A Sender-Side TCP Enhancement for Startup Performance in High-Speed Long-Delay Networks

Xiao Lu, Ke Zhang, Cheng Peng Fu, and Chuan Heng Foh
School of Computer Engineering
Nanyang Technological University, Singapore
Luxi0007, Y030069, ascpfu, aschfoh@ntu.edu.sg

***Abstract*—Many previous studies have shown that traditional TCP slow-start algorithm suffers performance degradation in high-speed and long-delay networks. This paper presents a sender-side enhancement, which makes use of TCP Vegas congestion-detecting scheme to monitor the router queue, and accordingly refines slow-start window evolution by introducing a two-phase approach to probe bandwidth more efficiently. Moreover, it achieves good fairness of bandwidth utilization in coexistence of multiple connections. Simulation results show that, compared with traditional slow-start and many other enhancements, it is able to significantly improve the startup performance without adversely affecting coexisting TCP connections.**

I. Introduction

TCP is a connection-oriented, reliable and in-order transport protocol. The current legacy TCP, namely TCP Reno [1], and its enhancements such as NewReno [2] use slow-start during startup phase to probe available bandwidth by gradually increasing the amount of data injected into the network. However, blind initial *ssthresh* (slow-start threshold) setting of legacy TCP leads to one possible problem that traditional slow-start algorithm may suffer performance degradation in high-speed long-delay networks. Before a TCP connection starts, slow-start sets initial *ssthresh* to an arbitrary default value within a range, depending on different operating system implementations. As a result, 1) if *ssthresh* is set too high compared with the bandwidth-delay product (BDP), multiple packet losses and timeout may occur; otherwise 2) If *ssthresh* is set too low, the TCP connection will exit slow-start and switch to congestion avoidance phase prematurely. Both cases cause low link utilization.

In this paper, we present a sender-side enhancement that is simple and efficient to improve TCP startup performance in high-speed long-delay networks. The key idea is to make use of TCP Vegas congestion-detecting scheme to monitor the router queue, and then a two-phase approach is used to refine *cwnd* evolution. We note that several existing startup modifications are based on the widely used congestion control algorithm, NewReno, i.e., Hoe's Change [5] and Limited Slow-Start [6]. Thus, to make performance comparison, our enhancement is also combined with NewReno. Simulation results demonstrate that, compared with traditional slow-start and many other enhancements, our methods significantly improves link utilization without adversely affecting coexisting TCP connections. Furthermore, the enhanced throughput performance is achieved by using the bandwidth effectively and fairly rather than aggressively depriving bandwidth from other TCP connections. Therefore, our algorithm causes little negative effect on other co-existing TCP connections.

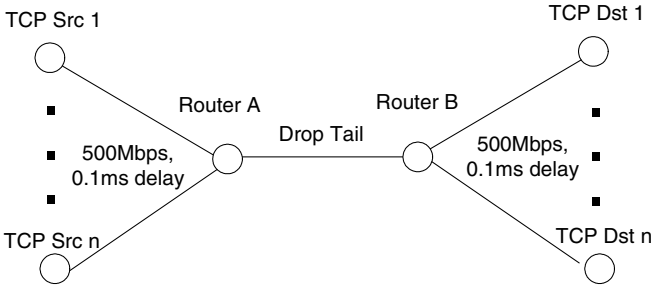

Fig. 1. Network simulation topology.

The remainder of the paper is organized as follows. We start by showing the traditional slow-start limitations demonstrated by stimulations in the next section. After summarizing some related works in Section III, we provide the analytical approach and describe the enhanced algorithm used by sender side in Section IV, validate it through simulations in Section V, and conclude this paper in Section VI.

II. Traditional slow-start limitations

We use ns-2.28 [16] [17] to simulate TCP startup performance in high-speed long-delay networks. Fig. 2 shows the simulation topology. TCP Src represents TCP sender and TCP Dst TCP receiver. Router A and Router B are two Droptail bottleneck routers. Side links are all with bandwidth of 500 Mbps, and one-way delay of 0.1 ms. Between the two routers there is a bottleneck link with 40 Mbps bandwidth and 50 ms one-way delay. For convenience, window size is measured in number of packets, and the packet size is 1000 bytes while ACK is 40 bytes. Thus the BDP value is 500 packets. The bottleneck router is with 250 packets buffer size (BDP/2). TCP sources run NewReno with traditional slow-start, and the maximum burst size is 5 packets. All the experiments in this paper are based on this topology.

In traditional slow-start [1], sender increases *cwnd* (congestion window) by one packet upon receiving every new ACK, until *cwnd* reaches initial *ssthresh*. Before a TCP connection starts, initial ssthresh is set to an arbitrary value, ranging from 4 KB to extremely high. Due to blind initial *ssthresh*

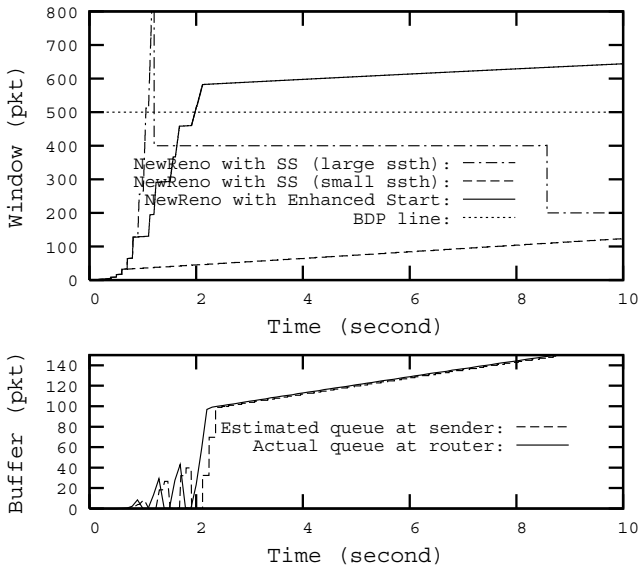

Fig. 2. Comparison of slow-start and Enhanced Start *cwnd* evolutions.

setting, TCP suffers from low startup performance, especially in high-speed long-delay networks. Fig. 2 shows two typical cases where initial *ssthresh* mismatches the BDP. In the first case, *ssthresh* is set to 800 packets, which is higher than the BDP represented by the dot line. We observe that the TCP connection, shown in the semi-dashed line, suffers many packets losses and long recovery time. In the second case, *ssthresh* is set to 32 packets, which is much below the BDP. It shows that the TCP connection, expressed by the dashed line, exits slow-start and switches to congestion avoidance phase prematurely, resulting in a low bandwidth utilization.

III. RELATED WORK

One critical reason of traditional slow-start performance suffering is that, sender lacks the ability to estimate the network condition properly. In recent years, various senderside modifications have been proposed to improve TCP startup performance.

Some approaches aim to solve arbitrary *ssthresh* setting problem by setting *ssthresh* to some estimation value. In [5], Hoe proposed to set initial *ssthresh* to the estimated value of BDP obtained by using packet pair measurement. This method avoids the slow-start limitations, mentioned above, of entering the congestion avoidance phase prematurely. However, attribute to *cwnd* increasing too fast towards the estimated BDP, Hoe's Change may suffer temporary queue overflow and multiple losses when the bottleneck buffer is not big enough compared to the BDP. In [10], a measurement improves Hoe's method by making use of multiple packet pairs to iteratively improve the estimate of *ssthresh* during startup progress. Nevertheless, simply using packet pairs cannot estimate available bandwidth accurately. Comparatively, Early Slow Start Exit (ESSE) [15] is robust against estimation errors. It adopts several approximations of pipesize estimation, based on the observation of few ACK arrival times, to set the initial *ssthresh* value and drastically reduces the packet drop rate. Another modified slow-start mechanism, called Adaptive Start [8], is proposed to make use of eligible rate estimation (ERE) mechanism [7], repeatedly resetting the slow-start *ssthresh* to a more appropriate value. This endues sender with the ability to grow *cwnd* efficiently without packet overflows. Agile Probing [4] uses a similar idea. However, an early transition form slow-start to Congestion Avoidance may occur to affect the throughput performance.

Another approach improves startup performance by modifying the *cwnd* evolution. Smooth-Start [14] splits the slow-start into slow and fast phases to adjust the congestion window in different ways. It is capable of reducing packet loss during startup, however, it still does not address the question of how the *ssthresh* and the threshold that distinguishes the two phases should be set. Limited Slow-Start [6] introduces an additional threshold *max_ssthresh* and modifies *cwnd* increase manner. Namely, when *cwnd* $\leq$ *max_ssthresh*, *cwnd* doubles per RTT, the same as slow-start. When *max_ssthresh*$<$*cwnd* $\leq$ *ssthresh*, *cwnd* is increased by a fixed amount of $max\_ssthresh/2$ packets per RTT. This method reduces the number of drops during the startup, especially for TCP connections that is able to reach very large congestion window. However, *max_ssthresh* is statically set before the TCP connection starts.

IV. ENHANCEMENTS

In this section, we provide the analytical approach that is the fundamental scheme of our measurement. Then, the two-phase approach is introduced, followed by the description of the pseudo code.

A. Analytical Approach

Our proposal, making use of TCP Vegas congestion-detecting scheme to monitor the router queue, attributes to that the congestion avoidance mechanism is based on changes in the estimated amount of extra data in the network rather than only on dropped segments. Past research work [3] [12] has shown this detection, in terms of how many extra buffers the connection is occupying, leads to a more accurate estimation of network traffic condition.

In Vegas, the throughput difference is calculated by:

$$Diff = (Expected - Actual) = (\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT})$$

where ***BaseRTT*** is the minimum of all measured RTT, and ***RTT*** is the actual round trip time of a tagged packet. Denote the backlog at router queue by N, we have,

$$RTT = BaseRTT + N/Actual.$$

Rearrange the above equation, we deduce that,

$$N = (\frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT}) \times BaseRTT. \qquad (1)$$

We note the Rerouting problem [11] this estimation method of TCP Vegas faces. Rerouting a path may change the propagation delay of the connection. More specifically, if a new route for the connection has a longer propagation delay, the

connection will not be able to tell whether the increase in the round trip delay is due to a congestion in the network or a change in the route. However, based on the fact that startup phase only last for a few seconds. Thus, Rerouting does not necessarily affect the startup performance. Therefore, during startup progress we can use (1) to estimate the backlog at router queue. This forms the basis to our enhancement of TCP startup behavior.

B. Startup Enhancement

The key idea here is to detect the backlog status of the bottleneck routers with TCP Vegas congestion detection detecting scheme first, and then modify the startup algorithm to properly react to the backlog. We propose a two-phase approach to adjust probing rate reacting to the changes of buildup queue in the router. A certain threshold of queue length can be used as a signal of queue being building up. The two phases of our measurement are for the queue buildup and non-buildup situation, respectively. After a certain threshold is set, changes of the estimated backlog can be used as a trigger to switch between two phases.

Now, from (1), if N, the estimation of backlog at router queue, exceeds a certain threshold of β packets, we can assume that the router queue is building up and here we call it a congestive event. Each occurring times of congestion event is recorded as a new parameter at TCP sender every time the estimated backlog is greater than the threshold. Next, the two-phase approach is described in detail as follows.

In Linear Increase Phase, when a TCP connection starts, the sender increases *cwnd* by one packet for every ACK received which the same as traditional slow-start. This process continues until the queue length exceeds the threshold β, which marks a congestive event. Such a congestive event may due to either the exponential growth of *cwnd* being too fast for the bottleneck to cope with [4], or bandwidth fully utilization. For the both causes, network bottleneck capacity can be assumed reached. Thus, increasing *cwnd* in a conservative linear manner is more appropriate. We design *cwnd* to increment one packet every round-trip time in this phase. In the former cause of congestion event detection, by switching to increase linearly, sender can quickly drain the temporary queue to avoid buffer overflow and multiple losses. In the latter one, sender can assume that *cwnd* has already met the available bandwidth. Switching to linear increase actually have the same effect as congestion avoidance. This skillfully solves the *ssthresh* setting problem.

In Adjustive Increase Phase, upon sensing that the router queue is drained, a sender enters this phase with the aim to adjust probing rate more intelligently. That is, every time the queue length draw back below the threshold β, implying under-utilization of bottleneck bandwidth, sender should speed up again its sending rate to probe the available bandwidth. However, the increase speed should be reduced because the spared bandwidth is less than before. Therefore, in this phase, the *cwnd* increase speed is set to half of that before the last congestive event.

The startup phase exits when a packet loss event occurs. The pseudo code of our proposed scheme, which we called Enhanced Start, is given in the following.

Algorithm 1 Enhanced Start

```
if (three DUPACKs are received)/*startup phase exits*/ then
    ssthresh=cwnd/2;
    /*switch to Congestion Avoidance Phase*/
else
    if (N ≥ β) then
        cwnd+=1/cwnd; for every ACK
        /*Linear Increase Phase*/
    else
        cwnd+=max(1/cwnd, 1/2exp{Congestion_Event_No});
        for every ACK
        /*Adjustive Increase Phase*/
    end if
end if
```

In the above pseudo code, *Congestion_Event_No* indicates the number of congestion events occurred with its initial value set to 0. Note that our algorithm is to probe eligible bandwidth intelligently during startup phase, when connection has no information about the network to set *ssthresh*. It is not executed after a timeout as *ssthresh* is no longer blindly set.

V. Performance evaluation

In this section, we present numerical results of Enhanced Start, compared with the tradition slow-start and other variant modifications, given different network environments with dissimilar parameter settings. We show the *cwnd* evolutions of Enhanced Start, and then comparisons of throughput achievement. We also show the fairness and friendliness of our enhanced approach.

A. *Enhanced Start* cwnd *Evolution*

Fig. 2 shows Enhanced Start queue estimation and *cwnd* evolution. We see that the estimated queue at sender and the actual queue at router match quite well. By correctly estimating the router queue, the sender increases *cwnd* in exponential-linear cycles, allowing *cwnd* to adaptively converge to the BDP in a timely manner. Our method also prevents the temporary queue from overflow when the buffer size is small.

In Fig. 3, we vary the value of threshold, β, to study the sensitivity of our algorithm. As can be seen from the figure, surprisingly, varying the value does not cause much difference in *cwnd* evolutions. This means that the setting of β is not a mainly decisive factor in the performance. However, as adopted in [9], we set β as 3 to be the test value for the remaininng experiments.

To assess the capability of our measurement in the network with heterogenous stacks, we add a burst UDP cross-traffic set to 10 Mbps starting at the first second and stopping at the fifth second. Fig. 4 shows Enhanced Start queue estimation

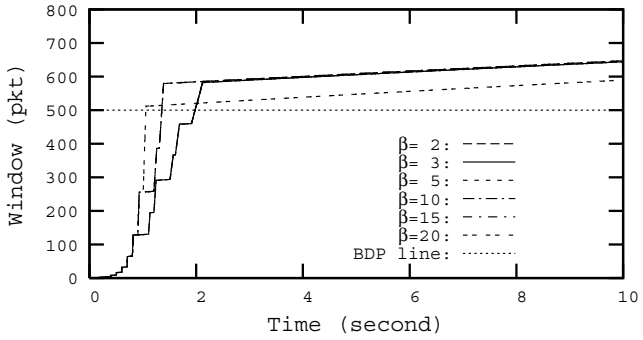

Fig. 3. Enhanced Start *cwnd* evolution under different values of β.

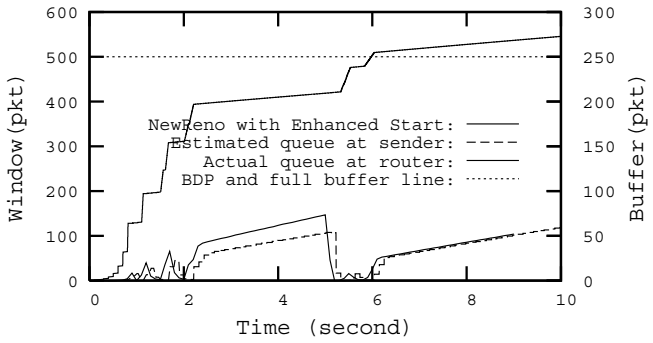

Fig. 4. Enhanced Start *cwnd* evolution under burst UDP cross-traffic (10Mbps starts at 1 sec and stops at 5 sec).

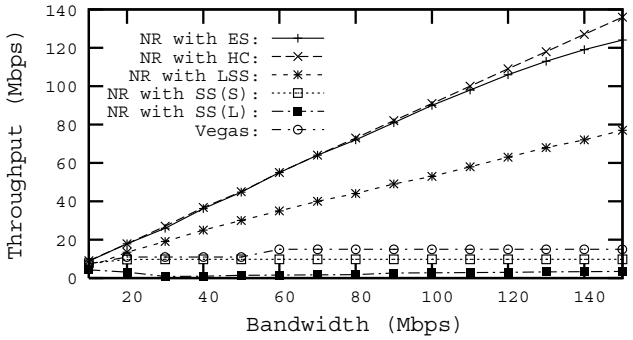

Fig. 5. NewReno (NR) throughput versus bottleneck bandwidth (first 20 s).

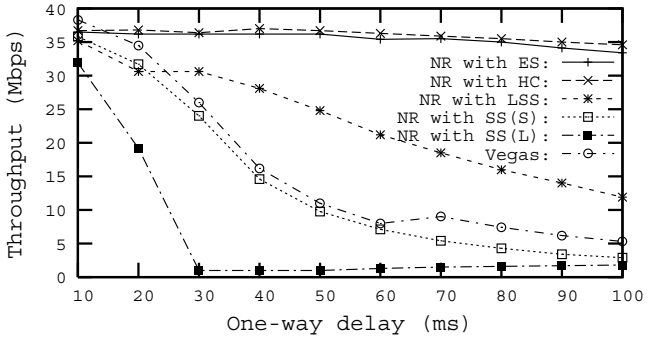

Fig. 6. NewReno (NR) throughput versus delay (first 20 s).

and *cwnd* evolution under this scenario. We see that the queue estimation is accurate in approaching the actual queue. The *cwnd* evolution reveals that during the first second, the exponential growth behavior is just as traditional slow start. After the burst of UDP traffic, the sender detects the decrease of available bandwidth quickly through the backlogged queue, and accordingly, halves *cwnd* growth rate each time congestion event happens. After reaching the spared bandwidth, *cwnd* tends to maintain its value by growing smoothly. Then, right after the termination of UDP traffic flow, *cwnd* grows exponentially again to reach the BDP swiftly. Eventually, *cwnd* seizes the BDP quite accurately. This shows that Linear Increase Phase is able to avoid buffer overflow as the sudden decrease of available bandwidth, and avoid congestion when BDP is reached. While, Adjustive Increase Phase plays the main role in speeding up *cwnd* growth again when more available bandwidth is released.

B. Throughput Performance

This subsection shows that the Enhanced Start significantly improves startup performance with regards to various bandwidth, one-way delay, and buffer size. To focus on the startup performance, we only calculate the throughput in the first 20 seconds.

Fig. 5 shows NewReno throughput with different startup algorithms under bottleneck bandwidth varying from 10 Mbps to 150 Mbps. We fix the bottleneck one-way delay to 50 ms and buffer size to BDP/2. We compare NewReno with Enhanced Start (ES), Hoe's Change (HC), Limited Slow-Start (LSS), slow-start with small *ssthresh* (32 packets), slow-start with large *ssthresh* (extremely high), and TCP Vegas. It is shown that, NewReno with Enhanced Start and Hoe's Change scale well with bandwidth. Other algorithms lack the ability to adapt to network bandwidth effectively, leading to poor throughput.

Fig. 6 shows the throughput comparison under bottleneck one-way delay varying from 10 ms to 100 ms. We fix the bandwidth to 40 Mbps and buffer size to BDP/2. The subtle changes in the throughput of NewReno with Enhanced Start and Hoe's Change shows their ability to scale well with delay, while other startup algorithms suffer from performance degradation as delay increases.

In Fig. 7, we fix the bandwidth to 40 Mbps and delay to 50 ms, and vary the buffer size from 100 packets to 300 packets. It is evident that the only desirable throughput is achieved by Enhanced Start which keeps high throughput in all test cases. Also as is shown, when the buffer size is small, Hoe's Change and Limited Slow-Start suffer severe performance degradation, while other startup algorithms fail to obtain a high throughput even with the help of increasing buffer size.

C. Enhanced Start Fairness and Friendliness

Fig. 8 shows the coexistence of multiple Enhanced Start and slow-start connections. We consider five NewReno connections, in which connections 1 and 2 are NewReno with slow-start (*ssthresh* 32 packets) and connections 3, 4, 5 are

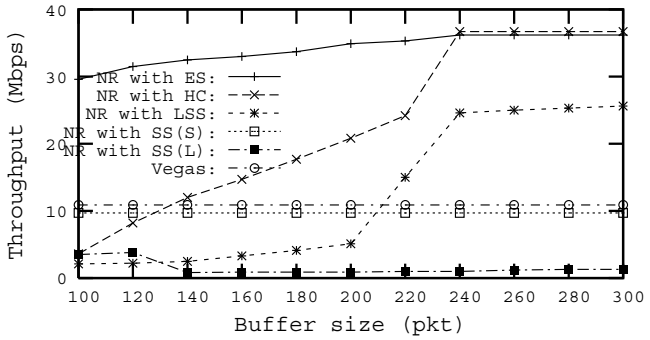

Fig. 7. NewReno (NR) throughput versus buffer size (first 20s).

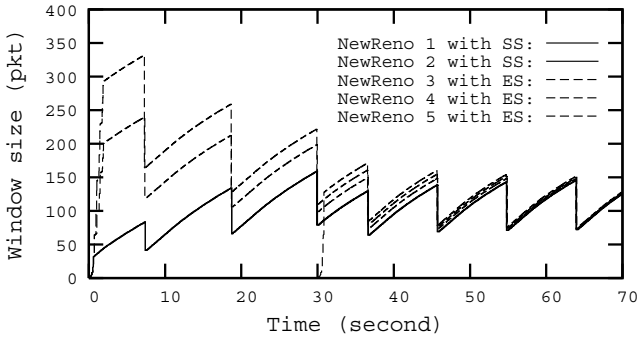

Fig. 8. Co-existence of multiple Enhanced Start and slow-start connections.

NewReno with Enhanced Start. Connections 1, 2, 3, 4 start at 0 second to investigate the effect of Enhanced Start and slow-start startup at the same time. Connection 5 starts at 30th second to estimate the effect of Enhanced Start on existing TCP connections. It is shown that, comparatively NewReno behavior is more aggressive at the very beginning, while Enhanced Start starts up to make better use of the network bandwidth left unused by other connections. As time proceeds, the window sizes of the connections incline to approach to each other. Later, the presence of connection 5 does not adversely affect coexisting TCP connections. After a burst, it joins the underway evolutions of the others. Finally, five connections converge to the same window size, which is around 100 packet sizes, one fifth of the BDP. Each link shares its own part fairly. Enhanced Start shows good fairness to connections with same stack and friendliness to connections with NewReno stack.

VI. Conclusions

In this paper, we present a sender-side enhancement to improve TCP startup performance in high-speed long-delay networks by introducing a two-phase approach in startup process. It makes use of TCP Vegas congestion-detecting scheme to monitor the router queue, and refines congestion window evolution to quickly reach the eligible window sizes, meanwhile avoids multiple packet losses. Simulation results demonstrated that it is capable of significantly improving TCP startup performance without adversely affecting coexisting TCP connections and it is robust to small buffer size and long delay. Moreover, the performance improvement is achieved by making better use of the link bandwidth, and therefore, our algorithm causes little negative effect on other co-existing connections.

References

[1] V. Jacobson, *"Congestion avoidance and control"*, in Proc. SIGCOMM'88, Stanford, CA, pp.314-329.

[2] S. Floyd, and T. henderson, *"Newreno modification to TCP's Fast Recovery"*, RFC 2582, April 1999.

[3] L. S. Brakmo, S. W. O'Malley, et al., *"TCP Vegas: New Techniques for Congestion Detection and Avoidance"*, in Proc. SIGCOMM'94, London, U. K., Oct. 1994, pp.24-35.

[4] R. Wang, K. Yamada, et al., *"TCP with Sender-Side Intelligence to Handle Evolution, Large, Leaky Pipes"*, IEEE Journal on Selected Areas in Communications. Vol.23, No.2,February 2005.

[5] J. C. Hoe, *"Improving the startup behavior of a congestion control scheme for TCP"*, in Proc. SIGCOMM'96, pp.270-280.

[6] S. Floyd, *"Limited Slow-Start for TCP with Large Congestion Windows"*, RFC 3742, March 2004.

[7] C.Casetti, M.Gerla, S.Mascolo, M.Y.Sanadidi, and R.Wang, *"TCP Westwood: bandwidth estimation for enhanced transport over wireless links"*, In Proceedings of Mobicom 2001, Rome, Italy, July 2001.

[8] Ren Wang Giowanni Pau, Kenshin Yamada, M.Y.Sanadidi, and Mario Gerla, *"Tcp startup Performance in Large Bandwidth Delay Networks"*, in Proc. INFOCOM, April 2004.

[9] C.P.Fu and S.C.Liew, *"TCP Veno: TCP enhancement for transmission over wireless access networks"*, IEEE (JSAC) Journal of selected Areas in Communications, Feb 2003.

[10] M.Aron and P.Druschel, *"TCP: Improving startup dynamics by adaptive timers and congestion control"*, Technical Report (TR98-318), Department of Computer Science, Rice University, 1998.

[11] Richard J.La, Jean Walrand, and Venkat Anantharam, *"Issues in TCP Vegas"*, Available at http://www.path.berkeley.edu/ hyongla, July 1998

[12] Lawrence S.Brakmo, and Larry L.Peterson, *"TCP Vegas: End to End Congestion Avoidance on a Global Internet"*, IEEE (JSAC) Journal of selected Areas in Communications, VOL.13, NO.8, OCTOBER 1995.

[13] G.Hengartner, J.Bolliger, and T.Gross, *"TCP Vegas revisited"*, in Proc. IEEE INFOCOM, Mar 2000, pp.1546-1555.

[14] Haining Wang and Care L. Williamson, *"A New TCP congestion control scheme: Smooth-start and dynamic recovery"*, In Proceedings of IEEE MASCOTS98, Montreal, Canada, 1998.

[15] S.Giordano, G.Procissi, F.Russo, and Raffaello Secchi, *"On the Use of Pipesize Estimators to Improve TCP Transient Behavior"*, Proccedings of the IEEE International Conference on Communications ICC2005, Vol.1,pp. 16-20, May 2005.

[16] K.Fall and K.Varadhan, *"The ns Manual"*, http://www.isi.edu/nsnam/ns/ns-documentation.html.

[17] NS. Network Simulator. http://www.isi.edu/nsnam/ns