International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Survey on Congestion Control Mechanism for TCP

Tejashri P. Mane[1], Snehal Kanade[2]

[1]Department of Computer Engineering, SKN Sinhgad Institute of Technology and Science,
Gat No. 309/310, Kusgaon (Bk.) Off Mumbai-Pune Expressway, Lonavala, Tal Maval, Dist Pune-410401, India

[2]Professor, Department of Computer Engineering, SKN Sinhgad Institute of Technology and Science,
Gat No. 309/310, Kusgaon (Bk.) Off Mumbai-Pune Expressway, Lonavala, Tal Maval, Dist Pune-410401, India

Abstract: ***Many-to-one traffic pattern is common in many data centered applications, where multiple senders sending data to one receiver in parallel .This many-to-one traffic patterns overwhelms the single receiver and leads to performance degradation in such applications. The reason for this performance degradation is the incast congestion occurred during data transfer. This article focusses on various congestion control mechanism for TCP which will detect congestion occurred during data transfer .Congestion may cause due to the switch buffer overflow or link in congestion. This study focusses on window based congestion control mechanisms. Here, the TCP incast is studied by considering the relationships between the TCP throughput, Round-Trip Time(RTT),sending window and receiver window.***

Keywords: Incast congestion, TCP, RTT, Many-to-one traffic pattern.

1. Introduction

TCP is a end-to-end protocol. The Transmission Control Protocol (TCP) is used as the transport-layer protocol for reliable data transfer in data center networks same as it is on the Internet. Reliable transmission is accomplished by means of the utilization of a retransmit clock: for the segments sent each one time, the sender expects an ACK from the recipient before the clock terminates. If ACK is not received within time, then a few fragments thought to be lost, because of the network congestion and will be retransmitted at some proper moment later. Sender side TCP uses a congestion window to do congestion avoidance.The congestion window demonstrates the vast measure of information that can be conveyed on an association without being acknowledged. When sender fails to receive an acknowledgement for a packet within the estimated timeout, TCP identifies the congestion.TCP incast collapse occurs due to the highly bursty traffic of multiple TCP connections which overflows the Ethernet switch buffer in a short period of time which causes intense packet loss and thus TCP retransmission and timeouts.

In recent years, the data center applications and web search generally shows the Partition/Aggregate communication pattern. First, a request is partitioned and sent to a number of worker nodes. And then, the response packets generated by the workers are transmitted to a common node for aggregation, i.e., aggregator node. Such type of traffic may cause network congestion, as multiple workers send the response packets to the same aggregator at the same time. This leads to the TCP performance degradation in terms of goodput and query completion time due to the severe packet loss at Top of Rack (ToR) switches. The TCP senders aggressively transmit packets without knowing the network pipe size, i.e., bandwidth-delay product which is extremely small and thus causes TCP throughput collapse.

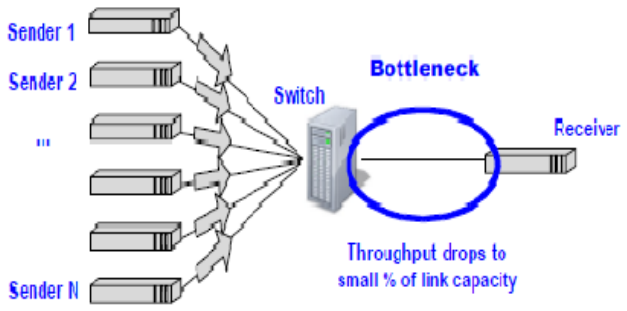

Figure 1: General scenario of to observe incast.

The TCP incast issue was accounted for first by D. Nagle et al. in the design of a scalable storage architecture. They found that the concurrent traffic between a client and many storage devices overflows the network as the number of storage devices increases. This results in multiple packet losses and timeout. To avoid the incast congestion, they reduce the clients receive socket buffer size to under 64kB.They also suggest to schemes such as reducing the duplicate ACK threshold and disabling the slow-start to avoid retransmission timeout.

2. TCP Tahoe

Tahoe refers to the TCP congestion control algorithm which was recommended by Van Jacobson in his paper[1]. TCP packet transmissions are timed by the approaching acknowledgements.It consist of slow-start and congestion avoidance phase.

2.1 Slow Start

This stage is begun at whatever point there is another TCP association or restarts after the packet loss. Slow starts recommends that the sender set the congestion window to 1 and afterward for every ACK got it expand the CWD by 1.

So in the first round trip time(RTT) we send 1 packet, in the second we send 2 and in the third we send 4. Thus we increase exponentially until we lose a packet which is a sign of congestion. When we experience congestion we diminishes our sending rate and we diminish congestion window to one. And start over again.

2.2 Congestion Avoidance

For congestion avoidance Tahoe uses 'Additive Increase Multiplicative Decrease'. A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window as a threshold value. It then set CWD to one and starts slow start until it reaches the threshold value. After that it increments linearly until it experiences a packet loss. Thus it increase it window slowly as it approaches the bandwidth capacity.

2.3 Problem

Tahoe utilizes the coarse-grained timeouts.It detects packet losses by timeouts. It take a complete timeout interval to detect a packet loss. Also since it doesn't send immediate ACK's, it sends aggregate acknowledgements, therefore it follows a 'go back n ' approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

3. TCP Reno

Reno requires that we receive immediate acknowledgement whenever a segment is received[3]. Whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. Reno introduced the 'Fast Re-transmit' algorithm.

3.1 Fast Re-Transmit

1) Each time we receive 3 duplicate ACK's it means that the segment was lost and re-transmit the segment immediately and enter 'Fast- Recovery'
2) Set SSthresh to half the current window size and also set CWD to the same value.
3) For each duplicate ACK receive increase CWD by one. If the increased CWD is greater than the amount of data in the pipe then transmit a new segment else wait. If there are 'w' segments in the window and one is lost, the we will receive (w-1) duplicate ACK's. Since CWD is reduced to W/2, therefore half a window of data is acknowledged before we can send a new segment. After retransmitting a segment, wait for atleast one RTT before we would receive a new acknowledgement. Whenever we receive a new ACK we reduce the CWND to SSthresh. If we had previously received (w-1) duplicate ACK's then at this point we should have exactly w/2 segments in the pipe which is equal to what we set the CWND to be at the end of fast recovery. After that continue with congestion avoidance phase of Tahoe.

3.2 Problem

It can only detect a single packet losses. If there is multiple packet drop,then the first insight about the packet loss comes when we receive the duplicate ACK's. But the data about the second packet which was lost will come when the ACK for the retransmitted first packet achieves the sender after one RTT. Also it is possible that the CWD is reduced twice for packet losses which occurred in one window. Another issue is that if the widow is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast re-transmit and we would have to wait for a coarse grained timeout.

4. TCP New-Reno

New RENO is a slight change over TCP-RENO[3]. It has the capacity to locate multiple packet loss and along these lines is considerably more effective that RENO in the occasion of multiple packet losses. Like Reno, New-Reno likewise goes into fast retransmit when it gets different duplicate packets, then again it contrasts from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWD multiples times[4]. The fast-retransmit phase is the same as in Reno.

4.1 Fast Recovery

Like Reno, New-Reno likewise goes into fast retransmit when it gets different duplicate packets, however it differs from RENO in that it doesn't exit fast-recovery until all the data which was out standing at the time it entered fast recovery is acknowledged. Thus it overcomes the problem faced by Reno of reducing the CWD multiples times. The fast-transmit phase is the same as in Reno.

4.2 Problem

New-Reno suffers from the fact that its take one RTT to detect each packet loss. At the point when the ACK for the initially retransmitted packet is gotten at exactly that point would we be able to find which other segment was lost.

5. SACK

TCP with 'Selective Acknowledgments' is an extension of TCP Reno and it works around the problems face by TCP RENO and TCP New-Reno, specifically detection of multiple lost packets, and re-transmission of more than one lost packet per RTT.

SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. In this manner the sender has a picture of which fragments have been recognized and which are as yet remarkable. Whenever the sender enters fast recovery, it initializes a variable pipe which describes the how much data is outstanding in the network, and it also set CWND to half the current size. Each time it receives an ACK it decreases

the pipe by 1 and every time it retransmits a segment it increments it by 1.Whenever the pipe goes smaller than the CWD window it checks which segments are unreceived and send them. If there are no such segments outstanding then it sends a new packet. Thus more than one lost segment can be sent in one RTT.

5.1 Problem

The most serious issue with SACK is that as of now selective acknowledgements are not given by the recipient.

6. Vegas

Vegas is a TCP implementation which is a modification of Reno. It is a proactive measure to encounter congestion which is much more efficient than reactive ones. It beats the issue of coarse grain timeouts by proposing a calculation which checks for timeouts at an extremely proficient schedule. Also it beats the issue of requiring enough duplicate acknowledgements to detect a packet loss, and it also recommend a modified slow start algorithm which prevent it from congesting the network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur.
The three major changes introduced by Vegas are:

6.1 New Re-Transmission Mechanism

Vegas extends on the re-transmission mechanism of Reno. It stays informed regarding when each one fragment was sent and it moreover computes an assessment of the RTT by staying informed regarding to what extent it takes for the affirmation to get back.

Whenever a duplicate acknowledgement is received, check whether (current time-segment transmission time)> RTT estimate; if it is then it immediately retransmits the segment without waiting for 3 duplicate acknowledgements or a coarse timeout. Thus it overcomes the problem faced by Reno of not being able to detect lost packets when it had a small window and it didn't receive enough duplicate Ack's.

To catch whatever other portions that may have been lost preceding the retransmission, when a non duplicate acknowledgement is received, if it is the first or second one after a fresh acknowledgement then it again checks the timeout values and if the segment time since it was sent exceeds the timeout value then it re-transmits the segment without waiting for a duplicate acknowledgment. Thus in this way Vegas can detect multiple packet losses. Also it only reduces its window if the re-transmitted segment was sent after the last decrease. Along with this it overcome Reno's problem of reducing the congestion window multiple time when multiple packets are lost.

6.2 Congestion avoidance

It decides the congestion by a decrease in sending rate as compared to the expected rate, as result of large queues developing up in the routers. Accordingly at whatever point the figured rate is too far from the normal rate it builds transmissions to make utilization of the accessible transfer speed, whenever the calculated rate comes too close to the expected value it declines its transmission to anticipate over saturating the bandwidth. Thus Vegas prevents congestion quite effectively and doesn't waste bandwidth by transmitting at too high a data rate and creating congestion and then cutting back, which the other algorithms do.

6.3 Modified Slow-start

When a connection first starts it has no idea of the available bandwidth and it is possible that during exponential increase it over shoots the bandwidth by a big amount and thus induces congestion. To this end Vegas increases exponentially only every other RTT, between that it calculates the actual sending through put to the expected and when the difference goes above a certain threshold it exits slow start and enters the congestion avoidance phase[6].

7. Conclusion

Congestion may happen due to the multiple senders overwhelms the single receiver or switch buffer overflows or link is in congestion. The mechanisms discussed above gives the congestion control mechanisms for TCP at sender side. All the mechanisms discussed above uses the congestion window to adjust the sending rate thus preventing the congestion. Vegas has the advantage over all other mechanisms. It detects the packet loss much sooner as compared to other mechanisms. It prevents congestion efficiently and doesn't waste the bandwidth by transferring data at high rate.

References

[1] V. Jacobson "Congestion Avoidance and Control" SIGCOMM Symposium no communication and architecture protocol.
[2] V.Jacobson "Modified TCP Control and Avoidance Alogrithms".Technical Report 30,Apr 1990.
[3] Fall, K., and Floyd, S. Simulation-based Comparisons of Tahoe, Reno, and Sack TCP. *ACM Computer Communications Review 26*, 3 (July 1996), 5–21.
[4] S.Floyd, T.Henderson "The New-Reno Modification to TCP's Recovery Algorithm" RFC 2582, Apr 1999.
[5] Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A. *TCP Selective Acknowledgment Options*. Internet Engineering Task Force, 1996. RFC 2018.
[6] L.S. Brakmo, L.L Peterson, "TCP Vegas:End to End Congestion Avoidance on Global Internet" IEEE Journal on Selected Areas in Communication Vol. 13, 1995(1465-1490).