IOSR Journal of Computer Science (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727
PP 05-09
www.iosrjournals.org

Control of Congestion in Datacenter Network Using ICTCP Algorithm

Gaurav Buddhawar[1], Megha Jain[2]
[1,2]*Department of Computer science & EngineeringVNS college, Bhopal, India.*
buddha.gaurav@gmail.com,meghajain37@gmail.com

ABSTRACT: *In this paper, we study Transport Control Protocol (TCP) in-cast congestion control. In cast may severely degrade their performances by increasing response time also we study among TCP throughput, round trip time (RTT) and receive window. Our idea is to design an ICTCP (In cast congestion Control for TCP) scheme at the receiver side. In particular, our method adjusts TCP receive window proactively before packet drops occur. The implementation and techniques demonstrate that we achieve al-most zero timeout and high good put for TCP in cast. In this paper, we discuss a cross layer congestion control technique of TCP. In cast congestion happens in high-bandwidth and low-latency networks when multiple synchronized servers send data to the same receiver in parallel. For many important data-centre applications such as Map Reduce and Search, this many-to-one traffic pattern is common. Hence TCP in cast congestion may severely degrade their performances, e.g., by increasing response time. In this paper, we study TCP in cast in detail by focusing on the relationships between TCP throughputs, round-trip time (RTT). In particular, our method adjusts the TCP receive window proactively before packet loss occurs. The implementation and experiments in our test bed demonstrate that we achieve almost zero timeouts and high good put for TCP in cast.*
KEYWORDS: *Transport Control Protocol (TCP), Congestion, Data center networks, incast congestion, round-trip time (RTT).*

I. INTRODUCTION

The root cause of TCP incast collapse is that the highly burst 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. Our 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.

1.1 TCP Incast Congestion

In fig1.1, a typical data-center network structure is there. There are three layers of switches/routers: the ToR switch, the Aggregate switch, and the Aggregate router. A detailed case for a ToR connected to dozens of servers.

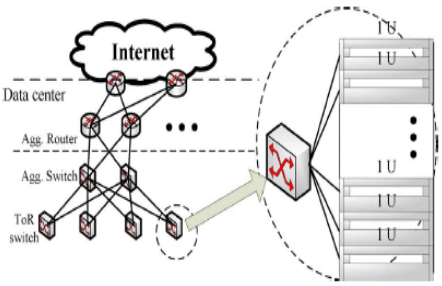

Fig.1.1: Data-center networks and a detailed illustration of a ToR switch connected to multiple rack-mounted servers.

Incast congestion happens when multiple sending servers under the same ToR switch send data to one receiver server simultaneously, as shown in Fig. 1.2.

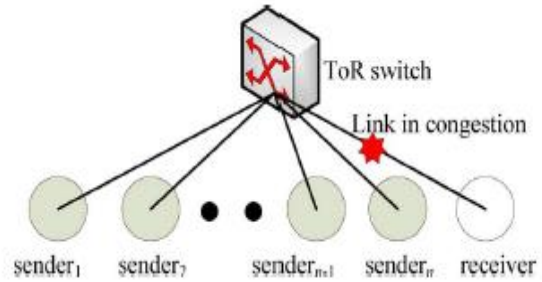

Fig.1.2: Scenario of incast congestion in data-center networks, where multiple ( ) TCP senders transmit data to the same receiver under the same ToR switch.

The amount of data transmitted by each connection is relatively small, e.g. 64 kB. The term goodputas it is effective throughput obtained and observed at the application layer. The multiple TCP connections are barrier-synchronized. First establish multiple TCP connections between all senders and the receiver, 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. Here observed similar goodput trends for three different traffic amounts per server, but with slightly different transition points.

II. LITERATURE REVIEW & RELATED WORK

As the Transport Control Protocol (TCP) is widely used on the Internet and generally works well. However, from recent studies A. Phanishayee, V. Vasudevan has shown that TCP does not work well for many-to-one traffic patterns on high-bandwidth, low-latency networks. Congestion occurs when manysynchronized servers under the same Gigabit Ethernet switch simultaneously send data to onereceiver in parallel. Only after all connections have finished the data transmission can the next round be issued.

V. Vasudevan focused on either reducing the wait time for packet loss recovery with faster retransmissions and M. Alizadeh focused on controlling switch buffer occupation to avoid overflow by using ECN and modified TCP on both the sender and receiver sides.

TCP incast has been identified and described by D. Nagle, D. Serenyi, and A. Matthews, the active Scale storage cluster delivering scalable high bandwidth storage in distributed storage clusters.

In distributed file systems, the files are deliberately stored in multiple servers.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 and the study of TCP characteristics on high-bandwidth, low-latency networks. Then root cause of packet loss in incast congestion, and after observing that the TCP receive window is the right controller to avoid congestion, seek to the TCP receive window adjustment algorithm.

S. Kandula, S. Sengupta work on the nature of data center traffic, as that in a data center, traffic under the same ToR is actually a significant pattern known as work-seeks-bandwidth, as locality has been considered during job assignment.

3. ICTCP Algorithm

ICTCP provides a receive-window-based congestion control algorithm for TCP at the end-system. The receive windows of all low-RTT TCP connections are jointly adjusted to control throughput on incast congestion. ICTCP algorithm closely follows the design points made. It is described how to set the receiver window of a TCP connection.

3.1 Control Trigger: Available Bandwidth

It is assumed there is one network interface on a receiver server, and define symbols corresponding to that interface. This algorithm can be applied to a scenario where the receiver has multiple interfaces, and the connections on each interface should perform this algorithm independently.

Assume the link capacity of the interface on the receiver server is G. Define the bandwidth of the total incoming traffic observed on that interface as BW_T, which includes all types of packets, i.e., broadcast, multicast, unicast of UDP or TCP, etc. Then, define the available bandwidth on that bandwidth BW_A interface as

$BW_A=\max (0, \alpha*C- BW_T)$

Where α € |0, 1 is a parameter to absorb potential oversubscribed bandwidth during window adjustment. A larger α (closer to 1) indicates the need to more conservatively constrain the receive window and higher requirements for the switch buffer to avoid overflow; a lower α indicates the need to more aggressively constrain the receive window, but throughput could be unnecessarily throttled. A fixed setting of BW_A in ICTCP, an available bandwidth as the quota for all incoming connections to increase the receive window for higher throughput. Each flow should estimate the potential throughput increase before its receiving window is increased. Only when there is enough quota (BW_A) can the receive window be increased, and the corresponding quota is consumed to prevent bandwidth oversubscription.

3.2Per-Connection Control Interval: 2*RTT

In ICTCP, each connection adjusts it's receive window only when an ACK is sending out on that connection. No additional pure TCP ACK packets are generated solely for receive window adjustment, so that no traffic is wasted. For a TCP connection, after an ACK is sent out, the data packet corresponding to that ACK arrives one RTT later. As a control system, the latency on the feedback loop is one RTT for each TCP connection, respectively. Meanwhile, to estimate the throughput of a TCP connection for a receive window adjustment; the shortest timescale is an RTT for that connection. Therefore, the control interval for a TCP connection is 2*RTT in ICTCP, and needed one RTT latency for the adjusted window to take effect and one additional RTT to measure the achieved throughput with the newly adjusted receive window.

3.3Fairness Controller for Multiple Connections

When the receiver detects that the available bandwidth has become smaller than the threshold, ICTCP starts to decrease the receiver window of the selected connections to prevent congestion.

Considering that multiple active TCP connections typically work on the same job at the same time in a data center, there is a method that can achieve fair sharing for all connections without sacrificing throughput. Note that ICTCP does not adjust the receive window for flows with an RTT larger than 2ms, so fairness is only considered among low-latency flows.

III. ANALYSIS OF PROBLEM

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 or controlling switch buffer occupation to avoid overflow by using ECN and modified TCP on both the sender and receiver sides. 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. Our 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).

In previous versions the senders are sending many packets to the main server, but if the senders increase then the load on receiver side increases. As the window size on receiving side is less before, now we are increasing the window size so that it can accommodate many retransmission acknowledgements.

IV. PROPOSED WORK AND OBJECTIVES

When multiple synchronized servers send data to the same receiver in parallel. The sender will send the packet to T_OR (Top of Rank) switch which send its packet to server. Due to the flexibility of the size of congestion window the rate of packet loss reduced but if the number of servers present is same and number of sender increases it leads to loss of packet because of the buffer overflow. To avoid this we are having one stack where the lost packets acknowledgement will be sent by the server to the sender to retransmit the lost packet. RTT of each packet will be given to each of the sender by the server to avoid the packet loss. In this paper we will be having one server and number of sender sending packet to the same receiver or server in this congestion may occur for this purpose we are designing a congestion window which can change its size according to the input and which can increase the throughput.

And if congestion occurs again because of the buffer overflow then we will check the node having heavy traffic and will change the path of the packet and will transfer to the neighboring node having less traffic by checking in the routing table. Thus congestion can be avoided to large extend, but in the previous method we have only the provision or way to change the size of congestion window but here we are changing the path of the packet and transferring the packet from heavy traffic node to less traffic node.

1. Retransmission of the lost packet.
2. The TCP receive window proactively active before packet loss occurs.

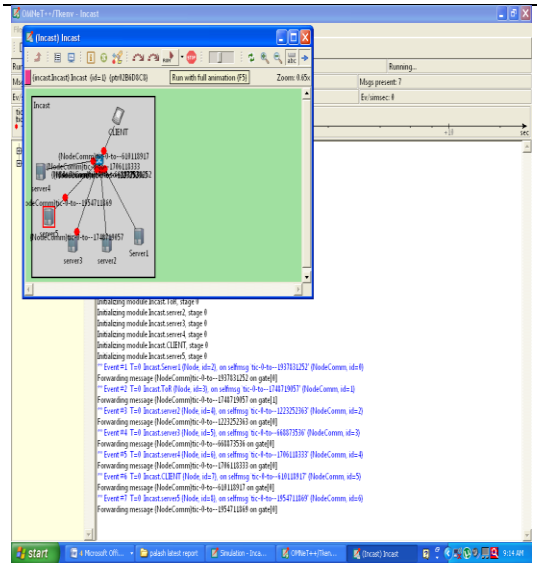

Fig 1: Proposed work

ACKNOWLEDGEMENTS

The making of the seminar needed co-operation and guidance of a number of people. I therefore consider it my prime duty to thank all those who had helped me through their venture. It is my immense pleasure to express my gratitude to **Prof. Megha Jain as** guide who provided me constructive and positive feedback during the preparation of this seminar. I express my sincere thank to the head of department **Mr.Vivek Sharma** and all other staff members of CSE department for their kind co-operation. I would like to thank **Dr.R.K.Nigam** Principal of our institution for providing necessary facility during the period of working on this report. I am thankful to my friends and library staff members whose encouragement and suggestion helped me to complete my seminar. I am also thankful to my parents whose best wishes are always with me.

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. ACMSIGCOMM*, 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. 14.
[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. ACMSIGCOMM*, 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 highperformance," 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.