International Journal For Technological Research In Engineering
Volume 1, Issue 12, August-2014 *ISSN (Online): 2347 - 4718*

FASTRA – SAFE AND SECURE

Christo Ananth[1], A. Ramalakshmi[2], S. Velammal[3]
B. Rajalakshmi Chmizh[4], M. Esakki Deepana[5]
[1]Assi Prof/ECE, [2, 3, 4, 5]UG Scholar/CSE, Francis Xavier Engineering College
Tirunelveli

Abstract: The innovative congestion control algorithm named FASTRA (Fast Active Stability TCP) is aimed for high-speed long-latency networks. Four major difficulties in FASTRA are highlighted at both packet and flow levels. The architecture and characterization of equilibrium and stability properties of FASTRA are robust. Experimental results of FASTRA outsmart TCP Reno, HSTCP, and STCP in terms of throughput, fairness, stability, and responsiveness. FASTRA aims to rapidly stabilize high-speed long-latency networks into steady, efficient and fair operating points, in dynamic sharing environments, and the preliminary results are produced as output of our project. The Proposed architecture is explained with the help of an existing real-time example as to explain why FASTRA download is chosen rather than FTP download. The Paper is concluded with the results of the new congestion control algorithm aided with the graphs obtained during its simulation in NS2. On proper implementation, many safe, FASTRA downloads and data transfers can be carried over a high speed internet network.

Index Terms: Congestion Control, FASTRA, HSTCP, Reno TCP, STCP

I. INTRODUCTION

The congestion control algorithm in the current TCP, which we refer to as Reno, was developed in 1988 and has gone through several enhancements since. It has performed remarkably well and is generally believed to have prevented severe congestion as the Internet scaled up by six orders of magnitude in size, speed, load, and connectivity, if is also well-known, however, that as bandwidth-delay product continues to grow, TCP Reno will eventually become a performance bottleneck itself. The following four difficulties contribute to the poor performance of TCP Reno in networks with large bandwidth-delay products: 1) at the packet level, linear increase by one packet per Round-Trip Time (RTT) is too slow, and multiplicative decrease per loss event is too drastic. 2) At the flow level, maintaining large average congestion windows requires an extremely small equilibrium loss probability.3) at the packet level, oscillation is unavoidable because TCP uses a binary congestion signal (packet loss). 4) At the flow level, the dynamics is unstable, leading to severe oscillations that can only be reduced by the accurate estimation of packet loss probability. These difficulties are explained in detail in Section II. Here delay-based approach is motivated. Delay-based congestion control has been proposed. Its advantage over loss-based approach is small at low speed, but decisive at high speed, as we will argue below. As pointed out in delay can be a poor or untimely predictor of packet loss and therefore using a delay-based algorithm to augment the basic AIMD (Additive Increase Multiplicative Decrease) algorithm of TCP Reno is the wrong approach to address the above difficulties at large windows. Instead, a new approach that fully exploits delay as a congestion measure, augmented with loss information, is needed. FASTRA uses this approach. Using queuing delay as the congestion measure has two advantages. First, queuing delay can be more accurately estimated than loss probability both because packet losses in networks with large bandwidth-delay product are rare events (probability on the order 10-8 or smaller), and because loss samples provide coarser information than) queuing delay samples. Indeed, measurements of delay are noisy, just as those of loss probability. Each measurement of packet loss (whether a packet is lost) provides one bit of information for the filtering of noise. Whereas each measurement of queuing delay provides multi-bit information, this makes it easier for the equation-based implementation to stabilize a network into a steady state with a target fairness and high utilization. Second, the dynamics of queuing delay seems to have the right scaling with respect to network capacity. This helps maintain stability as a network scales up in capacity. In Section III, the architecture of the proposed system is discussed. The architecture is laid to implement the design; Even though the discussion is in the context of FASTRA, the architecture can also serve as a general framework to guide the design of other congestion control mechanisms. Not necessarily limited to TCP, for high-speed networks. The main components in the architecture can be designed separately and upgraded asynchronously. Unlike the conventional design, FASTRA can use the same window and burstiness control algorithms regardless of whether a source is in the normal state or the loss recovery state. This leads to a clean separation of components in both functionality and code structure. We then present an overview of some of the algorithms implemented in our current prototype. A mathematical model of the window control algorithm is presented. In particular, FASTRA does not penalize flows with large propagation delays, and it achieves weighted proportional fairness. For the special case of single bottleneck link with heterogeneous flows, we prove that the window control algorithm of FASTRA is globally stable, in the absence of feedback delay. Moreover, starting from any initial state, a network converges exponentially to a unique equilibrium. In Section IV, the performance of FASTRA is compared with Reno, HSTCP (High-speed TCP, and STCP (Scalable TCP), using their default parameters. In these experiments, FASTRA achieved the best performance under

each criterion, while HSTCP and STCP improved throughput and responsiveness over Reno at the cost of fairness and stability. Section V concludes the paper with scope for future study.

II. PROBLEM STATEMENT

A. Modeling methods

The congestion avoidance algorithm of TCP Reno and its variants have the form of AIMD. The pseudo code for window adjustment is: Ack: w ← w+ (1/w), Loss: w← w-(1/w) this is a packet-level model, but it induces certain flow-level properties such as throughput, fairness, and stability. These properties can be understood with a flow-level model of the AIMD algorithm. The window of size increases by 1 packet per RTT and decreases per unit time by x_i (t) pi (t). (1/2). ($4w_i(t)/3$) packets where $x_i(t) = w_i(t)/T_i(t)$ packets/sec. $T_i(t)$ is the round-trip time and $p_i(t)$ is the (delayed) end to end loss probability, in period t. Here $4w_i$ (t)/3 is the peak window size that gives the "average" window of w_i (t). Hence a flow-level model of AIMD is:

$$w^*_i(t) = (1/T_i(t))-(2/3).x_i(t).p_i(t).w_i(t) \quad (1)$$

Setting $w_i(t)= 0$ in yields the well-known $1/\sqrt{p}$ formula for TCP Reno discovered, which relates loss probability to window size in equilibrium.

$$P^*_i = (3/ (2w^*_i)^2) \quad (2)$$

In summary (1) and (2) describe the flow-level dynamics and the equilibrium, respectively, for TCP Reno. It turns out that different variants of TCP all have the same dynamic structure at the flow level.

By defining $k_i (w_{i,} T_i) = (1/T_i)$ & $u_i (w_i, T_i) = 1.5/w_i^{2.}$ And noting that

$$w_i (t) = k (t). (1 – (p_i (t)/u_i (t)) \quad (3)$$

where we have used the shorthand $k_i(t)= k_i(w_i(t), T_i(t))$ and $u_i(t)= u_i(w_i(t), T_i(t))$. Equation (3) can be used to describe all known TCP variants, and different variants differ in their choices of the gain function k_i and the marginal utility function u_i, and whether the congestion measure p_i is loss probability or queuing delay. Next, we illustrate the equilibrium and dynamics problems of TCP Reno, at both the packet and flow levels, as bandwidth-delay product increases.

B. Problem

The equilibrium problem at the flow level is expressed in (2): the end-to-end loss probability must be exceedingly small to sustain a large window size, making the equilibrium difficult to maintain in practice, as bandwidth-delay product increases. Even though equilibrium is in flow-level notion, this problem manifests itself at the packet level, where a source increments its window too slowly and decrements it too drastically. When the peak window is 80,000-packct (corresponding to an "average" window of 60,000 packets), which is necessary to sustain 7.2Gbps using 1,500-byte packets with a RTT of l00ms, it takes 40,000 RTTs or almost 70 minutes, to recover from a single packet loss. The increment function for Reno (and for HSTCP) is almost indistinguishable from the x axis. Moreover, the gap between the increment and decrement functions grows rapidly as w_i increases. Since the average increment and decrement must be equal in equilibrium, the required loss probability can be exceedingly small at large w_i. This picture is thus simply a visualization of (2). To address the difficulties of Reno at large window sizes, HSTCP and STCP increase more aggressively and decrease more gently.

C. Motivation

The causes of the oscillatory behaviour of TCP Reno lie in its design at both the packet and flow levels. At the packet level, the choice of binary congestion signal necessarily leads to oscillation, and the parameter setting in Reno worsens the situation as bandwidth-delay product increases. At the flow level, the system dynamics given by (I) is unstable at large bandwidth-delay products.

III. ARCHITECTURE AND ALGORITHMS

The congestion control mechanism of TCP into four components in Figure 3. These four components are functionally independent so that they can be designed separately and upgraded asynchronously. In this section, we focus on the two parts that we have implemented in the current prototype.

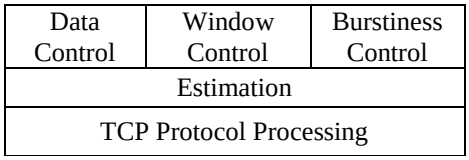

Figure 1: Architecture of FASTRA

The data control component determines which packets to transmit, window control determines how many packets to transmit, and burstiness control determines when to transmit these packets. These decisions are made based on information provided by the estimation component. Window control regulates packet transmission at the RTT timescale, while burstiness control works at a smaller timescale. In the following subsections, we provide an overview of window control and algorithms implemented in our current prototype.

D. Estimation of Input Parameters

This component provides estimations of various input parameters to the other three decision-making components. It computes two pieces of feedback information for each data packet sent. When a positive acknowledgment is received, it calculates the RTT for the corresponding data packet and updates the average queuing delay and the minimum RTT. When a negative acknowledgment (signaled by three duplicate acknowledgments or timeout) is received, it generates a loss indication for this data packet to the other components. The estimation component generates both a multi-bit queuing delay sample and a one-bit loss-or-no loss sample for each data packet. The queuing delay is smoothed by taking a moving average with the weight $\eta(t) := \min\{3w_i(t), 1/4\}$ that depends on me window $w_i(t)$ at time t as follows. The k-th RTT sample T_i (k) updates the average RTT $Ŧ_i$ (k) according to:

$$Ŧ_i (k+1) = (1 – \eta (t_k) Ŧ_i (k) + \eta (t_k) T_i (k) \quad (4)$$

where t_k is the time at which the k-th RTT sample is received. Taking $d_i(k)$ to be the minimum RTT observed so far , the average queuing delay is estimated as $q_i(k) = Ŧ_i(k) - d_i(k)$. The weight η (t) is usually much smaller than the weight (1/8) used in TCP Reno. The average RTT $Ŧ_i$ (k) attempts to track the average over one congestion window. During each RTT an entire window worth of RTT samples are received if every packet is acknowledged. Otherwise, if delayed Ack is used, the number of queuing delay samples is reduced so η (t) should be adjusted accordingly.

E. Window Control Component

The window control component determines congestion win-dow based on congestion information — queuing delay and packet loss, provided by the estimation component. A key decision in our design that departs from traditional TCP design is that the same algorithm is used for congestion window computation independent of the state of the sender. For example, in TCP Reno (without rate halving), congestion window is increased by one packet every RTT when there is no loss, and increased by one for each duplicate ack during loss recovery. In FASTRA, we would like to use the same algorithm for window computation regardless of the sender. The congestion control mechanism reacts to both queuing delay and packet loss. Under normal network conditions, FASTRA periodically updates the congestion window based on the average RTT and average queuing delay provided by the estimation component, according to (5):

$$W \leftarrow \min \{2w, (1-\gamma)\, w + \gamma\, ((\text{base RTT} / \text{RTT})\, w + a\, (w, q_{delay})\} \quad (5)$$

where γ € (0,1),base RTT is the minimum RTT observed so far, and q_{delay} is the end-to-end (average) queuing delay. In our current implementation, congestion window changes over two RTTs: it is updated in one RTT and frozen in the next. The update is spread out over the first RTT in a way such that congestion window is no more than doubled in each RTT. The function a (w, q_{delay}) is chosen to be a constant at all times. This produces linear convergence when the q_{delay} is zero. Alternatively, we can use a constant a only when qdelay is non-zero and a proportional to window. a (w, q_{delay}) = aw. In this case when q_{delay} is zero FASTRA performs multiplicative increase and grows exponentially at rate a to a neighbourhood of q_{delay} >0.

F. High Level Operations

It is important to maintain an abstraction as the code evolves. This abstraction should describe the high-level operations each component performs based on external inputs, and can serve as a road map for future TCP implementations as well as improvements to the existing implementation. Whenever a non-trivial change is required, one should first update this abstraction to ensure that the overall packet-level code would he built on a sound underlying foundation. Since TCP is an event-based protocol, our control actions should be triggered by the occurrence of various events. Hence, we need to translate our flow-level algorithms into event-based packet-level algorithms. There are four types of events that FASTRA reacts to: on the reception of an acknowledgment, after the transmission of a packet, at the end of a RTT, and for each packet loss. For each acknowledgment received, the estimation component computes the average queuing delay, and the burstiness control component determines whether packets can be injected into the network. For each packet transmitted, the estimation component records a time-stamp, and the burstiness control component updates corresponding data structures for book-keeping. At a constant time interval, which we check on the arrival of each acknowledgment, window control calculates a new window size. At the end of each RTT, burstiness reduction calculates the target throughput using the window and RTT measurements in the last RTT. Window pacing will then schedule to break up a large increment in congestion window into smaller increments over time. During loss recovery, congestion window should be continually updated based on congestion signals from the network. Upon the detection of a packet loss event, a sender determines whether to retransmit each un-acknowledged packet right away or hold off until a more appropriate time.

Each source I adapt w_i (t) periodically according to

$$w_i(t+1)=\gamma((d_i w_i(t)/(d_i+q_i(t))+a_i(w_i(t),q_i(t))+(1-\gamma)w_i(t) \quad (6)$$

Where γ € (0, 1), at time t, and a_i (w_i, $_{qi}$) is defined by:

$$a_i\,(w_{i,\,qi}) = \{a_i w_i \text{ if } q_i=0, \quad a_{i,} \text{ otherwise} \quad (7)$$

A key departure in our model from those in the literature is that we assume that a source's send rate defined as x_i (t) =w_i (t)/T_i (t), cannot exceed the through put it receives .This is justified because of self-clocking: one round-trip time after a congestion window is increased, packet transmission will be clocked at the same rate as the throughput the flow receives. A consequence of this assumption is that the link queuing delay vector, p(t), is determined implicitly by die instantaneous window size in a static manner: given wi(t)=w_i for all i, the link queuing delays $p_l(t) = p_l(t) \geq 0$ for all l are given by:

The equilibrium values of windows w^* and delays p^* of the network defined by Equations (6) & (7) can be characterized as follows. Consider the utility maximization problem.

$$\text{Max } \Sigma\, a_i \log x_i \text{ s.t. } R_x \leq c \quad (8)$$

And the following dual problem

$$\text{Min } \Sigma\, c_l p_l - \Sigma a_i \log \Sigma R_{li} p_l \quad (9)$$

G. Corollary

Suppose R has full row rank. The unique equilibrium point (w^*, p^*) of the network is defined by (6)—(8) exists and is such that $x^* = (x_i^* = w_i/ (d_i+q_i^*)$, ¥i) is the unique maximiser of (9) and $p^{*\,is}$ the unique minimiser of (10).This implies in particular that the equilibrium rate x^* is ai – weighted proportionally fair. Theorem I implies that FASTRA has the same equilibrium properties as TCP Vegas. It s throughput is given by

$$Xi = a_i / q_i \quad (10)$$

In particular it does not penalize sources with large propagation delays d_i the relation (11) also implies that in equilibrium source I maintains a_i packets in the buffers along its path. Hence the total amount off buffering in the network must be at least $\Sigma_i a_i$ packets in order to reach the equilibrium. Global stability in a general network in the presence of

feedback delay is an open problem. State-of-the-art results either prove global stability while ignoring feedback delay, or local stability in the presence of feedback delay. Our stability result is restricted to a single link in the absence of delay.

In theorem 2, suppose there is a single link with capacity c. Then the network defined by (6)-(8) is globally stable, and converges geometrically to the unique equilibrium (w^*, p*).The basic idea of the proof is to show that the iteration from w (t) to w (t + 1) defined by (6)--(8) is a CONTRACTION mapping. Hence w (t) converges geometrically to the unique equilibrium. Some properties follow from the proof of Theorem 2.

1) Starting from an initial point (w (0), p (0)) the link is fully utilized, i.e... Equality holds in (8), after a finite time.

2) The queue length is lower and upper bounded after a finite amount of time.

IV. SIMULATION RESULTS

Dummy net is configured to create paths or pipes of different delays, 50, 100, 150, and 200ms, using different destination port numbers on the receiving machine. We then created another pipe to emulate a bottleneck capacity of 800 Mbps and a buffer size of 2,000 packets, shared by all the delay pipes. Due to our need to emulate a high-speed bottleneck capacity, we increased the scheduling granularity of dummy net events. We recompiled the FreeBSD kernel so the task scheduler ran every 1ms. We also increased the size of the IP layer interrupt queue to 3000 to accommodate large bursts of packets. We instrumented both the sender and the dummy net router to capture relevant information for protocol evaluation. For each connection on the sending machine, the kernel monitor captured the congestion window, the observed base RTT, and the observed queuing delay. On the dummy net router, the kernel monitor captured the throughput at the dummy net bottleneck, the number of lost packets, and the average queue size every two seconds. We retrieved the measurement data after the completion of each experiment in order to avoid disk I/O that may have interfered with the experiment itself. We tested four TCP implementations: FASTRA, HSTCP, STCP, and Reno (Linux implementation). The FASTRA is based on Linux 2.4.20 kernel, while the rest of the TCP protocols are based on Linux 2.4.19 kernel. We ran tests and did not observe any appreciable difference between the two plain Linux kernels, and the TCP source codes of the two kernels are nearly identical. Linux TCP implementation includes all of the latest RFCs such as New Reno, SACK, D-SACK, and TCP high performance extensions.

A. Overall Evaluation

We use the output of iperf for our quantitative evaluation. Each iperf session in our experiments produced five-second averages of its throughput. This is the data rate (i.e., good put) applications such as iperf receives, and is slightly less than the bottleneck bandwidth due to IP and Ethernet packet headers. Let x_i (k) be the average throughput of flow i in the five-second period k. Most tests involved dynamic scenarios where flows joined and departed. For the definitions below, suppose the composition of flows changes in period k=1…..m, and changes again over period k = m+1 so that [1, m] is the maximum-length interval over which the- same equilibrium holds. Suppose there are n active flows in this interval, indexed by i=1……, n.

Let

$$\overline{x}_i \quad := \quad \frac{1}{m}\sum_{k=1}^{m} x_i(k) \tag{11}$$

be the average throughput of flows i over this interval. We now define our performance metrics for this interval [1, m] using these throughput measurements.

1) Throughput: The average aggregate throughput for the interval [l, m] is defined as:

$$E := \sum_{i=1}^{n} \overline{x}_i \tag{12}$$

2) Intra-protocol fairness: Jain's fairness index for the interval [1, m] is defined as:

$$F = \frac{\left(\sum_{i=1}^{n} \overline{x}_i\right)^2}{n\sum_{i=1}^{n} \overline{x}_i^2} \tag{13}$$

F € (0, 1) and F = 1 is ideal (equal sharing).

3) Stability: The stability index of flow i is the sample standard deviation normalized by the average throughput:

$$S_i \quad := \quad \frac{1}{\overline{x}_i}\sqrt{\frac{1}{m-1}\sum_{k=1}^{m}\left(x_i(k)-\overline{x}_i\right)^2} \tag{14}$$

The smaller the stability index, the lesser is the oscillation a source experiences. The stability index for interval [0, m] is the average over the n active sources:

$$S \quad := \quad \frac{1}{n}\sum_{i=1}^{n} S_i \tag{15}$$

4) Responsiveness: The responsiveness index measures the speed of convergence when the network equilibrium changes at k=1, i.e. when flows join or depart. Let x_i (k) be the running average by period k ≤ m:

$$\overline{x}_i(k) \quad := \quad \frac{1}{k}\sum_{t=1}^{k} x_i(k) \tag{16}$$

For each TCP protocol, we obtain one set of computed values for each evaluation criterion for all of our experiments. We plot the CDF (cumulative distribution function) of each set of values.

B. Real-time application

Torrent is a peer-to-peer file sharing protocol used for distributing large amounts of data. Bit Torrent is one of the most common protocols for transferring large files, and by some estimates it accounts for about 35% of all traffic on the

entire Internet. The protocol works initially when a file provider makes his file (or group of files) available to the network. This is called a seed and allows others, named peers, to connect and download the file. Each peer who downloads a part of the data makes it available to other peers to download. After the file is successfully downloaded by a peer, many continue to make the data available, becoming additional seeds. This distributed nature of Bit Torrent leads to a viral spreading of a file throughout peers. As more seeds get added, the likelihood of a successful connection increases. Relative to standard Internet hosting, this protocol of the new kind reduces the original distributor's hardware and bandwidth resource costs. It is now maintained by Cohen's company Bit Torrent, Inc. there are numerous Bit Torrent clients available for a variety of computing platforms. According to isoHunt, the total amount of shared content is currently more than 1.1 terabytes. A peer is any computer running an instance of a client. To share a file or group of files, a peer first creates a small file called a "torrent" (e.g. MyFile.torrent). This file contains metadata about the files to be shared and about the tracker, the computer that coordinates the file distribution. Peers that want to download the file must first obtain a torrent file for it, and connect to the specified tracker, which tells them from which other peers to download the pieces of the file.

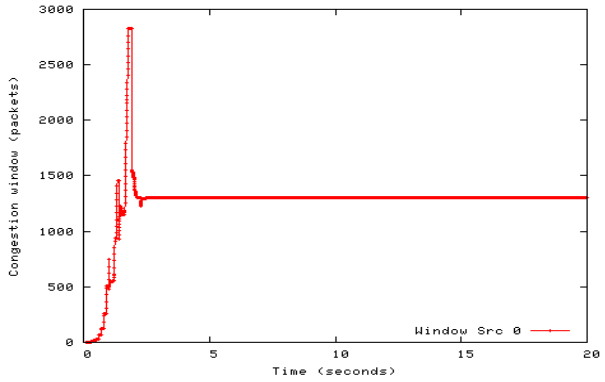

Figure 2: RTT (Seconds) of Vs Time (seconds) of FASTRA and Reno

Figure 2 shows the value of congestion window in packets w.r.t Time in seconds for FASTRA and Reno TCP. FASTRA outsmarts the performance of Reno Transmission Control Protocol in the former Graph with improved performance.

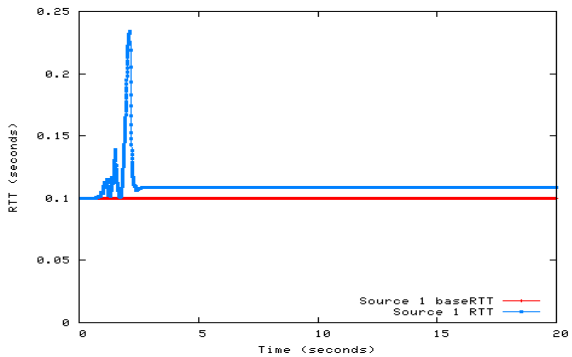

Figure 3: RTT (Seconds) Vs Time (seconds) of FASTRA and HSTCP

Figure 3 shows the value of Round Trip Time in seconds' w.r.t Time in seconds for FASTRA and HSTCP. FASTRA shows improved performance over HSTCP.

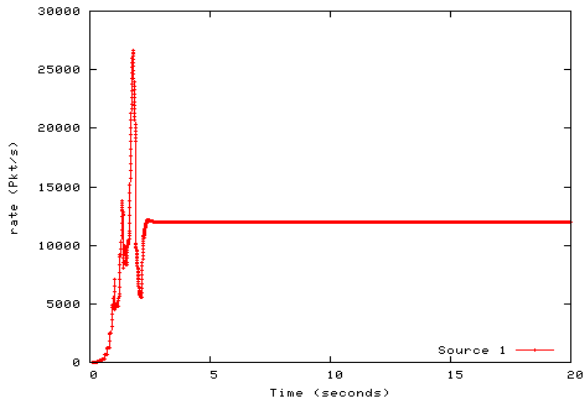

Figure 4: Rate (Packets) Vs Time (seconds) of FASTRA and STCP

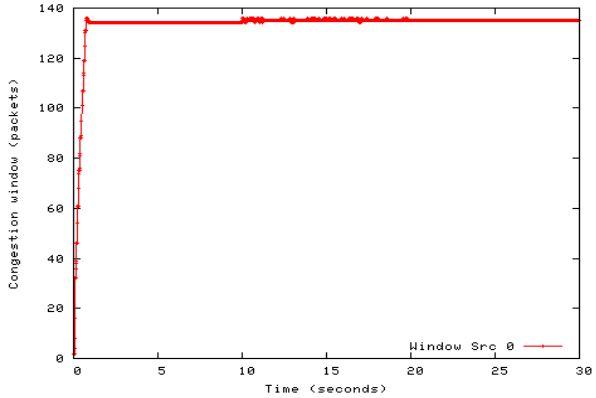

Figure 5: .Congestion window (Packets) Vs Time (seconds)

Figure 5. shows the Congestion window in Packets versus Time in seconds of the proposed method. Source 1 (FASTRA) leads other TCPs (Reno TCP, HSTCP, STCP, FTP).

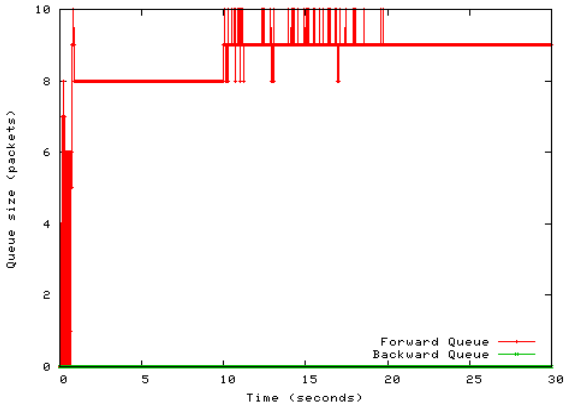

Figure 6: Queue Size (Packets) Vs Time(seconds)

Figure .6. shows the Queue size in Packets versus Time in seconds of the proposed method. Source 1 (FASTRA) leads other TCPs (Reno TCP, HSTCP, STCP) in forward queue and backward queue thus beating all other TCPs in following parameters like Round trip time, Congestion window and Queue size ensuring a safe secure downloading of information than the previous used techniques.

V. CONCLUSION AND FUTURE WORK

The Congestion control algorithm named FASTRA (Fast Active Queue Management Stability Transmission Control Protocol) is aimed for high-speed long-latency networks. Four major difficulties in FASTRA are highlighted at both packet and flow levels. The architecture and characterization of equilibrium and stability properties of FASTRA are robust. Experimental results of FASTRA outsmart TCP Reno, HSTCP, and STCP in terms of throughput, fairness, stability, and responsiveness. FASTRA aims to rapidly stabilize high-speed long-latency networks into steady, efficient and fair operating points, in dynamic sharing environments, and the preliminary results are produced as output of our project. The Proposed architecture is explained with the help of an existing real-time example as to explain why FASTRA download is chosen rather than FTP download. The Paper is concluded with the results of the new congestion control algorithm aided with the graphs obtained during its simulation in NS2. On proper implementation, many safe, FASTRA downloads and data transfers can be carried over a high speed internet network. On enhancement of the algorithm, the new algorithm holds the key for many new frontiers to be explored in case of congestion control. The congestion control algorithm is currently running on Linux platform. The Windows platform is the widely used one. By proper Simulation applications, in Windows we can implement the same congestion control algorithm for Windows platform also. The Torrents application which we are currently using can achieve speeds similar to or better than "Rapid share (premium user)" application. In our future work, we work on the implementation of the new tool which checks the effect of gold plating on the neural network systems.

REFERENCES

[1] David X. Wei and Steven H. Low, "A model for TCP model with burstiness effect," Submitted for publication, 2013.
[2] Fernando Paganini, "Scalable laws for stable network congestion control," in Proceedings of Conference on Decision and Control, December 2012, http://www.ee.ucla.edu/˜paganini.
[3] W. Feng and S. Vanichpun, "Enabling compatibility between TCP Reno and TCP Vegas," IEEE Symposium on Applications and the Internet (SAINT 2003), January 2010.
[4] Jacobson, R. Braden, and D. Borman, "TCP extensions for High performance," RFC 1323, ftp://ftp.isi.edu/in-notes/ rfc1323.txt, May 2011.
[5] Lawrence S. Brakmo and Larry L. Peterson, "TCP Vegas: end-to-end congestion avoidance on a global Internet," IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, pp. 1465–80, October 2010.

