Intra-host Rate Control with Centralized Approach

[1]Zhuang Wang, [1]Ke Liu, [1]Yifan Shen, [2]Jack Y. B. Lee, [1]Mingyu Chen, [1]Lixin Zhang
[1]*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences*
[2]*Department of Information Engineering, The Chinese University of Hong Kong*
{wangzhuang, liuke, shenyifan, cmy, zhanglixin}@ict.ac.cn
yblee@ie.cuhk.edu.hk

***Abstract*— Today's datacenter is shared among various applications with different QoS requirements, which poses a great challenge to deliver low delay transport with high throughput. Most of works address this challenge by reducing the in-network delay, but assumes a negligible local delay. However, we show that this assumption does not hold for a multi-tenant datacenter that a physical machine is shared by multiple tenants with virtual machines running different applications. As measured, we found that VMs in a PM competing for bandwidth resources introduce delays as high as 13 ms, resulted from the packet queueing at QDisc layer of that PM, because current VMs' rate control still operates in a distributed manner without exploiting knowledge of the QoS requirements of applications running in VMs. This work addresses this problem by proposing a centralized rate adaptation (CERA) that operates in the host PM, dynamically schedules the flows from all VMs in a centralized manner. We implemented a CERA prototype and evaluated CERA through testbed experiments. Our results show that CERA reduces the local delay significantly thus reduces the average request latency of delay sensitive applications, e.g., memcached, by a factor of 6.3, without sacrificing the throughput performance of throughput intensive applications, e.g., iperf.**

I. INTRODUCTION

Datacenters are shared by various applications with diverse Quality-of-Service (QoS) requirements: delay sensitive applications, e.g., memcached [1], require low delay to minimize their flow completion time (FCT); throughput intensive applications, e.g., data-parallel computation like MapReduce [2], require high throughput for large flows; and others require both high throughput and low delay, e.g., online data-intensive applications [10]. This dynamic environment in datacenter poses a great challenge to deliver low delay transport with high throughput.

Some existing approaches [3-5] address this challenge by reducing the in-network queueing length, while assuming negligible local delay at the edge, i.e., physical machines (PM). However, this assumption is not true for multi-tenant datacenter that every PM sets up a virtual network attached by multiple Virtual Machines (VM). Those VMs could run multiple applications thus could deliver over thousands of flows with diverse QoS requirements, which could result in a large backlog at the host PM thus increase the end-to-end delay. To address this problem one intuitive idea is to apply the methods proposed to reduce the in-network delay to reduce the local delay. We employed some conventional approaches including DCTCP [4], CoDel [6] and Multiple Level Feedback Queue (MLFQ) in a PM and evaluated them with testbed experiments. We showed that, compared to the vanilla PM, they could reduce the local queueing delay by at most a factor of 4.2 when the number of flows in a VM is small, i.e., 32, but also result in a local queueing delay of at least 4 ms when the number of flows is over 50. We point out, the previous approaches are operated in a distributed manner which controls the transmission rate on a per-flow basis without exploiting the knowledge of the other flows from all VMs sharing the same PM, e.g., flows' transmission rate, the number of flows, etc.

To this end, we replace the distributed rate control used in the virtual network of a PM with a centralized rate adaptation algorithm (CERA) that continuously estimates the queueing length at QDisc. If the estimated queue length exceeds a preconfigured target length CERA first reduces the aggregated sending window of every VM weighted proportional to the number of bytes it transmitted by every VM over the past fixed interval – which implicitly estimates the current transmission rate of every VM. Second, every VM equally distributes its aggregated window allocated in the first step among all flows from that VM, both of which enable CERA to achieve high throughput and low delay.

We implemented a CERA prototype that contains two components: (1) a Linux kernel module at the QDisc of host PMs to monitor the queue length and a user-space program that determines the aggregated sending window of each VM according to the estimated queueing length; and (2) a Linux kernel module implemented at every VM kernel to allocate the sending window of every flow from that VM.

We evaluated CERA on a testbed with a virtualized network attached by VMs created with KVM [8]. In our experiments, we find that CERA reduces the average queueing delay at QDisc by at least a factor of 0.53 when the number of flows in a VM is small, e.g., 32, and 1.6 when the number of flows in a VM is large, e.g., 50, thus improves the average latency.

The remainder of the paper is organized as follows. Section II introduces the motivations of this work. Section III presents the CERA's algorithms. Section IV evaluates CERA through testbed experiments. Section V reviews the related works and Section VII concludes this work.

II. MOTIVATIONS

Our studies were motivated by the unexpectedly large delay observed in the large scale virtual network of a host PM, thus we report the results of local delay measured from a virtual network built in a PM of our testbed, and find out the primary source of that delay.

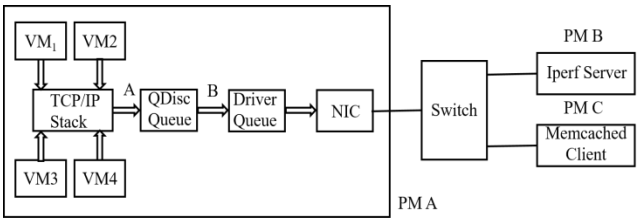

Fig. 1. Experiments setup for validating local delays.

In this section, we validate the problem that a large delay could be observed in virtual networks of host PMs. We conduct experiments over a testbed depicted in Fig. 1, consisting 3 PMs that are connected by a switch, where PM A sets up a virtual network attached by multiple VMs that send packets to PM B via a bridge connecting the physical NIC with 1Gbps link capacity. Specifically, we create four VMs in PM A's virtual network, one of them runs a memcached [1] using *memaslap* load generator [9] and measures its request latencies.

As shown in Fig. 2, the average latency of memcached requests are below 1ms that is mainly contributed by the transmission delay. However, the request latencies of memcached increase up to over 13ms when it concurrently runs with 8 *iperf* flows running in the other 3 VMs respectively. We measure that the average delay at QDisc queue is around 12ms.

III. CERA ALGORIHTMS

The idea of using the centralized approach in a datacenter is not new. Ghobadi et al. [11] first proposed a central controller, OpenTCP, to adjust TCP parameters and variants, e.g., initial congestion window size and retransmission time out (RTO) according to the measured network state and dynamics during that period. Perry et al. [3] proposed Fastpass that is a central arbiter instructing *every* host PM to control its *every* packet transmission timing explicitly over a set of timeslots to achieve the goals of both network-wide high bandwidth efficiency and near-zero queueing delay. However, the delay of the control loop, i.e., between the controller/arbiter and PMs, impacts the accuracy of the rate controls and the control packets also compete with normal traffics for bandwidth.

We seek a feasible solution that exploits the knowledge of flows from all VMs sharing the same PM and dynamically controls their transmission rates in a centralized manner. Similar to Fastpass [3] and OpenTCP [11], CERA algorithms are divided into two components: (1) the *central controller* running in the host PM. If the estimated queue length exceeds a preconfigured target length, it determines the aggregated sending window of each VM weighted proportional to the number of bytes it transmitted by that VM over the past fixed interval – which implicitly estimates the current transmission rate of every VM; and (2) *CERA client* running in every VM distributes the window reduction among all flows from that VM according to their current sending windows.

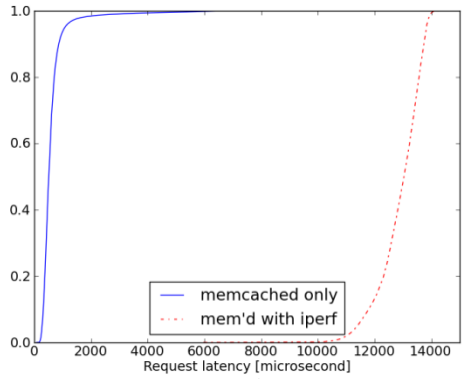

Fig. 2. The cumulative distribution of memcached requests latencies.

A. Central Controller

The central controller continuously monitors the queue length at QDisc of the host PM by implementing a loadable kernel module. Specifically, we divide time into small fixed epoch of Δ milliseconds in duration. The central controller monitors the queue length at the end of each epoch. In the meantime, the central controller intercepts every outgoing packet just before they are passed to the NIC thus determines the number of packets transmitted by every VM enqueued at QDisc during the past $M\Delta$ milliseconds – where M controls the duration of the estimation interval, by looking into the source IP address of packets' IP headers.

Let q_i denote the queue length at the end of the i^{th} epoch; $n_{k,i}$ denote the number of packets transmitted by the k^{th} VM during the past $M\Delta$ milliseconds at the end of the i^{th} epoch, thus $n_{k,i}$ implicitly estimates the sending rate of the k^{th} VM at the end of i^{th} epoch, i.e., $n_{k,i}/M\Delta$. If $q_i > T_u$, where T_u denotes the upper bound of the target queue length, the central controller reduces the aggregated sending window of every VM weighted proportional to the sending rate of that VM so that the queue length will oscillate between T_u and T_l, where T_l denotes the lower bound of the target queue length and $T_l < T$. Let d denote the RTT and W_i the current total aggregated sending window allocated for all VMs during the i^{th} epoch. To reduce the queue length to T_l we have

$$W_i = dC / S + T_l \tag{1}$$

where S denotes the packet size and C denotes the capacity of the NIC.

Let $\vec{k}_i$ denote the set of VMs that transmits nonzero packets during the past $M\Delta$ at the end of the i^{th} epoch; μ_k denote the weight for VM k; $w_{k,i}$ denote the aggregated sending window allocated for VM k at the i^{th} epoch, thus we have

$$w_{k,i} = \left\lfloor \frac{\mu_k n_{k,i}}{\sum_{\forall k \in \vec{k}} \mu_k n_{k,i}} W_i \right\rfloor \tag{2}$$

If μ_k=1 for all k, (2) becomes the ECN-based rate control such as DCTCP. After deriving w_k, the central controller sends w_k to VM k, for $\forall k \in \vec{k}_i$, using TCP connection that is established when VM k is created.

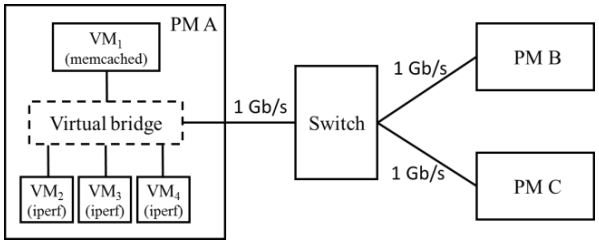

Fig. 3. Network testbed setup.

B. CERA Client

Once a VM, e.g., VM *k*, is created, CERA client establishes a TCP connection with the central controller thus can receive the aggregated window from the controller, i.e., $w_{k,i}$. In the meantime, VM *k* keeps track of all the active TCP flows. Specifically, when a TCP flow is established, it is added to a *circular list*, and is deleted from the list if that TCP flow is closed. Because the receiving window (AWnd) cannot become the throughput bottleneck by default until loss events are detected, we only modify the CWnd that mainly affects the sending window.

Let $cw_{m,k,i}$ denote the CWnd of the m^{th} TCP flow of the k^{th} VM at the end of the i^{th} epoch after the window distributions; $\vec{M}$ denote the set of the active TCP flow, the algorithm to distribute the aggregated sending window among all the active TCP flows is summarized into the following two ways:

(i) CERA client *equally* allocates CWnds among TCP flows in $\vec{M}$, thus we have,

$$cw_{m,k,i} = \left\lfloor w_{k,i} / \left|\vec{M}\right| \right\rfloor \quad (3)$$

But (3) will result in the remaining windows as $cw_{m,k,i}$ takes the floor of $w_{k,i} / \left|\vec{M}\right|$, thus the remaining windows is

$$w_{k,i} - cw_{m,k,i}\left|\vec{M}\right| \quad (4)$$

(ii) In order to efficiently make full use of the remaining windows CERA client allocates (4) one by one to those TCP flows in a round-robin manner. It is unfair to allocate (4) always from the head of the circular list, because the flows in the tail of the list do not have equal chance to obtain the remaining windows, hence we record the position of the last updated TCP flow in the circular list during the last epoch, and allocate (4) from the next one of that recorded position until (4) becomes zero.

After step (i) and step (ii), every TCP's CWnd will be governed by its TCP variant employed, e.g., TCP CUBIC, until the next end of epoch.

IV. PERFORAMCNE EVALUATIONS

A. Network Testbed Setup

We evaluate CERA over a testbed consisting of three Intel Xeon-based physical machines. As shown in Fig. 3, every PM adopts Linux kernel 3.10.25. PM A setups 4 VMs using KVM. Every VM also adopts Linux Kernel 3.10.25, which can access the networks to which a physical NIC is connected via a bridge. The queue length at QDisc of the bridge is set to 1000 packets by default. Three PMs are connected with each other through a physical switch with 1Gbps capacity. Similarly, 3 VMs, i.e., VM 2-4, in PM A are running different number of *iperf* flows using TCP to emulate different levels of traffic loads. The other VM, i.e., VM 1 in PM A, generates memcached requests using *memaslap* load generator from *libmemcached* v1.0.18 [9] using TCP. The GET and SET requests ratio of memcached requests is 9:1 and the concurrent number of requests is 32. The underlying transport protocol is TCP CUBIC by default in the following experiments unless specified.

B. Througput-delay Tradeoff

We evaluated the throughput-delay tradeoffs for various approaches including DCTCP, CoDel, FQ_CoDel and CERA by starting 8 *iperf* TCP flows from VM 2-4 and 32 memcached requests from VM 1 simultaneously. We also use the default TCP variant, TCP CUBIC, in current Linux platform as the underlying transport protocol and default TC module *pfifo_fast*, i.e., FIFO in Fig. 4, in our testbed experiments. We also started the memcached requests only to derive the optimal delays achievable as ground truth, i.e., memcached_only in Fig. 2. We tested two versions of CERA with the target queue length set to 25 and 60, i.e., CERA_25 and CERA_60, respectively.

As shown in Fig. 4 that plots the cumulative distributions of the queueing delay experienced by both *iperf* and memcached packets at QDisc achieved by the above approaches, CERA with the target queue length set to 25 achieves the minimal average queueing delay, 1.38 ms, which reduces the average queueing delay by a factor of 0.53 and 8.4 compared to DCTCP and TCP CUBIC, while always outperforms DCTCP in total throughput performance regardless of the number of *iperf* flows. CERA with the target queue length set to 60 also outperform DCTCP and TCP CUBIC in average queueing delay by a factor of 0.35 and 7.3, increasing the average queueing delay by only 12%, i.e., 1.57 versus 1.38 ms, compared to CERA with the target queue length set to 25, but achieves near-100% bandwidth utilization regardless the number of *iperf* flows, e.g., near full bandwidth utilization for 1 *iperf* flow, outperforming DCTCP by 40% as shown in Fig. 5, while CERA with the target queue length set to 25 achieves 75% bandwidth utilization when the number of *iperf* flows is set to 1. To achieve near-100% bandwidth efficiency for any number of *iperf* flows we set 60 for the target queue length by default, which can be tuned for a different traffic distribution thus we will evaluate the sensitivity of the target queue length over different traffic distributions in one of our future works. The other approaches also achieve the near full bandwidth utilization but result in an even larger average queueing delay.

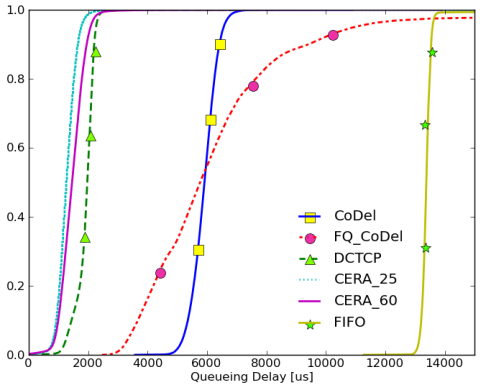

Fig. 4. The cumulative distributions of queuing delays achieved by different approaches.

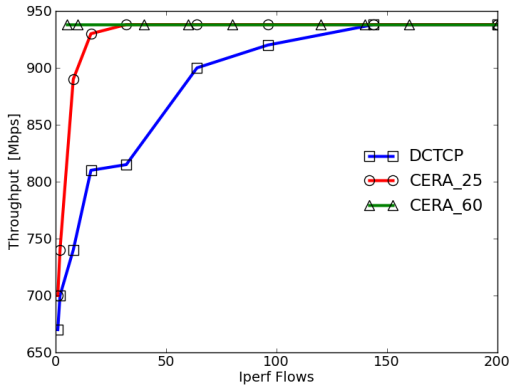

Fig. 5. Total throughput comparison between DCTCP and CERA with two target queue lengths.

V. Related Works

There are numerous recent works [3, 4, 5, 6, 11, 12, 13, 14] on improving in-network delay in datacenter networks, which can be classified into three categories: End-to-End approaches (E2E), active queue managements (AQM) and centralized approaches. In this section we literally introduce those works and compare them to CERA.

E2Es such as DCTCP [4] generally employ a novel rate control at end-hosts. TIMELY [12] revisits RTT to detect the network congestion for datacenter networks. Specifically, TIMELY leverages NIC hardware to make RTT measurements with microsecond accuracy thus adjusts transmission rates with RTT gradients to keep packet latency low while delivering high bandwidth utilization. TIMELY achieves more accuracy of in-network RTT by only considering RTT between any two NICs, while CERA improves delay from VMs to NIC. D2TCP [13] adds deadline-awareness on the top of DCTCP, thus adjusts CWnds based on both ECN and deadline information to meet deadlines, thus they also suffer from the problems in DCTCP if it is used in VMs.

AQMs such as CoDel [6] and ECN [14] that drops or marks queueing packets with congestion indicators such as the queue length and delay, and endpoints might also need to be modified to react to this signals to adjust their transmission rates.

Acknowledgment

We are thankful to all anonymous reviewers for their helpful feedback. The research was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61331008, 61502459, 61221062 and 61521092, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401.

VI. Conclusion and Future Works

This paper first reveals that VMs in a PM competing for bandwidth resources introduce a local delay as high as 13 ms which results from the packet queueing at QDisc layer of that PM, and addresses this problem by proposing CERA that adjusts the transmission rate of every VM/flow by using a centralized approach. Extensive testbed experiments showed that CERA achieves the best the throughput-delay tradeoff compared to other existing approaches. In the future we plan to evaluate CERA over various traffic distributions, e.g., web search, data mining, etc., and further improve CERA implementations, e.g., multi-level weight assignments, and test it in a real datacenter network.

References

[1] N. Rajesh, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung and V. Venkataramani, "Scaling Memcache at Facebook," in *USENIX NSDI*, 2013.

[2] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters," in *USENIX OSDI*, 2004.

[3] P. Jonathan, A. Ousterhout, H. Balakrishnan, D. Shah and H. Fugal, "Fastpass: A Centralized Zero-queue Datacenter Network," in *ACM SIGCOMM*, 2015.

[4] A. Mohammad, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2011.

[5] A. Mohammad, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, M. Yasuda, "Less is More: Trading a little Bandwidth for Ultra-low Latency in the Data Center," in *USENIX NSDI*, 2012.

[6] N. Kathleen and V. Jacobson. "Controlling Queue Delay," in *ACM Queue*, 2012.

[7] Microsoft SQL Server, Available: http://www.microsoft.com/en-us/server-cloud/products/sql-server/.

[8] KVM, Available: http://linux-kvm.org.

[9] Libmemcached, Available: http://libmemcached.org/.

[10] FQ_Codel, available: https://tools.ietf.org/html/draft-hoeiland-joergensen-aqm-fq-codel-01

[11] G. Monia, S. H. Yeganeh and Y. Ganjali. "Rethinking End-to-End Congestion Control in Software-defined Networks," in *ACM HotNet*, 2012.

[12] M. Radhika, T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM*, 2015.

[13] V. Balajee, J. Hasan and T. N. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *ACM SIGCOMM*, 2012.

[14] K. Ramakrishnan, S. Floyd and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," in *RFC 3168*, 2011