REVIEW OF TCP CONGESTION CONTROL SLOW-START MODUELS

Mudassar Ahmad, Md Asri Bin Ngadi*
Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia,
UTM Johor Bahru, 81310, Johor, Malaysia.
[e-mail: mudassar.utm@gmail.com, dr.asri@utm.my.]
*Corresponding author: Md Asri Bin Ngadi

ABSTRACT*—TCP provides reliable data transmission by using congestion control mechanisms. Congestion control mechanism consists of slow-start, congestion avoidance, fast re-transmit and fast recovery modules. Slow-start and congestion avoidance modules are used to control the data transmission, whereas fast re-transmit and fast recovery modules are used to re-transmit the lost data. This paper reviews problems history and development of various slow-start modules present in the literature. It also provides a detailed literature review of state-of-art congestion control mechanisms being used by different operating systems by discussing their strength and weaknesses. Finally, literature summary is presented in the form to tables with detailed analysis as a source of inspiration towards future development of new congestion control mechanisms.*

Index Terms*—*TCP, Slow-Start Moduels, Congestion Control Mechanisms.

1. INTRODUCTION

Congestion is a problem that occurs on shared networks when multiple users contend for access to the same network resources (bandwidth, buffers or queues). However, congestion concerns controlling traffic entry into a network, thus avoiding congestive collapse by attempting to avoid over-subscription of any of the processing or link capabilities of the intermediate nodes and networks and taking resource reducing steps, such as reducing the rate of sending packets [1]. Congestion occurs when there is too much traffic in the network routers. If a router cannot transmit packets at a given instance, it stores packets in a queue and waits for the next chance to transmit. Queue has limited size, if queue data exceeds limit, packet will be discarded [1]. If congestion occurs in the network, then packet transfers are delayed and discarded. Due to this reason, some protocols or applications try to retransmit data. Users try to retransmit the data or request the same data again and again. In this case ratio of valid data is decreasing and at the end congestion collapse occurs Therefore, it needs to control this congestion to improve network quality of service [1]. Congestion control is also difficult, because the Internet is designed to be autonomous and it is a very huge network and still it is expanding. There is no centralized management to control each user behavior. Thus, it is difficult to determine how many users or applications are sharing the network exactly. It is also difficult to determine the source of the congestion exactly. It is not possible to determine the exact capacity of the networks and difficult to determine how much networks are congested exactly. Finally, it is not possible to determine why packets are lost. Many efforts have been done in the last twenty years to solve this network problem [1]. However, due to these all reason's congestion control is still a critical issue for many researchers.

Congestion control mechanism consists of slow-start and congestion avoidance, fast retransmit and fast recovery modules. Slow-start and congestion avoidance modules control the transmission, whereas fast retransmit and fast recovery moduels retransmit the lost data. The congestion avoidance module acts as the main part of the congestion control mechanism. Congestion avoidance module consists of response function of the mechanism, which is responsible for the reduction and growth of Congestion Window (*cwnd*) size, which in turn controls the transmission. For convenience, congestion avoidance module along with slow-start module is normally called as a congestion control mechanism. TCP has estimation, window control and data control components to control the transmission. Data control components determines *which* packet to transmit and window control determines *how many* packets to transmit. These decisions are made based on information provided by estimation component. This paper reviews a comprehensive study of TCP congestion control slow-start moduels present in TCP literature. In Section 2, problems history and behavior of TCP congestion control techniques being used during slow-start phase is discussed. In Section 3, literature analysis of slow-start moduels is discussed.

2. TCP SLOW START MODULES

Slow-start module increases the size of Congestion Window (*cwnd*) exponentially whereas congestion avoidance module increases linearly. A number of slow-start modules are proposed in last few years. These slow-start modules sue different techniques to increase the size of *cwnd*, such that network congestion can be reduced. Slow-start modules are briefly explained with equations and *cwnd* graphs in next sub sections.

2.1 Hoe's Approach

Hoe [2] proposes a method for the calculation of Slow Start Threshold (*ssthresh*) at an early stage of the connection. According to Hoe *ssthresh* is equal to the product of delay which is also known as Round Trip Time (RTT) and the estimated bandwidth as denoted in Eq. 1. The bandwidth estimation is performed by using the least squares estimation on three closely spaced Acknowledges (ACKs), similar to the concept of packet pair. RTT is obtained by measuring the round trip time of the first segment. Hoe's approach avoids the source from premature transfer of the connection from slow-start phase to congestion avoidance phase. However, later on Dovrolis *et al.* [3] indicate that this estimation is not sufficient and need some sophisticated filtering to improve the bandwidth estimation. The problem in Hoe's approach is due to the cross traffic, which may hinder in accurate estimation, results in a frequent over estimation of the bottleneck link bandwidth. Hoe's approach fails when multiple losses occur because multiple flows get the same estimation of the link, results in overshooting the value of the *cwnd* from $(N \times C)$ value, where N is the number of competing flows and C is the

capacity of the link, thus a buffer size of half of BDP (BDP / 2) is required to prevent multiple losses for single connection. Figs. 1 and 2 show the *cwnd* behavior of single flows and four flows respectively.

$$ssthresh = Delay \times EstimatedBandwidth \quad (1) \qquad (1)$$

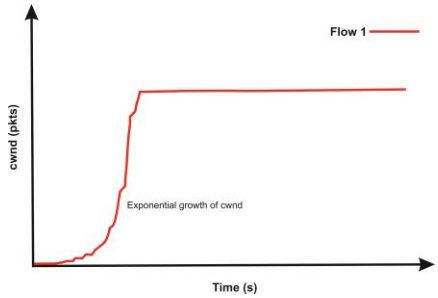

Fig. 1: Hoe's approach, congestion window growth with single flow

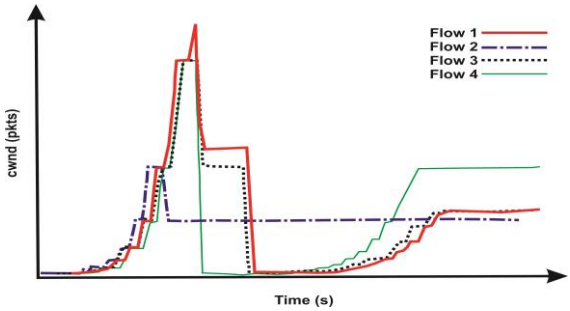

Fig. 2: Hoe's approach, congestion window growth with four flows

2.2 Vegas Approach

Vegas slow-start module allows exponential growth of *cwnd* at alternating round trip times. It calculates the expected throughput, actual throughput and the difference between them. The problem in the Vegas mechanism is the premature termination of the slow-start phase [4]. Ha and Adviser-Rhee [5] experimentally analyze this issue and show that TCP Vegas prematurely terminates the slow-start phase as shown in Fig. 3, while the BDP of the network is high. The reason is that when the actual throughput falls below the expected throughput, it changes from slow-start phase to linear increase or decrease phase as denoted in Eq. 2.

2.3 Standard Slow-Start

The standard slow-start module of Jacobson [6] starts with a *cwnd* size of one segment and for each ACK received, it increases the *cwnd* size by one extra segment. This logic causes the *cwnd* to doubles its size at each RTT for the TCP session, causing an exponential increase of the number of injected segments into the network per RTT. The exponential growth of *cwnd* may cause large segment losses for certain network scenarios as shown in Fig. 4

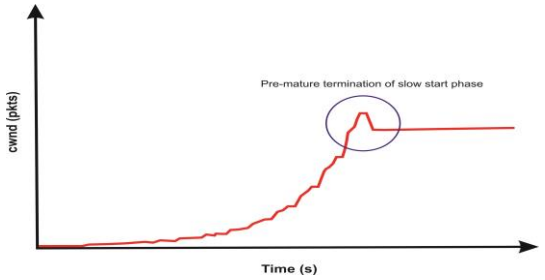

Fig. 3: Vegas approach, premature termination of slow-start phase

$$Mode = \begin{cases} Slow Start & At the Begining \\ & if\ Actual Throughput \\ Linear Increase/Decrease & < Expected Throughput \end{cases} \quad (2)$$

.

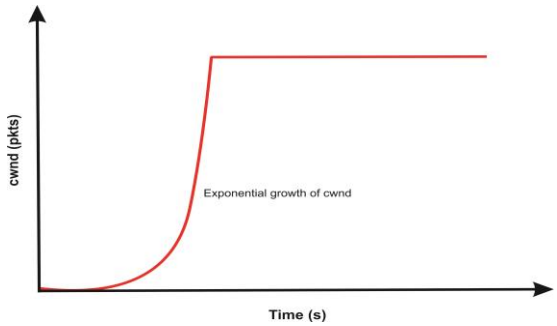

Fig. 4: Typical behavior of Standard slow-start with single flow

$cwnd = 1$

$cwnd \leftarrow cwnd + 1$

2.4 Limited Slow-Start

By limiting the growth of *cwnd* at each RTT, Limited Slow-Start (LSS) can reduce the packet loss rate. Thus, it improves the performance of TCP. LSS introduces a new parameter named as maximum slow-start threshold (*msx_ssthresh*). During slow-start phase, if $(cwnd \leq max\_ssthresh)$, *cwnd* is increased by one Maximum Segment Size (MSS) for every ACK. During LSS phase, if $(max\_ssthresh < cwnd \leq ssthresh)$, *cwnd* is increased by $(\frac{1}{K} \times MSS)$ per RTT and if $(ssthresh < cwnd)$, the connection ends the slow-start phase by entering into the congestion avoidance phase as shown in Eq. 3. Where K is equal to $\left(\frac{cwnd}{\frac{1}{2} \times max\_ssthresh}\right)$. Fig. 5 shows the behavior of LSS during slow-start phase.

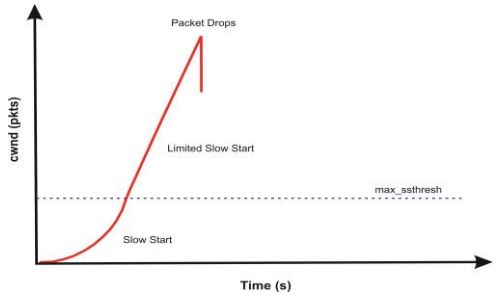

Fig. 5: Typical behavior of Limited Slow-Start

$$\begin{cases} cwnd \leftarrow cwnd + MSS & if\ cwnd \le max\_ssthresh \\ cwnd \leftarrow cwnd + \frac{1}{K} \times MSS & if\ max\_ssthresh < cwnd \le ssthresh \\ Congestion\ Avoidance\ Phase & if\ ssthresh < cwnd \end{cases} \quad (3)$$

2.5 Adaptive Limited Slow-Start

Nakauchi and Kobayashi, [7] proposes Adaptive Limited Slow-Start (ALSS), where the source node configures *ssthresh* and *max-ssthresh* using Simple Internet Resource Notification Scheme (SIRENS). The behavior of SIRENS is illustrated in Fig. 6. By using SIRENS, the source node updates its information about current *AvailableBandwidth* and *QueueSize* at each hop. After this updating, the source node configures the *ssthresh* and *max-ssthresh* parameters using minimum available queue size and bandwidth respectively on the communication link as denoted in Eq. 4, where $C_1 = 0.5$ and $C_2 = 0.75$ are ALSS constants. By this implementation, the source node can shift from slow-start phase to congestion avoidance phase without dropping any packets and can achieve maximum available link bandwidth in an efficient way as shown in Fig. 7.

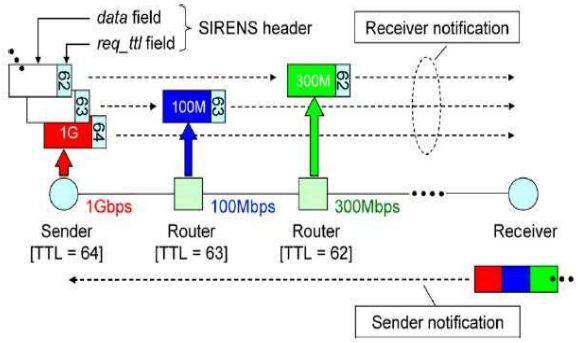

Fig. 6: SIRENS behavior [7]

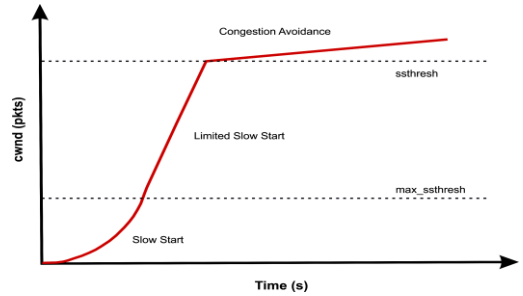

Fig. 7: Slow-start and congestion avoidance phases in ALSS

$$\begin{aligned} max\_ssthresh &= C_1 \times QueueSize \\ ssthresh &= C_2 \times max(cwnd) \\ max(cwnd) &= \frac{Available\ Bandwidth \times \frac{RTT}{2}}{MSS} \end{aligned} \quad (4)$$

2.6 Adaptive Start

Wang et al. [4] proposes an Adaptive Start (Astart) to improve the start-up performance in large bandwidth networks. At the beginning of the connection, Astart adaptively and repeatedly resets the *ssthresh* based on an Eligible Rate Estimation (ERE) [8] estimation mechanism being used by TCP Westwood [9]. Thus, it repeats the exponential and linear growth of the *cwnd* until a packet loss occurs. A start is slower than Standard slow-start, Eq. 5 shows the value of *ssthresh* and *cwnd* at each ACK. Fig.s 8 and 9 show the behavior of Astart with a single flow and with four flows respectively. In Fig. 9, each flow calculates the similar ERE value and after some time all flows overshot the link capacity.

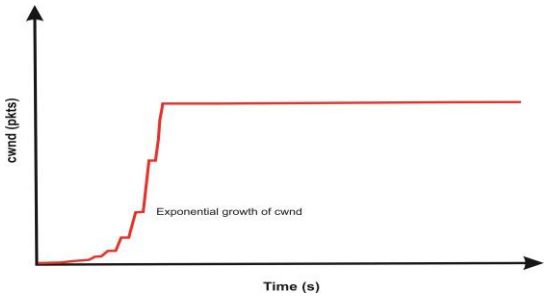

Fig. 8: Typical behavior of Adaptive Start with single flow

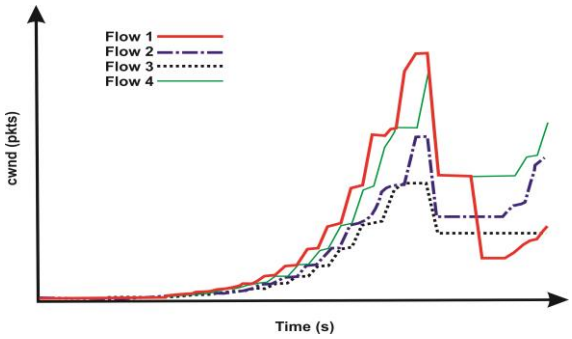

Fig. 9: Typical behavior of Adaptive Start with four flows

$$\begin{cases} ssthresh = \frac{ERE \times RTT_{min}}{seg\_size} & if\ ssthresh < \frac{ERE \times RTT_{min}}{seg\_size} & Astart \\ cwnd \leftarrow cwnd + \frac{1}{cwnd} & if\ cwnd \ge ssthresh & Linear\ Increase \\ cwnd \leftarrow cwnd + 1 & if\ cwnd < ssthresh & Exponential\ Increase \end{cases} \quad (5)$$

2.7 Paced Start

Hu and Steenkiste [10] proposes Paced Start which incorporates a bandwidth estimation mechanism into the Standard TCP start-up algorithm. Paced Start probes the available link bandwidth by measuring the gap between the data packets spacing and the ACK spacing. Standard slow-start ignores this information, while Paces Start uses it to estimate the *cwnd* for the network path. The main idea behind the Packed Start is to apply the available estimation algorithm to the packet sequence used by the Standard slow-start.

Sometimes this gap measurement is very difficult for Long Distance High Bandwidth (LDHB) networks.

2.8 Quick Start

Quick Start [11-14] is an experimental enhancement of the TCP slow-start module which uses explicit router feedback to determine the sending rate quickly. Quick Start requires modifications both in TCP algorithms and in routers. Scharf and Strotbek [14] first, show that Quick-Start can be added in real stack without causing any processing overhead. Secondly, Scharf and Strotbek [14] performed the experiments on webbased applications to measure the performance of Quick-Start. Finally, results show that Quick-Start can significantly speed up the data transmission. Thus, Quick Start may be useful for future Internet applications.

2.9 Cap Start

Cap Start [15] merge Standard slow-start (Reno slow-start) and Limited slow-start mechanisms to develop a new path estimation mechanism, which is a real-time estimation of current TCP path. Cap-Start is an adaptive slow-start module to achieve faster TCP communication in Long Distance High Bandwidth (LDHB) networks. After the estimation, it starts the communication with respect to available link bandwidth. As source interface capacity and network path capacity are two important entities in TCP communication. Cavendish *et al*. [15] evaluate the slow-start impact on different applications running on LDHB networks. Cavendish *et al.* [15] conclude that for LDHB networks, faster network interface cards may not achieve better network performance with Standard slow start (Reno slow-start) protocol. If C_{if} represents TCP source interface capacity, C_i represents router i outgoing interface capacity, C_{bn} represents bottleneck capacity, capacity expansion path scenario satisfies $(C_{if} < C_i, \forall_i)$ and capacity reduction path scenario satisfies $(C_{if} \geq C_i, \forall_i)$, then size of *cwnd* is given as in Eq. 6.

$$cwnd = \begin{cases} \frac{RTT \times C_{if}}{MSS} & Fill \\ \frac{RTT \times C_{bn}}{MSS} & Drain \\ \frac{2RTT \times C_{bn}}{MSS} & MaxQ \end{cases} \quad (6)$$

2.10 Hybrid Start (HyStart)

TCP CUBIC [16] uses HyStart [17] as its default slow-start module. HyStart uses a *Safe* exit point to switch the connection from slow-start phase to congestion avoidance phase. As the capacity of a network can be defined by the sum of unused link bandwidth on the forward path and the size of buffer at bottleneck router, the safe exit point must be less than *C*, whose value is given in Eq. 7, where *B* is the unused link bandwidth, D_{min} represents the minimum forward path one way delay and *S* represents the available buffer size. Packet loss will occur if the size of *cwnd* is greater than *C*. The *cwnd* curve behavior of HyStart during slow-start phase is shown in Fig. 10. Figure illustrates the exponential growth of *cwnd* in slow-start phase.

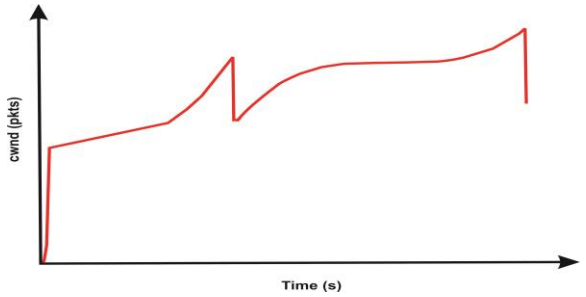

Fig. 10: Congestion window curve of HyStart

$$C = (B \times D_{min} + S) \quad (7)$$

if $(cwnd > C)$, *Packet loss will occur*

There are two types of bandwidth estimation techniques: packet-pair and packet-train. For bandwidth estimation, HyStart uses a concept similar to packet-train. Suppose a source transmits *N* back-to-back packets of size *L* to the destination. For *(N > 2)*, these back-to-back packets are called a packet-train. The length of this packet-train is denoted by $\Delta(N)$, which is equal to $\sum_{k=1}^{k=N-1} \delta_k$ as described in Eq. 8. Where *N* is the number of packets in train, δ_k is the inter interval time between packets k and $k+1$ as shown in Fig. 11. By using the packet-train length, a destination can measure the bandwidth $b(N)$ of the link as expressed in Eq. 9 and 10.

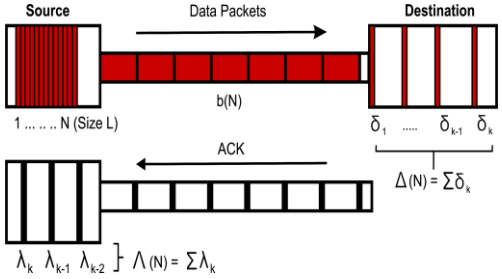

Fig. 11: Packet train concept

$$\Delta(N) = \sum_{k=1}^{k=N-1} \delta_k \quad (8)$$

$$b(N) = \frac{(N-1) \times L}{\Delta(N)} \quad (9)$$

$$b(N) = \frac{(N-1) \times L}{\sum_{k=1}^{k=N-1} \delta_k} \quad (10)$$

By using packet-train concept and an approach of [17], unused link bandwidth on the link can be calculated. Based on this approach, if B represents the unused link bandwidth for the forward path and $minD$ represents the minimum forward one way delay (RTT / 2), then the Bandwidth Delay Product (BDP) of the link path can be denoted as $(B \times minD)$, which is described in Eq. 11.

$$BDP = B \times minD = b(N) \times minD \quad (11)$$

Solving Eq. 9 and 11, $(B \times minD)$ is updated and is shown in Eq. 12.

$$BDP = B \times minD = \frac{(N-1) \times L}{\Delta(N)} \times minD \quad (12)$$

Based on [17], if $\Delta(N)$ is equal to $min\overline{D}$, then $(B \times min\overline{D})$ will equal to $(N-1) \times L$ as described in Eq. 13.

$$BDP = B \times minD = (N-1) \times L \quad (13)$$

Since $b(N) = \frac{(N-1) \times \overline{L}}{\Delta(N)}$, then $(N-1) \times \overline{L}$ represents the size of $cwnd$, means when $\Delta(N)$ is equal to $minD$, the $cwnd$ becomes equal to $(B \times minD)$ as described in Eq. 14.

$$BDP = B \times minD = cwnd \quad (14)$$

By dividing $minD$ on both sides, bandwidth B can be calculated as described in Eq. 15.

$$B = \frac{minD}{cwnd} \quad (15)$$

By using train of acknowledgements, $\Delta(N)$ is estimated, which is equal to the sum of inter arrival times of packets in train as shown in Fig. 11. $\Lambda(N)$ represents the time period between the receipt of first and last ACK in an acknowledgement train. $minD$ is calculated by dividing the minimum observed RTT by 2 as defined in Eq. 16

$$minD = \frac{minRTT}{2} \quad (16)$$

2.11 Early Slow-Start Exit

Giordano et al. (2005) [18] proposes Early Slow-Start Exit (ESSE) to improve the TCP startup performance by setting the *ssthresh* according to a pipe size estimation mechanism based on the observation of few acknowledgement's arrival times. ESSE module is easy to implement and preserves the compatibility with the Standard TCP, since it requires changes to the source side only. ESSE allows to speed-up TCP connections and reduces the packet loss rate. Better performance of TCP can be observed for any of the considered estimators, which indicates that the algorithm is robust against estimation errors. ESSE modified protocols guarantee fair utilization of bandwidth among homogeneous and heterogeneous (Newreno) connections sharing a common link. This technique is quite successful in reducing the packet loss rate at the bottleneck and in decreasing the completion time (indeed, the most interesting parameters for end-users) of short lived connections. An estimated value of *ssthresh* should be close to BDP and can be further expressed as the ratio between the delay and the spacing of packets at bottleneck. The optimal delay is between the round trip delay and the maximum expected delay. The upper and lower bounds of the interval can be estimated at real time by taking the minimum and maximum round trip times observed by the initial few packets. The optimal packet spacing is related to the inter arrival time of acknowledgements of two packets sent in close succession. If RTT_k represents the k^{th} packet RTT, Δt_k represents the inter arrival time between $k-1^{th}$ and k^{th} packet, then Table 1 represents the maximum cases of bandwidth estimators. The pipe-size is estimated first by using one of the estimation from Table 1 over a predefined number of acknowledgements, then

Table 1: Bandwidth estimations in ESSE

Name	Estimator
MinMin	$\frac{minRTT_k}{min\Delta t_k}$
MaxMin	$\frac{maxRTT_k}{min\Delta t_k}$
MinMed	$\frac{minRTT_k}{maxRTT_k - minRTT_k}$
MaxMed	$\frac{maxRTT_k}{maxRTT_k - minRTT_k}$

the $ssthresh$ is selected accordingly.

$$Pipe\ Size = \frac{Delay \times Bandwidth}{Packet - Size} = \frac{Delay}{Packet - Spacing}$$

2.12 Gallop-Vegas

Ho et al. (2006) [19] proposes Gallop-Vegas to enhance the throughput of TCP Vegas [20]. It does not increase $cwnd$ in first RTT, however, after second RTT, it increases the $cwnd$ with a stable growth rate between exponential and linear growth as shown in Fig. 12.

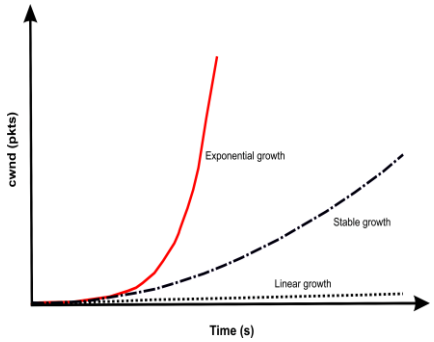

Fig. 12: Stable growth rate of congestion window in Gallop Vegas

3. LITERATURE ANALYSIS AND SUMMARY OF SLOW-START MODULES

This literature reviewed a comparative study of slow-start modules of TCP congestion control mechanism. It is found that, slow-start modules uses two variables: Congestion Window (*cwnd*) and Slow-Start Threshold (*ssthresh*) to control the transmission during show start phase. However, the values of these two variables are depending on the available link bandwidth. Thus, for link bandwidth estimations, different techniques are used by different slow-start modules. The aim of all slow-start modules is to switch the connection from slow-start phase to congestion avoidance phase without dropping much packets. For this purpose some of slow-start modules are also use a conditional variable.

Hoe proposes a method for the calculation of *ssthresh* at an early stage of the connection. Based on Hoe's approach, *ssthresh* is equal to the product of delay RTT and the estimated bandwidth. Hoe's approach fails when multiple losses occur because multiple flows get the same estimation of the link, results in overshooting of the value of the *cwnd*. However, later on Dovrolis *et al.* [3] indicate that this estimation is not sufficient and need some sophisticated filtering to improve it. Hu and Steenkiste (2003) [10] proposes Paced Start which incorporates a bandwidth estimation mechanism into the Standard TCP start-up algorithm. Paced Start probes the available bandwidth by measuring the gap between the data packets spacing and the ACK spacing. Standard slow-start ignores this information, while Paces Start uses it to estimate the *cwnd* for the network path. The main idea behind the Packed Start is to apply the available estimation algorithm to the packet sequence used by the standard slow-start. Sometimes this gap measurement is very difficult for high bandwidth, long distance networks.

Vegas slow-start mechanism allows exponential growth of *cwnd* at alternating round trip times. It calculates the expected throughput and actual throughput and the difference between these throughputs. The problem in the Vegas mechanism is the premature termination of the slow-start phase [4]. Ha and Adviser-Rhee [5] experimentally analyze this issue and show that TCP Vegas prematurely terminates the slow-start phase, while the Bandwidth Delay Product (BDP) of the network is high. Wang *et al.* [4] proposes an Adaptive Start (Astart) to improve the start-up performance in large bandwidth delay networks. At the beginning of the connection, Astart adaptively and repeatedly resets the *ssthresh* based on an Eligible Rate Estimation (ERE) [8] estimation mechanism being used by TCP Westwood. Thus it repeats the exponential and linear growth of the *cwnd* until a packet loss occurs.

By limiting the growth of *cwnd* at each RTT, Limited Slow-Start (LSS) can decrease the packet loss rate by improving the performance of TCP. LSS introduces a new parameter named as maximum slow-start threshold $(max\_ssthresh)$. During slow-start phase, if $(cwnd \le max\_ssthresh)$, $cwnd$ is increased by one Maximum Segment Size (MSS) for every ACK. During LSS phase, if $(max\_ssthresh < cwnd \le ssthresh)$, $cwnd$ is increased by $(\frac{1}{K} \times MSS)$ per RTT and if $(ssthresh < cwnd)$, the connection ends the slow-start phase by entering into the congestion avoidance phase. Giordano *et al*. [18] proposes Early Slow-Start Exit (ESSE) to improve the TCP start up performance by setting the *ssthresh* according to a pipe size estimation mechanism based on the observation of few acknowledgement's arrival times. ESSE allows to speed-up TCP connections and reduces the packet loss rate under several working conditions and load levels. Better performance of TCP can be observed for any of the considered estimators, which indicates that the algorithm is robust against estimation errors. ESSE modified protocols guarantee fair utilization of available link bandwidth among homogeneous and heterogeneous (Newreno) connections sharing a common link. This technique is quite successful in reducing the packet loss rate at the bottleneck and in decreasing the completion time (indeed, the most interesting parameters for end-users) of short lived connections.

An estimated value of *ssthresh* should be close to bandwidth delay product and can be further expressed as the ratio between the delay and the spacing of packets at bottleneck. The optimal delay is between the round trip delay and the maximum expected delay. The upper and lower bounds of the interval can be estimated at real time by taking the minimum and maximum round trip times observed by the initial few packets. The optimal packet spacing is related to the inter arrival time of acknowledgements of two packets sent in close succession. If (RTT_k) represents the k^{th} packet RTT, (Δt_k) represents the inter arrival time between $k-1^{th}$ and k^{th} packet, then Table 1 represents the maximum cases of bandwidth estimators. The pipe-size is estimated first then the *ssthresh* is selected accordingly.

Ho *et al* [19] proposes Gallop-Vegas to enhance the throughput of TCP Vegas. It does not increase *cwnd* in first RTT, after second RTT, it increases the *cwnd* with a stable growth rate between exponential and linear growth. [7] proposes Adaptive Limited Slow-Start (ALSS), where the source node configures *ssthresh* and *max-ssthresh* using SIRENS. By using Simple Internet Resource Notification Scheme (SIRENS), the source node updates its information about current *AvailableBandwidth* and *QueueSize* at each hop. After this updating, the source node configures the *ssthresh* and *max-ssthresh* parameters using minimum available queue size and bandwidth respectively on the communication link. By this implementation, the source node can shift from slow-start phase to congestion avoidance phase without dropping any packets and can achieve maximum available link bandwidth.

Quick Start is an experimental enhancement of the TCP slow-start mechanism which uses explicit router feedback to determine the sending rate quickly. Quick Start requires modifications both in TCP algorithms and in routers. Scharf and Strotbek [14] first, show that Quick-Start can be added in real stack without causing any processing overhead. Secondly, Scharf and Strotbek [14] performed the experiments on web based applications to measure the performance of Quick-Start. Finally, results show that Quick-Start can significantly speed up the data transmission. Thus Quick Start can be useful for future Internet applications.

Cap Start [15] merge Standared TCP slow-start (Reno slow-start) and Limited slow-start mechanisms to develop a new path estimation mechanism, which is a real-time estimation of current TCP path. Cap-Start is an adaptive slow-start mechanism to achieve faster TCP communication in high bandwidth long distance networks. After the estimation, it starts the communication with respect to available link bandwidth. As source interface capacity and network path capacity are two important entities in TCP communication, Cavendish *et al.* [15] evaluate the slow-start impact on different applications running on Long Distance High Bandwidth (LDHB) networks. Cavendish *et al*. [15] conclude that for LDHB networks, faster network interface cards may not achieve better network performance with Standard

slow-start (Reno slow-start) protocol. HyStart [17] uses a safe exit point for the termination of slow-start phase. As the capacity of a network can be defined by the sum of unused available link bandwidth on the forward path and the size of buffer at bottleneck router, the *Safe* exit point must be less than a run time defined variable's value. Different techniques being used by slow-start modules are summarized in Table 2.

Table 2: Literature analysis of slow-start modules

Slow-start modules	Techniques
Hoe's Approach	It uses a technique similar to Packet Pair for the estimation of the unused link bandwidth.
Vegas Approach	It uses actual throughput, expected throughput and their difference to make a decision to terminate the slow-start phase.
Standard Slow-Start	It uses a technique of exponential growth of congestion window (*cwnd*) during slow-start phase.
Limited Slow-Start	It uses maximum slow-start threshold (*max_ssthresh*) for the growth of congestion window (*cwnd*).
Adaptive-Limited Slow-Start	It uses Simple Internet Resource Notification Scheme (SIRENS) to estimated the unused link bandwidth and queue size.
Adaptive Start	It uses Eligible Rate Estimation (ERE) of TCP Westwood to estimate the unused link bandwidth.
Paced Start	It uses packet spacing and ACK spacing techniques to estimated the unused available bandwidth.
Quick Start	It uses the Explicit Router Feedback (ERF) technique to determine the sending rate.
Cap Start	It uses both techniques of Standard Slow-Start and Limited Slow-Start to develop a new path estimation mechanism.
Hybrid Start	It uses a conditional variable to switch the connection from slow-start phase to congestion avoidance phase.
Early Slow-Start Exit	It uses packet spacing based on ACK arrival time to estimate the unused link bandwidth.
Gallop Vegas	It used linear, stable and exponential growth of congestion window (*cwnd*).

4. FINDINGS OF LITERATURE REVIEW

At the beginning of transmission, slow-start module increases the size of *cwnd* rapidly because the size of unused link bandwidth is unknown. Slow-start modules increase *cwnd* exponentially, linearly and in stable way. During exponential growth, size of *cwnd* increases one packet in each acknowledgment (ACK) which doubles the size of *cwnd* at the end of each Round Trip Time (RTT). In linear growth, size of *cwnd* increases linearly, which increases one packet in each RTT instead of at each ACK. In stable growth, size of *cwnd* increases between exponential and linear way. However, exponential growth of *cwnd* increases the size of *cwnd* very fast as compared to linear or stable growth. Thus, exponential growth of *cwnd* causes much packet losses during slow-start phase and many state-of-the-art operating systems suffer this problem. Once the size of *cwnd* increase to an appropriate value, connections leaves the slow-start phase and enters into the congestion avoidance phase. Slow-start modules use different techniques to switch the connection from slow-start phase to congestion avoidance phase. Some slow-start modules use *ssthresh* variable and some modules use a condition, such as *safe* exit point, *early* exit point or *cut-off* point to switch the connection from slow-start phase to congestion avoidance phase without losing much packets in slow-start phase. If the size of *cwnd* is equal to or greater than *ssthresh*, connection switches automatically from slow-start phase to congestion avoidance phase. Different congestion control modules use different techniques to calculate the *ssthresh* and *cut-off* or *safe* exit points. Few slow-start modules also calculate the expected and actual throughput and the difference between these two throughputs to control the growth of *cwnd*. In short, the purpose of slow-start module is to start the transmission and to increase the size of *cwnd* to an appropriate size, so that, maximum available link bandwidth can be achieved for transmission by minimizing the packet loss rate during slow-start phase.

5. SUMMARY

In this paper, an extensive literature review is presented that mainly includes slow-start. However, research is being done to utilize the maximum available link bandwidth and to avoid from the congestion. Practically all Internet applications rely on the TCP and to deliver data reliably across the network. The most important element of TCP is congestion control that defines its performance characteristics. Literature highlighted the fact that research focus has changed with the development of the Internet, from the basic problem of eliminating the congestion collapse phenomenon to problem of using available network resources efficiently in different type of network environments.

REFERENCES

[1] Lar, S.-u. and Liao, X. (2013). An initiative for a classified bibliography on TCP/IP congestion control. Journal of Network and Computer Applications. 36(1), 126-133.

[2] Hoe, J. (1996). Improving the start-up behavior of a congestion control scheme for TCP. In ACM SIGCOMM Computer Communication Review, vol. 26. ACM, 270-280.

[3] Dovrolis, C., Ramanathan, P. and Moore, D. (2004). Packet-dispersion techniques and a capacity-estimation methodology. Networking, IEEE/ACM Transactions on. 12(6), 963-977.

[4] Wang, R., Pau, G., Yamada, K., Sanadidi, M. and Gerla, M. (2004). TCP startup performance in large bandwidth networks. In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 2. IEEE, 796-805.

[5] Ha, S. and Adviser-Rhee, I. (2009). Improving tcp congestion control for high bandwidth and long distance networks. North Carolina State University.

[6] Jacobson, V. (1988). Congestion avoidance and control. In ACMSIGCOMMComputer Communication Review, vol. 18. ACM, 314-329.

[7] Nakauchi, K. and Kobayashi, K. (2007). An explicit router feedback framework for high bandwidth-delay product networks. Computer Networks. 51(7), 1833-1846.

[8] Wang, R., Valla,M., Sanadidi,M. and Gerla,M. (2002). Using adaptive rate estimation to provide enhanced and robust transport over heterogeneous networks. In Network 61 Protocols, 2002. Proceedings. 10th IEEE International Conference on. IEEE, 206- 215.

[9] Mascolo, S., Casetti, C., Gerla,M., Sanadidi,M. andWang, R. (2001). TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In Proceedings of the 7th annual international conference on Mobile computing and networking. 60 ACM, 287-297.

[10] Hu, N. and Steenkiste, P. (2003). Improving TCP startup performance using active measurements: algorithm and evaluation. In Network Protocols, 2003. Proceedings. 11th IEEE International Conference on. IEEE, 107-118.

[11] Floyd, S. and Fall, K. (1999). Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Transactions on Networking (TON). 7(4), 458-472.

[12] Floyd, S., Allman, M., Jain, A. and Sarolahti, P. (2007). Quick-Start for TCP and IP. IETF RFC4782.

[13] Scharf,M., Hauger, S. and KÂ¨ogel, J. (2008). Quick-Start TCP: From theory to practice. In Proc. 6th Intern. Workshop on Protocols for FAST Long-Distance Networks (PFLDnet), Manchester.

[14] Scharf, M. and Strotbek, H. (2008). Performance evaluation of Quick-Start TCP with a Linux kernel implementation. In Proceedings of the 7th international IFIPTC6 networking conference on AdHoc and sensor networks, wireless networks, next generation internet. Springer-Verlag, 703-714.

[15] Cavendish, D., Kumazoe, K., Tsuru, M., Oie, Y. and Gerla, M. (2009). CapStart: An adaptive tcp slow-start for high speed networks. In Evolving Internet, 2009. INTERNETâ€™09. First International Conference on. IEEE, 15-20.

[16] Ha, S., Rhee, I. and Xu, L. (2008). CUBIC: A new TCP-friendly high-speed TCP variant. ACM SIGOPS Operating Systems Review. 42(5), 64-74.

[17] Ha, S. and Rhee, I. (2011). Taming the elephants: New TCP slow-start. Computer Networks. 55(9), 2092-2110