International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
International Conference on Humming Bird (01st March 2014)

RESEARCH ARTICLE OPEN ACCESS

Congestion Control for Avoiding Packet Loss in Data Centric Network

S.M.BaslinNixy,
PG Scholar Department of Information Technology
Francis Xavier Engineering College Vannarpettai,
Tirunelveli. nixylazar@gmail.com,9442022721

S.Agnes Joshy
Assistant Professor Department of Information Technology Francis Xavier Engineering College Vannarpettai, Tirunelveli. sagnesjoshy@gmail.com

Abstract
In TCP multiple synchronized servers send data to the same receiver in parallel, so the incast congestion will occur. Basically incast congestion happens in high-bandwidth and low latency networks. TCP incast congestion severely degrade the performances of the system.TCP incast congestion is studied by focusing the relationship between TCP throughput, round-trip time and receive window. In receiver side window, the packet received in parallel at same time. So packet loss occurs due to congestion. Before the packet loss occurs, the bandwidth calculation and adjusting of window size in receiver side can be done. Based on the packet, the window size is adjusted in the receiver side. By transmitting the packet to the receiver in different paths the performance time will be reduced. From this Zero timeouts and high goodput for TCP incast can be achieved.

Index Terms—Data-center networks, incast congestion, bandwidth calculation,TCP

I. INTRODUCTION

Transport Control Protocol (TCP) is widely used on the Internet and generally works well. However, recent studies 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.

In TCP the multiple servers send data to the receiver, so congestion may occur in network. If congestion occurred, the packet loss can be happened. For that using the TCP protocol used to calculate bandwidth for the each packet. If the incoming packet was larger, then the window size will be extend based on the size of that packet. So congestion will be avoided before the packet loss occurs. 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. 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. 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.

II. BACKGROUND AND MOTIVATION

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.

A. TCP Incast Congestion:

Incast congestion happens when multiple sending servers under the same ToR switch send data to one receiver server simultaneously. The amount of data transmitted by each connection is relatively small. As the TCP receive window has the ability to control TCP throughput and thus prevent TCP incast collapse, we consider how to dynamically adjust it to the proper value. We start with the window-based congestion control used in TCP. As we know, TCP

uses slow start and congestion avoidance to adjust the congestion window on the sender side. 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 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. This job is distributed to multiple servers, and each server is responsible for some documents on its local disk. Only after all servers return their counts to the receiving server can the final result be generated.

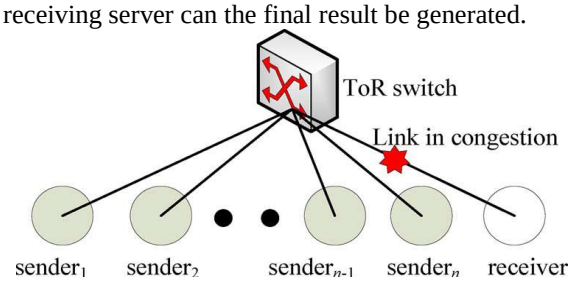

Fig. 1. Scenario of incast congestion in data-center networks

B. *TCP Goodput, Receive Window, and RTT:*

The TCP receive window is introduced for TCP flow control, i.e., preventing a faster sender from overflowing a slow receiver's buffer. The receive window size determines the maximum number of bytes that the sender can transmit without receiving the receiver's ACK. A previous study mentioned that a small static TCP receive buffer may throttle TCP throughput and thus prevent TCP incast congestion collapse. We observe that an optimal receive window exists to achieve high goodput for a given number of senders. As an application layer solution, a capped and well-tuned receive window with a socket buffer may work for a specific application in a static network. The background connection can be generated by other applications, or even from other VMs in the same host server. Thus, a static buffer cannot work for a changing number of connections and cannot handle the dynamics of the applications' requirements.

III. 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.

A. Available Bandwidth:

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 BW1.

$$Bw1 = Max(0, \alpha * c - BW)$$

Where $\alpha \epsilon [0,1]$ is a parameter to absorb potential oversubscribed bandwidth during window adjustment. In ICTCP, we use available bandwidth BW1 as the quota for all incoming connections to increase the receive window for higher throughput.

IV. MODULE IMPLEMENTATION

In this section, we describe the implementation of ICTCP, which is developed in an NDIS driver on the Windows OS.

A. Authentication Module:

The Major motto of the authenticate module is that is used for identifying the user and recognizing the user for network transmission , since authentication module first gently gets the clients information and then it assigns the proper authentication name with password using which the clients can enter into the network transmission.

B. Client Server Connection:

After allotting the proper authorization and authentication of the client, then the client enter in the client server connection module. In this module the client first generates the client request with client header and client request time and client request with server path i.e. server name and its IP header.

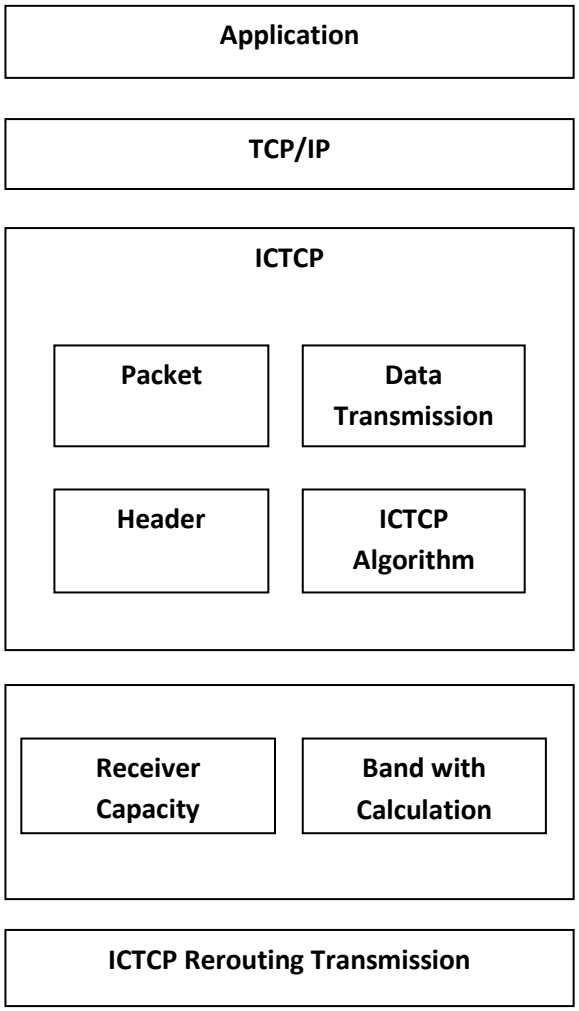

Fig. 2: Modules in ICTCP

C. Handling Requests:

Since all the client are dynamically transmitting the request to the server, the server using the TCP /IP connection it only accepts the one client at a time so that all other request are being made to be in waiting state. Thus the message will be transmitted to all other user. To avoid response time out process, the handling request module generate the unique id and assigns to the request, the response delay time will also been altered by this module.

D. ICTCP Module:

The ICTCP module is the Major module in which all the handling request will be handled properly without moving it to the response out. The ICTCP modules here calculate all the request bandwidth based on the bandwidth it re order the request. After proper reordering, it finally calculates the whole capacity of the request send by the clients and based on the bandwidth obtained. The ICTP changes the Receiver window size of the server. So that it can handle too many requests at a time. After changing receiver window size the reorder request will be send to the server based on the unique id, thus all the Request send by the client will be handled within the complete timeout of the client requests.

V. EXPERIMENTAL RESULTS

The implementation is to avoid the congestion during the packet transmission. The CPU, memory, and hard disk were never a bottleneck in any of our experiments. Construct the incast scenario where multiple sending servers generate TCP traffic to a receiving server under the same switch. The servers in our test bed have their own background TCP connections for various services, but the background traffic amount is very small compared to our generated traffic. The test bed is in an enterprise network with normal background broadcast traffic.

Note that all the TCP stacks were the same in our experiments, and ICTCP was implemented on a filter driver at the receiver side. The goodput shown is the average value of 100 experimental rounds. We observe the incast 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.

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.

VI. CONCLUSION

In this paper a lightweight and high-performance Window NDIS filter driver the implementation of ICTCP has been developed. ICTCP avoid congestion in the network during packet transmission. And the window size also adjusts based on the size of the packet. Based on the throughput the receiver side window size will be expanding. By transmitting the packet to the receiver in different paths the performance time will be reduced. From this the experimental results demonstrate that ICTCP is effective in avoiding congestion by achieving almost zero timeouts for TCP incast, and it provides high performance.

REFERENCES

[1] Haitao Wu, Zhenqian Feng, Chuanxiong Gu, Yongguang Zhang, "ICTCP: Incast Congestion Control for TCP in Data Center Networks," IEEE/ACM TRANSACTIONS

ON NETWORKING, VOL. 21, NO. 2, APRIL 2013.

[2] Lawrence S. brakmo, IEEE and Larry L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet IEEE journal on Selected Area in Communications, Vol. 13, No.8, October 1995

[3] Neil T. Spring, Maureen Chesire, Mark Berryman, Vivek Sahasranaman, Thomas Anderson and Brian Bershad., "Receiver Based Management of Low Bandwidth Access Links," IEEE INFOCOM 2000.

[4] Puneet Mehra and Avideh Zakhor, Christophe De Vleeschouwer, "Receiver-Driven Bandwidth Sharing for TCP", IEEE INFOCOM 2003

[5] Corin Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", USENIX Association

[6] David Nagle, Denis Serenyi, Abbie Matthews, "The Panasas ActiveScale Storage Cluster – Delivering Scalable High Bandwidth Storage," IEEE SC2004 Conference, November 6-12, 2004.

[7] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, Srinivasan Seshan, Measurement and Analysis of TCP Throughput Collapse in Cluster-Based Storage System," Carnegie Mellon University, September 2007.

[8] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shiy, Yongguang Zhang, Songwu Luz "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers," SIGCOMM'08, August 17–22, 2008.

[9] Mohammad Al-Fares, Alexander Loukissas, Amin Vahdat, " A Scalable, Commodity Data Center Network Architecture ", SIGCOMM'08, August 17–22, 2008.

[10] Mather Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian1;4, Yongguang Zhang1, Songwu Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers ",SIGCOMM'09, August 2009.