Simulation Based Analysis of FAST TCP using OMNET++
Umair ul Hassan
04030038@lums.edu.pk

Mid Term Report

CS678 Topics in Internet Research
Spring, 2006
Introduction

Internet traffic is doubling roughly every 3 months because of rapid increase in number of users connected with Internet as well as more bandwidth demanding multimedia applications. Internet traffic patterns are not uniform during whole day which results is high times when everyone tried to send data on to network. Because Internet relies on best effort delivery protocols, congestion is unavoidable.

As TCP/IP is the de Facto standard for data transfer on Internet, numerous research efforts were directed towards congestion control in it. Because of the connectionless nature of IP, the TCP protocol becomes obvious choice for flow and congestion control. TCP uses self clocking; the mechanism of acknowledgments sent regularly by receiver [1]. This allows TCP to adjust sender to adjust its rate according to receiver's rate for flow control. Missing acknowledgments indicate congestion is network. If rate at which source gets acknowledgments decreases, the source has can not differentiate whether this is due to receiver's flow control or network congestion. So TCP uses sliding window mechanism to handle both these problems [1].

Congestion Window Control

Most of research efforts for TCP congestion control try to improve send window management. In absence of explicit feedback, two types of feedback signal are used by TCP as an indication of congestion.

Loss-based techniques consider packet loss as congestion signal. Packets are injected into the network periodically to generate packet loss. Then measurements are taken from packet losses to perform congestion control. Current Internet standard (RFC 2581) for TCP congestion control (known as TCP Reno) is a loss based congestion control technique. Reno includes mechanisms like slow start, dynamic widow sizing, fast retransmit and fast recovery for congestion window management [1].

Delay-based techniques use end-to-end packet queuing delay as measure of congestion. Packets are time stamped when they are sent, and when acknowledgments arrive the round trip time is calculated. Transmission time is subtracted from RTT to calculate queuing delay. Finally this queuing delay is used to adjust congestion window accordingly. TCP Vegas [6] was the first variation for TCP congestion control algorithm which uses delay based approach. It modifies the slow start method of TCP Reno and uses a new retransmission mechanism. TCP Vegas tries to avoid congestion by minimizing the difference between expected and actual congestion window.

FAST TCP

A congestion control algorithm can be thought of as working on two levels from design perspective. Flow level design decisions require high utilization, low delays and loss, stability and fairness [2]. While packet level design involves strategies to implement flow levels goals on end to end flows.
Current mostly widely used variation of TCP is Reno, which uses packet loss as congestion signal. As bandwidth-delay product increases for high speed networks this algorithms become bottleneck. According to [2] congestion window management in TCP Reno faces problems at both packet and flow level. The additive increase per round trip time is very slow and multiplicative decease is very fast for large congestion windows. Also at packet level one bit congestion signal (packet loss) causes severe oscillations in window size. At flow level the dynamics is unstable and very small loss probability is required to maintain large congestion windows. As a result loss-based algorithms do not scale properly according to increase in bandwidth.

FAST TCP proposes a delay based technique augmented with loss information to eliminate these problems [2]. Because it uses queuing delay (multi-bit information) as congestion signals, the algorithm can also choose not to change congestion window. It helps to stabilize congestion window even at large window sizes. Also it does not require gradual injection for packets for generation of packet loss event for measurements.

**Architecture and Algorithms**

The proposed architecture for FAST TCP breaks it into four main components [2]. The data control component is responsible for determining which packet to transmit; this involves transmission new packets according to logical sequence of data or retransmission in case of loss. Estimation component provides measurements like average and minimum round trip time and queuing delay to other components. Congestion window size is adjusted by Window Control component according to information provided by estimation component. Burstiness Control component is used to avoid long queues and large number of packet losses due to bursty data transmission in high speed networks.

FAST TCP works at two levels of time scale; per data send/ACK and per round trip time. The Coarse Tuning Algorithm given below is run per send/ACK [7]
For each source $i$:

1. Initialization: $count_i = w_i$ and $fastOn_i = 1$
2. On arrival of each ACK of packet $l$
   (a) $RTT_i(l) = \text{CurrentTime} - \text{TimeStamp}_i(l)$
   (b) $\rho = \min \left( \frac{3}{w_i}, \frac{1}{8} \right)$
   (c) $\text{ARTT}_i(l) = (1 - \rho) \text{ARTT}_i(l) - (\rho) RTT_i(l)$
   (d) If $fastOn_i = 1$:
      i. Calculate $\Delta w = \gamma \left[ \frac{w_i(l)}{\text{ARTT}_i(l)} d_i + \alpha - w_i \right]$
      ii. If $\Delta w_i > \beta$: $w_i = w_i + 1$
      iii. If $\Delta w_i < \beta$: $w_i = w_i - 1$
   (e) $count_i = count_i - 1$
   (f) If $count_i \leq 0$ (One RTT is finished)
      i. $fastOn_i = 1 - fastOn_i$
      ii. $count_i = w_i$
      iii. Call Fine Tuning Algorithm

---

**Algorithm 1: Coarse Tuning Algorithm [7]**

The Fine Tuning Algorithm is run every round trip time [7]

For each source $i$, on completion of each RTT

1. Calculate $\Delta w = \gamma \left[ \frac{w_i(l)}{\text{ARTT}_i(l)} d_i + \alpha - w_i \right]$
2. If $\Delta w_i \geq 1$: $w_i = w_i + 1$
3. If $\Delta w_i \leq -1$: $w_i = w_i - 1$

---

**Algorithm 2: Fine Tuning Algorithm [7]**

Where $\text{ARTT}(l)$ is average round trip time for packet $l$ and $w$ congestion window.
In case of packet loss current FAST TCP implementation enter fast recovery stage of TCP Reno [2]. And after recovery from congestion FAST TCP is again turned on.

**Simulations**

OMNET++ is discrete event simulation package, now widely used for network simulations. I have implemented FAST TCP according to above given algorithm is
OMNET++. As part of this project I will compare my implementation with the NS-2 based implementation of FAST TCP available of Internet. The five test cases with different network topologies and configurations used for evaluation of NS-2 implementation are [8]

- Test 1: Multiple source sharing one link
- Test 2: Multiple source sharing multiple link
- Test 3: Low buffer size
- Test 4: High bandwidth delay product
- Test 5: Random loss

I have tested my code for Test 1, where 3 follow were setup between three source and destination pairs.
- Flow 1 from between s1 and d1 was for time 0s-100s
- Flow 2 from between s2 and d2 was for time 20s-80s
- Flow 3 from between s3 and d3 was for time 40s-60s

The topology is shown in following figure

![Test 1 topology](image)

Figure 1: Test 1 topology

The difference between TCP Reno and TCP Fast was the initial exponential increase during slow start for TCP Reno, but overall trend was same for both variations. Since the was no packet loss so there is no indication of congestion window changes in case of loss recovery. These results are very different form the results of shown by NS-2, simulation parameters and TCP parameters difference can be reasons for this difference. The congestion window size change according to time is given below
Figure 2: TCP Reno results

Figure 3: TCP Fast results
References


