A Modified Congestion Control Algorithm for Evaluating High BDP Networks

Ehab Aziz Khalil

Dept. of Computer Science & Engineering Faculty of Electronic Engineering (Minufiya University)

Summary
It is well known that the TCP congestion control algorithm has been remarkably successful in improving the current TCP/IP function better and efficiently. However, it can perform poorly in networks with high Bandwidth Delay Product (BDP) paths. This paper presents modification to a congestion control algorithm that may help the TCP better utilize the bandwidth provided by huge bandwidth long delay links. It also presents results to show a comparison of the original algorithm.

Key words:
TCP congestion control, Swift Start algorithm, round trip time, high BDP, performance evaluation.

1. Introduction

Today as well as tomorrow the main problem in the design of networks is the development of congestion control algorithms. Conventional congestion control algorithms were deployed for two principle reasons: the first one is to ensure avoidance of network congestion collapse [1], [2]; and second one is to ensure a degree of network fairness. Roughly speaking, network fairness refers to the situation whereby a data source receives a fair share of available bandwidth, whereas congestion collapse refers to the situation whereas an increase in network load results in a decrease of useful work done by the network (usually due to retransmission of data). Attempts to deal with network congestion have resulted in the widely applied transmission control protocol [3].
While the current TCP congestion control algorithm has proved remarkably durable, it is likely to be less effective on next generation networks featuring gigabit speed connectivity and heterogeneous traffic and sources. These considerations have led to widespread acceptance that new congestion control algorithms must be developed to accompany the realization of next generation systems and perhaps also to better exploit the resources of existing networks [4].
In recent years, several more aggressive versions of TCP have been proposed [5-20], and there are some researches investigate the congestion control and long delay bandwidth product such as in [21-30], have been published and many are still in progress, all of them investigate and discuss the congestion control mechanisms in the Internet which consists of the congestion window algorithms of TCP, running at the end-systems, and Active Queue Management (AQM) algorithm at routers, seeking to obtain high network utilization, small amounts of queuing delay, and some degree of fairness among users. This paper presents a comparison performance evaluation of a modified TCP congestion algorithm [19],[20].

2. Background and Motivation

It is well known that network congestion control occurs when too many sources attempt to send data at too high rates. At the sender end, this is detected by packet loss. Due to congestion, the network experiences large queue delays and consequently the sender must retransmissions in order to compensate for the lost packets. Hence the average transmission capacity of the upstream routers is reduced. A TCP connection control its transmission rate by limiting its number of unacknowledged segments; the TCP window size W. TCP congestion control is based on dynamic window adjustment. The TCP connection is begins in slow start phase the congestion window is doubled every RTT until the window size reaches slow-start threshold. After this threshold, the window is increased at a much slower rate of about one packet each RTT. The window cannot exceed a maximum threshold size that is advertised by the receiver. If there is a packet loss then the threshold drops to half of the present value and the window size drops to the one Maximum Segment Size (MSS). A congestion avoidance mechanism maintains the network at an operating point of low delay and high throughput.
However, there are four basic congestion algorithms that should be included in any modern implementation of TCP, these algorithms are: Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery [15]. The last two algorithms were developed to overcome the short comings of earlier implementations, like TCP Tahoe [31], where TCP Tahoe was getting into the slow start phase every time a packet was lost and thus valuable bandwidth was wasted and the slow start algorithm is used to gradually increase the size of the TCP congestion window. It operates by observing that the rate at which new packets

Manuscript received November 5, 2012
Manuscript revised November 20, 2012

should be injected into the network is the rate at which the acknowledgments are returned by the other end.

Indeed, a modern TCP implementation that includes the above four algorithms is known as TCP Reno which is the dominant TCP version. J.C. Hoe [32] modified the Reno version of TCP to improve the start up behavior of TCP congestion control scheme. These improvements include finding the appropriate initial threshold window (ssthresh) value to minimize the number of packets lost during the start up period and creating a more aggressive fast retransmit algorithm to recover from multiple packet losses without waiting unnecessarily for the retransmission timer to expire. We have to mention here that the TCP Reno is like TCP Tahoe except that if the source receives three "duplicate" Acks, it consider this indicative of transient buffer overflow rather than congestion, and does the following: (1) it immediately (i.e., without waiting for timeout) resends the data requested in the Ack; this is called fast retransmit. (2) it sets the congestion windows and slow start threshold to half the previous congestion window (i.e., avoids the slow start phase) this is called fast recovery. The two variable congestion window (CWND) and slow start threshold (ssthresh), are used to throttle the TCP input rates in order to much the network available bandwidth. All these congestion control algorithms exploit the Additive Increase Multiplication Decrease (AIMD) paradigm, which additively increases the CWND to grab the available bandwidth and suddenly decreases the CWND when the network capacity is hit and congestion is experienced via segment losses, i.e., timeout or duplicate acknowledgments. AIMD algorithms ensure network stability but they don't guarantee fair sheering of network resources [33],[34],[35]. The next section will highlight the comparison of different versions of TCP.

2.1- Swift Start Algorithm

Swift Start is a new congestion control algorithm was proposed and designed by BBN Technologies [36] to increase the performance of TCP over high delay-bandwidth product networks by improving its start up. Swift Start tries to solve the congestion control problems by using packet pair and pacing algorithms together.

As known that the traditional packet pair algorithm has a problem in which it assumes that the ACK path does not affect the delay between ACKs. But both ACKs may be subjected to different queuing delays in their path, which may causes an over or under estimate of the bottleneck capacity. However, to avoid congestion due to over estimation the Swift Start uses only a fraction of the calculated bandwidth; this fraction is determined by a variable α which indicates the rates between 1 and 8.

The following sub-section presents the propose modification in packet pair algorithm to avoid the mentioned defects in traditional packet pair, and use the modification packet pair algorithm in the Swift Start.

2.2- Modification Swift Start Algorithm [20]

The objective of this modification is to avoid error sources in the traditional packet pair algorithm. However the idea behind that is instead of time depending on the interval between the acknowledgments that may cause errors, the time between the original messages will be calculated by the receiver when they arrive it, and then the receiver sends this information to the source when acknowledging it. The sender sends its data in form of packet pairs, and identifies them by First/Second (F/S) flag. When the receiver receives the first message, it will record its sequence number and its arrival time, and it will send the acknowledgment on this message normally according its setting. When it receives the second one, it will check whether it is the second for the recorded one or not, if it is the second for the recorded one, the receiver will calculate the interval Δt between the arrival time of the second one and that of the first one:-

$$\Delta t = t\_seg1 - t\_seg2 \qquad \mu\ sec \ldots\ldots (1)$$

Where: t_seg1 and t_seg2 are the arrival time of the first and second segments respectively. However, when the receiver sends the acknowledgment for the second segment, it will insert the value of Δt into the transport header option field. The sender's TCP will extract Δt from the header and calculate the available bit rate BW :

$$BW = \Delta t \ x\ SegSize \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Where: SegSize is the length of the second segment.

If the receiver uses the *DACK* technique, it will record the first segment arrival time and wait for 200 ms, when it receives the second one it will calculate Δt and wait for new 200 ms, if it receives another packet pair it will calculate another Δt, whenever it sends an acknowledgment, it will send Δt in it.

By this way the error sources are avoided and the estimated capacity is the actual capacity without neither over estimation nor under estimation. So it is not needed to use only a fraction of the capacity like the traditional Swift Start.

2.3 –Motivation

Swift Start faces many problems when combining with other techniques such Delayed Acknowledgment and acknowledgment compression.

I- Effect of Delayed Acknowledgment (DACK)

The majority of TCP receivers implement the delayed acknowledgment algorithm [37], [38] for reducing the number of pure data less acknowledgment packets sent. A TCP receiver, which is using that algorithm, will only send acknowledgments for every other received segment. If no segment is received within a specific time, an *ACK* will send, (this time typically is 200 ms.). The algorithm will directly influence packet pair estimation, because the *ACK* is not sent promptly, but it may be delayed some time (200 ms). If the second segment of the packet pair arrives within 200 ms the receiver will send single *ACK* for the two segments instead of sending an *ACK* for each segment. Hence the sender can not make that estimation.

II- Effect of Acknowledgment Compression [39], [40]

Router that supports acknowledgment compression will send only one *ACK* if it receives two consecutive *ACK* messages of a connection within small interval. This will also affect the Swift Start and cause it to falsely estimate the path capacity.

III-Effect of ACK Path

It is well known that the traditional packet pair algorithm has a problem in which it assumes that the *ACK* path does not affect the delay between *ACKs*. But both *ACKs* may be subjected to different queuing delays in their path, which may causes an over or under estimate of the bottleneck capacity. However, to avoid congestion due to over estimation the Swift Start uses only a fraction of the calculated bandwidth; this fraction is determined by a variable α which indicates the rates between 1 and 8.

3- Slow Start over High BDP Networks

As stated in TCP congestion control [13], the slow start and congestion avoidance algorithms must be used by a TCP sender to control the amount of outstanding data being injected into the network.

To implement these algorithms, two variables are added to the TCP per-connection state. The congestion window (*CWND*) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (*ACK*), while the receiver's advertised window (*RWND*) is a receiver-side limit on the amount of outstanding data. The minimum of *CWND* and *RWND* governs data transmission. Another state variable, the slow start threshold (*ssthresh*), is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission. When a new connection is established with a host, the congestion window is initialized to a value that is called Initial window (*IW*) which typically equals to one segment. Each time an acknowledgement (*ACK*) is received; the *CWND* is increased by one segment. So TCP increases the *CWND* by percentage of 1.5 to 2 each round trip time (*RTT*), The sender can transmit up to the minimum of the *CWND* and the advertised window "*RWND*". When the congestion window reaches the *ssthresh* the congestion avoidance should starts to avoid occurrence of congestion. The congestion avoidance increases the *CWND* when receiving an acknowledge according to equation 3.

$$CWND \mathrel{+}= SMSS*SMSS/CWND \quad --- \quad (3)$$

Where *SMSS* is the sender maximum segment size

TCP uses slow start and congestion avoidance until the *CWND* reaches the capacity of the connection path, and an intermediate router will start discarding packets. Timeouts of these discarded packets informs the sender that its congestion window has gotten too large and congestion has been occurred. At this point TCP reset *CWND* to the *IW*, and the *ssthresh* is divided by two and the slow start algorithm starts again.

Many other additions such as fast retransmit, fast recovery], the new Reno Modification to TCP fast recovery algorithm, and increasing TCP's initial window were added to TCP congestion control.

The current implementations of Slow Start algorithm are suitable for common link which has low-delay and modest-bandwidth. Because it takes a small time to correctly estimate and begin transmitting data at the available capacity. While, over high delay-bandwidth product networks, it may take several seconds to complete the first slow start and estimate available path capacity.

Fig. 1 shows a network model used to illustrate the effect of Round Trip Time (*RTT*) on the connection time, using different *RTTs* and bottleneck bandwidth of 49.5 Mbps. Fig 2 shows the effect of *RTT* on the connection time, for a connection that transmit a 10 Mbytes file. Fig. 3 shows the bandwidth utilization for the same *RTTs*.

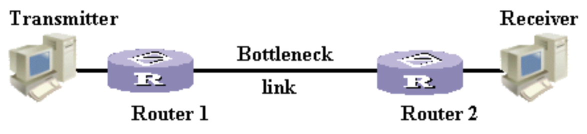


Fig. 1 Network Model

From Fig 2, 3, it is clear that as *RTT* increases the bandwidth utilization decreases and the connection time increases although there is a large amount of bandwidth is available. The second note shown in Fig 3 is that as the *RTT* increases the time to reach the maximum transfer rate increases. These are two problems in slow start over high delay bandwidth product connections.

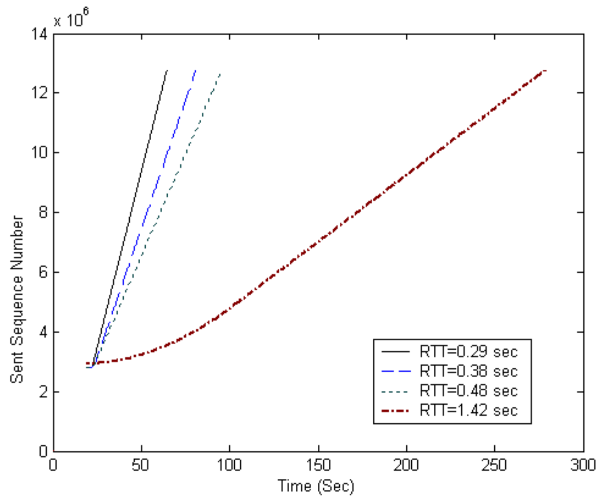


Fig. 2 Effect of RTT on the TCP Connection Time

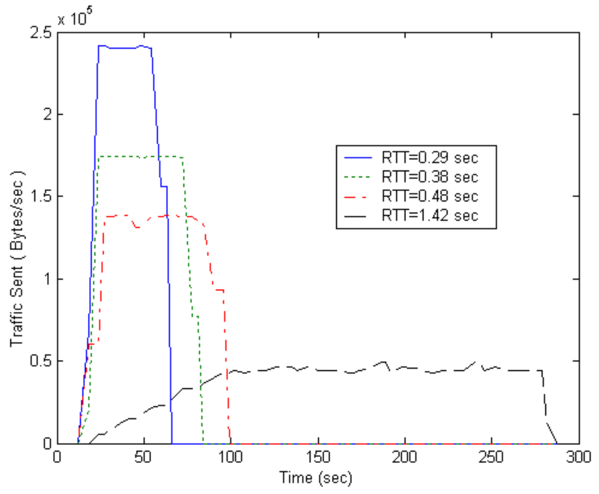


Fig. 3 Effect of RTT on the TCP Transmission Rate

The first problem is due to the fact that each *RTT* the slow start can not send greater than the minimum of *CWND* and *RWND*, the later one is limited to 65535 bytes because its advertised by the receiver in a 16 bit field. It can be overcome by using Window Scaling option in TCP.
The second problem is the longer time to reach the maximum transfer rate, many algorithms was proposed to overcome this problem such Increasing TCP's Initial Window, Fast TCP, TCP Fast Start, Explicit Control Protocol (XCP), High speed TCP, Quick-Start for TCP and IP [32] and Swift start for TCP**.**

4- Simulation and Results

We implement the Modified swift start model using Opnet modeler to compare its performance with that of the slow start in deferent network conditions of bandwidth and path delay, then we compare between them using single flow and multiple flow to show the effect of these flows on each other.

4.1- Single Flow Low BDP Networks

The network shown in Fig 1 was used to show the performance of Swift start TCP and compare it with Slow start using single flow the sender and the receiver. The sender uses FTP to send a 10 MB file to the receiver. The TCP parameters of both the sender and the receiver is shown in Table-1.
The sender and the receiver are connected to the routers with 100 Mbps Ethernet connections. both of the routers are CISCO 3640 with forwarding rate 5000

Table-1 TCP Parameters of the Sender and Receiver

Maximum Segment Size	1460 Bytes
Receive Buffer	100000 Bytes
Receive Buffer Usage Threshold	0
Delayed *ACK* Mechanism	Segment/Clock Based
Maximum *ACK* Delay	0.200 Sec
Slow-Start Initial Count	4
Fast Retransmit	Disabled
Fast Recovery	Disabled
Window Scaling	Disabled
Selective *ACK* (*SACK*)	Disabled
Nagle's SWS Avoidance	Disabled
Karn's Algorithm	Enabled
Initial RTO	1.0 Sec
Minimum RTO	0.5 Sec
Maximum RTO	64 Sec
RTT Gain	0.125
Deviation Gain	0.25
RTT Deviation Coefficient	4.0
Persistence Timeout	1.0 Sec

packets/second and memory size 265 MB. The two router are interconnected with point to point link that link is used as a bottleneck by changing its data rate , also the path delay is controlled using this link.
Fig. 4 shows the congestion window for both slow start TCP and Modified swift start TCP when the bottleneck data rate is 1.5 Mbps (T1) and the path *RTT* is 0.11674 second, which is low rate, low delay network. its clear that the modified swift start is faster and better than slow start TCP in estimating the path congestion window which is = 22120 bytes after only one *RTT* , then the packet pair is disabled and the slow start runs normally. The estimated congestion window is proportional to the link bandwidth and round trip time it can be calculated as follow:
Assuming that packet pair delay deference is *D*

CWND = the amount of data that can be sent in *RTT*
= *RTT* * *MSS* / *D*

Theoretically the packet pair delay deference is the frame length on the bottleneck link, so

D = frame length /link rate
= (1460+20+7) * 8 / 1544000 = 0.007705 sec

and *RTT* is measured for the first pair (RTT = 0.11674 sec) **so** *CWND* = 0.11674 * 1460 / 0.007705 = 22120.75 bytes. We neglect the processing delay which may affect the value of *D* and so decrease *CWND*. The result in the simulation shows that the delay difference is 0.007772 sec and the *CWND* is 21929 bytes, these results very close to the mathematical results.

In Fig. 4 also we note that the next value for the congestion window is 24849 bytes, because this window was calculated when receiving the *ACK* for the second pair, this pair was buffered in the network, so the round trip time for it was increased, the simulation results shows that RTT for the second pair is 0.13228 sec, and the delay between the first and second packet is the same 0.007772 sec.

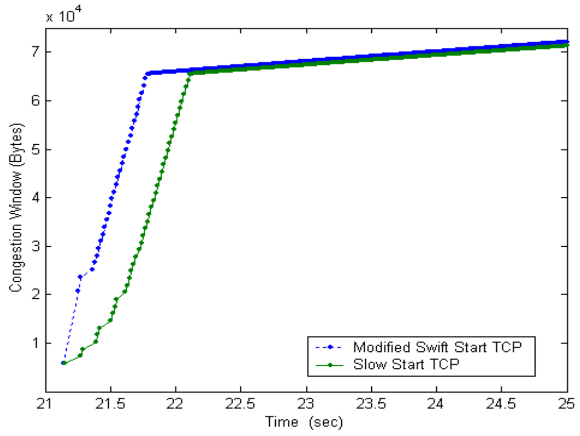


Fig-4. Congestion Window for Slow Start TCP and Modified Swift Start TCP for BW = 1.5 Mbps and Path RTT= 0.11674 Sec

Fig. 5-a and 5-b show the sent segment sequence number for this connection. it is shown that both algorithms start the connection by sending 4 segments, after 1 *RTT* (0. 11674 sec) slow start send 6 segments with in the second *RTT*, while modified swift start send a large number of segments because of its large congestion window which is 20722 bytes which is about 14 segments, these segments were paced along the second *RTT*, until the sender receives an other *ACK* that indicates the end of the second *RTT* and the beginning of the third *RTT*, at this time the pacing was stopped and the slow start was used to complete the connection.

In Fig 5-a we note that after a certain time both algorithms reaches a constant transmission rate , we roughly calculate this rate as :

Transmission rate = 187848 bytes / sec,

Rate * RTT = 187848 * 0.343 = 64431,

This is a proximately equal to the max RWND,

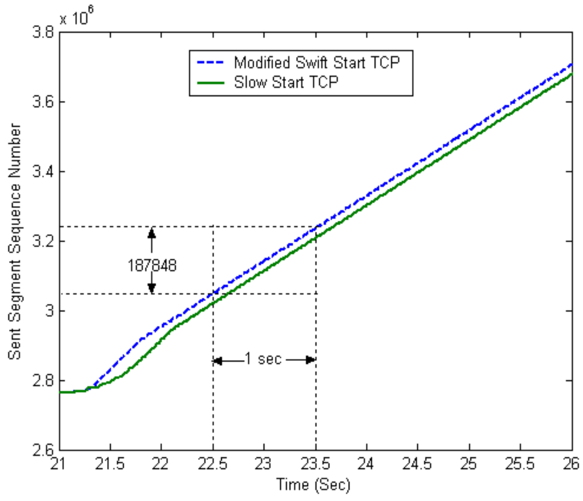


Fig-5-a . The Sent Segment Sequence Number for Slow Start TCP and Modified swift start TCP for BW = 1.5 Mbps and Path RTT= 0.11674 Sec.

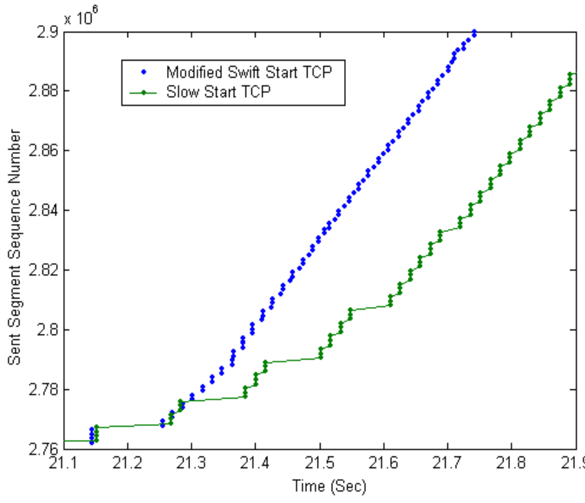


Fig-5-b. The Sent Segment Sequence Number for Slow Start TCP and Modified Swift Start TCP for BW = 1.5 Mbps and Path RTT= 0.11674 Sec

We also not that the modified swift start algorithm reaches this rate more faster than the slow start so it enhances the startup for the connection.

Fig.6 shows the received sequence number at the receiver, combined with the sent sequence number from the sender, from this Figure we have two notes. The first is that for Modified swift start the sending rate at the first and second *RTT* form the sender is equal to the to the receiving rate at the receiver this means no buffering in the routers, this is the objective of using pacing in connection to avoid over flow in the routers buffers. then slow start also tries to avoid congestion by slowly increasing the window each *ACK*, so it also tries to avoid congestion. The second note is that for slow start there are some idle intervals that is not used and so the network resources are wasted, this time wasting is avoided in modified swift start.

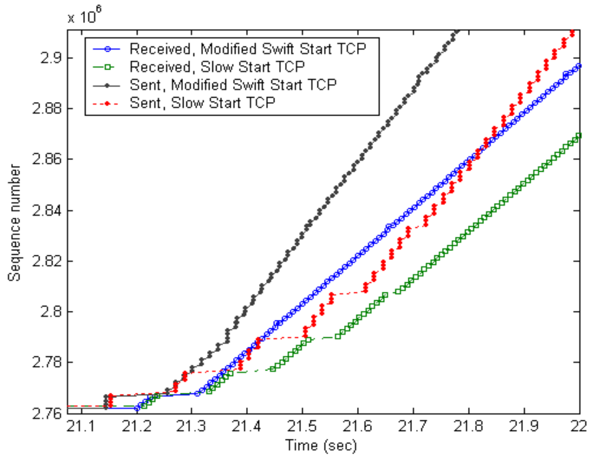


Fig-6. The Received Segment Sequence Number at the Receiver and the Sent Segment Sequence Number at the Sender for Slow Start TCP and Modified Swift Start TCP for BW = 1.5 Mbps and path RTT=0.11674 Sec

Fig. 7 shows the calculated *RTT* for both slow start and modified swift start, it is clear that the modified swift start quickly calculate the mean *RTT* because it start sending data with a high rate faster that the former, which leads to more buffering in the routers so each segment *RTT* will increase.

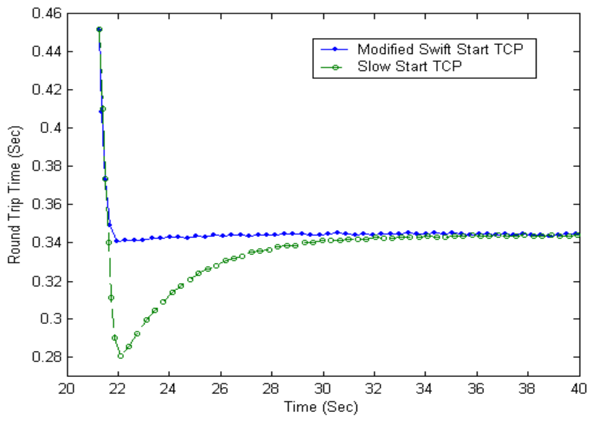


Fig-7. Calculated Mean RTT for Slow Start TCP and Modified Swift Start TCP for BW = 1.5 Mbps and Path RTT= 0.11674 Sec

4.2- Low Bandwidth, Long Delay Networks

We also test the modified swift start model on this connection with the same bandwidth but with longer delays to check the performance for long delay paths. for link delay 0.1 sec the first *RTT* was 0.31281 sec, the second *RTT* was 0.32836 sec, the first *CWND* was 58762 bytes and the second was 61683 bytes, Fig 8 shows the congestion window for this connection, its clear that the modified swift start is very faster than slow start takes in estimating the congestion window, Fig 9 shows the sent segment sequence number for *RTT* = 0.32836 Sec.
Comparing between Fig.4 with Fig 8 and Fig.5 with Fig.9, we conclude that as *RTT* increased the difference between slow start and modified swift start increases.

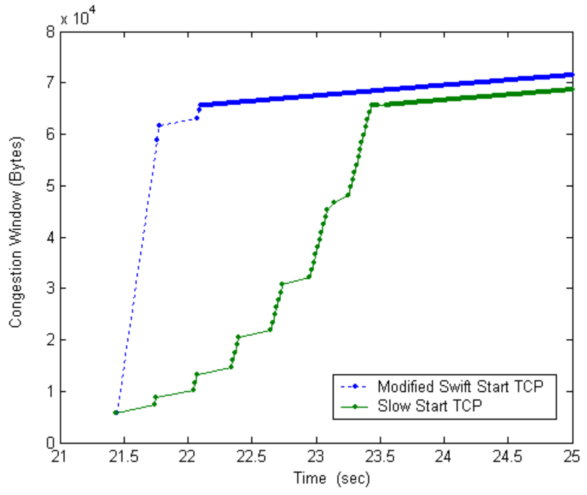
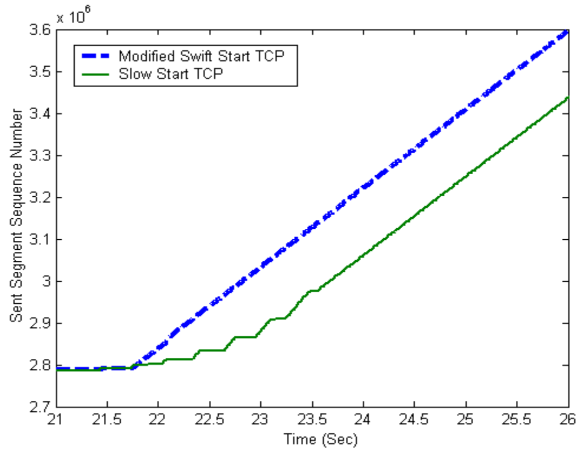


Fig-8. Congestion Window for Slow Start TCP and Modified Swift Start TCP for BW = 1.5 Mbps and Path RTT= 0.32836 Sec



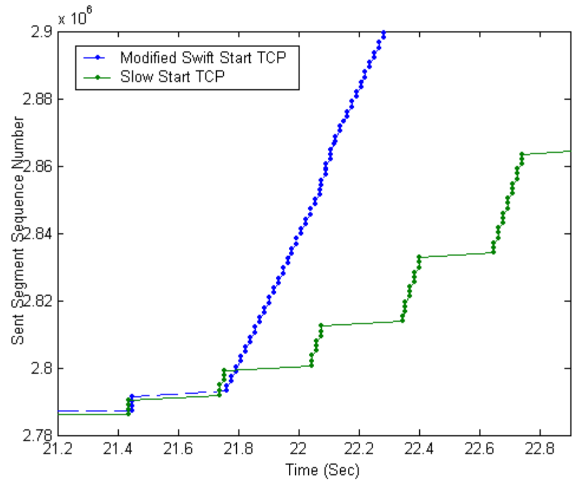


Fig- 9 . The sent segment sequence number for Slow start TCP and Modified swift start TCP for BW = 1.5 Mbps and path RTT= 0.32836 Sec

For test the modified swift start on high bandwidth networks we use the same model in Fig. 1 with *PPP* link of rate OC1 (518400000 bps) and with different *RTT*.

Table-2 shows the summary information for link bandwidth T1 and different delays. The table shows that as *RTT* increases the first estimated congestion window also increases and the connection time difference between slow start and modified swift start also increases which means the larger *RTT* the better performance of modified swift start**.**

Table–2 Information for T1 Connection for Bandwidth T1 and Different Link Delays

Link Delay	***RTT***	**First *CWND***	**Second *CWND***	**Conn. Time diff.**
0.0001	0.11674	21929	24849	0.0326
0.0005	0.11381	21379	24300	0.0236
0.001	0.11481	21567	24488	0.0362
0.005	0.12281	23070	25991	0.0893
0.01	0.13281	24948	27869	0.0902
0.05	0.21281	39977	42898	0.3697
0.1	0.31281	58762	61683	0.8290
0.2	0.51281	96333	99254	0.2451
0.3	0.71281	133903	136824	1.0378
0.4	0.91281	171474	174395	1.8887
0.5	1.11281	209045	211966	26.0561
0.6	1.31336	246719	249638	30.8050
0.7	1.51281	284186	287107	35.9797
0.8	1.71281	321757	324678	40.9113

4.3- High Bandwidth Networks

First we check for small *RTT* to test low delay–high bandwidth networks. We check for RTT= 0. 07327 sec, Fig.10 shows the congestion window for this connection, we note the large congestion window which equals 462128 bytes and 539251 bytes which were estimated by the modified swift start TCP.

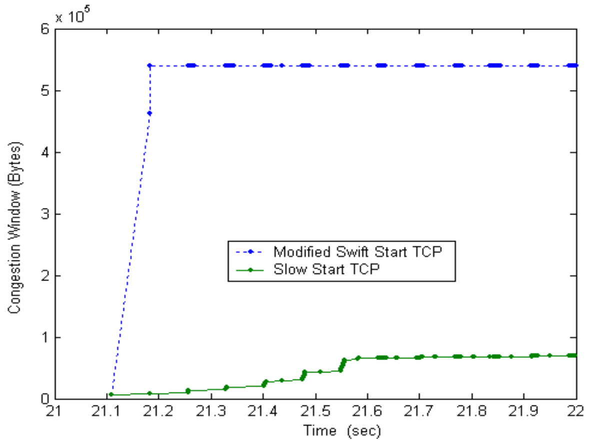


Fig-10. Congestion Window for Slow Start TCP and Modified Swift Start TCP for BW = OC1 Mbps and Path RTT= 0.07327 Sec.

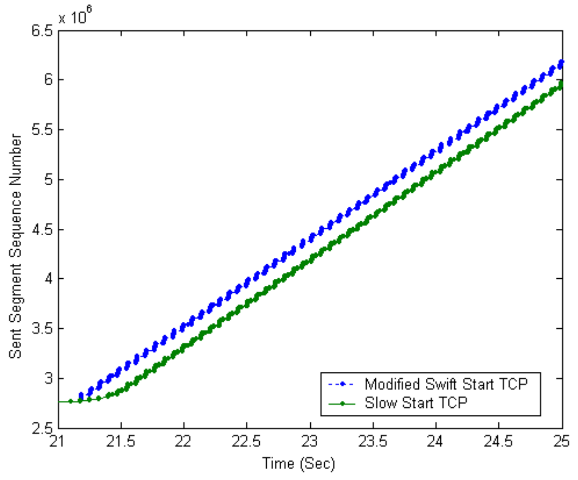
This congestion window can be calculated as follow:

$$CWND = RTT * MSS / D$$

$$D = (1460+20+7) * 8 / 51840000 = 0.0002295 \text{ sec}$$

$$CWND = 0.07327 * 1460 / 0.0002295 = 466168 \text{ bytes}$$

Fig.11 shows the sent sequence number for this connection, Fig.11 shows the effect of large congestion window on the traffic sent in the second *RTT* slow start transmits six segments only while modified swift start send a bout 44 segments, that's equal to the maximum *RWIND*. For long delay-high bandwidth networks we increase the link delay to achieve a longer *RTT*.



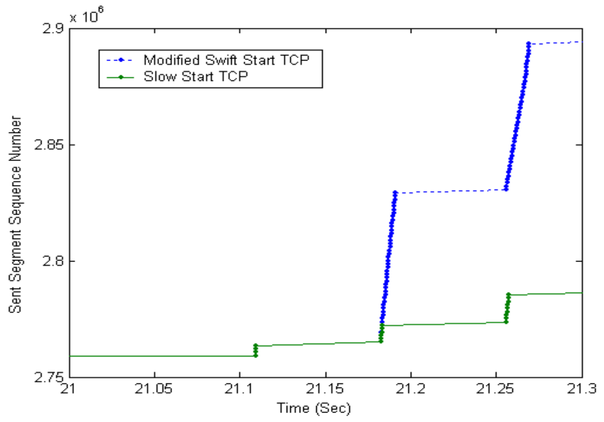


Fig- 11 . The Sent Segment Sequence Number for Slow Start TCP and Modified Swift Start TCP for BW = OC1 Mbps and Path RTT= 0.07327 Sec

Table-3 summarizes the information for OC1 connection with different RTT. The table show s that the modified swift start finished before the slow start TCP.

Table–3 Information for OC1 Connection for Bandwidth T1 and Different Link Delays

Link Delay	RTT	First CWND	Second CWND	Conn. Time diff.
0.0001	0.07327	462128	539251	0.281211
0.0005	0.07407	467174	545091	0.282627
0.001	0.07507	473481	552391	0.229694
0.005	0.08307	523939	610791	0.317579
0.01	0.09307	587011	683791	0.284387
0.05	0.17307	1091587	1267791	0.522022
0.01	0.27307	1722307	1997791	0.873903
0.2	0.47307	2983747	3457791	1.42374
0.3	0.67307	4245187	4917791	2.024241
0.4	0.87307	5506627	6377791	3.478777
0.5	1.07307	6768067	7837791	26.86863
0.6	1.27307	8029507	9297791	31.85851
0.7	1.47307	9290947	10757791	36.81996
0.8	1.67307	10552387	12217791	41.82369

5- Conclusions

This paper presents modification to what is called a Swift Start congestion control algorithm that may help the TCP better utilize the bandwidth provided by huge bandwidth long delay links. It also presents results to show a comparison of the original algorithm.
However, the modified swift start algorithm combines three algorithms to enhance the connection start up, it uses packet pair to quickly estimate the available bandwidth and calculate the congestion window, then uses pacing to avoid overflowing the networking nodes that may occur if this window sent in burst, and then uses slow start to try using the available buffers capacity on the networking nodes. The algorithm avoids the drawbacks of each algorithm by using all of them each in suitable time. It succeeded in enhancing the start up of the connection even in low speed or moderate networks.
Modified swift start maintains the core of current TCP implementations; it needs only simple modification to current TCP.

References

[1] V. Jacobson, "Congestion Avoidance and Control," Proceedings of the ACM SIGCOMM ’88 Conference, August 1988, pp. 314–329.
[2] S. Floyd and K. Fall, " Promoting the use of End – to - End Congestion Control in the Internet," IEEE / ACM Transactions on Networking, Vol.7, No.4, 1999, pp. 458-472.
[3] R.N. Shorten, D.J.Leith, J.Foy, R.Kiduff, “ Analysis and Design of Congestion Control in Synchronized Communication Networks,” Hamilton Institute, NUI Maynooth, June 20, 2003.
[4] Steven H. Low, F. Paganini, and John C. Doyle, “Internet Congestion Control,” IEEE Control Systems Magazine,Vol.22, No.1, Feb.2002, pp.28-39.
[5] J. Postl, " Transmission Control Protocol," RFC793, September 1981.
[6] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S.Singh, " Fast TCP: From Theory to Experiments," IEEE Network, Vol.19, No.1, Jan. / Feb. 2005, pp.4-11.
[7] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental Evaluation of TCP Protocols For High-Speed Networks," IEEE Trans. On Networking 2005.
[8] Y. J. Zhu and L. Jacob, " On Making TCP Robust Against Spurious Retransmissions, " Computer Communication, Vol.28, I.11, Jan. 2005 , pp.25-36.
[9] F. Paganini, Z. Wang, J.C. Doyle and S. H. Low, "Congestion Control For High Performance, Stability and Fairness in General Network," IEEE / ACM, Transactions on Networking, Vol.13, No.1, Feb.2005 , pp.43-56.
[10] C. Jin, D. X. Wei, and S. Low, " Fast TCP Modification, Architecture, Algorithm, Performance," Proc. of IEEE INFOCOM'04, 2004.
[11] C. Jin, D. Wei, and S. Low, “ Fast TCP Modification, Architecture, Algorithm, Performance,” Caltech CS Report Caltech CSTR:2003:01Q 2003.
[12] S . Floyd, and T. Henderson " The new Reno Modification to TCP Fast Recovery Algorithm,” RFC 2582, April 1999.
[13] M. Allman, W. Richard Stevens "TCP Congestion Control," RFC 2581 NASA Glenn Research Center, April 1999.

[14] V.Padmanabhan and R. Katz," TCP Fast Start: A Technique for Speeding up Web Transfers,"Globecom Sydney Australia, Nov. 1998.

[15] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 Jan. 1997.

[16] S. Floyd, "TCP and Successive Fast Retransmits," ftp://ftp.ee.lbl.gov/papers/fastretransmit.pps. Feb 1995.

[17] V. Jacobson, " Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno, " Proceedings of the British Columbia Internet Engineering Task Force, July 1990.

[18] V. Jacobson "Fast Retransmit, Message to the End2End," IETF Mailing List , April 1990.

[19] E. A. Khalil, and etc., " A Modification to Swifter Start Algorithm for TCP Congestion Control," Accepted for Publication in the, VI. International Enformatika Conference IEC 2005, Budapest, Hungary, October 26-28, 2005.

[20] E.A. Khalil, " Comparison Performance Evaluation of a Congestion Control Algorithm," Accepted for publication in the 2nd IEEE International Conference on Information & Technologies From Theory to Applications (ICTTA'06) which has been held at Damascus, Syria, April 24-28, 2006.

[21] R. El-Khoury, E. Altman, R. El-Azouzi, "Analysis of Scalable TCP Congestion Control Algorithm," IEEE Computer Communications, Vol.33, pp.41-49, November 2010.

[22] K. Srinivas, A.A. Chari, N. Kasiviswanath, "Updated Congestion Control Algorithm for TCP Throughput Improvement in Wired and Wireless Network," In Global Journal of Computer Science and Technology, Vol.9, Issue5, pp. 25-29, Jan. 2010.

[23] Carofiglio, F. Baccelli, M. Piancino, "Stochastic Analysis of Scalable TCP," Proceedings of INFOCOM, 2009.

[24] A. Warrier, S. Janakiraman, Sangtae Ha, I. Rhee, "DiffQ.: Practical Differential Backlog Congestion Control for Wireless Networks," Proceedings of INFOCOM 2009.

[25] Sangtae Ha, Injong Rhee, and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," ACM SIGOPS Operating System Review, Vol.42, Issue 5, pp.64-74, July 2008.

[26] Injong Rhee, and Lisong Xu, "Limitation of Equation Based Congestion Control," IEEE/ACM Transaction on Computer Networking, Vol.15, Issue 4, pp.852-865, August 2007.

[27] L-Wong, and L. –Y. Lau, "A New TCP Congestion Control with Weighted Fair Allocation and Scalable Stability," Proceedings of 2006 IEEE International Conference on Networks, Singapore, September 2006.

[28] Y. Ikeda, H. Nishiyama, Nei. Kato, "A Study on Transport Protocols in Wireless Networks with Long Delay," IEICE, Rep. Vol.109, No.72, pp.23-28, June 2009.

[29] Yansheng Qu, Junzhou Luo, Wei Li, Bo Liu, Laurence T. Yang, " Square: A New TCP Variant for Future High Speed and Long Delay Environments," Proceedings of 22nd International Conference on Advanced Information Networking and Applications, pp.636-643, (aina) 2008.

[30] Yi- Cheng Chan, Chia – Liang Lin, Chen – Yuan Ho, "Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth Delay Product Networks," IEICE Transactions on Communications Vol.E91-B, No.4, pp.987-997, April, 2008.

[31] T. V. Lakshman, U. Madhow, " The Performance of TCP/IP for networks with High Bandwidth delay Products and Random loss," IEEE/ACM Trans. on Networking, June 1997.

[32] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," Proce., of ACM SIGCOMM'96, Antibes, France, August 1996, pp.270-280.

[33] K. Chandrayana, S Ramakrishnan, B. Sikdar, S. Kalyanaraman, "On Randomizing the Sending Times in TCP and other Window Based Algorithm," Vol.50, Issue5, Feb. 2006, pp.422-447.

[34] C. Dah-Ming and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," Computer Networks and ISDN Systems, Vol.17, No.1, 1989, pp.1-14.

[35] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Throughput: A Simple Model and it's Empirical Validation," Proceedings of ACM SIGCOMM'98, Sept.1998, pp.303-314.

[36] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, J. Sterbenz "A Swifter Start For TCP" BBN Technical Report No. 8339, 2002.

[37] Afifi, H., Elloumi, O., Rubino, G. " A Dynamic Delayed Acknowledgment Mechanism to Improve TCP Performance for Asymmetric Links," Computers and Communications, 1998. ISCC '98. Proceedings. 3rd IEEE Symposium, on 30 June-2 July 1998, pp.188 – 192.

[38] D. D. Clark, "Window and Acknowledgement Strategy in TCP," RFC 813, July 1982.

[39] Mogul, J.C., " Observing TCP Dynamics in Real Networks," Proc. ACM SIGCOMM '92, Baltimore, MD, August 1992, pp. 305-317.

[40] Zhang, L., S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," Proc. ACM SIGCOMM'91, Zurich, Switzerland, August 1991, pp. 133-148

Ehab A. Khalil, (B.Sc'78 – M.Sc.'83 – Ph.D.'94), Got B.Sc. in the Dept. of Industrial Electronics, Faculty of the Electronic Engineering, Minufiya University, Menouf – 32952, EGYPT, in May 1978, M.Sc in the Systems and Automatic Control, with the same Faculty in Oct. 1983, Research Scholar from 1989-1994 with the Dept. of Computer Science & Engineering, Indian Institute of Technology (IIT) Bombay-400076, India, Ph.D. in Computer Network and Multimedia from the Dept. of Computer Science & Engineering, Indian Institute of Technology (IIT) Bombay-400076, India in July 1994. Lecturer, with the Dept. of Computer Science & Engineering, Faculty of Electronic Engineering, Minufiya University, Menouf – 32952, EGYPT, Since July 1994 up to now. Participated with the TCP of the IASTED Conference, Jordan in March 1998. With the TPC of IEEE IC3N, USA, from 2000-2002. Consulting Editor with the "Who's Who?" in 2003-

2004. Member with the IEC since 1999. Member with the Internet2 group. Manager of the Information and Link Network of Minufiya University, Manager of the Information and Communication Technology Project (ICTP) which is currently implementing in Arab Republic of EGYPT, Ministry of Higher Education and the World Bank. Published more than 70 research papers and article reviews in the international conferences, Journals and local newsletter.
For more details you can visit http://ehab.a.khalil.50megs.com or http://www.menofia.edu.eg/network_administrtor.asp