Contents lists available at www.innovativejournal.in

Asian Journal of Computer Science and Information Technology

Journal homepage: http://www.innovativejournal.in/index.php/ajcsit

ISSN 2249 - 5126

INCREASING TCP'S INITIAL CONGESTION WINDOW AS A MANIFEST CONSTANT TO SCALE

Oludele Awodele*, Enem Theophilus Aniemeka

Computer Science Department, Babcock University, Ilisan-Remo, Ogun state, Nigeria.

This is a plagiarism that combines parts ripped out from (at least) the following three works.

John Kristoff (?)
TCP Congestion Control
March 2000 (?)
https://condor.depaul.edu/jkristof/technotes/congestion.pdf
This note, which is similar to Ref-[6], does not seem to have been published formally.

Van Jacobson
Congestion Avoidance and Control
ACM SIGCOMM '88 Communication Architecture and Protocols, (1988)
https://doi.org/10.1145/52324.52356
This is cited as Ref-[7].

Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, Natalia Sutin
An argument for increasing TCP's initial congestion window
ACM SIGCOMM Computer Communication Review 40(3), July 2010
https://doi.org/10.1145/1823844.1823848

Contents lists available at www.innovativejournal.in

Asian Journal of Computer Science and Information Technology

Journal homepage: http://www.innovativejournal.in/index.php/ajcsit

ISSN 2249 - 5126

INCREASING TCP'S INITIAL CONGESTION WINDOW AS A MANIFEST CONSTANT TO SCALE

Oludele Awodele*, Enem Theophilus Aniemeka

Computer Science Department, Babcock University, Ilisan-Remo, Ogun state, Nigeria.

ARTICLE INFO

Corresponding Author:
Oludele Awodele,
Computer Science Department,
Babcock University, Ilisan-Remo,
Ogun state, Nigeria

ABSTRACT

This paper discusses some Transmission Control Protocol (TCP) congestion control algorithm, and proposes increasing TCP's initial congestion window to at least fifteen segments (about 25 KB). Transmission Control Protocol (TCP) flow start with an initial congestion window at most three (3) segments or about 4 KB of data. Most Web transactions are short-lived and TCP's initial congestion window is a critical parameter determining how quickly flows can finish. The rapid growth of the internet in terms of the volume of activity and traffic it carries over the past decades represents a remarkable example of the scalability of the internet architecture which in spite of the growth, the standard TCP's initial congestion value has not changed.

INTRODUCTION

Internet network have experienced an explosive growth over the past few years and with that growth have come severe congestion problems. For Internet to continue to thrive, its congestion control algorithm must remain effective [1]. TCP is the most widely used protocol for the transport layer on the Internet. Frankly speaking, TCP has changed very little since its design in the early 1980's and a few "tweaks" and "knobs" have been added but the standard value of TCP's initial congestion window has remained unchanged since 2002 [2].

This paper proposes to increase TCP's initial congestion window to reduce Web latency during the slow start phase of a connection. TCP uses the slow start algorithm early in the connection lifetime to grow the amount of data that may be outstanding at a given time. Slow start increases the congestion window by the number of data segments acknowledged for each received acknowledgment. Thus the congestion window grows exponentially and increases in size until packet loss occurs, typically because of router buffer overflow, at which point the maximum capacity of the connection has been probed and the connection exits slow start to enter the congestion avoidance phase. The initial congestion window is at most four segments, but more typically is three segments for standard Ethernet (approximately 4KB) [3]. The majority of connections on the Web are short-lived and finish before exiting the slow start phase, making TCP's initial congestion window (init_cwnd) a crucial parameter in determining flow completion time. The premise is that the initial congestion window should be increased to speed up short Web transactions while maintaining robustness. A 2009 study [4], reveals that the average connection bandwidth globally is 1.7Mbps with more than 50% of clients having bandwidth above 2Mbps, while the usage of narrowband (<256Kbps) has shrunk to about 5% of clients. At the same time, applications devised their own mechanisms for faster download of Web pages. Popular Web browsers, including Internet Explorer 8 (IE8) [5], Firefox 3 and Google's Chrome, open up to six TCP connections per domain, partly to increase parallelism and avoid head-offline blocking of independent HTTP requests/responses, but mostly to boost start-up performance when downloading a Web page.

This paper does not cover the basics of the TCP protocol itself, but rather the underlying designs and algorithm as they apply to problem of network overload and congestion.[6]

2.0 STANDARD TCP CONGESTION CONTROL ALGORITHMS

The standard fare in TCP implementations today can be found in RFC 2581. This reference document specifies five standard congestion control algorithms that are now in common use. Each of the algorithms noted within that document was actually designed long before the standard was published [7].

The five algorithms, Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery and Selective Acknowledgement are described below.

2.1 Slow Start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. In other words, the rate of acknowledgements returned by the receiver determine the rate at which the sender can transmit data. When a TCP connection first begins, the Slow Start algorithm initializes a congestion window to one

segment, which is the maximum segment size (MSS) initialized by the receiver during the connection establishment phase. When acknowledgements are returned by the receiver, the congestion window increases by one segment for each acknowledgement returned. Thus, the sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the transmission window. Slow Start is actually not very slow when the network is not congested and network response time is good. It takes time $R\log_2 W$ where R is the round-trip-time and W is the window size in packets (fig. 1). For example, the first successful transmission and acknowledgement of a TCP segment increases the window to two segments. After successful transmission of these two segments and acknowledgements completes, the window is increased to four segments. Then eight segments, then sixteen segments and so on, doubling from there on out up to the maximum window size advertised by the receiver or until congestion finally does occur. At some point the congestion window may become too large for the network or network conditions may change such that packets may be dropped. Packets lost will trigger a timeout at the sender. When this happens, the sender goes into congestion avoidance mode as described in the next section.

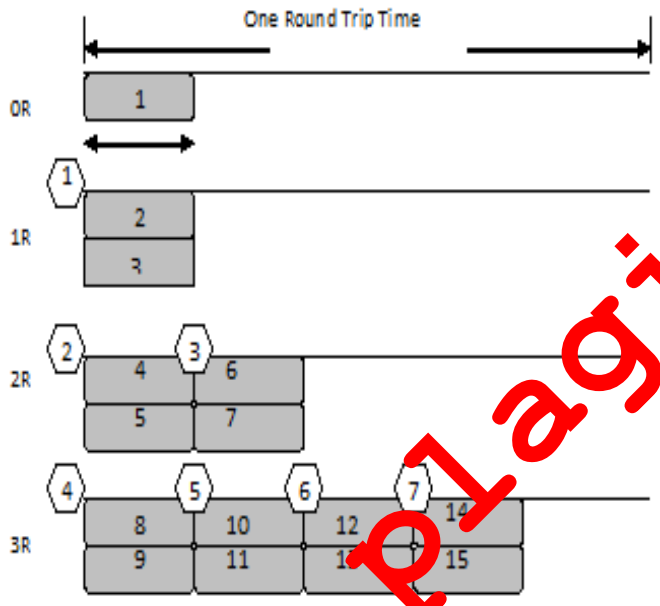

Figure 1: The Chronology of a Slow-start

2.2 Congestion Avoidance

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion. If this happens, Congestion Avoidance is used to slow the transmission rate [9]. However, Slow Start is used in conjunction with Congestion Avoidance as the means to get the data transfer going again so it doesn't slow down and stay slow. In the Congestion Avoidance algorithm a retransmission timer expiring or the reception of duplicate ACKs can implicitly signal the sender that a network congestion situation is occurring. The sender immediately sets its transmission window to one half of the current window size (the minimum of the congestion window and the receiver's advertised window size), but to at least two segments. If congestion was indicated by a timeout, the congestion window is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was indicated by duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are invoked.

As data is received during Congestion Avoidance, the congestion window is increased.

However, Slow Start is only used up to the halfway point where congestion originally occurred. This halfway point was recorded earlier as the new transmission window. After this halfway point, the congestion window is increased by one segment for all segments in the transmission window that are acknowledged. This mechanism will force the sender to more slowly grow its transmission rate, as it will approach the point where congestion had previously been detected.

2.3 Fast Retransmit

When a duplicate ACK is received, the sender does not know if it is because a TCP segment was lost or simply that a segment was delayed and received out of order at the receiver. If the receiver can re-order segments, it should not be long before the receiver sends the latest expected acknowledgement. Typically no more than one or two duplicate ACKs should be received when simple out of order conditions exist. If however more than two duplicate ACKs are received by the sender, it is a strong indication that at least one segment has been lost. The TCP sender will assume enough time has lapsed for all segments to be properly re-ordered by the fact that the receiver had enough time to send three duplicate ACKs. When three or more duplicate ACKs are received, the sender does not even wait for a retransmission timer to expire before retransmitting the segment (as indicated by the position of the duplicate ACK in the byte stream). This process is called the Fast Retransmit algorithm and was first defined in [8]. Immediately following Fast Retransmit is the Fast Recovery algorithm.

2.4 Fast Recovery

Fast Retransmit algorithm is used when duplicate ACKs are being received, the TCP sender has implicit knowledge that there is data still flowing to the receiver. Why? The reason is because duplicate ACKs can only be generated when a segment is received. This is a strong indication that serious network congestion may not exist and that the lost segment was a rare event. So instead of reducing the flow of data abruptly by going all the way into Slow Start, the sender only enters Congestion Avoidance mode. Rather than start at a window of one segment as in Slow Start mode, the sender resumes transmission with a larger window, incrementing as if in Congestion Avoidance mode. This allows for higher throughput under the condition of only moderate congestion [10].

2.5 Selective Acknowledgements

Whenever a TCP segment has been sent and the sender's retransmission timer expires, the sender is forced to retransmit the segment, which the sender assumes has been lost. However, it is possible that between the time when the segment was initially sent and the time when the retransmission window expired, other segments in the window may have been sent after the lost segment. It is also possible that these later segments arrived at the receiver and are simply queued awaiting the missing segment so they can be properly reordered. The receiver has no way of informing the sender that it has received other segments because of the requirement to acknowledgement only the contiguous bytes it has received. This case demonstrates a potential inefficiency in the way TCP handles the occasional loss of segments.

Ideally, the sender should only retransmit the lost segment(s) while the receiver continues to queue the later segments. This behaviour was identified as a potential improvement in TCP's congestion control algorithms as early as 1988 [11]. It was only until recently that a mechanism to retransmit just the lost segments in these situations was put into standard TCP implementations [12], [13].

Selective Acknowledgement (or SACK) is this technique implemented as a TCP option that can help reduce unnecessary retransmissions on the part of the sender. If the TCP connection has negotiated the use of SACK (through the use of the TCP header option fields), the receiver can offer feedback to the sender in the form of the selective acknowledgement option. The receiver reports to the sender, which blocks of data have arrived using the format show in figure 2 below.

bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Kind = 5	length
Left Edge of 1st Block	
Right Edge of 1st Block	
Left Edge of nth Block	
Right Edge of nth Block	

Figure 2 SACK Option

This list of blocks in the SACK option tells the sender which contiguous byte stream blocks it has received. At maximum, four SACK blocks can be sent in one TCP segment because of the maximum size of the options field in a TCP head is 40 bytes and each block report consists of 8 bytes plus the option header field of 4 bytes (for a total of 36 bytes). Note that the SACK information is advisory information only. The sender cannot rely upon the receiver to maintain the out-of-order data. Obviously the performance gain is to be had when the receiver does queue and re-order data that has been reported with the SACK option so that the sender limits its retransmissions.

3.0 ANALYTICAL EVALUATION ON THE TCP SLOW-START ALGORITHM

TCP uses two main window-based mechanism, the receive window and the congestion window. The former is the receive-side limit. The later is the number of segments that will be sent before waiting for an acknowledgement. In a perfect world (no packet loss, orderly arrival of segments, compliant devices, etc) increasing the initial congestion window lowers the initial latency as less round trips are required to transfer the same amount of data. The inefficiency in the way TCP handles the occasional loss of segments. Ideally, the sender should only transmit the loss segment(s) while the receiver continues to queue the later segments. This was not the case, which gave rise the SACK option.

The congestion window is best expressed in multiples of the MSS (Maximum Segment Size). RFC 3390 defines the allowed initial_cwnd as:

*min (4 * MSS, max(2 * MSS, 4380 bytes))*

A related parameter is the *ssthresh* (slow-start threshold). RFC 5681 states that congestion avoidance is used if *cwnd > ssthresh* or *cwnd >= ssthresh*, otherwise slow-start is in effect. The default congestion avoidance algorithm in Linus 2.6.19+, CUBIC, sets the initial *ssthresh* to 0, so initial congestion avoidance is used, unless an *ssthresh* metric higher than congestion window is cached from the previous connection. During congestion avoidance, congestion window is incremented, but CUBIC does not follow the recommended formula *cwnd += min (N, SMSS)* or *cwnd += SMSS * SMSS/cwnd*, where N is the number of ACKed bytes and SMSS is the Sender Side MSS. Instead, the window is set according the cubic function of time since the last congestion. It does not rely on the ACKed byte count, allowing the window to grow at the same rate for low and high-latency flows [21].

TCP is a complex protocol, its specifications spread over tens of RFCs. Every modification affects the behaviour of several other mechanisms. Increasing the congestion window increases the burstiness of traffic.

4.0 BENEFITS OF ALLOWING TCP TO START WITH HIGHER INIT_CWND

In light of these trends, allowing TCP to start with a higher init_cwnd offers the following Benefits:

4.1 ***Reduce latency.*** Latency of a transfer completing in slow start without losses [14], As link speeds scale up, TCP's latency is dominated by the number of round-trip-times (RTT) in the slow-start phase. Increasing init_cwnd enables transfer to finish in fewer RTTs.

4.2 ***Keep up with growth in Web page sizes.*** The Internet average Web page size is 384KB [15] including HTTP headers and compressed resources. An average sized page requires multiple RTTs to download when using a single TCP connection with a small init_cwnd. To improve page load times, Web browsers routinely open multiple concurrent TCP connections to the same server. Web sites also spread content over multiple domains so browsers can open even more connections [16]. A study on the maximum number of parallel connections that browsers open to load a page [17] showed Firefox 2.0 opened 24 connections and Internet Explorer 8 (IE8) opened 180 connections while did not reaching its limit. These techniques not only circumvent TCP's congestion control mechanisms [18], but are also inefficient as each new flow independently probes for end-to-end bandwidth and incurs the slow start overhead. Increasing init_cwnd will not only mitigate the need for multiple connections, but also allow newer protocols such as SPDY [19] to operate efficiently when downloading multiple Web objects over a single TCP connection.

4.3 ***Allow short transfers to compete fairly with bulk data traffic.*** Internet traffic measurements indicate that most bytes in the network are in bulk data transfers (such as video), while the majority of connections are short-lived and transfer small amounts of data. Statistically, on start-up, a short-lived connection is already competing with connections that have a congestion window greater than three segments. Because short-lived connections, such as Web transfers, don't last long enough to achieve their fair-share rate, a higher init_cwnd gives them a better chance to compete with bulk data traffic. [20]

4.4 ***Allow faster recovery from losses.*** An initial window larger than three segments increases the likelihood that losses can be recovered through Fast Retransmit rather than the longer initial retransmission timeout. Furthermore, in the presence of congestion, the widespread deployment of Selective Acknowledgments (SACK) enables a TCP sender to recover multiple packet losses within a round-trip time.

The proposal to increase TCP's init_cwnd to at least fifteen segments (approximately 25KB) was born out of the need to satisfy some properties, which includes:

(i) Minimize average Web page download.

(ii) Minimize impact on tail latency due to increased packet loss.

(iii) Maintain fairness with competing flows.

The increase in TCP's initial congestion window to fifteen segments improves the average TCP latency compared to using three segments and yet it is sufficiently robust for use in the internet.

There are numerous studies in literature on speeding up short transfers over new TCP connections. These techniques range from faster start-up mechanisms using cached congestion windows such as TCP Fast Start and Congestion Manager to more complex schemes requiring router support such as Quick Start. These solutions are neither widely deployed, nor standardized, and do not have practical reference implementations.

5.0. CLIENT RECEIVE WINDOWS

Since TCP can only send the minimum of the congestion window and the client's advertised receive window, the receive window (rwnd) may limit the potential performance improvement of increasing init cwnd. To this end, clients need to advertise at least a 25KB receive window on a connection to fully benefit. Overall, many client connections have a large enough receive window to fully benefit from using init cwnd=15 segments.

5.1 Negative impact

Having enumerated the overall benefits of using a higher initial congestion window; I want to mention the costs, specifically cases where latency increases. Increase in latency primarily arises from packet losses caused by overflowing bottleneck buffers, either at end-systems or at intermediate routers and switches. Losses prolong TCP flows by adding extra RTTs required to recover lost packets, and occasionally even resulting in retransmission timeouts. Internet measurements and studies show that a critical bottleneck in the Internet lies in the last mile at the user's access link. Thus, if there is a cost associated with init cwnd=15, it is likely that we will observe increased congestion and packet losses.

CONCLUSIONS

Increasing TCP's initial congestion window is a small change with a significant positive impact on Web transfer latency. While the numerous studies in literature to speed up short transfers may be viable solutions in the future, none are deployed or standardized today. In contrast, a far simpler solution of increasing TCP's initial congestion window to a value commensurate with current network speeds and Web page sizes is practical, easily deployable, and immediately useful in improving Web transfer latency. In the longer term, a larger initial congestion window will also mitigate the need for applications to use multiple concurrent connections to increase download speed. The paper recommends that the IETF to standardize TCP's initial congestion window to at least fifteen segments. Interested reserachers should focus on eliminating the initial congestion window as a manifest constant to scale to even large network speeds and Web page sizes.

REFERENCES

[1] Allman, M., Paxson, V., & Stevens, W. (1999). TCP congestion control. RFC 2581.

[2] Semke, J., Mahdavi, J., & Mathis, M. (1998). Automatic tcp buffer tuning. Computer Communications Review, ACM SIGCOMM, volume 28, Number 4.

[3] Allman, M., Floyd, S,. & Partridge, C. (2002). Increasing tcp's initial window. RFC 3390.

[4] Akamai (2009). The State of the Internet. http://www.akamai.com/stateoftheinternet.

[5] AJAX (2010). Connectivity enhancements in internet explorer 8. http://msdn.microsoft.com/enus/library/cc304129(VS.85).

[6] Kristoff, J. (2000). The Transmission control protocol.

[7] Jacobson, V. (1998). Congestion avoidance and control. Computer Communications Review, volume 18 number 4, pp. 314-329.

[8] Jacobson, V. (1990). Modified tcp congestion control avoidance algorithm. End-2-end-interest mailing list.

[9] Michael J. Karels,. & Van Jacobson Van. (1998). Congestion avoidance and control. University of California at Berkeley.

[10] Stevens, W. (1997). TCP Slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001.

[11] Jacobson, V., & Braden, R. (1998). TCP Extensions for long-delay paths. RFC 1072.

[12] Belshe, M. (2010). A client-side argument for changing tcp slow start. http://sites.google.com/a/chromium.org/dev/spdy/.

[13] Mathis, M., Mahdavi, J., Floyd, S., & Romanow, A. (1996). TCP selective acknowledgement options. RFC 2018.

[14] Cardwell, N., Savage, S., & Anderson, T. (2000). Modeling tcp latency: In proceedings of IEEE infocom.

[15] Ramachandran, S., & Jain, A. (2010). Web page stats, size and number of resources. http://code.google.com/speed/articles/ web-metrics.html.

[16] Nandita, D., Tiziana, R., Jerry, C., Natalia, S. (2010). An argument for increasing TCP's initial congestion window.

[17] Souders, S. (2008). Roundup on parallel connections. http://www.stevesouders.com/blog/2008/03/20/roundupon- parallel-connections/.

[18] Floyd, S., & Fall, K. (1999). Promoting the use of end-to-end congestion control in the internet. In IEEE/acm, transactions on networking.

[19] SPDY (2009): An experimental protocol for a faster web. http://dev.chromium.org/spdy.

[20] Iyengar, J., Caro, A., & Amer, P. (2003). Dealing with short tcp flows: A survey of mice in elephant shoes. In Tech Report, CIS Dept, University of Delaware.

[21] http://netsrv.csc.ncsu.edu/export/cubic_a_ nw_tcp_2008.

Enem Theophilus Aniemeka
I was born on 23 Sep 1964 in Eke, Udi Local Government Area of Enugu State. I had my primary education at Colliery Primary School Iva-Valley Enugu, Secondary education at College of Immaculate Conception (C.I.C) Enugu, I joined the Nigerian Air Force in 1985 and was posted to Benin, However, from 1989 to 1995, I studied Diploma in Data Processing and BS.c(HONS) Computer Science from University of Benin. I bagged my Master degree in Information Technology from National Open University of Nigeria in 2010. I play squash game. I am happily married to Chinyere Enem, and we are blessed with two kids (Onyedikachukwu and Chinekwu).

Awodele, Oludele holds a Ph.D. in Computer Science from the University of Agriculture, Abeokuta, Nigeria. He has several years experience of teaching computer science courses at the university level. He is currently a lecturer in the department of Computer Science, Babcock University, Nigeria. He is a full member of the Nigeria Computer Society (NCS) and the Computer Professional Registration Council of Nigeria. His areas of interest are Artificial Intelligence, Cloud Computing and Computer Architecture. He has published works in several journals of international repute.