ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

IJIRSET

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)

On 21st & 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Network Congestion Control over TCP Using BW Estimation Technique

Arun Kumar.R.S.,

Department of Computer Science, Arunai Engineering College, Tiruvannamalai, Tamilnadu, India.

ABSTRACT—When multiple synchronized servers send data to the same receiver in parallel Transport Control Protocol (TCP) incast congestion happens in high-bandwidth and low-latency networks.In data-center applications such as MapReduce and Search, this many-to-one traffic pattern is frequent. Hence TCP incast congestion may severely degrade their performances, e.g., by increasing response time. In this paper, we study TCP incast in detail by focusing on the relationships between TCP throughput, round-trip time (RTT), and receive window. Unlike previous approaches, which mitigate the impact of TCP incast congestion by using a fine-grained timeout value, ouridea is to design an Incast congestion Control for TCP (ICTCP) scheme on the receiver side. In particular, our method adjusts the TCP receive window proactively before packet loss occurs. The implementation and experiments shows that almost zero timeouts and high goodput for TCP incast is achieved.

KEYWORDS—Data-Center Networks, Incast Congestion, Synchronized Servers , TCP.

I. INTRODUCTION

Transport Control Protocol (TCP) is widely used on the Internet and generally works well. However, recent studies [1], [2] have shown that TCP does not work well for many-to-one traffic patterns on high-bandwidth, low-latency networks. Congestion occurs when many synchronized servers under the same Gigabit Ethernet switch simultaneously send data to one receiver in parallel. Only after all connections have finished the data transmission can the next round be issued. Thus, these connections are also called barrier-synchronized. The final performance is determined by the slowest TCP connection,which may suffer from timeout due to packet loss. The performance collapse of these many-to-one TCP connections is called TCP incast congestion.

The traffic and network conditions in data-center networks create the three preconditions for incast congestion as summarized in [2]. First, data-center networks are well structured and layered to achieve high bandwidth and low latency, and the buffer size of top-of-rack (ToR) Ethernet switches is usually small. Second, a recent measurement study showed that a barrier-synchronized many-to-one traffic pattern is common in data-center networks [3], mainly caused by MapReduce [4] and similar applications in data centers. Third, the transmission data volume for such traffic patterns is usually small, e.g.,ranging from several hundred kilobytes to several megabytes in total.

The root cause of TCP incast collapse is that the highly bursty traffic of multiple TCP connections overflows the Ethernet switch buffer in a short period of time, causing intense packet loss and thus TCP retransmission and timeouts. Previous solutions focused on either reducing the wait time for packet loss recovery with faster retransmissions [2], or controlling switch buffer occupation to avoid overflow by using ECN and modified TCP on both the sender and receiver sides [5].

This paper focuses on avoiding packet loss before incast congestion, which is more appealing than recovery after loss. Of course, recovery schemes can be complementary to congestion avoidance. The smaller the change we make to the existing system, the better. To this end, a solution that modifies only the TCP receiver is preferred over solutions that require switch and router support (such as ECN) and modifications on both the TCP sender and receiver sides.

The idea is to perform incast congestion avoidance at the receiver side by preventing incast congestion. The receiver side is a natural choice since it knows the throughput of all TCP connections and the available bandwidth. The receiver side can adjust the receive window size of each TCP connection, so the aggregate burstiness of all the synchronized senders are kept under control. We call our design Incast congestion Control for TCP (ICTCP).

However, adequately controlling the receive window is challenging: The receive window should be small enough to avoid incast congestion, but also large enough for good performance and other nonincast cases. A well-performing throttling rate for
one incast scenario may not be a good fit for other scenarios due to the dynamics of the number of connections, traffic volume,network conditions, etc.

This paper addresses the above challenges with a systematically designed ICTCP. We first perform congestion avoidance at the system level. We then use the per-flow state to finely tune the receive window of each connection on the receiver side. The technical novelties of this work are as follows:

1) To perform congestion control on the receiver side, we use the available bandwidth on the network interface as a quota to coordinate the receive window increase of all incoming connections.

2) Our per-flow congestion control is performed independently of the slotted time of the round-trip time (RTT) of each connection, which is also the control latency in its feedback loop.

3) Our receive window adjustment is based on the ratio of the difference between the measured and expected throughput over the expected. This allows us to estimate the throughput requirements from the sender side and adapt the receiver window accordingly.

We also find that live RTT is necessary for throughput estimation as we have observed that TCP RTT in a high-bandwidth low-latency network increases with throughput, even if link capacity is not reached We have developed and implemented ICTCP as a Windows Network Driver Interface Specification (NDIS) filter driver.

Our implementation naturally supports virtual machines that are now widely used in data centers. In our implementation, ICTCP as a driver is located in hypervisors below virtual machines. This choice removes the difficulty of obtaining the real available bandwidth after virtual interfaces' multiplexing. It
also provides a common waist for various TCP stacks in virtual machines. We have built a testbed with 47 Dell servers and a 48-port Gigabit Ethernet switch. Experiments in our testbed demonstrated the effectiveness of our scheme.

The rest of this paper is organized as follows. Section II discusses research background. Section III describes the design rationale of ICTCP. Section IV presents the ICTCP algorithms. Section VI shows the implementation of ICTCP as a Windows driver. Section VII presents experimental results. Section VIII discusses the extension of ICTCP. Section IX presents related work. Finally, Section X concludes the paper.

II. BACKGROUND AND MOTIVATION

TCP incast has been identified and described by Nagle et al. [6] in distributed storage clusters. In distributed file systems, the files are deliberately stored in multiple servers. However, TCP incast congestion occurs when multiple blocks of a file are fetched from multiple servers at the same time.

Several application-specific solutions have been proposed in the context of parallel file systems. With recent progress in data-center networking, TCP incast problems in data-center networks have become a practical issue. Since there are various data-center applications, a transport-layer solution can obviate the need for applications to build their own solutions and is therefore preferred.

In this section, we first briefly introduce the TCP incast problem, then illustrate our observations for TCP characteristics on high-bandwidth, low-latency networks. Next, we explore the root cause of packet loss in incast congestion, and finally, after observing that the TCP receive window is the right controller to avoid congestion, we seek a general TCP receive window adjustment algorithm.

A. *TCP Incast Congestion*

In Fig. 1, we show a typical data-center network structure. There are three layers of switches/routers: the ToR switch, the Aggregate switch, and the Aggregate router. We also show a detailed case for a ToR connected to dozens of servers. In a typical setup, the number of servers under the same ToR ranges from 44 to 48, and the ToR switch is a 48-port Gigabit switch with one or multiple 10-Gb uplinks.

Incast congestion happens when multiple sending servers under the same ToR switch send data to one

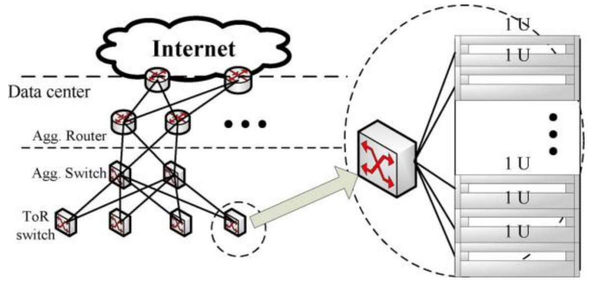

Fig. 1. Data-center network and a detailed illustration of a ToR switch connected to multiple rack-mounted servers.

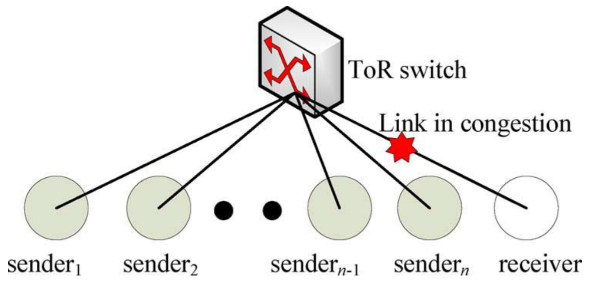

Fig. 2. Scenario of incast congestion in data-center networks, where multiple (n) TCP senders transmit data to the same receiver under the same ToR switch.

receiver server simultaneously, as shown in Fig. 2. The amount of data transmitted by each connection is relatively small, e.g, 64 kB. In Fig. 3, we show the goodput achieved on multiple connections versus the number of sending servers. Note that we use the term goodput as it is effective throughput obtained and observed at the application layer. The results are measured on a testbed with 47 Dell servers (at most 46 senders and one receiver) connected to one Quanta LB4G 48-port Gigabit switch. The multiple TCP connections are barrier-synchronized in our experiments. We first establish multiple TCP connections between all senders and the receiver,

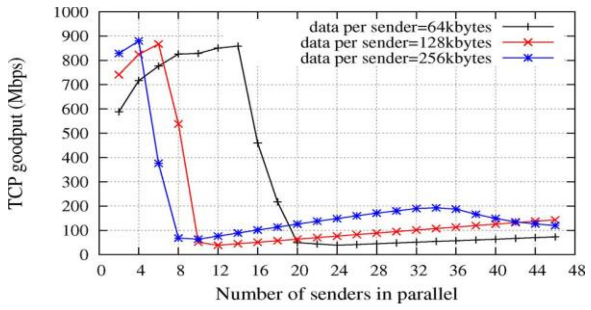

Fig. 3. Total goodput of multiple barrier-synchronized TCP connections versus the number of senders, where the data traffic volume per sender is a fixed amount.

simultaneously, as shown respectively. Then, the receiver sends out a (very small) request packet to ask each sender to transmit data, respectively, i.e., multiple requests packets are sent using multiple threads. The TCP connections are issued round by round, and one round ends when all connections on that round have finished their data transfer to the receiver.

We observe similar goodput trends for three different traffic amounts per server, but with slightly different transition points. Note that in our setup, each connection has the same traffic amount with the number of senders increasing, which is used in [1]. Reference [2] uses another setup, in which the total traffic amount of all senders is a fixed one, so that the data volume per server per round decreases when the number of senders increases. Here, we just illustrate the incast congestion problem and later will show the results for both setups in Section VII.

TCP throughput is severely degraded by incast congestion since one or more TCP connections can experience timeouts caused by packet drops. TCP variants sometimes improve performance, but cannot prevent incast congestion collapse since most of the timeouts are caused by full window losses [1] due to Ethernet switch buffer overflow. The TCP incast scenario is common for data-center applications. For example, for search indexing we need to count the frequency of a specific word in multiple documents.

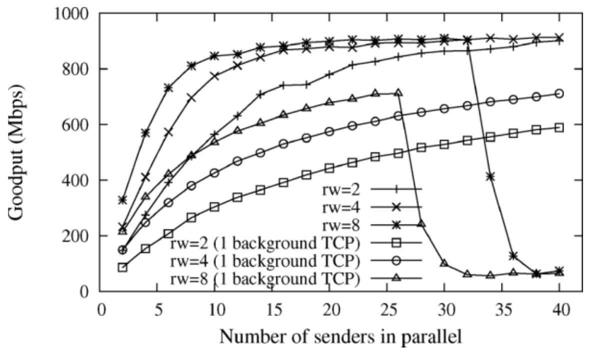

Fig. 4. Incast goodput of capped and fixed receive window (rw) with and without background TCP connection. The value of rw is with MSS, and the background TCP connection is not capped by the receive window.

V. ANALYSIS

In this section, we present a simplified flow model for ICTCP in a steady state. In the steady-state control loop, we assume that ICTCP has successfully optimized the receive window of all TCP flows.We are interested in how much buffer space ICTCP needs in the steady state. We use it to judge whether the buffer space requirement of ICTCP is reasonable for existing low-end Ethernet switch buffer space in practice.

We assume that there are infinitely long-lived TCP flows under the control of ICTCP. All flows go to the same destination server as shown in Fig. 2. The receive window size of these flows is assumed to be ωi, i=1,…,n. The RTT of the flows is denoted as ri, i=1,…..,n. The link capacity is denoted as C, and packet size is denoted as δ.

We assume the base RTT for those flows is the same, and denoted by . Like the RTT shown in Fig. 5, the base RTT is the least RTT experienced for a connection when there is no queue at the switch. As the queue builds up at the output port of the switch, the RTT of all flows keeps growing. Correspondingly, the base BDP (BDP without queuing) is , where is the bottleneck link capacity.

For connection i, its base BDP is denoted as . In the steady state, we assume that all the connections have occupied the base BDP, i.e.,

$$\sum_{i=1}^{n} bdp\, i = \text{R.C}$$

Therefore, in the steady state, the number of data packets for connection in the switch buffer is bounded by,

$$Qi \leq \omega i - bdpi/\delta$$

To meet the condition where there are no packet drops for any connection, we have the buffer size as

$$Q \geq \sum_{i=1}^{n} qi\delta i$$

We consider the worst case where a synchronization of all connections happens. In this case, the packets on the flight are all data packets, and there are no ACK packets in the backward direction. For the worst case under this assumption, all those data packets are in the forward direction, in transmission, or waiting in the queue. To ensure there is no packet loss, the queue size should be large enough to store packets on the flight for all connections.Hence, in the worst case, if all connections have the same receive window , we have

$$Q \geq n\omega\delta - R.C.$$

The synchronization of all connections is likely in an incast scenario, as the senders are transmitting data to the receiver almost at the same time. Note that the synchronization assumption greatly simplifies the receive window adjustment of ICTCP. In practice, packets arrive at the receiver in order, making the connections with earlier packet arrivals have a larger receive window.

We calculate the buffer size requirement to avoid incast buffer overflow from the above equation using an example of a 48-port Ethernet switch. The link capacity C is 1 Gb/s, and the base RTT R is 100µs, so the BDP R.C=12500B. The default packet length MSS on Ethernet is 1500 B. The minimal receive window is by default 2MSS. When considering a 48-port Gigabit Ethernet switch, to avoid packet loss in a synchronized incast scenario with a maximum of 47 senders, the buffer space should be larger than 47*2*1500-12500=128.5 kB. Preparing for the worst case, incast congestion may happen at all ports simultaneously, then the total buffer space required is
128.5*48 ≈6 MB.

We then consider the case of low-end ToR switches, which are also called switches with a shallow buffer in [5]. The switch in our testbed is a Quanta LB4G 48-port Gigabit Ethernet switch, which has a small buffer of about 4 MB. Correspondingly, by a reverse calculation, if the buffer size is 4 MB and the minimal receive window is 2MSS, then at most 32 senders can be supported. Fortunately, existing commodity switches use dynamic buffer management to allocate a buffer pool shared by all ports. That is to say, if the number of servers is larger than 32, incast congestion may happen with some probability, which is determined by the usage of the shared buffer pool.

III. EXPERIMENTAL RESULTS

We deployed a testbed with 47 servers and one Quanta LB4G 48-port Gigabit Ethernet switch. The topology of our testbed was the same as the one shown on the right side of Fig. 1, where 47 servers each connect to the 48-port Gigabit Ethernet switch with a Gigabit Ethernet interface. Each server has two 2.2-GB Intel Xeon CPUs E5520 (four cores), 32 GB RAM, a 1-TB hard disk, and one Broadcom BCM5709C NetXtreme II Gigabit Ethernet NIC.

TheOS on each server isWindows Server 2008 R2 Enterprise 64-bit version. The CPU, memory, and hard disk were never a bottleneck in any of our experiments. We use iperf to construct the incast scenario wheremultiple sending servers generate TCP traffic to a receiving server under the same switch. The servers in our testbed have their own background TCP connections for various services, but the background traffic amount is very small compared to our generated traffic. The testbed is in an enterprise network with normal background broadcast traffic. All comparisons are between a full implementation of ICTCP described in Section VI and a state-of-the-art TCP New Reno with SACK implementation on a Windows server. The default timeout value of TCP on aWindows server is 300 ms. Note that all the TCP stacks were the same in our experiments, and ICTCP was implemented on a filter driver at the receiver side.

A. *Fixed Traffic Volume per Server With the Number of Senders Increasing*

The first incast scenario we considered was one in which a number of senders generate the same amount of TCP traffic to a specific receiver under the same switch. Similar to the setup in [1] and [14], we fix the traffic amount generated per sending server.

The TCP connections are barrier-synchronized per round, i.e., one round finishes only after all TCP connections in it have finished, and then the next round starts. The goodput shown is the average value of 100 experimental rounds. We observe the incast

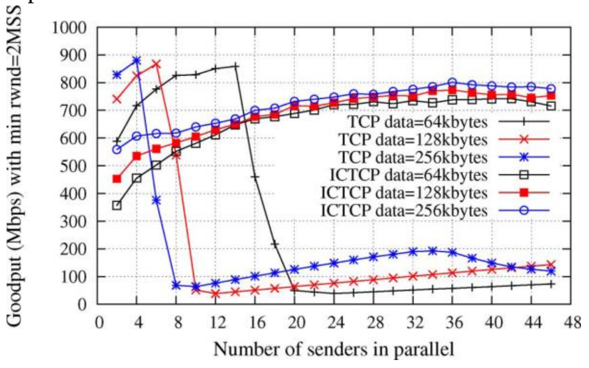

Fig. 8. Total goodput of multiple barrier-synchronized ICTCP/TCP connections versus the number of senders, where the data traffic volume per sender is a fixed amount.

congestion: With the number of sending servers increasing, the goodput per round actually drops due to TCP timeout on some connections. The smallest number of sending servers to trigger incast congestion varies with the traffic amount generated per server: With a larger amount of data, a smaller number of sending servers is required to trigger incast congestion.

1) ICTCP With Minimal Receive Window at 2MSS: Under the same setup, the performance of ICTCP is shown in Fig. 8. We observe that ICTCP achieves smooth and increasing goodput with the number of sending servers increasing. A larger data amount per sending server results in a slightly higher goodput. The averaged goodput of ICTCP shows that incast congestion is effectively throttled. The goodput of ICTCP, with a varying number of sending servers and traffic amount per sending servers, shows that our algorithm adapts well to different traffic requirements. We observe that the goodput of TCP before incast congestion is actually higher than that of ICTCP.

For example,TCP achieves 879 Mb/ps while ICTCP achieves 607 Mb/s with four sending servers at 256 kB per server. There are two reasons: 1) During the connection initiation phase (slow start), ICTCP increases the receive window slowly than TCP increases the congestion window. Actually, ICTCP doubles the receive window at least every two RTTs, while TCP doubles its congestion window every RTT. Otherwise, ICTCP increases the receive window by 1 MSS when the available bandwidth is low. 2) The traffic amount per sending server is very small, and thus the time taken in the “slow-start” dominates the transmission time if incast congestion does not occur. Note that the low throughput of ICTCP during the initiation phase does not affect its throughput during the stable phase in a larger

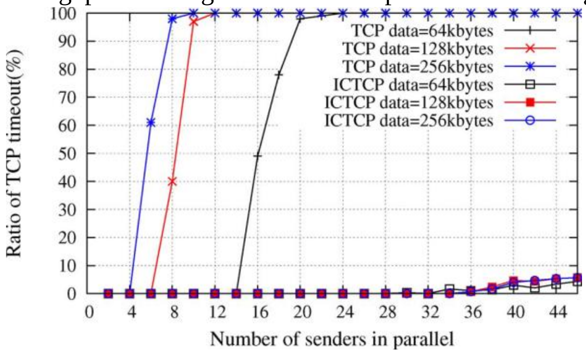

Fig. 9. Ratio of experimental rounds that suffer at least one timeout.

timescale, e.g., hundreds of milliseconds, which will be evaluated in Section VII-D.

To evaluate the effectiveness of ICTCP in avoiding timeouts, we use the ratio of the number of experimental rounds experiencing at least one timeout4 over the total number of rounds. The ratio of rounds with at least one timeout is shown in Fig. 9. We observe that TCP suffers at least one timeout when incast congestion occurs, while the highest ratio for ICTCP experiencing timeout is 6%. Note that the results in Fig. 9 show that ICTCP is better than DCTCP[5], as DCTCP quickly downgrades to the same as TCP when the number of sending servers is over 35 for the static buffer. The reason that ICTCP effectively reduces the probability of timeouts is that ICTCP avoids congestion and increases the receive window only if there is enough available bandwidth on the receiving server. DCTCP relies on ECN to detect congestion, so a larger (dynamic) buffer is required to avoid buffer overflow during control latency, i.e., the time before control takes effect.

2) ICTCP With Minimal Receive Window at 1MSS: ICTCP has a possibility (although very small) to timeout since we use a 2MSS minimal receive window. In principle, with the number of connections becoming larger, the receive window for each connection should become smaller proportionately. This is because the total BDP including the buffer size is actually shared by those connections, and the minimal receive window of those connections determines whether such sharing may cause buffer overflow when the total BDP is not enough to support those connections.

The performance of ICTCPwith aminimal receivewindow at 1MSS is shown in Fig. 10. We observe that timeout probability is 0, while the averaged throughput is lower than those with a 2MSS minimal receive window. For example, for 40 sending serverswith 64 kB per server, the goodput is

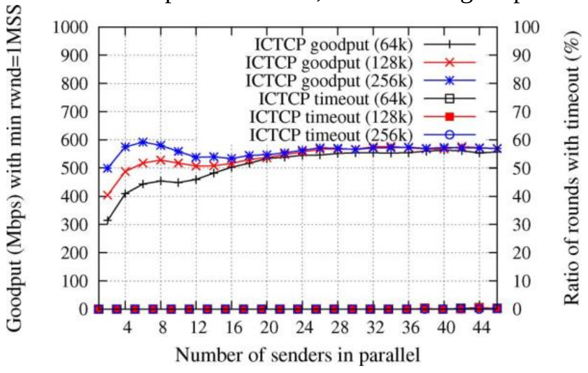

Fig. 10. ICTCP goodput and ratio of experimental rounds suffer at least one timeout with a minimal receive window of 1MSS.

741 Mb/s for 2MSS as shown in Fig. 8, while it is 564 Mb/s for 1MSS as shown in Fig. 10. Therefore, the minimal receive window is a tradeoff between a higher average incast goodput and a lower timeout probability.

Note that the goodput here only lasts for a very short time, 40*64k*8/564 Mb/s ms. For a larger data size request and a longer connection duration, ICTCP actually achieves a goodput that is close to link capacity, which is shown in detail in Section VII-D.

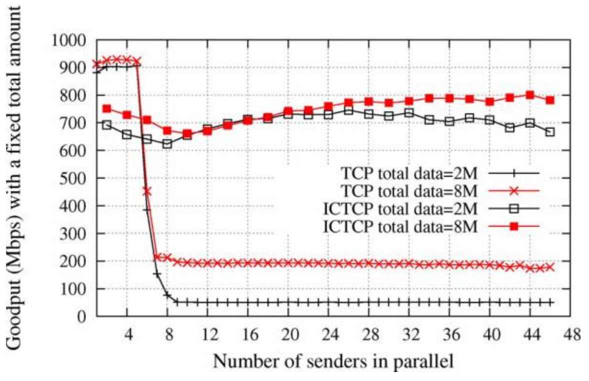

Fig. 11. Goodput of ICTCP (with a minimal receive window of 2MSS) and TCP in the case that the total data amount from all sending servers is a fixed value.

IV. DISCUSSIONS

In this section, we discuss three issues related to the further extension of ICTCP. The first issue regards the scalability of ICTCP, in particular, how to handle incast congestion with an extremely large number of connections. Reference [5] shows that the number of concurrent connections to a receiver of 50 ms duration is less than 100 for the 90th percentile in a real data center. ICTCP can easily handle a case of 100 concurrent connections with 1MSS as the minimum receive window.

In principle, if the number of concurrent connections becomes extremely large, then we require a much smaller minimal receive window to prevent buffer overflow. However, directly using a receive window less than 1 MSS may degrade performance greatly. We propose an alternative solution: switching the receive window between several values to effectively achieve a smaller receive window averaged for multiple RTTs. For example, a 1MSS window for one RTT and a 0 window for another RTT could achieve a 0.5MSS window on average for 2RTT. Note that it still needs coordination between multiplexed flows at the receiver side to prevent concurrent connections overflow buffers.

The second issue we consider is how to extend ICTCP to handle congestion in general cases where the sender and the receiver are not under the same switch and the bottleneck link is not the last hop to the receiver. ECN can be leveraged to obtain such congestion information. However, it differs from the original ECN, which only echoes the congestion signal on the receiver side, because ICTCP can throttle the receive window considering the aggregation of multiple connections. The third issue is whether ICTCP will work for future highbandwidth low-latency networks. A big challenge for ICTCP is that the bandwidth may reach 100 Gb/s, while the RTT may not decrease by much. In this case, the BDP is enlarged, and the receive window on incast connections also becomes larger. While in ICTCP, a 1MSS reduction is used for window adjustment, requiring a longer time to converge if the window size is larger. To make ICTCP work for a 100-Gb/s or even higher bandwidth network, we consider the following solutions:1) the switch buffer should be enlarged correspondingly;2) the MSS should be enlarged so that the window size with regard to the MSS number does not enlarge greatly. This is reasonable as a 9-kB MSS is available for Gigabit Ethernet.

V. CONCLUSION

In this paper, I have presented the design, implementation, and evaluation of ICTCP to improve TCP performance for TCP incast in data-center networks. In contrast to previous approaches that used a fine-tuned timer for faster retransmission, it focus on a receiver-based congestion control algorithm to prevent packet loss. ICTCP adaptively adjusts the TCP receive window based on the ratio of the difference of achieved and expected per-connection throughputs over expected throughput, as well as the last-hop available bandwidth to the receiver. Here, a lightweight and high-performance Window NDIS filter driver to implement ICTCP is developed. Compared to directly implementing ICTCP as part of the TCP stack, our driver implementation can directly support virtual machines, which are becoming prevalent in data centers. A testbed is build with Multiple servers along with a 48-port Ethernet Gigabitswitch. The experimental results exhibit that ICTCP is effective in avoiding congestion by achieving almost zero timeouts for TCP incast, and it provides high performance and fairness among competing flows.

REFERENCES

[1] A. Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, *“Measurement and analysis of TCP throughput collapse in cluster-based storage systems,”* in *Proc. USENIX FAST*, 2008, Article no. 12.
[2] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, *“Safe and effective fine-grained TCP retransmissions for datacenter communication,*” in *Proc. ACM SIGCOMM*, 2009, pp. 303–314.
[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, *“The nature of data center traffic: Measurements & analysis,”* in *Proc. IMC*, 2009, pp. 202–208.
[4] J. Dean and S. Ghemawat, *“MapReduce: Simplified data processing on large clusters,”* in *Proc. OSDI*, 2004, p. 10.
[5] M. Alizadeh, A. Greenberg, D.Maltz, J. Padhye, P. Patel, B.Prabhakar, S. Sengupta, and M. Sridharan, *“Data center TCP (DCTCP),”* in *Proc. SIGCOMM*, 2010, pp. 63–74.
[6] D. Nagle, D. Serenyi, and A. Matthews, *“The Panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage,”* in *Proc. SC*, 2004, p. 53.
358 IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 21, NO. 2, APRIL 2013
[7] E. Krevat, V. Vasudevan, A. Phanishayee, D. Andersen, G. Ganger, G.Gibson, and S. Seshan, *“On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems,”* in *Proc. Supercomput.*, 2007, pp. 1–4.
[8] C. Guo, H.Wu,K.Tan,L. Shi,Y.Zhang, and S. Lu, *“DCell:Ascalable and fault tolerant network structure for data centers,*” in *Proc. ACM SIGCOMM*, 2008, pp. 75–86.
[9] M. Al-Fares, A. Loukissas, and A. Vahdat, *“A scalable, commodity data center network architecture,”* in *Proc. ACMSIGCOMM*, 2008, pp. 63–74.
[10] 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,”* in *Proc. ACM SIGCOMM*, 2009, pp. 63–74.
[11] L. Brakmo and L. Peterson, *“TCP Vegas: End to end congestion avoidance on a global internet,” IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.
[12] R. Braden, *“Requirements for internet hosts—Communication layers,”* RFC1122, Oct. 1989.
[13] V. Jacobson, R. Braden, and D. Borman, *“TCP extensions for high performance,”* RFC1323, May 1992.
[14] Y. Chen, R. Griffith, J. Liu, R. Katz, and A. Joseph, *“Understanding TCP incast throughput collapse in datacenter networks,”* in *Proc. WREN*, 2009, pp. 73–82.
[15] N. Spring,M. Chesire,M. Berryman, and V. Sahasranaman, *“Receiver based management of low bandwidth access links,”* in *Proc. IEEE INFOCOM*, 2000, vol. 1, pp. 245–254.
[16] P. Mehra, A. Zakhor, and C. Vleeschouwer, *“Receiver-driven bandwidth sharing for TCP,”* in *Proc. IEEE INFOCOM*, 2003, vol. 2, pp. 1145–1155.
[17] R. Prasad, M. Jain, and C. Dovrolis, *“Socket buffer auto-sizing forhigh-performance data transfers,” J. Grid Comput.*, vol. 1, no. 4, pp.361–376, 2003.
[18] H. Wu, Z. Feng, C. Guo, and Y. Zhang, *“ICTCP: Incast congestion control for TCP in data center networks,”* in *Proc. CoNEXT*, 2010, Articleno. 13.Dr. Zhang was an Associate Editor for the IEEE TRANSACTIONS ON MOBILE
COMPUTING and a General Co-Chair for ACM MobiCom 2009.