Simulative Analysis of High Speed TCP Variant using OMNet++

Navdeep Singh[1] and Rajesh Kumar [2]

[1]*M.Tech Scholar, BGIET, Sangrur*
navdeepsingh84@gmail.com
[2]*Assist. Prof., BGIET, Sangrur*
rajeshkengg@gmail.com

Abstract

The demand for fast and long distance network increasing day by day. However, TCP is the dominant transport protocol of today, it does not meet this demand because it fails to fully utilize the network capability. A large variant of TCP have been proposed to improve the connections throughput. In this paper, we propose a novel Compound TCP (CTCP) approach using OMNET++ and evaluate with the regular TCP Reno. On the basis of some parameters, result shows how CTCP utilize network capacity in any environment to provide the better throughput and fairness. The results have been shown in support of the proposed method.

Keywords: CTCP, OMNET++, Reno, TCP.

1. INTRODUCTION

TCP is a connection oriented protocol which provides a reliable, ordered of packets between computers connected on the internet [1]. TCP senders add sequence numbers to packets and TCP receivers acknowledge packet receptions. To ensure that packet won't be loss due to corruption during transmission, unacknowledged TCP packets are retransmitted based on information about successfully received sequence numbers, and in some cases, timeouts. The various TCP congestion control algorithms aim to maximize the throughput while avoiding congestion in the connection flow. All TCP congestion control algorithms have several features in common. The TCP sender keeps a dynamic congestion control window that limits the number of outstanding transmitted packets before an acknowledgement need to arrive from the receiver. Most TCP connections operates in two modes, slow start for which the window size is incremented rapidly, and a congestion avoidance mode for which the window size is incremented in a way to avoid congestion over the link. It is rather vague how one can say a TCP congestion avoidance algorithm is good, however most TCP algorithm tend utilize the network throughput whilst maintaining a fairness among other TCP flows [7]. In this paper, we focus on the implementation of Compound TCP and analysis with regular TCP Reno [3].

2. TCP COMPOUND

Compound TCP is a congestion control protocol developed by Microsoft in an attempt to maximize performance. Specifically it aims to maximize bandwidth utilization. When multiple agents send packets on a limited bandwidth network, the network gets over congested, buffers in the network start to reach maximum capacity and eventually lead to dropping packets.TCP connection on the end nodes, being reliable, recognizes the dropped packets when the time expire and starts retransmitting these packets again, which only makes the problem worse as the network gets more and more congested. Therefore TCP needs a mechanism to control congestion.

ISSN: 2456-1118

3. PROBLEM FORMULATION

3.1 Packet loss based congestion control- One of the basic mechanisms used to achieve this goal is to react upon dropped packets. Dropped packets can mean the network can't handle the current transmission rate, especially when multiple packets get dropped successively. So in this mechanism once the TCP connection recognizes a dropped packet, it lowers the congestion window, i.e. the amount of bytes that can be transmitted to the network before receiving acknowledge from the receiver. And in the opposite direction, after a series of acknowledges is received from the receiver the transmitter infers that it is not utilizing the maximum bandwidth available to it starts to gradually increase the congestion window. This mechanism is implemented in famous TCP versions like TCP Rheno [3]. While it introduces a partial fix to the problem described above, it has a few drawbacks that motivate us to look for improvements.

Two significant drawbacks are:

- Waiting for packets to start getting dropped to start tuning the congestion window is highly costly. Especially, on Big Fat Networks– high product network.
- After dropping packets the congestion window is dropped aggressively and takes a significant amount of time to return to a high utilization mode.

3.2 RTT based congestion control- Another mechanism to achieve congestion control is to react upon an increase in the round trip time of the packets being transmitted. It is a known fact that once the network start to get over congested and queues start to fill up the average amount of time each packet spends in a queue increases. Which translates to an increase in the total time from send to acknowledgement this packet takes.

So in this mechanism the sender TCP client will measure the round trip time of each packet it sends. Once an increase in RTT is realized, the sender will gently decrease the congestion window in a hope to prevent the congestion from reaching a level where packets start to get dropped. Once the RTT starts to get back to lower values indicating shorter queues, the sender can increase the congestion window again to achieve maximum utilization [6].

This mechanism manages to prevent packet loss which increases the efficiency of the network and the total bandwidth. But it suffers from one significant issue which prevents mainstream TCP implementation from adopting this mechanism. And the issue is this mechanism relies on the good behavior of other users of the network. i.e. the TCP agent will lower its sending rate expecting other agents to do the same in order to prevent packet loss [8]. In reality though other network users might use the packet drop based congestion control [9]. That will keep sending at a high rate until packets starts getting dropped. In this case, delay based connection will receive an unfair share of the network bandwidth. This issue means delay based congestion control can't be relied upon to implement a robust TCP connection for mainstream use.

4. PROPOSED WORK

The issues described above paved the way for this direction. This is the goal of this congestion control mechanism. In this mechanism TCP sender agent will consider two different measurements to calculate optimal window size. This window size will composed – hence the name- of a delay based component and a packet drop based component.

ISSN: 2456-1118

The idea is that the delay based component will offer its ability to fine tune the congestion window in advance before dropping packets in order to achieve high reaction times and optimal efficiency [9]. While the packet drop component will insure that the connection gets its fair share of the bandwidth even when it has competition against unfriendly connections.

5. METHODOLOGY

In this implementation the congestion window of the connection was the sum of two congestion windows each one of those was calculated independently. The first one according to packet drop rates and the other according to packet round trip time rates.

Now we will review in detail the handling of this connection upon receiving expected acknowledgments, receiving duplicate acknowledgments and receiving time outs.

5.1 Packet drop based component:

a. **Upon expected data acknowledge:**
upon each received acknowledgment the drop based is incremented by one.
b. **Upon duplicate data acknowledge:**
3 duplicate data acknowledgments are considered a dropped packet, therefore the packet dropped based component is cut in half.
c. **Upon a time out expiration:**
Expired time out is considered a severe congestion state and the congestion window is reset to the starting value of slow start.

5.2 Delay based component:

a. **Upon expected data acknowledge:** Upon each received acknowledgment the TCP sender will calculate the Round Trip Time (RTT) of the acknowledged packet (by subtracting the transmission time and the acknowledge receive time of the acknowledged sequence number). Then it will use this RTT to update a smoothed RTT, which is an average of the last few received acknowledgments. This smoothed RTT is used instead of the actual RTT to prevent abnormal RTT from severely affecting the flow. Other than that the Transmitter maintains a BaseRTT which is the minimal RTT observed on this connection [7]. Then this BaseRTT is compared to the smoothed RTT to determine the state of the network according to this formula:

$$Expected := \frac{window}{BaseRTT} \quad (1)$$

$$Actual := \frac{window}{Smoot\ hedRTT} \quad (2)$$

$$Diff := (Expected - Actual) * BaseRTT \quad (3)$$

This Diff value is an indication of the congestion state of the network. It is compared to a given value Gamma (calculated by experiments) and the delay window is updated as follows [7]:

$$cwnd(t+1) = \begin{cases} cwnd(t) + (\alpha \cdot win(t)^{k} - 1)^{+}, \text{if } diff < \gamma \\ (cwnd(t) - \zeta \cdot diff)^{+}, \text{if } diff \geq \gamma \\ (win(t) \cdot (1-\beta) - cwnd/2)^{+}, \text{if loss is detected} \end{cases} \quad (4)$$

Where the parameters α, k, η, β are also fine-tuned by experiments, and can be configured in each implementation.

ISSN: 2456-1118

b. **Upon duplicate data acknowledge:** duplicate acknowledgments indicate packet loss, which requires drastic decrement in the delay based window (see equation above). Since no packet drop is expected in this mechanism.
c. **Upon a time out expiration:** Upon expired time out the delay based component becomes zero and the feature is disabled.

6. SIMULATION SETUP

To better understand the performance of the new implemented algorithms we refer to [5] and try to recreate the simulation setup given by the paper. In this paper, the simulation setup consists of two senders and two receivers. The simulation topology is a dumbbell topology as given by the following figure 1 [2].

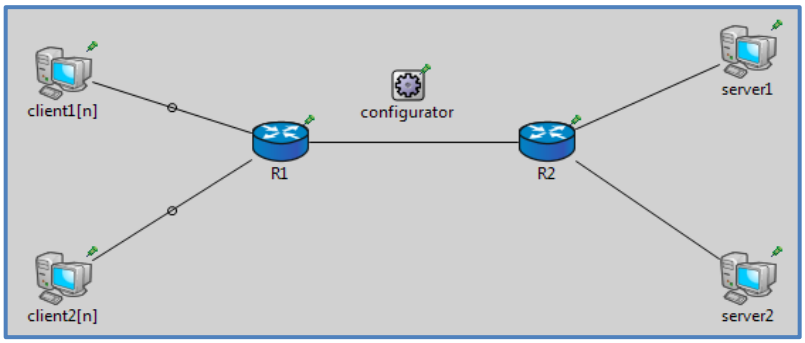

Fig. 1: Dumbell simulation for evaluating the performance of the TCP congestion control algorithims

The bottleneck link has a transfer rate of value 10Mbps, whilst the other links has transfer rate of 100Mbs. The propagation delay is set to be 100ms [2]. The congestion control protocols for high-speed networks are evaluated based on the following properties: congestion window over time, throughput, and fairness.

7. RESULTS

7.1 Congestion window over time

Simulation 1 – TCP New Reno:

As can be seen from the figure 2, for New Reno algorithm both flows converge to the same value. In addition, the window doesn't grow very aggressively thus when compared with other flows, we see that the cwnd of new reno doesn't reach high cwnd in overall.

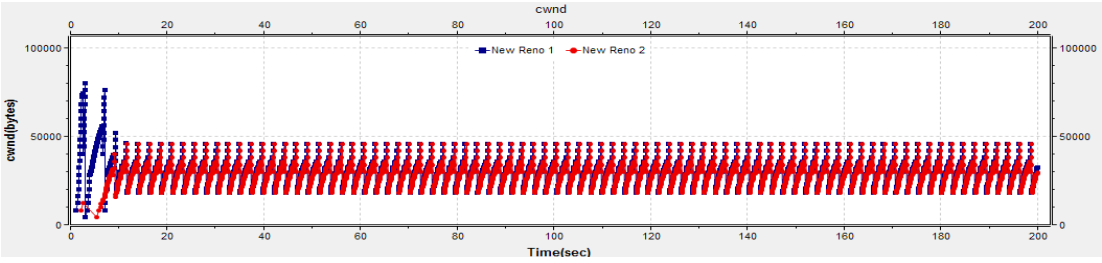

Fig. 2: Congestion window over time of Two New Reno TCP flows

ISSN: 2456-1118

Simulation 2 – TCP Compound:

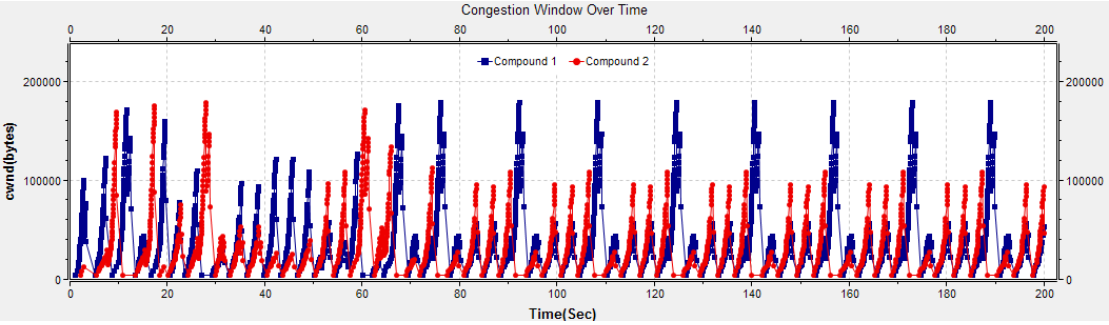

Fig. 3: Congestion window overtime of Two TCP Compound flows

From the figure 3, we see that both clients have the exponential behavior of Compound. Due to the low data rate in the simulation, and since both clients have exponential behavior that causes the link to reach its maximum throughput which causes drops later.

In addition, we see that Compound reach congestion window of the same order, unlike New Reno which has relatively small congestion window.

7.2 Throughput

Simulation 1 – TCP New Reno:

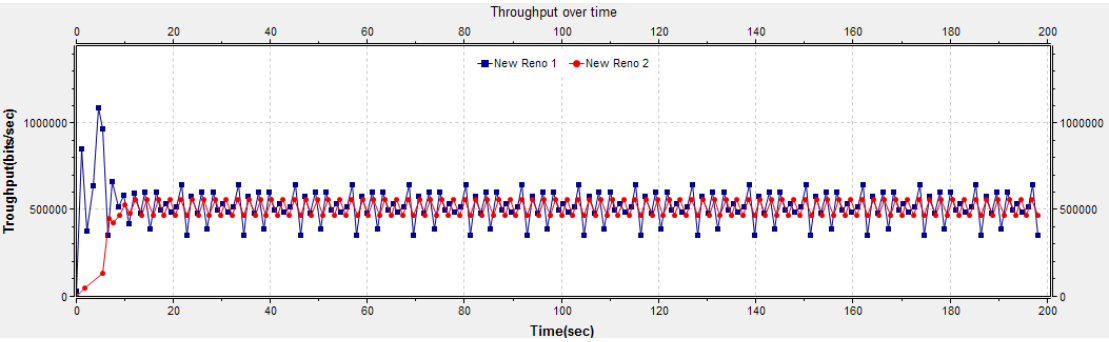

Fig. 4: Throughput overtime of two TCP New Reno Clients

In the figure 4, we see that for New Reno algorithm, the both flows maintain a consistent throughput which is exactly half the bottleneck data rate.

The average throughput is:

New Reno 1 – 520Kbs

New Reno 2 – 501Kbs

ISSN: 2456-1118

Simulation 2 – TCP Compound:

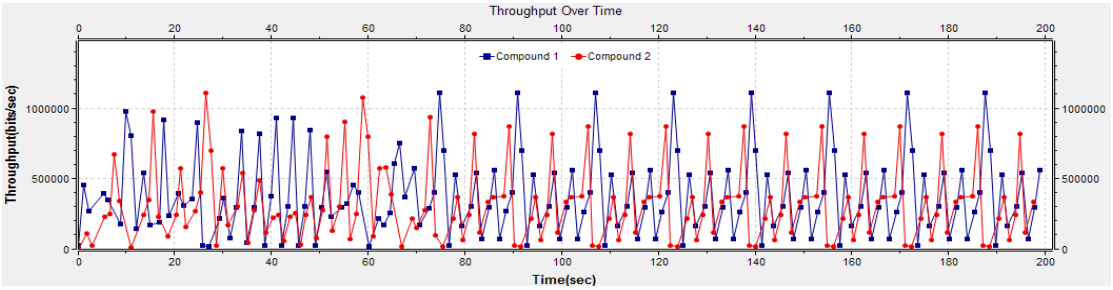

Fig. 5: Throughput over time of Two Compound TCP clients

We see from the figure 5, each client reaches the maximum throughput for only short time. This is because the two clients are in exponential growth, and since compound evaluates the congestion of the network, when one client reaches the maximum throughput it causes both clients to drop.

The average throughput is as follow:

Compound 1 – 380Kbps

Compound 2 – 320 Kbps

7.3 Fairness

In terms of fairness we see that compound achieve higher congestion window, which could lead to a higher resource utilization comparing to the NewReno client.

In the specific topology above we got that the average NewReno throughput is actually higher than compound. But we believe the reason for that is the small buffers in the network above which meant most packets sent by compound in a high congestion window state were dropped and therefore didn't contribute to the average throughput.

And after the dropping the congestion window of these flavors goes to slow start so we get lower performance than NewReno that kept its congestion window consistent.

That said, on a topology with buffers with higher frame capacity, the result would not be the same. As the network can handle the high windows and compound will have more than their fair share of the bandwidth.

8. CONCLUSION

We saw from the congestion window over time graphs, that TCP Compound reaches much higher congestion window due to the aggressive increment of the congestion window. The design of the newly implemented algorithm (TCP Compound) is purposed for High speed networks, this could be seen from the congestion window behavior as stated before.

ISSN: 2456-1118

All the simulation done in this project, use links with low data rate. We have chosen this because in OMNeT++ we couldn't increase the advertised window because of a limitation imposed in the TCP implementation [2].

However, we seen that when there are two TCP flows of the same congestion control algorithm, both clients reach the same throughput. However, due to the exponential behavior [4] of TCP compound which causes drops to the TCP flows, we saw that TCP Compound had the lowest throughput. Whilst in TCP New Reno, which keeps a consistent behavior, we saw that the throughput of the New Reno flows were higher. In overall, TCP Compound reaches higher throughput, but due to the drop in the congestion window, the throughput drops to almost 0. However, in TCP new Reno, it keeps a constant throughput over time.

In terms of fairness, we saw that when the two tcp flows use the same congestion control window, both clients have the same average throughput. So fairness is achieved. In the simulation, we saw that New Reno got better throughput due to the "fight over link" behavior of the Compound, which cause drops to be made to their congestion window.

REFERENCES

[1]. K. F. James and K. W. Ross, Computer Networking : A top-down approach, PEARSON, 2013.
[2]. "OMNeT++ Discrete Event Simulator," OMNeT++, [Online]. Available: https://omnetpp.org/.
[3]. "C/C++ Reference: function cbrt," cplusplus, [Online]. Available: http://www.cplusplus.com/reference/cmath/cbrt/.
[4]. "Newton's method," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Newton%27s_method.
[5]. H. Sangtae, Y. Kim, L. Le, I. Rhee and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants," Department of Computer Science North Carolina State University , Department of Computer Science and Engineering University of Nebraska.
[6]. K. Tan and J. Song, "A Compound TCP Approach for High-speed and Long Distance Networks".
[7]. D. J. Leith, A. L. H. Lachlan, T. Quetchnebach and K. Lavi, "Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms," Hamilton Institute, Ireland ; Caltech, Pasadena, CA, USA.

Authors

Navdeep Singh is M.Tech (Research Scholar) at Bhai Gurdas Institute of Engineering and technology, Sangrur. He is having 7+ years of teaching experience. During his tenure of services he has published national and international Research papers in the area of Computer science and Engineering (Research Area-Networking).

Rajesh Kumar is Assistant Professor at Bhai Gurdas Institute of Engineering and Technology, Sangrur. He is having 10+ years of teaching experience. During his tenure of services he has published national and international Research papers in the area of Computer science and Engineering. Currently, he is pursuing Ph. D. degree in the area of Computer Networks.

ISSN: 2456-1118