
Proceedings Paper:
Adesanmi, A and Mhamdi, L (2015) Controlling TCP Incast Congestion in Data Centre Networks. In: IEEE International Conference on Communication Workshop. International Conference on Communication Workshop, 08-12 Jun 2015, London, UK. IEEE , pp. 1827-1832.

https://doi.org/10.1109/ICCW.2015.7247446

Controlling TCP Incast Congestion in Data Centre Networks

Akintomide Adesanmi Lotfi Mhamdi
School of Electronic and Electrical Engineering
University of Leeds, Leeds, UK
l.mhamdi@leeds.ac.uk

***Abstract*— The rapid increase in data centre communication in recent years has led to a wave of interest in the problems that affect communications in the data centre environment. Many data centre applications rely on TCP/IP protocols for reliable data transport, which while successful for many Internet applications, does not integrate with and perform seamlessly in the data centre environment. One of these problems arises in data centre setups where many servers communicate simultaneously, and effectively in parallel, with one client through a single switch. In this scenario, the servers could experience a large drop in throughput due to severe packet loss, a phenomenon known as Incast congestion.**

In this paper, we investigate the Incast problem and introduce a variant of TCP, namely M21TCP-A, which is developed with the idea that the switch can be used to inform each parallel sender (server) of how many other senders are communicating simultaneously. Each sender can then limit its congestion window based on this feedback from the switch. M21TCP-A is evaluated against RED, ECN and DCTCP, which are the most successful congestion control proposals for mitigating Incast. Performance results show that M21TCP-A mitigates Incast thoroughly and outperforms previous proposals.

***Index Terms*— Congestion, TCP, Incast.**

I. Introduction

In recent years, data centres have become very important and instrumental in driving and supporting the new era of Internet cloud computing. Modern data centres host a variety of services and applications such as web search, scientific computing, social networks, distributed files systems, etc [1]. These data centres usually run TCP/IP over an Ethernet network since TCP is low cost and easy to use [2]. Over the years, TCP has succeeded in meeting most communication requirements of Internet-based communications because of its assurance of reliable delivery and its congestion control mechanisms. However, recent research has shown that it falls short in the unique data centre environment with its advanced network topologies, unique workloads and high throughput requirements [3].

Many data centres run soft real time applications that employ a Partition/Aggregate pattern; these have been used for applications such as e-commerce, web search, retail etc. As shown in Figure 1, this pattern usually consists of a multi-layered hierarchical structure where the deadline of computations and responses on one layer affects computations and responses on higher layers. In a simple two-layered structure, for instance, aggregators on higher layers designate tasks amongst the workers on lower layers [4]. When each worker completes a task, it sends the results back to the aggregator (parent node), which compiles these results into a final response. Latency targets drive the performance of more multi-layered partition-aggregate structures. The latency targets, obtained from customer surveys [5], determine the deadlines of each layer in the algorithm tree. If children nodes (workers) miss their deadlines, then their parent nodes (aggregators) send out incomplete responses, leading to lower quality results and thus lower revenue. Therefore, the slowest sending child node limits each parent node.

Partition/Aggregate workflow patterns are just one of the "many to one communication" workload patterns that are affected by a communication problem that is usually found in data centres, called Incast congestion. Incast congestion is a catastrophic loss in throughput that occurs when the number of servers sending data increases beyond the ability of an Ethernet switch to buffer packets. It occurs when multiple servers are communicating through an Ethernet switch and the limited buffer is overwhelmed by a concurrent flood of highly bursty traffic from parallel communicating servers, leading to severe packet loss and consequently one or more TCP timeouts. Parallel senders, in this case, include multiple servers that use the same Ethernet switch or router. These timeouts impose hundreds of milliseconds delay, which almost guarantees that a server (worker in partition/aggregate pattern) misses the deadline and that subsequent data to be sent is not received [6]. Timeouts have been shown to reduce throughput by up to 90% of the link capacity [7].

Considerable research efforts have been advocated to addressing the Incast problem. These proposals can be mainly divided into two categories. The first is an extension of traditional TCP [7] [8]. These approaches inherit the TCP properties and fall short in overcoming the Incast problem. The second category uses rate control and Active Queue Management (AQM) mechanisms. This includes DCTCP [9], D^2CTCP [10], D^3 [11] and pFabric [12]. While these techniques improve the data centre transport capabilities in some aspects, they fail in others, especially with respect to performance (throughput and scalability) and implementation cost.

This article introduces M21TCP-A, a novel approach to solving the Incast problem, where the router informs each sender of the total number of parallel senders by setting a "number of senders" TCP field in every packet that passes through it. The protocol leverages the idea of router based flow

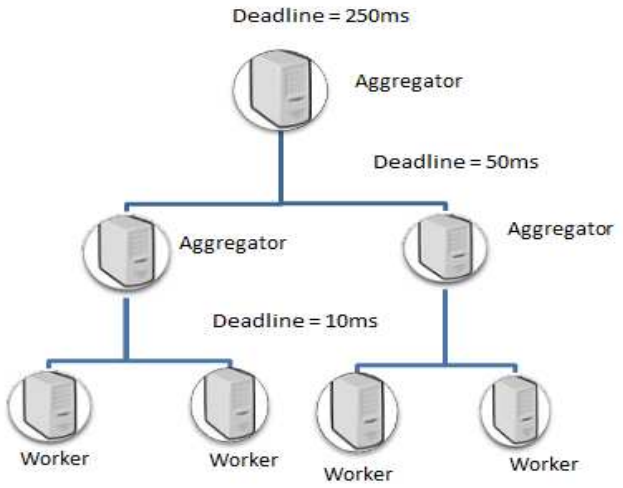

Fig. 1. Partition Aggregate pattern.

and rate control proposals such as pFabric [12] and D^3 [11]. It targets the root cause of Incast: buffer overflow due to bursty traffic. The switch allocates a number of senders to each flow using an additional field in the TCP or IP header. A maximum congestion window is then calculated at the sending server, such that if each sender sends packets concurrently, the switch buffer still has enough memory to handle all the packets. M21TCP-A is compatible with any other congestion algorithm as the senders only set a maximum that the congestion window must not supersede.

The remainder of this article is structured as follows. Section II introduces the Incast problem and characterize the data centre traffic workload. Section III briefly discusses relevant existing congestion control algorithms for data centres. In Section IV, we introduce the M21TCP-A congestion control algorithm and discuss its properties. Section V presents the performance study of our algorithm with a comparison to existing schemes. Finally, Section VI concludes the paper.

II. The TCP Incast Problem

TCP Incast congestion occurs in barrier synchronized many to one communication patterns like the Partition/aggregate pattern and cluster based storage workloads, where effectively parallel concurrent senders communicate with a client through a bottleneck link. Figure 2 depicts an example of this scenario. Many flows running simultaneously can overwhelm a switch by exhausting the switch memory and leading to severe packet loss.

When Incast occurs, a client may observe a TCP throughput drop of one or two orders of magnitude below its link capacity when packets overfill the buffers on the client port of the switch [7]. So far, there has been no widespread accepted solution to Incast. Individual application solutions such as that discussed in [13] are tedious because they require that each application be built and set up according to their specific needs, and the capabilities of the network. In order to better describe and analyse the Incast problem, it's worth going through an

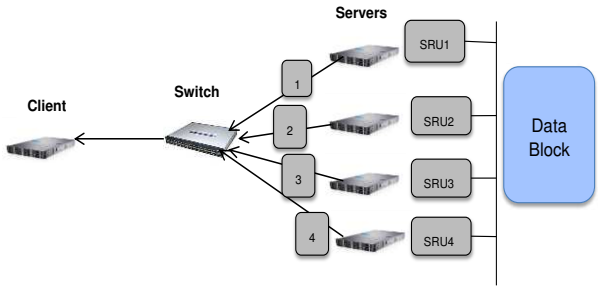

Fig. 2. The classical Incast scenario showing multiple servers communicating with a single client through a bottleneck link .

example of data centre environment with representative traffic workloads as described next.

A. Workload characterization

The workload used in this paper is inspired by distributed cluster based storage systems, and bulk block transfers in batch processing tasks like MapReduce [14]. In these environments, many servers/senders communicate with a single client through a switch (the classic scenario under which Incast occurs). Each server/sender stores a part of a data block usually referred to as a Server Request Unit (SRU). The client requests a data block from the servers by sending request packets to all the servers simultaneously. The servers reply with the SRU, leading to a simultaneous many to one communication pattern. Important still, the client can only request the next block from servers after it receives all the previous data for the block requested. This is called a barrier-synchronized workflow pattern. The workload is a fixed fragment workload, which means that when the number of senders is increased, the size of each SRU remains constant. Therefore, given n servers, if each server sends 256KB of data as shown in Figure 3, the total block size is $n \times SRU$. It should be noted that in some applications that run partition aggregate patterns like that researched in [12] , the client has a deadline and sends all the data received during that deadline to its parent node, whether or not it has received all the data it requested. The effect of Incast in such cases is decreased quality of results. Despite this fact, we believe that the workload presented here will allow a thorough examination of Incast scenarios. The default network parameters used are typical of data centre communications and were obtained from system administrators and switch specifications and are detailed in [2] [3].

B. Incast analysis

Figure 4 shows the result of simulating Incast with the default parameters in Figure 3. This plot is consistent with previously obtained Incast patterns [9] [3] [2] [15]. At low server numbers, the total throughput of the whole system is close to the bandwidth of the bottleneck link i.e. 1Gbps. However the throughput collapses to less than 400Mbps at 16 servers. At greater server numbers, the throughput is even less that 200Mbps. This situation replicates the idea of Incast congestion.

Parameter	Default
SRU size	256KB
Maximum Segment Size	576 bytes
Link Bandwidth	1 Gbps
Link delay	25us
TCP Variant	NewReno
Device Transmit Buffer Size	128KB
Retransmission Time Out (RTO)	200ms
Switch Buffer Size	64KB
Limited Transmit	disabled
Switch Queue	Droptail

Fig. 3. Exemplar network communication scenario parameters.

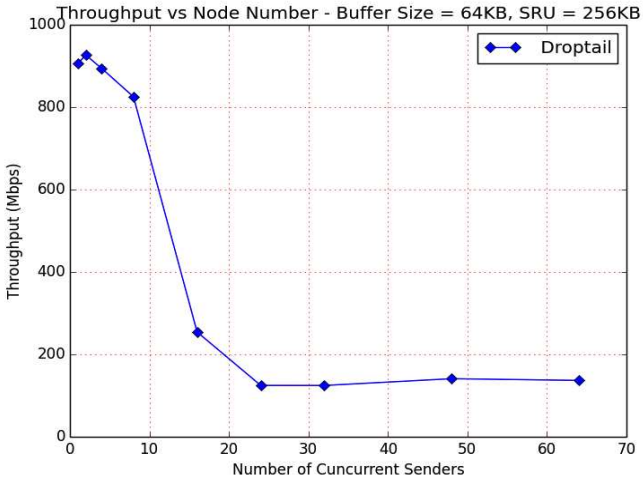

Fig. 4. The total throughput of multiple barrier synchronised connections vs the number of senders, under a fixed block workload.

Figure 5 shows that increasing the buffer size, which is equivalent to increasing the switch's memory, improves the performance of the system and delays the onset of Incast. Approximately, doubling the buffer size, doubles the number of servers at which Incast will set in. One problem with increasing buffer sizes is cost. For instance, The E1200 (1024KB buffer) switch costs $500,000 USD [7].

TCP timeouts are usually responsible for Incast congestion. When one or more servers experiences a timeout as a result of severe packet loss at the queue, the other servers may complete their transfers but do not receive the next request until that timeout(s) expires, and all the servers complete their transfers. Therefore, the bottleneck link remains idle or underutilised for extended periods of time.

III. Existing Congestion Control Algorithms

As stated earlier, while there have been many proposals for Incast and congestion control in data centres, our focus is on the few relevant efforts. TCP's state of the art congestion control algorithms assume that a network is a black box. For congestion avoidance, the end hosts gradually probe

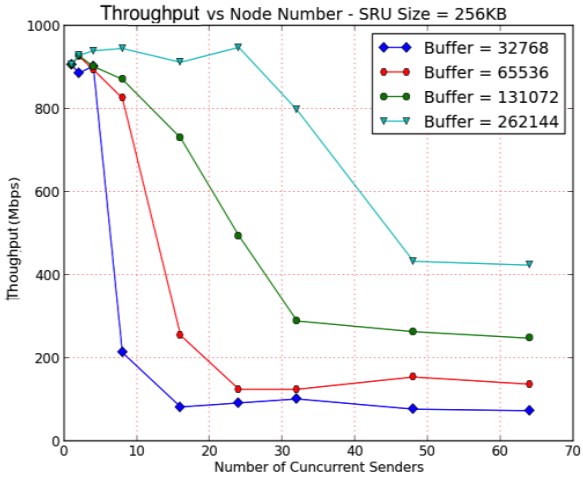

Fig. 5. The total throughput of multiple barrier synchronized connections vs the number of senders, under a fixed block workload.

the network by increasing the congestion window gradually, decreasing it (the feedback) only when a packet is lost. This method of congestion avoidance cannot cope with the unique requirements of data centres that run high throughput, latency sensitive applications. Previous attempts to solve this problem involve Active Queue Management (AQM) at the switch, Explicit Congestion Notification (ECN) and other proprietary congestion control algorithms.

Random Early detection (RED) [16] is an AQM scheme which aims to reduce the number of packets dropped by the router, provide lower delay by slowing queue build-up and make sure that there will always be buffer space available for an incoming packet. The RED algorithm drops packets probabilistically. The probability that the router will drop a packet increases as the estimated average queue size increases i.e. the switch is more likely to drop a packet if its transmit buffer was recently full and less likely to drop packets if it was recently empty. The RED algorithm involves two main functions:

- Estimating the average queue length with the algorithm described in [17].
- Dropping Packets based on the average queue length and two configurable parameters, minth (minimum threshold) and maxth (maximum threshold). No packet is dropped if the queue size is below the minimum threshold while all packets are dropped if the queue size is above the maximum threshold. In between, packet drop decisions are made probabilistically.

Explicit congestion notification refers to a router side action where the switch provides an indication of congestion by marking packets that exceed a threshold, instead of dropping them. ECN is usually used in conjunction with AQM schemes like RED, marking packets anytime RED (or others) would drop them. The IP header has two ECN bits: four code points. Three are set by the sending server to indicate whether or not a flow is ECN capable and the last (CE code point) is set by the switch to indicate congestion. On receipt of a packet with

the CE codepoint set, the receiver encodes an ECN ECHO in the TCP header of the Acknowledgement. The server then responds to the ECHO as it would to a timeout (halves the congestion window) [18].

Data centre TCP (DCTCP) is a congestion control algorithm designed to achieve high burst tolerance and high throughput with commodity shallow buffered switches [9]. It uses ECN and a proprietary marking scheme similar to RED. The key contribution of DCTCP is the act of deriving multibit feedback from the single bit ECN marks. The DCTCP AQM scheme and RED have two main differences:

- DCTCP has only one marking threshold, K. Above this threshold; an arriving packet is marked with the CE codepoint, sending an indication of congestion.
- Marking is based on the current queue length not the average queue length.

Therefore a RED queue can easily be repurposed for DCTCP by setting the maximum and minimum thresholds to the same value and marking packets based on the instantaneous queue length.

IV. The M21TCP Algorithm

It has been established that the main cause of Incast congestion is TCP timeouts caused by severe packet loss at the switch's transmission buffer. This severe packet loss occurs during highly bursty transmissions that overflow the switch buffer. In some next generation data centre congestion control, the routers are actively involved in preventing Incast [19] [11] [12]. M21TCP-A leverages router technology to inform each parallel server of the total number of parallel servers that are communicating through that router. Given that each server has an idea of the router size, they can calculate a maximum congestion window above which, each server cannot send. The maximum congestion window is calculated such that if packets from each sender reach the switch simultaneously, the switch will still not overflow.

Therefore M21TCP-A ensures that TCP senders do not exceed a sending rate limit that could cause a buffer overflow by encoding the number of senders transmitting concurrently in each packet's header. Like ECN, a packet with the encoded information traverses the routers along that path to the receiver. The encoded information is transmitted by the receivers (client) back to the senders through ACK packets. Each router along the path encodes a new value if and only if the value it hopes to set is more than the value encoded in the header. The M21TCP-A algorithm has three main components:

1) **Router/Switch Operation:** A router that supports M21TCP-A operation allocates a number of senders to each flow by counting the number of flows currently traversing the interface. This sender number is encoded in an additional IP or TCP field and is valid for the next RTT.

   In order to properly perform this function, the router must track the number of flows traversing the interface. The M21TCP-A router does this by keeping a list of all the different flows with varying flow parameters in its memory and checking for new flows by comparing incoming packets to the list of already attained flow parameters. If the packet's flow parameters are already contained in the list, the list remains as it is. If the packet's flow parameters are not contained in the list, then they are added to the list. The flow parameters maintained in the list are, similar to the TCP 5-tuples, given below:

   - Source address
   - Destination address
   - Source port
   - Destination port
   - Protocol name

   When multiple switches operate between end hosts, routers may set the number of senders in the packet if and only if the number of senders which that specific router hopes to set is more than that which is already set in the packet. Thus a packet obtained by the receiver contains the maximum flow/sender number value contained in any of the routers on the path the packet traversed.
2) **Receiver Operation:** The M21TCP-A receiver conveys the flow/sender number received in a packet back to the sender by encoding it in the ACK packet. It can either encode the sender number information in an additional IP or TCP header field. In the case of delayed ACKS, the flow/sender value in the latest received packet is used in the ACK. The ACK, which contains the number of senders encoded in an additional TCP header field is sent from the receiver to the sender.
3) **Sender Operation:** The first difference between the normal TCP sender and the M21TCP-A sender is that the M21TCP-A sender must support an additional 32-bit TCP field or an additional IP field (Both TCP and IP fields are used to obtain results in this paper).
   The sender calculates a maximum congestion window using this value with the formula in Equation 1.

$$Max\_Wind = \frac{B - (MHS \times N)}{N} \tag{1}$$

   B is the buffer size. The constant, MHS, is the minimum header size, which represents the combined minimum IP and TCP header size; it usually has a value of 42, N is the number of enders determined by the router. The sender uses the value received to limit its congestion window. The operation does not change the TCP's congestion control algorithm itself. Instead, it simply limits the congestion window by setting a maximum congestion window assignment. Equation 2 depicts an example of this.

$$cwnd = min\{cwnd + 1, maxcwnd\} \tag{2}$$

It is worth noting that M21TCP-A is designed for single path TCP. For topologies with multiple paths between end hosts [20], this relies on Equal Cost MultiPath (ECMP), Valiant Load Balancing (VLB) and other existing mechanisms used with TCP to ensure that a given flow follows a single path. For flows that use multiple paths, additional mechanisms

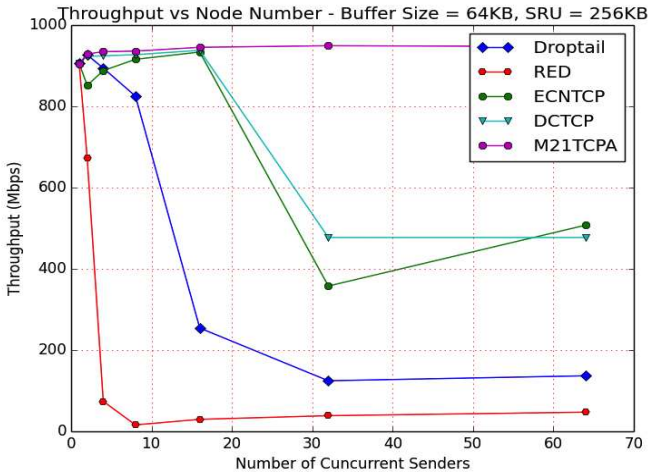

Fig. 6. The total throughput of DROPTAIL, RED, ECNTCP, DCTCP and M21TCPA in the Incast scenario under FFW.

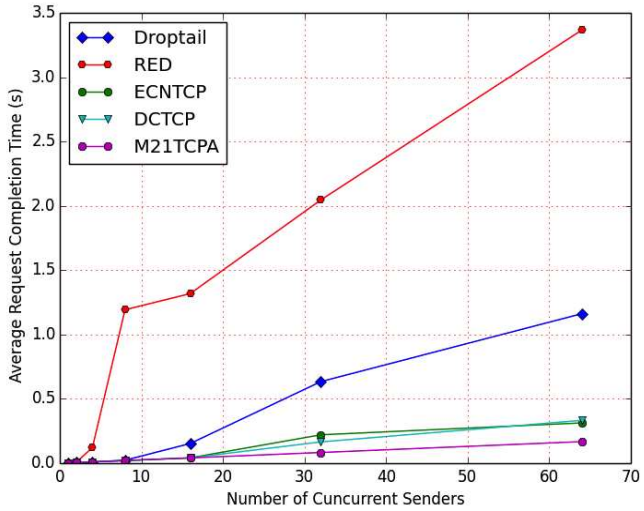

Fig. 7. The Completion time of DROPTAIL, RED, ECNTCP, DCTCP and M21TCPA in the Incast scenario under FFW.

will be required to get M21TCP-A to function properly. However, this hardly seems necessary because the main focus of this paper is the classical Incast scenario, with one switch between the sending servers and the client.

V. Performance Results

The performance of RED, ECN with RED, and DCTCP is evaluated and compared to that M21TCP-A in the classical Incast scenario using the NS-3 simulator. RED is simulated with $min_{th} = 15$ and $max_{th} = 25$. K is set to 20 and g to 0.16, as suggested by Alizedah [9]. Simulations are performed in NS-3 using the classical Incast scenario.

We start by evaluating the performance of Droptail, ECN, RED (ECNTCP) and DCTCP under a fixed fragment workload. The fixed fragment SRU size is 256KB, thus from the fixed fragment workload characterization, the total block size is $n \times SRU$ when the number of severs is n. The two metrics of interest are the throughput and latency of the flows.

Figure 6 shows the throughput of each of the compared solutions. From the plot, RED performs worse than Droptail, ECN performs better than Droptail while DCTCP similar to ECN, delaying the onset of Incast substantially but not eliminating it.

RED is a fair algorithm that is not designed to deal with the uniqueness of highly congested data centre traffic. It is not suited to the Incast scenario because it drops packets prematurely. This means that even at lower traffic volumes, there is a probability that a TCP flow will experience timeouts: fewer senders could cause Incast. The server then responds to the ECHO as it would normally respond to a timeout (halves the congestion window) [18]. In addition, since packet drops occur probabilistically, server timeouts may be staggered such that the total effective timeout from packet drops is much greater than RTOmin. In droptail queues, packets are dropped at close intervals and are more likely to be from the same server. Therefore, timeouts occur close to each other, leading to a lesser effective total timeout than with RED.

ECN and DCTCP show great improvements on Droptail. They both also achieve roughly the same amount of throughput before Incast occurs (circa 940Mbps). Since TCP aggressively drops the window size on receipt of ECN ECHO, some [9] claim that it leads to low link utilisation because of a mismatch between the input rate and the link capacity. The high throughput in Figure 6 shows that this is not the case under fixed fragment DCN workloads. ECN actually causes short flows to complete quickly [21]. Nevertheless, algorithms like RED with ECN that function based on the queue length, find it difficult to deal with situations where there is low statistical multiplexing and the queue length oscillates rapidly [9].This causes queue build-up with little room to absorb microbursts. This is why Incast still occurs at 32 servers with ECN.

Figures 6 and 7 shows that for fixed fragment workloads, DCTCP performs slightly better than ECN: Incast occurs at around 48 servers. By the admission of [9], Incast is the most difficult DCN traffic problem to solve. In their research, DCTCP is found to be ineffective under conditions where the number of senders is large enough such that each of the senders sending around 2 packets exceeds the static buffer size. Thus, even at its best, DCTCP still imposes limits on the number of senders at which Incast will not occur. We observe that M21TCP-A achieves and maintains a high throughput close to 900Mbps, while the throughput of other solutions experience a great collapse at some point.

The latency of M21TCP-A increases gradually with increasing number of senders simply because the block size is greater. Expectedly, M21TCP-A performs much better than other solutions under the workload. There is a slight drop in throughput at 64 senders, but the decline is slight. At lower sender numbers, servers running M21TCP-A maintain a throughput greater or equal to other solutions. M21TCP-A prevents queue oscillations and build up, leading to a consistent, predictable solution which guarantees that the switch's transmission buffer will not overflow and therefore there will be no timeout (the main cause of Incast). The request completion times show that M21TCPA requests have a high probability of completing quicker than other solutions and below the latency targets and

deadlines of partition/aggregate workflow patterns, which can be as low as 10ms.

VI. Conclusion

Incast occurs when many parallel senders communicate with one client through a bottleneck link. It is a catastrophic throughput loss that disrupts the high throughput, low latency applications in data centre networks. In this report, the Incast problem was presented and simulated in the NS-3 simulator. It was validated that the root cause of Incast is buffer overflow at the congested switch, which leads to severe packet loss and consequently, TCP timeouts.

M21TCP-A was proposed and tested against normal TCP with droptail, ECNTCP, and DCTCP. M21TCP-A is a congestion control scheme that informs senders of the number of parallel senders so that senders that they must not exceed, to prevent the switch buffer from overflowing. It was proved to prevent Incast for the maximum number of senders investigated: 64. In general, many to one modifications on the transport layer level offer an opportunity for data centre networks to be emancipated from previous limits on the number of concurrent servers involved in barrier synchronized flows like MapReduce.

References

[1] A. Hammadi and L. Mhamdi, "Review: A survey on architectures and energy efficiency in data center networks," *Comput. Commun.*, vol. 40, pp. 1–21, Mar. 2014.

[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 *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08, 2008, pp. 1–14.

[3] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09, 2009, pp. 73–82.

[4] R. Birke, D. Crisan, K. Barabash, A. Levin, C. DeCusatis, C. Minkenberg, and M. Gusat, "Partition/aggregate in commodity 10g ethernet software-defined networking," in *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, June 2012, pp. 7–14.

[5] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: Survey and practical guide," *Data Min. Knowl. Discov.*, vol. 18, no. 1, pp. 140–181, Feb. 2009.

[6] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 303–314, Aug. 2009.

[7] 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 *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 12:1–12:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364813.1364825

[8] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592604

[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192

[10] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12, 2012, pp. 115–126.

[11] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: meeting deadlines in datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018443

[12] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013, pp. 435–446.

[13] M. Podlesny and C. Williamson, "An application-level solution for the tcp-incast problem in data center networks," in *Proceedings of the IWQoS*, 2011, pp. 23:1–23:3.

[14] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[15] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: Incast congestion control for tcp in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, Apr. 2013.

[16] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the internet," United States, 1998.

[17] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, Aug. 1988.

[18] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," United States, 2001.

[19] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[20] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, Aug. 2008.

[21] F. Kelly, G. Raina, and T. Voice, "Stability and fairness of explicit congestion control with small buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 51–62, Jul. 2008.