ISSN:0975-9646

Congestion Control for TCP in Data-Center Networks

Samba Siva Reddy Maripalli[#] , Abirami G[*]

[#]*M.Tech, Computer Science and Engineering, SRM University, Chennai, India*

[*]*Asst professor .Department of Computer Science and Engineering SRM University, India*

Abstract— **The several corresponding servers may send the data concurrently in a circle manner to a specific receiver. For example, data center applications such as Map-Reduce and Search applications are using many-to-one TCP traffic pattern i.e. parallel data delivery. While TCP connection is slow down the concurrent data deliver. The data delivery delay ensured the packet loss and time out in the TCP connection namely called TCP incast congestion. This congestion may reduce the performance of TCP connection such as increase the waiting time and increase bandwidth utilization. The proposed system is implemented for avoid the ICTCP congestion with assist of receive(r)-window-based congestion control algorithm. Initially, the proposed algorithm shared the available bandwidth namely called quota for all TCP connections. Next, assign the control interval to every TCP connections using Round Trip Time. The control interval will be 2*RTT for every TCP connections. Finally, adjust (increase/decrease) receive-widow size based on two threshold values, available quota, Round Trip Time, current TCP connection throughput in receiver side and expected throughput of the TCP connection. If ratio of difference between measured throughput and expected throughput is less than one expected throughput increase the receive-window-size. Otherwise, the ratio is large decrease the receive window. Further, improve our proposed system that converted many-to-many traffic congestion into many-to-many traffic congestion**

Keywords— **Data-Center ,incast congestion, Round Trip Time ,TCP.**

I. INTRODUCTION

Internet datacenters support a myriad of services and applications. Companies like Google, Microsoft, Yahoo, and Amazon use datacenters for web search, storage, e-commerce, and large-scale general computations. Business cost efficiencies mean that datacenters use existing technology. In particular, the vast majority of datacenters use TCP for communication between nodes. TCP is a mature technology that has survived the test of time and meets the communication needs of most applications. However, the unique workloads, scale, and environment of the Internet datacenter violate the WAN assumptions on which TCP was originally designed. For example, in contemporary operating systems such as Linux, the default RTO timer value is set to 200ms, a reasonable value for WAN, but 2-3 orders of magnitude greater than the average roundtrip time in the datacenter.

As a result, we discover new shortcomings in technologies like TCP in the high-bandwidth, low-latency datacenter environment. One communication pattern, termed "Incast" by other researchers, elicits a pathological response from popular implementations of TCP. In the Incast communication pattern, a receiver issues data requests to multiple senders. The senders, upon receiving the request, concurrently transmit a large amount of data to the receiver. The data from all senders traverses a bottleneck link in a many-to-one fashion. As the number of concurrent senders increases, the perceived application-level throughput at the receiver collapses. The receiver application sees goodput that is orders of magnitude lower than the link capacity. The incast pattern potentially arises in many typical datacenter applications. For example, in cluster storage, when storage nodes respond to requests for data, in websearch, when many workers respond near simultaneously to search queries, and in batch processing jobs like Map Reduce , in which intermediate key-value pairs from many Mappers are transferred to appropriate Reducers during the "shuffle" stage.

II. CONGESTION IN DATA-CENTER NETWORKS

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. 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.

Consider many-to-one or many-to-many traffic pattern The several synchronized servers will send the data file to one/multiple receivers concurrently in a circle manner. All TCP connections are shared the same bandwidth group. The connections final performance is determined by the slowest TCP connection, which may direct to connection timeout or packet loss.

The several corresponding servers may send the data concurrently in a circle manner to a specific receiver. For example, data center applications such as Map-Reduce and Search applications are using many-to-one TCP traffic pattern i.e. parallel data delivery. While TCP connection is slow down the concurrent data deliver. The data delivery

delay ensured the packet loss and time out in the TCP connection namely called TCP incast congestion. This congestion may reduce the performance of TCP connection such as increase the waiting time and increase bandwidth utilization. The proposed system is implemented for avoid the ICTCP congestion with assist of receive(r)-window-based congestion control algorithm. Initially, the proposed algorithm shared the available bandwidth namely called quota for all TCP connections. Next, assign the control interval to every TCP connections using Round Trip Time. The control interval will be 2*RTT for every TCP connections. Finally, adjust (increase/decrease) receive-widow size based on two threshold values, available quota, RoundTripTime, current TCP connection throughput in receiver side and expected throughput of the TCP connection. If ratio of difference between measured throughput and expected throughput is less than one expected throughput increase the receive-window-size. Otherwise, the ratio is large decrease the receive window. To improve our proposed system, the proposed receive(r)-window-based congestion control algorithm implemented in many-to-many traffic pattern (For example international share trading,e-applications such as multimedia sharing).The proposed algorithm implemented in every receiver side. Every receiver will group into same bandwidth i.e. sharing same bandwidth.

III. ESTABLISHMENT OF CONNECTIONS BETWEEN DATA-CENTRES AND USERS

A. Network Creation

First install number of sender servers at data center side. Then install one switch device for switching the data packets from multiple senders to receivers.

B. Many–to-One

In this module, multiple servers act as a sender (data center) and single system act as a receiver. The receiver sends the request to corresponding data center. Once multiple servers receive the request, create TCP connection between them. Then initiate the sending window size based on the capacity of server's network interface. Then request file divided into number of packets and then ready to send the data packets corresponding receiver.

C. Control Interval

Receiver and multiple servers share the available bandwidth based on the network interface capacity. Data packets are sending to receiver based on the time. The time has split into number of slots. Every slot categorized into two sub slots such as first and second. Once connection has TCP established, receiver initialize the receiver window size as 2*maximum segmentation size. Then, measure control interval to adjust the receive window size. The control interval is measured by Round Trip Time. Then measure the sub slot length using control interval of all active TCP connections.

D. Receive Window Adjustment

The receiver side receive-window of every TCP connection is adjusted based on TCP connection throughput. The TCP connection throughput is calculated using current throughput of TCP connection and expected throughput. The receive window is increased when the ratio of the difference between measured and expected throughput over the expected one is small, otherwise decrease the receive window when the ratio is large.

E. Many-to-Many

In this module, multiple server acts as a senders (data-center)and multiple request system act as receivers. Every receiver sends the request to every server center to obtain same data. All receivers and senders share the same bandwidth and same switch. Every receiver activate the above discussed receive window adjustment scheme for effective congestion avoidance.

IV. 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. Here is that how to set the receive window of a TCP connection.

*A. **Control Trigger***

Available BandwidthWithout loss of generality, we assume there is one network interface on a receiver server, and define symbols corresponding to that interface.

Our algorithm can be applied to a scenario where the receiver has multiple interfaces, and the connections on each interface should perform our algorithm independently.Assume the link capacity of the interface on the receiver server is C . 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, we define the available bandwidth BW_A on that interface as

$$BW_A = \max(0,\alpha \times C - BW_T) \quad \text{.....(1)}$$

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 necessarily throttled. In all of our implementations and experiments, we have a fixed setting of $\alpha = 0.9$.

In ICTCP, we use available bandwidth BW_A 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 receive 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.

To estimate the available bandwidth on the interface and provide a quota for a later receive window increase, we divide the time into slots. Each slot consists of two subslots of the same length T . *For each network interface, we measure all the traffic received in the first subslot and use it*

to calculate the available bandwidth as a quota for window increase on the second subslot. The receive window of any TCP connection is never increased at the first subslot, but may be decreased when congestion is detected or the receive window is identified as being over satisfied,In Fig. 1, the arrowed line marked by "Global" denotes the slot allocation for available bandwidth estimation on a network interface. The first subslot is marked in gray. During the first subslot, none of the connections' receive windows can be increased (but they can be decreased if needed). The second subslot is marked in white in Fig. 1. In the second subslot, the receive window of any TCP connection can be increased, but the total estimated increased throughput of all connections in the second subslot must be less than the available bandwidth observed in the first subslot. Note that a decrease of any receive window does not increase the quota, as the quota will only be reset by incoming traffic in the next first subslot. We discuss how to choose T and its relationship to the per-flow control interval next.

B.Per-Connection Control Interval: 2 RTT*

In ICTCP, each connection adjusts its 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, as we need 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. Note that the window adjustment interval is performed per connection. We use connections i and j to represent two arbitrary TCP connections in Fig. 1 to show that one connection's receive window adjustment is independent from the other.

The relationship of subslot length T and any flow's control interval is as follows: Since the major purpose of available bandwidth estimation on the first subslot is to provide a quota for window adjustment on the second subslot. length T should be determined by the control intervals of all active connections.

The changed throughput of any connection is with its RTT, and thus T should be with the RTT to represent the changes in available bandwidth .We use weighted average RTT of all TCP connections as T , i.e.,$T=\sum_i w_i RTT_i$. The weight w_i is the normalized traffic volume of connection i that has updated in RTTi the last time period T. In Fig. 1, we illustrate the relationship of two a rbitrary TCP connections i/j with $RTT_{i/j}$ and the system estimation subinterval T . Each connection adjusts its receive window based on the observed RTT. The time it takes for a connection to increase its receive window is marked with an up arrow in Fig. 1. For TCP connection i, if "now" is in the second global subslot and the elapsed time is larger than2*RTT_i since its last receive window adjustment, it may increase its window based on the newly observed TCP throughput and current available bandwidth.

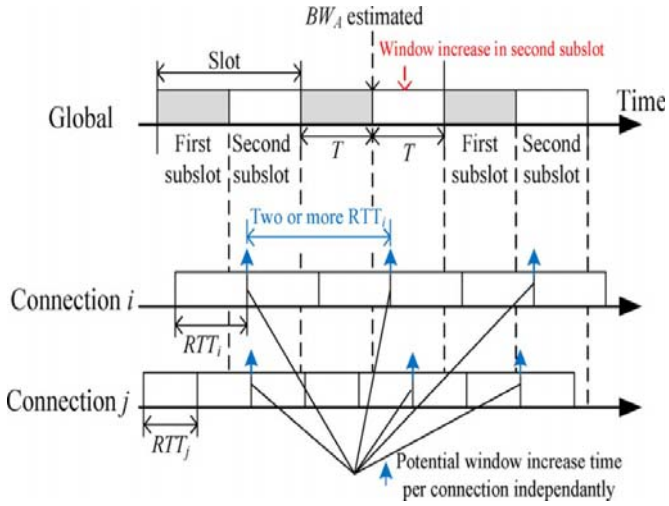

Fig. 1 *Slotted time on global (all connections on that interface) and two arbitrary TCP connections i/j are independent.*

C.Window Adjustment on Single Connection

For any ICTCP connection, the receive window is adjusted based on its incoming *measured throughput* (denoted as b^m) and its *expected throughput* (denoted as b^e). The measured throughput represents the achieved throughput on a TCP connection and also implies the current requirements of the application over that TCP connection. The expected throughput represents our expectation for the throughput on that TCP connection if the throughput is only constrained by the receive window.

Our idea for receive window adjustment is to increase the receive window when the ratio of the difference between measured and expected throughput over the expected one is small, and to decrease the receive window when the ratio is large. A similar concept has previously been introduced in TCP Vegas but it uses the throughput difference instead of the ratio of throughput difference, and it is designed for the congestion window on the sender side to pursue available bandwidth. ICTCP window adjustment sets the receive window of a TCP connection to a value that represents its current application's requirements. An oversized receive window is a hidden problem as the throughput of that connection may reach the expected one at any time, and the traffic surge may overflow the switch buffer, a situation that is hard to predict and avoid.The measured throughput b_i^m of connection i is obtained and updated for every RTT_i, where RTT_i is the RTT of connection i . For every RTT_i on connection i , we obtain a sample of current throughput, denoted as b_i^s, calculated as the total number of received bytes divided by the time interval RTT_i .We smooth the measured throughput using the exponential filter as

$$b^m_{i,new} = \max\left(b^s_i, \beta * b^m_{i,old} + (1-\beta) * b^s_i\right) \qquad (2)$$

β is the exponential factor, and the default value of β is set to 0.75. Note that the max procedure here is to update

quickly if the receive window is increased, especially when the receive window is doubled. The expected throughput of connection *i* is obtained as

$$b_i^e = max\ (b_i^m, rwnd_i/RTT_i) \quad (3)$$

where $rwnd_i$ is the receive window of connection *i*. We have the max procedure to ensure $b_i^m \leq b_i^e$.We define the ratio of throughput difference d_i^b as the ratio of the difference of the measured and expected throughput over the expected one for connection *i*

$$d_i^b = \frac{b_i^e - b_i^m}{b_i^e}. \quad (4)$$

By definition, we have$b_i^m \leq b_i^e$, thus $d_i^b \epsilon [0,1]$. We have two thresholds $\gamma 1$and$\gamma 2$ ($\gamma 2 > \gamma 1$) to differentiate three cases for receive window adjustment:

1) $d_i^b \leq \gamma 1$or $d_i^b \leq$ $MSS_i/rwnd_i^2$: Increases the receive window if it is in the global second subslot and there is enough quota of available bandwidth on the network interface; decreases the quota correspondingly if the receive window is increased.

2) $d_i^b > \gamma 2$:Decrease the receive window by 1 MSS^3 if this condition holds for three continuous RTT. The minimal receive window is 2*MSS.

3) Otherwise, keep the current receive window.The available bandwidth calculated at the end of the first subslot is used for the quota of the second subslot right after the first one. The potential throughput increase of connection is estimated as the increase in the receive window divided by RTT_i .The quota is consumed by First Come First Service (FIFS), determined by the order of ACKs sent on the second subslot.In all of our experiments, we had $\gamma 1=0.1$ and $\gamma 2=0.5$. Similar to TCP's congestion window increase at the sender, the increase of the receive window on any ICTCP connection consists of two phases: *slow start* and *congestion avoidance*. If there is enough quota in the slow start phase, the receive window is doubled, while it is enlarged by atmost oneMSS in the congestion avoidance phase. A newly established or prolonged idle connection is initiated in the slow start phase. Whenever the second and third conditions are met, or the first condition is met but there is not enough quota on the receiver side, the connection goes into the congestion avoidance phase.

D ***Fairness 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, we have sought 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 2 ms, so fairness is only considered among low-latency flows.

In our experiment, we decrease the receive window for fairness when BW_A <0.2C. This condition is designed for high band width networks, where link capacity is underutilized most of the time. If there is still enough available bandwidth, we argue that the requirement of better fairness is not strong considering the potential impact on achieved throughput when decreasing the receive window. The purpose of the 0.2C gap is to leave enough room for other flows to increase their receive window, and should be larger than the increased throughput when the receive window of a flow is increased by 1 MSS.We adjust the receive window to achieve fairness for incoming TCP connections with low latency as follows: 1) For a window decrease, we cut the receive window by 1 MSS, for some selected TCP connections. We select those connections that have a receive window larger than the average window value of all connections. 2) For a window increase, this is automatically achieved by our window adjustment described in this Section , as the receive window is only increased by 1 MSS during congestion avoidance. In principle, the receive window decrease only happens when the available bandwidth on that interface is small. Furthermore, the connection with the larger receive window is decreased slightly to achieve fairness. If all connections happen to have the same receive window, then none of them decrease the receive window.

Readers may raise some questions concerning our interpretation of fairness to address a congestion control scheme.TCP uses additive increase multiplicative decrease (AIMD) to achieve both stability and fairness among flows sharing the same bottleneck. However, MD happens only when packet loss is observed for a TCP connection. In DCTCP, packet loss during incast congestion due to buffer overflow is largely eliminated.

Consider a scenario in which some flows have a large receive window (because they started earlier) while others have a much smaller receive window (because they started later), and the link capacity is almost reached. In this case, none of the connections can increase their receive windows as there is not enough available bandwidth. This situation may persist as long as there is no packet loss so that unfairness between flows becomes an issue. Therefore, we propose slightly reducing the receive window for flows that have a larger receive window, and the throughput of all TCP connections should smoothly converge.

CONCLUSIONS

This implemented the ICTCP to improve TCP performance for TCP incast in many-to-many data-center networks. In contrast to previous approaches that used a fine-tuned timer for faster retransmission, we 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. In future, the proposed mechanism can be implemented in real time data centers and then evaluate their performance.

ACKNOWLEDGMENT

The authors are grateful to express sincere thanks and gratitude to our Professor and HOD, Dr.E.Poovammal, Department of Computer Science and Engineering, SRM University for her encouragement and the facilities that were offered to me for carrying out this project. And we take this opportunity to thank our Director, Dr.C.Muthamizchelvan, Ph.D, Faculty of Engineering and Technology, SRM University for providing us with excellent infrastructure that is required for the development of our project.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. OSDI, 2004, p. 10.

[2] 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.

[3] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage," in Proc. SC, 2004, p. 53

[4] 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

[5] 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.

[6] R. Prasad, M. Jain, and C. Dovrolis, "Socket buffer auto-sizing for high-performance data transfers," J. Grid Comput., vol. 1, no. 4, pp. 361–376, 2003.

[7] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in Proc. CoNEXT, 2010, Article no. 13

[8] 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.

[9] R. Braden, "Requirements for internet hosts—Communication layers," RFC1122, Oct. 1989.

[10] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC1323, May 1992.

[11] 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.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in Proc. ACMSIGCOMM, 2008, pp.

[13] 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.

[14] http://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm.