Communications and Network, **2011, 3, 85-98**
doi:10.4236/cn.2011.32011 Published Online May 2011 (http://www.scirp.org/journal/cn)

TCP Window Based Congestion Control Slow-Start Approach

Kolawole I. Oyeyinka[1], Ayodeji O. Oluwatope[2], Adio. T. Akinwale[3], Olusegun Folorunso[3], Ganiyu A. Aderounmu[2], Olatunde O. Abiona[4]
[1]*Department of Computer Science*, *Yaba College of Technology*, *Lagos*, *Nigeria*
[2]*Comnet Lab*., *Department of Computer Science and Engineering*, *Obafemi Awolowo University*, *Ile-Ife*, *Nigeria*
[3]*Department of Computer Science*, *UNAAB*, *Abeokuta*, *Nigeria*
[4]*Department of Computer Information Systems*, *Indiana University Northwest*, *Garry*, *USA*
E-mail: *ikoyeyinka@yahoo.com*, {*aoluwato*, *gaderoun*}*@oauife.edu.ng*, *oabiona@iun.edu*
Received February 23, 2011; *revised March* 1, 2011; *accepted April* 8, 2011

Abstract

Transmission control protocol (TCP) has undergone several transformations. Several proposals have been put forward to change the mechanisms of TCP congestion control to improve its performance. A line of research tends to reduce speed in the face of congestion thereby penalizing itself. In this group are the window based congestion control algorithms that use the size of congestion window to determine transmission speed. The two main algorithm of window based congestion control are the congestion avoidance and the slow start. The aim of this study is to survey the various modifications of window based congestion control. Much work has been done on congestion avoidance hence specific attention is placed on the slow start in order to motivate a new direction of research in network utility maximization. Mathematical modeling of the internet is discussed and proposals to improve TCP startup were reviewed. There are three lines of research on the improvement of slow start. A group uses the estimation of certain parameters to determine initial speed. The second group uses bandwidth estimation while the last group uses explicit request for network assistance to determine initial startup speed. The problems of each proposal are analyzed and a multiple startup for TCP is proposed. Multiple startups for TCP specify that startup speed is selectable from an n-arry set of algorithms. We then introduced the e-speed start which uses the prevailing network condition to determine a suitable starting speed.

Keywords: Data Communication, Network Protocols, TCP, Congestion Control, Slow-Start

1. Introduction

Originally, the Internet was designed to support best-effort applications meaning that then it could only deliver data not necessarily guaranteeing the delivery. However, before 1988, TCP was used to compliment the Internet by ensuring that data delivery was reliable. This same TCP version did not include congestion avoidance, fast-recovery and fast-retransmit mechanisms. The direct impact on user applications is low network utility derivation as a result of heavy network congestion. This resulted in congestion collapse throughout the mid 80s. This continues until [1] introduced the concept of control through the adaptation of the source rate using packet loss. Jacobson algorithm has been modified variously by several researchers among these were [2-7].

Worthy of note is the domination of the Internet by TCP flows carrying data from applications such as FTP, HTTP, SMTP etc in the early days of TCP. However, the nature of Internet traffic has changed dramatically such that it includes traffic from other data transmission protocols, which are TCP unfriendly. Protocols and applications that are not malleable to the dynamics of the Internet are reffered to as TCP-unfriendly. But, TCP in an attempt to respond to the dynamics [8,9] of the Internet, it penalizes itself by reducing transmission speed specifically, in the face of network congestion. However, these TCP-unfriendly protocols and applications are aggressive at bandwidth consumption and do not respond to network congestion indications. TCP-unfriendly applications include video streaming, voice-over-IP, and videoconference.

In simple terms, congestion control is the adaptation of an application's rate of packets injection into the Internet in response to changing network conditions such as packets loss and/or end-to-end delay. There are two types of congestion control techniques—window- and rate-based. In window-based approach, data transmission rate is adjusted based on setting a congestion window size using additive increase and multiplicative decrease (AIMD) algorithm. While in rate-based approach, a set of equations is used to control data transimission speed. TCP uses the window-based approach as a congestion control technique.

The fate of Jacobson's AIMD algorithm and its subsequent modifications in the face of cross traffics and heterogeneous flows is a motivation for this work. From literature, substantial research efforts had been concentrated no the understanding, modification and implementation of window-based congestion control with particular focus on the congestion avoidance stage. But, recent study has shown that attention should shift towards better understanding and modification of window-based congestion control with focus on the slow-start stage and acknowledgement congestion control. Therefore, the focus of this paper is to review existing proposals on TCP congestion avoidance and slow- start mechasims with view to motivating a new direction in the network utility maximisation.

The rest of this paper is organized as follws: Section 2 discusses various variants of TCP implementations, Section 3 looks at some rate based congestion control proposals. Section 4 discusses mathematical modeling of the internet congestion control, Section 5 looks at the various modifications of the slow start state of TCP, Section 6 analyses the various problems associated with each TCP variants, Section 7 suggests future research directions while section 8 concludes the paper.

2. Variants of TCP

Variants of TCP, which are of interest, include those implemented already, yet to be implemented and employing convectional slow-start algorithm.

2.1. TCP Tahoe

The TCP Tahoe was released in 1988 by V. Jacobson in [1] being the first implementation of TCP to employ congestion control mechanism. Tahoe contains the AIMD (additive increase, multiplication decrease) being its control mechanism. Tahoe achieved congestion control through adjusting its windows size additively to increase and multiplicatively to descrease. AIMD at the initial stage increases windows size exponentially but, after a certain threshold, it switches to linear window size increase *i.e.* by one packet per RTT before conges- tion occurs (Additive increase). At this point, Tahoe switches to the congestion avoidance state. If the ACK for a packet is not received before a time out, the thresh- old set is reduced by half and the congestion window is reduced to one packet (Multiplicative decrease). In summary, TCP Tahoe controls congestion as follows:

- When congestion window is below the threshold, the congestion window grows exponentially (slow start state)
- When the congestion window is above the threshold the congestion window grows linearly (additive increase) *i.e.* congestion avoidance
- Whenever there is a timeout, the threshold is set to one half of the current congestion window and the congestion window is set to one while the packet is retransmitted (multiplicative decrease)
- Algorithms implemented are slow start and congestion avoidance.

2.2. TCP Reno

Proposal to modify Tahoe was given in [11]. Like its predecessor, Reno sets its congestion window to one packet upon a time out (RTO). However, Reno extended its algorithm to include the fast retransmit mechanism. The fast retransmit involves the re-transmission of a dropped packet if three duplicate ACKs for a packet are received before the RTO. Reno also introduces the fast recovery mechanisms which prevent transmission to re-enter the slow start state after a fast retransmit. Instead the window size is halved and the threshold is adjusted accordingly and TCP remain in congestion avoidance until a timeout occurs. This is discussed in detail in [3,5]. TCP Reno became the standard TCP algorithm implemented in most computers. Algorithms implemented by Reno are slow start, congestion avoidance, fast retransmit and fast recovery.

2.3. TCP New Reno

The new-Reno TCP includes a change to the Reno algorithm at the sender end with a view to eliminate Reno's wait for a retransmit time-out whenever multiple packets are lost from a window [7,12]. This change modifies the sender's behaviour during fast recovery. When this happens, New Reno does not exit from the fast recovery state as in the case of Reno, but waits for the receipt of all the outstanding ACKS for that window.

The followings are the summary of New-Reno fast recovery actions;

- It notes the maximum packet s outstanding while en-

tering fast recovery
- When a new ACK is received and it acknowledges all the outstanding packets, then fast recovery is exited and cwnd is set to half the value of ssthresh, then it transits to the congestion avoidance state. But, if a partial ACK is received, then, it assumes the next packet in the link is lost and tries to retransmit
- It exits fast recovery when all data in the window is acknowledged [6].

2.4. SACK (TCP with Selective Acknowledgement)

One challenge with the New Reno algorithm is its inability to detect other lost packets until the ACK for the first retransmitted packet was received. This implies that New Reno suffers from the fact that the detection of each packet loss takes one RTT. Hence selective acknowledgment (SACK) was proposed by [13]. SACK is an extension of TCP Reno and TCP New Reno. It intends to solve two problems of TCP Reno and New Reno *i.e.* detection of multiple packet loss and Retransmission of more than one lost packet per RTT.

SACK retains the slow start and fast retransmits of Reno. It also has the coarse grained time out of Tahoe. SACK algorithms specify that instead of cumulative acknowledgement of packets as contained in TCP Tahoe, Reno and New-Reno, Packets should be acknowledged selectively. This requires each ACK to contain an entry for which packet that is being acknowledged. This enables the sender to figure out which packets have been acknowledged and which ones are still outstanding.

SACK specifies that whenever a sender enters into fast recovery state, a variable "pipe" be initiated and used to estimate the number of packets that are missing along the path. It then sets the size of cwnd to half its current size as usual.

Each time an ACK is received, the size of the pipe is decreased by one and when a packet is transmitted or retransmitted, the pipe is increased by one. Whenever the size of the pipe becomes smaller then cwnd, it checks which packets are yet to be acknowledged and retransmits immediately. If there are no outstanding ACK, it sends a new packet. Thus the sender only sends new or retransmitted packet if the pipe is less the cwnd. This way SACK can send more than one lost packet in a single RTT. Use of pipe variable separates the decision of when to send a packet from which packet to send.

Other features of SACK are as follows:
- The score board: The sender maintains a data structure call scoreboard. This was proposed by [13]. The scoreboard remembers all the ACKS that has been received for the data sent. Hence the sender is able to deduce which packet has not been acknowledged and resend them when it is able.
- When a retransmitted packet is dropped, SACKs detects this through the retransmit timeout. This packet will be retransmitted and then SACKs transits to the slow-start state.
- Recovery ACK: The sender exits fast recovery when a recovery acknowledgement is received acknowledging that all outstanding data when fast fecovery was entered have been received.
- Partial ACKS: SACK handles partial ACKs in a special way. Partial ACKs are those received during Fast Recovery but do not take the sender out of Fast recovery. When a partial ACK is received, the pipe is decreased by two packets rather than one, at first, when fast retransmit is initiated, the pipe is decreased by one for the lost packet and increased by one for the retransmitted packet in subsequent partial ACKs received, the pipe was incremented when the packet was transmitted initially but the pipe was never decreased when that packet is assumed lost and retransmitted. Hence when the succeeding partial ACK arrives, it represents two packets (the original packet and the retransmitted packets) Hence the pipe is decremented by two rather than one.
- The max burst parameter: This limits the number of packets that can be sent in response to a single incoming ACK packet. This is still at the experimental stage.

Other TCP congestion control algorithms that use selective acknowledgement include that of [4,14] etc. One major drawback of the SACK algorithm is the relative difficulty in implementation of selective acknowledgement.

2.5. TCP Vegas

The TCP congestion control schemes that have been described so far use packet loss based approach to measure congestion. There is a class of congestion control algorithms that adapt its congestion window size based on end-to-end delay. This approach originated from [15] and is presented by [16,17] as TCP Vegas.

The followings are the differences between TCP Vegas and TCP Reno:
- In the slow start-state, congestion control was incorporated by a deliberate delay in congestion window growth.
- When packet-loss occurs, TCP Vegas treats the receipt of certain ACKs, as a trigger to check if a timeout should occur [16].
- It updates its congestion window based on end-to-end delay instead of using packet-loss as the window up-

date parameter.

Vegas extended Reno re-transmission strategy. It keeps track of when each packet was sent and calculates an estimate of RTT for each transmission. This is done by monitoring how long it took each ACK to get back to the sender. Whenever a duplicate ACK is received, it performs the following check:

if (current RTT > RTT estimate)

If this is true, it retransmits the packet without waiting for 3 duplicate ACK or a time out as in Reno [17]. Hence, Vegas solves the problem of not detecting lost packets when the window is very small *i.e.* less than three and could not receive enough duplicate ACKs.

TCP Vegas congestion avoidance behavior is different from other TCP implementations. It determines congestion states using the sending rate. If there is a decrease in calculated rate of transmission as a result of large queue in the link, it reduces its window. When the sending rate increases, the window size also increases.

2.6. Delay Based Congestion Control Algorithms

These types of algorithms use queuing delay to signal the need for window adjustment. The issue of fairness comes with these algorithms. Delay based congestion control is attractive because it can solve the problem of fairness using queuing delay. To implement delay based congestion control, it is necessary to measure propagation delay. Propagation delay is the time it takes a packet to travel from the sender to its destination. The propagation delay is usually set to the smallest observed RTT. There are several observed problems with the estimation of the queuing delayed. Estimating queuing delay is challenging if the RTT contains more elements then affixed propagation delay, e.g. retransmission delay in wireless link, a high loaded Internet link etc. There are very few researches done on delay-based congestion control in wireless mobile network with the exception of [18] which proposed delay based congestion control scheme for commercial CDMA (Code Division Multiple Access).

Other lines of researches are in the area of high-speed, large delay networks. Prominent among these are the High Speed TCP (HSTCP) proposed by [19] and scalable TCP (STCP) proposed by [20] which are experimental protocols that attempt to improve TCP performance under large bandwidth-delay product. They make TCP increments rule become more aggressive. Loss-delay based Strategy was used by TCP Africa, Compound TCP [21] and TCP Illinois [22]. These protocols try to increase window size more aggressively than TCP New Reno as long as the network is not fully utilized and it switches to AIMD behavior of Reno when the network is near congestion.

2.7. Summary of Proposed TCP Congestion Control Implementation

Henceforth, TCP Tahoe, Reno and New Reno are referred to as New- Reno. This is the transport protocol of choice and it is implemented in over 90% of Internets traffic today. It became officially recognized in 2004 [6]. But, currently, Compound TCP which is the version implemented in Ms-Window 7 is expected to grow in usagen across the Internet. **Table 1** shows the comparison between TCP New-Reno and other proposed TCP algorithms.

In **Table 1**, the various proposed TCP variants, are categorized based on their control mechanism or type of

Table 1. Variants of TCP congestion control implementation (using TCP new reno as basis).

Protocol	Type	Purpose
TCP New-Reno [6]	Loss based	The standard TCP protocol
STCP [20]	Loss based	Higher throughput with high capacity and large delay
HSTCP [19]	Loss based	Higher throughput with high capacity and large delay
BIC – TCP [23]	Loss based	Higher throughput with high capacity and large delay
CUBIC [24]	Loss based	Higher throughput with high capacity and large delay
TCP Vegas [17]	Delay based	Higher through puts and reduced loss rate
Fast TCP [25]	Delay based	Higher throughput with high capacity and large delay
TCP Africa [26]	Lossdelay based	Higher throughput with high capacity and large delay
Compound TCP [21]	Lossdelay based	Higher throughput with high capacity and large delay
TCP Illinois [22]	Lossdelay based	Higher throughput with high capacity and large delay
West wood + [27]	Bandwidth estimation	Higher throughput over wireless networks. High capacity & large delay networks.
XCP [28]	Extra signaling	Higher throughput over wireless networks. Also for high capacity and large delays. Smaller queues. Separate fairness control

feedback. It also considers the performance challenge of New-Reno which it attempts to address. The TCP like algorithms highlighted uses control mechanism like loss-based, delay- based, loss-delayed based, bandwidth estimating, and extra signalling. TCP New-Reno was upgraded from experimental status to full protocol status in 2004. Several proposals and researches had been put forward to improve its performance. Many of these proposals tend to improve TCP in a high speed network where it has been showed that TCP mechanism may lead to network resources underutilization. [29]. TCP Westwood proposed bandwidth estimation as congestion measure [27]. It specified that a TCP sender continuously computes the connection bandwidth estimate by properly averaging the returning ACKs and the rate at which the ACKs are received. After a loss has occurred, the sender uses the estimated bandwidth to properly set the sending rate and the congestion window. This is an improvement on standard TCP which half its window on loss detection [30]. However, TCP Westwood hase not proved any better in term of stability and fairness when it co-habit with the standard TCP and its suitability for general deployment has not been ascertained. Other proposed protocols in this category include XCP [31] which requires modification to router algorithm. However, it is not visible to modify all existing routers' algorithms hence XCP will remain experimental protocol for a long time. HSTCP [19] was also designed for high bandwidth delay product. It uses loss-delay to detect congestion. Wei *et al.* [25] proposed the FAST TCP which uses delay instead of loss to signal congestion. Other protocols in this category include BIC TCP [23], STCP [20], CUBIC TCP [24], TCP Illinois [22] etc. These protocols deal with modifying the window growth function to TCP to match large bandwidth-delay product. This appears easy but the issue of fairness that comes with these protocols is enormous and remains a challenge. These fairness issues include both intra and inter-protocol fairness. In addition, none of these protocols targets the startup behaviour of AIMD.

3. Rate-Based Congestion Control Scheme

According to [32], a great percentage of the current researches on congestion control concentrate on the use of network utility maximization framework [33] as guidance for design and analysis [28].

The optimization-based framework introduced by [33] formed the derived operating point for congestion control algorithms. The framework, according to [34] associates a utility function with each flow and maximizes the aggregate system utility function subject to link capacity constraints. This is referred to as Kelly's system problem and it is an optimization problem.

Under the rate based congestion control, congestion control schemes can be viewed as algorithms that compute the optimum or sub-optimum solutions to the Kelly's system optimization problem. Congestion control schemes can be categorized into three: primal, dual and primaldual.

3.1. Primal Algorithms

Here the endpoints adapt the source rates dynamically based on the route prices and the links select a static law to determine the links prices directly from the arrival rate [35].

3.2. Dual Algorithms

This is a direct opposite of the primal algorithms, the links adapt the links prices dynamically based on the link rates and the end points select a static law to determine the source rates directly from the route prices and other source parameters [28,35,36].

3.3. Primal-Dual Algorithm

The algorithms in primal family measure congestion using the links aggregate rate. This involved the averaging of feedback from the network by end points (sources). On the other hand, the algorithms in the dual family calculate the source rate from the route congestion measures which corresponds to averaging the source rate before the sources get a feedback of explicit congestion information. The primal-dual algorithms viewed congestion control as decomposable into two parts: Congestion avoidance at the source and active queue management at the links. Primal-dual algorithms relate rate change with route congestion measure at the source and relate packet marking probability change with link aggregate rate at the router [34,37,38]. An example is the work of Liu [34], where a new class of algorithms is introduced, which is of primal-dual type. That is, they feature dynamic adaptations at both the source and the link ends.

Stability of primal mechanism under communication is analyzed by [39,40] and reported in [32]. Paganini *et al* [41] proposed a dual algorithm and showed that it is stable in arbitrary topologies and delays. Alpcan and Basar's [37] algorithm for primal-dual was shown to be stable in the absence of delay. It was also proved to be stable for networks with a single bottleneck link and several users when each user may have different RTT.

4. Mathematical Modelling of the Internet Congestion Control

Prior to 1997, when Kelly [33] introduced the system problem, researches in congestion control was intuitive, based on laboratory experiment, simulation and validation. But with the Kelly [35] paper titled "Rate control in Communication Networks", research community began to model congestion control mathematically. There has been a vast research effort in this area. Currently, research activities in this area are large and quite a number of models have been proposed in literature. One important research direction is the search for new models to replace the Jacobson algorithm [1].

Jacobson AIMD has worked so well on the Internet metamorphosising from a few number of users/network to a very big giant networks with million of networks and over a billion users world wide. However, the adequacy of these models has been questioned by many researchers in today's and future Internet traffic which is changing rapidly. Currently over 80% of the Internet traffic is TCP traffic. However this ratio will change rapidly in the face of anticipated growth of traffics like multimedia application protocol, voice over Internet, video conferencing, games etc. which use protocols that are quite different from TCP. While TCP is self regulating in the face of congestion, other protocols that these heterogeneous traffics are using are quite aggressive and hence the case of fairness comes in when TCP traffic share a bottleneck link with other traffics on the Internet.

Hence mathematical models of congestion control are being proposed in literature. Moller [32] classified models for congestion control and related tools as;

- Packet level models: A packet level model accounts for the location of each individual packet as the packets are queued and forwarded by the network. The system state evolves as a series of discrete events. Events in Internet are arrival and departure of packets, and timeouts.
- Fluid flow models: A fluid flow model sees the data transport as a continuous fluid, with no packet boundaries. State variables vary continuously, and are described using differential equations. In congestion control, fluid flow models do not capture all details of the dynamics. Instead the state variables represent averages of the true system state.
- Hybrid Models: Here, evolution of the state is a result of discrete events together with continuous changes between events. A continuous model is used for queuing dynamics and end host actions while action like multiplicative decrease of TCP is modeled as discrete event.

Jacobson's congestion control algorithm operates in two phases: slow start and congestion avoidance phase.

4.1. Slow Start Phase

1) *cwnd* = 1; *Start with a window size of* 1
2) *while* (*ssthresh* ≠ *cwnd OR not* 3 *DUPACK*) {
   IF ACK then
   cwnd=cwnd + 1; } *Increase the window size by* 1 *for every ACK received. Repeat until*:
   The ssthresh is reached OR packet-loss is detected
3) *if ssthresh == cwnd then*
   Transit to congestion avoidance; *If ssthresh is reached, go to congestion avoidance phase.*
4) *if* 3*DUPACK then*
   ssthresh = 0.5 * *cwnd*;
   cwnd = 1;
   branch to step 2; *If a packet loss is detected*

Note: Set the initial value of ssthresh to fraction (say half) of the maximum window size. This is determine at the beginning of transmission

4.2. Congestion Avoidance Phase

1) Increase the window size by 1 (cwnd) for every ACK received. This implies that the window size is increased by 1 after all ACKs for that window has been received.

2) When packets loss is detected, decrease the window size, then transit to slow start phase.

Detecting and decreasing the window size has been the major focus of researched in literature and quite a huge number of proposals have been made resulting in the formulation of different TCP variants like TCP-Tahoe, TCP-Reno, TCP SACK, TCP New-Reno etc. For the purpose of modeling hence forth we will refer to all of them as TCP New-Reno. In TCP New-Reno, packet-loss is detected either through:

- The loss of 3 consecutive packets or
- By the RTO time-out

4.3. The Slow Start Phase Analysis

To explain the algorithm of Jacobson [1] further, let the following example be considered as presented in [42]. Assume a single TCP source is accessing a single link that was discussed in [43]. Suppose c is the link capacity in packets/second. Let r denote the round trip propagative delay and T (the sum of propagation delay and queuing delay) is given by

$$T = r + 1/c$$

Now suppose that link capacity is 50 Mbps and packet size of 4000 bytes then

$$c = 50,000,000/4000$$

$$c = 12{,}500 \ \text{packet/sec}$$

$$1/c = 0.08 \ \text{msec}$$

Assume the transmission is over a distance of 2000 km of fibre-optic with speed of light = 3 × 108 m/sec (ignore the refractive index of transmission media)

Round trip propagation delay (r);

$$r = \left(2 \times 10^3\right) / \left(3 \times 10^8\right)$$
$$r = 2/3 \times 10^{-5}$$
$$r = 0.66 \times 10^{-5}$$
$$r = 6.6 \times 10^{-6}$$
$$r = 6.6 \text{ msec.}$$
$$T = 6.6 + 0.08 \text{ msec}$$
$$T = 6.68 \text{ msec}$$

The Product cT is called bandwidth-delay product.

Denote by B, the buffer size of the link (route).

Assumptions:

1) $cT \geq B$

2) *Propagation delay from source to sink is negligible*

3) *Propagation delay from link to source is T.*

Note that packets are released by the source at rate c and they are released at 1/c seconds.

ACKs in transit = cT

Number of packets in buffer = B

Total unacknowledged packets = $cT + B$

Therefore, maximum window size,

$$wmax = cT + B$$

any increase above this quantity will lead to buffer overflow and packed loss. If packet loss occurs, as explained, the ssthresh will be set to half the current window size. (this will be slightly larger than wmax)

Let us assume that

$$ssthresh = \left(mT + B\right)/2$$

where $c \leq \mu$

Hence, the transitions at the slow start can be viewed as presented in **Table 2**. In **Table 2**, a cycle refers to the time it takes the window size to double. At time t = 0, the source released the first packets, the ACK for this packet got to the source after one RTT which is denoted by T time units. The receipt of this ACK increases the window size from 1 to 2 and triggers the release of 2 packets into the network. This begins the next cycle. It takes 1 RTT for the first of the packets to be acknowledged. The window size becomes 3 and only 1 unACKed packet is in the network, hence 2 additional packets are released into the network when the next ACK is received, window size is increased to 4 at time t = 3T and the next, cycle is started. Then remains 2 unACKed packets in the network, hence 2 additional packets are released [42]. From the table, we observed the following:

Window size is given by

$$W\left(nT + m/c\right) = 2n - 1 + m + 1,\ 0 \leq m \leq 2n-1 \quad (1)$$

Queue length is given by

$$Q\left(\mu T + m/c\right) = m + 2,\ 0 \leq m \leq 2n-1 \quad (2)$$

Table 2. Slow start transition cycles (source: modified from [42]).

Cycle	Time	Acked packet	Window size	Max packet released
Cycle 0	O	-	1	1
Cycle 1	T	1	2	3
Cycle 2	2T	2	3	5
	2T + 1/c	3	4	7
Cycle 3	3T	4	5	9
	3T + 1/c	5	6	11
	3T + 2/c	6	7	13
	3T + 3/c	7	8	15
Cycle 4	4T	8	9	17
	4T + 1/c	9	10	19
	4T + 2/c	10	11	21
	4T + 3/c	11	12	23
	4T + 4/c	12	13	25
	4T + 5/c	13	14	27
	4T + 6/c	14	15	29
	4T + 7/c	15	16	31
Cycle 5	5T	16	17	33
	5T + 1/c	17	18	35
	5T + 2/c	18	19	37
	…			

[1]*Cycle = Time for which it takes the window size to double*

Max queue length is a cycle

$$Qm = 2n - 1 + 1 \quad (3)$$

Max window size in a cycle

$$Wm = 2n \quad (4)$$

Note that $Qm \approx wm/2 \leq \left(cT + B\right)/4$ and $Qm \leq B$

At the slow start state, the sufficient condition for buffer not to overflow;

$$\left(cT + B\right)/4 \leq B$$
$$B \geq cT/3$$

If $B < cT/3$, buffer overflow occurs because

$Wmax = cT + B$, meaning that $Q > B$.

Therefore at the point where the window size is $Wmax/2$,

$$Q = Wmax/4$$
$$Q = Wmax = \left(cT + B\right)/4 > B$$

from here

$$B < cT/3$$

and overflow does occur.

From the foregoing, there are two possible cases:

Case 1: $B \geq cT/3$ (Buffer does not overflow)

Denote the length of the slow-start phase by Tss. From the **Table 2** observe that

$$W(t) \approx 2t/T$$

Hence Tss is given by

$$2Tss/T = \left(cT + B\right)/2$$
$$\therefore Tss = T\log 2\left(cT + B\right)/2$$

Case 2: $B < cT/3$

In this case, there will be two slow start phase. The first phase, buffer overflows and there is a packet loss which reduces the window size to 1 and slow start is re-entered.

During the first slow start

$$Tss1 = T \log 2 \ (2B+T)$$

(the additional T is added because it takes one RTT to detect packet loss).

$$Nss1 = 2B$$

Window size at packet loss is

$$\min (4B - 2, ssthresh)$$

where $ssthresh = (cT + B)/2$

In the second slow-start

$$ssthresh = \min\left(2B-1,(cT+B)/4\right)$$

(Since ssthresh is half of window size)

Hence

$$Tss2 = T \log 2\left(\min\left(2B-1,(cT+B)/4\right)\right)$$

$$Nss2 = \min\left(2B-1,(cT+B)/4\right)$$

Generally

$$Tss = Tss1 + Tss2$$

and

$$Nss = Nss1 + Nss2$$

5. Modifications of Slow-Start

There have been several modifications of the slow start stage to overcome the problem of performance associated with it. From literature, there are three lines of studies. A group of researchers used capacity estimation techniques to estimate available bandwidth and set the congestion window size using the estimated bandwidth. In this group belongs the work of Patridge [44] which proposed Swift Start for TCP. Swift Start used an initial window (cwnd) of 4 packets and thereafter estimates the available bandwidth in the first round trip time. It uses the estimated bandwidth to calculate the bandwidth-delay product (BDP) of the network and set the cwnd to a percentage of the calculated BDP. Lawas-Grodek and Tran [45] carried out a performance evaluation of Swift Start and submitted that Swift Start improves network performance when the network is not congested however when the network overflows, the estimation of the cwnd drift away from accuracy. This is due to retransmission of delayed or lost ACKs and RTO timeouts.

Another proposal that uses capacity estimation to determine the size of the congestion window is Restricted slow-start by [46]. Restricted Slow start used a PID control algorithm as proposed by [47] to determine the rate of increase at the slow start phase. In PID control approach, the controller calculates an output that determines the new value of the sender cwnd. This approach has extra overhead for the computation of PID and furthermore, it was not designed to work in large BDP network.

Shared Passive Network Performance Discovery (SPAND) proposed by [48] is another technique classified into this group. SPAND collects current network state and gains optimal initial parameters to determine an initial sending rate. The weakness of this approach is in it needs for leaky bucket pacing for outgoing packets which can be costly and problematic.

Adaptive Start (Astart) by Ren Wang *et al* [49] uses the Eligible Rate Estimation (ERE) mechanism proposed by TCP Westwood adaptively and repeatedly resetting ssthresh during the slow start phase. When ERE indicates that there is more available capacity, the connection increases its cwnd at a faster rate, on the other hand, when ERE indicates that the network is close to congestion, the connection switches to congestion avoidance limiting the risk of buffer overflow.

Capstart proposed by Canvendish *et al.* [50] estimates path capacity after the TCP session has been established and uses it to tune TCP to deliver higher transfer speed. Capacity estimation is done using two network scenarios. These are capacity expansion and capacity reduction which is defined in relation to TCP sender speed and path bottleneck speed. If bottleneck link capacity is greater than sender speed then the capacity expanded otherwise, it is capacity reduced. This protocol is capable of adjusting itself to the available bandwidth whether high or low, however, it did not take other network conditions into account, for instance, network dynamism such as available bandwidth at various moments depending on changes occurring within the network such as connections establishments or terminations. The second group employs parameter-based manipulations to determine transmission speed. Commonly used network parameters are the ssthresh and cwnd. TCP Fast Start by [51] records the recent network parameters (cwnd and ssthresh) to reduce the start time of a new connection and to reduce transmission delays. These parameters may be too aggressive or too conservative when network condition changes. Chen *et al.* [52] proposed the collection of recent history information on the network which shall be used to initialize parameters for a new connection. Parameter setting based history information can not fit the dynamic changes of network and violates slow start principle.

In TCP Vegas, [17] restricts the growth of cwnd by doubling cwnd only at every other RTT. TCP Vegas can not handle multiple packet losses in one window. Limited slow start by [53] aims at eliminating exponential growth of cwnd which causes large packet losses. The approach limits the exponential growth up to a max-ssthresh parameter value over which the cwnd is in-

creased by a fraction of the current cwnd value. This replaces the exponential growth with a near linear cwnd growth when the congestion window size is greater or equal to max-ssthresh. This will make the congestion window grow slowly after max ssthresh is reached and this makes limited slow start not suitable for large capacity network.

Other proposals in the second group include New Parameter-Config Based Slow Start Mechanism (P-Start) by [27] increases the congestion window exponentially while the cwnd is less than $\text{ssthresh}/2$, otherwise increases by $(\text{ssthresh} - \text{cwnd})/2$ and gradually approaches ssthresh until (ssthresh-cwnd) is less than the factor of d where $\text{ssthresh}/2 \geq d \geq 2$. There after congestion avoidance is entered. The major feature of P-start is in the manner of cwnd increases that is small amplitude at start and transition to congestion avoidance. Changing in sending rate is smooth and has minor impact on the flows in the network; however, P-start may waste bandwidth and may not get to maximum transmission speed in good time thereby performing worst than slow start. In addition, P-start will perform poorly in a high BDP network thereby not suitable for future gigabit networks.

The third group obtains information and/or request assistance from the network/link. Congestion manager proposed by [54] collects congestion status information and feedback from receivers and share it with endpoints and connections in the network. This will enable connections determine the congestion status of the network and thereby determine an initial sending rate. The congestion manager has a weakness of being beneficial to only connections that are initiated almost at the same time and secondly, it is only those connections and endpoints that supplied feedback that can benefit from this scheme. Some techniques require explicit network assistance to determine a practicable starting sending rate. Prominent among this group is the work of [55] called Quick Start explained below.

According to Floyd *et al.* [55], the experimental Quickstart TCP extension is currently the only specified TCP extension that realizes a fast startup. A large amount of work has already been done to address the issue of choosing the initial congestion window for TCP. RFC 3390 [56] allows an initial window of up to four packets. Quick start is based on the fact that explicit feedback from all routers along the path is required to be able to use an initial window larger than those specified by RFC 3390.

In quick start proposals, a sender (TCP host) would indicate its intention to send at a particular rate in bytes per second. Each router along the path could approve, reduce, disapprove or ignore the request. Approval of the request by the router indicates that it is being underutilized currently and it can accommodate the sender's requested sending rate. The quick start mechanism can detect if there are routers in the path that disapproved or do not understand the quick start request. The receiver communicates its response to the sender in an answering TCP packet. If the Quick start request is approved by all routers along the path, then the sender can begin transmission at the approved rate. Subsequently, transmissions will be governed by the default TCP congestion control mechanism. If the request is not approved, then the sender transmits at the normal TCP startup speed [57].

According to [57], TCP is effective for both flow control and congestion control. The TCP flow control is a receiver-driven mechanism that informs the sender about the available receive-buffer space and limits the maximum amount of outstanding data. Flow control and congestion control are independent and the size of receive buffer space depends on the capacity of the receiver network. However, if the TCP is used with a link with large bandwidth-delay product, both congestion window and flow control window need to be large in other for TCP to perform well [55]. This makes both flow control and congestion control to overlap. The Quickstart TCP extension assumes independence of flow and congestion control. This is not true which makes it inappropriate for the global internet. In addition, most routers and other network components on the global Internet are not built with capabilities to be quick start aware which implies that they need either to be replaced or modified before it can be used. Hence, we propose a milder approach to fast startup which we call (E-speed startup). This is a form of fast startup that does not need routers' response or modification; rather it builds end-to-end principles. This proposal is compatible with the current Internet as well as the future Internet.

Table 3 summarizes the various slow start modifications that exist in literature. E-speed start combines the feature of the two groups (Parameter based manipulation capacity/bandwidth estimation).

6. Discussion on Problems Associated with Individual TCP Variant

The control mechanism of TCP Tahoe has a problem. If a packet is lost, the sender may have to wait a long period of time for a time-out to occur for such loss to be detected and retransmitted. The data may not be retransmitted for a very long time in networks with large delay. TCP Reno was designed to solve this problem. Although Reno perform well over TCP Tahoe when the packet losses are small, its performance is not good when there are multiple packet losses in a single RTT. Reno can handle a single packet loss. If there are multiple losses, the first duplicate ACKs received will trigger the retransmission of the first packet that was lost. The next

lost packet will be detected when the sender receive the ACK for the retransmitted data after one RTT. In addition cwnd may be reduced twice for packet losses which occurred in one window. Another problem of RENO is that if the window is very small when the loss occurs, the sender may not receive enough duplicate ACKS for a fast retransmit. Hence it has to wait for a time out to occur before detecting that the packet is lost. These problems are solved with the introduction of TCP New-Reno.

One problem with the New-Reno algorithm is its inability to detect any other packet-loss in the window. This implies that New-Reno suffers from long delay in detecting each packet loss *i.e.* it takes more one RTT. Hence selective acknowledgment (SACK) was proposed by [6]. SACK is an extension of TCP Reno and TCP New-Reno. It intends to solve the problems of TCP Reno and New Reno *i.e.* detection of multiple packet loss and Retransmission of more than one lost packet per RTT. One major set back of SACK is in its selective acknowledgement which is a bit cumbersome to implement.

Delay-based congestion control algorithms were proposed to solve the problems associated with the loss based congestion control models. Most delay based congestion control algorithm has fairness problems when sharing link with TCP New Reno. This is because they become more aggressive when they are at the peak of transmission when they suppose to slow down transmission [26]. BIC-TCP and CUBIC try to solve this problem by monitoring the window size when it experienced a packet loss and slows down transmission as the window size approach this monitored window size. [24]. **Table 4** summarizes the deficiencies of the various TCP Variants:

7. Research Direction

Currently there are three different research directions intending to develop or make TCP better: Improving TCP performance over links with large bandwidth delay product, improving TCP performance over wireless and reducing the queuing delay at bottleneck links thereby improving quality for real time applications. Other focus of research includes finding a suitable startup speed for slow start most especially in a high bandwidth delay product. TCP modifications of the future will work well with gigabit networks as well as backward compatible with low speed networks, a protocol fair to both itself and to other flows in the network *i.e.* intra and inter protocol fairness. A new line of research bothers on congestion control of ACK packets. Controlling acknowledgment packets congestion is a novel problem and differs from the techniques used in controlling data packet congestion. This is a new research direction that requires investigation.

Table 3. Modifications of slow start.

Protocol	Type	Purpose
TCP Vegas [16]	Parameter based	Double cwnd every other RTT.
TCP Fast Start [51]	Parameter based	Used network parameters to reduce connections' start time and transmission delay.
Recent history information collection [58]	Parameter based	Collects recent history information and use it to initialize connection parameters.
Limited slow start [53]	Parameter based	Eliminates the exponential growth of cwnd.
P-Start [59]	Parameter based	Increases cwnd with small calculated amplitude after ssthresh/2 has been reached.
FAST-FOR-WARD START [60]	Parameter based	uses low-priority data segments, namely, supplement segments to calculate cwnd.
SPAND ([48]	Bandwidth Estimation	Obtain network state to determine initial sending rate.
Swift Start [44]	Bandwidth Estimation	Used initial packet of 4 and then estimate available bandwidth in the first round trip.
PACED START [60]	Bandwidth Estimation	Used the difference between the data packet spacing and the acknowledgement spacing for estimating appropriate cwnd.
Astart [49]	Bandwidth Estimation	Used Eligible Rate Estimation mechanism to determine available bandwidth to adaptively and repeatedly reset ssthresh and cwnd.
Restricted slow start [46]	Bandwidth Estimation	Used PID control algorithms to calculate a value of the sender cwnd
Capstart [50]	Bandwidth Estimation	Estimate path capacity and tune TCP to deliver at higher transfer speed.
Congestion Manager [54]	Require network assistance	Collects congestion status information and share with endpoints.
Quick Start [55]	Require network assistance	Obtain explicit feedback from all routers along the path to transmit at an approved large rate.

Table 4. Deficiencies of TCP variants.

TCP Variants	Issues solved	Inherent problems
TCP Tahoe [1]	Introduced the concept of congestion control	Loss detection is after a time out
TCP Reno (Jacobson, 1990)	Introduced fast retransmit and fast recovery	Could not detect multiple losses within one RTT
TCP New Reno [6]	Solved the problem of multiple losses	detection of each packet loss takes one RTT
TCP SACK [13]	Introduced Selective Acknowledgement	Difficult to implement
TCP Vegas [16]	Delay based	Fairness issue
Fast TCP [25]	Delay based	Fairness issue
HSTCP(Floyd S. 2003)	Loss based	Fairness issue
STCP [20]	Loss based	Fairness issue
BIC – TCP [23]	Loss based	Fairness issue
CUBIC [24]	Loss-Delay based	Fairness issue
TCP Africa [26]	Loss-Delay based	Fairness issue
Compound TCP [21]	Loss-Delay based	Fairness issue
TCP Illinois [22]	Loss-Delay based	Fairness issue
West wood + ([27]	Bandwidth estimation	Fairness issue
XCP [28]	Extra signaling	Fairness issue
SPAND [48]	Obtain network state to determine initial sending rate.	It requires the costly and problematic leaky bucket.
TCP Fast Start [51]	Reduced connection start time and transmission delay	Too aggressive when network conditions change.
Congestion Manager [54]	Collects and share congestion status information	Beneficial to only connections that are initiated almost at the same time and endpoints that supplied feedback.
Swift Start [44]	TCP start up using 4 packets.	The estimation of the cwnd drift away from accuracy when there is congestion.
Recent history information collection [58]	Collects recent history information and use it to initialize connection parameters.	Can not fit the dynamic changes of network and violates slow start principle
Limited slow start [53]	Eliminates the exponential growth of cwnd.	May be too conservative in a high BDP.
Restricted slow start (Allcock *et al*, 2004)	used a PID control algorithm	extra overhead for the computation of PID
Astart (Wang R. *et al*, 2004)	Uses ERE mechanism adaptively and repeatedly resetting ssthresh during the slow start phase.	May not be able to use its fair share of available bandwidth
Capstart [50]	Capstart proposed by estimates path capacity and use it to tune TCP to deliver higher transfer speed	Did not take other continuously changing network conditions into account.
P-Start [59]	Increases cwnd by small amplitude after ssthresh/2 has been reached.	May waste bandwidth resource as it may not get to a high transmission rate in good time.

8. Conclusions

We have reviewed here, various modifications of TCP in history. There is a large number of works done in this area. The review carried out here focused on window based congestion control. The internet protocol of choice, TCP New-Reno, has performed well in today's internet,the question is, how will TCP co-habit with other protocols that are more aggressive and non responsive to congestion indication? A possible answer is that future protocols may have the requirements of matching up with this aggressiveness while controlling congestion effectively. Furthermore, super gigabit network will replace today's network. How will TCP increase its transmission speed to match this high bandwidth network? What should be the slow start behaviour of TCP under this situation? Will TCP be both backward and forward compatible with low speed and high speed network without necessarily using network resources sub-

optimally? These are questions that must be answered in finding a replacement for today's internet protocol of choice- the TCP. In addition, it was observed that all reviewed congestion control techniques used only a single startup for TCP, this is in most cases the slow start or its modification. However, the option of having multiple startups for TCP using the prevailing operating conditions at the time of connection is a research line that should attract the attention of the Internet research community. A research in this area called e-speed start uses environmental (*i.e.* operating) parameters to determine whether to use the conventional TCP slow start or any other startup algorithms depending on the network operating conditions. It is hoped that when completed, e-speed start will address the problem of TCP start up in both highspeed and lowspeed networks.

9. References

[1] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of ACM Sigcomm*, Scanford, 26-30 August 1988, pp. 314-329.

[2] V. Jacobson, R. Braden and D. Borman, "TCP Extensions for High Performance," RFC 1323, Internet Engineering Task Force, 1992. http://www.left.org/rfc/rfc/1323.txt

[3] W. R. Stevens, "TCP/IP Illustrated," *The Protocols*, Vol. 1, Addison-Wesley, Reading, 1997.

[4] M. Mathew and M. Jamshidi, "Forward Acknowledgment: Refining TCP Congestion Control," *Proceedings of the ACM SIGCOMM*, Vol. 26, No. 4, 1996, pp. 281-291.

[5] M. Allman, V. Paxson and W. R. Stevens, "TCP Congestion Control," RFC 2581, 1999. www.icir.org/mallman/papers/rfc2581.txt

[6] S. Floyd and T. Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm," RFC 2582, Internet Engineering Task Force, April 1999. http.//www.ietf.org/rfc/rfc2528.txt

[7] J. Hoe, "Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes," Master Theses, Massachusetts Institute of Technology, 1995.

[8] J, T, Wang, "The Study of Internet Congestion Control: Equilibrium and Dynamics," PhD Thesis, California Institute of Technology, 2005.

[9] L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control for Fast Long Distance Networks," *Proceedings of IEEE INFOCOM*, 2004.

[10] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *Technical Report*, April 1990.

[11] J. C. Hoe, "Improving the Start up Behaviour of a Congestion Control Scheme for TCP," *Proceedings of ACM Sigcomm*, 1996, pp. 270-280.

[12] K. Fall and S. Floyd, "Simulation Based Comparisons of Tahoe, Reno and SACK TCP," *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 3, July 1996. doi:10.1145/235160.235162

[13] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Internet Engineering Task Force, October 1996. http://www.ietf.org/rfc/rfc2018.txt

[14] S. Keshav and H. Saran, "Semantics and Implementation of a Native-Mode ATM Protocol Stack," 1994. citeseer.ist.psu.edu/52943.html

[15] R. Jaine, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Network," *ACM Computer Communication Review*, Vol. 19, No. 5, 1989. pp. 56-71. doi:10.1145/74681.74686

[16] L. S. Brakmo, S. W. O'Malley and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proceedings ACM SIGCOMM*, London, August 31-September 2, 1994.

[17] L. Brakmo and L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on Global Internet," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, 1995, pp. 1465-1480. doi:10.1109/49.464716

[18] R. S. Jayaram and I. Rhee, "A Case of Delay-Based Flow Control in CDMA 2.5G Networks," *International Conference on Ubitiquitous Computing*, Washington, 2003.

[19] S. Floyd, "High Speed TCP for Large Congestion Windows," RFC 3649, 2003. www.ietf.org/rfc/rfc3649.txt

[20] T. Kelly, "Scalable TCP Improving Performance in High Speed Wide Area Networks," *Computer Communications Review*, Vol. 32, No. 2, 2003, pp. 83-92. doi:10.1145/956981.956989

[21] K. Tang, J. M. Song, Q. Zhang and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," *Proceedings of IEEE INFOCOM*, Barcelona, 23-29 April 2006.

[22] S. Liu, T. Basar and R. Srikant, "TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks," *Proceedings of First International Conference on Performance Evaluation Methodololy Tools*, 2006.

[23] L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," *Technology Report*, Computer Science Department, NC State University, Raleigh, 2003.

[24] I. Rhee and L. Xu, "CUBIC: A New TCP—Friendly High Speed TCP Variant," *International Workshop on Protocols for Fast Long—Distance Networks*, Lyon, 3-4 February 2005.

[25] D. X. Wei, C. Jin, S. H. Low, and S. Hedges, "Fast TCP: Motivation, Architecture, Algorithms, Performance," *IEEE-ACM Transaction on Networking*, Vol. 14, No. 6, 2006. pp. 1246-1259. doi:10.1109/TNET.2006.886335

[26] R. King, R. Baraniuk and R. Riedi, "TCP—Africa; an Adaptive and Fair Rapid Increase Rule for Scalable TCP," *Proceedings of IEEE INFOCOM*, Vol. 3, 2005, pp. 1838-1848.

[27] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi and R. Wang, "TCP Westwood: Bandwidth Estimation for En-

hanced Transport over Wireless Links," *Mobile Computing and Networking*, Rome, 16-21 July 2001, pp. 287-297.

[28] S. H. Low and D. E. Lapsley, "Optimization Flow Control: Basic Algorithm and Convergence," *IEEE/ACM Transaction on Networking*, Vol. 7, No. 6, 1999, pp. 861-874.

[29] P. Sarolahti, M. Allman and S. Floyd, "Determining an Appropriate Sending Rate over an Underutilized Network Path," *Computer Networks*, Vol. 51, No. 7, 2007, pp. 1815-1832. doi:10.1016/j.comnet.2006.11.006

[30] A. O. Oluwatope, A. B. Obadire, G. A. Aderoumu and M. O. Adigun, "End-to-End Performance of Selected TCP Variants Across a Hybrid Wireless Network," *Issues in Information Science and Technology*, Vol. 3, 2006.

[31] D. Katabi, M. Handley and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," *Proceedings of the* 2002 *Conference on Applications*, *Technologies*, *Architectures*, *and Protocols for Computer Communications*, Pittsburgh, 19-23 August, 2002.

[32] M. Neils "Window Based Congestion Control, Modeling Analysis and Design," Doctoral Thesis, KTH, Stockholns, 2008.

[33] F. Kelly, "Changing and Rate Control for Elastic Traffic," *European Transactions on Telecommunications*, Vol. 8, No. 1, 1997, pp. 33-37. doi:10.1002/ett.4460080106

[34] S. Liu, "Primal-Dual Congestion Control Algorithms and E-Red AQM Scheme," PhD Thesis, University of Illinois Urbana, Illinois, 2003.

[35] F. P. Kelly, A. K. Maulloo and D. K. H. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of Operational Research Society*, Vol. 49, No. 3, 1998, pp. 237-252.

[36] H. Yaiche and R. R. Mazumdar, "A Game-Theoretic Framework for Rate Allocation and Charging of Elastic Connections in Broadband Networks," IEEE/ACM Transactions on Networking, Vol. 8, No. 5, 2000. doi:10.1109/90.879352

[37] T. Alpcan and T. Basar, "A Globally Stable Adaptive Congestion Control Scheme for Internet-Style Networks with Delay," *IEEE Wireless Communications*, Vol. 12, No. 6, 2005, pp. 42-49.

[38] G. Vinnicombe, "On the Stability of Networks Operating TCP—Like Congestion Control," *IFAC World Congress*, Barcelona, 2002.
Available at http://www.eng.cam.ac.uk/gv.

[39] R. J. La and P. Ronjan, "Asymptotic Stability of a Ratio Control System with Communication Delays," *IEEE Transactions on Automatic Control*, Vol. 52, No. 10, 2007, pp. 1920-1925.

[40] L. Ying, G. E. Dullerud and R. Srikant, "Global Stability of Internet Congestion Controllers with Heterogeneous Delays," *IEEE/ACM Transactions on Networking*, Vol. 14, No. 3, 2006, pp. 579-591. doi:10.1109/TNET.2006.876164

[41] F. Paganini, J. Doyle and S. Low, "Scalable Laws for Stable Network Congestion Control," *Proceedings of the* 40*th IEEE Conference on Decision and Control*, Orlando, 4-7 December 2001, pp. 185-190.

[42] R. Srikant, "The Mathematics of Internet Congestion Control," Birkhauser, Boston, 2004.

[43] T. V. Lakshman, B. Suter and U. Madhow, "Window-Based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A Study of TCP/IP Performance," *Proceedings of the INFOCOM*'97. 16*th Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution* (*INFOCOM*), Washington, DC, 9-11 April, 1997, p. 1199.

[44] C. Patridge, D. Rockwell, M. Allman, R. Krishman and J. Slerbenz, "A Swifter Start for TCP," Technical Report TR8339 BBN Technologies, 2002.

[45] F. J. Lawas-Grodek and D. Trans, "Evaluation of Swift Start TCP in Long-Delay Environment," Tran Glenn Research Center, Cleveland, 2004.

[46] W. Allcock, S. Hedge and R. Kettimuthu, "Restricted Slow Start," *IEEE International Conference on Cluster Computing*, Boston, 26-30 September, 2005.
http://doi.ieee.computersociety.org/10/1109/cluster2005.347079.

[47] J. Gerry, "A Comparison of PID Control Algorithm," *Journal of Control Engineering*, Vol. 34, No. 41, 1987, pp. 102-105.

[48] S. Seshan, M. Stemm and R. Katz, "Shared Passive Network Performance Discovery (SPAND)," *Proceedings of USITS*'97, Monterey, 1-12 May, 1992.

[49] R. Wang, G. Pan, K. Yamada, M. Sanadidi and M. Gerla, "TCP Startup Performance in Large Bandwidth Networks," *IEEE IFOCOMM*, Vol. 2, 2004, pp. 796-805.

[50] D. Canvendish, K. Kumazoe, M. Tsuru and Y. Oie, "Capstart: An Adaptive TCP Slow Start for High Speed Networks," Department of Computer Institute of Technology, Japan, 2009.

[51] V. Padmanabhan and R. Katz, "TCP Fast Start: A Technique for Speeding up Web Transfers," *Proceedings of IEEE GLOBECOM*, Sydney, 8-12 November, 1998, pp. 41-46.

[52] J. Chen, M. Zhang and Q. Meng, "A Network Congestion Control Algorithm Based History Connections and Its Performance Analysis," *Journal of Computer Research and Development*, Vol. 40, No. 10, 2003, pp. 1470-1475.

[53] S. Floyd, "Limited Slow Start for TCP with Large Congestion Window," RFC3742, 2004.
www.apps.ietf.org/rfc/rfc3742. html

[54] H. Balakrishman, H. Racheal and S. Seshan, "An Integrated Congestion Manager Architechture for Internet Hosts," *Proceedings of ACM SIGCOM*, Cambridge, August 30-September 3, 1999.

[55] S. Floyd, M. Allman, A. Jain and P. Sarolahti, "Quick-Start for TCP and IP," RFC 4782, 2007.
www.rfc-editor.org/rfc/rfc4782.txt;

[56] M. Allman, S. Floyd and C. Partridge, "Increasing TCP's Initial Wiondow," RFC 3390, 2002.
www,icir.org/mallman/papers/rfc3390.txt

[57] M. Scharf, S. Floyd and P. Sarolahti, "TCP Flow Control

for Fast Startup," 2009. http://www.ietf.org/ietf/1id−abstracts.txt.

[58] Z. Chen, X. Deng, L. Zhang and B. Zeng, "A New Parameter-Config Based Slow Start Mechanism," *Journal of Communication and Computer*, Vol. 2, No. 5, 2005, pp. 56-62.

[59] S. Utsumi, S. M. S. Zabir and N. Shiratori, "TCP-Cherry: A New Approach for TCP Congestion Control Over Satellite IP Networks," *Computer Communications*, Vol. 31, No. 10, 2008, pp. 2541-2561. doi:10.1016/j.comcom.2008.03.029

[60] N. Hu and P. Steenkiste, "Improving TCP Startup Performance Using Active Measurements: Algorithm and Evaluation," *Proceedings of the* 11*th IEEE international Conference on Network Protocols* (*ICNP*), Washington, DC, 30 September 2003, p. 107.

