SIMULATION AND ANALYSIS STUDIES FOR A MODIFIED ALGORITHM TO IMPROVE TCP IN LONG DELAY BANDWIDTH PRODUCT NETWORKS

Ehab A. Khalil
Department of Computer Science & Engineering, Faculty of Electronics Engineering, Menoufiya University, Menouf, Egypt.

ABSTRACT

It is well known that TCP has formed the backbone of the Internet stability and has been well tuned over years. Today the situation has changed, that is because the internetworking environment become more complex than ever, resulting in changes in TCP congestion control are produced and still in progress. In this paper we use an analytic fluid approach in order to analyze the different features of slow start, traditional swift start, and modified swift start algorithms. We then use simulations to confirm our analytic results which are promising enough.

KEYWORDS: *TCP, congestion control, Slow Start and Swift Start algorithms, high-speed networks, long-delay bandwidth product networks.*

I. INTRODUCTION

Since more than three decades Cerf and Kahn have been initiated in their paper [1] the first work of Transmission Control Protocol (TCP), which originally defined in RFC 793 [2]. However, when a TCP connection is opened and data transmission starts, TCP uses an algorithm known as slow start to probe the network to determine the available capacity over the connection’s path. It is well known that the TCP is responsible for detecting and reacting to overloads in the Internet and has been the key to the Internet’s operational success in the last few decades. However, as link capacity grows and new Internet applications with high-bandwidth demand emerge, TCP’s performance is unsatisfactory, especially in high-speed and long-distance networks. In these networks TCP underutilizes link capacity because of its conservative and slow growth of congestion window. The congestion window governs the transmission rates of TCP [3]. TCP is often blamed that is cannot use efficiently network paths with high Bandwidth Delay Product (BDP). The BDP is of fundamental importance because it determines the required socket buffer size for maximum throughput [4].

The basic implementations of TCP are based on Jacobson's classical slow start algorithm for congestion avoidance and control [5,6]. A number of solutions have been proposed to alleviate the a aforementioned problem of TCP by changing its congestion control algorithm such as BIC-TCP [7], congestion control [8], CUBIC [9], FAST [10], HSTCP [11], H-TCP [12], LTCP [13], STCP [14], TCP-Westwood [15], TCP-Africa [16], fast retransmit, fast recovery [17-20], the new Reno Modification to TCP fast recovery algorithm [21], and Increasing TCP's Initial Window [22] which was evaluated in [23]. All these enhancements were added to TCP congestion control and others are still in progress to avoid unnecessary retransmissions and to enhance the connection efficiency without altering the fundamental underlying dynamics of TCP congestion control [24]. Other congestion control algorithms were suggested for TCP such as the delay-based approach for congestion avoidance [25] and explicit congestion notification (ECN) [26,27].

Technology trends indicate that the future Internet will have a large number of very high bandwidth links such as fiber links, and very large delay satellite links. These trends are problematic because TCP reacts adversely to increases in bandwidth or delay. Mathematical analysis of current slow start TCP congestion control algorithms reveals that, as the delay-bandwidth product increases, TCP bandwidth utilization decreases especially for large delay links. So many other congestion control algorithms were suggested to enhance the performance of TCP of high delay-bandwidth product network such as Fast TCP [18-21], TCP Fast Start [22], Explicit Control Protocol (XCP) [23], High Speed TCP [24], Quick-Start for TCP and IP [25] and others.

II. BACKGROUND

F. J. Lawas-Grodek and Diepchi T. Tran have tested the results of the Swift Start algorithm in single-flow and multiple-flow test beds under the effects of high propagation delays, various bottlenecks, and small queues sizes. Also, they've estimated the capacity and implements packet pacing; the results were that in a heavily congested link, the Swift Start algorithm would not be applicable. The reason is that the bottleneck estimation is falsely influenced by timeouts included by retransmissions and the expiration of delayed acknowledgment (ACK) timers, the causing their modified Swift Start code to fall back to regular TCP [28]. In the previous work [29-32], we've modified the traditional (original) Swift Start algorithm [33,34] to overcome its drawbacks. However, the modified Swift Start algorithm results have confirmed its succeed in improving the start up connection by quickly estimating to the available bottleneck rate in the connection path, and its performance does not affected when using Delayed Acknowledgment or acknowledges compression.

III. SLOW START OVER LONG DELAY-BANDWIDTH PRODUCT NETWORKS

Recently there are some researches investigate the congestion control and long delay bandwidth product such as [35-44]. To determine the data flow, the Slow Start TCP uses two main variables, the first is the Congestion Window (CWND) in which the sender-side is limit on the amount of data, and can transmit into the network before receiving an ACKnowledgment (ACK), the second is the Receiver's advertised window (RWND) in which the receiver-side is limit on the amount of outstanding data. The minimum of CWND and RWND governs data transmission. Another state variable is the Slow Start threshold (SSTHRESH), which 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), it equals to one segment. Each time an ACK is received; the CWND is incremented 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 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 ACK according to equation 1.

$$CWND += SMSS \times SMSS/CWND \quad \ldots\ldots\ldots\ldots\ldots (1)$$

Where: ***SMSS*** is the sender maximum segment size.

CP 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. However, there are many researches have been done such as fast retransmit; fast recovery [17-20], the New Reno Modification to TCP fast recovery algorithm [21], and increasing TCP's Initial Window [22] 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. That takes a small time to correctly estimate and begin transmitting data at the available capacity. Meanwhile, over long delay-bandwidth product networks, it may take several seconds to complete the first Slow Start and estimate available path capacity.

IV. SWIFT START ALGORITHM

The Swift Start algorithm was proposed to improve the TCP connection startup by quickly estimating the path bottleneck capacity and so the congestion window by using packet pair algorithm [45] and using packet pacing [46] to spread out the congestion window over RTT to avoid router buffers over flow. In this algorithm, the TCP connection starts with four-segment (IW = 4) which are sent in burst. However, when the acknowledgements of the segments are received, the sending TCP uses the packet pair algorithm to calculate the bottleneck capacity as follow**:**

$$\mathbf{BW = \Delta t \times SegSize} \qquad \ldots\ldots\ldots\ldots\ldots \textit{(2)}$$

$$\textit{Capacity} = \textit{BW} \times \textit{RTT} \qquad \ldots\ldots\ldots\ldots\ldots (3)$$

Where: Δt is the time delay between the arrival time of the acknowledgment of the first and second segment. Also the sending *TCP* uses pacing to spread the packets over the *RTT*.
However, the Swift Start can not work properly when combing it with some other techniques such as Delayed Acknowledgment DACK [47, 48] which is used in mostly all TCP implementations to reduce the number of pure data less acknowledgment packets sent by the receiver. DACK states that the TCP receiver will only send data less acknowledgment for every other received segment. If no segment is received within a specific time, the data less acknowledgment will be sent. The DACK algorithm will directly influence packet pair estimation, because the ACK is not sent promptly, but it may be delayed some time within the receiver, not due to congestion, so the sender can not correctly estimate the available bandwidth. Another problem facing Swift Start is the acknowledgment compression [49, 50], which causes the ACKs to be bunched up in the network path from the data receiver to the data sender. This compression will decrease the time gap between the ACKs which will lead to bandwidth over estimation. The third problem with Swift Start is that the employing packet pair algorithm does not take in account the delay that faces acknowledges in the reverse path. However, to overcome the three drawbacks mentioned above, a simple modification is considered to the original Swift Start algorithm [29-32] which is compared with other congestion control algorithms.

4.A. The Modified Swift Start Algorithm

The Modified Swift Start (MSS) algorithm aims to avoid drawbacks with the original Swift Start algorithm by modifying the packet pair algorithm, the idea behind the modification is that instead of time depending on the interval between the acknowledgments that may cause errors, it will use the time between the original messages which will be calculated by the receiver when the original messages arrive it, and then the receiver sends these information to the source when acknowledging it. The sender starts the connection by *CWND* = 4 segment, these packets are send in form of pairs, and identifies the first and the second segment of each pair 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 an acknowledgment on this message normally according to its setting. When it receives a second one, it will check whether this 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 the first one using the following equation :-

$$\Delta t = t\_seg1 - t\_seg2 \qquad \mu\,sec \qquad \ldots\ldots\ldots\ldots\ldots (4)$$

where: t_seg1 and *t_seg2* are the arrival time of the first and second segments respectively. However, when the receiver sends the second segment's acknowledgment, 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* by using the above equation (2).

4.B. How does the Modified Swift Start Overcome the Drawbacks?

If the receiver uses the *DACK* technique, it will record the first segment arrival time and wait for another segment, when it receives the second one it will calculate Δt, and whenever it sends an acknowledgment, it will send Δt along with it. However, the *DACK* dos not affect the calculation of

Δt. Also acknowledgment compression will not affect the calculation of Δt, because Δt is the time gab between the data segment itself. If *ACKs* face a delay in the reverse path, this delay will not affect Δt. because Δt is carried explicitly with in the header and not in the time delay between *ACKs*. However, error sources are avoided and the estimated capacity is the actual capacity without neither over estimation nor under estimation.

4.C. Mathematical Analysis

The purpose of the mathematical analysis is to drive a mathematical model to estimate the throughput of the transmission for both MSS (Modified Swift Start) and Slow Start. Since MSS is used to enhance the connection startup, so we would be interest in the slow start phase of the connection, and the difference between the slow start and the MSS in this phase.

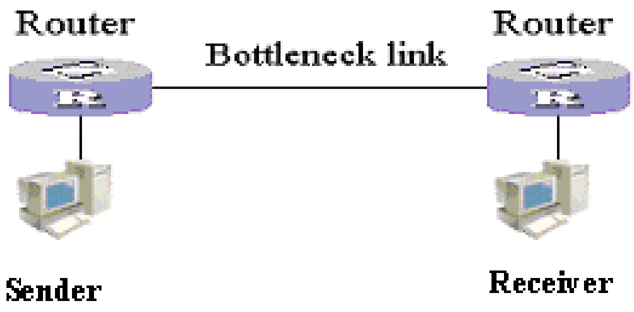

Figure 1: Topology of the network model

Figure 1 shows the topology of the network model that we've used to implement the mathematical analysis. The analysis based on the model driven in [51]. In this analysis we will ignore the 3-way handshaking, and we assume that *RTT* is constant for simplicity. This assumption is used in many researches specially when working in long delay paths, in which queuing delay is very small with respect to propagation delay see [52-54]. The following parameters are used in the analysis.

$CWND_i$	**the** congestion window at the *ith RTT*.
$CWND_1$	the initial congestion window
b	is a parameter depends on the use of *DACK* where b=1 if *DACK* is disabled and b=2 if *DACK* is enabled
γ	$= 1+1/b$
d_n	the number of data segments sent in the interval form 0 to n **RTT*
B	The throughput which is the amount of data sent in a certain time interval from 0 to *n* RTT*.
C	the bottleneck capacity in Bit/sec.
S	The segment size. We assume that all segments have the same length, this happens when the sender always has a data to send.

4.D. Slow Start Analysis

In slow start phase

$CWND_{i+1} = CWND_i + CWND_i / b$

$CWND_{i+1} = (1+1/b) * CWND_i$

$CWND_{i+1} = \gamma\, CWND_i$

$$CWND_i = \gamma^{i-1} * CWND_1$$

Let N be the *RTT* in which the congestion window is *CWND*

$$N = Log_{\gamma}\left(\frac{CWND}{CWND_1}\right) + 1 \qquad \ldots\ldots\ldots\ldots\ldots (1)$$

$d_n = CWND_1 + CWND_2 + \ldots\ldots + CWND_n$

$d_n = CWND_1 + \gamma CWND_1 + \gamma^2 CWND_1 + .. + \gamma^{n-1} CWND_1$

$$d_n = \sum_{i=1}^{n} CWND_i$$

$$d_n = CWND_1 * \frac{\gamma^n - 1}{\gamma - 1} \qquad \ldots\ldots\ldots\ldots\ldots (2)$$

Let N (d) be the number of *RTT* needed to send d segments

$$N(d) = \log_\gamma\left(\frac{d*(\gamma-1)}{CWND_1} + 1\right) \qquad \ldots\ldots\ldots\ldots.. (3)$$

From equation………..(2)

$$d_n = \frac{CWND_1 * \gamma^n - CWND_1}{\gamma - 1}$$

$$d_n = \frac{\gamma * CWND_n - CWND_1}{\gamma - 1} \qquad \ldots\ldots\ldots\ldots (4)$$

$$CWND_i = \frac{d_i(\gamma-1) + CWND_1}{\gamma} \qquad \ldots\ldots\ldots\ldots (5)$$

Equation (5) was driven in [40].

B (n) = d_n/ (n * *RTT*)

$$B(d) = \frac{d_n}{RTT * \log_\gamma\left(\frac{d_n*(\gamma-1)}{CWND_1} + 1\right)} \qquad \ldots..(6)$$

And

$$B(CWND) = \frac{CWND_1 \frac{\gamma^n - 1}{\gamma - 1}}{RTT * Log_\gamma\left(\frac{CWND_n}{CWND_1}\right)}$$

$$B(CWND) = \frac{\gamma * CWND - CWND_1}{RTT\,(\gamma-1)\left(\log_\gamma\left(\frac{CWND}{CWND_1}\right) + 1\right)} \qquad \ldots\ldots\ldots. (7)$$

$$B(n) = \frac{CWND_1}{n * RTT} * \frac{\gamma^n - 1}{\gamma - 1} \qquad \ldots\ldots\ldots\ldots (8)$$

Let Ns be the number of *RTT* in which the *CWIND* reaches the *SSTHRESH.*

$$Ns = \log_\gamma\left(\frac{ssthresh}{CWND_1}\right) + 1$$

In the slow start phase n is the number of *RTT* for *CWND* to reach the *SSTHRESH.*
The amount of data sent before reaching the *SSTHRESH* will be

$$B(SSTHRESH) = \frac{d_{ssthresh}}{\mathrm{RTT} * N\mathrm{s}}$$

where: $d_{ssthresh}$ is the number of segments sent until the congestion window reaches *SSTHRESH*
From ……equation (4)

$$d_{ssthresh} = \frac{\gamma * \mathrm{s}sthresh - CWND_1}{\gamma - 1}$$

$$B(SSTHRESH) = \frac{\gamma * \mathrm{s}sthresh - CWND_1}{\mathrm{RTT}\ (\gamma - 1)\left(\log_\gamma\left(\frac{\mathrm{s}sthresh}{CWND_1}\right) + 1\right)} \qquad \text{………………..(9)}$$

When tn > Ts

$$CWND_{i+1} = CWND_i + 1/\ CWND_i$$

4.E. Modified Swift Start

In case of modified swift start let $CWND_1$ is the initial congestion window and the inter arrival delay between the two packets arriving the receiver is τ, the segment size is S. So, after the first *RTT CWND* will be

$CWND_2 = RTT / \tau$

Then the TCP will use the slow start to increase the congestion window so: In the model of Figure 1, τ = frame length / C For PPP connections the frame length equals to S + IP_header_length + frame_header_length

$\tau = 8 * (S+27) / C$

$$CWND_i = \begin{cases} CWND_1 & i = 1 \\ \gamma^{i-2} * RTT / \tau & i \geq 2 \end{cases}$$

$$d_n = CWND_1 + \frac{\mathrm{RTT}}{\tau} * \frac{\gamma^{n-1} - 1}{\gamma - 1} \qquad \text{…………..…..(10)}$$

For $RTT/\tau \leq SSTHRESH$.

This condition is to grant that the connection is in slow start. Because if RTT/τ > *SSTHRESH* the congestion avoidance will start, and the slow start time in this case is only one *RTT*.

$$d_n = CWND_1 + \frac{\gamma * CWND_n - RTT / \tau}{\gamma - 1} \qquad \text{……..…(11)} \qquad \text{for } n>2$$

From equation (10), the number of *RTTs* needed to send d segments is

$$\mathrm{N}\ (\mathrm{d}) = \log_\gamma\left((d - CWND_1)\frac{\tau * (\gamma - 1)}{RTT} + 1\right) + 1$$

$$\mathrm{N}(CWND) = \log_\gamma\left(\frac{CWND * \tau}{RTT}\right) + 2$$

$$B(n) = \frac{CWND_1 + \frac{\mathrm{RTT}}{\tau} * \frac{\gamma^{n-1} - 1}{\gamma - 1}}{RTT * n}$$

$$B(cwnd) = \frac{CWND_1 + \dfrac{\gamma * CWND_n - RTT / \tau}{\gamma - 1}}{RTT * \log_\gamma\left(\dfrac{CWND * \tau}{RTT}\right) + 2 * RTT}$$

$$B(\text{s}sthresh) = \frac{CWND_1 + \dfrac{\gamma * sssthresh - RTT / \tau}{\gamma - 1}}{RTT * \log_\gamma\left(\dfrac{\text{s}sthresh * \tau}{RTT}\right) + 2 * RTT} \text{ For } CWND_i \neq CWND_1$$

$$B(d) = \frac{d}{RTT * \left(\log_\gamma\left((d - CWND_1)\dfrac{\tau * (\gamma - 1)}{RTT} + 1\right) + 1\right)}$$

V. SIMULATION AND RESULTS.

The Modified Swift Start model has been implemented using Opnet modeler [55], to compare the performance results with that of the original swift start and the slow start in deferent network conditions of bandwidth and path delay. The comparison between them implemented using a single.

5.A Single Flow

5.A.a) Low Delay-Bandwidth Product Networks

The network model shown in Figure 1 implemented to study the performance of swift start TCP and compare it with the traditional (original) swift start and the slow start using single flow between 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 are shown in Table-1. In the simulation both the sender and the receiver uses DACK. This configuration has been used to study the difference between the original and modified swift start. The sender and the receiver are connected to the routers through a 100 Mbps Ethernet connections.

Table-1 TCP Parameters of the sender and receiver

Maximum Segment Size	1460 Bytes
Receive Buffer	100000 Bytes
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

Both of the routers are CISCO 3640 with forwarding rate 5000 packets/second and memory size 265 MB. The two routers 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.
Figure 2 shows the simulation and the analytical results of the congestion window for slow start TCP, traditional and Modified swift start TCP when the bottleneck data rate is 1.544 Mbps (T1) and the path RTT is 0.11674 second, which is low rate, low delay network. First we note some differences between the analytical results and the simulation results, these differences because we use a fixed RTT (RTT=0.11674 sec which is the initial RTT) in the analysis, meanwhile the RTT actually changes according to CWND due to queuing delay. We also note that the difference increases as time increases, this is logical because in the first few RTTs CWND is very small so the RTT is around the initial RTT, however the results are very close in the first few RTTs. Anyway this difference is not important for us because we are concerned on the first few RTTs.

It is clear that the modified swift start is faster and better than slow start TCP in estimating the path congestion window which is = 21929 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 + D_Q

= (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

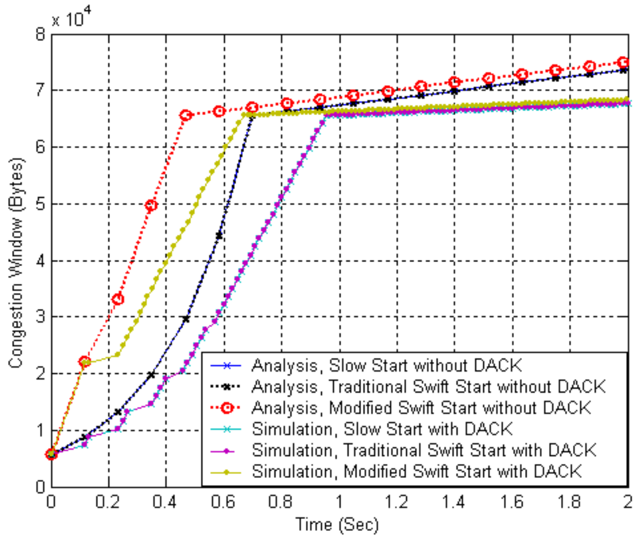

Figure 2 Congestion window for BW = 1.5 Mbps and path RTT= 0.11674 Sec

Obviously, the result in the simulation shows that the delay difference is 0.007772 sec and the CWND is 21929 bytes, these results are very close to the mathematical results. This difference between the results because in the calculation we've neglected the processing delay which may affect the value of D and so decrease CWND. The simulation also shows that after estimating the congestion window in the first RTT, the swift start stopped and the slow start runs normally, Figures 3-a and 3-b show the sent segment sequence number for this connection. It is shown that the three algorithms start the connection by sending 4 segments, after 1 RTT (0. 11674 sec) each of the slow start and traditional (original) swift start send 6 segments with in the second RTT, while the modified swift start send a

large number of segments because of its large congestion window which is 21929 bytes which is about 14 segments, these segments were paced along the second RTT, until the sender receives an other ACK that indicates that 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 Figure 3-a shows that after a certain time both algorithms reaches a constant transmission rate, we roughly calculate this rate, Transmission rate = 187848 bytes / sec

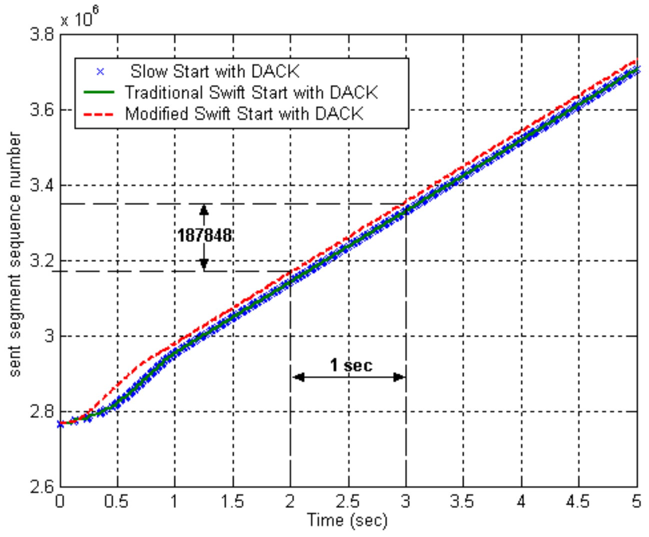

Figure 3-a the sent segment sequence number for BW = 1.5 Mbps and path RTT= 0.11674 Sec

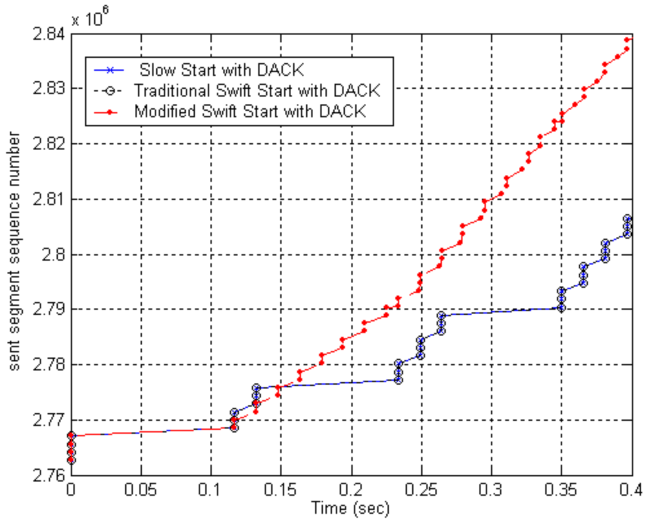

Figure 3-b the sent segment sequence number for BW = 1.5 Mbps and path RTT= 0.11674 Sec

5.A.b) Low Bandwidth, Long Delay networks

We’ve also tested the traditional and modified swift start models 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 RTT was 0.31343 sec, and the estimated CWND was 58878 bytes a. Figure 4 shows the congestion window for this connection, it's clear that the modified swift start is faster than slow start.

5.A.c) High Bandwidth Networks

To compare between the three algorithms on high bandwidth networks we’ve used the same model in Figure 1 with PPP link of rate OC1 (518400000 bps) and with different RTT. First we check for short RTT to test low delay–high bandwidth networks. We’ve checked for RTT= 0. 07307 sec.

Figure 4 shows the congestion window for this connection; we've noted that the large congestion window which equals 460867 bytes which was estimated by the modified swift start TCP. This congestion window can be calculated as follow

$$CWND = RTT * MSS / D$$

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

$$CWND = 0.07307 * 1460 / 0.0002295 = 464846 \text{ bytes}$$

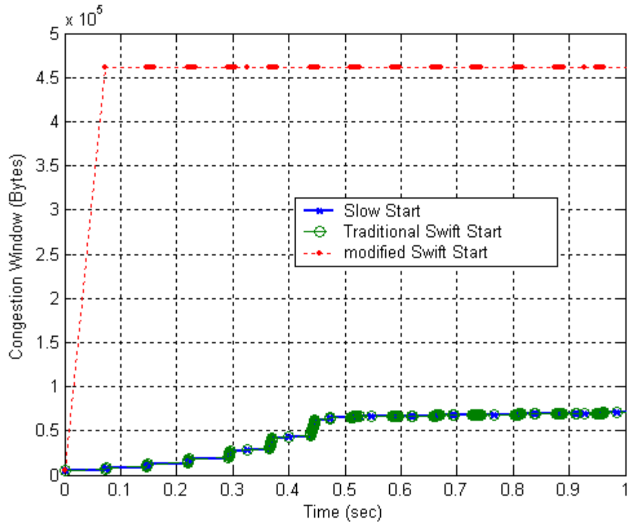

Figure 4 Congestion window for BW = OC1 Mbps and path RTT= 0.07327 Sec

Figure 5 shows the sent sequence number for this connection, also, 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.

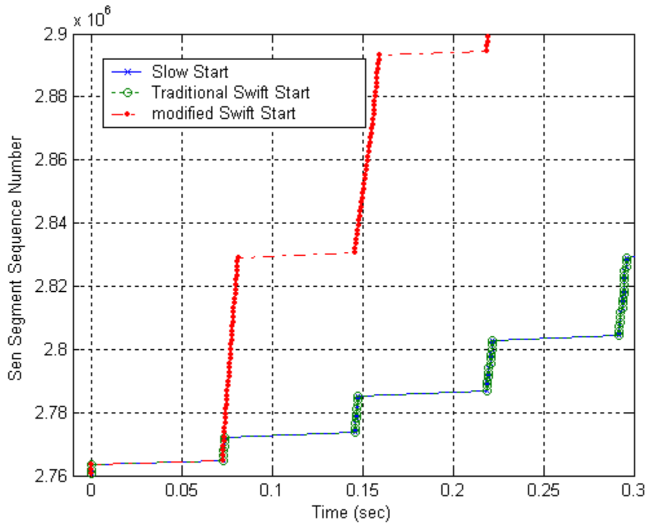

Figure 5 the sent segment sequence number for BW = OC1 Mbps and path RTT = 0.07327 Sec.

VI. CONCLUSION

The paper presents methods of simulation and analysis for the slow start, traditional and modified swift start algorithms. The results are compared and confirm that the modified algorithm promised enough. We've to mention here that the modified swift start algorithm maintains the core of current TCP.

REFERENCES

[1].V. Cerf, and R. Kahn, A Protocol for Packet Network Intercommunication, IEEE Trans. on Comm., Vol.22, No.5, pp.637-648, May 1974.
[2].J. Postel, Transmission Control Protocol; RFC793, Internet Request for Comments 793, Sept. 1981.
[3].Sangtae Ha, Long Le, Injong Rhee, Lisong Xu, Impact of Background Traffic on Performance of High-Speed TCP Variant Protocols, Computer Networks, Vol.51, Issue 7, May 2007.
[4].M. Jain, R.S. Prasad, C. Dovrolis, The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-Sizing, CERCS, GIT-CERCS-03-02, Institute of Technology, 2003.
[5].V. Jacobson, Congestion Avoidance and Control, Proceedings of the ACM SIGCOMM '88 Conference, pp. 314–329, August 1988.
[6].M. Allman, W. Richard Stevens, TCP Congestion Control, RFC 2581, NASA Glenn Research Center, April 1999.
[7].Lisong Xu, Khaled Harfoush, Injong Rhee, Binary Increase Congestion Control For Fast, Long Distance Networks, Proceedings of IEEE INFOCOM, March 2004.
[8].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.
[9].Injong Rhee, 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.
[10]. Cheng Jin, David X. Wei, Steven H. Low, Fast TCP: Motivation, Architecture, Algorithms, Performance, Proceedings of IEEE NFOCOM, March 2004.
[11]. Sally Floyd, High-Speed TCP For Large Congestion Windows, RFC 3649, December 2003.
[12]. Douglas Leith, Robert Shorten, H-TCP Protocol For High-Speed Long Distance Networks, International Workshop on Protocols For Fast Long-Distance Networks, February 2004.
[13]. Sumitha Bhandarkar, Saurabh Jain, A. L. Narasimha Reddy, Improving TCP Performance in High Bandwidth RTT Links Using Layered Congestion Control, International Workshop on Protocols For Fast Long-Distance Networks, February 2005.
[14]. Tom Kelly, Scalable TCP: Improving Performance on High-Speed Wide Area Networks, ACM SIGCOMM Computer Communication Review, 2003.
[15]. Ren Wang, Kenshin Yamada, M. Yahya Sanadidi, Mario Gerla, TCP with Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes, IEEE Journal on SACs Vol23, No.2, 2005.
[16]. Ryan King, Richard Baraniuk, Rudolf Riedi, Evaluating and Improving TCP-Africa: An Adaptive and Fair rapid Increase Rule for Scalable TCP, International Workshop on Protocols For Fast Long-Distance Networks, February 2005.
[17]. W. Stevens, TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001 Jan. 1997.
[18]. S. Floyd, TCP and successive fast retransmits, ftp://ftp.ee.lbl.gov/papers/fastretransmit.pps. Feb 1995.
[19]. V. Jacobson, Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno, Proceedings of the British Columbia Internet Engineering Task Force, July 1990.
[20]. V. Jacobson Fast Retransmit, Message to the End2End, IETF Mailing List , April 1990.
[21]. S. Floyd, and T. Henderson, The new Reno Modification to TCP Fast Recovery Algorithm, RFC 2582, April 1999.
[22]. M. Allman, S. Floyd, C. Partridge, Increasing TCP's Initial Window, RFC 2414, September 1998.
[23]. M. Allman, C. Hayes, and S. Ostermann, An Evaluation of TCP with Larger Initial Windows, ACM Computer Communication Review, 8(3), July 1998.
[24]. Y. J. Zhu, and L. Jacob, On Making TCP Robust Against Spurious Retransmissions, Computer Communications, Vol.28, Issue 1, pp.25-36, Jan. 2005.
[25]. Raj Jain, A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks, ACM Computer Communication Review, 19(5):56–71, Oct. 1989.
[26]. K. Ramakrishnan, S. Floyd, A Proposal to add Explicit Congestion Notification (ECN) to IP, RFC 2481, January 1999.
[27]. K. Ramakrishnan, S. Floyd, and D. Black, The addition of explicit congestion notification (ECN) to IP, IETF, RFC3168, September 2001.
[28]. Frances J. Lawas-Grodek and Diepchi T. Tran, Evaluation of Swift Start TCP in Long-Delay Environment, NASA/TM-2004-212938, Glenn Research Center, Cleveland, Ohio October 2004.

[29]. E. A. Khalil, and etc., A Modification to Swifter Start Algorithm for TCP Congestion Control, Proceedings of VI. International Enformatika Conference IEC 2005, Budapest, Hungary, October 26-28, 2005.
[30]. 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.
[31]. E. A. Khalil, , A Modified Congestion Control Algorithm for Evaluating High BDP Networks, Accepted for publication in the International Journal of Computer Science and Network Security (IJCSNS), Vol.10, No.11, November 2010.
[32]. E. A. Khalil, , A Proposal Algorithm for TCP Congestion Control, Accepted for publication in the International Journal of Computer Science and Information Security, Vol.8, No.8, November 2010.
[33]. C. Partridge, D. Rockwell, M. Allman, R. Krishnan, J. Sterbenz, A Swifter Start For TCP, BBN Technical Report No. 8339, 2002.
[34]. Frances J. Lawas-Grodek and Diepchi T. Tran, Evaluation of Swift Start TCP in Long-Delay Environment, Glenn Research Center, Cleveland, Ohio October 2004.
[35]. 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.
[36]. 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.
[37]. Carofiglio, F. Baccelli, M. Piancino, Stochastic Analysis of Scalable TCP, Proceedings of INFOCOM, 2009.
[38]. Warrier, S. Janakiraman, Sangtae Ha, I. Rhee, DiffQ.: Practical Differential Backlog Congestion Control for Wireless Networks, Proceedings of INFOCOM 2009.
[39]. 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.
[40]. 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.
[41]. 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 Newtorks, Singapore, September 2006.
[42]. 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.
[43]. 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.
[44]. 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.
[45]. Ningning Hu, Peter Steenkiste, Estimating Available Bandwidth Using Packet Pair Probing, CMU-CS-02-166 School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213 September 9, 2002.
[46]. Aggarwal, A.; Savage, S.; and Anderson, T., Understanding the Performance of TCP Pacing, Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1157-1165, vol. 3, 2000.
[47]. 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. Third IEEE Symposium pp.188 – 192, on 30 June-2 July 1998.
[48]. D. D.Clark, Window and Acknowledgement Strategy in TCP, RFC 813, July 198.
[49]. Mogul, J.C., Observing TCP Dynamics in Real Networks, Proc. ACM SIGCOMM '92, pp. 305-317, Baltimore, MD, August 1992.
[50]. 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, pp. 133-148, Zurich, Switzerland, August 1991.
[51]. Neal Cardwell, Stefan Savage, Thomas Anderson Modeling TCP Latency, Department of Computer Science and Engineering University of Washington.
[52]. E. Altman, J. Bolot, P. Nain, D. Elouadghiri- M. Erramdani, P. Brown, and D. Collange, Performance Modeling of TCP/IP in a Wide-Area Network, 34th IEEE Conference on Decision and Control, Dec 1995.

[53]. Eitan Altman, Konstantin Avrachenkov, Chadi Barakaty, A Stochastic Model of TCP/IP with Stationary Random Losses, National Research Institute in Informatics and Control (INRIA), IEEE/ACM Transactions on Networking (TON) April 2005.
[54]. D. Leith_, R. Shorten, H-TCP: TCP for high-speed and long-distance networks, Hamilton Institute, NUI Maynooth , www.hamilton.ie/net/htcp3.pdf
[55]. Opnet web site http://www.opnet.com

Authors Biography

Ehab A. Khalil, (B.Sc'78 – M.Sc.'83 – Ph.D.'94), B.Sc. in the Dept. of Industrial Electronics, Faculty of the Electronic Engineering, Menoufiya 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 1988-1994 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, Menoufiya 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 Menoufiya 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