M21TCP: Overcoming TCP Incast Congestion in Data Centres

Akintomide Adesanmi Lotfi Mhamdi
School of Electronic & Electrical Engineering
University of Leeds, UK
E-mail: L.Mhamdi@leeds.ac.uk

***Abstract*—Modern data centres host a myriad of cloud services and applications with stringent delay and throughput requirements. The vast majority of these applications are of type partition/aggregate, where many servers simultaneously communicate with one client to produce a result. Unfortunately, the standard TCP/IP protocol, originally proposed for the Internet one-to-one transport, falls short in this environment. This is due to the TCP throughput collapse in such environment, known as TCP Incast congestion problem.**

This paper revisits the Incast congestion problem and identifies its root cause: severe packet drops that result from the switch buffers overflow. We propose a method of controlling congestion, named 'Many-To-one' (M21TCP). The intuition is that a switch can inform all parallel senders of the maximum rate at which they can send packets that will not cause buffers overflow. M21TCP has been tested and evaluated against known previous proposals such as DCTCP, RED and ECN. Results show that M21TCP outperforms previous solutions and completely eliminates Incast for the maximum number of servers.

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

I. Introduction

The venture of large internet service providers such as Amazon, Google, Yahoo and Microsoft into cloud computing, and the consolidation of enterprise IT into data centre hubs, has led to the ubiquitous presence of data centres. These data centres are used for web search, cluster based storage, e-commerce and retail, social networking, map-reduce and other applications that involve large scale computations. Unfortunately a new breed of challenges, specific to the communication networks that support them, accompanies data centre networks (DCNs) [1].

TCP has stood the test of time, consistently adjusting to new environments and technologies over a 40 year history [2]. Therefore, building data centre networks using TCP/IP is both intuitive and desirable because of the low cost, ease of use, and the opportunity it poses to leverage existing technologies [3]. However, TCP was originally designed to operate in Wide Area Networks (WAN) and has been difficult to adapt to the unique workloads, scale and environment of data centres with their high throughput and low latency requirements [4]. DCNs commonly employ barrier synchronised many-to-one traffic patterns [5] that are limited by the slowest sending node [6]. One example is the Partition/Aggregate workflow pattern where a single query operates on data that spans across thousands of servers. Applications with this pattern have soft real time constraints including deadlines on results that translate into latency targets for individual tasks in the workflow. They employ divide and conquer algorithms where parent nodes in the algorithm tree return incomplete responses if children nodes miss their targets. This behaviour is undesirable because incomplete responses affect the quality of query results and diminish revenue [1].

One of the major barriers to the smooth communication in DCNs is a problem termed Incast [7]. Incast is a catastrophic throughput collapse that occurs when the number of servers sending data increases beyond the ability of a switch to buffer packets. It arises as a result of a combination of limited switch buffer sizes, the data centre application communication patterns previously described and the TCP loss recovery mechanism [3].

Simply put, Incast occurs when multiple servers are communicating through a switch and the small buffer is overwhelmed by a concurrent flood of highly bursty traffic from servers that are effectively communicating in parallel. This leads to packet loss and, consequently, one or more TCP timeouts. Timeouts impose hundreds of milliseconds delay, which almost guarantee that a sender misses its deadline. These timeouts and the resulting delay can reduce throughput by 90% or more [2].

Substantial work has been carried to address and solve the Incast problem. These proposals can be mainly divided into two categories. The first is an extension of traditional TCP [8] [2]. 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 [1], D^2CTCP [7], D^3 [9] and pFabric [10]. While these techniques improve the data centre transport capabilities in some aspects, they fail in others, especially with respect to performance and implementation cost.

Motivated by the shortcomings of previous proposals and to completely solve the Incast problem, this paper proposes M21TCP. In particular, it offers the following contributions:

- We introduce M21TCP, a novel approach to solving the Incast problem, where the router allocates a Maximum Congestion Window (MCW) to every flow, for every Round Trip Time (RTT). The protocol leverages the idea of router based flow and rate control proposals such as pFabric [10] and D^3 [9]. It aggressively and explicitly targets the root cause of Incast congestion: buffer overflow. The switch allocates a MCW to each flow using a 32 bit TCP option. This MCW is calculated such

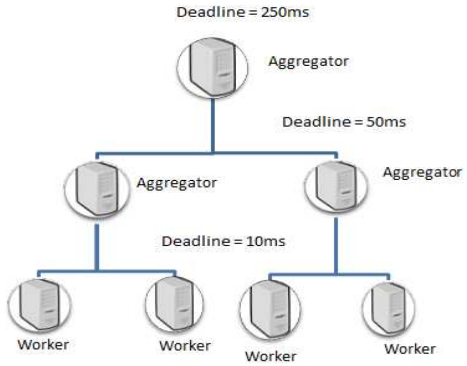

Fig. 1. Partition Aggregate workflow.

that if all senders send packets concurrently, the buffer still does not overflow. M21TCP is compatible with any congestion control algorithm as it simply sets a maximum that the congestion window must not supersede.

- We study the effectiveness of M21TCP against other alternatives including regular TCP variants and AQM schemes, in mitigating Incast. Random Early Detection (RED), TCP with Explicit Congestion Notification (ECN), and Data Centre TCP (DCTCP) are explored. While basic ECN relays the presence of congestion, DCTCP relays the extent of congestion and used this knowledge to size windows effectively [1]. We find that for sufficiently large concurrent senders, RED, ECN and DCTCP are not completely effective at solving the Incast problem. M21TCP on the other hand is shown to prevent Incast congestion for the maximum number of senders.

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

II. Data Centre Communication and Incast Congestion

In this section, we explore the workflow and application patterns that lead to Incast congestion in data centres, describing the constraints that challenge the use of TCP in this environment. Some previous work has explored Incast in cluster based storage systems which involve parallel file reads on data striped across many servers [3] [2]. Other works have explored the Incast problem in Partition/Aggregate workflows [1] [9].

A. Workload characterisation

Researchers who study the Incast problem usually use either of two workloads. The fixed fragment workload (FFW) assumes that the fragment sent by each server remains constant as the number of servers increases, while the fixed block workload (FBW) assumes that the total block size is fixed and partitioned amongst an increasing number of servers. Both workloads are equally valid based on the application. Subsequent Sections use either FFW or FBW or both to explore the Incast and its problems.

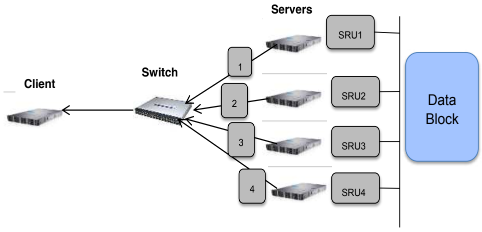

Fig. 2. The classical Incast scenario showing multiple servers communicating with a single client through a bottleneck link.

The general configuration consists of one client connected to multiple servers through the same switch as shown in Figure 2. The client runs an aggregator application that requests a data block from n servers by sending a request packet to each server for one SRU (Server Request Unit) worth of data. The servers respond with the SRU and clients do not request a new block of data until all the individual SRUs that make up the previous block have been received. This leads to a synchronised read pattern of data requests: Barrier Synchronised workflow.

This workload requires a persistent TCP connection, and in simulations, data is repeatedly requested over a 5 second period in order to obtain a more accurate representation of the system. While some researchers [11] make assumptions for non-persistence in the hope that further work be done to improve their solutions for the typical data centre workflows, we will see that this simplification must not be made since persistent connections play a huge role in causing Incast. This is because transfers start with the TCP connection already open as opposed to going through slow start [12].

In order to get more insights to the problem, we simulate the Incast behaviour in the network simulator, NS-3 [13] and analyse the main reasons for its onset: timeouts. We also explore the network and flow characteristics that affect Incast, and detail their effectiveness as possible solutions to Incast. The default network parameters used are typical of data centre communications and were obtained from system administrators and switch specifications and are detailed in [3] and [4]. Therefore, if each server sends $56KB$ of data as shown in Figure 3, the total block size is $n \times SRU$. The total block used in these experiments is $1MB$. The client requests $1MB/n$ Bytes from n different servers, and each server responds with the required amount of data. As usual, the client waits until it receives all the data requested before making another query. This process is repeated for 5 seconds in simulations to obtain a more accurate idea of the average system behaviour.

B. TCP incast congestion

Parallels can be drawn between cluster based storage systems and partition aggregate workflow pattern. One key similarity is the many to one communication pattern where effectively parallel concurrent senders communicate with a

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. Default Network Parameters.

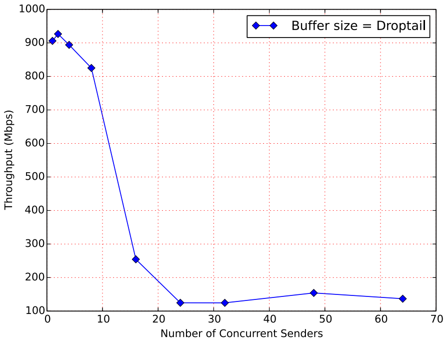

Fig. 4. The total throughput of multiple barrier synchronised connections vs. the number of senders, under a fixed block workload.

client through a bottleneck link. Another is the limitation of the applications by the slowest sending node.

This communication pattern leads to a phenomenon identified and described [14] as Incast. 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, causing many losses.

In partition aggregate patterns, if Incast occurs as a result of severe packet loss at the switch, it could take $RTO\_min = 200ms$ for TCP to recover. This causes the delayed flow to miss the aggregator deadline (usually 10s of milliseconds). In cluster based storage, the delay for reading data increases. There has been no widespread accepted solution to Incast. Individual application solutions such as [11] are tedious because they require that each application be built and set up according to their specific needs, and the capabilities of the network.

Figure 4 shows the result of simulating Incast with the default parameters in Figure 3. This plot is consistent with previously obtained Incast patterns [3] [4] [1] [15]. The many to one communication has a high throughput close to the bandwidth of the bottleneck link -1Gbps - but falls well below 400Mbps at 16 servers.

A closer examination of traces indicates that TCP timeouts are primarily responsible for the Incast phenomenon observed. 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. This leads to a situation where the bottleneck link is idle or underutilised for extended periods of time.

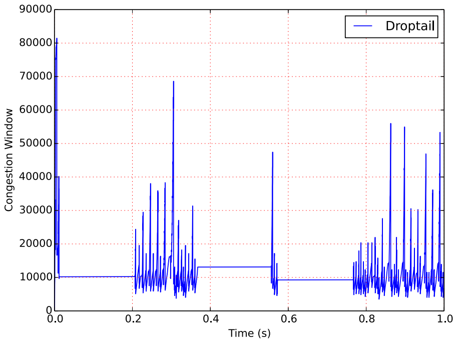

Fig. 5. Congestion window of a single sending server with 7 other sending servers communicating with the client concurrently.

Another reason for the Incast problem is that some TCP connections in data centres are persistent. This means that when a request is completed, the congestion window will continue from where it stopped during the previous connection. Flows that finish later will thus have larger congestion windows, which would be received concurrently on the start of the next request. If a flow loses its whole window - (which can easily happen since the window of each flow becomes smaller as the number of senders increases), timeout can occur. This phenomenon is known as Block Head Time Out [16].

Figure 5 shows the congestion window when the total number of concurrent sending servers is increased to 8. Figure 5 shows three 200ms periods when the congestion window does not change, indicating one of two things:

1) The server times out, or
2) One of the servers parallel to it times out. This server completes transmission but cannot start transmitting the next SRU block because it has not received a request from the client which is waiting for the timed out server(s) to complete its (their) transmission.

In steady state, when a new request is received, the congestion window continues increasing from where it left off from during the previous transmission until a packet is eventually lost due to buffer overflow at the switch. The TCP connection is persistent and does not go into slow start at the beginning of every request; this is central to the Incast problem [10] as flows have no opportunity to slowly probe the network for congestion.

When packet loss is not extreme, the sending server receives triple duplicate ACKS, decreases its congestion window and goes into fast recovery mode. The congestion window spikes up again in fast recovery because the congestion window is increased by one MSS (Maximum Segment Size) for every additional duplicate ACK, until it receives a full ACK. When this happens, the congestion window is decreased to one MSS. The process continues until packet loss is severe. In this case, severe packet loss occurs at 0.37 seconds and a timeout occurs.

Since the root cause of the timeouts that lead to Incast is buffer overflow, it follows that increasing the buffer size of the switch delays the onset of Incast. While it is true that increasing the buffer size mitigates Incast, this characteristic as a potential solution is impractical as switches with large buffer sizes are very costly and not suitable for the considerations of system designers. In addition, switch manufacturers may need to move to faster, more expensive memory in anticipation for higher link capacities in the data centre environment of the future. Therefore, increasing buffer sizes, as a solution, is not cost ineffective.

III. The Many-To-One Scheme

A. Design Rationale

The goal is to develop a method of mitigating Incast congestion that is more effective in DCNs than existing solutions. Previous Sections examined proposed solutions that involved reducing the impact of timeouts, using ECN and implementing clever congestion control (DCTCP) algorithms. However, all these methods have their shortcomings. Following previous discussions, the following goals for data centre congestion control are identified as:

- Maximum application throughput
- Low latency
- High link utilisation

Furthermore, in the technologies detailed in [17] [10], routers are involved in the control of flows at the packet level. In D^3, routers explicitly assign sending rates to end hosts based on their deadlines while in [10], routers prioritise packets based on information set in their headers. The Many To One transport layer modifications for Incast avoidance combine the router rate allocation concept of D^3 and the realisation that lower MTUs mitigate Incast.

The central theme to the M21TCP concept is that the switch can determine the number of flows passing through it and send a message through the packets back to the sender informing them of either the number of flows parallel to each sender or the maximum number of packets that they can each send at a go that will not overflow the switch's buffers. The senders use this information to set a maximum transmission rate parameter (congestion window for TCP) that must not be exceeded.

B. The Many to one TCP M21TCP

The Many-to-One TCP (M21TCP) ensures that TCP senders do not exceed a sending rate limit that could cause a buffer overflow. The routers encode a maximum congestion window each sender must not exceed in each packets header.

Like ECN, a packet with the encoded information traverses the routers along that path to the receiver. The encoded information is then transmitted by the receivers to the senders through ACK packets. If the encoded value is the Maximum congestion window, each router along the path encodes a new value if and only if the value it hopes to set is less than the value already encoded in the packet. If the packet is encoded with the number of senders, routers set a value if and only if the value it hopes to set is more than the value already encoded in the header.

C. The M21TCP Algorithm

The M21TCP algorithm has three main components:

1) **Router/switch operation:** A router that supports M21TCP operation allocates a MCW to each flow based on the number of flows currently traversing the interface. This MCW is encoded in a TCP option field and is valid for the next RTT. In order to properly perform this function, the router must track the following parameters:
   - N**:** The number of flows traversing the interface. Routers use flow initiation and termination packets (TCP SYN/FIN) to increment and decrement N respectively..
   - **max_cwnd:** The MCW for each flow, which will allow maximum utilization of the link while preventing queue build-up and loss due to buffer overflow. In an advanced system, extensive mathematical analyses should be done to obtain a formula for this parameter. For these purposes however, a simple effective setting is derived by assuming a worst case scenario where there is bursty traffic from all concurrent senders. The MCW is derived from N using 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. When M21TCP is used in a situation where the length of IP and TCP headers are not the minimum value, it is the responsibility of the sender to reduce the congestion window by the number of bytes used by the IP and TCP options.

   In pFabric [10], routers capture packet metadata, while in D^3, routers encode rates (in a similar manner to M21TCP) in packet headers. Regular routers are capable of capturing and modifying TCP SYN packets, which are used to negotiate the TCP maximum segment size during the TCP handshake [18]. Therefore it must follow that routers like those proposed for D^3 and pFabric are easily capable of M21TCP's operation while regular routers can be adapted without extensive modifications. When multiple switches operate between end hosts, routers may set the MCW option in a TCP packet if and only if the maximum congestion window value, which that specific router hopes to set is less than that which is already set in the packet. Thus a packet obtained by the receiver contains the least MCW value calculated by any of the routers on the path the packet traversed.
2) **Receiver operation:** The M21TCP receiver is not unlike an ECN receiver. It conveys the MCW received in a packet back to the sender by encoding it in the ACK packet. In the case of delayed ACKS, the MCW value in the latest received packet is used in the ACK.
3) **Sender operation:** The first difference between the normal TCP sender and the M21TCP sender is that the M21TCP sender always sends packets with a TCP

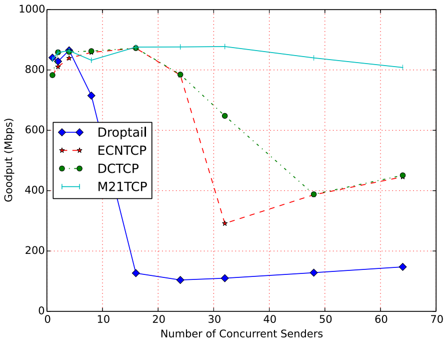

Fig. 6. The total Goodput vs number of senders of DROPTAIL, ECNTCP, DCTCP, M21TCP in the incast scenario under FFW.

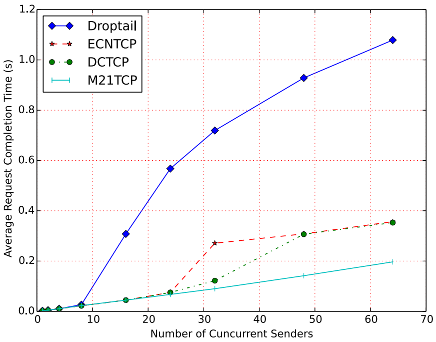

Fig. 7. Latency of DROPTAIL, ECNTCP, DCTCP, M21TCP vs. number of senders in the Incast scenario under FFW workload.

MCW option field: a proprietary 32 bit TCP option. The sender uses the MCW value received to limit its congestion window. The operation does not change TCPs congestion control algorithm itself. It simply limits the congestion window by setting a maximum congestion window assignment. Equation 2 is a simple example of how this works

$$cwnd = min\{cwnd + 1, maxcwnd\} \qquad (2)$$

It is worth mentioning that, as described in Section III-A, the implementation cost of M21TCP is just similar to other router based flow and rate control proposals, such as pFabric [10] and D^3 [9]. In what follows, we shall show the superior performance of M21TCP in overcoming the Incast problem.

IV. Experimental Results

The performance of M21TCP is evaluated and compared to that of RED, ECN with RED, and DCTCP 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 as suggested by [1].

A. Throughput and latency under fixed fragment workload

We start by evaluating the performance of M21TCP against ECN with RED (ECNTCP) and DCTCP under a fixed fragment workload. The fixed fragment SRU size is 256KB, which means that 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 goodput of each solution while Figure 7 shows the average completion time or latency of a request under a fixed fragment workload. We observe that M21TCP achieves and maintains a high throughput close to the maximum with increasing sending servers. We also observe that the latency of M21TCP increases gradually with increasing number of senders simply because the block size is greater. We further observe that RED performs worse than Droptail, while ECN and DCTCP delay the onset of Incast substantially but do not completely eliminate it.

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, researchers [1] 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 [19].

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 [1]. This causes queue build-up with little room to absorb microbursts. This is why Incast still occurs at 32 servers with ECN.

As depicted in Figure 6, DCTCP performs slightly better than ECN: Incast occurs at around 48 servers. In [1], 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. In this experiment, 48 senders are enough to cause Incast. The system suffers timeouts when the number of senders is such that each sending around 3 packets ($48 \times 57 \times 3 > 64KB$) is enough to exceed the static buffer size. These results match the results obtained in [1].

Expectedly, M21TCP performs much better than other solutions under a fixed fragment workload. There is a slight drop in goodput at 64 senders, but the decline is slight. At lower sender numbers, servers running M21TCP maintain a goodput greater or equal to other solutions. M21TCP prevents queue oscillations and build up, leading to a consistent, predictable solution which guarantees that the switchs transmission buffer will not overflow and therefore there will be no timeout (the main cause of Incast).

B. Throughput and latency under fixed block workload

RED with ECN and DCTCP are compared with droptail under a fixed block workload. Figure 8 shows the application level goodput when the block size is fixed at 1MB and each sender is required to transmit $1MB/n$ Bytes. Figure 9 shows the average request latency. Beyond 32 senders, partition

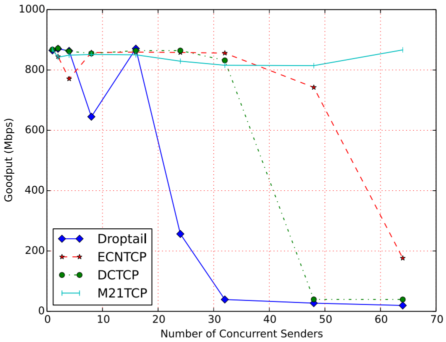

Fig. 8. The total Goodput vs number of senders of DROPTAIL, ECNTCP, DCTCP, M21TCP in the incast scenario under FBW.

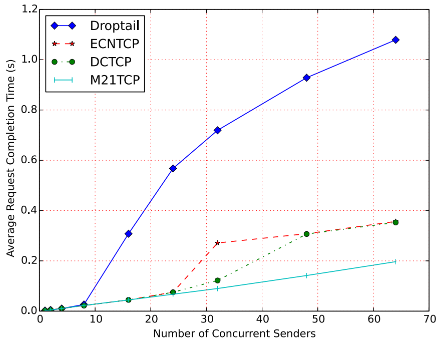

Fig. 9. Latency of DROPTAIL, ECNTCP, DCTCP, M21TCP vs. number of senders in the Incast scenario under FFW workload.

aggregate requests that have deadlines of 10ms will miss their deadlines with DCTCP. However ECN maintains a completion time around 10ms until a little more than 48 senders transmit in parallel. Similarly, the throughput of DCTCP under FBS drops at 48 senders, while the throughput of RED with ECN drops at 64 senders: Unexpectedly, ECN performs better than DCTCP under fixed block workloads.

While DCTCP provides significant performance improvements in delaying Incast, this suggests that sometimes it performs worse than currently deployed TCP congestion control schemes. This is because as the number of flows grows, the bottleneck queue gradually oscillates with increasing amplitude, thereby not meeting design goals. DCTCPs failure to inform the sending servers of changing congestion windows fast enough is what causes queue oscillations. Figure 8 shows that M21TCP maintains a high goodput and even trends upward as the number of servers increases beyond 64. Figure 9 shows that latency sensitive applications that run M21TCP should expect to meet 10ms deadlines even at a high number of sending servers. The results again validate expectations that M21TCP not only achieves higher throughput and lower delay than DCTCP and ECNTCP, but completely prevents Incast congestion at the switch.

V. 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 DCNs. In this paper, the Incast problem was investigated and analysed. In particular, a new congestion mechanism, M21TCP, was proposed and tested against normal TCP with Droptail, ECNTCP, and DCTCP. M21TCP is a congestion control scheme that informs senders of the Maximum Congestion Window that they must not exceed, to prevent the switch buffer from overflowing. It proved to prevent Incast for the maximum number of senders investigated (64) and outperformed all previously proposed solutions. In general, many to one modifications on the transport layer level offer an opportunity for DCNs to be emancipated from previous limits on the number of concurrent servers involved in barrier synchronised flows like MapReduce.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010, pp. 63–74.

[2] 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.

[3] 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 *USENIX Conference*, ser. FAST'08, 2008, pp. 1–14.

[4] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *ACM WREN*, 2009, pp. 73–82.

[5] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM SIGCOMM*, ser. IMC'09, 2009, pp. 202–208.

[6] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Fail-stutter fault tolerance," in *HOTOS'01*, 2001, pp. 33–.

[7] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *ACM SIGCOMM*, 2012, pp. 115–126.

[8] 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 *USENIX Conference*, ser. FAST'08, 2008, pp. 12:1–12:14.

[9] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: meeting deadlines in datacenter networks," in *ACM SIGCOMM conference*, ser. SIGCOMM'11, 2011, pp. 50–61.

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

[11] M. Podlesny and C. Williamson, "An application-level solution for the tcp-incast problem in data center networks," in *IWQoS'11*, pp. 1–3.

[12] M. Alizadeh and T. Edsall, "On the data path performance of leaf-spine datacenter fabrics," in *IEEE HOTI*, 2013, pp. 71–74.

[13] (2011) The ns-3 consortium. ns-3, 2011. [Online]. Available: http://www.nsnam.org/9

[14] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *ACM/IEEE SC '04*, 2004, pp. 53–.

[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] J. Zhang, F. Ren, L. Tang, and C. Lin, "Modeling and solving tcp incast problem in data center networks," in *IEEE TPDS*, vol. PP, no. 99, 2014.

[17] 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.

[18] P. Zhang, H. Wang, and S. Cheng, "Shrinking mtu to mitigate tcp incast throughput collapse in data center networks," in *ICCMC'11*, pp. 126–129.

[19] 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.