Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks

Kun Tan, Jingmin song, Qian Zhang, Murari Sridharan

Presented By

Shams Feyzabadi

Introduction

- As Internet evolves the number of Long distance and High speed networks grows
- Standard TCPs use Additive Increase Multiplicative Decrease (AIMD) and increase their congestion window size slowly
- Example: Bandwidth 10Gbps, RTT 100ms, Packet size 1250 bytes, takes 10,000 seconds to fully utilize

Introduction (Cont.)

- TCP-friendliness: The protocol should not reduce the performance of other regular TCP (Reno) flows competing on the same path
- For utilizing full network capacity, the protocol should be agressive, and TCP friendly

Two Types of Congestion Control

- Loss Based Approaches: Use Packet Loss as the only indication of Congestion like BIC and HSTCP
- Delay Based Approaches: Use increase in Round-Trip Time(RTT) as the indication of Congestion like FAST TCP
- In a race for using the rescources, Delay-Based approaches are losers

Related Works (Loss Based)

- STCP alters AIMD, to MIMD
- Increases cwnd by 0.01 MSS on every received ACK and reduces cwnd to 0.875 times upon a packet loss.
- HSTCP mimics the AIMD approach with some parameters, increase between 38 to 83,333 and decrease between 0.5 to 0.1 while the increase parameter increases accordingly
- STCP is more aggressive than HSTCP

Related Works (Delay Based)

- FAST
  - Core idea: The increase of RTT is considered as early indicator of congestion, and the sending rate is reduced to avoid self induced buffer overflow
  - It will not cause large queuing delay and reduce packet losses.
  - It is a scalable version of Vegas
  - Inreases multiplicatively, if the buffer occupancy $< \alpha$
  - Inreases additively, if the buffer occupancy $> \alpha$
  - Reduces the sending rate if delay increases

Compound TCP

- Objective:Satisfy efficiency and TCP friendliness requirement simultaneously
- Idea:
  - Like delay-based approaches, decrease the speed when RTT is increased
  - Like HSTCP, being very aggressive to reach network network maximum utilization.
- A scaleable, delay-based TCP congestion Control
- Default in Windows Vista

Algorithm

- It uses both delay-based and loss-based approaches
- A new state variable is defined as dwnd (controlling delay-based)
- The conventional cwnd remains untouched (controlling loss-based)
- Advertised Congestion Window awnd
- The window size is computed as

$$\text{win} = \mathit{min}(\text{cwnd} + \text{dwnd}, \text{awnd})$$

Algorithm (cont.)

- The increament of cwnd is like reno, AIMD. One MSS in each RTT.
- The inreament of dwnd is discussed later
- Slow start behaviour doesn't change
- Initially dwnd is set to zero
- The delay-based part is active only when the connection is working in congestion avoidance mode

Delay-Based Congestion Avoidance

- Derived from TCP Vegas
- *baseRTT* an estimation of transmission delay of a packet
- During the connection *baseRTT* is updated by the minimum observed RTT
- Expected = win / baseRTT
- Actual = win / RTT
- Diff = (Expected – Actual). baseRTT

Delay-Based Congestion Avoidance(cont.)

- *Diff* means the number of packets in the queue
- Threshold γ is defined
- If $diff < \gamma$, the network is underutilized, so the sending speed should increase
- Else, the queue is long and the sending speed should decrease

Formulas

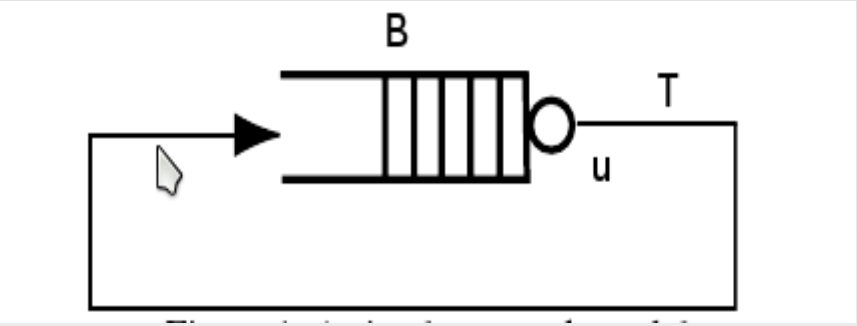
$$dwnd(t+1)=\begin{cases} dwnd(t)+(\alpha\cdot win(t)^{k}-1)^{+}, \text{if } diff<\gamma \\ (dwnd(t)-\zeta\cdot diff)^{+}, \text{if } diff\geq\gamma \\ (win(t)\cdot(1-\beta)-cwnd/2)^{+}, \text{if loss is detected} \end{cases}$$

- Where (.)+ means max(.,0)
- Parameters α, β and k are tunable ones to reach scalability

Packet Loss

- Loss is detected by three duplicate ACKs
- If retransmition time out occurs, *dwnd* should be set to zero
- After a time-out, it will go to slow-start state
- Delay-based approach is disabled until it goes to congestion avoidance part
- If the window size is less than a threshold, the delay-based part will be disabled, and it works as standard TCP

Simple Network



Simple Network:

B is the size of buffer

uT is the Bandwidth Delay Product (BDP)

Gamma auto-tuning

- Gamma can have a fixed value
- It is also possible to compute it
- Having m flows at the same time, $\gamma > B/m$ to have something in the queue

Gamma auto-tuning (cont.)

- So Gamma will be as follows

$$\gamma = \max(\gamma_{\min}, W_{low} \cdot \frac{\kappa}{1+\kappa})$$

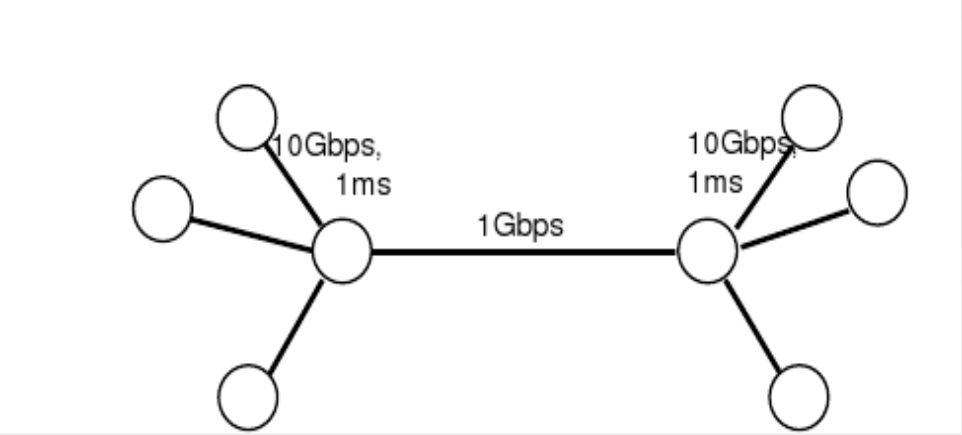
- Where K is

$$\kappa = \frac{R_{\max} - R_{\min}}{R_{\min}}$$

- R_{min} is the minimum RTT and R_{max} is the maximum RTT observed

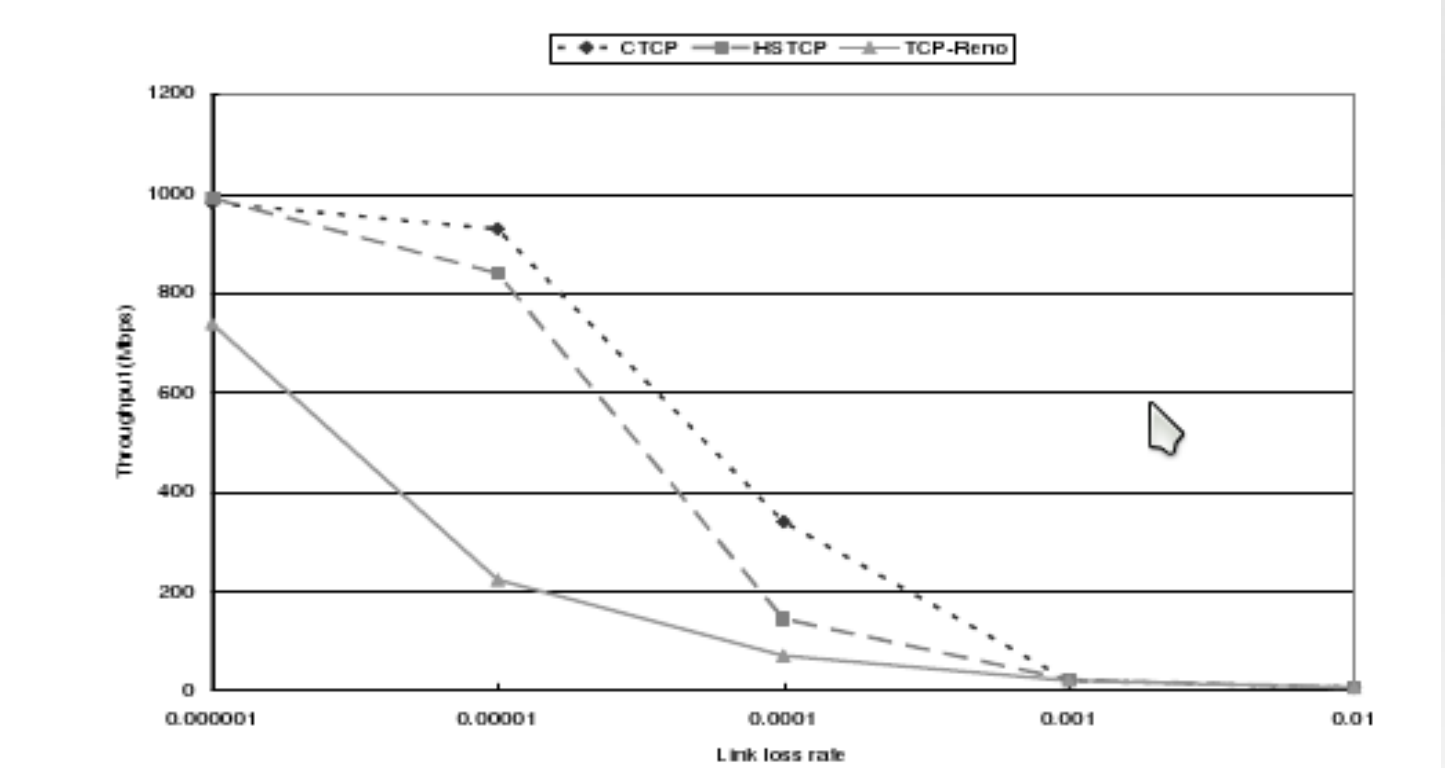
Performance Evaluation

- The following topology was used in the experiments



They used NS-2, run for 150s, γ_{min} is 3 packets, W_{low} is 41

Throughput

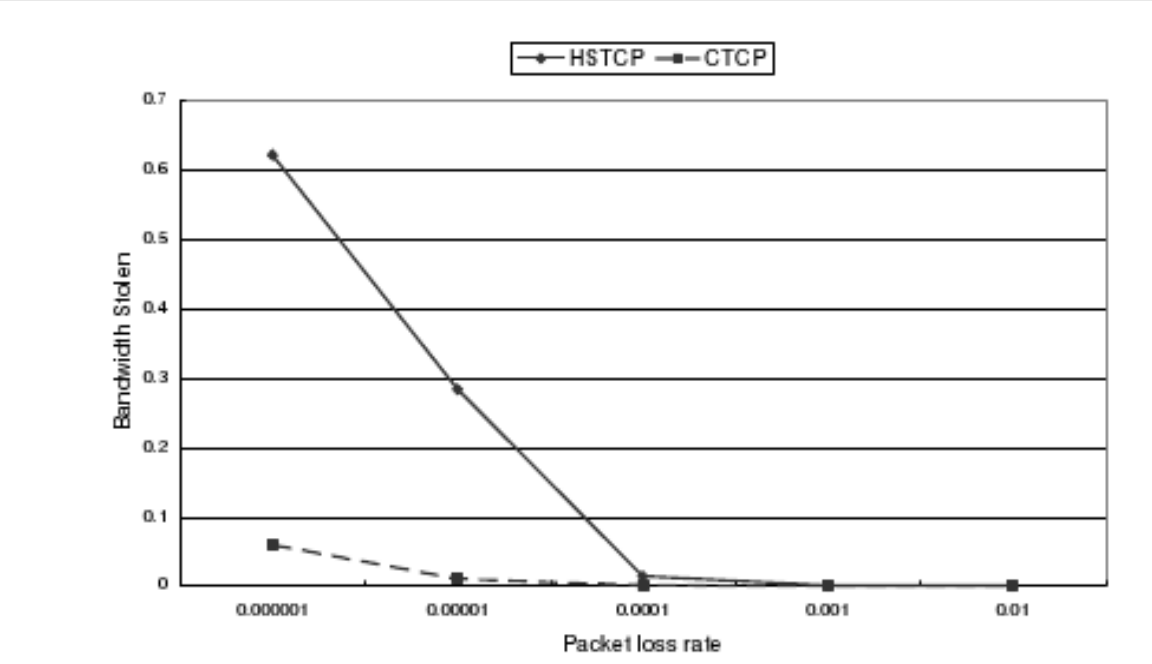


RTT is 100ms, buffer size is 1500 packets, packet loss rate is from 0.01 to 0.000001

TCP Fairness

- *bandwidth-stolen*: a new definition
- B = (P-Q)/P
- Where P is aggregated throughput of m regular TCP flows when they compete with l other R**egular** TCP flows
- And Q is aggregated throughput of m regular TCP flows when they compete with l other **High-Speed** TCP flows

Bandwidth Stolen



Baseline is the average throughput of 8 reglar TCP flows

Test is 4 regular versus 4 high-speed TCP flows

Reverse Traffic

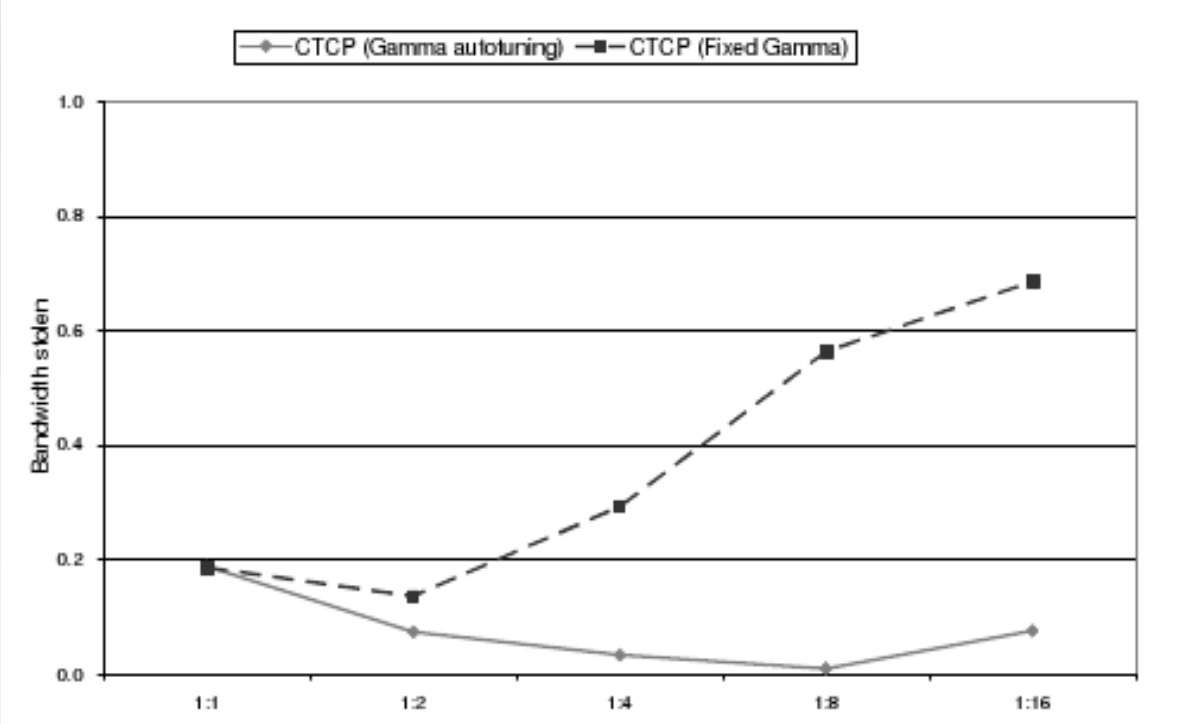
- RTT is 30ms
- Forward and backward bottleneck are 750 packets
- One forward flow and different number of reverse regular flows were on the link

Reverese Traffic

RF #	HSTCP			CTCP			Regular TCP		
	FW	R	Sum	FW	R	Sum	FW	R	Sum
1	818	338	1156	557	496	1053	491	531	1022
2	705	430	1136	397	662	1059	357	664	1021
4	653	442	1096	307	842	1133	291	827	1134
8	648	437	1085	272	850	1121	243	876	1119
16	480	619	1099	300	898	1198	271	900	1170

Throughput of different flows in Mbps

Bandwidth Stolen Using Auto-tuning



Thanks A Lot

Questions ???