International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 1, January- 2013

Application Based Approach to Address TCP Incast Problem in Data Center Networks

Anurag Pathak
B.E. (Computer Engineering)

Abstract

Data Center Network (DCN) typically serving millions of data and services to the client. DCN is having a typical architecture of clients and servers in which one client is served by number of stripped servers which are connected via network switch. A client is generally expecting TCP/IP application service form DCN servers, as client has TCP/IP based applications. But as the data on sever is stripped and Data Center Network has millions of data, sometimes congestion comes to an account. This congestion is called as TCP's Incast Problem, which is not fully addressed yet.

This paper is trying to focus on a solution which can be implemented at the client side by splitting RTOmin (Retransmission Timeout) of Operating System.

1. Introduction

TCP Incast Problem is recently introduced, which has very large impact on Data Center Networks, as DCN is continuously serving the clients. Typical DCN is having a structure of one client is connected to a switch which stripes the number of severs, and the data that client needed is also stripped on it. The terminology is DCN servers having RTT (Round Trip Time) which is quiet less than RTOmin of the client. That means, for one timeout (RTO) at the client, there might have several RTT at the server end. Hence the whole bombarding of retransmitted TCP packets at the switch. This scenario is quite responsible for the congestion at the switch and this is bottleneck as shown in following Figure 1. Therefore the performance is drastically goes down. This problem is named as TCP Incast behaviour. The incast behaviour can be observed by using simple setup of client and servers as shown in Figure 1. The client is connected to stripped server via network switch, and requesting for block number k to the server. This k block is stripped over the network that means, if there are four stripped server then the four equal parts of block k is stored on each server. In order to complete the request, server has to respond through the switch. As the server has very short RTT as compare to RTO and hence congestion (TCP incast) occurs. In this case the due to congestion the server is unable to send the block k and until client doesn't receive the block k, client is going to request block k+1. Therefore this is responsible for degradation of the performance drastically [1].

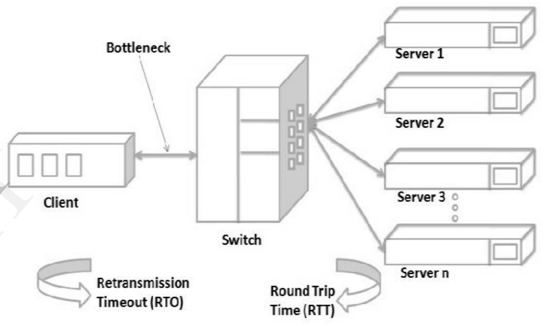

Fig. 1: Simple Setup to observe Incast

2. Background

Business cost efficiencies mean that a vast majority of data centers rely on off-the-shelf rack-mount servers interconnected via high-speed Ethernet switches (DCN). Today, entry level gigabit Ethernet switches support up to 40 ports and switch upwards of 50 million packets per second when operating at full data rates. Commodity 10 Gbps Ethernet is now cost-competitive with specialized interconnects such as Infiniband and FibreChannel, with the added benefits of wider brand recognition [2]. To reduce costs however, Ethernet switches often sacrifice expensive, power hungry SRAM packet buffers, the effect of which explore throughout this work. The desire for commodity parts extends to transport layer as well. Here, TCP provides a kitchen sink of protocol features, giving reliability, retransmission, congestion control, and inorder byte stream at the receiver. While not all applications need all of these features [3] [4] or benefit from more rich transport abstractions [5], TCP is mature and well-understood by developers, leaving it as the transport protocol of choice even in many high performance environments. Without link level flow control, TCP is solely responsible for coping with and avoiding packet loss in (often small) Ethernet switch egress buffers. Unfortunately, the workload examine in the context of incast has three features that challenge TCP's performance: a highly parallel, synchronized request workload; buffers much smaller than the bandwidth delay product of the network; and low latency that results in TCP having windows of only a few segments.

Following tables are showing the RTOmin of different Operating System and RTT of different networks. [1]

Table 1: Default TCP Minimum Retransmission Timeout Values

Operating System	**Default TCP RTOmin**
Linux	200 ms
BSD	200 ms
Solaris	400 ms

Table 2: Typical Round Trip Time Values for Different Networks

Network	**Round Trip Time**
Wide Area Network	100ms
Data Center	< 1ms
Storage Area Network	< 0.1ms

As far as the performance is concern, it is always measured in terms of two parameters. First is the number of servers and second is Throughput.

3. Existing Solutions

The existing solution has been specified at the TCP level and has quite better result. There are three solution which are common and that address the TCP incast behaviour up to some extent. First is Large Switch Buffer, second is Increasing SRU (Server Request Unit) and third is Reducing Timeout Penalty [1].

3.1. Large Switch Buffer

This technique [2] tries to mitigate the root cause of timeouts – packet losses – by increasing the buffer space allocated per port on the switches. Figure 3.1 shows that doubling the size of the switch's output port buffer, doubles the number of servers that can supported before the onset of incast.

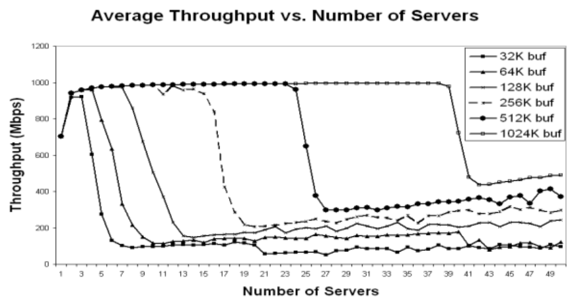

Fig. 2: Simulation Result of Large Switch Buffer

Consequently, given the number of servers, incast can be avoided with a large enough buffer space. Unfortunately, switches with larger buffers tend to cost more, forcing system designers to choose between over-provisioning and hardware budgets. This suggests that a more cost-effective solution is needed to address the problem of incast.

3.2. Increasing Server Request Unit

This is another incast countermeasure discussed in [2]. It aims to mask incast by utilizing the spare link capacity of the stalled flow in transferring larger SRUs belonging to other flows. Figure 3.2 illustrates that increasing the SRU size improves the overall throughput. With 7 servers, the throughput for 1000KB SRU is two orders of magnitude greater than that of the 256KB SRU.

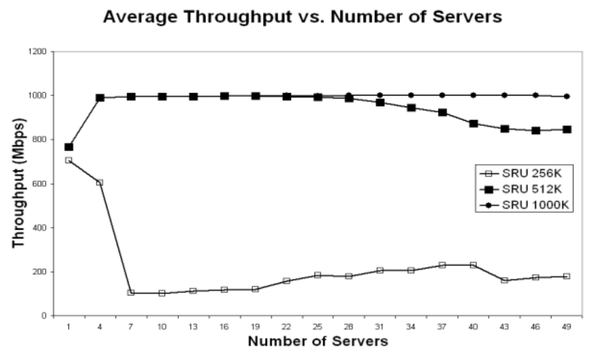

Fig. 3: Simulation Result of Increasing Server Request Unit

TCP performs well in settings without synchronized reads, which can be modelled by an infinite SRU size. With larger SRU sizes, active servers will use the spare link capacity made available by any stalled flow waiting for a timeout event; this effectively reduces the ratio of the timeout times, during transfer time. Unfortunately, an SRU size of 1 megabyte is quite impractical: most applications ask for data in small chunks, corresponding to an SRU size range of 1-256KB. This is because, larger the SRU size, greater is the prefetching that the storage system has to commit to. With prefetching, the storage system needs to allocate pinned space in the client kernel memory, increasing the memory pressure at the client. This increased pressure at the client, often leads to a kernel failure. Hence it is really not advisable to use larger SRUs on the cluster storage system.

3.3. Reducing Timeout Penalty

This technique is proposed in [6], aims to address incast by reducing the time spent waiting for a timeout. The amount of time a flow waits before retransmitting a lost packet without the Fast Retransmit mechanism provided by the three duplicate ACKs, is determined by TCP's Retransmission Timeout (RTO). Estimating the RTO value trades timely response to losses for premature timeouts. A premature timeout has two negative effects:
3.3.1. It leads to a spurious retransmission and
3.3.2. With every timeout, TCP reduces its slow start threshold (ssthresh) value by half and enters Slow Start even though no packets were lost.
Since there is no congestion, TCP thus would underestimate the link capacity and throughput would suffer. TCP has a conservative minimum RTO (RTOmin) value to guard against such spurious retransmissions [7] [8].

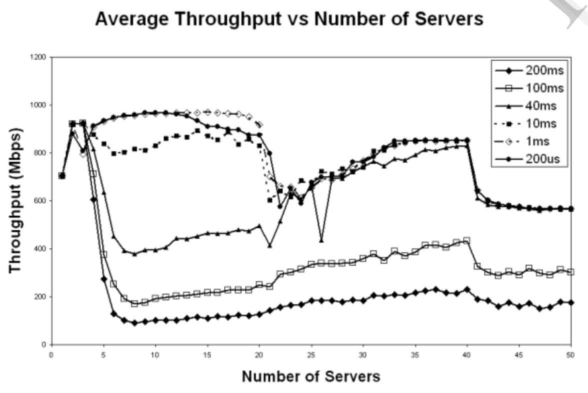

Fig. 4: Simulation Result of Reducing Timeout Penalty

Popular TCP implementations use an RTOmin value of 200ms [9]. Unfortunately, this value is orders of magnitude greater than the round-trip times in Storage Area Network settings, which are typically around 100µs. This large RTOmin imposes a huge throughput penalty because the transfer time for each data block is significantly smaller than RTOmin. Figure 4 show that reducing RTOmin from 200ms to 200µs improves throughput by an order of magnitude beyond 5 servers. In general, for any given SRU size, reducing RTOmin results in an order of magnitude improvement in TCP's throughput. The figure also shows that even with an Aggressive RTOmin value of 200µs, TCP still observes a 30% decrease in goodput for 40 or more servers. Unfortunately, setting RTOmin to such a small value poses significant implementation challenges and raises questions of safety too.
1) Implementation Challenges: Reducing RTOmin to 200µs requires TCP clock granularity of 100µs, according to the standard RTO computation algorithm [7] [8]. BSD TCP and Linux TCP implementations are currently unable to provide this fine grained timer. BSD implementations expect the OS to provide two coarse-grained "heartbeat" software interrupts every 200ms and 500ms, which are used to handle internal per-connection timers [10]; Linux TCP uses a TCP clock granularity of 10ms. A TCP timer in microseconds needs either hardware support that does not exist or efficient software timers [11] that are not available in most operating systems.
2) Safety and Generality: Even if sufficiently fine grained TCP timers were supported, reducing RTOmin value can be harmful, especially in situations where the servers communicate with clients in the wide-area. In [12], the authors note that RTOmin can be used for trading "timely response with premature timeouts" but there is no optimal balance between the two in current TCP implementations; a very low RTOmin value increases premature timeouts. Earlier studies of RTO estimation in similar high-bandwidth, low-latency ATM networks also show that very low RTOmin values result inspurious retransmissions [13] because variation in round-triptimes in the wide-area clash with the standard RTO estimator's short RTT memory.
In summary, the proposed solution in [6] should be viewed with caution as it increases the risk of premature timeouts.

4. Application Level Solution

The problem is at the receiver side, because whatever the server sending data with high RTT, receiver (client) is not capable of accepting those data packets. As we have TCP protocol form Switch to Client.

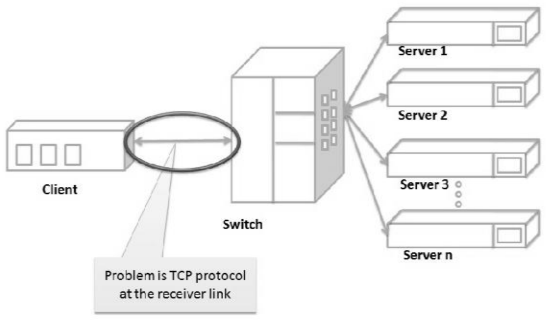

Fig. 5: Solution at the application level

Author names and affiliations are to be centered beneath the title and printed in Times 12-point, non-boldface type. Multiple authors may be shown in a two- or three-column format, with their affiliations italicized and centered below their respective names. Include e-mail addresses if possible. Author information should be followed by two 12-point blank lines.

4.1. TCP's Default Recovery Mechanism

In Figure 6, there are 3 duplicate ACKS's for only one packet loss. That will be more penalties to server in order to retransmit the packets.

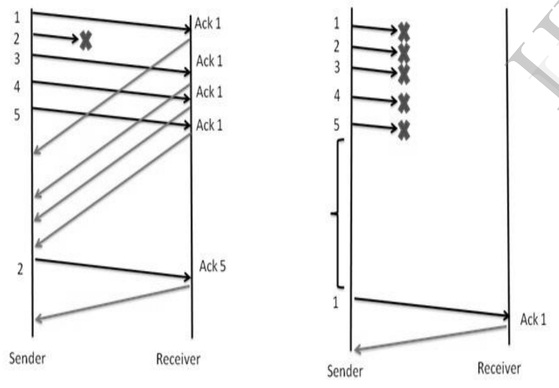

Fig. 6: TCP's default packet loss recovery mechanism

In Figure 6, there are 3 duplicate ACKS's for only one packet loss. That will be more penalties to server in order to retransmit the packets. If all the packets were lost then TCP protocol comes one by one sending of packets on the link. It is more time consuming because it is considering that the problem with link.

4.2. How to minimize Duplicate ACK to improve TCP throughput at the time of Incast?

The above question is valid as far as the link performance is concern, In order to fix the problem we can use following architecture as shown in Figure 7. If we divide the RTOmin of the client into some timeslots then the client can save the link idleness. It can be possible and shown in following Figure 8

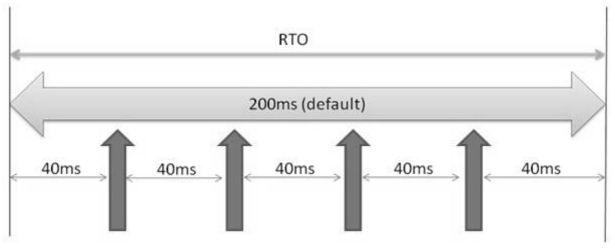

Fig. 7: Architecture of Solution at the application level

In Figure 8, in first TCP communication there is an acknowledgement of only last packets, so here it says that the client had also received the first packet. Similar in second case, after complication of timeslot, it is been check that any packet is receiving or not. If there is no packet received then it automatically sends an acknowledgement of received packet. And in third case there is only acknowledgement of packet that is lost.

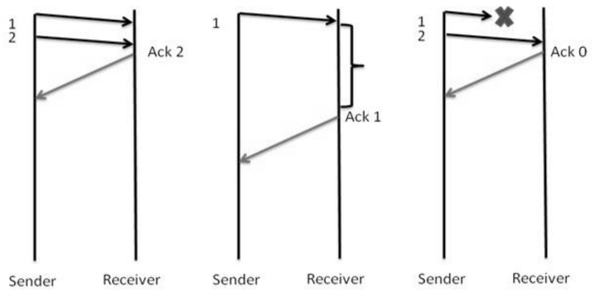

Fig. 8: TCP's recovery mechanism after timeslots

5. Conclusion

TCP Incast problem really affects the TCP Throughput in Data Center Network. Receiver TCP link is weak as far as performance is concern. This solution is dealing with the Retransmission Timeout of client to avoid link idleness. And it can be done by taking Linux operating system into the consideration that TCP operation can be handle very effectively into open source system. Hence to improve TCP performance, application level modification is kind of solutions to establish an Incast Congestion free Data Center Network.

6. References

[1] Santosh Kulkarni and Prathima Agrawal, "A Probabilistic Approach to Address TCP Incast in Data Center Networks", *2011 31st International Conference on Distributed Computing Systems Workshops*, IEEE, Location, Date, pp. 26-33.
[2] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems," in FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.
[3] E. Kohler, M. Handley, and S. Floyd, "Designing dccp: Congestion control without reliability," 2003.
[4] S. Raman, H. Balakrishnan, and M. Srinivasan, "Itp: An image transport protocol for the internet," 2000.
[5] B. Ford, "Structured streams: a new transport abstraction," SIGCOMM Comput. Commun. Rev., vol. 37, no. 4, pp. 361–372, 2007.
[6] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, B. Mueller, and P. Inc, "Safe and effective fine-grained tcp retransmissions for datacenter communication."
[7] V. Jacobson and M. J. Karels, "Congestion avoidance and control," 1988.
[8] V. Paxson and M. Allman, "Rfc 2988: Computing tcp's retransmissiontimer," November 2000.
[9] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp," in In Proceedings of USENIX. Springer, 2002, pp. 49–62.
[10] M. Aron and P. Druschel, "Tcp implementation enhancements for improving webserver performance," 1999.
[11] ——, "Soft timers: Efficient microsecond software timer support for network processing," in In Proc. of the 17th Symp. on Operating Systems Principles, 1999.
[12] M. Allman and V. Paxson, "On estimating end-to-end network path properties," 1999.
[13] A. Romanow and S. Floyd, "Dynamics of tcp traffic over atm networks," IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, vol. 13, pp. 79–88, 1994.
[14] Yan Zhang, Student Member, IEEE, and Nirwan Ansari, Fellow, IEEE "On Mitigating TCP Incast in Data Center Networks. Advanced Networking Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07012, United States, 2011"