International Journal of Pure and Applied Mathematics
Volume 116 No. 8 2017, 235-240
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

CONGESTION CONTROL IN DATACENTER NETWORK USING ICTCP ALGORITHM

[1]Jeya Priya D, [2]Kavitha.S
[1,2]Assistant Professor, Dept of CSE, BIST, BIHER
BHARATH UNIVERSITY, Chennai-73.
[1]jeyapriya.cse@bharathuniv.ac.in

Abstract: Transport Control Protocol is widely used on the Internet and generally works well. TCP does not work well for many-to-one traffic patterns on high-bandwidth, low-latency networks. Congestion occurs when many synchronized servers simultaneously send data to one receiver in parallel. The performance is determined by the slowest TCP connection, which may suffer from timeout due to packet loss. In this paper, we study Transport Control Protocol (TCP) in-cast congestion control. Incast 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. Incast congestion typically happens in a data center network when multiple servers, all connected to the same switch, are sending data simultaneously to a common destination, such that the switch buffer overflows, resulting in packet loss. For many important data-center 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).

Keywords: Data center networks, incast congestion, Transport Control Protocol (TCP), round-trip time (RTT).

1. Introduction

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
The root cause of TCP incast collapse is that the highly burst traffic of multiple TCP connections overflows the 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 non incast 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.

Data centers have become very popular for storing large volumes of data. In particular, companies like Amazon, Google, and Yahoo! routinely use data centers for storage, Web search, and large-scale computations. The main characteristics of a data center network are high-speed links, low propagation delays, and limited-size switch buffers.TCP is designed to be end-to-end: there is no global Coordinator and each host relies on implicit signals to infer the condition of the network and then adjusts its own sending rate. Reliable

transmission is achieved via the use of a retransmit timer: for the segments sent each time, the sender expects an ACK from the receiver before the timer expires; without receiving the ACK in time[6,7], some segment is considered to be lost, presumably due to network congestion and will be retransmitted at some appropriate instant later.

The Transmission Control Protocol (TCP) is used as the transport-layer protocol for reliable data transfer in data center networks, just as it is on the Internet. However, the network configurations in data centers are very different from the general Internet conditions, for which TCP was originally designed. In particular, the typical propagation round-trip delay in a data center network is 0.1 ms, while the default retransmission timeout (RTO) on the Internet is 200 ms. On each user request for data, many servers transmit data over a data center network concurrently, which, in combination with the small switch buffers, leads to packet losses. For short-lived flows, packet losses cause TCP retransmission timeouts, which in turn degrade the good put of data center applications. Such a decrease of good put is called TCP-incast throughput collapse.

2. Background and Past Works

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 uses high-resolution timers to enable microsecond-granularity TCP time outs, TCP does not work well for many-to-one traffic patterns on high-bandwidth, low-latency 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. 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. L. Peterson Used fluid-flow model to overcome the delay .the main draw backs are Re-routing and persistent congestion occurs among the network. 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. P. Mehra used to control bandwidth sharing among multiple TCP flows on the receiver side. Only focused on the ratio of achieved bandwidth of those multiple independent TCP flows.

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 networkshave 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[10,9]. 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. Proposed Work And Methodologies

When multiple synchronized servers send data to the same receiver in parallel. 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. 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[14,13]. 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. 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. The main objective is to analyze the congestion problems and develop an available bandwidth estimation technique to increase all of incoming connections which is cost effective, reduces Round trip time and adaptive to dynamic environment for which we used Network driver interface specification filter functionalities like collecting network statistics, monitoring and filtering the unauthorized ones. This formulation achieves high bandwidth and incast congestion problems can be eliminated.

To perform congestion control on the receiver side, we use the available bandwidth to increase of all incoming connections.

Our per-flow congestion control is performed independently of the slotted time of the round-trip time of each connection, which is also the control latency in its feedback loop. Well controlling the receive window is challenging: The receive window should be small enough to avoid incast congestion, when we use per-flow and also large enough for good performance. We also design a receive window Buffer is based on loss of the packets. We have developed and implemented ICTCP as a Windows Network Driver Interface Specification filter driver

4. 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.

Control Trigger: Available Bandwidth

Using available bandwidth to increase all of incoming connections on the receiver server. Developing ICTCP as an NDIS driver on Windows OS. Our NDIS driver intercepts TCP packets and modifies the receive window size if needed. 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 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, define the available bandwidth to increase all of incoming connections 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[15,16,17], 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.

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 sub slots of the same length. For each network interface, we measure all the traffic received in the first sub slot and use it to calculate the available bandwidth as a quota for window increase on the second sub slot. The receive window of any TCP connection is never increased at the first sub slot, but may be decreased when congestion is detected or the receive window is identified as being over satisfied

Flow stack information:

A flow table maintains the key data structure in the Receiver server. A flow is identified by a 5-tuple: source/destination IP address, source/destination port, and protocol. The flow table stores flow information for all the active flows. The packet header is parsed and the corresponding information is updated in the flow table[18,19]. Network driver interface specification filter functionalities are performed like collecting network statistics, monitoring activities and filtering the unauthorized ones.

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[20], 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[21,22,23].

Window Adjustment on a single connection:

For any ICTCP connection, the receive window is adjusted based on its incoming measured throughput and its expected throughput. The measured throughput represents the achieved throughput on a TCP connection, also implies the current requirement of the application over that TCP connection.

$$b^m_{i,new} = \max (b_i^s, \beta * b^m_{i,old} + (1- \beta)*b_i^s)$$

b^m represents Incoming measured throughput b_i^s represents Sample of current throughput (on connection

i) .The expected throughput represents our expectation of the throughput on that TCP connection if the throughput is only constrained by receive window.

$b_i^e = \max(b^m_{i,} \cdot rwnd_i / RTT_i)$

b_i^e represents Expected throughput of i ,$rwnd_i$ represents Receive window of i. The ratio of throughput difference of connection i is represented as $d_i^b = (b_i^e - b_i^m) / b_i^e$

Our idea on receive window adjustment is to increase window when the difference ratio of measured and expected throughput is small, while decrease window when the difference ratio is large.

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, 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.

We decrease the receive window for fairness when $BW_A <$ 0.2 C. This condition is designed for high bandwidth networks, where link capacity is underutilized most of the time.

5. Implementation

Develop ICTCP as a NDIS driver on Windows OS. Naturally supports the case for virtual machine. The incoming throughput in very short time scale can be easily obtained. Does not touch TCP/UDP implementation in Windows kernel.

Figure 1. Incast congestion control in a network

The operations of an ICTCP driver are as follows:

- Redirect the packet to header parser module
- Packet header is parsed and the information on flow table is updated
- Algorithm module is responsible for receive window calculation
- If a TCP ACK packet is sent out, the header modifier change the receive window field in TCP header if need.

Our ICTCP driver does not introduce extra CPU overhead for packet checksum when the receive window is modified for an outgoing ACK packet, as the checksum calculation is loaded to NIC on the hardware, which is the normal setup in a data center.

There is also no packet reordering in our driver.

Support for Virtual Machines:

Virtualization technology is a key enabler of cloud computing. The hypervisor allows multiple VMs to run on a single physical host by multiplexing the underlying physical resources, such as CPU, memory and I/O. The total capacity of virtual NICs is typically configured high than physical as most virtual machine won't be busy at the same time. The observed virtual link capacity and available bandwidth does not represent the real value. Figure 1 represents the incast problem .There are two solution, Change the setting to make the total capacity of virtual NICs equal to physical NIC,and Deploy a ICTCP driver on virtual machine host server.

Obtain fine-grained RTT at receiver:

Define the reverse RTT as the RTT after a exponential filter at the TCP receiver side. The reverse RTT can be obtained in data traffic on both sides. The data traffic on reverse direction may not be enough for keep obtaining live reverse RTT. Use TCP timestamp, for implement; modify the timestamp counter into 100ns granularity.

6. Conclusion

This paper presented a practical, effective, and safe solution to eliminate TCP incast collapse in datacenter environments. Our approach utilizes the link bandwidth as fully as possible but without significant packet losses by limiting the round trip time value. Based on the concept of bandwidth delay product, our approach conservatively estimates the reasonable number of concurrent senders. In this work we used Network driver interface specification filter functionalities like collecting network statistics; monitoring, filtering unauthorized ones are done. We can avoid retransmission, safely send the data to the receiver and also avoid congestion and

traffic. our system is able to match user preferences while achieving full utilization of the receiver's access in many different scenarios.

References

[1]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.

[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]Udayakumar R., Kaliyamurthie K.P., Khanaa, Thooyamani K.P., Data mining a boon: Predictive system for university topper women in academia, World Applied Sciences Journal, v-29, i-14, pp-86-90, 2014.

[9]Kaliyamurthie K.P., Parameswari D., Udayakumar R., QOS aware privacy preserving location monitoring in wireless sensor network, Indian Journal of Science and Technology, v-6, i-SUPPL5, pp-4648-4652, 2013.

[10]Brintha Rajakumari S., Nalini C., An efficient cost model for data storage with horizontal layout in the cloud, Indian Journal of Science and Technology, v-7, i-, pp-45-46, 2014.

[11]Brintha Rajakumari S., Nalini C., An efficient data mining dataset preparation using aggregation in relational database, Indian Journal of Science and Technology, v-7, i-, pp-44-46, 2014.

[12]Khanna V., Mohanta K., Saravanan T., Recovery of link quality degradation in wireless mesh networks, Indian Journal of Science and Technology, v-6, i-SUPPL.6, pp-4837-4843, 2013.

[13]Khanaa V., Thooyamani K.P., Udayakumar R., A secure and efficient authentication system for distributed wireless sensor network, World Applied Sciences Journal, v-29, i-14, pp-304-308, 2014.

[14]Udayakumar R., Khanaa V., Saravanan T., Saritha G., Retinal image analysis using curvelet transform and multistructure elements morphology by reconstruction, Middle - East Journal of Scientific Research, v-16, i-12, pp-1781-1785, 2013.

[15]Khanaa V., Mohanta K., Saravanan. T., Performance analysis of FTTH using GEPON in direct and external modulation, Indian Journal of Science and Technology, v-6, i-SUPPL.6, pp-4848-4852, 2013.

[16]Kaliyamurthie K.P., Udayakumar R., Parameswari D., Mugunthan S.N., Highly secured online voting system over network, Indian Journal of Science and Technology, v-6, i-SUPPL.6, pp-4831-4836, 2013.

[17]Thooyamani K.P., Khanaa V., Udayakumar R., Efficiently measuring denial of service attacks using appropriate metrics, Middle - East Journal of Scientific Research, v-20, i-12, pp-2464-2470, 2014.

[18]R.Kalaiprasath, R.Elankavi, Dr.R.Udayakumar, Cloud Information Accountability (Cia) Framework Ensuring Accountability Of Data In Cloud And Security In End To End Process In Cloud Terminology, International Journal Of Civil Engineering And Technology (Ijciet)Volume 8, Issue 4, Pp. 376–385, April 2017.

[19]R.Elankavi, R.Kalaiprasath, Dr.R.Udayakumar, A fast clustering algorithm for high-dimensional data, International Journal Of Civil Engineering And Technology (Ijciet), Volume 8, Issue 5, Pp. 1220–1227, May 2017.

[20]R. Kalaiprasath, R. Elankavi and Dr. R. Udayakumar. Cloud. Security and Compliance - A Semantic Approach in End to End Security, International Journal Of Mechanical Engineering And Technology (Ijmet), Volume 8, Issue 5, pp-987-994, May 2017.

[21]Thooyamani K.P., Khanaa V., Udayakumar R., Virtual instrumentation based process of agriculture by automation, Middle - East Journal of Scientific Research, v-20, i-12, pp-2604-2612, 2014.

[22]Udayakumar R., Thooyamani K.P., Khanaa, Random projection based data perturbation using geometric transformation, World Applied Sciences Journal, v-29, i-14, pp-19-24, 2014.

[23]Udayakumar R., Thooyamani K.P., Khanaa, Deploying site-to-site VPN connectivity: MPLS Vs IPSec, World Applied Sciences Journal, v-29, i-14, pp-6-10, 2014.