Network Burst Peak TCP Congestion Control

Chun-lin WANG, Jian-yong SUN, Wan-jin XU and Xiao-lin CHEN

School of Information Science & Technology, Chuxiong Normal University, P.R. China
*Corresponding author

Keywords: Congestion control, TCP, DCTCP, NewReno.

Abstract. In this paper, the common greedy congestion control protocols are studied, and their shortcomings and improvement methods are analyzed. In data centers, network applies online data-intensive (OLDI) applications which have the characteristics of low latency, high throughput, and high burst, requiring query response latency control of less than 300 milliseconds in the data center. The DCTCP congestion control protocol based on explicit congestion notification (ECN) achieves better performance in data center network applications, but requires all Switches to support ECN and are not compatible with all Switches. The greedy congestion control protocol is improved to meet the needs of the data center. In this paper, there are two contributions. First, the experimental research on greedy congestion control and DCTCP protocol is carried out; Secondly, a unified management congestion control protocol based on the sending window is proposed. It can have better versatility and performance similar to DCTCP, and the packet loss rate is 56% lower than NewReno.

Introduction

TCP/IP is currently the most widely used communication protocol and is the core of the Internet. The function of TCP is to divide the data generated by the communication into multiple sub-segments and provide connection-oriented, reliable communication services and traffic control, so that the data can quickly and correctly reach the receiver through the network. When an application sends data, it needs to compete with other programs for bandwidth, resulting in network congestion. Congestion is closely related to network utilization rate. At present, the focus of research is on the matching of congestion with utilization rate. TCP limits the amount of data entering the network through congestion control. According to the different methods of congestion control, there are currently Tahoe, Reno, NewReno, SACK, Vega, DCTCP and so on.

TCP is the core protocol for Internet to achieve reliable communication. With the development and application of the network, while the role of TCP is becoming more and more important, new issues have emerged along with it, and the improvement and promotion of TCP has continued to be studied. Its purpose is to enhance the performance of TCP. Earlier versions of TCP are defined in RFC 793. TCP is a connection-oriented and reliable transmission protocol with traffic control and congestion mechanisms, which controls the transmission rate by dynamically adjusting the value of the window(cwnd). Before sending data, the message needs to be divided into multiple packets and numbered to send them in sequence. When a packet is received, the receiver carries an AckN through the Datagram and returns it to the sender to indicate that the first N-1 packet has been correctly received. cwnd = min [Reception window, congestion window]. In this discussion, the receiving window is assumed to be large enough, that is, the cwnd = congestion window. TCP congestion control can be divided into five stages: Slow start, Confusion Avoidance, Fast Retransmission, Fast Recovery, and Timeout Retransmission. Through Ack and RTT, the congestion status of the network is judged to adjust the transmission rate. TCP distinguishes between slow start and Confusion Avoidance through the values of ssthresh and cwnd, such as formula (1):

\[
\begin{align*}
\text{slow start phase: } & cwnd(t) + 1, \\
\text{congestion avoidance phase: } & \frac{1}{cwnd(t)}, \\
\text{if } cwnd(t) < \text{ssthresh; } & \\
\text{if } cwnd(t) \geq \text{ssthresh}; &
\end{align*}
\]
When the value of cwnd is smaller than sssthresh, TCP is in the slot-start stage. After each RTT time, cwnd is twice as long as cwnd-1, the cwnd growth in this stage is exponential. When cwnd is greater than or equal to sssthresh, TCP is in the congestion avoidance phase. When each RTT passes through this stage, cwnd only adds one unit, cwnd +1, and greedily approaching the maximum value, also known as linear growth. When checking for network congestion, if a packet is lost or the timer is timeout, then sssthresh ← cwnd/2, reset cwnd←1, and enter the slow start phase again. The TCP sender uses Ack to confirm that the packet is received correctly. When a packet arrives at the receiving end in the correct order, the sender receives Ack and deletes the message that has completed transmission and begins sending the next packet. If a discontinuous packet is received, the same Ack value is returned to the sender. TCP uses the Ack confirmation mechanism to achieve reliable communication.

Common TCP Congestion Control

TCP congestion control focuses on how to reduce the cwnd value of the sending window after congestion occurs. Add Fast Retransmission mechanism to TCP Tahoe method [1], if three identical Acks is received in a row, enters the fast retransmission phase, IE sssthresh ← cwnd / 2, reset cwnd←1, and enter the slow start phase. This method causes cwnd to fluctuate greatly, thus affecting the rapid fluctuation of the transmission rate, resulting in low network utilization. In order to solve the above problems existing in TCP Tahoe, the TCP Reno is proposed in the literatures [2-4], which is currently the most widely used TCP version. The mechanism cited for Fast recovery changes Tahoe's receipt of three duplicate Acks immediately into the slow start stage to receive three duplicate Acks and then enter the Fast recovery phase, that is, sssthresh←cwnd/2 and cwnd←cwnd/2, and then enter congestion avoidance stage. This method can effectively solve the problem of rate fluctuation and network congestion, and improve network utilization rate.

The literature [5] proposes that NewReno will improve Reno's fast recovery. NewReno will not immediately end the fast recovery when it receives Partial Ack. Instead, it waits for the sender to resend the Datagram after the Partial Ack, and fast recovery will not end until all lost packets are retransmitted. This does not need to wait for Timeout to achieve retransmission and improve TCP efficiency. Although NewReno can solve the problem of the loss of a large number of data packets, NewReno can only retransmit one data loss at an RTT time. In order to solve the problem of the loss of a large number of data packets more effectively, another solution is proposed, that is, let the sender know what has been received so that the lost array packet can be selectively retransmitted instead of retransmitting all the Datagrams after the lost Datagram. In response to this issue, SACK [6] is proposed, namely, adding a SACK option to Acknowled to allow the receiver to return the received range to the sender when it returns the Duplicate Ack. Through this information, the sender can determine which information has been received and which information should be retransmitted, so SACK can choose how many packets to retransmit within a RTT time at the transmitter. The disadvantage is that more receiving caches need to be used at the receiving end to store unordered incoming datagrams, wait for the retransmission of lost datagrams, and eventually hand it over to the upper layer protocol in sequence.

Brakmo and Peterson propose another congestion control algorithm that uses RTT to measure the status of the network, called TCP Vega [7], it is another derivative version of Reno. It determines whether to increase or reduce the congestion window value by comparing the actual RTT with the expected effect. The algorithm is described as follows:

\[
\text{Diff} = \frac{\text{Expected} - \text{Actual}}{\text{WindowSize} \div \text{BaseRTT}}
\]

(2)

\[
\text{BaseRTT} = \text{minimum of all measured RTT}
\]

(3)
Vegas adjusts the size of the CND according to the dynamically low queue length of the Switch. When the value of Diff is greater than the beta value, it means that the speed of transmission is too fast and the value of cwnd needs to be reduced to slow down the speed of transmission; Conversely, when the value of the Diff is less than α, it means that the network utilization rate is too low, and the speed of sending needs to be increased by increasing the value of cwnd. The value of Diff is similar to the queue length in the network path, and Vega's ideal expectation is that the length of the queue will fluctuate between α and β. 3. Modify the Slow-start phase: Compared to TCP Reno, TCP Vega not only hopes to use bandwidth quickly and efficiently, but also hopes to avoid data loss due to rapid growth. At this stage, TCP Vegas slows the growth of CND, and the value of cwnd will not double until about two RTT times. Unlike TCP, TCP is doubled after one RTT time, only 50% of TCP. Vegas adjusts the value of ssthresh based on the difference between the expected transmission rate and the actual transmission rate. When TCP's Vegas detects that the network begins to be queued, Vegas enters the Congestion Avoidance phase from Slow-start. In this algorithm, α and β are a fixed value.

TCP Vegas improves and perfects the three places of TCP Reno. This method is feasible in small networks, where the cost of measuring queue length in complex networks is very high. Second, in the Slot-start phase, the speed of transmission has grown too slowly, but in fact, congestion rarely occurs during this period.

Data Center Congestion Control

With the rise of data centers in recent years, there is a kind of important application in data centers, called online data-intensive (OLDI) applications, such as search engines, electronic shopping and advertising, which have the characteristics of low latency, high throughput and high burst. For a better user experience, the query response latency in the datacenter is controlled below 300 milliseconds, which is also known as Soft timeout constraints. Traditional TCP protocols cannot be implemented. In OLDI applications, the client decomposes the query task into multiple sub-tasks assigned to hundreds of servers. When the server completes the query and returns the query result to the client through the Switch, the server sends a Datagram (N :1) to the Switch at the same time, as shown in Figure 1, the queue of the Switch will be used up instantaneously until a buffer overflow occurs. After the packet is lost, TCP has two ways to solve the data retransmission: First, when the timer arrives, it triggers a time-out retransmission; The second is that the sender receives three duplicate Ack confirmation messages that trigger retransmission. The retransmission timer (RTO) of the traditional TCP is generally not less than 200 ms, while the round-trip delay (RTT) of the data center network environment is generally in the order of order of microseconds. During this period of time, the server's sending window continuously sends data until the sending window is used up. From the time when the sending window runs out and stops sending to the beginning of retransmission, the Switch is idle, resulting in a sharp drop in network throughput. This phenomenon is called TCP Incast [8].

![Figure 1. Network center OLDI model.](image)
In 2010, at the SIGCOMM conference, Microsoft Research proposed the DCTCP protocol [9], the idea of which is to use explicit congestion notification (ECN). It first monitors the queue length in the Switch, if the queue length exceeds the upper limit K, the TCP receiver is explicitly notified through the ECN field. The receiver reduces the sender's sending window through the window field, reduces the speed of sending, reduces the number of packets entering the Switch queue, maintains the equilibrium of the length of the Switch queue at a certain length, reduces packet loss, and improves network utilization.

The only difference between DCTCP and TCP sender is that the sender receives an Ack packet with an ECN flag. The treatment adopted is different: The TCP enters the fast recovery phase when it conducts slow-start or receives three identical ACK confirmation packages; DCTCP adjusts the window by Formula 6. When α is close to 0, the window size is only fine-tuned. When the queue exceeds K, the DCTCP sender begins to slowly reduce the window. This allows DCTCP to maintain a low queue length while ensuring high network utilization. When α = 1, it is the same as the TCP protocol, and the window value is halved.

\[ \text{cwnd} \leftarrow \text{cwnd} \times \left(1 - \frac{\alpha}{2}\right) \]  

(5)

The Proposed Solution

This paper proposes a congestion control algorithm for centralized transmission window control. The idea is: in the OLDI application of the data center, if the sending window is too large, multiple senders will continuously send data to the network, causing the Switch queue to overflow and lose packets and idle, thus reducing the performance of the network. If the sending window is too small, the network utilization and throughput will be reduced. In this algorithm, the sum of the sending windows of all service groups is centrally controlled and constrained to match the cache size of the Switch, so that the queue length of the Switch fluctuates within a certain range. It not only avoids overflow and packet loss in the Switch queue, but also ensures high network utilization and low latency. In the algorithm, the general window is initialized first, and the average value of the general window is taken as the upper limit of each server sending window. If no packet is lost, the value of the total window should be adjusted; If the retransmission lowers the value of the total window, the upper limit of the server send window should be updated again. If the data to be sent by the server is smaller than that of other servers, the upper limit of the sending window of this server is reduced and the upper limit of the sending window of other servers is increased. In this algorithm, the size of the total window is one of the research focuses of this paper:

Algorithm Description:

Initial: The upper limit of the total window of the link is \( \text{wnd\_cap}=\min[\text{swich\_link\_buf}_1,\text{swich\_link\_buf}_2,\ldots,\text{swich\_link\_buf}_n] \); \( \text{swich\_link\_buf}_n \) is the number of Switches in the link, the upper limit of the link window is \( \text{window\_}=0 \); link data \( N \) = the number of servers.

Update the total size of the wnd\_cap sending window: \( g=0.8 \), queue\_length: current Switch queue length.

\[
\begin{align*}
\text{wnd\_cap} &= \text{wnd\_cap} - 1 & \text{queue\_length} &> g * \text{swich\_link\_buf} \\
\text{wnd\_cap} &= \text{wnd\_cap} + 1 & \text{queue\_length} &< g * \text{swich\_link\_buf}
\end{align*}
\]  

(6)

Experiments

In this paper, Tahoe, TCP Reno, NenReno, SAck, and DCTCP are tested for performance in the data center application model in Figure 1 to compare their performance. In the experiment, the NS2[10] emulator is used to simulate the data center network structure shown in Figure 1, and eight servers sent packets to the client at the same time. In the experimental setting, the link bandwidth is 1Gb, the upper limit of the sending window is 100 packets, and 8 services send a total of 2MB of data. In the experimental verification, many network performance parameters such as the length of the Switch
queue, the change in the size of the sending window, the completion delay, and the packet loss rate are emphatically observed.

**Queue Length**

In the Switch queue length test, the cache upper limit of the Switch is set to 50 packets. The queue length is shown in Figure 2. The best performance is the DCTCP protocol, and its queue length has always been fluctuating between [13, 15], indicating that the speed of delivery of the eight senders is controlled and there is no greedy use of bandwidth. The performance of the proposed algorithm is poor, its queue length has been fluctuating around 45, maintaining a high Switch utilization rate, and its transmission delay is the best of all tests. As for other types of congestion control, at the beginning, due to greedy algorithms constantly accelerating the sending speed of the sender, the queue of the Switch overflowed, causing the loss of Datagram, which triggered the transmission of Datagram repeatedly. Since the sending window has been used up while waiting for retransmission, the Switch is periodically idle and the network utilization rate is reduced.

**Sending Window**

In the experiment, because the data is sent to the client by 8 servers at the same time, in order to facilitate the observation of window size changes, the change in the window value of server 2 is uniformly observed, as shown in Figure 3. In the Sending window change test, the size of the Sending window is controlled by DCTCP using explicit congestion notification (ECN), so the change in window size is very small throughout the test period, sacrificing network utilization to reduce network congestion. All other methods and the proposed method use the greedy congestion control method, so the window value fluctuates greatly, in all the greedy congestion control methods, the fluctuation of the proposed method is small, and a better balance is struck between the network utilization rate and the congestion control.

**Transmission Delay**

![Figure 4. Transmission delay.](image)

![Figure 5. Network loss rate.](image)
In the transmission delay test, the transmission delay is tested by sending 8KB, 24KB, 72KB, 216K, 648KB ... 157MB of data. As shown in Figure 4, when sending smaller data, data loss occurs when the greedy congestion algorithm overflows the Switch queue, especially the loss of the end Datagram, a large number of network bandwidth processing is in space and causes data transmission failure. Conversely, when sending large data, assuming that a Datagram is lost, subsequent data can continue to be sent without causing network idle.

Packet Loss Rate

The packet loss rate test is shown in Figure 5. The network with bandwidth from 100Mb to 100Gb is adopted to send 200KB of data, the loss rate of different congestion control methods is tested in different network bandwidth. In the test of packet loss rate, the method used is very similar to DCTCP, which is much better than other congestion control methods and has a small packet loss rate.

Summary

This paper first analyzes the different congestion control protocols, including the greedy algorithm NewReno and the explicit congestion notification DCTCP protocol. DCTCP has good performance in data center congestion control, but it requires all exchange nodes to support (ECN), which limits its applicability to low-end Switches. The proposed congestion control method is based on the greedy algorithm, which reduces the loss rate by centralizing the cwnd window on the sender and improving the network utilization rate. In the data center network environment simulation experiment, it is close to DCTCP in performance and has good versatility. Compared with NewReno, it reduces the loss rate by 56% and increases the transmission delay by 30%.

Acknowledgement

This work is supported in part by Yunnan Provincial Department of Education Project and the Science and Technology Program of Yunnan Province, China under grant No.2018JS454 and 2017FH001-124.

References