to the application, the receive window size (i.e., RWND in TCP) advertised to the sender would throttle subflows from transmitting. As a result, the host is unable to attain available bandwidths on employed paths and may even gain a throughput that is lower than that from only using TCP over the best path. Such an effect is particularly critical in our design as WC subflows may easily lead to out-of-order packets due to its low priority.

MPTCP handles this issue by penalizing the slow path (i.e., halving its congestion window) and re-injecting packets from the slow path to the fast path when the throttling caused by receive window happens [20]. While such a scheme is effective in avoiding the fast path from being throttled due to out-of-order packets, it loses some bandwidth on the slow path due to the penalty. This would make the WC subflow unable to take idled bandwidth effectively.

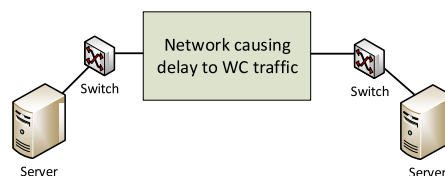


FIGURE 6. Setup used to show the effect of buffer size on WC subflow.

TABLE 1. Receive buffer sizes (in bytes) in the 1 ms delay scenario.

	Minimum	Default	Maximum
setting 1	2048	6144	131072
setting 2	3072	8192	262144
setting 3	4096	10240	524288
setting 4	5120	14336	1048576
setting 5	6128	87380	6291456

TABLE 2. Receive buffer sizes (in bytes) in the 2 ms delay scenario.

	Minimum	Default	Maximum
setting 1	2048	6144	131072
setting 2	3072	8192	262144
setting 3	4096	10240	524288
setting 4	5120	12288	786432
setting 5	6144	14336	1048576

To illustrate the above drawback, we conducted an experiment with an exemplary setup shown in Figure 6. In this setup, a sending server sends two subflows (1 BG subflow and 1 WC subflow) to a receiving server over a single link with 1Gbps capacity. As the setup cannot lead to enough delay difference between the BG subflow and the WC subflow, we purposely added additional 1ms and 2ms delay to the WC subflow through *netem*. We varied the receive buffer size as shown in Tables 1 and 2. We tested under two configurations of bandwidth guarantee: 200 Mbps and 500 Mbps. We measured the throughput of the two subflows under the two configurations and plotted the results in Figures 7 and 8.

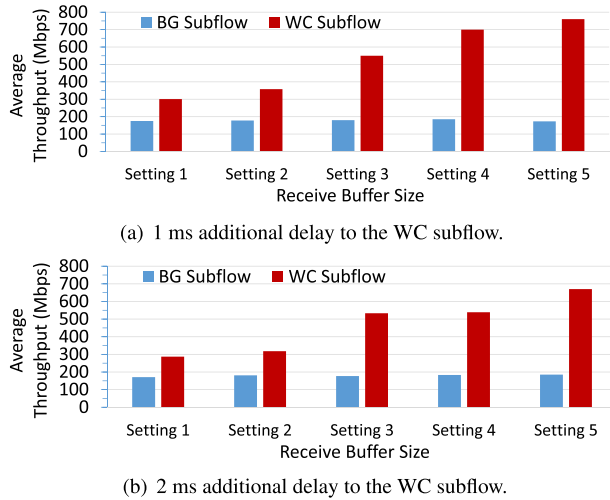


FIGURE 7. Effect of the receive buffer size on the throughput under 200 Mbps bandwidth guarantee.

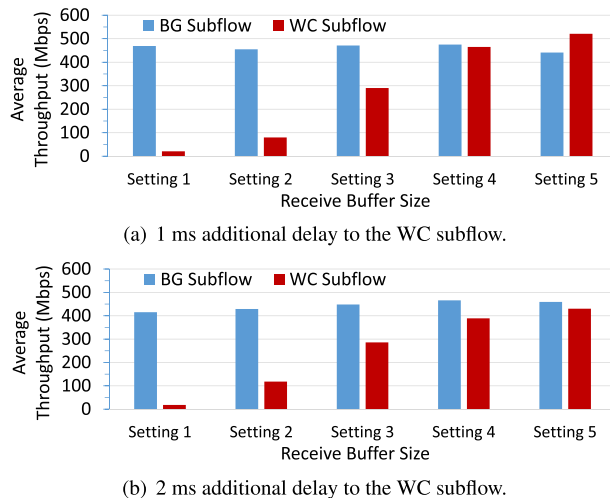


FIGURE 8. Effect of the receive buffer size on the throughput under 500 Mbps bandwidth guarantee.

We see from the four figures that in both configurations, when the receive buffer size is small, the average throughput of the WC subflow is very small, even though there is idled bandwidth available. This is caused by the penalty imposed by current MPTCP, which aims to prevent throttling the fast subflow (i.e., the BG subflow) when the receive buffer is full. When the receive buffer size increases, the throughput of the WC subflow increases substantially. This is because a large receive buffer size can tolerate more out-of-order packets, which makes MPTCP less likely to trigger the penalty to the WC subflow. We see from the four figures that in all tests, the WC subflow can take all idled bandwidth left by the BG subflow when the receive buffer size is large enough.

Solution: Based on the above observation, it is preferable to have a large receive buffer in our context, thus allowing more efficient work conservation. However, always using a large receive buffer is a wastage of the memory resource.

Therefore, we propose to dynamically determine the size of the receive buffer that is enough to tolerate out-of-order packets without holding both subflows from sending. Specifically, we require the receive buffer size (denoted B) to satisfy the following requirement.

$$B \geq \beta * \max\{RTT_i\} * \sum_{i=1}^n BW_i \quad (1)$$

where $\max\{RTT_i\}$ is the maximal RTT of all subflows, and BW_i is the bandwidth of the i -th subflow, $\beta \geq 1$ is a safety factor. The rationale of this equation is that in case a packet is lost in the slowest subflow, the receive buffer should be large enough to hold all subsequent out-of-order packets until the packet is retransmitted. In other words, the retransmission of the packet takes the $\max\{RTT_i\}$ amount of time in the worst case. The value of β should be decided according to the dynamism of network load in the cloud. The severer the network load dynamism is, the larger it should be. The work in [43] adopts 3 in the Internet context. The performance of such a solution will be evaluated later in Section V-D2.

D. FAIR WORK CONSERVATION

Unused bandwidth maybe unfairly shared when MPTCP is adopted directly for WC. We demonstrate this issue and propose a solution in this subsection.

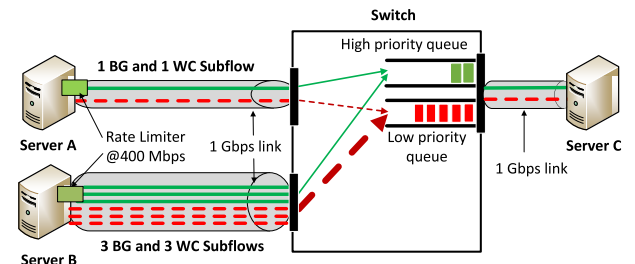


FIGURE 9. Unfair work conservation.

1) UNFAIR SHARE OF UNUSED BANDWIDTH

As shown in Section III-C3, we adopt an uncorrelated congestion control algorithms, e.g., TCP Reno, in this paper. Therefore, the allocation of unused bandwidth follows flow-level fairness, as defined in TCP congestion control. This indicates that a VM can obtain more bandwidth over a bottleneck link by generating more WC subflows on the link. Such an effect is demonstrated in the scenario shown in Figure 9. In this scenario, both server A and server B own 400 Mbps guaranteed bandwidth on the path to server C. Therefore, there is 200 Mbps idled bandwidth on the link connecting the switch and server C. Furthermore, server B creates 3 TCP flows (and subsequently 3 WC subflows) towards server C, while server A just has one. Consequently, three WC subflows from server B compete for idled bandwidth with the WC subflow from server A. Note that the competition on WC bandwidth does not compromise the bandwidth guarantee.

Due to the flow-level fairness, server Bw would get 2 times more work conservation bandwidth than server A.

It is not hard to see that this feature can be exploited to distort the share of idled bandwidth. A tenant can simply generate more TCP flows (and consequently more WC subflows) for its applications to gain more bandwidth through work conservation, rather than paying fairly for guaranteed bandwidth. Such behaviors would deteriorate the service model in the datacenters. Consequently, it is necessary to further regulate how idled bandwidths are shared among VMs under the context of bandwidth guarantee.

2) FAIRNESS POLICY

To solve the aforementioned issue, we propose the concept of “fairness policy” for work conservation. Note that the “fairness” does not refer to tenants but rather to the cloud owner. The purpose of the policy is not to enforce absolute fairness on allocating unused bandwidths but rather to allow the cloud owner to enforce/extend the pricing model beyond the guarantees by controlling how unused bandwidths are shared. For example, a datacenter owner may choose to allocate the unused bandwidth for free in proportional to the number of purchased VMs. The owner may also sell “the ability to obtain unused bandwidth” at a certain price and thus better satisfy diverse tenant demands. As a result, the cloud owner’s economic interest is better protected, which is critical to the sustainability of the cloud ecosystem.

We enable the fairness policy by defining a WC competition level for each tenant. This level decides the tenant’s ability to obtain unused bandwidths. The competition level of a VM (denoted A_l) is decided by the fairness policy adopted by the datacenter owner. We further assume that competition level A_l is an integer and $A_l \in [1, N_a]$. We also only assume integer level. We explain how this competition level is implemented in the following subsection.

3) ENABLING THE FAIRNESS POLICY

Obviously, the rate limit at VMs cannot implement the fairness policy in the dynamic cloud environment. We thus adopt a solution that is inspired by how TCP works. TCP ensures fair sharing of a link through its congestion control that determines how fast a flow increases its sending rate upon an ACK and how much to reduce upon a packet loss. By adopting an additive increase and multiplicative decrease (AIMD) strategy in the two processes, all TCP flows own the same aggressiveness in competing for the bandwidth. Therefore, we can enable the fairness policy by changing the aggressiveness of WC subflows.

Specifically, we want the combined aggressiveness of a VM’s WC subflows (denoted as its ability to obtain WC bandwidth) equals to that of A_l single-path TCP flows, where A_l is the VM’s WC competition level, no matter how many WC subflows it has. As a result, idled bandwidth is allocated to VMs in proportional to their A_l (i.e., competition level). Note that a tenant that wishes to use only spare bandwidth will not receive any guaranteed bandwidth and will have its

A_l set to the minimal value. Such an idea has been used in literature such as [44]. We present the mathematical modeling that shows how to achieve this goal.

We use A_l and K to denote the WC competition level and the number of WC subflows of a VM, respectively. We use MSS to denote the maximum segment (packet) size, and α_s and β_s to denote the increment and decrement factor of the congestion control of a standard single-path TCP flow, respectively. By default, $\alpha_s = 1$, i.e., increase congestion window by one MSS per round trip time (RTT), and $\beta_s = 0.5$, i.e., reduce the congestion window by half upon a packet loss. Since the congestion window size denotes the throughput of a TCP flow (i.e., a flow transmits a window of data in every RTT), the two parameters essentially decide the aggressiveness of a WC subflow in competing for bandwidth. Thus, the aggressiveness of a WC subflow can be changed by adapting the two parameters. Since they have the same effect, we only change the increment factor in this paper.

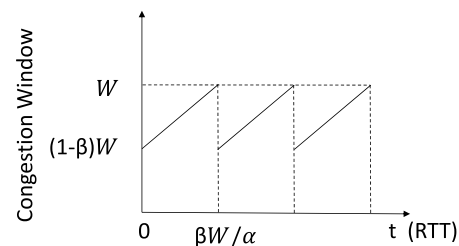


FIGURE 10. Illustration of TCP congestion window under increment factor α and decrement factor β .

To find the increment factor needed to achieve fair WC, we first deduce the flow throughput under increment factor α and decrement factor β by modeling how the congestion window evolves, as adopted in the modeling of the Mathis Equation [45]. As shown in Figure 10, we assume W as the maximal window size the flow can achieve. In other words, a packet loss would happen when the congestion window reaches W . In this case, the window size will be dropped to $(1 - \beta)W$. Afterward, the number of RTTs needed for the flow to grow from $(1 - \beta)W$ to W is $\beta W / \alpha$ (i.e., the window increases by α in each RTT). This process then repeats. We can see that in each period, the number of packets transmitted is $\frac{(1-\beta)*\beta*W^2}{\alpha} + \frac{\beta^2*W^2}{2\alpha} = \frac{(2-\beta)*\beta*W^2}{2\alpha}$. Since there is one packet loss every $\frac{(2-\beta)*\beta*W^2}{2\alpha}$ packets transmitted, we can get the packet loss rate p represented by $\frac{1}{p} = \frac{(2-\beta)*\beta*W^2}{2\alpha}$. We thereby get $W = \sqrt{\frac{2\alpha}{p*(2-\beta)*\beta}}$. Then, the throughput of the flow can be modeled as the following

$$\begin{aligned}
 T &= \frac{MSS * \frac{(2-\beta)*\beta*W^2}{2\alpha}}{RTT * \beta * \frac{W}{\alpha}} \\
 &= \frac{MSS}{RTT} * \frac{(2-\beta)W}{2} \\
 &= \frac{MSS}{RTT} * \sqrt{\frac{\alpha}{p}} * \sqrt{\frac{2-\beta}{2\beta}} \tag{2}
 \end{aligned}$$

We then look at the design goal of the fair WC policy: make K WC subflows obtain the same amount of bandwidth as A_l single-path TCP flows. We use T_w and T_s as the throughput of a subflow and a standard single-path TCP flow on the same path, respectively. The design goal indicates that we then need to have

$$K * T_s = A_l * T_w \tag{3}$$

Based on Eq. (2), T_s can be modeled as the following.

$$T_s = \frac{MSS}{RTT} \sqrt{\frac{\alpha_s}{p_s}} \sqrt{\frac{2 - \beta_s}{2 * \beta_s}} \tag{4}$$

and T_w can be modeled as

$$T_w = \frac{MSS}{RTT} \sqrt{\frac{\alpha_w}{p_s}} \sqrt{\frac{2 - \beta_s}{2 * \beta_s}} \tag{5}$$

where α_s and α_w denote the increment factor of the standard single-path TCP flow and the WC subflow, respectively. The two flows both have a decrement factor of β_s , since we did not adapt the decrement factor. Further, RTT and P_s denote the round trip time and loss rate of the path, respectively.

By replacing T_s and T_w in Eq. (3) with the above two equations, we get the following.

$$K * \frac{MSS}{RTT} \sqrt{\frac{\alpha_w}{p_s}} \sqrt{\frac{2 - \beta_s}{\beta_s}} = A_l * \frac{MSS}{RTT} \sqrt{\frac{\alpha_s}{p_s}} \sqrt{\frac{2 - \beta_s}{\beta_s}} \tag{6}$$

By solving the above equation, we can obtain that

$$\alpha_w = \frac{A_l^2}{K^2} \alpha_s \tag{7}$$

Eq. (7) means that when the incremental factor of every WC subflow is adjusted to $\frac{A_l^2}{K^2}$ of that of a standard single-path TCP flow, i.e., increase the congestion window by $\frac{A_l^2}{K^2}$ (instead of 1) per RTT, the K WC subflows in total obtain the same bandwidth as A_l standard single-path TCP flows. This exactly achieves the goal of the fair WC policy.

4) FAIRNESS POLICY EXAMPLE

We further present an exemplary fairness policy. The policy requires that VMs of a tenant share unused bandwidth in proportion to the paid and guaranteed bandwidth, i.e., the A_l of each VM is proportional to its guaranteed bandwidth. The next question is how to map the guaranteed bandwidths to the competition level (i.e., A_l). We adopted a mechanism that counts how many bandwidth units the guaranteed bandwidth share contains. Specifically, $A_l = B_g/U$, where B_g and U represent the guaranteed bandwidth share and the bandwidth unit, respectively. For example, suppose U is 75 Mbps, then 400 Mbps yields to 5, 300 Mbps yields to 4, and 100 Mbps yields to 1. The value of U can be decided according to the bandwidth guarantee granularity in the cloud. We use 75 Mbps in the setup of this paper.

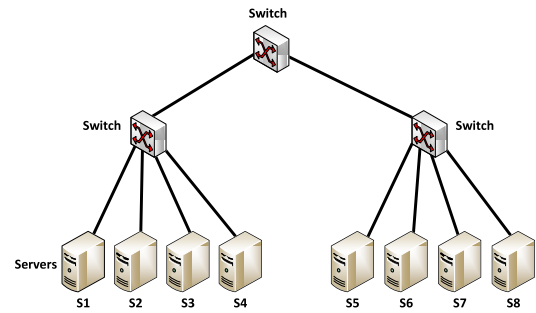


FIGURE 11. Testbed topology.

IV. IMPLEMENTATION OF OUR SOLUTION

We have implemented the proposed system on a small cluster with eight servers, as shown in Figure 11. Each server runs Ubuntu 14.04 and is installed with a 1 Gbps network card. The three switches in the cluster are gigabit smart switch. Therefore, every link in the topology has 1 Gbps bandwidth. For simplicity, we treat each server as a VM directly.

We further installed Linux kernel MPTCP on all servers. We set TCP Reno as the congestion control algorithm for MPTCP. We adopted the ‘‘BG-Prioritized’’ scheduler proposed in Section III-C4. We used *ndiffports* as the path manager for MPTCP and set its value to 2. This means that each TCP flow will generate two subflows. The two subflows are differentiated by the ToS value in the IP header. We modified the MPTCP source code to set the default ToS value of the first and second subflow to 0×00 and 0×20 , respectively. Thus, the two subflows of each TCP flow have different ToS values. We treat them as BG subflow (i.e., BG traffic) and WC subflow (i.e., WC traffic), respectively. Furthermore, the ToS value of all UDP flows is set to 0×00 by default, which matches with our design that regards it as the BG traffic. We further configured priority queues on each smart switch to prioritize the BG traffic over WC traffic, i.e., packets whose ToS value equals to 0×00 are forwarded to the high-priority queue.

We used Linux tc [26] (i.e., HTB queuing discipline) to set up the rate limit on each server for implementing the bandwidth guarantee. Note that the rate limit only applies to BG traffic (i.e., by matching to ToS value 0×00). We also developed a script on each server to be called (with an input of bandwidth guarantee) for setting up the rate limit.

We have further modified the MPTCP code to implement the fairness policy for work conservation. We changed the congestion window increment factor of each WC subflow to ‘‘1 per $\frac{K^2}{A_l^2} * RTT$ ’’. We have added two additional sysctl variables to receive the value of K and A_l . The two variables are named *sysctl_num_parallel_subflows* and *sysctl_wc_competition_level*. A user space program is developed to set the two variables.

The above implementation process only requires minor modification of MPTCP code. It does not need to change any hardware or other software. This demonstrates that our approach can be easily deployed in current datacenters.

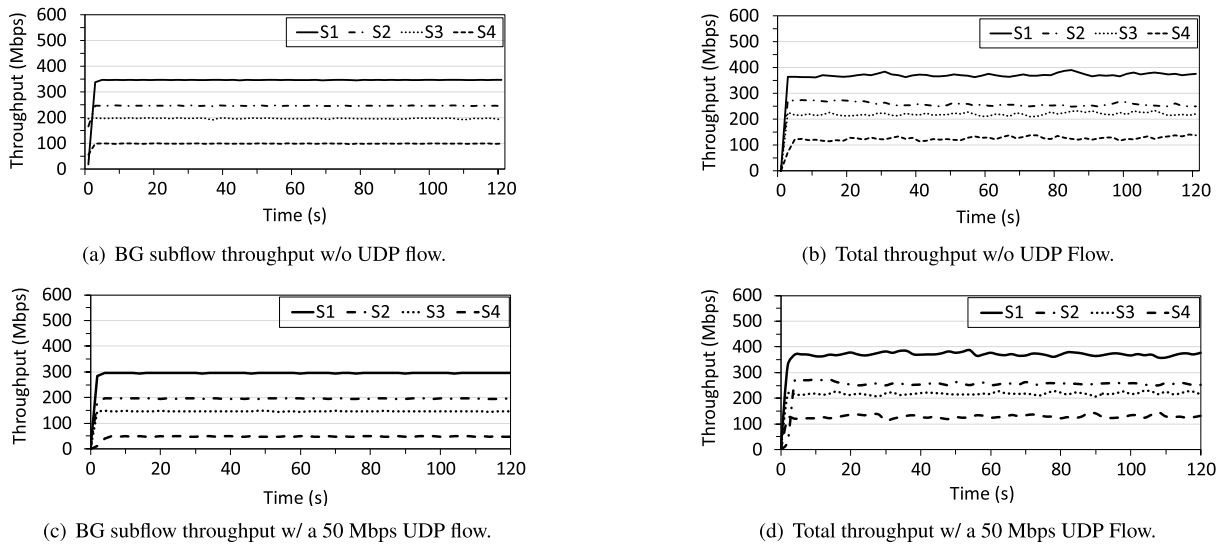


FIGURE 12. Throughput of the four sending servers in evaluating the performance of bandwidth guarantee.

V. PERFORMANCE EVALUATION

In this section, we first conducted testbed based experiment to evaluate the performance of the proposed system in terms of bandwidth guarantee, work conservation, and fairness in Sections V-A, V-B, and V-C, respectively. We then evaluated the overall performance in a MapReduce task in Section V-E. In these tests, we adopted the same testbed as the one implemented in Section IV (i.e., Figure 11). In the test, traffic is generated from the left branch to the right branch, i.e., servers S1, S2, S3, and S4 send data to S5, S6, S7, S8, respectively. Each one of the four left branch servers generates one flow to its receiver with *iperf3*. We measured the throughput achieved by each server with *iftop*. Unless explicitly indicated, we set the bandwidth guarantee of servers S1, S2, S3, and S4 to 350 Mbps, 250 Mbps, 200 Mbps, and 100 Mbps, respectively.

Due to the scale limit of the testbed, we further developed a simulator to evaluate the performance of our system at a large scale with 400 servers (i.e., Section V-F). We adopted a Facebook datacenter MapReduce trace [46] to drive the generation of MapReduce jobs in the simulation.

A. BANDWIDTH GUARANTEE

We evaluate the performance of our scheme on bandwidth guarantee under both TCP and UDP traffic. We first let servers generate only TCP traffic, i.e., each of the four left branch servers creates a TCP flow. Afterward, we also tested when each server further generates a UDP flow at the rate of 50 Mbps. We then measured the BG subflow throughput and the total throughput of each server in the two tests. The measurement results are plotted in Figure 12.

The results show that the bandwidth guarantee is effectively achieved. The four left servers obtain a bandwidth of 350 Mbps, 250 Mbps, 200 Mbps, and 100 Mbps, respectively, through the BG subflow. Furthermore, they shared the 30 Mbps idled bandwidth over the path from the left branch

to the right branch through their WC subflows. As a result, the total throughput of the four servers is slightly higher than their bandwidth guarantees, as shown in Figure 12(b).

Figures 12(c) and 12(d) show the BG subflow throughput and total throughput of the four sending servers when the 50 Mbps UDP flow is added. By comparing Figure 12(c) with Figure 12(a), we can see that the throughput of each server's BG subflow is 50 Mbps fewer than that without the UDP flow. This means that the UDP flow is regarded as part of BG traffic. Figure 12(d) further shows that the bandwidth guarantee is still achieved on the four servers.

Summarizing the above results, we can see that bandwidth guarantee is successfully enforced with the proposed scheme when the flow generation does not set a rate limit.

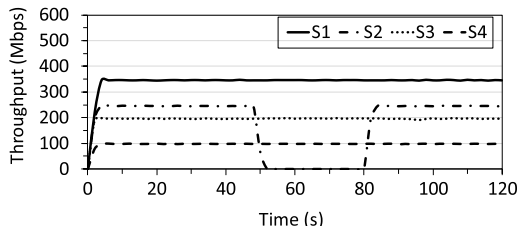
B. WORK CONSERVATION

We further evaluate our scheme's performance on work conservation. The experiment configuration is the same as in the previous experiment. However, we stopped the TCP flow between S2 and S4 between 50s and 80s. This leads to about 280 Mbps idled bandwidth over the path. We measured the BG subflow throughput and the total throughput of the four sending servers (i.e., S1 to S4) in this process. The measurement results are plotted in Figures 13(a) and 13(b).

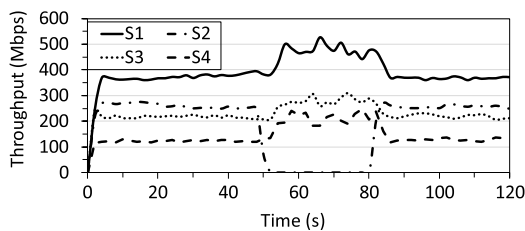
We see from Figure 13(b) that the idled bandwidth created by stopping S2's traffic can be efficiently taken by others. When S2 recovers, its throughput increases to the guaranteed bandwidth quickly, and other servers' total throughput all fall back to the amount before stopping S2's traffic. We also see from Figure 13(a) that the throughput of the BG subflow on servers S1, S3, and S4 remains stable throughout the experiment. This illustrates that the work conservation is achieved solely by WC subflows. Consequently, by combining all those results, we conclude that efficient and responsive work conservation is achieved through the proposed scheme.

TABLE 3. 3 servers sharing a 1 Gbps link proportionally. Each server gets a WC share that is proportional to its BG share.

Server A BG Mbps	Server B BG Mbps	Server C BG Mbps	Server A WC Mbps	Server B WC Mbps	Server C WC Mbps	proportional fairness parameter (A_I)		
						Server A	Server B	Server C
400	200	200	62	31	31 Mbps	5	2	2
300	100	100	212	106	106	2	1	1
400	200	100	120	65	35	5	2	1
400	200	Not sending	217	110	Not sending	5	2	0



(a) BG subflow throughput.



(b) Total throughput.

FIGURE 13. Throughput of the four sending servers in evaluating the performance of work conservation.

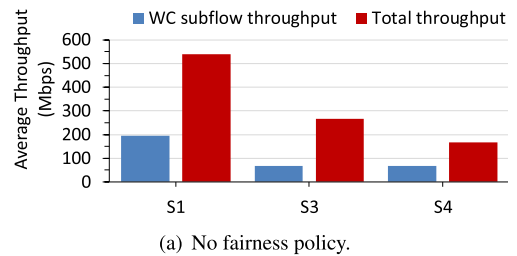
C. FAIR WORK CONSERVATION

In this subsection, we evaluate the fair work conservation scheme proposed in Section III-D.

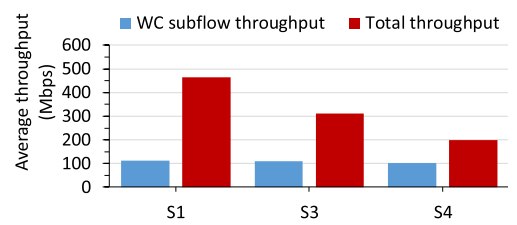
We have first used the same topology as in Figure 4 to evaluate the effectiveness of our scheme for fair WC under different bandwidth guarantees. We adopted the example fairness policy presented in Section III-D4. The achieved throughput of each server is recorded in Table 3. The results show that the idled bandwidth is shared by the three servers in proportion to their guaranteed bandwidth when different configurations are adopted. Such a result shows that this fairness policy is effectively enforced.

We then used the testbed topology (i.e., Figure 11) to conduct evaluation in more realist scenarios. Specifically, we stopped the traffic on S2 to create about 280 Mbps idled bandwidth on the path. We then tested with and without the fairness policy. In the former test, we adopted the fairness policy that makes the three servers other than S2, i.e., S1, S3, and S4, share the idled bandwidth evenly. To show that the proposed scheme is resilient to the number of flows a server generates, we start N_c TCP flows on S1 and only one TCP flow on servers S3 and S4. We measured the throughput of the three servers when N_c is set to 3 and 5. The results are plotted in Figures 14 and 15, respectively.

The two figures show that, when the fairness policy is not enforced, the three servers obtain the WC bandwidth

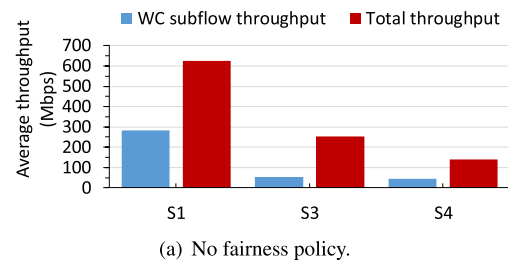


(a) No fairness policy.

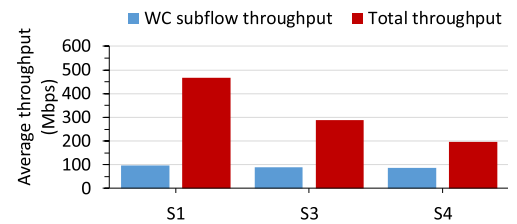


(b) Fair sharing policy with 1:1:1 sharing of unused bandwidth.

FIGURE 14. Evaluation of the performance of fair work conservation with 3 connections on S1.



(a) No fairness policy.



(b) Fair sharing policy with 1:1:1 sharing of unused bandwidth.

FIGURE 15. Evaluation of the performance of fair work conservation with 5 connections on S1.

in proportional to the number of WC subflows. Particularly, the WC throughput of S1 is 3 (or 5) times of that of S3 and S4. This is because the idled bandwidth is allocated fairly among all WC subflows. However, when the fairness

policy is enforced, the idled bandwidth is shared by the three servers by following the policy, regardless of the number of WC subflows each server owns. Specifically, as shown in Figures 14(b) and 15(b), all servers get the same amount of WC throughput. More importantly, we see from the two figures that the bandwidth guarantee is always enforced (i.e., each server gets at least the guaranteed bandwidth) when different fairness policies are adopted.

The above test results show that the proposed scheme can achieve bandwidth guarantee and work conservation effectively and simultaneously in cloud datacenters. Moreover, our scheme can be easily deployed with current commodities, as shown in Section IV.

D. EVALUATION OF CRITICAL COMPONENTS

In this subsection, we evaluate two critical components proposed in our scheme.

1) BG-PRIORITIZING SCHEDULER

We first evaluate the performance of the BG-prioritizing scheduler proposed in Section III-C4. For direct comparison, we conducted the same experiment used to illustrate the issue of the default scheduler in Section III-C4, i.e., Figure 5. The result is shown in Figure 16. We see from the figure that server A keeps its throughput over the BG subflow up to its guaranteed bandwidth all the time, even when no other servers are sending data over the shared link. This is because the prioritization of BG subflow helps avoid the throughput degradation experienced when using the default scheduler, as mentioned in Section III-C4.

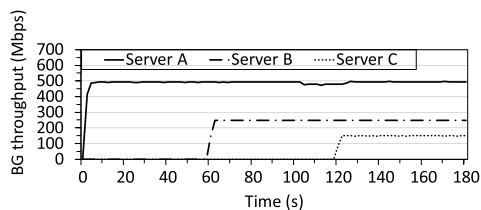


FIGURE 16. BG-prioritizing scheduler: Server A maintains a constant BG rate even in the initial stage.

2) DYNAMIC RECEIVE BUFFER ADAPTATION

We evaluate the performance of the dynamic receive buffer adaptation method (presented in Section III-C5) in this subsection. We adopted the same configuration as in Figure 6 but with the dynamic receive buffer adaptation scheme enabled on both servers. In the test, we increased the additional delay to the WC subflow by 1ms every 15 seconds and measured its throughput every 1 second. The test lasts for 45 seconds. The results are shown in Figure 17.

We see from the figure that whenever the delay added to the WC subflow increases, the throughput of the WC subflow decreases. This is because the sudden increase of the delay causes out-of-order packets at the receiver, which triggers the penalty on the WC subflow. However, as our

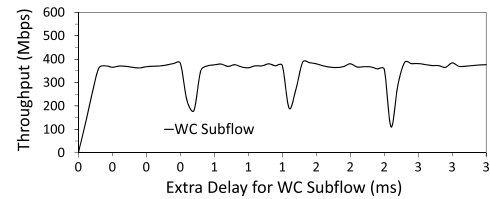


FIGURE 17. Effect of the dynamic receive buffer adaptation.

scheme could detect the change of the delay and dynamically increase the receive buffer size, the reduction is recovered quickly, which is shown in the figure too. Such a result shows that dynamically adjusting the receive buffer size could effectively prevent the penalty to the WC subflow due to the receiver's inability to handle all out-of-order packets, thereby allowing it to effectively take idled bandwidth.

E. RUNNING HADOOP TESTDFSIO

In order to evaluate our system under real applications, we compare the performance of a Hadoop task with and without our scheme. We set up a Hadoop cluster that consists of 1 namenode and 7 datanodes running Apache Hadoop 2.7.3 [47]. We use the same topology as the one shown in Figure 11. TestDFSIO, which is a benchmark application, is launched to measure the cluster performance in terms of parallel write operations of a MapReduce job. The test writes 10 files with a size of 1 GB and a replication parameter of 3. Because TestDFSIO is unable to generate enough traffic to saturate the 1 Gbps links in the cluster, we have reduced the capacity of the links to 128 Mbps by throttling the switches port. We further run bidirectional background traffic using *iperf3* between the two branches of the topology.

With our solution in place, background flows are configured with ToS = 0×20 , i.e., WC traffic. We adopt the same software and settings on these servers as those used in our initial implementation in Section IV. We assign each server a bandwidth guarantee of 30 Mbps. In the test without our solution, we run the same application using standard TCP on Linux Ubuntu 14.04 servers. For a fair comparison with the result obtained in the test with our scheme enabled, the *iperf3* background traffic is kept in this test. Then, due to no bandwidth guarantee in place, the background traffic and Hadoop task traffic compete for bandwidth following the TCP fairness.

Table 4 compares the average task completion time of the two tests out of 25 runs. Although every run gives slightly different job completion time, the average job completion time when our scheme is used is always less than that without our scheme. The major reason is that our solution provides bandwidth guarantee to Hadoop nodes, which makes sure that they can transfer data robustly. In addition, they can compete for unused bandwidth through the work conservation provided by our scheme. In the test without our scheme, Hadoop nodes compete for bandwidth with background traffic, which may delay the data transfer needed to complete the tasks.

TABLE 4. Completion time (sec) of a Hadoop cluster running TestDFSIO with and without the deployment of our solution.

No. of background flows	w/o our solution	w our solution
1	863.489	692.592
2	807.036	733.454
3	825.22	747.95
4	920.52	826.2

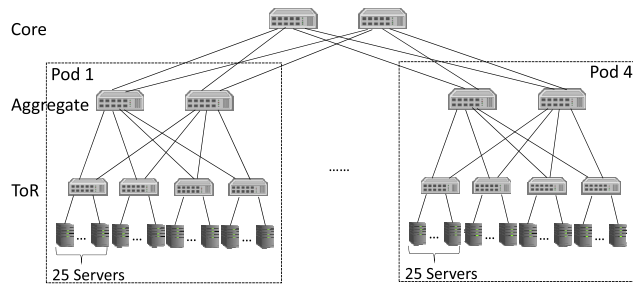


FIGURE 18. Simulation Clos topology.

Such a result demonstrates the benefits of the proposed scheme in supporting cloud applications.

F. REAL-TRACE DRIVEN LARGE SCALE SIMULATION

Since the testbed in our lab has a limited scale, we developed a simulator to evaluate the performance of the proposed system under datacenter traffic. We adopted a clos topology with 400 hosts distributed in 4 pods, as shown in Figure 18. The topology contains 2 core switches, 8 aggregate switches, and 16 top-of-rack (ToR) switches. Each ToR switch serves 25 servers. The links between servers and ToR switches, between ToR switches and aggregate switches, and between aggregate switches and core switches have a bandwidth of 1 Gbps, 10 Gbps, and 20 Gbps, respectively.

We adopted the trace from Facebook datacenter [46] to drive the generation of MapReduce jobs in the simulation. By following the common configuration of Hadoop, we converted each job to $S/64$ mappers and $S/64$ reducers, where S is the input data size in MB. We also assume that each server can hold 2 mappers and 2 reducers. We compared our system with 3 schemes: FullCompetition, BGOOnly, WCOOnly, which have no bandwidth guarantee nor work conservation, only bandwidth guarantee, and only MPTCP based work conservation, respectively. Note that the WCOOnly method attains work conservation also through a low priority MPTCP subflow. When bandwidth guarantee is enabled, we set every server to have 800 Mbps guaranteed bandwidth. We measured the amount of time for each job to finish the shuffle phase data transfer in the test.

The CDF of the shuffle phase completion time for the first 1000 jobs under the four schemes are shown in Figure 19. We also plot the average shuffle phase completion time in Figure 20. We see that the majority of the jobs are quite small and finish really quickly. However, there is a performance

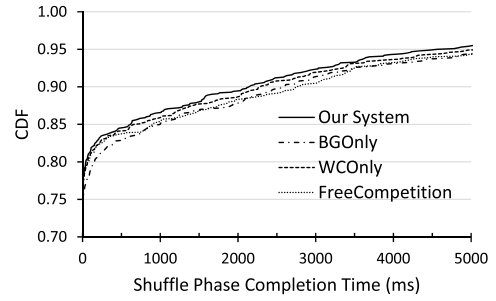


FIGURE 19. CDF of the shuffle phase completion time (only the CDF for completion time less than 5000ms is shown for better illustration).

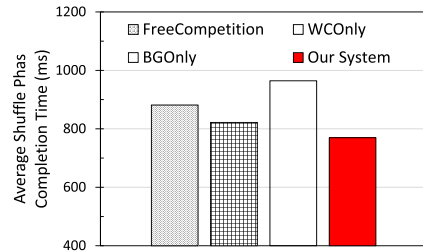


FIGURE 20. Average shuffle phase completion time.

difference regarding large jobs, which follows “Our System > WCOOnly > FreeCompetition > BGOOnly”.

BGOOnly has the worst performance because the strict bandwidth guarantee (which is implemented through rate limiting) wastes a great amount of idled bandwidth. WCOOnly and FreeCompetition perform better than BGOOnly because flows in these methods can fully utilize all bandwidth. WCOOnly has a slightly better performance than FreeCompetition because the adoption of MPTCP allows each flow to grow faster (i.e., since each flow has two subflows), which is another advantage of employing MPTCP in datacenters. Our system can also fully use the capacity of the network fabric due to the work conservation feature. Meanwhile, the bandwidth guarantee feature protects the performance of large jobs under competition with other flows. Consequently, our system performs the best.

The above results are consistent with our previous testbed based experiments, which shows the effectiveness of the proposed scheme.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel scheme that exploits MPTCP to achieve efficient, fair, and work-conserving bandwidth guarantee in cloud datacenters. The high-level ideas include: 1) let every VM generates BG traffic and WC traffic with MPTCP; and 2) conduct in-network separation of the types of traffic by assigning the BG traffic a higher priority. As a result, work conservation is efficiently supported since the WC traffic can take idled bandwidth timely and efficiently without deteriorating bandwidth guarantees. We also propose to adapt MPTCP to better serve the design goals by strictly

prioritizing the BG subflow in the MPTCP scheduler and adopting a large receive buffer size. We further enable fair work conservation by controlling the overall aggressiveness of WC subflows on each VM. The system can be easily deployed since it does not require any software or hardware components that cannot be directly supported by current commodities equipment and systems. The performance of the proposed system has been demonstrated through real testbed based experiments and trace-driven simulations. In the future, we plan to study whether we can improve the efficiency of bandwidth guarantee and work conservation by exploiting redundant paths in datacenters and bandwidth demand patterns from applications.

ACKNOWLEDGMENT

Initial result of this paper was presented at the Proceedings of ICCCN 2017 [48].

REFERENCES

- [1] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proc. IMC*, Nov. 2010, pp. 1–14.
- [2] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *Proc. SIGCOMM*, Sep. 2012, pp. 187–198.
- [3] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proc. OSDI*, Dec. 2012, pp. 349–362.
- [4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 242–253, Aug. 2011.
- [5] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS 28, 2009, no. 13.
- [6] S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and stochastic netshare: Predictable bandwidth allocation for data centers," *Comput. Commun. Rev.*, vol. 42, no. 3, pp. 5–11, 2012.
- [7] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. NSDI*, Nov. 2011, p. 23.
- [8] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. CoNext*, Nov. 2010, p. 15.
- [9] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 199–210, 2012.
- [10] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J. M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 467–478, 2014.
- [11] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. WIOV*, 2011, pp. 784–789.
- [12] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim, "EyeQ: Practical network performance isolation at the edge," in *Proc. NSDI*, 2013, pp. 297–311.
- [13] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 351–362, Aug. 2013.
- [14] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," in *Proc. INFOCOM*, Apr. 2016, pp. 1–9.
- [15] F. Douglas, L. Popa, S. Banerjee, P. Yalagandula, J. Mudigonda, and M. Caesar, "Harmonia: Tenant-provider cooperation for work-conserving bandwidth guarantees," Hewlett-Packard, Palo Alto, CA, USA, Tech. Rep. HPE-2016-16, 2016.
- [16] K. LaCurtis, J. C. Mogul, H. Balakrishnan, and Y. Turner, "Cicada: Introducing predictive guarantees for cloud networks," *Proc. HotCloud*, vol. 14, 2014, pp. 14–19.
- [17] D. M. Divakaran and M. Gurusamy, "Towards flexible guarantees in clouds: Adaptive bandwidth allocation and pricing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1754–1764, Jun. 2015.
- [18] J. Guo, F. Liu, T. Wang, and J. C. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. ATC*, 2017, pp. 69–81.
- [19] L. Yu, H. Shen, Z. Cai, L. Liu, and C. Pu, "Towards bandwidth guarantee for virtual clusters under demand uncertainty in multi-tenant clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 450–465, Feb. 2018.
- [20] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath TCP," in *Proc. NSDI*, 2012, pp. 399–412.
- [21] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, 2011.
- [22] Y. Zhang, X. Wang, S. Xiao, Z. Xu, H. Wu, X. Chen, and Y. Han, "DC²-MTCP: Light-weight coding for efficient multi-path transmission in Data center network," in *Proc. IPDPS*, Jun. 2017, pp. 419–428.
- [23] (2018). *Mptcp Linux Implementation*. [Online]. Available: <http://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP>
- [24] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *Proc. NSDI*, 2015, pp. 455–468.
- [25] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 491–502, 2014.
- [26] (2017). *Linux Traffic Control*. [Online]. Available: <http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html>
- [27] H. Yu, J. Yang, C. Xu, H. Wang, and Z. Liang, "Spongenet: Towards bandwidth guarantees of cloud datacenter with two-phase VM placement," in *Proc. NOMS*, Apr. 2016, pp. 410–417.
- [28] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC*, Nov. 2010, pp. 267–280.
- [29] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. IMC*, Nov. 2009, pp. 202–208.
- [30] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. NSDI*, vol. 13, 2013, pp. 171–184.
- [31] F. Liu, J. Guo, X. Huang, and J. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.
- [32] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, and P. Patel, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.
- [33] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.
- [34] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.
- [35] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, *Architectural Guidelines for Multipath TCP Development*, document Informational RFC 6182, IETF, 2011, pp. 1721–2070.
- [36] S. Ferlin, Ö. Alay, O. Mehani, and R. Borelli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *Proc. IFIP Netw. Conf. Workshops*, May 2016, pp. 431–439.
- [37] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol.*, Nov. 2017, pp. 147–159.
- [38] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "STMS: Improving MPTCP throughput under heterogeneous networks," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 719–730.
- [39] S. Hu et al., "Explicit path control in commodity data centers: Design and applications," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2768–2781, Oct. 2016.

- [40] C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, document RFC 6356, IETF, Oct. 2011.
- [41] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is not pareto-optimal: Performance issues and a possible solution," in *Proc CoNext*, Aug. 2012, pp. 1–12.
- [42] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 596–609, Feb. 2016.
- [43] F. Zhou, T. Dreibholz, X. Zhou, F. Fu, Y. Tan, and Q. Gan, "The performance impact of buffer sizes for multi-path TCP in Internet setups," in *Proc. AINA*, Mar. 2017, pp. 9–16.
- [44] Y. Kim, L. Xu, and K. Chon, "G-TCP to govern aggregate throughput of the parallel TCP flows," in *Proc. INFOCOM*, Apr. 2007, pp. 1–10.
- [45] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.
- [46] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," *VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.
- [47] (2015). *Apache Hadoop Releases*. [Online]. Available: <http://hadoop.apache.org/releases.html>
- [48] B. Ali and K. Chen, "Fair work-conserving bandwidth guarantees in datacenters using MPTCP," in *Proc. ICCCN*, Aug. 2017, pp. 1–9.



BARAA SAEED ALI received the B.S. degree in computer engineering and information from the University of Mosul, Iraq, in 2007, the M.S. degree in electrical and computer engineering from Southern Illinois University Carbondale, in 2013. He was a Ph.D. student with the Department of Electrical and Computer Engineering, Southern Illinois University, from 2016 to 2017. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Wayne State University. His research interests include distributed systems, cloud computing, and network protocols.



KANG CHEN received the B.S. degree in electronics and information engineering from the Huazhong University of Science and Technology, China, in 2005, the M.S. degree in communication and information systems from the Graduate University of the Chinese Academy of Sciences, China, in 2008, and the Ph.D. degree in computer engineering from Clemson University, in 2014. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale. His research interests include emerging wireless networks and software defined networking.



IMRAN KHAN received the B.S. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, in 2014. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, Southern Illinois University, Carbondale. His primary research interests include resource allocation algorithm, software defined networking, and cloud computing.

...