# SIMULATION-BASED COMPARISONS OF TCP CONGESTION CONTROL

Ehab A. Khalil
Department of Computer Science & Engineering, Faculty of Electronics Engineering,
Menoufiya University, Menouf-32952, EGYPT

***ABSTRACT***

*It is well known that the Transmission Control Protocol (TCP) congestion control has formed the backbone of the Internet stability and has 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 without altering the fundamental underlying dynamics of TCP congestion control. This paper is an attempt to covers and compares the various aspects of congestion control algorithms which have been published and considered. It includes TCP congestion control motivation, architecture, algorithms, and performance.*

***KEYWORDS*****:** *TCP congestion control, Slow Start algorithm, Modified Swift Start, performance comparisons of different TCP congestion control.*

## I. INTRODUCTION

Since more than three decades Cerf and Kahn initiated the first work of Transmission Control Protocol (TCP) [1], the Internet Engineering defined the TCP standard in the Request of Comment (RFC) standards document number 793 (RFC 793) [2]. Then, the original specifications written in 1981 was based on earlier research and experimentation in the original Advanced Research Project Agency Network (ARPANET). In 1986 V. Jacobson observed what is called by congestion collapse, and well known slow-start and congestion avoidance algorithms proposed in 1988 [3], however, the design of TCP was heavily influenced by what has come to be known as the end-to-end argument [4,5]. So, the networking community has proposed hug enhancement and optimization to the original proposition in order to make TCP more efficient in large variety of network conditions and technologies [6,7]; wireless links, asymmetric links, and very high speed and long delay links [8-10]. Despite the fact that TCP is a stable and mature protocol, and has been well tuned over years, small changes in its congestion control are still in progress to avoid unnecessary retransmit timeouts, to reverse the responses to reordered packets, to allow viable mechanisms for corruption notification, ..etc., without altering the fundamental underlying dynamics of TCP congestion Control [11].
However, 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 consideration 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 [12]. During the last two decades, several congestion control algorithms have been proposed to achieve network stability, fair bandwidth allocation and high resource utilization. However, several more aggressive versions of the TCP have been proposed which leave the Additive Increase Multiplicative Decrease (AIMD) paradigm to achieve network stability, fair bandwidth allocation and high resource utilization, so there are many up to date and old research papers have been published [13-53], and same or more 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. We have to mention here that the motivation is to adapt TCP to networks with very large bandwidth delay products.
The organization of manuscript is as follow, in section 2, the background and motivation would be presented. Section 3 comparison between congestion control simulation results and describes their main features and limitations. The conclusion is given in section 4.

## II. 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 controls 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 [54]. The last two algorithms were developed to overcome the short comings of earlier implementations, like TCP Tahoe [55], 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 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 [56] 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 [57-59]. The next section will highlight the comparison of different versions of TCP.

### 2.1. A Transmission Control Protocol and Congestion Control

TCP congestion control was introduced into the Internet in the late 1980's by Van Jacobson, roughly eight years after the TCP/IP protocol stack had become operational. Immediately preceding this time, the Internet was suffering from congestion collapse hosts would send their packets into the Internet as fast as the advertised window would allow, congestion would occur at some router causing

packets to be dropped, and the hosts would timeout and retransmit their packets, resulting in even more congestion. Broadly speaking, the idea of TCP congestion control is for each source to determine how much capacity is available in the network, so it knows how many packets it can safely have in transit. Once a given source has this many packets in transit, it uses the arrival of an acknowledgment (ACK) as a signal that one of its packets has left the network, and it is therefore safe to insert a new packet into the network without adding to the level of congestion. By using ACKs to pace the transmission of packets, TCP is said to be self clocking. Of course, determining the available capacity in the first place is no easy task. To make matters worse, because other connections come and go, the available bandwidth changes over time, meaning that any given source must be able to adjust the number of packets it has in transit. The rest of this section describes the algorithms used by TCP to address these, and other, problems.

## 2.2. B Additive Increase/Multiplicative Decrease (AIMD)

AIMD, Additive Increase, Multiplicative Decrease. Congestion control algorithm. (RFC 2914) In the absence of congestion, the TCP sender increases its congestion window by at most one packet per Round Trip Rime (RTT). In response to a congestion indication, the TCP sender decreases its congestion window by half. More precisely, the new congestion window is half of the minimum of the congestion window and the receiver's advertised window. A congestion control strategy that only decreases the window size is obviously too conservative. We also need to be able to increase the congestion window to take advantage of newly available capacity in the network. This is the ``additive increase'' part of the mechanism, and it works as follows. Every time the source successfully sends a CongestionWindow's worth of packets that is, each packet sent out during the last RTT has been ACK'ed it adds the equivalent of one packet to CongestionWindow. This linear increase is illustrated in Figure 1.

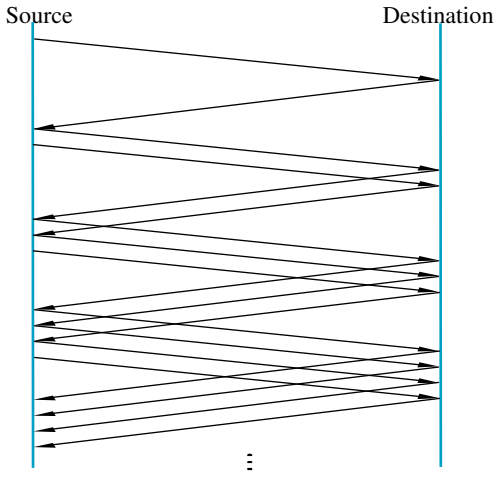


**Figure1** Packets in Transit during Additive Increase: Add one Packet each RTT.

Note that in practice, TCP does not wait for an entire window's worth of ACKs to add one packet's worth to the congestion window, but instead increments CongestionWindow by a little for each ACK that arrives. Specifically, the congestion window is incremented as follows each time an ACK arrives:

$$\text{Increment} = (\text{MSS} \times \text{MSS})/\text{CongestionWindow}$$
$$\text{CongestionWindow} \mathrel{+}= \text{Increment}$$

That is, rather than incrementing CongestionWindow by an entire MSS each RTT, we increment it by a fraction of Maximum Segment Size (MSS) every time an ACK is received. Assuming each ACK acknowledges the receipt of MSS bytes, then that fraction is MSS/CongestionWindow.

This pattern of continually increasing and decreasing the congestion window continues throughout the lifetime of the connection. In fact, if you plot the current value of CongestionWindow as a function of time, you get a ``sawtooth" pattern, as illustrated in Figure 2. The important thing to understand about additive increase/multiplicative decrease is that the source is willing to reduce its congestion window at a much faster rate than it is willing to increase its congestion window. This is in contrast to an additive increase/additive decrease strategy in which the window in incremented by 1 packet when an ACK arrives and decreased by 1 when a timeout occurs. It is has been shown that additive increase / multiplicative decrease is a necessary condition for a congestion control mechanism to be stable.

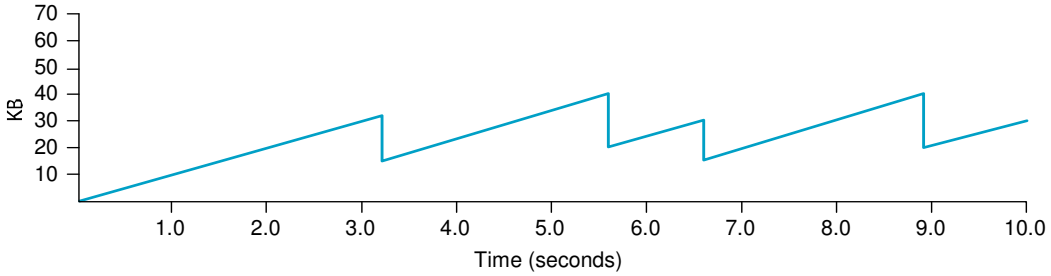


**Figure 2** Typical TCP sawtooth pattern.

## 2.3. C Slow Start Algorithm

Simply slow start operates by observing that the rate at which new packets should be injected into the network is the rate at which the acknowledgments are returned by the other end. To control this, slow start uses the congestion window "CWND". When a new connection is established with another host, the congestion window is initialized to one segment (i.e., the segment size announced by the other end, or the default, typically 536 or 512). Each time an ACK is received, the congestion window is increased by one segment as shown in Figure 1. The sender can transmit up to the minimum of the congestion window “CWND” and the advertised window “RWND”. The former is based on the sender's assessment of perceived network congestion; the latter is related to the amount of available buffer space at the receiver for this connection. The sender starts by transmitting one segment and waiting for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is acknowledged, the congestion window is increased to four. This provides an exponential growth, although it is not exactly exponential because the receiver may delay its ACKs, typically sending one ACK for every two segments that it receives. At some point the capacity of the Internet can be reached, and an intermediate router will start discarding packets. This tells the sender that its congestion window has gotten too large.

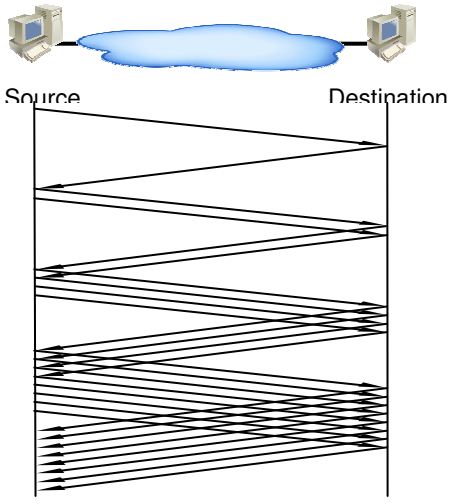


**Figure 3** Packets in Transit During Slow Start.

## 2.4. D Congestion Avoidance

The assumption of the algorithm is that packet loss caused by noise is very small (much less than 1%); therefore the loss of a packet signals congestion somewhere in the network between the source and destination. There are two indications of packet loss: a timeout occurring and the receipt of duplicate ACK as mentioned above. Congestion avoidance and slow start are independent algorithms with different objectives. But when congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to get things going again. In practice they are implemented together. Congestion avoidance and slow start require that two variables be maintained for each connection: a congestion window, "CWND", and a slow start threshold size, "ssthresh". The combined algorithm operates as follows:

1. Initialization for a given connection sets CWND to one segment and ssthresh to 65535 bytes.
2. The TCP output routine never sends more than the minimum of CWND and the receiver's advertised window RWND.
3. When congestion occurs (indicated by a timeout or the reception of duplicate ACKs), one-half of the current window size (the minimum of CWND and RWND, but at least two segments) is saved in ssthresh. Additionally, if the congestion is indicated by a timeout, CWND is set to one segment (i.e., slow start)
4. When the other end acknowledges new data, increase CWND, but the way it increases depends on whether TCP is performing slow start or congestion avoidance. If CWND is less than or equal to ssthresh, TCP is in slow start; otherwise TCP is performing congestion avoidance.

Figure 4 shows the congestion window for slow start and congestion avoidance. Slow start has CWND begin at one segment, and be incremented by one segment every time an ACK is received. As mentioned earlier, this opens the window exponentially: send one segment, then two, then four, and so on. When a timeout occurs, the congestion avoidance starts. The congestion avoidance increases the CWND be according to equation (1) as specified in [60].

$$\mathbf{CWND \mathrel{+}= SMSS * SMSS/CWND} \quad \mathbf{(1)}$$

Each time an ACK is received, Where SMSS is the Sender Maximum Segment Size and CWND is maintained in bytes. This is a linear growth of CWND, compared to slow start's exponential growth. The increase in CWND should be at most one segment each round-trip time RTT (regardless how many ACKs are received in that RTT), whereas slow start increments CWND by the number of ACKs received in a round-trip time.

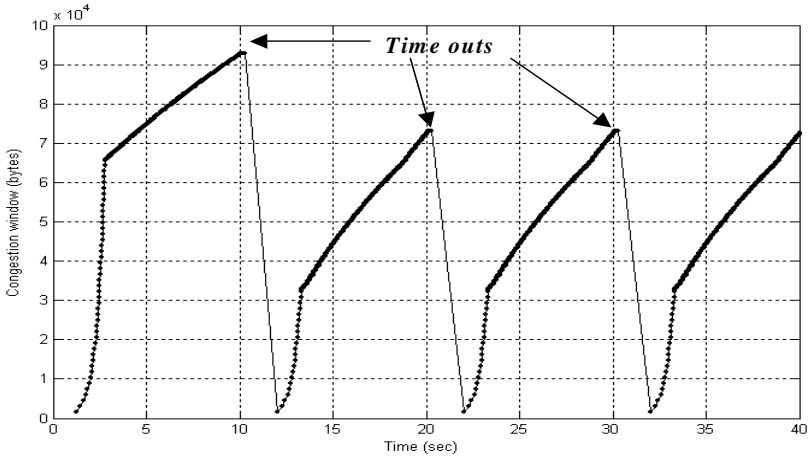


**Figure 4** The Congestion Window versus Time for Slow Start and Congestion Avoidance.

## 2.5.E Fast Retransmit (TCP Tahoe)

Modifications to the congestion avoidance algorithm were proposed in 1990 [60], before describing the change, realize that TCP may generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received. This duplicate ACK should not be delayed.

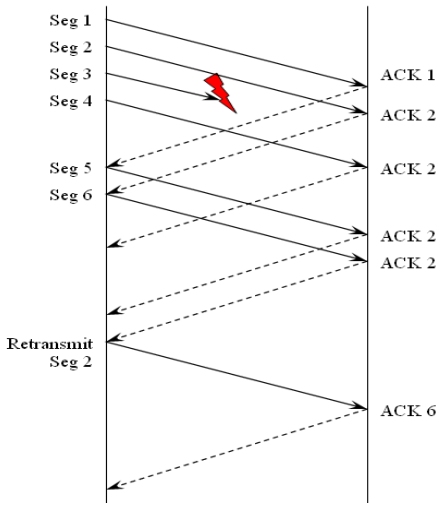


**Figure 5** Fast Retransmission

The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire, this is shown in Figure 5.
Figure 6 shows the congestion window versus time for TCP Tahoe, it's the same as Figure 4 because Tahoe modification only changes how to detect time out, but the TCP behavior responding to that time out does not changed.

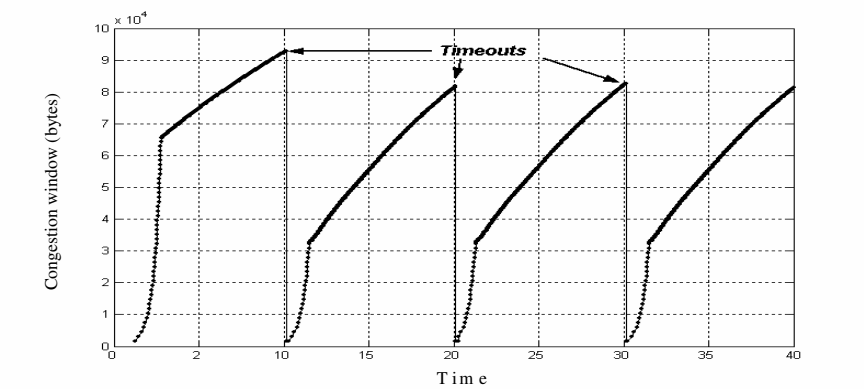


**Figure 6** Congestion Window versus Time for TCP Tahoe

## 2.6.F Fast Recovery (TCP Reno)

After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed. This is the fast recovery algorithm whose behavior is shown in Figure 7 [61].

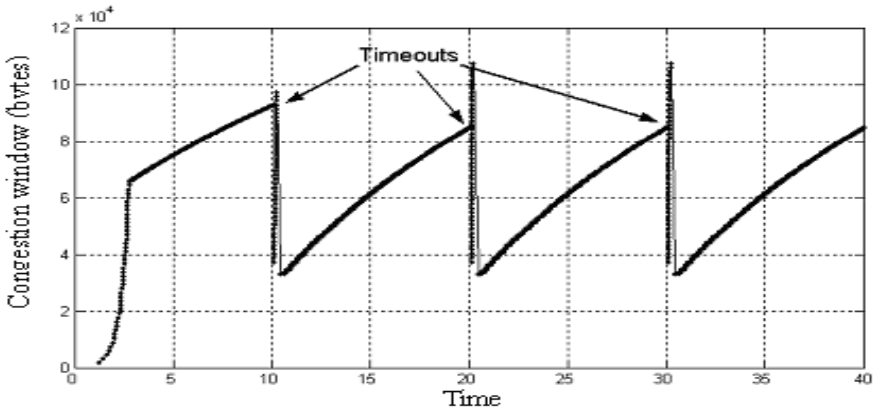


**Figure 7** Congestion Window versus Time for TCP Reno

It is an improvement that allows high throughput under moderate congestion, especially for large windows .The reason for not performing slow start in this case is that the receipt of the duplicate ACKs tells TCP more than just a packet has been lost. Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer. That is, there is still data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow start.

The fast retransmit and fast recovery algorithms are usually implemented together as follows [60].

1-When the third duplicate ACK in a row is received, set ssthresh to one-half the current congestion window, CWND, but no less than two segments. Retransmit the missing segment. Set CWND to ssthresh plus 3 times the segment size

$$\mathbf{CWND = ssthresh + 3 * SMSS} \qquad \mathbf{(2)}$$

This inflates the congestion window by the number of segments that have left the network and which the other end has cached.

2-Each time another duplicate ACK arrives, increment CWND by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of CWND.

3-When the next ACK arrives that acknowledges new data, set CWND to ssthresh (the value set in step 1). This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

Figure 8 shows the sent segment sequence number for slow start TCP, TCP Tahoe and TCP Reno. It is clear that TCP Reno is faster than TCP Tahoe and TCP Tahoe is faster than slow start only.

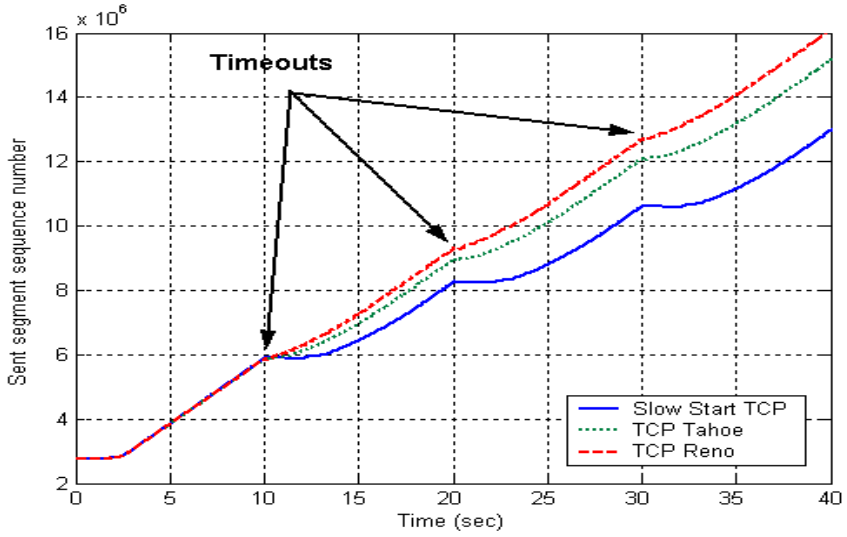


**Figure 8** Sent Segment Sequence number for Slow Start TCP, TCP Tahoe and TCP Reno

### 2.9.G Swift Start TCP [62]

In a trial to solve the startup problems in slow start, BBN Technologies designed an algorithm called Swift Start [62, 63] (which will be discussed later), which is a combination of packet-pair [64] and packet-pacing [65] technique. Packet pair is used to estimate the path bottle neck bandwidth based on the spacing between two initial acknowledgments (ACKs) from data that have been sent in immediate succession. The algorithm provides an estimation of the current network bandwidth, thus allowing a TCP connection to obtain the network capacity much faster than slow start. Packet pacing is a technique that uses an estimated RTT to spread out packets across an RTT, avoiding bursts of packets that are released to the network, and to establish an initial self-clocking process. An extensive studies have been done to the Original Swift Start (OSS) algorithm, and improve its drawbacks [66].

## III. SIMULATION RESULTS COMPARISON

The simulation results of OSS (Original Swift Start) algorithm shows that it has a lot of drawbacks, which make it inefficient, so we've proposed a modification to that algorithm to overcome these drawbacks, and then we have compared between the performances of the Original Swift Start, Modified Swift Start (MSS) and that of the Slow Start in different network cases studies as shown in Figure 9.
Figure 10 shows the amount of data sent versus the number of RTTs. It is clear that the modified swift start throughput is better than that of slow start also in both cases, because the modified swift start estimates the congestion window faster.

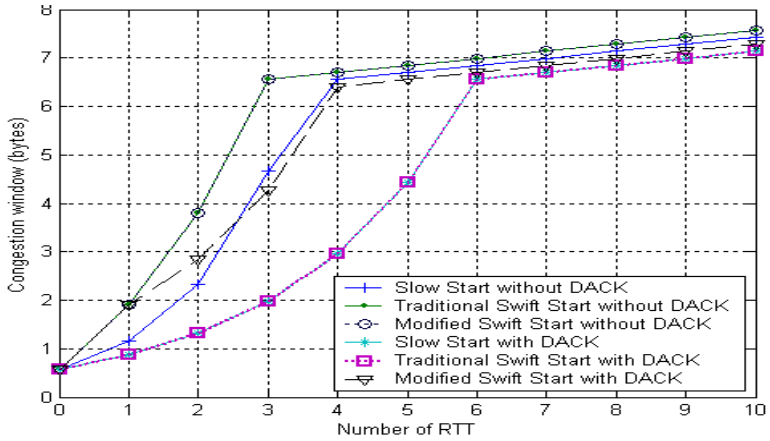


**Figure 9** Congestion Window versus Number of Roundtrip Times for (RTT=0.1 Sec, C=1544000 Bps).

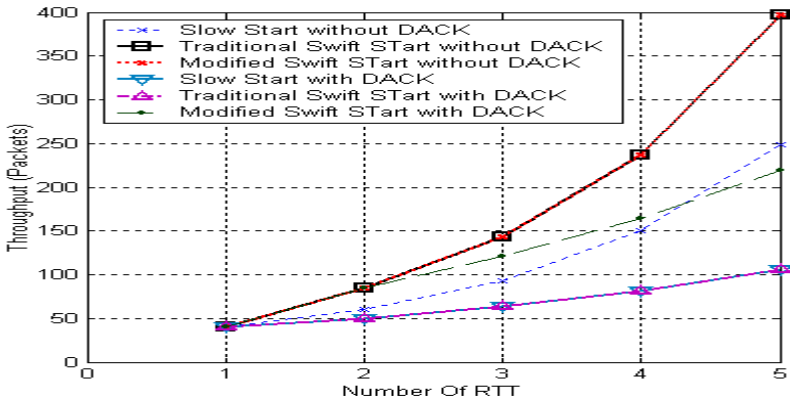


**Figure 10** Throughput versus Number of RTT for (RTT=0.1 Sec, C=1544000 Bps)

Figure 11 shows the amount of data sent versus the number of RTTs. It is clear that the modified swift start throughput is better than that of slow start also in both cases, because the modified swift start estimates the congestion window faster.

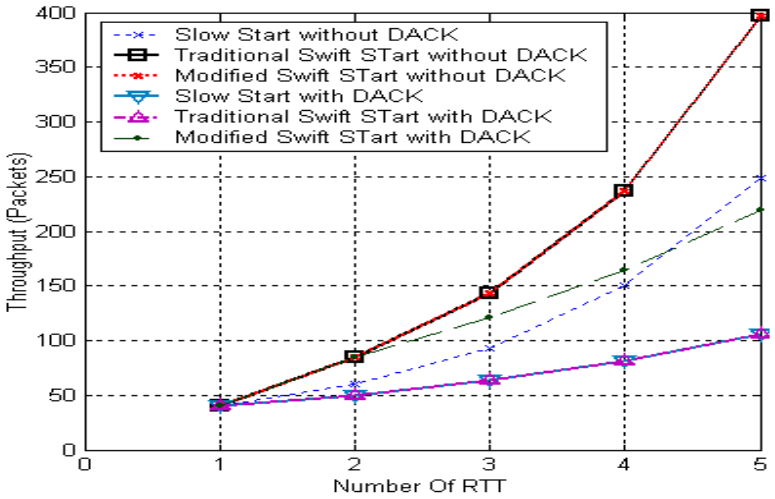


**Figure 11** Throughput versus Number of RTT for (RTT=0.1 Sec, C=1544000 Bps)

Figure 12 shows that as RTT/τ increase, the number of RTTs which are needed to reach ssthresh decreases for the modified swift start in both cases of using DACK and without using DACK, while it is constant for the slow start. For the original swift start the number of RTTs which are needed to reach ssthresh depends on wither DACK is used or not.

The modified swift start is suitable for high delay-bandwidth product networks because as the bandwidth increases τ decreases, and for long networks RTT is very large. However for high delay bandwidth product networks RTT/ τ is very large and TCP needs few RTTs. If RTT/ τ > 45, then TCP will reach ssthresh after the first RTT.

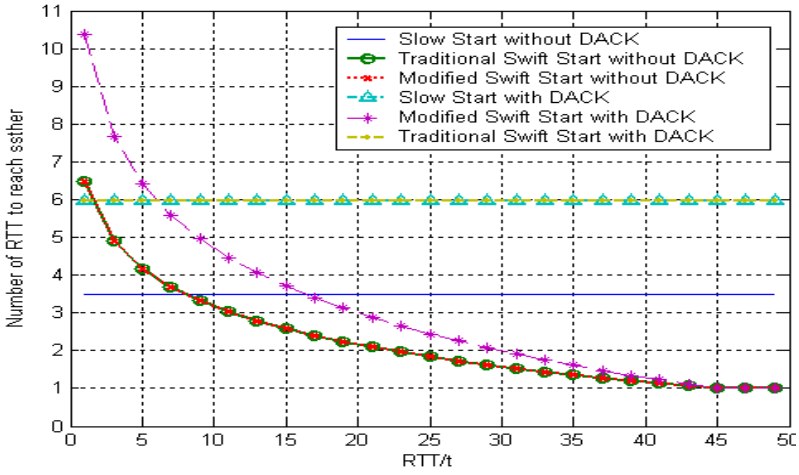


**Figure 12** Number of RTTs which is needed to reach ssthresh Versus RTT/τ.

Table 1 shows both the congestion window and the throughput until the $n^{th}$ RTT for different values of RTT/τ and n=2, 4 and 6 RTT. Note that Table 1 does not consider the effect of the ACK path on the OSS in other words it considers δ = 0.

Table 1 shows that

- For small value of RTT/τ the CWND and B for slow start is better than that of MSS, which means that for low delay bandwidth product the MSS is inefficient.

- As RTT/τ increases the CWND and throughput of the MSS getting better than that of slow start and OSS for certain value of n. which mans that as the delay bandwidth product increases the MSS gets more inefficient.

## IV. CONCLUSION

The simulation results which have been produced indicate that the modified swift start has succeeded in estimating the path capacity and the congestion window quickly. At the same time it has overcome the errors that can affect that estimate. The modified swift uses the modified packet pair algorithm to quickly estimate the congestion window which results in faster connections start up and faster utilization of the bandwidth especially for high delay-bandwidth product network paths, and it uses the packet pacing in the first and the second RTTs to clock the packet transmission over each RTT in stead of sending the window in burst. This pacing avoids overwhelming the intermediate routers.
The modification which introduced in this research was succeed to avoid the error sources in the original swift start algorithm such as using DACK algorithm and the effect of ACK path on the original swift start. The modification avoids these errors and makes the modified swift start applicable in real networks. The drawback of that modification is that it adds 8 bytes of redundant traffic for each connection. This redundant traffic may cause overflow on networking devices buffers. 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.

**Table 3-1** The Congestion Window (CWND) and Throughput (B) at the $n^{th}$ RTT for Different Values of RTT/τ

| DACK | RTT/τ | Slow Start | | | | | | OSS | | | | | | MSS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CWND | | | B | | | CWND | | | B | | | CWND | | | B | | |
| | | n=2 | n=4 | n=6 | n=2 | n=4 | n=6 | n=2 | n=4 | n=6 | n=2 | n=4 | n=6 | n=2 | n=4 | n=6 | n=2 | n=4 | n=6 |
| Used | 5 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 5 | 12 | 26 | 4.5 | 7 | 11.67 |
| | 10 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 10 | 23 | 51 | 7 | 13 | 22.67 |
| | 15 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 15 | 34 | 76 | 9.5 | 19 | 33.67 |
| | 20 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 20 | 45 | 102 | 12 | 24.75 | 44.67 |
| | 25 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 25 | 57 | 127 | 14.5 | 30.75 | 55.67 |
| | 30 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 30 | 68 | 152 | 17 | 36.75 | 66.67 |
| | 35 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 35 | 79 | 178 | 19.5 | 42.75 | 77.67 |
| | 40 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 40 | 90 | 203 | 22 | 48.5 | 88.67 |
| | 45 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 45 | 102 | 228 | 24.5 | 54.5 | 99.67 |
| | 50 | 6 | 14 | 31 | 5 | 8.25 | 14 | 6 | 14 | 31 | 5 | 8.25 | 14 | 50 | 113 | 254 | 27 | 60.5 | 110.7 |
| Not Used | 5 | 8 | 32 | 128 | 6 | 15 | 42 | 5 | 20 | 80 | 4.5 | 9.75 | 26.5 | 5 | 20 | 80 | 4.5 | 9.75 | 26.5 |
| | 10 | 8 | 32 | 128 | 6 | 15 | 42 | 10 | 40 | 160 | 7 | 18.5 | 52.33 | 10 | 40 | 160 | 7 | 18.5 | 52.33 |
| | 15 | 8 | 32 | 128 | 6 | 15 | 42 | 15 | 60 | 240 | 9.5 | 27.25 | 78.17 | 15 | 60 | 240 | 9.5 | 27.25 | 78.17 |
| | 20 | 8 | 32 | 128 | 6 | 15 | 42 | 20 | 80 | 320 | 12 | 36 | 104 | 20 | 80 | 320 | 12 | 36 | 104 |
| | 25 | 8 | 32 | 128 | 6 | 15 | 42 | 25 | 100 | 400 | 14.5 | 44.75 | 129.8 | 25 | 100 | 400 | 14.5 | 44.75 | 129.8 |
| | 30 | 8 | 32 | 128 | 6 | 15 | 42 | 30 | 120 | 480 | 17 | 53.5 | 155.7 | 30 | 120 | 480 | 17 | 53.5 | 155.7 |
| | 35 | 8 | 32 | 128 | 6 | 15 | 42 | 35 | 140 | 560 | 19.5 | 62.25 | 181.5 | 35 | 140 | 560 | 19.5 | 62.25 | 181.5 |
| | 40 | 8 | 32 | 128 | 6 | 15 | 42 | 40 | 160 | 640 | 22 | 71 | 207.3 | 40 | 160 | 640 | 22 | 71 | 207.3 |
| | 45 | 8 | 32 | 128 | 6 | 15 | 42 | 45 | 180 | 720 | 24.5 | 79.75 | 233.2 | 45 | 180 | 720 | 24.5 | 79.75 | 233.2 |
| | 50 | 8 | 32 | 128 | 6 | 15 | 42 | 50 | 200 | 800 | 27 | 88.5 | 259 | 50 | 200 | 800 | 27 | 88.5 | 259 |

## References

[1] V. Cerf, and R. Kahn, A Protocol for Packet Network Intercommunication, IEEE Transaction on Communications, Vol. COMM-22, No.5, pp.637-648, May 1974.
[2] John Postal, Transmission Control Protocol, RFC 793, Internet Request for Comments 793, Sept. 1981.
[3] V. Jacobson, Congestion Avoidance and Control, Proceedings of the ACM SIGCOMM '88 Conference, pp. 314–329, August 1988. (refrence repeated here in [75,1]
[4] D.D. Clark, The Design Philosophy of the DARPA Internet Protocols, Proc. of SIGCOMM'88 Computer Communication Review Vol.18, No.4, pp.106-114, August 1988.

[5] http://web-it.kth.se/~haritak/project/default.html
[6] C. Barakat, E. Altman, W. Dabbous, On TCP Performance in a Heterogeneous Network: A Survey, IEEE Communication Magazine, Jan.2000.
[7] G. Hasegawa, M. Murata, Survey on Fairness Issues in TCP Congestion Control Mechanisms, IEICE Trans. On Communications, Jan. 2001.
[8] S. Floyd, High-Speed TCP for Large Congestion Windows, RFC 3649, Experimental, Dec. 2003.
[9] C. Jin, D.X. Wei, S.H. Low, Fast TCP: Motivation Architecture, Algorithms, performance, IEEE INFOCOM'04, 2004.
[10] D. Katabi, M. Handley, C. Rohrs, Congestion Control for High Bandwidth Delay Product Network, ACM SIGCOMM'02, 2002.
[11] Y.J. Zhn, and L. Jacob, On Making TCP Robust Against Spurious Retransmissions, Computer Communications, Vol.28, Issue 1, pp.25-36, Jan.2005.
[12] Steven H. Low, F. Paganini, and John C. Doyle, Internet Congestion Control," IEEE Control Systems Magazine, Vol.22, No.1, pp.28-39, Feb.2002.
[13] Yanping Teng, Haizhen Wang, Mei Jing, Zuozheng Lian, A Study of Improved Approaches for TCP Congestion Control in Ad Hoc Networks, Procedia Engineering, Volume 29, pp. 1270-1275, 2012.
[14] Dmitri Moltchanov, A Study of TCP Performance in Wireless Environment Using Fixed-Point Approximation, Computer Networks, Volume 56, Issue 4, pp. 1263-1285, March 2012.
[15] Hui-Wen Kan, Ming-Hsin Ho, Adaptive TCP Congestion Control and Routing Schemes Using Cross-Layer Information for Mobile Ad Hoc Networks, Computer Communications, Volume 35, Issue 4, pp. 454-474, February, 2012.
[16] Adnan Majeed, Nael B. Abu-Ghazaleh, Saquib Razak, Khaled A. Harras, Analysis of TCP performance on multi-hop Wireless Networks: A Cross Layer Approach, Ad Hoc Networks, Volume 10, Issue 3, pp.586-603, May 2012.
[17] Jie Feng, Lisong Xu, Stochastic TCP Friendliness: Expanding the Design Space of TCP-Friendly Traffic Control protocols, Computer Networks, Volume 56, Issue 2, pp. 745-761, February, 2012.
[18] Fu XIAO, Li-juan SUN, Ru-chuan WANG, Yue-chao FANG, BIPR: a New TCP New Variant Over Satellite Networks, The Journal of China Universities of Posts and Telecommunications, Volume 18, Supplement 1, Pages 34-39, September, 2012.
[19] Ehab A. Khalil, Simulation and Analysis Studies for A Modified Algorithm to Improve TCP in Long Delay Bandwidth Product Networks, International Journal of Advances in Engineering & Technology, Vol. 1, Issue 4, pp. 73-85, Sept 2011.
[20] Venkataramana Badarla, C. Siva Ram Murthy, Learning-TCP: A Stochastic Approach for Efficient Update in TCP Congestion Window in Ad Hoc Wireless Networks, Journal of Parallel and Distributed Computing, Volume 71, Issue 6, pp. 863-878, June, 2011.
[21] Minsu Shin, Mankyu Park, Byungchul Kim, Jaeyong Lee, Deockgil Oh, Online Loss Differentiation Algorithm with One-Way Delay for TCP Performance Enhancement, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.4, pp. 26-36, April 2011.
[22] Sumedha Chokhandre, Urmila Shrawankar, TCP over Multi-Hop Wireless Mesh Network, International Conference on Computer Communication and Management Proc .of CSIT vol.5 (2011), pp. 461-465, IACSIT Press, Singapore, 2011.
[23] Ghassan A. Abed, Mahamod Ismail and Kasmiran Jumari, A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions, Australian Journal of Basic and Applied Sciences, 5(12). pp. 1345-1352, ISSN 1991-8178, 2011.
[24] Yew, B.S., B.L. Ong and R.B. Ahmad, Performance Evaluation of TCP Vegas versus Different TCP Variants in Homogeneous and Heterogeneous Wired Networks, World Academy of Science, Engineering and Technology, pp: 74, 2011.
[25] Abed, G.A., M. Ismail and K. Jumari, A Comparison and Analysis of Congestion Window for HSTCP, Full-TCP and TCP-Linux in Long Term Evolution System Model. ICOS 2011, Langkawi, Malaysia, pp:364-368, 2011.
[26] Ekiz, N., A.H. Rahman and P.D. Amer, Misbehaviors in TCP SACK generation. ACM SIGCOMM Computer Communication Review, 41(2): pp.16-23, 2011.
[27] R. El Khoury, E. Altman, R. El Azouzi, Analysis of scalable TCP congestion control algorithm, Journal of Computer Communications, Volume 33, November, 2010
[28] Zvi Rosberg, John Matthews, Moshe Zukerman, A Network Rate Management Protocol With TCP Congestion Control and Fairness For All, Computer Networks, Volume 54, Issue 9, pp. 1358-1374, June, 2010.
[29] William H. Lehr, John M. Chapin, On the Convergence of Wired and Wireless Access Network Architectures, Information Economics and Policy, Volume 22, Issue 1, pp. 33-41, March 2010.

[30] Haiyan Luo, Song Ci, Dalei Wu, Hui Tang, End-to-end optimized TCP-friendly Rate Control For Real-Time Video Streaming Over Wireless Multi-Hop Networks, Journal of Visual Communication and Image Representation, Volume 21, Issue 2, pp. 98-106, February, 2010.

[31] Ehab A. Khalil, A Modified Congestion Control Algorithm for Evaluating High BDP Networks, Accepted for publication in IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.11, November 2010.

[32] Jehan.M, G.Radhamani, T.Kalakumari, A survey on congestion control algorithms in wiredand wireless networks, Proceedings of the International conference on mathematical computing and management (ICMCM 2010), Kerala, India, June 2010.

[33] Da-wei DING, Jie ZHU, Xiao-shu LUO, Lin-sheng HUANG, Yan-jun HU, Nonlinear Dynamics in Internet Congestion Control Model With TCP Westwood Under RED, The Journal of China Universities of Posts and Telecommunications, Volume 16, Issue 4, pp. 53-58, August, 2009.

[34] Ben-Jye Chang, Shu-Yu Lin, Jun-Yu Jin, LIAD: Adaptive Bandwidth Prediction Based Logarithmic Increase Adaptive Decrease For TCP Congestion Control in Heterogeneous Wireless Networks, Computer Networks, Volume 53, Issue 14, pp. 2566-2585, September, 2009.

[35] Jongmin Lee, Hojung Cha, Rhan Ha, Improving TCP Fairness and Performance with Bulk Transmission Control Over Lossy Wireless Channel, Computer Networks, Volume 53, Issue 16, pp. 2767-2781, November, 2009.

[36] C. Caini, R. Firrincieli, D. Lacamera, T. de Cola, M. Marchese, C. Marcondes, M.Y. Sanadidi, M. Gerla, Analysis of TCP Live Experiments On a Real GEO Satellite Testbed, Performance Evaluation, Volume 66, Issue 6, Pages 287-300, June, 2009.

[37] Xin Wang, Koushik Kar, Steven H. Low, End-to-End Fair Rate Optimization in Wired-Cum-Wireless Networks, Ad Hoc Networks, Volume 7, Issue 3, pp. 473-485, May, 2009.

[38] Xiaodong Ma, Xiao Su, A New TCP Congestion Control algorithm for Media Streaming, Proceedings of the 2009 IEEE international conference on Multimedia and Expo, New York, NY, USA, pp. 746-749. ISBN ~ ISSN:1945-7871 , 978-1-4244-4290-4 , 2009.

[39] Anup K. Ghosh, Amitava Mukherjee, Debashis Saha, TCP Throughput Enhancement in Wired-Cum-Wireless Network, Computer Communications, Volume 31, Issue 17, pp.4162-4166, November, 2008.

[40] Dzmitry Kliazovich, Fabrizio Granelli, Daniele Miorandi, Logarithmic Window Increase For TCP Westwood+ for Improvement in High Speed, Long Distance Networks, Computer Networks, Volume 52, Issue 12, pp. 2395-2410, August, 2008.

[41] Jiwei Chen, Mario Gerla, Yeng Zhong Lee, M.Y. Sanadidi, TCP With Delayed Ack for Wireless Networks, Ad Hoc Networks, Volume 6, Issue 7, pp.1098-1116, September, 2008.

[42] Sherali Zeadally, Bin Wei, Bjorn Landfeldt, End-to-end support over Heterogeneous Wired-Wireless Networks, Computer Communications, Volume 31, Issue 11, pp.2643-2645, July 2008.

[43] Hala ElAarag, Andrew Moedinger, Chris Hogg, TCP Friendly protocols for media streams over heterogeneous Wired-Wireless Networks, Computer Communications, Volume 31, Issue 10, pp.2242-2256, June 2008.

[44] Satoshi Utsumi, Salahuddin Muhammad Salim Zabir, Norio Shiratori, TCP-Cherry: A new approach for TCP Congestion Control Over satellite IP networks, Computer Communications, Volume 31, Issue 10, pp.2541-2561, June, 2008.

[45] Zhao-wei QU, Juan ZHANG, Optimization of the transmission mechanism for Wireless TCP, *The* Journal of China Universities of Posts and Telecommunications, Volume 15, Issue 1, March 2008.

[46] Qian Wu, Mingwei Gong, Carey Williamson, TCP fairness issues in IEEE 802.11 Wireless LANs, Computer Communications, Volume 31, Issue 10, pp. 2150-2161, June, 2008.

[47] Lachlan Andrew, Cesar Marcondes, Sally Floyd, Lawrence Dunn,Romaric Guillier, Wang Gang, Lars Eggert, Sangtae Ha and Injong Rhee, Towards a Common TCP Evaluation Suite, PFLDnet, Manchester, UK, 2008.

[48] Sangtae Ha and Injong Rhee, Hybrid Slow Start for High-Bandwidth and Long-Distance Networks, PFLDnet, Manchester, UK, 2008.

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

[50] H. Cai, D. Eun, S. Ha, I. Rhee and L. Xu, Stochastic Ordering for Internet Congestion Control and its Applications, Computer Networks: The International Journal of Computer and Telecommunications Networking, 2008.

[51] Sakib A. Mondal, Faisal B. Luqman, Improving TCP Performance over Wired-Wireless networks, Computer Networks, Volume 51, Issue 13, Pages 3799-3811, September, 2007.

[52] Vivek Raghunathan, P.R. Kumar, A counterexample in congestion control of Wireless Networks, Performance Evaluation, Volume 64, Issue 5, Pages 399-418, June, 2007.

[53] Li-Ping Tung and Wei-Kuan Shih, TCP Throughput Enhancement over Wireless Mesh Network, IEEE Communication magazine, 2007
[54] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 Jan. 1997.
[55] 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.
[56] 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.
[57] 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.
[58] 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.
[59] 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.
[60] M. Allman, W. Richard Stevens "TCP Congestion Control" RFC 2581, NASA Glenn Research Center, April 1999.
[61] Go Hasegawa "Congestion and Fairness Control Mechanisms of TCP for the High-Speed Internet" Department of Informatics and Mathematical Science Graduate School of Engineering Science Osaka University, February 2000
[62] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, J. Sterbenz "A Swifter Start For TCP" BBN Technical Report No. 8339, 2002
[63] Frances J. Lawas-Grodek and Diepchi T. Tran "Evaluation of Swift Start TCP in Long-Delay Environment," Glenn Research Center, Cleveland, Ohio October 2004
[64] 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.
[65]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.
[66] A. A. Elbery, E. A. Khalil, and al. et., Swift Start for TCP Congestion Control, performance, Problems, Modification and Evaluation, Mater Thesis, Department of Computer Science & Engineering, Faculty of Electronic Engineering, Menoufiya University, 2007.

# Authors Biography

**Ehab Aziz Khalil,** (B.Sc'78 – M.Sc.'83 – Ph.D.'94), Ph.D. in Computer Network and Multimedia in the Dept. of Computer Science & Engineering, Indian Institute of Technology (IIT) Bombay-400076, India in July 1994, Research Scholar from 1988-1994 with the Dept. of Computer Science & Engineering, Indian Institute of Technology (IIT) Bombay-400076, India, M.Sc in the Systems and Automatic Control, Faculty of Electronic Engineering, Minufiya University, Menouf – 32952, EGYPT, Oct. 1983, B.Sc. in the Dept. of Industrial Electronics, Faculty of Electronic Engineering, Minufiya University, Menouf – 32952, EGYPT, May 1978. Since July 1994 up to now, working as a Lecturer, with the Dept. of Computer Science & Engineering, Faculty of Electronic Engineering, Minufiya University, Menouf – 32952, EGYPT.. Participated with the TPC of the IASTED Conference, Jordan in March 1998, and 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 87 research papers and articles review in the international conferences, Journals and local newsletter.