Improved TCP congestion control algorithms for HTTP/2

Praveen Kumar Saminathan
ID#: 110739913
psaminathan@cs.stonybrook.edu

Vinothkumar Raja
ID#: 110422174
vinraja@cs.stonybrook.edu

Aathira Prabhakar
ID#: 110749512
aprabhakar@cs.stonybrook.edu

Dept. of Computer Science, Stony Brook University

***Abstract*: In this paper we evaluate the performance of TCP CUBIC under high packet loss. CUBIC, an enhanced version of BIC, is the default TCP variant in Linux environment and is widely used in modern web servers today. TCP Reno is another popular congestion control algorithm. In this paper, we evaluate the performance of above mentioned two algorithms with HTTP/2 (SPDY) under packet loss, delay and other bad network conditions and attempt to improve their performance by modifying the congestion window back off behavior. In the new variants, the congestion window back-off is not as aggressive as it is in the original version. This allows the algorithms to work well with HTTP/2 under poor network conditions. We also evaluate RTT fairness for TCP CUBIC and perform sensitivity analysis for TCP Reno.**

Keywords: TCP; congestion control algorithm; TCP CUBIC; TCP Reno; HTTP/2; SPDY

I. INTRODUCTION

SPDY is a networking protocol developed at Google which aims to speed up web and provide better web experience to the user. HTTP/2 is the second major version of the HTTP network protocol and it is based on SPDY. HTTP/2 augments HTTP with several speed-related features such as multiplexed requests over single TCP connection, compressed headers, active push of resource to client namely server push and more. Experiments and studies state that HTTP/2 performs better than HTTP/1.1. However, it is found to be detrimental under high packet loss [1]. Studies indicate that under high packet loss, SPDY aggressively reduces the congestion window compared to HTTP/1.1, thereby delaying the packet transfer drastically [1]. This demands need for a better congestion control algorithm that helps SPDY to perform better even under high packet loss. In this paper, we modify the parameters of TCP variants, CUBIC and Reno to help SPDY perform well with the congestion control algorithm.

In TCP Reno, when triple duplicate ACKs are received, it will halve the congestion window, perform a fast retransmit, and enters fast recovery. If a timeout event occurs, it will enter slow-start phase. TCP Reno is effective to recover from a single packet loss, but it still suffers from performance problems when multiple packets are dropped from a window of data.

The default congestion control protocol for TCP, CUBIC modifies the linear window growth function of existing TCP standards to

be a CUBIC function [2]. This feature improves scalability and stability of TCP. The window function in TCP CUBIC is defined in real time and is independent of RTT. During steady state, TCP CUBIC increases the window size insistently when the window is far from saturation point, and then slowly when it is close to saturation point. This feature allows TCP CUBIC to be very scalable when the bandwidth and delay product of network is large, and at the same time, be highly stable and fair to standard TCP flows [2].

The new modified algorithm which we call as CUBIC+ is designed to work well with SPDY. The main scenario where the performance of SPDY hurts is under high packet loss. When there is a packet loss, the congestion window backs off drastically causing the performance of SPDY to deteriorate. This can be improved by adjusting the congestion window to back off less belligerently during an event of packet loss. We also study the RTT fairness with respect to slow networks and networks with varying delay for TCP CUBIC. As per prior studies [3], TCP CUBIC behaves unfair for different network delay times. Further, we conduct a sensitivity analysis for TCP Reno by analyzing congestion window behavior over a range of window reduction factor.

II. RELATED WORK

There are many variants to default TCP CUBIC algorithm aimed at improving performance. One such variant is CUBIC-FIT [5]. CUBIC-FIT presents the idea of delay-based TCP into the TCP CUBIC algorithm framework. CUBIC-FIT can improve throughput performance over wireless networks more than original CUBIC and maintain graceful friendliness with widely deployed plain CUBIC servers.

Another variant aims at improving the RTT fairness of TCP CUBIC [3]. This is done by adjusting the recovery time according to RTT. A new function of RTT that monotonically decreases as RTT increases is introduced to achieve this.

III. PROBLEM DESCRIPTION

Performance of SPDY degrades under high packet loss. From previous studies [1], we infer that performance of SPDY is beneficial in most of the situations. SPDY helps under below conditions, when there are:

- Many small objects under low loss
- Many large objects under low loss

The benefits of SPDY stems from the fact that it uses a single connection. However, use of single connection is found to be detrimental under certain conditions. The usage of single pipe causes SPDY to hurt under high loss. In the event of packet loss, congestion window is reduced more aggressively compared to HTTP/1.1 and this is leads slow packet transfer and degraded performance. There is a need for better and improved congestion window back off technique to perform under packet loss and to maximize benefits of SPDY. To solve this problem, we propose modification to congestion back off technique for TCP CUBIC and TCP Reno to work well with HTTP/2 on the event of packet loss.

IV. APPROACH

Our approach is to study and evaluate TCP congestion control algorithms, CUBIC and Reno for various network conditions. We studied various parameters like RTT, packet loss rate, initial congestion window, number of objects and size of objects [1]. Based on our study and analysis by other researchers [1][3], we introduced changes to current TCP CUBIC algorithm, specifically congestion window back off logic for packet loss and recovery time calculation on packet loss with respect to RTT. For TCP Reno, we performed sensitivity analysis over a range of window reduction factors to understand the congestion window behavior. We evaluated the extent to which these changes improved performance. Detailed explanation of the changes introduced is given in Section 5.

V. SOLUTION METHODOLOGY

Congestion window changes with respect to the congestion back off algorithm for TCP CUBIC and TCP Reno. In TCP CUBIC, congestion window is reduced based on a factor β as shown in Figure 1. *cwnd* is congestion window size, $W_{last\text{-}max}$ is a congestion window size at the last loss and s*sthresh* is the slow start threshold which is called when the TCP detects a loss. In the new modified algorithm CUBIC+, a new back off rate, β' is calculated based on β. *cwnd* and *ssthresh* values are calculated based on the new value, β'.

Packet loss:
begin
  $epoch\_start \longleftarrow 0$
  if $cwnd < W_{last\_max}$ **and** $fast\_convergence$ **then**
    $W_{last\_max} \longleftarrow cwnd * \frac{(2-\beta)}{2}$ .....
  else $W_{last\_max} \longleftarrow cwnd$
  $ssthresh \longleftarrow cwnd \longleftarrow cwnd * (1-\beta)$ .....
end

Figure 1: CUBIC algorithm for Packet Loss

We calculate the new back off rate,

$$\beta' = 1-(1-\beta)/n$$

Here n is set to 6 [1]. Thus, the back-off behavior on the event of congestion would be less aggressive.

The window growth function of TCP CUBIC is as shown in Figure 2. C is a CUBIC parameter, t is the elapsed time from the last window reduction, and variable K is the time to the inflection point and *cwnd* achieves *Wmax* at time K. Therefore, we understand K as time for recovery.

As per previous studies [3], TCP CUBIC algorithm does not consider network delay factor for calculating the recovery time K.

$$cwnd = C(t-K)^3 + W_{\max}$$

$$K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$$

Figure 2: Recovery time calculation on packet loss.

In CUBIC+, we try to improve recovery time calculation by considering RTT. This is done by introducing a monotonically decreasing function of RTT. Hence, RTT fairness between different network with varying delay is maintained.

In TCP Reno congestion control algorithm, *cwnd* is reduced to halve the current congestion window when packet loss occurs. We conduct sensitivity analysis for TCP Reno for various network conditions, by varying congestion window reduction factor from 0.5 to 0.9.

VI. EXPERIMENTS

In this section, we first present the experimental setup and followed by details of experiment.

6.1 Experimental Setup:

Server: The server is a 64-bit machine with 2.70 GHz with Intel Core i5 processor CPU. It runs Ubuntu 14.04 with Linux Kernel 4.2.0. An apache server module, Apache 2.4 is running in the server with Protocol h2 to support HTTP/2. Server hosts a number of webpages created for testing.

Client: The client accesses the webpages hosted at server via web browser. To emulate packet loss and other real network conditions, network tool *netem* is used. Client is connected to the server via a WLAN cable.

Tools: *Netem* network tool is used to emulate the properties of a real network such as packet loss. *Iperf* tool is used to measure bandwidth and throughput in a network. *Tcpdump* tool is used to capture network packets during web page load. *Wireshark* is used to trace and analyze the packets captured using *Tcpdump*.

Webpages: To experiment with different object sizes, four HTML webpages are created with objects of varying sizes, less than 1KB, 1KB-10KB, 10KB-100KB and greater than 1MB. These HTML pages mainly contain image objects.

6.2 Experimenting with webpages:

We consider four different web pages; each containing a set of flag images of different size. Client requests for each web page individually from the server via web browser (Firefox). Packet transfer is captured using *Tcpdump* tool and analyzed using *Wireshark*. The same process is repeated by emulating poor network conditions: inducing packet loss, delay and by changing initial congestion window size. Details on various key parameters in included in Table 1. Any previously cached content from the web page is cleared before each analysis. Throughput (average bits per second) and page load time are the metrics used to evaluate the performance.

Factor	Range
RTT	20ms, 100ms, 200ms
Packet Loss	0, 0.02, 0.06, 0.08, 0.10
Object Size	1KB, 10KB, 100KB, 1MB
Initial cwnd	3, 10, 21, 32

Table 1: Key parameters and value ranges

We first investigate the performance of default algorithms TCP CUBIC and TCP RENO. Later, we make changes to the kernel image as mentioned in section 5 [6] and load the server with the new modified kernel. Linux supports pluggable congestion control algorithm. This feature is used to modify existing congestion algorithm and switch

between different congestion algorithms. Now experiment is repeated for the modified version.

Multiple iteration of the experiment was done for different conditions. We collect network traces at the client. In our experiments, page load time (PLT) is taken as the elapsed time between when the first object is requested and when the last object is received.

VII. EVALUATION AND RESULTS

In this section, we study the performance of our new improved algorithm, CUBIC+ and the effect of varying window reduction factor on TCP Reno.

7.1 CUBIC+ under Packet Loss:

We analyze the performance of the new variant, CUBIC+ (only β factor modified) with default TCP CUBIC under packet loss. Packet loss is varied over a range of zero to ten percentage. Experiment is repeated on varying object sizes as mentioned in Table 1.

We observe that the throughput of webpages with CUBIC+ is better than that of original version. For webpages with larger object size, we see considerable variation in throughput with induced packet loss (Figure 3,4). For medium size objects, we still see improvement in throughput for the modified version but not as contrasting as for larger object sizes (Figure 5). For really small objects (<1KB size), both the algorithms were found to perform same in terms of packet loss.

It was also observed that page load time for webpages using CUBIC+ almost equals that of default CUBIC TCP under low percentage of packet loss. However, as packet loss% increases, CUBIC+ was observed to outperform default TCP CUBIC. Figure 6 shows observed page load time for objects greater than 1MB size.

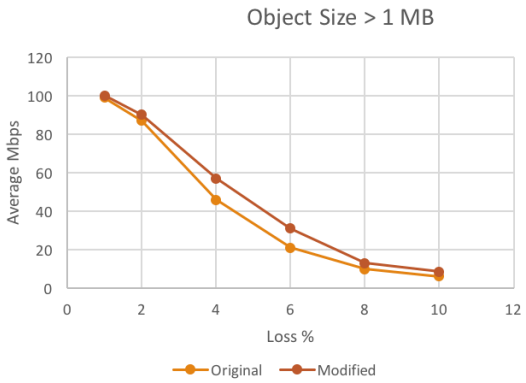

Figure 3: Throughput vs Packet loss % for objects greater than 1MB.

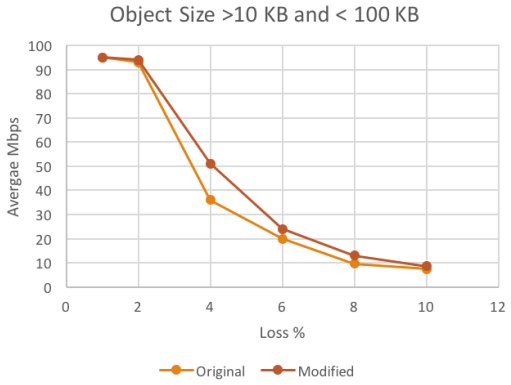

Figure 4: Throughput vs Packet loss % for objects greater than 10 KB but lesser than 100KB.

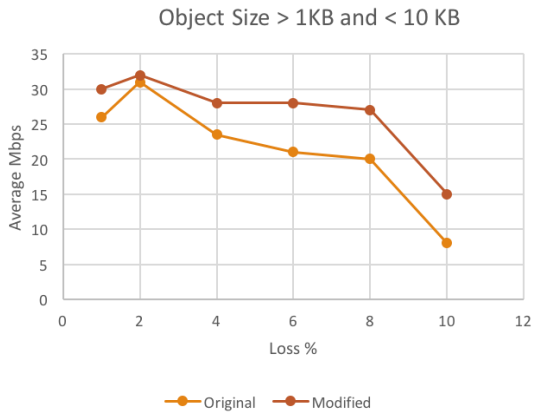

Figure 5: Throughput vs Packet loss % for objects greater than 1 KB but lesser than 10KB.

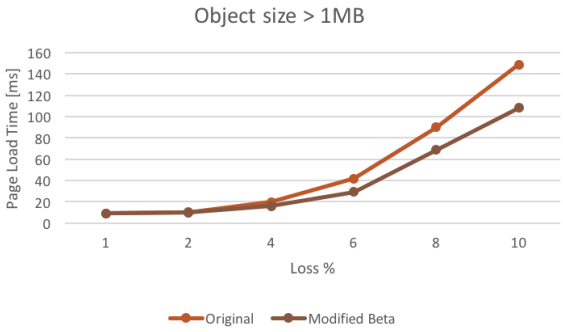

Figure 6: Page Load Time vs Packet loss % for objects greater than 1MB.

7.2 CUBIC+ under delay:

Experiments were conducted to measure the performance of CUBIC+ (with β factor modified and recovery time, K modified) for varying delay factors as mentioned in Table1.

From observation, we infer that throughput of the two modified versions of TCP CUBIC - β factor modification and recovery time, K modification, is close to that of original TCP CUBIC for smaller objects and medium sized objects (Figure 7,8). However, for larger objects, performance of modified version deteriorates. Default CUBIC TCP outperforms both modified versions under varying delay for larger object size (Figure 9).

7.3 CUBIC+ under varying initial congestion window size:

In this section, we evaluate the performance of CUBIC+ (with β factor modified and recovery time, K modified) for varying initial congestion window size as mentioned in Table 1. By default, initial congestion window size is set to 10 in Linux kernel.

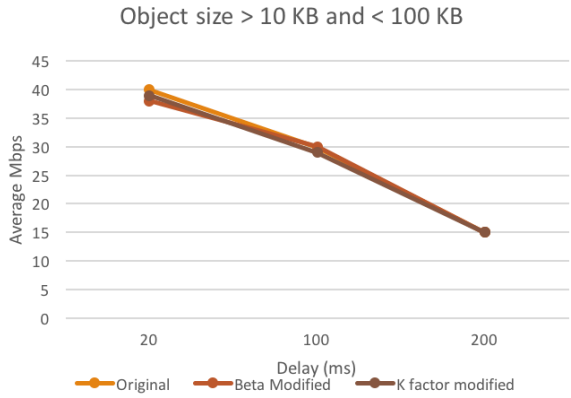

Figure 7: Throughput vs Delay for objects greater than 10KB and lesser than 100KB.

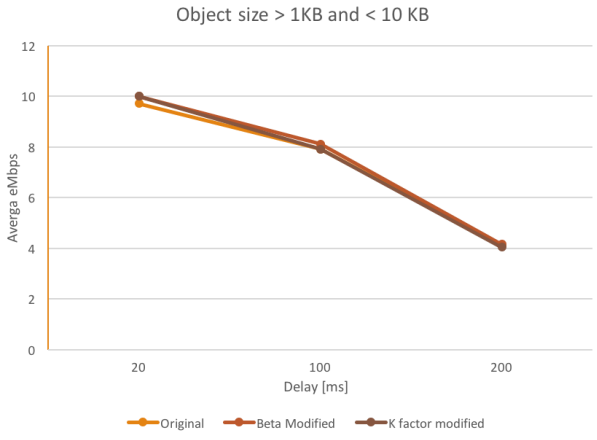

Figure 8: Throughput vs Delay for objects lesser than 10KB.

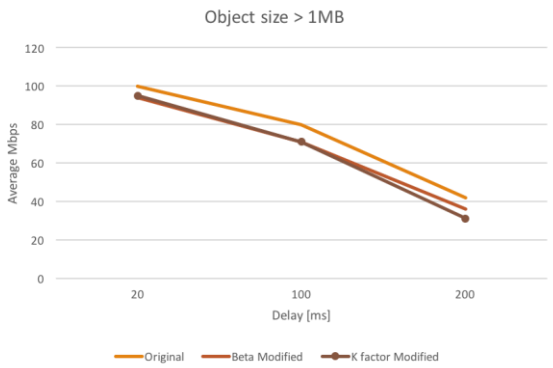

Figure 9: Throughput vs Delay for objects greater than 1MB.

It was observed that initial congestion window has very less role in evaluating the performance of modified algorithms and default TCP CUBIC. All three variants are observed to perform the same under varying values of initial congestion window size (Figure 10,11).

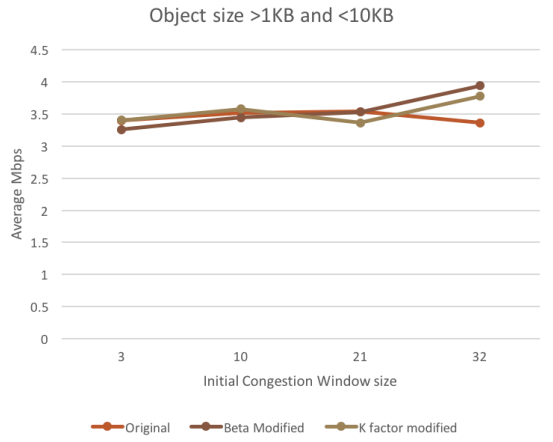

Figure 10: Throughput vs init cwnd for objects greater than 1KB and less than 10KB.

Figure 11: Throughput vs init cwnd for objects greater than 10KB and less than 100KB.

7.4 Sensitivity Analysis of TCP Reno:

In this section, we evaluate the performance of TCP Reno with regard to a range of congestion window reduction factors. Congestion window reduction factor ranged from 0.5 to 0.9, where 0.5 is the default reduction factor.

Although, for different congestion window reduction factors, modified variant performed similar at a lower loss percentage, as the loss percentage increased, larger reduction factor consumed longer duration for complete page load. Surprisingly, smaller reduction factor performed better on high loss. This can be inferred from Figure 9.

This observation stems from the fact that under high loss and larger reduction factor, congestion control algorithm reduces the congestion window aggressively and successively. However, in similar condition, smaller reduction factor reduces the congestion window less aggressively and hence performs better.

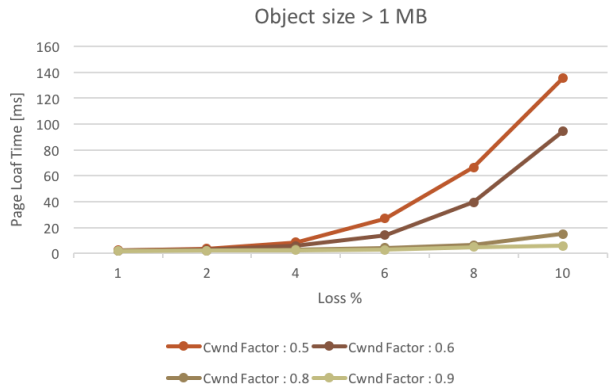

Figure 12: Page Load Time vs Packet loss % for objects greater than 1MB.

VIII. CONCLUSION

HTTP/2 has multiple features enhanced over HTTP/1.1. Prior studies show the performance of HTTP/2 deteriorates under high packet loss due to aggressive congestion window back-off technique. In our paper, we modify the congestion window back off behavior of two TCP variants, CUBIC and Reno and evaluate their performance under various network conditions. Our modified algorithm, CUBIC+ was found to perform well under varying packet loss for all object sizes. Also to improve RRT fairness in multiple networks with varying delay times, we incorporated changes as described in the study [3] and experiments were conducted. It was also found that varying initial congestion window size has less effect on the performance of three variants. Further, sensitivity analysis with regard to congestion window was conducted for TCP RENO, over a range of congestion reduction factor. Although, different reduction factor performed similar under low loss, under high packet loss, conservative reduction factor performed better that aggressive reduction.

REFERENCES

[1] How speedy is SPDY? https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-wang_xiao_sophia.pdf

[2] CUBIC: A New TCP-Friendly High-Speed TCP Variant. https://pdfs.semanticscholar.org/2d49/06884bc5309f1539195ff5b181d41a15ff60.pdf

[3] Improving RTT fairness on CUBIC TCP. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6726892&tag=1

[4] CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6530831