iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 3 **Issue:** VII **Month of publication:** July 2015

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Mitigating Tcp Incast With Fec For Many To One Communication In Data Centers

Sreerekha K[1], Kiran V K[2]
[1,2]Dept. of Computer Science and Engineering,
NSS College of Engineering, Palakkad, Kerala

Abstract— The Transmission Control Protocol (TCP) is an important transport layer protocol which provides reliable data delivery. In this scientific world, number of users of internet and flow of data increases day by day making reliable data transmission difficult. As a result, in many to one communications there arises a problem known as TCP Incast which causes a severe drop in throughput. This arises due to the overflow of output buffer of switch and TCP's low Retransmission Timeout as compared to Round Trip Time. Earlier solutions included controlling switch buffer or updating OS/hardware. In this paper, a new transport protocol is proposed by improving the UDP-based transmission with a simplified FEC scheme and Walsh code. Through ns2 based simulations it is observed that the newly designed protocol will not degrade the throughput in many to one communications when the number of servers is increased and provides more security when compared with LTTP protocol.

Keywords— TCP/IP, RTT, RTO, TCP Incast, FEC scheme, Walsh code, throughput, LTTP.

I. INTRODUCTION

Transmission Control Protocol, TCP is one among the fundamental protocols in TCP/IP networks. While the IP protocol deals only with packets, TCP empowers two hosts to establish a connection and exchange streams of information. TCP ensures reliable delivery of information such that packets will be delivered in the same order in which they were sent. TCP is also known for its congestion control and error flow control. TCP is a transport layer protocol utilized by applications that require guaranteed delivery. It is a sliding window protocol that gives handling for both timeouts and retransmissions. TCP is a connection-oriented protocol, which means a full duplex virtual connection is established and maintained between two endpoints until the application programs at each end have completed exchanging messages. It determines how to divide application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and—because it is meant to provide error-free data transmission—handles retransmission of garbled or dropped packets as well as acknowledgement of all the packets that arrive.

TCP is used extensively for data transmission between servers in today's data center networks since it has showed a great conquest in the internet for both reliable delivery and congestion control. Nevertheless, the network environment and fixed application design in data centers acquaint new challenges with TCP to work easily. In such conditions, another issue called TCP Incast was found to happen which creates severe goodput collapse of many to one communication workloads [1]. The basic pattern of Incast starts when a particular Parent server puts a request for data to a cluster of Nodes which all get the solicitation at the same time. The cluster of Nodes may thus all simultaneously react to the solitary Parent. The result is a micro burst of many machines simultaneously sending TCP data streams to one machine (many to one).

TCP Incast causes goodput breakdown for two reasons. Firstly, when servers all the while send response packets over to the client, the response packets will flood the output buffer of the switch which specifically connects the client. Besides, the default estimation of TCP's Retransmission Timeout (RTO) is 200 milliseconds in most operating Systems. It implies that once a timeout happens, the TCP associations will be idle for quite a long period before the servers retransmit the dropped packets, since the RTT (Round-Trip Time) is just several microseconds in data center network systems. After the retransmission timer timeouts, the servers will again all the while send the response packets together, which causes switch buffer flood and retransmission for another round, so and so forth [1].

Cloud computing understands the fantasy of "computing as a utility". Individuals outsource their computing and software abilities to cloud suppliers and pay for the service use based on demand. Cloud data centers run both online services and back-end processing. Since most distributed computations in data centers are eager for bandwidth, they propose to expand the network capacity. They in this way oblige congestion free reliable data transmission. The goodput breakdown in many-to-one communication will significantly delay the task completion time of distributed processing, which is further meant the infringement of SLA. Since the underlying cause for TCP Incast is the small buffer in switches and the mismatch between RTO and RTT. TCP incast has recently become a crucial issue in data center networks because of its severe goodput breakdown [2].

When network congestion occurs, the constant sending rates at the servers will worsen the congestion condition and also grab an unfair share of bandwidth against TCP flows.

The synchronous many-to-one bust up can result in the emanation of congestion at the network port attached to the Parent server, embedding the port departure buffer. The subsequent packet drop obliges Nodes to identify the packet drop (missing ACKs), re-send data (after RTO), and gradually increase throughput per standard TCP behaviour. The application issuing the original request may hold up until all data has been received before the job can finish, or simply choose to return partial results obtained after a certain delay threshold is surpassed. In any case, the speed, quality, and consistency of performance endure. The congestion brought on by Incast has the effect of expanding the latency observed by the application and its users. There's a give and take with application performance as Node cluster size increases. Bigger clusters give more distributed processing capacity permitting more jobs to finish faster.

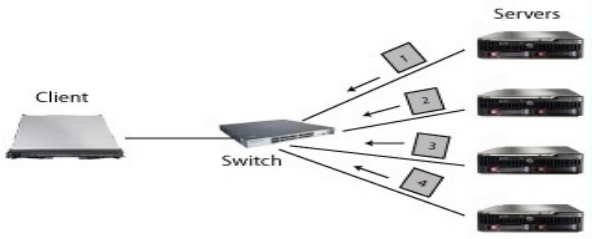


Fig. 1: TCP Incast Setup

A new transport protocol is used to support many-to one communications in data centers, which is called LTTP (LT-code based Transport Protocol). A typical TCP Incast scenario is shown in Figure 1. Since TCP's timeout is the root cause of low link utilization and goodput deterioration in TCP Incast, LTTP improves UDP-based LT (Luby Transform) code for reliable delivery, which depends on FEC (Forward Error Correction) with data redundancy. Since UDP cannot fairly share bandwidth with other protocols (such as TCP), TFRC (TCP Friendly Rate Control) is also applied to adjust the data sending rates at servers for congestion control. In this paper a simplified forward error correction mechanism is proposed which will reduce the required processing time than LTTP.

II. PRIOR WORKS

Many classic ways to handle congestion control have been proposed, and is being made use today for network traffic management. There are mainly two classes of approaches: implicit congestion control and explicit congestion control. While explicit techniques make use of any specific bit/fields in the packets to indicate control measures or network traffic status, the implicit techniques make use of mechanisms like packet acknowledgement to control the traffic congestion.

When a popular resource is shared without regulation, the result is always over-utilization. With the introduction of TCP in1983, users can write networking applications that require reliability with greater ease. When more applications are available, more data and information are exchange on the internet. In such cases congestion occurs. There have been many and increasingly sophisticated congestion avoidance mechanisms added to TCP [2]. The congestion control mechanism in TCP is an ever developing process. Most popular versions are as follows.

A. TCP Tahoe

This is the original version of TCP congestion control
Congestion detection mechanism is based on packet loss
Techniques used for congestion control:
- Slow start
- Congestion avoidance
- Fast retransmit

B. TCP Reno

This is the most popular version of TCP congestion control mechanism
Techniques used for congestion control:

Same as TCP Tahoe (Slow start, Congestion avoidance, Fast retransmit) plus Fast Recovery

C. TCP Vegas

This is completely new implementation

Congestion detection mechanism is based on end-to-end delay

In past years there have been many solutions to the TCP Incast problem mainly ICTCP and DCTCP, others are FQCN and AF-QCN.

1) *ICTCP:* In ICTCP, TCP incast is studied in detail by concentrating on the relationship among round trip time (RTT), receive window and TCP throughput. The notion was to form an ICTCP (Incast congestion Control for TCP) design at the recipient side [3]. Specifically, the strategy reconcile TCP receive window proactively before packet drops happen. The execution and experiments showed that zero timeout and high goodput for TCP incast was attained. The execution and assessment of ICTCP, was to enhance TCP performance for TCP incast in data center networks. Main focus was on receiver based congestion control algorithm to forestall packet drops. ICTCP adaptively reconcile TCP receive window based on the proportion of difference of achieved and expected per connection throughputs over expected ones, and additionally the last-hop available bandwidth to the recipient. A light-weighted, high performance Window NDIS filter driver was made to execute ICTCP. Compared with directly executing ICTCP as a part of the TCP stack, the driver execution can straightforwardly support virtual machines, which propagate in data centers. ICTCP was viable to maintain a strategic distance from congestion by achieving almost zero timeout for TCP incast, and it gives high performance and fairness among competing flows [3].

2) *DCTCP:* DCTCP is a TCP-like protocol for data center networks. DCTCP provides multi-bit feedback to the end hosts [4] by influencing Explicit Congestion Notification (ECN) in the network. End-to-end notification of network congestion without dropping packets [6] is the main function of ECN. ECN is only used when both endpoints support it and are willing to use it. Thus it is an optional feature. It is said to be effective only when supported by the underlying network. DCTCP uses much lesser buffer space, while delivers same or better throughput than TCP. Unlike TCP, DCTCP also provides low latency for short flows and high burst tolerance. DCTCP (Data Center TCP), a new variant of TCP, was designed. Also several performance impairments were noticed and linked to the behaviour of the commodity switches used in the cluster. Detailed traffic measurements from thousands of server data center cluster, running production soft real time applications motivated the work. It was found that switch buffer occupancies need to be persistently low, to meet the needs of the observed diverse mix of short and long flows, while maintaining high throughput for the long flows. These needs are met by DCTCP [4]. DCTCP relies on a feature now available on commodity switches called ECN (Explicit Congestion Notification). Through the use of the multi-bit feedback derived from the series of ECN marks, DCTCP succeeds, allowing it to react early to congestion.

3) *FQCN/AFQCN:* QCN (Quantized Congestion Notification) was developed by the IEEE Data Center Bridging Task Group to give congestion control at the Ethernet Layer or Layer 2 in data center network systems (DCNs) for IEEE 802.1Qau. The rate unfairness of distinct streams when offering one bottleneck link [5] is the fundamental buffet of QCN. That is the Quantized Congestion Notification algorithm was an unsophisticated and stable algorithm, yet it doesn't give weighted decency. In FQCN, FQCN (fair QCN), an upgraded QCN congestion notification algorithm, was proposed to enhance the rate assignment fairness of multiple streams imparting one bottleneck link in DCNs. FQCN distinguishes congestion offenders through per flow monitoring and joint queue, then feedbacks individual congestion data to every offenders through multi-casting, and hence ensures convergence to statistical fairness. The fairness and stability of FQCN was measured and through simulations in terms of the connection throughput, rate allocations to traffic streams with distinct traffic dynamics under three system topologies and the queue length stability, the performance of FQCN was assessed. In weighted fairness, the weights can be set by the operator. Simulation results affirmed the rate distribution unfairness of QCN, and validate that FQCN effectively allocates the fair share rate to each traffic source sharing the link capacity, maintains the queue length stability and improves TCP throughput performance in the TCP Incast setting [5]. The QCN (Quantized Congestion Notification) algorithm was designed to be responsive, simple to implement and stable. Even so, it does not provide weighted fairness. In weighted fairness, the weights can be set by the operator on a per-class or per-flow basis. Such a feature can be very useful in Data Center environments and multi tenanted Cloud Computing. This issue is addressed by AFQCN. It combines the QCN algorithm, and the AFD algorithm. AF-QCN requires no modifications to a QCN source (Reaction Point) and introduces a very light-weight addition to a QCN capable switch (Congestion Point) [5]. The results obtained through simulations show that AF-QCN retains the good congestion management performance of QCN while achieving rapid and programmable (approximate) weighted fairness.

III. PROPOSED SYSTEM

Existing approaches either require updating the OS/hardware, or cannot fundamentally solve the problem when the number of servers is large enough. In this paper, LTTP, a novel transport protocol for many-to-one communications in data center networks, is proposed. LTTP improves the LT code for data transmission from servers to the client, and adopts TFRC to control the traffic sending rates at servers.

1) System Model

The fundamental reason for goodput collapse in TCP Incast can be attributed to three factors [6]: 1) barrier-synchronized workload, 2) mismatch between TCP's RTO and RTT in data center networks, and 3) limited buffer size in switches. Since barrier-synchronized workload is determined by applications with many-to-one communication pattern, all the previous solutions focus on the latter two factors, i.e., either reducing the RTO or avoiding switch buffer overflow. The first category of solutions mitigates TCP Incast by setting the RTO value to microsecond granularity [2], [6], which matches the RTT in data centers. Though these types of solutions are effective in mitigating TCP Incast, updates to OS and hardware are required to implement microsecond-grained timers. Furthermore, with the introduction of optical fibers and optical switches in data centers, even microsecond-grained RTO may not be able to solve the problem [7].

The second category of solutions modify QCN (Quantized Congestion Notification), a Layer 2 end-to-end congestion management scheme introduced in IEEE 802.1Qau, to smartly control utilization of the switch buffer. In essence, the framework of QCN includes two parts: 1) QCN-enabled switch. The switch is responsible for monitoring the presence of congestion in the network and notifying it to the source. 2) QCN-enabled Network Interface Card (NIC), which receives the congestion notification sent from switches, and decreases the sending rate to alleviate the congestion. However, QCN gets poor performance in TCP Incast setting [10]. The reason is that QCN cannot ensure rate fairness among different flows which share the same bottleneck link. FQCN modifies the algorithm in QCN-enabled switches to achieve fairness among flows sharing the same bottleneck link, and mitigates the TCP Incast. AF-QCN proposes an algorithm used by the switches to achieve flexible weighted fair share of bandwidth for flows. However, these solutions need to update both the end hosts and switches.

2) LTTP

Since TCP's timeout is the root cause of goodput deterioration and low link utilization in TCP Incast, it naturally leads us to consider UDP instead of TCP for data transmission. It is because UDP has no timeout-and-retransmit mechanism. A new protocol named LTTP is proposed which is a UDP based transport protocol to support many to one communication in data centers. However, it is difficult to apply UDP in the many-to-one communication in data centers mainly because of two reasons. First of all, UDP only provides best-effort service, and cannot guarantee reliable data delivery. Secondly, UDP does not provide congestion control. LTTP applies TCP-friendly mechanism to congestion control and utilizes the FEC technique to guarantee reliable packet delivery. Digital fountain code (specifically, LT code) [8] was chosen to achieve reliable data delivery, and adopt TFRC [9] to adjust the data sending rates and maintain reasonable bandwidth utilization.

There are different implementations for digital fountain code. LT code and Raptor code are two. Among them, Raptor code outperforms LT code. Yet LT code is chosen to realize digital fountain code in the new protocol, LTTP. It is because that when the data size is small, the performance difference between LT code and Raptor code is insignificant. LTTP includes both data channel and control channel. Bigger clusters give more distributed processing capacity permitting more jobs to finish faster. The size of data exchanged between client and server is usually small in the TCP Incast scenario. Also, Raptor code needs to produce intermediate symbols first, and then utilizes these intermediate symbols as the input of LT code's encoding algorithm to produce the encoding data [7]. Therefore, the implementation complexity of LT code is much lower than that of Raptor code.

3) LTTP Framework

Every ns2 based project starts with the normal wireless or wired procedure [12]. Here a wired many to one communication is designed. A simulation environment of 10-11 nodes which are wired is created. The system starts with designing a network topology of the described kind as depicted in Fig. 2. Blue node represents the intermediate switch and yellow node represents the client to whom packets are intended to transfer. The nodes are created and the duplex connections are made between the nodes. The corresponding agents are attached and the flows id's are also given. The complete framework of LTTP to support many-to-one communication in data centers includes two parts, i.e., the data channel and the control channel between the client and each server. In the data channel, LT code was improved for reliable data transport, and adopts TFRC for controlling the traffic sending rate at servers. The control channel is employed by the client to issue data requests to servers and send terminating signals to the servers as soon as the requested data have been restored.

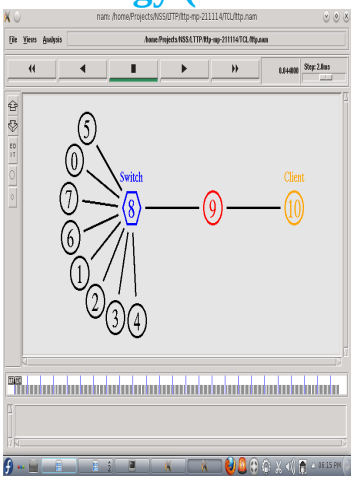


Fig. 2: Network Topology

The servers also use the control channel to send decoding parameters to the client. The decoding parameters include the original data size and block size, which are used by the client to execute the decoding process. For the control channel messages, the data size is small enough to be put into a single packet. Hence, it is unnecessary to employ coding for transmission. Instead, a TCP connection is established for each client server pair to deliver the control channel messages reliably.

4) Workflow Of LTTP

Fig. 3 illustrates the workflow of LTTP. First, the client establishes control channels (TCP connections) to all the servers. Second, the client sends requests to all the servers simultaneously through the control channel, asking the servers to start sending the data. Third, once receiving the request, the servers use control channel to send the decoding parameters back to the client. Meanwhile, each server starts to employ LT code [11] to produce and send encoding packets continually. TFRC is used by both servers and the client to control the sending rate. Finally, as soon as the original data is successfully restored, the client sends a terminating signal through control channel back to the corresponding server, which informs the server to stop encoding.

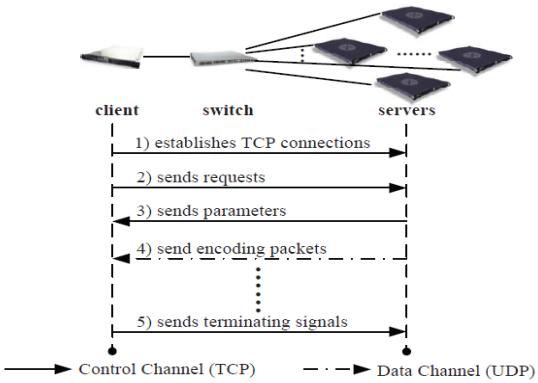


Fig. 3: Workflow of LTTP

Here TCP is used to transport the control packets in LTTP, incast congestion will not happen [14]. Firstly, the requests and terminating signals are sent from the client to the servers. The data size is small enough to be put into a single packet. The control packets are disseminated through different switch ports to servers, so the switch buffer will not overflow. Secondly, although the decoding parameters are transmitted from the servers to the client, the data size is very small.

5) Transmission Of LT Codes

Encoding process of digital fountain code, LT code is implemented as explained below. First of all, the original data is divided into number of blocks with equal size where each block is called as an input symbol. Size of each input symbol is termed as the block size. Then LT code produces encoding symbols on demand. Each encoding symbol is generated by performing simple Exclusive OR (XOR) operations on distinct d input symbols, where d is called the degree number of this encoding symbol [11].

LT code generates an encoding packet by combining the encoding symbol with the degree number and the indices of all the input symbols of the encoding symbol. This extra information is included in each encoding packet for decoder to restore the data. The encoding process is finished when the encoder receives a terminating signal from the decoder. The decoding parameters, i.e., original data size and block size, are used in the following ways. The decoding process uses block size to decide the size of restored input symbols and assemble restored data. The original data size divided by block size equals the number of input symbols, which is used as input of the standard decoding algorithm [11]. The decoding process starts when a certain number of encoding packets have been received.

6) Congestion Control Based On TFRC

LTTP employs TFRC [9] to conduct rate-based congestion control in the data channel. TFRC has similar phases to TCP's slow start and congestion avoidance. The server uses an initial (low) rate to start sending data and enters the slow start phase. During the slow start phase, in order to reach the fair share of bandwidth rapidly, the server doubles its sending rate in every RTT, but the increase of sending rate is no more than that of TCP, which results in that the slow start phase of TFRC takes more time than that of TCP. If a feedback packet from the client (receiver) indicates that there is packet loss (i.e., loss event rate is larger than zero), the slow start phase terminates, and enters the congestion avoidance phase, in which the server uses information contained in feedback packets to measure the RTT, and uses this RTT and loss event rate to calculate the upper bound of the sending rate *T* in bytes/sec. In contrast to TCP, TFRC depends on the feedback from receiver to control the data volume injected into network when congestion occurs.

7) Modified LTTP

When the UDP protocol used along with a forward error correction code, the overall output obtained increases, but the packet loss happening is found to be high. A new protocol, modified LTTP, is proposed in which a simple forward error correction method based on time and frame delay is employed to control packet loss and errors. The output is then given to Walsh encoding [12].

The Walsh code is an error-correcting code that is used for providing more security when transmitting messages over very noisy or unreliable channels. It is a linear code such that any linear combination of codeword becomes another codeword. Thus it provides more efficient encoding and decoding. Walsh code maps messages of length k to codewords of length 2^k. Errors can be corrected in extremely noisy conditions. Main advantage is that codewords are orthogonal. Any codeword is known as orthogonal if correlation between two codes is zero. Thus codes will be unique. After Walsh code encoding random bit inversion is done providing more security in data transmission. The receiver can decide whether the check bits agree with the data or not, to substantiate whether an error occurred in transmission or not with a certain degree of probability. The technique is also applied to data storage devices, such as a disk drive at times. In this occasion, each block on the disk would have check bits. When an error is detected, the hardware might report the error to the software or a reread of the block is automatically initiated by the hardware. It should be understood that the technique applies to storage reading and writing as well.

IV. CONCLUSION

As utility of computers increases day by day, Throughput collapse is a severe problem faced in many to one communication mechanism. TCP Incast in data center networks causes goodput collapse. Existing approaches either require updating the OS/hardware, or cannot fundamentally solve the problem when the number of servers is large enough. In this report, a new protocol named LTTP, a novel transport protocol for many-to-one communications in data center networks, was proposed which employs LT code for reliable data delivery and adopts TFRC for congestion control. LTTP will outperform all current solutions. Also a simplified forward error correction scheme is proposed along with walsh code which will provide more secured data tansmission without droping average throughput.

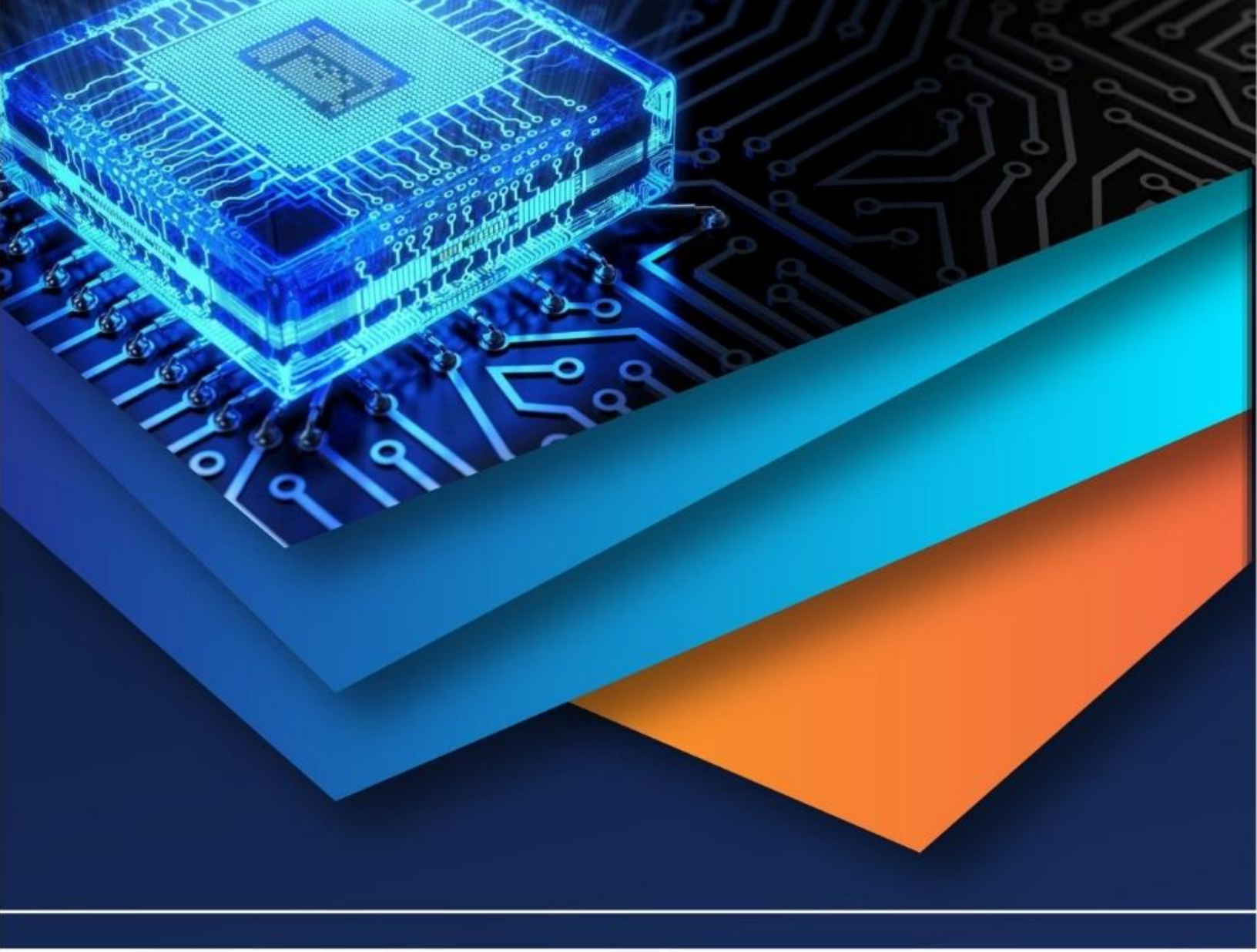
V. ACKNOWLEDGEMENT

The authors would like to thank professors of NSSCE, Palakkad for suggestions and support on this paper.

REFERENCES

[1] Jiang, Changlin, Dan Li, and Mingwei Xu. "LTTP: An LT-Code Based Transport Protocol for Many-to-One Communication in Data Centers." Selected Areas in Communications, IEEE Journal on 32.1 (2014): 52-64.
[2] Y. Chen, J. Liu, R. H. Katz and R. Griffith, "understanding tcp incast throughput collapse in data center networks," in Proc. ACM WREN 2009, Apr. 2011, pp. 43–53.
[3] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: incast congestion control for TCP in data center networks," in Proc. Co-NEXT '10, New York, NY,

USA, 2010.
[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in Proc. ACM SIGCOMM 2010, New York, NY, USA, 2010, pp. 63–74.
[5] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "Afqcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in IEEE 18th Annual Symposium on High Performance Interconnects (HOTI), 2010, pp. 58–65.
[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 data center communication," in Proc. SIGCOMM '09, 2009, pp. 303-314.
[7] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in Proc. IEEE INFOCOM 2011, Apr. 2011, pp. 1377 – 1385.
[8] P. Cataldi, M. Shatarski, M. Grangetto, and E. Magli, "Implementation and Performance Evaluation of LT and Raptor Codes for Multimedia Applications," in Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP '06, 2006, pp. 263–266.
[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification." [Online]. Available: http://www.ietf.org/rfc/rfc5348.txt
[10] Z. Yan and N. Ansari, "On mitigating tcp incast in data center networks," in Proc. IEEE INFOCOM 2011, Apr. 2011, pp. 51–55.
[11] M. Luby, "LT codes," in Proc. 43rd Symposium on Foundations of Computer Science, 2002, pp. 271 – 280.
[12] Farooq Ullah Khan, "Method of adaptive Walsh code allocation." U.S. Patent No. 7,280,581. 9 Oct. 2007.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY