THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Enhanced ICTCP to Avoid Congestion Control in Wireless Sensor Networks

A. Prema
PG Scholar, M.E Communication Systems
Regional Centre of Anna University, Madurai, Tamil Nadu, India
S. Janardhana Prabhu
Assistant Professor, Department of ECE
Regional Centre of Anna University, Madurai, Tamil Nadu, India

Abstract
TCP throughput collapse is known as Incast. Multiple senders communicate with a single receiver in high bandwidth and low latency networks using TCP. Fast data transmission overfills the Ethernet switch buffer and it leads to TCP timeout. Congestion degrades the performance of the network. Enhanced ICTCP is used to improve the performance of the network and increase the throughput. In this method adjusts the receive window size before the packet loss occurs. The implementation and experiments in our testbed demonstrate that we achieve almost zero timeouts.

Key words*: Enhanced ICTCP, TCP*

1. Introduction

TCP is widely used on the internetwork but its not suitable for many to one traffic patterns.ie multiple servers send the data parallel to the same receiver incast congestion happens When too many packets are present in the subnet, performance degrades. This situation is called congestion. When large number of packets injected in to the subnet by the host is within its carrying capacity. They all are delivered and the number of packets delivered is proportional to the number sent. As traffic increases, the routers are not able to handle large number of packets, so the packets are lost. At very high traffic, performance collapses completely and almost no packets are delivered.

Incast congestion happens when the switch buffer overflows because the network pipe is small. It is not enough to contain all TCP packets injected into the network. The capacity of the network pipe is known as bandwidth delay product (BDP). The network delay consists of three parts:1) The transmission delay and propagation delay, which are relatively constant; 2) The system processing delay at the sender and receiver server, which increases as the window size increases; and 3) The queuing delay at the Ethernet switch. We define the base delay as the RTT observed when they receive window size is one MSS, which covers the first two parts without queuing.

Correspondingly, we have a base BDP, which is the network pipe without queuing at the switch. For the incast scenario, the total BDP consists of two Parts: the base BDP and the queue size of the output port on the Ethernet switch. Given that the base RTT is around 100 s with the full bandwidth of Gigabit Ethernet, the base BDP without queuing is around 12.5 kB (s Gb/s). Considering that the Ethernet packet size is 1.5 kB, the base BDP only holds around eight TCP data packets.

Our idea is to perform incast congestion avoidance at the receiver side by preventing incast 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 adjust the receive window size of each TCP connection, so the aggregate burstiness of all the synchronized senders are kept under control. We call our design Enhanced Incast congestion Control for TCP (ICTCP).

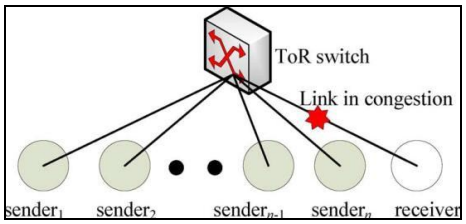

Figure 1: Scenario of incast congestion in data-center networks, where multiple (n) TCP senders transmit data to the same receiver under the same ToR switch

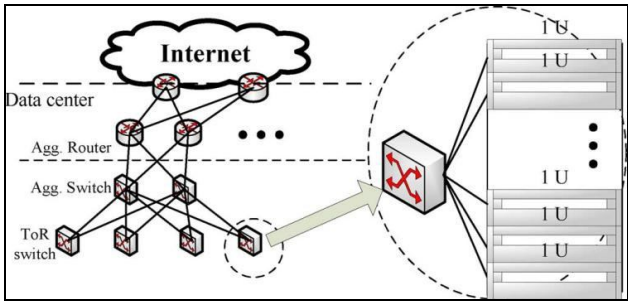

Figure 2: Data-center network and a detailed illustration of a ToR switch connected to multiple rack-mounted servers

The receive window should be small, it is suitable to avoid incast congestion and the receive window is large, it is good for non incast cases. A well-performing throttling rate for one incast scenario may not be a good fit for other scenarios due to the dynamics of the number of connections, traffic volume, network conditions, etc. This paper addresses the above challenges with a systematically designed Enhanced ICTCP. Our implementation naturally supports virtual machines; it is widely used in data centers. This choice removes the difficulty of obtaining the real available bandwidth after virtual interfaces multiplexing.

2. Problem Description

Congestion control is the key problem for Where $\alpha \epsilon [0,1]$ is a parameter to absorb potential oversubscribed bandwidth during window adjustment. Wireless sensor networks. The curious properties of a multi-hop channel are not handled by the standard transport control protocol. The shared nature of the wireless channel and the numerous changes of network topology are the significant challenges. Previous approaches have been projected to conquer these difficulties. The wireless networks are analogous data rates hence the packet delivery requirements may have different. Due to congestion the packets are dropped. The proposed enhanced ICTCP adjusts the window size in the receiver side which reduces the congestion and improves the performance of the network.

3. Enhanced ICTCP Algorithm

Enhanced ICTCP afford a congestion control algorithm based on receive window. All the low RTT (Round Trip Time) TCP connections of the receive window are collectively adjusted to control throughput on incast congestion. The following section describes the procedure for setting the receive window of TCP connection.

3.1. Control Trigger

Available Bandwidth There is one network interface interface on a receiver and define the corresponding symbols.

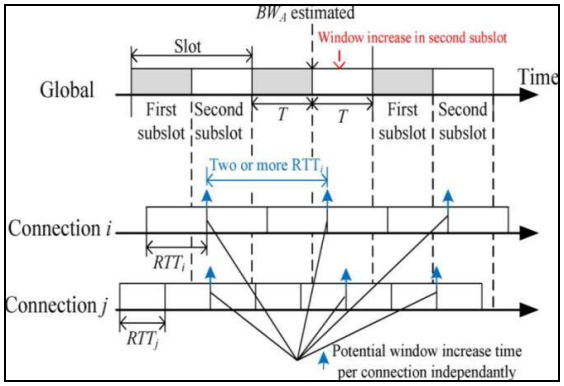

Figure 3: Slotted time on global (all connections on that interface) and two arbitrary TCP connections i/j are independent

Our algorithm can be applied where the receiver has multiple interfaces, and the connections on each interface should perform our algorithm independently

Assume C is the link capacity of the interface on the receiver server, the bandwidth of the total incoming traffic BW_T observed on that interface. It includes all the type of packets like broadcast, multicast, unicast of UDP or TCP etc. Available bandwidth BW_A on that interface as

$BW_A = \max(0, \alpha * C - BW_T)$ (1)

A larger α (closer to 1) indicates the need to more conservatively constrain the receive window and higher requirements for the switch buffer to avoid overflow; a lower α indicates the need to more aggressively constrain the receive window, but throughput could be unnecessarily throttled. In all of our implementations and experiments, we have a fixed setting of $\alpha = 0.9$ .In Enhanced ICTCP, we use available bandwidth BW_A as the quota for all incoming connections to increase the receive window for higher throughput. Each flow should estimate the potential throughput increase before its receive window is increased. Only when there is enough quota (BW_A) can the receive window be increased, and the corresponding quota is consumed to prevent bandwidth oversubscription.

To estimate the available bandwidth on the interface and provide a quota for a later receive window increase, we divide the time into slots. Each slot consists of two subslots of the same length. For each network interface, we measure all the traffic received in the first subslot and use it to calculate the available bandwidth as a quota for window increase on the second subslot. The receive window of any TCP connection is never increased at the first subslot, but may be decreased when congestion is detected or the receive window is identified as being oversatisfied,

In fig.3, the arrowed line marked by global denotes the slot allocation for available bandwidth estimation on a network interface, the first subslot is marked in gray. During the first subslot none of the connections receive windows can be increased. The second subslot is marked in white in Fig. 3. In the second subslot, the receive window of any TCP connection can be increased, but the total estimated increased throughput of all connections in the second subslot must be less than the available bandwidth observed in the first subslot. Note that a decrease of any receive window does not increase the quota, as the quota will only be reset by incoming traffic in the next first subslot.

*3.2. Per-Connection Control Interval: 2*RTT*

In Enhanced ICTCP, each connection adjusts its receive window only when an ACK is sending out on that connection. No additional pure TCP ACK packets are generated solely for receive window adjustment, so that no traffic is wasted. For a TCP connection, after an ACK is sent out, the data packet corresponding to that ACK arrives one RTT later. As a control system, the latency on the feedback loop is one RTT for each TCP connection, respectively. Meanwhile, to estimate the throughput of a TCP connection for a receive window adjustment, the shortest timescale is an RTT for that connection. Therefore, the control interval for a TCP connection is 2*RTT in ICTCP, as we need one RTT latency for the adjusted window to take effect, and one additional RTT to measure the achieved throughput with the newly adjusted receive window. Note that the window adjustment interval is performed per connection. We use connections i and j to represent two arbitrary TCP connections in fig.3 to show that one connection's receive window adjustment is independent from the other.

The relationship of subslot length T and any flow's control interval is as follows: Since the major purpose of available bandwidth estimation on the first subslot is to provide a quota for window adjustment on the second subslot, length T should be determined by the control intervals of all active connections. The changed throughput of any connection is with its RTT, and thus should be with the RTT to represent the changes in available bandwidth. We use a weighted average RTT of all TCP connections as T, ie

The weight w_i normalized traffic volume of connection i that has updated RTT_i in the last time period T.In Fig. 3, we illustrate the relationship of two arbitrary TCP connections i/j with $RTT_{i/j}$ and the system estimation subinterval T. Each connection adjusts its receive window based on the observed RTT. The time it takes for a connection to increase its receive window is marked with an up arrow in Fig. 3. For TCP connection i, if ―now‖ is in the second global subslot and the elapsed time is larger than 2*RTTi since its last receive window adjustment, it may increase its window based on the newly observed TCP throughput and current available bandwidth. Note the RTT of each TCP connection is drawn as a fixed interval in Fig. 3.

3.3. Window Adjustment on Single Connection

For any ICTCP connection, the receive window is adjusted based on its incoming measured throughput and its expected throughput (denoted as b^m and b^e). The measured throughput represents the achieved throughput on a TCP connection and also implies the current requirements of the application over that TCP connection. The expected throughput represents our expectation for the throughput on that TCP connection if the throughput is only constrained by the receive window.

Our idea for receive window adjustment is to increase the receive window when the ratio of the difference between measured and expected throughput over the expected one is small, and to decrease the receive window when the ratio is large. A similar concept has previously been introduced in TCP Vegas, but it uses the throughput difference instead of the ratio of throughput difference, and it is designed for the congestion window on the sender side to pursue available bandwidth. ICTCP window adjustment sets the receive window of a TCP connection to a value that represents its current application's requirements. An oversized receive window is a hidden problem as the throughput of that connection may reach the expected one at any time, and the traffic surge may overflow the switch buffer, a situation that is hard to predict and avoid.

The measured throughput of connection i is obtained and updated for every RTTi , where RTTi is the RTT of connection i. for every RTTi on connection i, we obtain a sample of current throughput, denoted as b_i^s, calculated as the total number of received bytes divided by the time interval RTTi . We smooth the measured throughput using the exponential filter as

$$b_{i,new}^m = \max(b_i^s, \beta * b_{i,old} + (1-\beta) * b_i^s \quad (2)$$

β is the exponential factor, and the default value of β is set to 0.75 . Note that the max procedure here is to update quickly if the receive window is increased, especially when the receive window is doubled. The expected throughput of connection is obtained as

$$b_i^e = \max(b_i^m, \frac{rwnd_i}{RTT_i}) \quad (3)$$

Where rwndi is the receive window of connection i. We have the max procedure to ensure $b_i^m \leq b_i^e$. We define the ratio of throughput difference d_i^b as the ratio of the difference of the measured and expected throughput over the expected one for connection i

$$d_i^b = (b_i^e - b_i^m)/b_i^e \quad (4)$$

By definition, we have $b_i^m \leq b_i^e$ thus $d_i^b \in [0,1]$.
We have two thresholds γ1 and γ2 (γ2>γ1) to differentiate three cases for receive window adjustment:

1) $d_i^b \leq \gamma 1$ or $d_i^b \leq MSS_i / rwnd_i$ increases the receive window if it is in the global second subslot and there is enough quota of available bandwidth on the network interface; decreases the quota correspondingly if the receive window is increased.

2) $d_i^b > \gamma 2$: Decrease the receive window by 1 MSS3 if this condition holds for three continuous RTT. The minimal receive window is 2*MSS.

3) otherwise, keep the current receive window

The available bandwidth calculated at the end of the first subslot is used for the quota of the second subslot right after the first one. The potential throughput increase of connection i is estimated as the increase in the receive window divided by RTTi The quota is consumed by first come first service (FIFS) determined by the order of ACKs send on the second subslot. In all of our experiments, we had γ1=0.1 and γ2=0.5. Note that we set γ2=0.5 to conservatively decrease the receive window from the ratio of measured and expected throughput obtained in our experiments for greedy connections. Similar to TCP's congestion window increase at the sender, the increase of the receive window on any ICTCP connection consists of two phases: slow start and congestion avoidance. If there is enough quota in the slow start phase, the receive window is doubled, while it is enlarged by at most one MSS in the congestion avoidance phase. A newly established or prolonged idle connection is initiated in the slow start phase. Whenever the second and third conditions are met, or the first condition is met but there is not enough quota on the receiver side, the connection goes into the congestion avoidance phase.

3.4. Fairness Controller for Multiple Connections

When the receiver detects that the available bandwidth BW_A has become smaller than the threshold, ICTCP starts to decrease the receiver window of the selected connections to prevent congestion. Considering that multiple active TCP connections typically work on the same job at the same time in a data center, we have sought a method that can achieve fair sharing for all connections without sacrificing throughput. Note that ICTCP does not adjust the receive window for flows with an RTT larger than2 ms, so fairness is only considered among low- latency flows.

In our experiment, we decrease the receive window for fairness when $BW_A < 0.2C$. This condition is designed for high-bandwidth networks, where link capacity is underutilized most of the time. If there is still enough available bandwidth, we argue that the requirement of better fairness is not strong considering the potential impact on achieved throughput when decreasing the receive window. The purpose of the 0.2C gap is to leave enough room for other flows to increase their receive window, and should be larger than the increased throughput when they receive window of a flow is increased by 1 MSS. We adjust the receive window to achieve fairness for incoming TCP connections with low latency as follows:

1) For a window decrease, we cut the receive window by 1 MSS, 3 for some selected TCP connections. We select those connections that have a receive window larger than the average window value of all connections.
2) For a window increase, this is automatically achieved by our window adjustment as the receive window is only increased by 1MSS during congestion avoidance. In principle, the receive window decrease only happens when the available bandwidth on that interface is small. Furthermore, the connection with the larger receive window is decreased slightly to achieve fairness. If all connections happen to have the same receive window, then none of them decrease the receive window.
TCP uses additive increase multiplicative decrease (AIMD) to achieve both stability and fairness among flows sharing the same bottleneck. However, MD happens only when packet loss is observed for a TCP connection. In DCTCP, packet loss during incast congestion due to buffer overflow is largely eliminated. Consider a scenario in which some flows have a large receive window (because they started earlier) while others have a much smaller receive window (because they started later), and the link capacity is almost reached. In this case, none of the connections can increase their receive windows as there is not enough available bandwidth. This situation may persist as long as there is no packet loss so that unfairness between flows becomes an issue. Therefore, we propose slightly reducing the receive window for flows that have a larger receive window, and the throughput of all TCP connections should smoothly converge.

4. Results & Discussion

- Comparison between TCP and Enhanced ICTCP

4.1. Average Delay

In TCP the delay was high, but in Enhanced ICTCP average delay reduced. Here, the average delay plot reveals that the TCP delay is linear in nature whereas the Enhanced ICTCP plot is comparatively not linear in nature when compared with TCP. The Enhanced ICTCP is initially not linear and then it holds the property of linearity as like TCP.

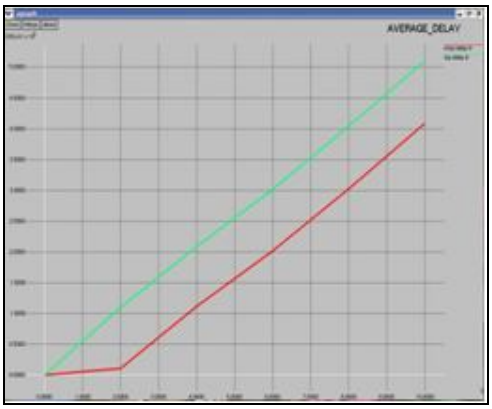

Figure 4: Average Delay For TCP And Enhanced ICTCP

4.2. Energy_Spent

Here in the energy spent plot, both the existing (TCP) and (Enhanced ICTCP), the plots initially increase abruptly and then both the plots remain constant. In the existing TCP, the average energy spent is relatively larger when compared with the proposed Enhanced ICTCP algorithm. So, it is obvious that the efficiency of the Enhanced ICTCP is much better than the existing TCP

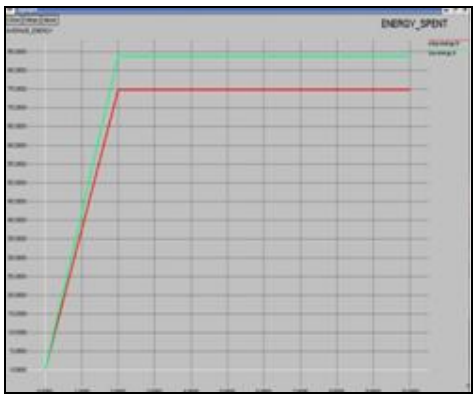

Figure 5: Energy_Spent For TCP And Enhanced ICTCP

4.3. Throughput

Throughput is defined as the number of successive packets delivered per second. Here we increase the Throughput in Enhanced ICTCP compared to TCP. The number of successful packets delivered per
second is improved to a great deal in the proposed Enhanced ICTCP algorithm when compared with TCP.

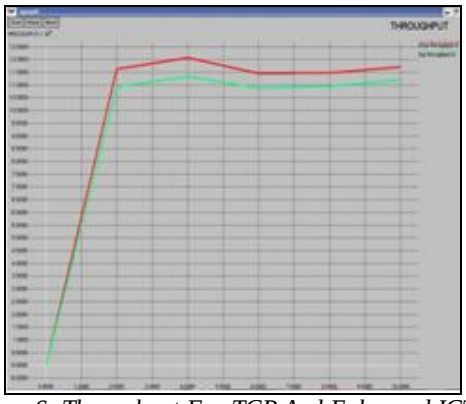

Figure 6: Throughput For TCP And Enhanced ICTCP

4.4. Packet Delivery Ratio

The ratio of the number of packets delivered to the destination to the number of packets transmitted is called packet delivery ratio. In Enhanced ICTCP the packet delivery ratio is very high when compared to TCP. Packet delivery ratio is increased to a significant value in Enhanced ICTCP whereas it is of meager value in case of the existing algorithm TCP.

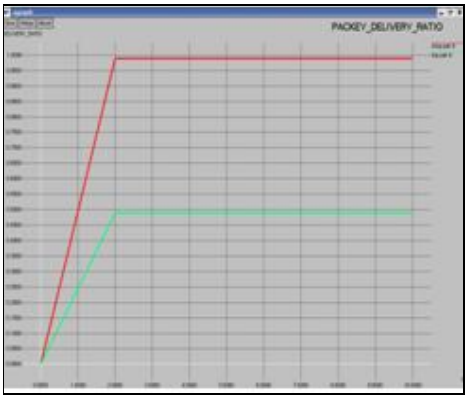

Figure 7: Packet Delivery Ratio For TCP And Enhanced ICTCP

5. Conclusion and Future Work

The proposed work using the Enhanced ICTCP algorithm reduces the congestion to a great deal in comparison with the existing TCP algorithm. In this network, synchronized servers send data to the same receiver in parallel which induces TCP Incast congestion. It happens in high bandwidth and low latency networks. Many data center applications such as map reduce and search, this many to one traffic is common.TCP Incast congestion may severely degrade their performance. That Enhanced ICTCP algorithm is effective in avoiding congestion by achieving almost zero timeouts for TCP Incast, and it provides high performance and fairness among competing flows. Due to congestion there is a chance to loss the high priority packets. in future the packets are prioritized as low and high priority

6. References

1. Al-Fares.M, Loukissas.A, and Vahdat.A 2008, ―A scalable, commodity data center network architecture,‖ in Proc. ACMSIGCOMM, pp. 63–74.
2. Alizadeh.M, Greenberg.A, Maltz.D, Padhye.J, Patel.P, Prabhakar.B, Sengupta.S, and Sridharan.M 2010, ―Data center TCP (DCTCP),‖ in Proc. SIGCOMM, pp. 63–74.
3. Braden.R Oct. 1989, ―Requirements for internet hosts—Communication layers,‖ RFC1122.
4. Brakmo.L and Peterson.L Oct. 1995, ―TCP Vegas: End to end congestion avoidance on a global internet,‖ IEEE J. Sel. Areas Commun., vbol. 13, no. 8, pp. 1465– 1480.
5. Chen.Y, Griffith.R, Liu.J, Katz.R, a and Joseph .A 2009, ―Understanding TCP incast throughput collapse in datacenter networks,‖ in Proc. WREN, pp. 73–82.
6. Dean.J and Ghemawat .S 2004, ―MapReduce: Simplified data processing on large clusters,‖ in Proc. OSDI, p. 10.
7. Guo.C Wu.H,Tan,.K,Shi.L,,Zhang.Y, and Lu .S 2008, ―DCell:Ascalable and fault tolerant network structure for data centers,‖ in Proc. ACM SIGCOMM, pp. 75–86.

8. Guo.C, Lu.G ,Li.D, Wu.H, Zhang.X, Shi.Y, Tian.C, Zhang.Y, and Lu.S 2009, ―BCube: A high performance, server-centric network architecture for modular data centers,‖ in Proc. ACM SIGCOMM, pp. 63–74.
9. Jacobson.V, Braden.R, and Borman.D May 1992,―TCP extensions for high performance,‖ RFC1323.
10. Kandula.S, Sengupta.S, Greenberg.A, Patel.P, and Chaiken.R 2009, ―The nature of data center traffic: Measurements & analysis,‖ in Proc. IMC, pp. 202–208.
11. Krevat.E, Vasudevan.V, Phanishayee.A, Andersen.D, Ganger.G, Gibson.G, and Seshan .S W 2007, ―On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems,‖ in Proc. Supercomput., pp. 1–4