

Providing Reliable Service in Data-center Networks

A.Suresh¹, S. Jaya Kumar²

¹M.Tech (CSE) Student, Department of CSE, SRM University, Ramapuram, Chennai, India suresh_hce2004@yahoo.co.in ²Assistant Professor, Department of CSE, SRM University, Ramapuram, Chennai, India, ven bsc@yahoo.co.in

Abstract

Transport Control Protocol (TCP) in cast congestion happens in high-bandwidth and low-latency networks when multiple synchronized servers send data to the same receiver in parallel. For many important data-center applications such as Map Reduce and Search, this many-to- one traffic pattern is common. Hence TCP in cast congestion may severely degrade their performances, e.g., by increasing response time. In this paper, we study TCP in cast in detail by focusing on the relationships between TCP throughputs, round-trip time (RTT), and receive window. Unlike previous approaches, which mitigate the impact of TCP in cast congestion by using a fine- grained timeout value, our idea is to design an In cast congestion Control for TCP (ICTCP) scheme on the receiver side. In particular, our method adjusts the TCP receive window proactively before packet loss occurs. The implementation and experiments in our test bed demonstrate that we achieve almost zero timeouts and high good put for TCP in cast. Hence, TCP connections with short round-trip times may receive unfairly large allocations of network bandwidth when compared to connections with longer round-trip times.

Keywords: Distributed Network Management, Data Center Network, TCP Incast Congestion

1. Introduction

This paper focuses on avoiding packet loss before in cast congestion, which is more appealing than recovery after loss. Of course, recovery schemes can be complementary to congestion avoidance.

The smaller the change we make to the existing system, the better. To this end, a solution that modifies only the TCP receiver is preferred over solutions that require switch and router support (such as ECN) and modifications on both the TCP sender and receiver sides. Our idea is to perform in cast congestion avoidance at the receiver side by preventing in cast congestion. The receiver side is a natural choice since it knows the throughput of all TCP connections and the available bandwidth. The receiver side can ad-just the receiver window size of each TCP connection, so the aggregate burstiness of all the synchronized senders are kept under control. We call our design in cast congestion Control for TCP (ICTCP).

TCP does not work well for many-to -one traffic pat-terns on high-bandwidth, low-latency networks. Congestion occurs when many synchronized servers under the same Gigabit Ethernet switch simultaneously send data to one receiver in parallel. Only after all connections have finished the data transmission can the next round be issued. Thus, these connections are also called barrier-synchronized. The final performance is determined by the slowest TCP connection, which may suffer from timeout due to packet loss. The performance collapse of these many-to-one TCP connections is called TCP in cast congestion. The root cause of TCP in cast collapse is that the highly burst traffic of multiple TCP connections overflows the Ethernet switch buffer in a short period of time, causing intense packet loss recovery with faster retransmissions [2], or controlling switch buffer occupation to avoid overflow by using ECN and modified TCP on both the sender and receiver sides [5].

However, adequately controlling the receive window is challenging: The receive window should be small enough to avoid in cast congestion, but also large enough for good performance and other non in cast cases. A well-performing throttling rate for one in cast scenario may not be a good fit for other scenarios due to the dynamics of the number of connections, traffic volume, network conditions, etc.



2. Problem statement

The existing algorithm focuses on the issue of estimating the used bandwidth by counting ACK packets and by filtering the information they convey. The ability of this method to perform well when the data is not perfect is crucial for the extension of the method from spheres (for which it was initially developed) to arbitrary shapes [2]. The Proposed algorithm redirects the packet to header parser module. Packet header is parsed and the information on flow table is updated. Algorithm module is responsible for receive window calculation .If a TCP ACK packet is sent out, the header modifier change the receive window field in TCP header if need. Client server communication acknowledgement is available .It is a variable bandwidth based on packet size. Packet retransmission can be done by using intermediate router. With the support of TOR switch we can make connection with dozens of servers [4]. Amount of data transmitted by each connection relatively small. The files are deliberately stored in multiple servers. Packet loss is controlled by using ICTCP algorithm [5].

2.1 Bandwidth allocation for Data Center

Running multiple virtual networks over a real physical substrate is a promising way to provide agility in current data centers.[2] However, such virtual networks may experience severely degraded performance due to the competing of network traffic on shared physical links. Based on the idea of the Stack solution from non-cooperative game theory, this paper presents a hierarchical game theoretic model for dynamic bandwidth allocation between virtual networks, which can be stable and can maximize the revenue of both infrastructure providers who manage the physical infrastructure and service providers who utilize the virtual networks to provide services. In the model, the data center owner as a leader designs a pricing mechanism for bandwidth allocation that attempts to drive the virtual networks to the social optimal solution, each virtual network as a follower chooses a willingness-to-pay to maximize its own profit. Experimental results show that the bandwidth allocation between virtual networks is efficient and fair [6].

The traffic and network conditions in data-center networks create the three preconditions for in cast congestion as summarized in [2]. First, data-center networks are well structured and layered to achieve high bandwidth and low latency, and the buffer size of top - of-rack (ToR) Ethernet switches is usually small. Second, a recent measurement.

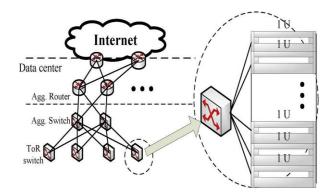


Fig. 1.Data-Center Network



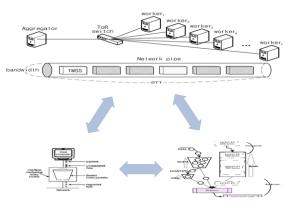


Figure2. System Architecture Diagram

Once the system has been designed, the next step is to convert the designed one in to actual code, so as to satisfy the user requirements as excepted. If the system is approved to be error free it can be implemented. When the initial design was done for the system, the department was consulted for acceptance of the design so that further proceedings of the system development can be carried on. After the development of the system, a demonstration was given to them about working of the system. The aim of the system illustration was to identify any malfunctioning of the system.

2.2 Citations and References

[1].S.Kandula,S.Sengupta, A.Greenberg,P.Patel,and R.Chaiken, "The nature of data center traffic: Measurements & analysis,"inProc.IMC,2009-We explore the nature of traffic in data centers, designed to sup-port the mining of massive data sets. We instrument the servers to collect socket-level logs, with negligible performance impact. In a 1500 server operational cluster, we thus amass roughly a terabyte of measurements over two months, from which we obtain and report detailed views of traffic and congestion conditions and patterns. We further consider whether traffic matrices in the cluster might be obtained instead via tomography inference from coarser-grained counter data. [2].Ganger,G.Gibson,and B.Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," inProc.ACM SIGCOMM, 2009-This paper presents a practical solution to a problem facing high-fan-in, high-bandwidth synchronized TCP workloads in datacenter Ethernets the TCP in cast problem.

[3]. S.Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. OSDI, 2004, -Map Reduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the pro-gram's execution across a set of machines, handling ma-chine failures, and managing the required inter-machine communication.

3. ICTCP Algorithm

ICTCP provides a receive window- based congestion control algorithm for TCP at the end-system. The receive windows of all low-RTT TCP connections are jointly adjusted to control throughput on incast Congestion [7]. Our ICTCP algorithm closely follows the design points made in Section III. In this section, we describe how to set the receive window of a TCP connection [8]. There are several benefits that can be achieved when ICTCP is implemented in a driver: [2] 1) It naturally supports virtual machines, which are widely used in data centers. We discuss this point in detail in the following section. 2) ICTCP needs the incoming throughput on a very small time



granularity (comparable to RTT at hundreds of microseconds) to estimate available bandwidth, and this information can be easily obtained at a driver. Note that the incoming traffic includes all types of traffic arriving on that interface, besides TCP. 3) It does not touch TCP/IP implementation in the Windows kernel. As a quick and dirty solution, it supports all OS versions instead of patching each one by one for deployment in a large data-center network with various TCP implementations.

3.1 ICTCP pseudo-code

Initially: cwnd = 1; ssthresh = infinite; New ack received: if (cwnd < ssthresh) /* Slow Start*/ cwnd = cwnd + 1; else /* Congestion Avoidance */ cwnd = cwnd + 1/cwnd; Timeout: /* Multiplicative decrease */ ssthresh = cwnd/2; cwnd = 1;

Sliding windows, a technique also known as windowing, is used by the Internet's Transmission Control Protocol (TCP) as a method of controlling the flow of packets between two computers or network hosts [3]. TCP requires that all transmitted data be acknowledged by the receiving host. Sliding windows is a method by which multiple packets of data can be affirmed with a single acknowledgment. DCTCP is an enhancement to the TCP congestion control algorithm for data center networks. It leverages Explicit Congestion Notification (ECN)[1], a feature which is increasingly becoming available in modern data center switches. DCTCP sources extract multi-bit feedback on congestion from the single-bit stream of ECN marks by estimating the fraction of marked packets. In doing so, DCTCP sources react to the extent of congestion, not just the presence of congestion as in TCP[2]. This finer level of control allows DCTCP to operate with very low buffer occupancies while simultaneously achieving high throughput [2].



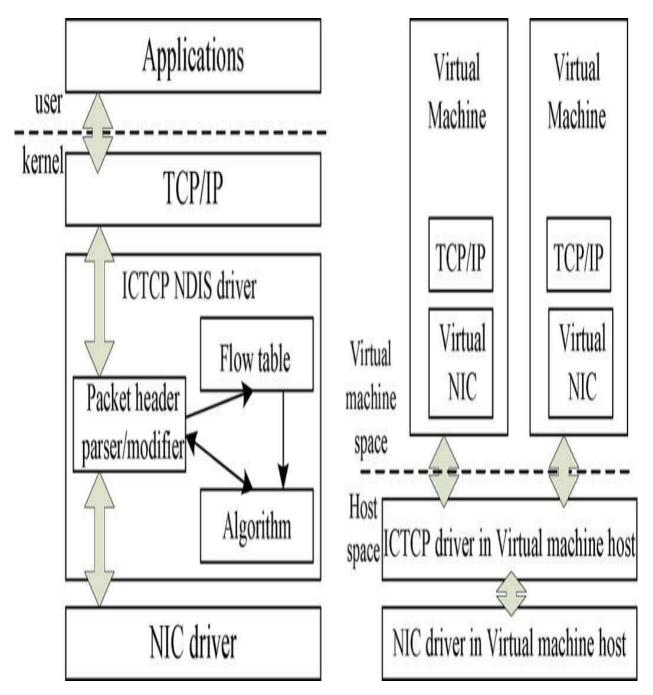


Figure. 3. Modules in ICTCP Driver and Software Stack For Virtual Machine Support.



# gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=0 Mbit/s. rtt=0 ms. Loss=0 %. qs=50 # == em0 = Input Utput Utput # gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=0 Mbit/s. rtt=0 ms. Loss=0 %. qs=50 117201323 0: 41071 0 2710686 61297 0 92786986 0 0 763.981 Mbps 117201323 0: 56180 0 3708474 83665 0 126662754 0 0 1042.970 Mbps 117201323 0: 56180 0 3708474 83665 0 1226662754 0 0 1042.970 Mbps 117201323 0: 56180 0 3708474 83665 0 1226662754 0 0 1005.955 Mbps 117201323 0: 56160 0 3574560 80770 0 122504864 0 0 1006.955 Mbps 117201323 0: 5615 0 3670590 82961 0 125610524 0 0 1094.249 Mbps # gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=200 Mbit/s. rtt=0 ms. Loss=0 %. qs= 117201323 0: 6816 0 474644 8071 0 13567482 0 0 122.337 Mbps 1172013230 0: 12213 0 813042 16504 0 2497868 0 0 266.331 Mbps 1172013240 0: 12210 0 813942 16504 0 2497868 0 0 266.331 Mbps 1172013241 0: 12215 0 1432638 31764 0 44978788 0 0 266.332 Mbps # gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=400 Mbit/s. rtt=0 ms. Loss=0 %. qs= 1172013241 0: 2155 0 143623 81764 0 49208776 0 0 396.179 Mbps 1172013241 0: 2155 0 1437632 32760 49298014 0 0 406.184 Mbps # gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=600 Mbit/s. rtt=0 ms. Loss=0 %. qs= 1172013244 0: 22079 0 1470546 32576 0 49316182 0 0 466.282 Mbps 1172013246 0: 22087 0 1470863 32585 0 49316182 0 0 466.282 Mbps 1172013246 0: 22087 0 1471086 32585 0 49316182 0 0 466.288 Mbps 1172013250 0: 31512 0 2095964 46385 0 7020794 0 0 577.994 Mbps 1172013250 0: 31512 0 2095964 46385 0 7020794 0 0 577.994 Mbps 1172013250 0: 31512 0 2095964 45357 0 7059420 0 0 577.994 Mbps 1172013250 0: 31664 0 2101956 46594 0 7055420 0 0 577.994 Mbps 1172013250 0: 31664 0 2101956 46594 0 7055420 0 0 577.994 Mbps 1172013250 0: 31664 0 2101956 46594 0 7055420 0 0 577.994 Mbps 1172013250 0: 13604 0 2101956 45587 0 70207948 0 0 566.824 Mbps # gigEnn.ctl.sh: nic1=em0. nic2=em1. mtu=1500 bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= 1172013250 0: 13042 0 2085122 4557 0 4336															FreeBSD3# cat /t
# epoch-time status packets errs bytes packets errs bytes coll drops Thruput gigfm.n.cl.sh: nicl=em0. nicl=em0. 2710686 61297 0 92786986 0 763.981 Mbps 172013234 0: 56189 3708474 83665 0 12662734 0 0 142.970 Mbps 172013235 0: 50419 0 3377666 75155 0 113787833 0 936.924 Mbps 172013236 0: 55615 0 670590 82910 0 1225418524 0 1066.955 Mbps 172013238 0: 151616 474644 89710 0 13567482 0 212.337 Mbps 172013240 0: 12213 8 813042 165944 24978368 0 266.337 Mbps 172013240 0: 12215 0 13174 1659449 24978368 0	slot	qs=50	8	oss=0	ns	tt=0		Mbit/s.			. mtu=1500	em1		nicl=er	
igipEn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bu=0 Mbit/s. rtt=0 ms. Loss=0 %. qs=50 172013231 0: 41071 0 2710666 61297 0 92786986 0 0 163.981 Mbps 172013234 0: 56119 0 3370666 75155 0 113787833 0 0 936.924 Mbps 172013235 0: 56419 0 3374560 80770 0 1225418524 0 0 1042.979 Mbps 172013236 0: 55615 0 3670590 82961 0 1225618524 0 0 1006.955 Mbps 172013238 0: 6516 0 474644 8971 0 13567482 0 0 112.337 Mbps 172013239 0: 12210 0 813421 16544 0 24978368 0 0 206.331 Mbps 172013241 0: 12210 0 81374 16544 0 24978368 0 0 206.332 Mbps 172013241 0: 12211 0 81374 16544 0 24978368 0 0 206.332 Mbps 172013243 0: 2152 0 1432638 31764 0 48089776 0 396.179 Mbps 172013244 0: 22079 0 1470546 3276 0 49298014 0 0 406.148 Mbps 172013245 0: 12952 0 1432638 31764 0 49298014 0 0 406.288 Mbps 172013246 0: 22087															
172513233 0: 41071 0 2710686 61297 0 92786986 0 763.981 Mbps 172013234 0: 56189 0 3337666 75165 0 113787333 0 936.924 Mbps 172013235 0: 50419 0 3337666 75165 0 113787333 0 936.924 Mbps 17201327 0: 55515 0 3574560 80770 0 125618524 0 0 1084.249 Mbps gigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytesbw=200 Mbit's.rtrt=0 msLoss=0 %qs= 12337 Mbps 172013240 0: 12210 813042 165644 24976854 0 206.331 Mbps 172013241 0: 12210 813174 165044 24978368 0 206.332 Mbps 172013242 0: 12210 165044 24978368 0 206.332 Mbps 172013243 0: 215276 1432638 31764 0 49089776 0 366.17 Mbps 172013245 1 19954 0 327692 29542 0															
172013234 0: 56189 0 3768474 83665 0 126662754 0 1042.979 Mbps 172013235 0: 50419 0 337666 75155 0 113787833 0 0 366.924 Mbps 172013236 0: 55615 0 3677560 82961 0 1225418524 0 0 1034.249 Mbps 172013236 0: 15515 0 367950 82961 0 125618524 0 0 1034.249 Mbps 172013238 0: 12210 0 11342 15644 0 24978368 0 266.331 Mbps 172013241 0: 12211 0 81374 16544 0 24978368 0 266.332 Mbps 172013241 0: 12515 0 1432638 31764 0 49298014 0 266.332 Mbps 172013244 0: 21525 0 1432638 31764 0 349298014 0 366.79 Mbps	slot														
1172013255 0: 50419 0 3327666 75165 0 113787333 0: 936.524 Mbps 1172013256 0: 54160 0 3574560 80770 0 12254864 0: 1086.955 Mbps 0 gigEnn.ctl.sh: nicl=em0nic2=em1. mtu=1500 bytes. bw=200 MbitYs. rt=0 ms. Loss=0 s. qse 172013239 0: 12213 0 813042 16504 0 24976854 0: 266.317 Mbps 172013240 0: 12210 0 813174 16504 0 24976856 0: 266.332 Mbps 172013241 0: 12215 0 813174 16504 0 24976368 0: 266.332 Mbps 172013241 0: 2152 0 432638 31764 0 40808776 0: 396.179 Mbs 172013244 0: 22079 1470546 325760 49298014 0: 406.128 Mbps 172013246 0:															
172013236 0: 54160 0: 3574560 80770 0: 12224464 0: 0: 1066.955 Mbp 17201327 0: 55615 0: 3670590 295610 125618524 0: 0: 134.249 Mbps 17201328 0: 6816 0: 474644 8971 0: 13567482 0: 0: 12.337 Mbps 17201329 0: 12213 0: 813042 15564 0: 24978368 0: 0: 266.331 Mbps 172013240 0: 12211 0: 813741 16594 0: 24978368 0: 0: 266.332 Mbps 172013241 0: 12210 0: 81374 16594 0: 24978368 0: 0: 266.332 Mbps 172013243 0: 12512 0: 1432638 31764 0: 49298014 0: 0: 366.179 Mbps 172013245 0: 29954 0: 1327692 29542 0: 47080782 0: 368.291 Mbps 172013246 0: 220870 <td></td>															
172013237 0: 55615 0 3670590 62961 0 125616524 0 1034.249 Mbps igigEnn.ctl.sh:nicl=em0nic2=em1mtu=1500 bytes. bu=200 Mbit/srtt=0 ms. Loss=0 %., qs= 172013238 0: 1213 0 134242 16544 0 24976854 0 266.317 Mbps 172013240 0: 12215 0 81374 165644 0 24976854 0 266.337 Mbps ir2013241 0: 12215 0 81374 165844 0 24978868 0 266.332 Mbps ir2013241 0: 12215 0 132638 31764 0 49089776 0 396.179 Mbps 172013244 0: 21979 1470546 325760 49298014 0 406.288 Mps 172013246 0: 22079 14710866 32585 0 49316182 0 406.288 Mps 172013246 0: 22087 14710866 32585 0 49316182 0 406.2															
figEnn.ctl.sh: nicl=em0nic2=m1.entu=1500 bytes.bu=200 Mbit/s.rtt=0 ms. loss=0 s. generation 1772013289 6816 0 474644 8971 0 13567482 0 0 216.331 Mbps 177201329 0: 12213 0 813042 16504 0 24978368 0 0 266.331 Mbps 1772013240 0: 12215 0 813174 16504 0 24978368 0 0 266.332 Mbps 1772013241 0: 12210 0 813174 16504 0 24978368 0 0 266.332 Mbps 172013243 0: 1212 0 813174 16504 0 24978368 0 0 266.332 Mbps 172013243 0: 2152 0 1432638 31764 0 4808977 0 0 396.179 Mbps 172013244 0: 22070 0 1470646 32570 0 442988014 0 0 466.488 Mbps 172013246 0: 220870 0 1470863258 0 49315210 0 0 466.328 Mbps 172013249 0: 31612 0 209654 46385 0 70207934 0 0 578.439 Mbps 172013251 0: 31612 0 2096564 46385 0 70207934 0 0 578.439 Mbps 172013251 0: </td <td></td>															
1172013238 0: 6816 0 474644 6971 0 13567482 0 0 112:337 Mbps 1172013239 0: 12209 0 812778 16504 0 24978368 0 0 266.331 Mbps 1172013240 0: 12215 0 813174 16504 0 24978368 0 0 266.332 Mbps 1172013241 0: 12215 0 813174 16504 0 24978368 0 0 266.332 Mbps 1172013241 0: 21525 0 432638 31764 0 49089776 0 396.179 Mbps 1172013245 0: 22079 0 1470546 325760 49298014 0 0 466.288 Mbps 1172013246 0: 22087 0 1470868 32585 0 49316182 0 0 466.288 Mbps 1172013246 0: 22087 1471806 32585 0 49316182 0 0 576.439 Mbps 1172013240 0: 36971 0 2576.4															
17201329 0: 12213 0 813042 16504 0 24978368 0 266.313 Mbps 172013240 0: 12215 0 81374 16504 0 24978368 0 266.317 Mbps 172013241 0: 12210 0 81374 16504 0 24978368 0 266.317 Mbps 172013242 0: 12211 0 81374 16504 0 24978368 0 266.332 Mbps 172013243 0: 21525 1432638 31764 0 48089776 0 396.179 Mbps 172013244 0: 22070 1470863 32586 0 49319210 0 466.328 Mbps 172013246 0: 220870 1470863 32586 0 49319210 0 466.328 Mbps 172013250 0: 31512 2090930 45660 60128320 0 559.499 Mbps 172013251 0: 31604 210956 456540 70287934 0 <t< td=""><td>50 slot</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>bw=</td><td></td><td></td><td>eml</td><td></td><td></td><td></td></t<>	50 slot								bw=			eml			
172013240 0: 12209 0: 812778 16504 0: 24976854 0: 266.317 Mbps 172013241 0: 12215 0: 813174 16504 0: 24978368 0: 266.332 Mbps 172013241 0: 12215 0: 813174 16504 0: 24978368 0: 266.332 Mbps gigEnn.ctl.sh: nic2=eml. mtu=1509 bytes. bu=409 Mbit's. rtt=0 ms.<															
172013241 0: 12215 0 813174 16504 0 24078368 0 206.332 Mbps 172013242 0: 12210 0 813074 16504 0 24078368 0 0 266.332 Mbps 172013243 0: 12525 1432638 31764 0 48089776 0 396.179 Mbps 172013244 0: 22070 1470546 32576 0 4298014 0 406.148 Mbps 172013245 0: 220870 1470864 32586 0 43919210 0 406.322 Mbps 172013246 0: 220870 1470864 32586 0 49319210 0 406.322 Mbps 172013246 0: 220870 1470866 32586 0 4931612 0 406.322 Mbps 172013250 0: 31512 2090930 45660 69128320 0 578.439 Mbps 172013251 0: 31612 2090930 456654 70207934 0 578.439<		Mbps	.331	206.											
1172013242 0: 12211 0 812010 16594 0 24078368 0 206.330 Mbps igigEnn.ctl.sh: ncl=em0. nic2em1. mul1500 bytes. bw=400 Mbit/s. rtt=0 ms. Loss=0 s. qse 172013243 0: 21525 0 1432638 31764 0 49269014 0 0 406.148 Mbps 172013244 0: 22079 0 1470546 32576 0 49298014 0 0 406.282 Mbps 172013246 0: 22087 0 1470806 32585 0 49316182 0 0 406.288 Mbps 172013248 0: 22087 0 1470806 32585 0 49316182 0 0 576.439 Mbps 172013249 0: 309710 2059030 45660 60128320 0 576.439 Mbps 172013250 0: 31624 20906964 46385 70207934 0 577.994 Mbps 172013251 </td <td></td>															
rigigen.ctl.sh: nicl=em0nic2=mlmtu=1500 bytes. bu=400 Mbit/s.rtt=0 ms. Loss=0 s. q= 172012243 0: 2152 0 1432638 31764 0 40080776 0 336.179 Mbps 172012343 0: 22079 0 1470546 32570 0 42080776 0 366.179 Mbps 172012344 0: 22079 0 1470546 32570 0 42080716 0 0 368.291 Mbps 172013245 0: 22087 0 1470866 32585 0 43319210 0 466.328 Mbps 172013246 0: 22087 0 1470866 32585 0 49319210 0 466.328 Mbps 172013246 0: 22087 0 1470866 32585 0 49319210 0 466.328 Mbps 172013250 0: 31512 2090930 45660 60128320 0 578.439 Mbps 172013251 0: 31512 2090936 46534 0 70528836 0 558.439 Mbps 172013251 0: 31612 2090936 45654 0 70528836 0 558.424 Mbps 172013251															
1172013243 0: 21525 0 1432638 31764 0 48089776 0 0 396.179 Mbps 1172013244 0: 22079 0 1470546 32570 0 49298014 0 0 366.179 Mbps 1172013245 0: 19954 0 1337692 29542 0 44708728 0 368.291 Mbps 1172013246 0: 22087 0 1469868 32585 0 49316122 0 0 406.282 Mbps 172013249 0: 20970 0 1469868 32586 0 49316122 0 0 576.439 Mbps 172013250 0: 31512 2096964 46385 0 70159420 0 577.994 Mbps 172013251 0: 31644 2101956 46594 0 70159420 0 577.094 Mbps 172013252 0: 31664 2101956 46594 0 7058436 0 566.824 Mbps 172013251 0: 19060 205122 45457 0 68799448 0		Mbps	.330	206.	Θ	Θ	3	2497836	Θ	16504	812910	Θ	12211	Θ:	172013242
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	50 slot						i	00 Mbit/							
1172013245 0: 19954 0 1327602 29542 0 47086728 0 368.201 Mbps 172013246 0: 2087 0 147086 32585 0 49319210 0 0 466.322 Mbps 172013246 0: 22087 0 1469868 32586 0 49319120 0 0 466.288 Mbps gigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytesbw=600 Mbit/srtt=0 msLoss=0 %qs= 172013250 0: 31912 2096964 46385 0 70159420 0 578.439 Mbps 172013251 0: 31428 2096964 46385 0 70159420 0 577.994 Mbps 172013251 0: 31664 2191956 46594 0 7058836 0 581.846 Mbps 172013253 0: 30669 203122 45457 0 68799648 0 566.824 Mbps 172013254 1. nput Output 0utput 0utput 1039.703		Mbps	.179	396.	Θ	Θ	5	4808977	Θ	31764	1432638	Θ	21525	Θ:	172013243
$\begin{array}{cccccccccccccccccccccccccccccccccccc$		Mbps	.148	406.	Θ	Θ	1	4929801	Θ	32570	1470546	Θ	22079	Θ:	172013244
172013248 0: 22070 0: 1469868 32586 0: 49316182 0: 0: 466,288 Mbps gigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytes.bw=600 Mbit/s.rtt=0 s. Loss=0 s. Loss=0 s. A66,288 Mbps 172013249 0: 38971 0: 259930 45660 0: 69128320 0: 0: 578.439 Mbps 172013250 0: 31412 0: 2096964 46385 0: 70159420 0: 577.994 Mbps 172013251 0: 31428 0: 2096964 46385 0: 70159420 0: 577.994 Mbps 172013253 0: 31664 0: 2191956 46594 0: 70558436 0: 566.824 Mbps gigEnn.ctl.sh: nic1=em0nic2=em1mtu=1500 bytesbw=600 Mbit/s.rtt=0 ms. <loss=0< td=""> s. qs= epoch-time status packets errs bytes.bw=600 Mbit/s.rtt=0 ms.<loss=0< td=""> s. qs= 17201325</loss=0<></loss=0<>		Mbps	.291	368.	Θ	Θ	3	4470872	Θ	29542	1327692	Θ	19954	Θ:	172013245
gigEnn.ctl.sh: nicl=em0nic2=em1. mtu=1500 bytes. bu=600 Mbir/s.rtt=0 coss=0 *.ge= 172013250 0: 30071 2090904 45660 0 6128320 0 550.490 Mbps 172013250 0: 31512 0<2090904		Mbps	.322	406.	Θ	Θ	2	4931921	Θ	32585	1471086	Θ	22087	Θ:	172013246
172013249 0: 30971 0 2059030 45660 0 6122320 0 0 569.499 Mpps 172013250 0: 31512 0 2096964 46385 0 70207934 0 0 578.439 Mpps 172013251 0: 31428 0 2089836 46347 0 70159420 0 0 577.944 Mpps 172013252 0: 31664 0 2181956 46594 0 7852836 0 581.046 Mpps 172013253 0: 38669 0 253122 45457 0 68799848 0 556.824 Mpps gigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytes.bw=800 Mbit/s.rtt=0 ms.bw=80 Mbit/s.rtt=0 ms.bw=80 Mbit/s.rtt=0 ms.bw=80 Mbit/s.rtt=0 ms.bw=80 Mbit/s.rtt=0 ms.bw=80 % qs gigEnn.ctl.sh:nicl=em0.nic2=em1.mtu=1500 bytes.bw=800 Mbit/s.rtt=0 ms.bw=80 Mbit/s.rtt=0 ms.bw=80 % qs 17201325 0 126300 28877 0 4366634		Mbps	.288	406.	Θ	Θ	2	4931618	Θ	32586	1469868	Θ	22070	Θ:	172013248
172013250 0: 31428 2006064 46385 0 70207304 0: 578.439 Mbps 172013251 0: 31428 20090836 46347 0 70150420 0: 578.439 Mbps 172013251 0: 31604 0 20191956 46547 0 70150420 0: 578.439 Mbps 172013252 0: 30669 0 2053122 45457 0 68799848 0: 566.824 Mbps gigEnn.ttl.sh: nic2=em1. mtu=1500 bytes. bu=800 Mbit/s. rtt=0 ms. Loss=0 %. qs f epoch-time status packets errs bytes. bu=800 Mbit/s. rtt=0 ms. Loss=0 %. qs 172013254 0: 19552 1 206300 2831271 0: 233.177 Mbps 172013255 0: 12610 834636 18736 2381273 0: 233.177 Mbps 172013257 0: 18707 0 2431533 0: 231.249 Mbps 172013257 0: 18707 1242258 27569	50 slot	qs=5	=0 %.	Loss=	9 ms.	rtt=	s	00 Mbit/	bw=	bytes	. mtu=1500	=em1	n0 nic	nicl=er	gigEnn.ctl.sh:
1172013251 0: 31428 0 2069836 46347 0 70150420 0 577.904 Mbps 1172013252 0: 30869 0 2101956 46594 0 70528836 0 0 581.046 Mbps 1172013252 0: 30869 0 2053122 45457 0 68799848 0 566.824 Mbps 4 gigEnn.t1.sh: nicl=em0nic2=em1mtu=1500 bytesbw=B00 Mbit/srtt=0 Thruput 4 4 epoch-time status packets errs bytes. bytesbw=B00 Mbit/srtt=0 Nos=0%qs= 172013254 0: 19552 0 1296300 28877 0 43666634 0 359.703 Mbps 172013255 0: 12010 834636 18736 0 231.177 Mbps 172013256 0: 12010 834636 18736 0 24831533 0 244.491 Mbps 172013257 0: 18707 0 1242258 27569 0 4165925 0 343.265 Mbps </td <td></td> <td>Mbps</td> <td>.499</td> <td>569.</td> <td>Θ</td> <td>Θ</td> <td>9</td> <td>6912832</td> <td>Θ</td> <td>45660</td> <td>2059030</td> <td>Θ</td> <td>30971</td> <td>Θ:</td> <td>172013249</td>		Mbps	.499	569.	Θ	Θ	9	6912832	Θ	45660	2059030	Θ	30971	Θ:	172013249
1172013252 0: 31664 0 2101956 46504 0 79528336 0 581.646 Mpps 1172013253 0: 30869 0 2053122 45457 0 68799848 0 566.824 Mpps gigfnn.ctl.sh: nicl=em0. nicl=em1. mtu=1500 bytes. byeb0 Mbit/s. rtt=0 ms. Los=0 %. qse epoch-time status packets errs bytes byeb0 Mbit/s. rtt=0 ms. Los=0 %. qse figEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. byeB0 Mbit/s. rtt=0 ms. Los=0 %. qse 172013254 0: 19552 0 206300 2831271 0 233.177 Mbps 172013255 0: 12610 0 834636 18736 0 24831533 0 0 244.491 Mbps 172013257 0: 18707 0 1242258 27569 0 4165925 0 343.265 Mps.		Mbps	.439	578.	Θ	Θ	£	7020793	Θ	46385	2096964	Θ	31512	Θ:	172013250
172013253 0: 30869 0 2053122 45457 0 68799848 0 566.824 Mbps 4 gigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytesbu=800 Mbit/s.rt=0 msLoss=0 %qs= 4 epoch-time status packets errs bytes. bytes.lbs00 Mbit/s.rt=0 msLoss=0 %qs= 4 epoch-time status packets errs bytes. bytes.lbs00 Mbit/s.rt=0 msLoss=0 %qs= 172013254 0: 19552 0 1206300 28877 0 43666634 0 259.703 Mbps 172013255 0: 12610 834636 18736 2 28312471 0 233.177 Mps 172013256 0: 118707 0 1242258 27569 0 4165925 0 433.656 Mbps 172013257 0: 18707 0 1242258 27569 0 4165925 0 433.256 Mps 19Gen.ctl.sh: nicl=em0nic2=em1mtu=1500 bytesbw=1000 Mbit/srtt=0 msLoss=0 msloss=0 msloss=0 msl		Mbps	.994	577.	Θ	Θ		7015942	Θ	46347	2089836	Θ	31428	Θ:	172013251
# gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= # == m0 == Input Output # qsoch-time status packets errs bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mbit/s. rtt=0 ms. Loss=0 %. qs= l172013254 0: 19552 0: 296300 28877 0: 43666634 0: 0: 359.703 Mbps l172013255 0: 11640 0: 24831533 0: 0: 244.491 Mbps l172013257 0: 118707 0: 1242258 27569 0: 41665925 0: 343.265 Mbps l172013258 0: 11970 0: 792480 17770 0: 26862484 0: 0: 221.240 Mbps f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=1000 Mbit/s rtt=0 ms. Loss=0 %. qs 1172013250 0: 13918 0: 21727 20653 0: 3121333 0: 0: 257.075 Mbps l172013260 0: 14936 0: 9860688 2176 0: 3353218 0: 0: 276.75 Mbps l172013261 0: 16763 1110678 24781 0: 3433530 0: 368.352 Mbps l172013261 0: 16763		Mbps	.046	581.	Θ	Θ	5	7052883	Θ	46594	2101956	Θ	31604	Θ;	172013252
mem0 == Input Output f epoch-time status packets errs bytes. bw=800 Mit/s. rtt=0 f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mit/s. rtt=0 s. f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mit/s. rtt=0 s. loss=0 s. gs 1172013255 0: 12610 834636 18736 28312471 0 233.177 Mps 1172013256 0: 11640 779876 16409 24831533 0 244.491 Mps 1172013257 0: 18707 0 242258 27569 41655925 0 343.265 Mps 1172013258 0: 11972 0 792480 17770 0 26862444 0 221.240 Mps 1172013259 0: 11972 291072 20653 0 31213333 0 277.675 Mps 1172013250 <t< td=""><td></td><td>Mbps</td><td>.824</td><td>566.</td><td>Θ</td><td>Θ</td><td>3</td><td>6879984</td><td>Θ</td><td>45457</td><td>2053122</td><td>Θ</td><td>30869</td><td>Θ:</td><td>172013253</td></t<>		Mbps	.824	566.	Θ	Θ	3	6879984	Θ	45457	2053122	Θ	30869	Θ:	172013253
mem0 == Input Output f epoch-time status packets errs bytes. bw=800 Mit/s. rtt=0 f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mit/s. rtt=0 s. f gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bw=800 Mit/s. rtt=0 s. loss=0 s. gs 1172013255 0: 12610 834636 18736 28312471 0 233.177 Mps 1172013256 0: 11640 779876 16409 24831533 0 244.491 Mps 1172013257 0: 18707 0 242258 27569 41655925 0 343.265 Mps 1172013258 0: 11972 0 792480 17770 0 26862444 0 221.240 Mps 1172013259 0: 11972 291072 20653 0 31213333 0 277.675 Mps 1172013250 <t< td=""><td>50 slot</td><td> as=5/</td><td>=0 %.</td><td>Loss=</td><td>9 ms.</td><td>rtt=</td><td></td><td>90 Mbit/</td><td>bw=1</td><td>bytes</td><td>mtu=1500</td><td>=em1</td><td>NO., nic</td><td>nicl=er</td><td>≠ aiaEnn.ctl.sh:</td></t<>	50 slot	as=5/	=0 %.	Loss=	9 ms.	rtt=		90 Mbit/	bw=1	bytes	mtu=1500	=em1	NO., nic	nicl=er	≠ aiaEnn.ctl.sh:
# epoch-time status packets errs bytes packets errs bytes packets errs bytes clish ricles Thruput # gigEnn.tl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bu=800 Mbit/s. rt=0 ms. Loss=0 s. qs qs<															
f jigEnn.ctl.sh: nicl=em0nic2=em1mtu=1500 bytes.bu=800 Mbit/s.rtt=0 ms. Loss=0 %, g= 1172013255 0: 19552 0: 26300 28877 0: 43666634 0 0: 359.703 Mbps 1172013255 0: 12610 0: 834636 18736 0: 28312471 0 0: 233.177 Mbps 1172013255 0: 12610 0: 834636 18736 0: 24831533 0 0: 234.177 Mbps 1172013257 0: 18707 0: 1242258 27569 0: 41665925 0: 343.2656 Mbps 1172013258 0: 11972 0: 792480 17770 0: 2662484 0: 221.240 Mbps 1172013259 0: 18918 9:21072 20653 0: 31213333 0: 277.675 Mbps 1172013260 0: 14936 9:89608 2176 33553218 0: 276.735 Mbps 1172013261			put	Thrup	drops	olls	0	s byte	s er	packet	bytes	errs	packets	status	≠ epoch-time
1172013254 0: 19552 0 1296300 28877 0 43666634 0 0 359.703 Mbps 1172013255 0: 12610 0 834636 18736 0 2812471 0 0 233.177 Mbps 1172013255 0: 11040 0 729876 16409 0 24831533 0 0 204.491 Mbps 1172013257 0: 18707 0 1242258 27569 0 41665925 0 343.265 Mbps 172013258 0: 11970 0 724870 17760 256862484 0 221.240 Mbps 172013250 0: 13918 0 21072 26653 31213333 0 0 257.075 Mbps 172013260 0: 14936 0 988608 22176 33553218 0 276.035 Mbps 172013260 0: 14936 988608 22176 33553218 0 308.352 Mbps 172013261 0: 16763 1110678 <td>50 slot</td> <td> as=5/</td> <td>-0 %.</td> <td>Loss=</td> <td>9 ms.</td> <td>rtt=</td> <td>÷</td> <td>00 Mbit/</td> <td>bw=</td> <td>bytes</td> <td>mtu=1500</td> <td>em1</td> <td>n0., nic</td> <td>nicl=er</td> <td>aigEnn.ctl.sh:</td>	50 slot	as=5/	-0 %.	Loss=	9 ms.	rtt=	÷	00 Mbit/	bw=	bytes	mtu=1500	em1	n0., nic	nicl=er	aigEnn.ctl.sh:
1172013255 0: 12610 0 834636 18736 0 2831271 0 0 233.177 Mbps 1172013256 0: 11040 0 779876 16409 0 24831533 0 0 234.491 Mbps 1172013257 0: 18707 0 1242258 27569 0 41665925 0 0 343.265 Mbps 172013257 0: 18707 0 792480 17770 0 26862484 0 0 231.2140 Mbps gigfcnn.ctl.sh: nic1=em0 nic2=em1 mtu=1500 bytes bu=1000 Mbit/s rt=0 ms Loss=0 % gs gs 76.333 0 257.675 Mbps 172013260 0: 14936 0 921072 20653 0 3123333 0 257.675 Mbps 172013261 0: 14936 0 921072 20653 0 31535218 0 368.352 Mbps 172013261 0: 14936 0 9110678 24781 0 <td></td>															
1172013256 0: 11040 0 729876 16409 0 2483153 0 0 264.491 Mbps 1172013257 0: 18707 0 1242258 27569 0 41665925 0 343.265 Mbps 1172013258 0: 11972 0 792480 17770 0 26862444 0 0 221.240 Mbps f gigEnn.ctl.sh: nic2=em1. mtu=1500 bytes. by=1000 Mbir/s. rtt=0 %.0 327.075 Mbps 172013260 0: 14936 988668 22176 0 33553218 0 0 276.335 Mbps 172013261 0: 16763 0 110678 24781 0 3743350 0 0 276.335 Mbps 172013261 0: 20645 0 1327398 29813 0 45655650 0 0 308.352 Mbps					Θ	Θ	6 3	2831247	Θ	18736	834636	Θ	12610	Θ:	172013255
172013257 0: 18707 0 1242258 27560 0 4165925 0 0 343.265 Mbps 172013258 0: 11970 0 792480 17770 0 26862484 0 0 221.240 Mbps gigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 bytes. bu=1000 Mbit/s. rtt=0 ms. Los=0 %. qs 172013250 0: 13918 0 921072 20653 0 3121333 0 0 257.075 Mbps 172013260 0: 14936 0 986088 2176 33553218 0 0 257.075 Mbps 172013261 0: 16763 1110678 24781 0 3743353 0 308.352 Mbps 172013262 0: 20045 0 317398 29813 0 45659650 0 311.096 Mbps					Θ	Θ	3	2483153	Θ	16409	729876	Θ	11040	Θ:	172013256
172013258 0: 11972 0 792480 17770 0 26862484 0 0 221.240 Mbps gigEnn.ctl.sh: nicl=em0 nic2=em1. mtu=1500 bytes. bw=1000 Mbir/s rt=0 ms Loss=0 % qs 172013250 0: 13918 0 927.075 Mbps 1213333 0 0 276.75 Mbps 172013260 0: 14936 988608 22176 0 33553218 0 276.335 Mbps 172013261 0: 16763 0 110678 24781 0 37433350 0 386.352 Mbps 172013261 0: 20045 0 1327398 29813 0 45659659 0 371.096 Mbps					Ð	Θ	5	4166592	Ð	27569	1242258	Ø	18707	Θ:	172013257
ipigEnn.ctl.sh: nicl=em0. nic2=em1. mtu=1500 by=1000 Mbir/s rtt=0 ms Loss=0 % qs 172013250 0: 13018 0 921072 26653 0 31213333 0 257.075 Mbps 172013260 0: 14936 0 988608 22176 0 33553218 0 276.335 Mbps 172013261 0: 16763 1110678 24781 0 3743335 0 308.352 Mbps 172013261 0: 20045 0 317398 29813 0 45059650 0 371.006 Mbps					Ð	Θ			Θ			Θ		Θ:	
172013259 0: 13918 0 921072 20653 0 31213333 0 0 257.075 Mbps 172013260 0: 14936 0 988608 22176 0 33553218 0 0 276.335 Mbps 172013261 0: 16763 0 1110678 24781 0 37433350 0 368.352 Mbps 172013262 0: 20045 0 1327398 29813 0 45059650 0 371.096 Mbps	=50 slot.				=0 ms	rtt	S.	DOD Mbit	bw=	bytes.	mtu=1500	em1.	n0. nic	nicl=er	aigEnn.ctl.sh:
172013260 0: 14936 0 988608 22176 0 33553218 0 0 276.335 Mpps 172013261 0: 16763 0 1110678 24781 0 3753326 0 308.352 Mpps 172013262 0: 20045 0 1327398 29813 0 45959659 0 0 31.096 Mpps		Mbps	.075	257.	Θ	Θ		3121333	Θ	20653	921072	Θ	13918	Θ:	172013259
172013261 0: 16763 0 1110678 24781 0 37433350 0 0 308.352 Mbps 172013262 0: 20045 0 1327398 29813 0 45059650 0 371.096 Mbps															
172013262 0: 20045 0 1327398 29813 0 45059650 0 0 371.096 Mbps															
					ē	ø			Θ	19449	863796		13070	Θ:	
reeBSD3#		Combine C											22370		

Figure.4. Bandwidth limitations for Round Trip Time using TCP

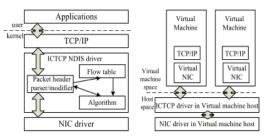


Figure 5: Modules in ICTCP driver and software stack for virtual machine support

Transport Control Protocol (TCP) in cast congestion happens in high-bandwidth and low-latency networks when multiple synchronized servers send data to the same receiver in parallel [2]. For many important data-center applications such as Map Reduce and Search, this many-to-one traffic pattern is common [3]. Hence TCP in cast congestion may severely degrade their performances, e.g., by increasing response time. In this paper, we study TCP in cast in detail by focusing on the relationships between TCP throughput, round-trip time (RTT)[4], and receive window. Unlike previous approaches, which mitigate the impact of TCP incast congestion by using a fine-grained timeout value, our idea is to design an In cast congestion Control for TCP (ICTCP) scheme on the receiver side. In particular, our method adjusts the TCP receive window proactively before packet loss occurs[5].

3.2 Bucket leaky algorithm and rate allocation algorithm

Bucket Leaky Algorithm is an algorithm used to check that data transmissions, in the form of packets, conform to defined limits on band width [4]. A fixed capacity bucket, associated with each virtual connection or user, leaks at a



fixed rate [3]. If this amount of water would cause the bucket to exceed its capacity then the packet does not conform and the water in the bucket is left unchanged [4].

3.3 Bucket leaky pseudo-code

We consider rate allocation algorithm for resolving fundamental problem of bandwidth allocation among flows in a packet-switched network [2]. The classical max-min rate allocation has been widely regarded as a fair rate allocation policy. We generalize the theory of the classical max-min rate allocation with the support of both the minimum rate and peak rate constraints for each flow [3].

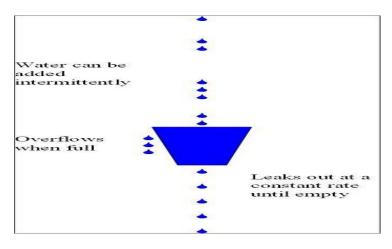


Figure 6. Bucket leaky algorithm

The bucket leaky pseudocode is as follows,

- 01 earches=5
- 02 per_second=60
- 03 current_allowed=searches
- 04 last_check = time()
- 05 while process(search_terms):
- 06 # we determine how many seconds since the last search
- 07 time_now=time()
- 08 time_passed = time_now checked_at now
- 09 # and set when our last search was
- 10 checked_at=time_now
- 11 # add tokens to bucket:
- 12 current_allowed += time_passed * (searches / per_second)
- 13 # check if we have any tokens
- 14 if (current_allowed > searches):
- 15 # we have reached our max. tokens
- 16 current_allowed = searches
- 17 if (current_allowed < 1):
- 18 #partial token, ignore search



- 19 discard search()
- 20 else:
- 21 # we have at least one token
- 22 do search()
- 23 # and we "spend" the token
- 24 current allowed = current allowed -1

4. Conclusion

In this paper, we have presented the design, implementation, and evaluation of ICTCP to improve TCP performance for TCP in cast in data-center network s. In contrast to previous approaches that used a fine-tuned timer for faster retransmission, we focus 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. Our experimental results demonstrate that ICTCP is effective in avoiding congestion by achieving almost zero timeouts for TCP incast, and it provides high performance and fairness among competing flows. The number of connections becomes extremely large. Switching the receive window between several value. How to handle congestion while sender and receiver are not under the same switch. Use ECN to obtain congestion information.

References

[1].Phanishayee, E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, G. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in Proc. USENIX FAST, 2008. [2].V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, "Safe and effective

fine-grained TCP retransmissions for datacenter communication,"

[3].S.Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. OSDI, 2004.

[4].C.Guo,G.Lu,D.Li,H.Wu,X.Zhang,Y.Shi,C.Tian,Y.Zhang,and S.Lu,"BCube:A high performance, server-centric network architecture for modular data centers," in Proc. ACM SIGCOMM, 2009.

[5].C.Guo,H.Wu,K.Tan,L.Shi,Y.Zhang, and S.Lu,"DCell: Ascalable and fault tolerant network structure for data centers," in Proc. ACM SIGCOMM, 2008.

[6]. The New Data Center - Brocade FIRST EDITION New technologies are radically reshaping the data center TOM CLARK

[7]. NX-OS and Cisco Nexus Switching: Next-Generation Data Center Architectures, 2nd Edition

[8]. Using TRILL, FabricPath, and VXLAN: Designing Massively Scalable Data Centers (MSDC) with Overlays.

A Brief Author Biography

A. Suresh – Completed Bachelor of Engineering in Computer Science and Engineering and presently doing M.Tech(CSE) at SRM University, Ramapuram.

S. Jayakumar – Working as an Assistant Professor in the Department of Computer Science and Engineering at SRM University, Ramapuram.