ISSN: 2321-8169
2791 – 2795

A Method Based on Data Fragmentation to Increase the Performance of ICTCP During Incast Congestion in Networks

Sneha Sebastian
P G Scholar, Dept. of Computer Science and Engg.
Amal Jyothi College of Engg.
Kottayam, India
sneha.seb89@gmail.com

Neethu C Sekhar
Asst. Professor, Dept. of Computer Science and Engg.
Amal Jyothi College of Engg.
Kottayam, India
neethucsekhar@amaljyothi.ac.in

Abstract— Transport Control Protocol (TCP) is a reliable protocol of the transport layer in networks. However TCP causes incast congestion in high bandwidth and low latency networks. This happens when multiple servers send data to the same receiver in parallel. This many-to-one traffic pattern is common in many important data center applications sch as MapReduce. Data centers provide large volumes of computation and storage resources for supporting today's Internet services. Hence TCP incast congestion may severely degrade their performances, e.g., by increasing response time and reducing throughput. In this paper, the TCP incast is studied in detail. The existing Incast Congestion Control for TCP (ICTCP) to handle the problem of incast congestion involves modification only at the receiver side. However ICTCP has limitations like reduced throughput and increased delay both of which can be improved appropriately. These limitations are overcome using data fragmentation in ICTCP that has an improved performance than the exixting ICTCP in incast situations.

Keywords*-Incast, data centers, ICTCP, throughput, latency*

*****

I. INTRODUCTION

The Transmission Control Protocol (TCP) is one of the important protocols of the Internet protocol suite (IP), used for digital communication. It is a connection oriented protocol and provides a reliable, ordered and error checked stream of data. TCP protocol has several standard mechanisms for congestion control as well. However, TCP does not work well for many-to-one type of data traffic found in high-bandwidth, low latency networks. TCP in such scenarios may sometimes encounter the problem of incast congestion[1]. Incast is a many-to-one communication pattern usually associated with cloud data centers. These usually implement computing and distributed storage frameworks such as Hadoop, MapReduce, etc. which are used for different applications such as web search engines, social networks, maps, data warehousing and analytics. For example, there can be a parent node that requests 40KB of data across 20 Nodes. Each node that receives this request responds simultaneously with 2KB of data. This is just two packets from each responding node. In real world situations, the parent node could in fact, be a database server that may request an 80KB photo of a friend added to a social network.

Incast usually happens when a server node sends a request for data to a group of nodes all of which receive the request from the server simultaneously. The group of nodes may all synchronously respond to the server. This results in a burst of many machine sending TCP streams simultaneously to one machine (many to one). This kind of communication may lead to a congestion called incast congestion which occurs as a result of overflow at the switch buffer during many to one communication. The buffer overflow occurs as a result of several simultaneous flows to the switch which may not have sufficient buffer space.

This paper identifies the existing ICTCP as a suitable method for controlling incast congestion but finds that the performance of ICTCP is not sufficiently achieved. A method based on data fragmentation in ICTCP during incast (DF-IC) has thus been proposed in this paper to improve the performance of ICTCP. The paper is organized into the following sections. Section II consists of the background, section III consists of the problem definition, section IV consists of the related works, section V consists of the proposed systems, section VI consists of the results and analysis, section VII consists of the conclusion.

II. BACKGROUND

Incast congestion occurs when many synchronized servers under the same switch simultaneously send data to one receiver in parallel. The performance collapse of these many-to-one TCP connections is called TCP incast congestion. In the real communication environment, data blocks are striped over a number of servers, such that each server stores a fragment of a data block, denoted as a Server Request Unit (SRU)[2]. A client that requests a block of data sends request packets to all the storage servers that contain data for that specific block; the client will request the next block of data only after it has received all data for the current block. These pattern of reads are known as synchronized reads. The TCP incast problem is caused by the many-to-one synchronized traffic pattern, large latency of online queries, TCP performance deterioration in virtualized data centers, and malicious users in multi-tenant data centers. The root cause of TCP incast collapse is that the many- to-one traffic of multiple

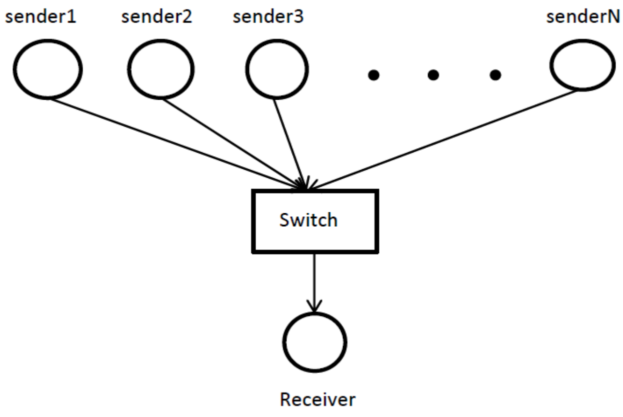

Fig 1. Incast Scenario

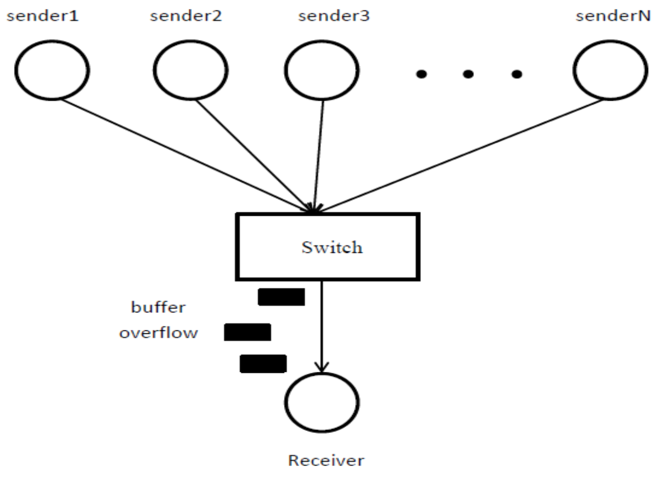

Fig 2. Incast Scenario Switch Buffer Loss

TCP connections overflow the Ethernet switch buffer in a short interval, causing a large number of packet loss and thus TCP retransmission and timeouts. A solution that modifies only the TCP receiver is hence preferred to solutions that require hardware support and modifications on both the TCP receiver and sender sides.

The incast scenario is depicted in the Fig 1 above. In this, several sender nodes sender1, sender2, …, senderN transmit data to a single Receiver through a switch simultaneously. As a result, there is overflow at the switch due to shortage of buffer space as shown in Fig 2. The TCP incast problem widely exists in today's data centers:

- Found in distributed storage systems. Numerous data are stored in many distributed nodes, such as BigTable or HBase[3]. When a client retrieves data, parallel access to some of these distributed nodes is needed.
- Found in data-intensive scalable computation systems, such as MapReduce[4]. These systems quickly deal with large amounts of data by parallel processing across many servers. Thus, all-to-all or many-to-one transmissions are needed to transfer data between sender and receiver nodes. Data-intensive applications have been increasing in a variety of fields, including web-search engines, social networks, and e-commerce.
- Found in partition/aggregation workflows. In most large-scale web applications, every requested task is broken into small pieces. The workers in the layer below it are assigned these pieces. Then the responses from multiple workers are transmitted to the aggregator and generated into the final result. This partition/aggregation design naturally leads to a many-to-one communication pattern.

The idea is to perform incast congestion avoidance at the receiver side by preventing incast congestion. The receiver side can adjust the receive window size of each TCP connection, so the aggregate traffic of all the synchronized senders are kept under control. The design is called Incast congestion Control for TCP (ICTCP)[1]. However, controlling the receive window adequately is quite challenging. The receive window should be small enough to avoid incast congestion, but also large enough for good performance and other nonincast cases. These challenges are addressed with a systematically designed ICTCP. But even in ICTCP, there may be packet loss as the overflow at the switch buffer is not controlled fully. As a result, there is reduced throughput. Thus, in this paper, DF-IC (Data Fragmentation during Incast) has been proposed which is a novel method to improve the performance of ICTCP during incast congestion.

III. PROBLEM DEFINITION

Incast congestion occurs when multiple servers under the same switch send data to one receiver simultaneously. In each connection, the amount of data transmitted is relatively small, e.g, 64 kB. TCP throughput is severely degraded by incast congestion as the TCP connections can experience timeouts caused by drop of packets. The TCP incast scenario is common for data-center applications. As an example, for search indexing, counting the frequency of a specific word in a number of documents is needed. This job is distributed to several servers, and each server machine is held responsible for some documents on its local disk. Only after all servers return their counts to the receiving server can the final result be generated. MapReduce is another application where TCP incast congestion occurs. TCP flow control is enforced with the help of TCP receiver window, i.e., preventing a faster sender from overflowing a slow receiver's buffer. A static buffer may work for a specific application in a static network, but not for a data center with fairly diverse traffic and application requirements. For a changing number of connections, a static buffer cannot work and cannot handle the dynamics of the applications' requirements. The TCP receive window has the ability to control TCP throughput and so it can be adjusted dynamically to the proper value. Incast congestion happens when the switch buffer overflows. The benefit of an incast congestion control scheme at the receiver side is that the receiver knows how much available bandwidth remains and the throughput it has achieved. Thus a transport layer solution is required and a solution that requires only a change at the receiver side is preferred. ICTCP has been found to be such a solution. The problem with the existing ICTCP is that it does not address the problem of switch buffer overflow completely. In this paper, DF-IC has been proposed which can increase the performance of ICTCP in incast case.

IV. RELATED WORKS

There are several methods for handling the incast congestion problem of TCP. Some of these are given below. Data Center TCP (DCTCP)[5] is a method used to detect incast scenario. It is a TCP-like protocol for data center networks. DCTCP uses Explicit Congestion Notification (ECN), a feature already available in modern commodity switches. DCTCP is a TCP-like protocol for data center networks. It uses Explicit Congestion Notification (ECN) in the network to provide multi-bit feedback to the end hosts. ECN is a feature that is already available in modern switches. DCTCP uses a simple marking scheme at the switches which sets the Congestion Experienced (CE) codepoint of packets as soon as the occupancy of the buffer exceeds a fixed threshold value. The DCTCP algorithm has three main components:

- Simple Marking Method at the Switch: DCTCP uses a very simple active queue management mechanism. There is just a single parameter, which is the marking threshold, K. A packet that is arriving is marked with the CE codepoint if the occupancy of the queue is greater than K upon arriving. Otherwise, the packet is not marked.
- ECN-Echo at Receiver: The difference between a TCP receiver and a DCTCP receiver is the way information in the CE codepoints is sent back to the sender.
- Controller at the Sender Node.

DCTCP, hence, combines Explicit Congestion Notification (ECN) with a novel control mechanism at the source nodes. Both the TCP sender and TCP receiver are modified slightly for a novel congestion window adjustment. Reduced occupation of the switch buffer can effectively mitigate potential overflow of incast in more general cases, i.e., not only the last hop. The experimental results conducted using DCTCP show that DCTCP outperforms TCP for TCP incast, but eventually, it converges to an equivalent performance when the incast degree increases[6], ie., for over 35 senders. The difference between ICTCP and DCTCP is that ICTCP only modifies the TCP receiver whereas DCTCP modifies both TCP sender and receiver.

QCN is Quantized Congestion Notification [7]. QCN is being developed as an optional standard for Ethernet. QCN requires hardware rate limiters (implemented on the NICs). This adds to hardware complexity, and hence increases server cost. To reduce cost, QCN rate limiters must lump flows into (a single or few) flow sets, which then share fate, leading to collateral damage to flows that do not share bottleneck links with congested flows. Moreover, QCN cannot cross layer-3 boundaries, which abound in many data centers today. The IEEE 802.1 QAU Congestion Notification is a standard that specifies protocols, procedures, and objectives for congestion management in low bandwidth delay product networks. The switches used are able to send congestion information to the sources of traffic, which can rate the limit of their flows to avoid frame loss. Currently, the standard being examined the most is QCN or Quantized Congestion Notification. While other groups of nodes also use various algorithms to control congestion, QCN provides congestion control for data centers where the IP protocol such as UDP and TCP are running. The two main components of QCN are: the Congestion Point (CP) and the Reaction Point (RP). When data is transmitted from a source node to the destination node, the intermediate switches might become congested, and hence, these are called congestion points. A feedback value is calculated by the CP based on the state of the queue, and if the feedback computed is negative (which implies congestion), this feedback is sent to the source of the sampled packet. The feedback is quantized into a 6 bit value (between 0 and 63), and hence it is called Quantized Congestion Notification. QCN requires hardware rate limiters (implemented on the NICs). This adds to hardware complexity, and hence increases server cost. For QCN, the mechanism should be implemented at the switch which requires a lot of money and hardware support.

Cluster-based storage systems depend on standard TCP/IP protocol for access of the client to data[8]. Unfortunately, when the data is split over several networked nodes, a client can experience a TCP throughput collapse that results in much lower bandwidth than that to be provided by the available network links which is basically incast congestion. TCP Reno, TCP NewReno and TCPSACK were developed to improve the performance of TCP in different network scenarios. The TCP NewReno is an improved version of TCP Reno, whereas TCPSACK uses the method of selective acknowledgements. Both TCP NewReno and TCP SACK outperform TCP Reno. The TCP NewReno offers up to an order of magnitude better performance when compared to TCP Reno. Unfortunately, none of these implementations of TCP can eliminate the large penalty to throughput that is caused by incast congestion.

ICTCP (which is Incast Congestion Control for TCP)[1] provides a receive-window-based incast congestion control algorithm for TCP at the end-system. The receive windows of all the TCP connections are jointly adjusted to control incast congestion. Incast congestion happens when multiple sending servers under the same switch send data to one receiver server. TCP throughput is severely degraded by incast congestion since the TCP connections can experience packet drops. In ICTCP, the receive window size of each TCP connection can be adjusted by the receiver side, so the bursty data of all the synchronized senders are kept under control. Incast Congestion Control for TCP or ICTCP on the other hand requires no hardware support at the switch. It requires modifications only at the TCP receiver side. It works as normal TCP protocol in nonincast cases. Hence it is relatively inexpensive. It works by dynamically adjusting the receive window of all incast TCP connections. Due to the inexpensiveness and easier implementation of ICTCP, it has been identified as a suitable method to handle incast scenarios. But even in ICTCP, it has been found that there is decrease in throughput and increased latency and the problem of incast congestion, is therefore not addressed fully. A solution is thus, required to address this which has been proposed in this paper.

V. PROPOSED SYSTEM

Due to the limited performance of ICTCP, some method has to be introduced to improve its performance. A method based on reducing the size of the transmitted packets in incast situations has been proposed here. The TCP incast congestion usually happens when several synchronized servers send data to the same receiver simultaneously. As mentioned before, there are several methods for handling the incast congestion problem of TCP. ICTCP (which is Incast Congestion Control for TCP) [1] uses a receive window based incast congestion control method for TCP at the end-system. Incast congestion happens when multiple sending servers under the same switch send data to one receiver server. TCP throughput is severely degraded by incast congestion since the TCP connections can experience packet drops. In ICTCP, the receive windows of all the TCP connections are jointly adjusted to control incast congestion. The receive window size of each TCP connection can be adjusted by the receiver side, so the bursty data of all the synchronized senders are kept under control.

As mentioned earlier, the drawback of ICTCP is that high throughput performance is not achieved. There is still overflow at the switch buffer due to which there is only limited throughput. Also, the average end to end delay is also high. As a result, a better method is needed to improve the performance of ICTCP. The proposed system is Data Fragmentation during Incast (DF-IC) to improve the performance of ICTCP. This method addresses the problem of switch buffer overflow that is still found in ICTCP and therefore helps in increasing the throughput of ICTCP. In the proposed method, the switch buffer overflow of the existing method of ICTCP is controlled to an extent by reducing the size of the data packets transmitted by the sender nodes. When data loss is observed at the switch buffer, the size of the data packets are reduced by the sender nodes by observing the roundtrip time and also by the pending acknowledgements. In DF-IC, the sender thus sends packets that are reduced in size upon packet loss at the switch buffer. The efficiency of the protocol is now improved and can give better performance than the existing ICTCP during incast scenario.

The proposed system can be depicted by the block diagram shown in the Fig 3 below. In non incast cases, where there is no problem of incast congestion at the switch buffer, the protocol works as the normal TCP protocol. On the other hand, whenever there is many to one communication causing incast congestion due to limited space at the switch buffer, the protocol works as DF-IC. In the flowchart given in the Fig 4, the mechanism of the proposed system is given. In this, the size of the packet is halved when there are pending acknowledgements to be received at the sender. Otherwise, the window size is decremented by 1. Pending acknowledgements indicate the loss of data at the switch due to limited space in the buffer. Hence, this parameter has been taken as an indicator to reduce the size of the packets. As a result, better performance is achieved by this method, both in the case of throughput and end to end delay than the existing ICTCP in incast situations. These have been studied and analyzed.

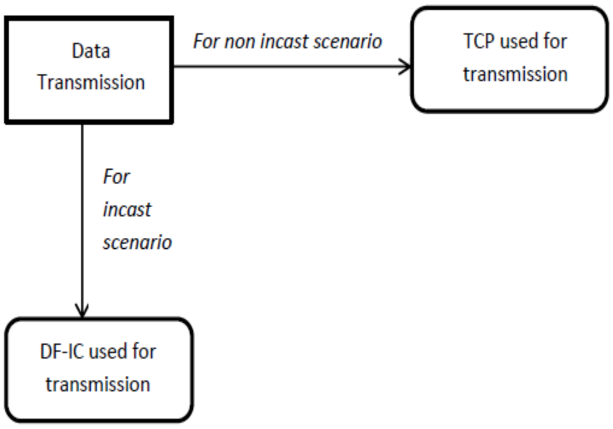

Fig 3. Proposed system in incast and non incast scenarios

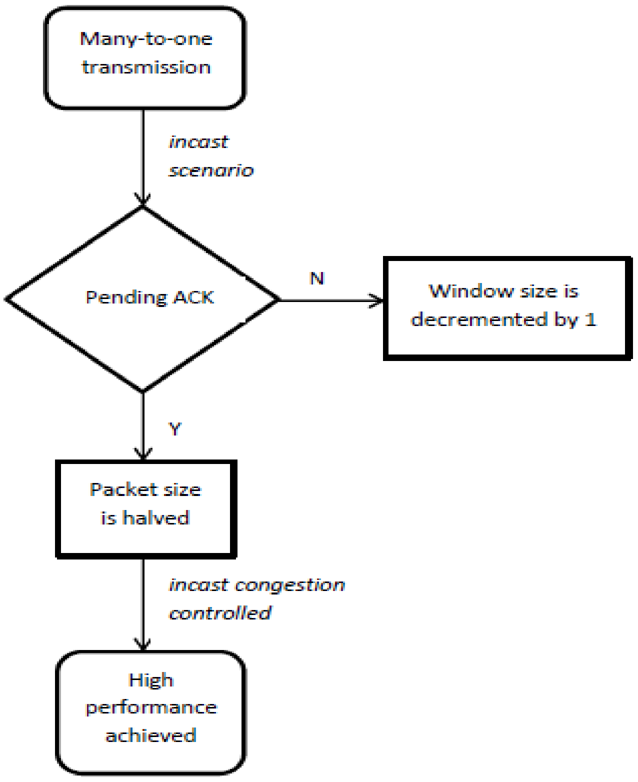
Many-to-one transmission
incast scenario
Pending ACK
N
Window size is decremented by 1
Y
Packet size is halved
incast congestion controlled
High performance achieved

Fig 4. Proposed system in detail

VI. RESULT AND ANALYSIS

Average delay is the time taken by the data packets to reach the destination. This is the time from the generation of the packet by the sender up to their reception at the destination's application layer and is expressed in seconds.

Throughput or network throughput is the rate of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second (bits/s or bps) or bytes per second, and sometimes in data packets per second or data packets per time slot.

The DF-IC was implemented and simulations were carried out on different number of sender and receiver nodes. The following results were obtained. The impact of TCP, ICTCP, and the proposed DF-IC on the throughput were found to be as shown below in Fig 4. from which it is clear that DF-IC gives improved throughput than ICTCP or TCP.

The average end to end delay in the case of TCP, ICTCP, and DF-IC for one of the different simulation scenarios has been tabulated below in Table I. It has been found that average end to end delay in the case of DF-IC and ICTCP is almost the same. Hence, from these two measures, the performance of DF-IC has been found to be higher than that of TCP and ICTCP.

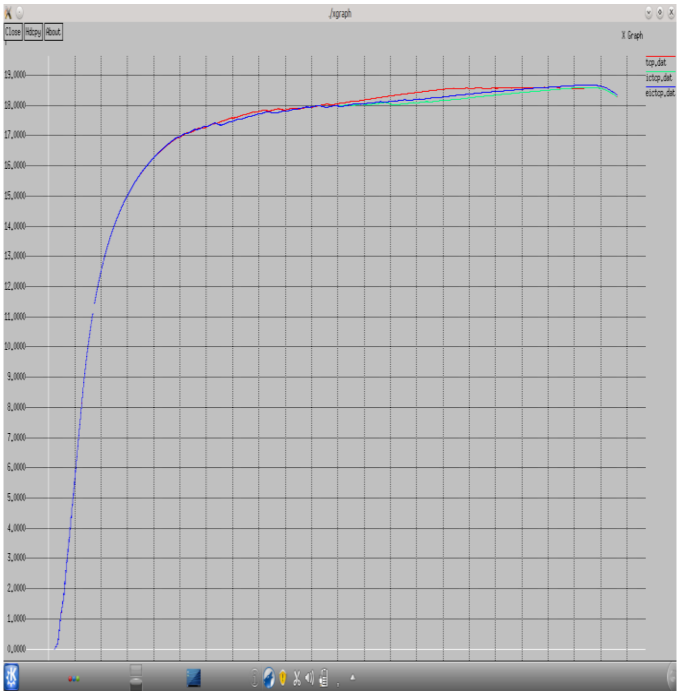

Fig 4. Impact of TCP, ICTCP and DF-IC on throughput

TABLE I AVERAGE END TO END DELAY FOR A SCENARIO

TCP(ms)	ICTCP(ms)	DF-IC(ms)
12.79	12.8302	12.966

VII. CONCLUSION

TCP has the problem of incast congestion for data center networks. Incast congestion occurs at the switch when several senders send data simultaneously to a single receiver all under the same switch. Previous methods that address the problem of incast congestion use modifications at the sender and receiver side. In this paper, the design, implementation, and evaluation of ICTCP has been presented to improve TCP performance for TCP incast in data-center networks. The focus is on a receiver based congestion control algorithm to prevent packet loss. ICTCP adaptively adjusts the TCP receive window based on the ratio of the difference of achieved and expected per connection throughputs over expected throughput, as well as the last-hop available bandwidth to the receiver. In ICTCP, modification is done only at the receiver side and also it does not require modification of TCP header. ICTCP works as TCP in non incast scenario. However, ICTCP has certain limitations. It has limited performance in the case of throughput and delay. Therefore, DF-IC has been proposed in this paper that has improved the performance of ICTCP. The experimental results demonstrate that DF-IC has better throughput than ICTCP or TCP.

[1] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang, "ICTCP: Incast Congestion Control for TCP in Data-Center Networks", Net- working, IEEE/ACM Transactions, April 2013, Volume:21, Issue: 2.

[2] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, Srinivasan Seshan,"On Application-level Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems", Proceedings of the 2nd international workshop on Petascale data storage, 2007.

[3] Jiao Zhang, Fengyuan Ren, and Chuang Lin, Tsinghua University,"Survey on Transport Control in Data Center Networks", Network, IEEE, 2013, Volume:27, Issue: 4, p. 22 - 26 .

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. OSDI, 2004, p. 10.

[5] Mohammad Alizadehzy, Albert Greenbergy, David A. Maltzy, Jitendra Padhyey, Parveen Pately, Balaji Prabhakarz, Sudipta Senguptay, Murari Srid- harany, "Data Center TCP:DCTCP", Proceedings of the ACM SIGCOMM 2010 Conference, p. 63-74

[6] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", Proceedings of the ACM SIGMETRICS joint international conference on Measurement and Modeling of computer systems, Pages 73-84

[7] Prajjwal Devkota, A. L. Narasimha Reddy,"Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers", Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium, p. 235 - 243.

[8] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, Srinivasan Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems",

[9] Proceedings of the 6th USENIX Conference on File and Storage Technologies, 2008, Article No. 12