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

Kun Tan, **Jingmin Song**, Qian Zhang, Murari Sridharan

Microsoft Research Asia

Outline

- Motivation
- The design of Compound TCP
- Evaluation
  - Simulation results
  - Production network testing
- Conclusions

Motivation

- The protocol design requirements for high-speed are mainly two things:
  - Efficiency – effectively utilize the high-speed link even with large delay
  - TCP fairness – be able to be progressively deployed

It is easy to meet efficiency requirement, but it is difficult to be both efficient and TCP fairness

Existing protocols

- Loss-based
  - HSTCP, STCP, BIC -> aggressive
  - Cause self-induced packet losses -> TCP unfairness
- Delay-based
  - FAST
  - React to RTT increase to avoid self-induced loss
  - Not competitive to loss-based protocols

How about combine these two classes together?

The Compound TCP

- A synergy of both delay-based approach and loss-based approach
- Two components
  - A loss-based component
    - The standard TCP Reno, provide base-line perf
  - A scalable delay-based component
    - Aggressively obtain bandwidth if the link is under-utilized
    - Gracefully retreat if the queue is built

Realization

- Two window state variables
  - *cwnd* – Congest window
    dwnd – Delay window
  - *win* = min(*cwnd* + *dwnd, awnd*)

- *cwnd* updated as standard Reno
  - *cwnd* = *cwnd* + *1/win* – upon an ACK
  - *cwnd* = *cwnd / 2* – upon a loss

Design of delay component

- Scalable
  - The overall CTCP window evolves binomially
- Reduce on detecting queue on the link
  - By sensing backlogged packets with the RTT increases
- React to loss efficiently
  - Multiplicatively reducing window

Delay window control

- Calculate diff (backlogged pkts) samely as in TCP Vegas:

$$Expected = win / baseRTT$$
$$Actual = win / RTT$$
$$Diff = (Expected - Actual) \cdot baseRTT$$

- Control functions:

$$dwnd(t+1) = \begin{cases} dwnd(t) + (a \cdot win(t)^k - 1)^+, \text{if } diff < g \\ (dwnd(t) - z \cdot diff)^+, \text{if } diff \geq g \\ (win(t) \cdot (1-b) - cwnd/2)^+, \text{if loss is detected} \end{cases}$$

Parameters setting

- Set directly
  - $z = 1$, and $b = 1/2$
- Set by Comparing Aggressiveness with HSTCP

CTCP	HSTCP
$W_{CTCP} \propto \frac{1}{p^{\frac{1}{2-k}}}$	$W_{HSTCP} \propto \frac{1}{p^{0.833}}$

  - $k = 0.75$, $a = 1/8$

Parameters setting (cont.)

- Fixed Gamma value
  - A tradeoff between efficiency and TCP fairness
- *Auto-tuning Gamma algorithm* – to dynamically select *gamma,* based on link configuration
  - Conditions for ineffective of gamma settings for early congestion detection

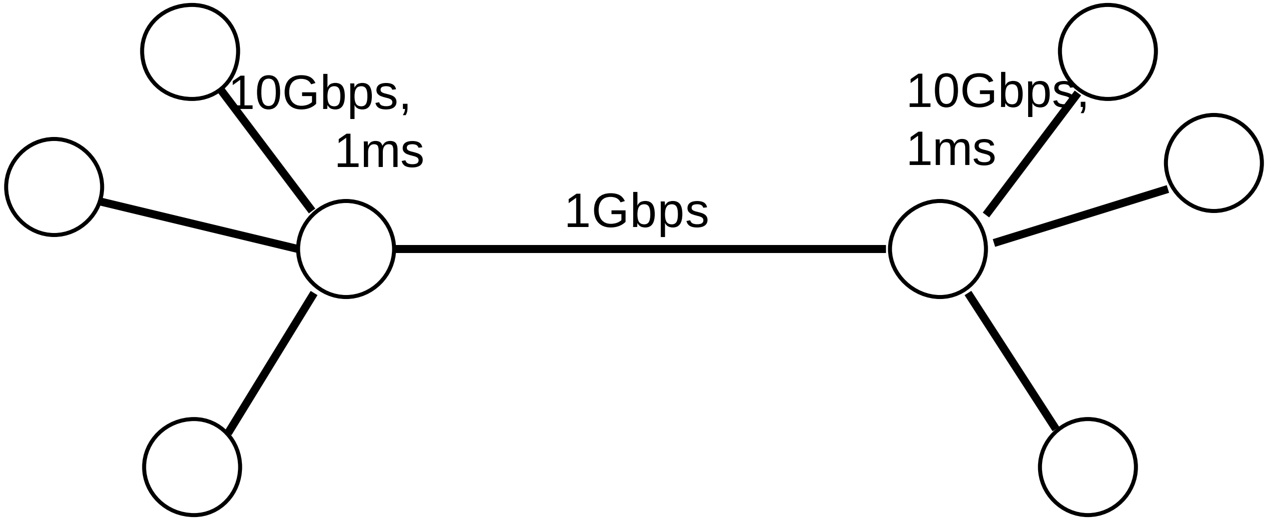
1) $W_{low} < \frac{B + uT}{m}$ and 2) $g > \frac{B}{m}$ ➔ 3) $g > W_{low} \cdot \frac{k}{1+k}$

where $k = B / uT \approx \frac{Rtt_{\max} - Rtt_{\min}}{Rtt_{\min}}$

- Choosing *gamma* as $g = \max(g_{\min}, W_{low} \cdot \frac{k}{1+k})$

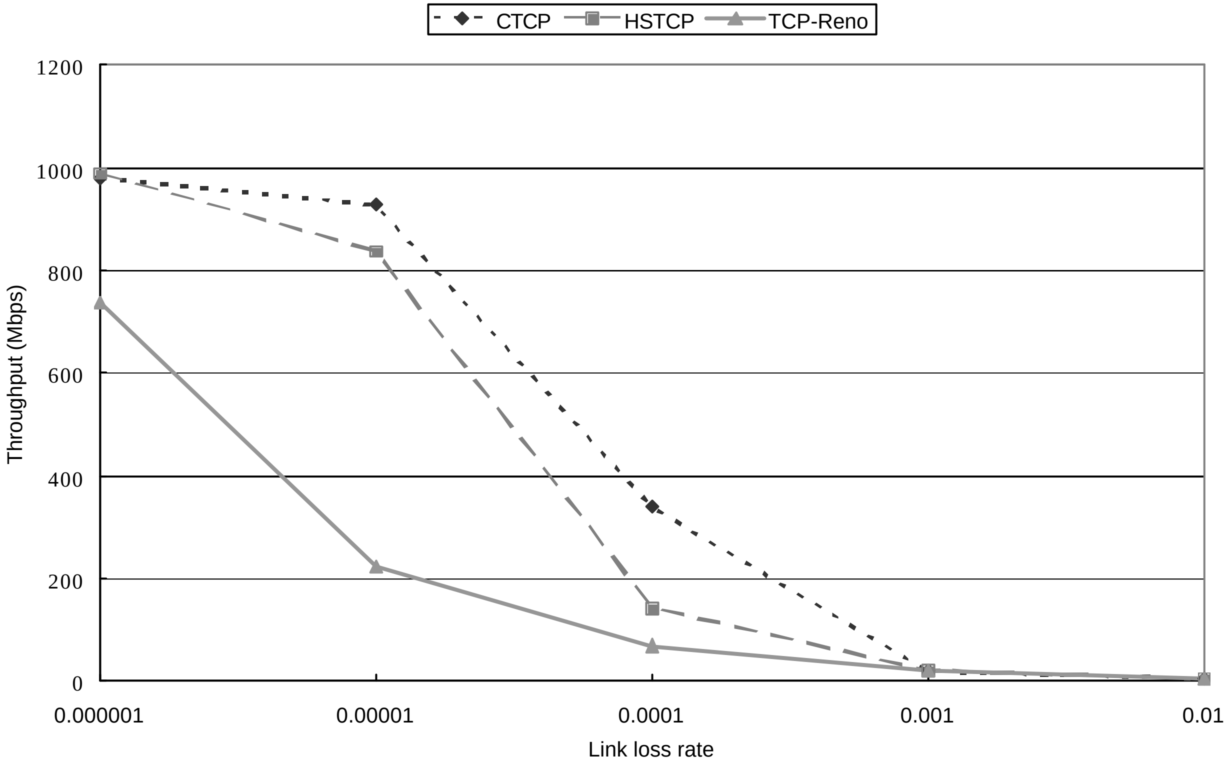
Simulation

- NS 2
- Dumbbell topology



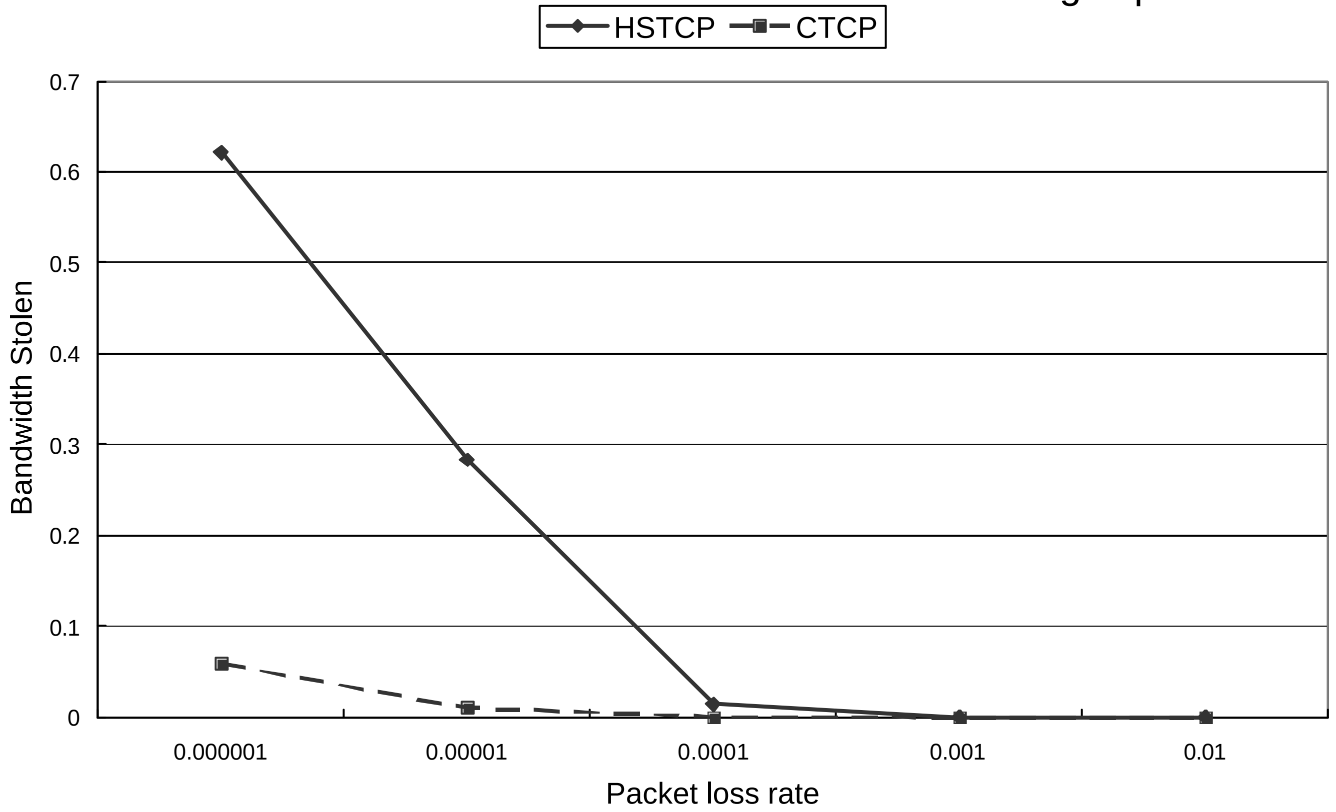
Results – random link loss(1)

- Speed 1Gbps, RTT = 100ms
- Buffer = 1500 packets
- Aggregated throughput of 4 flows



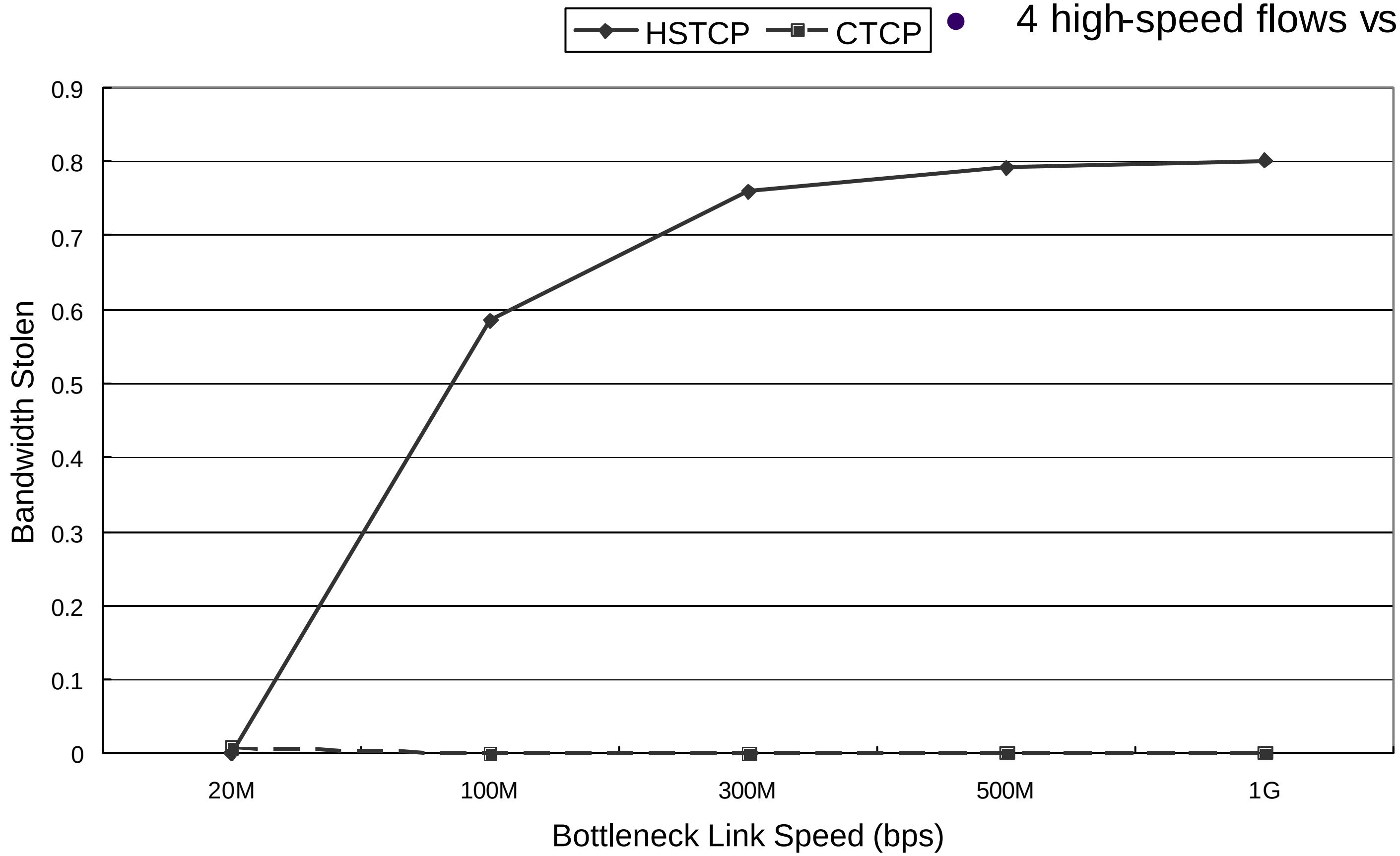
Results – random link loss(2)

- Speed 1Gbps, RTT = 100ms
- Buffer = 1500 packets
- 4 high-speed flows vs 4 TCP NewReno



Results – various link speed

- RTT = 100ms
- Buffer = BDP of the link
- 4 high-speed flows vs 4 TCP NewReno



Results - reverse traffic

- Symmetric link
- 1 forward high-speed flow and *n* reverse TCP NewReno flow
- Speed = 1Gbps, RTT=30ms
- Buffer = 750 packets

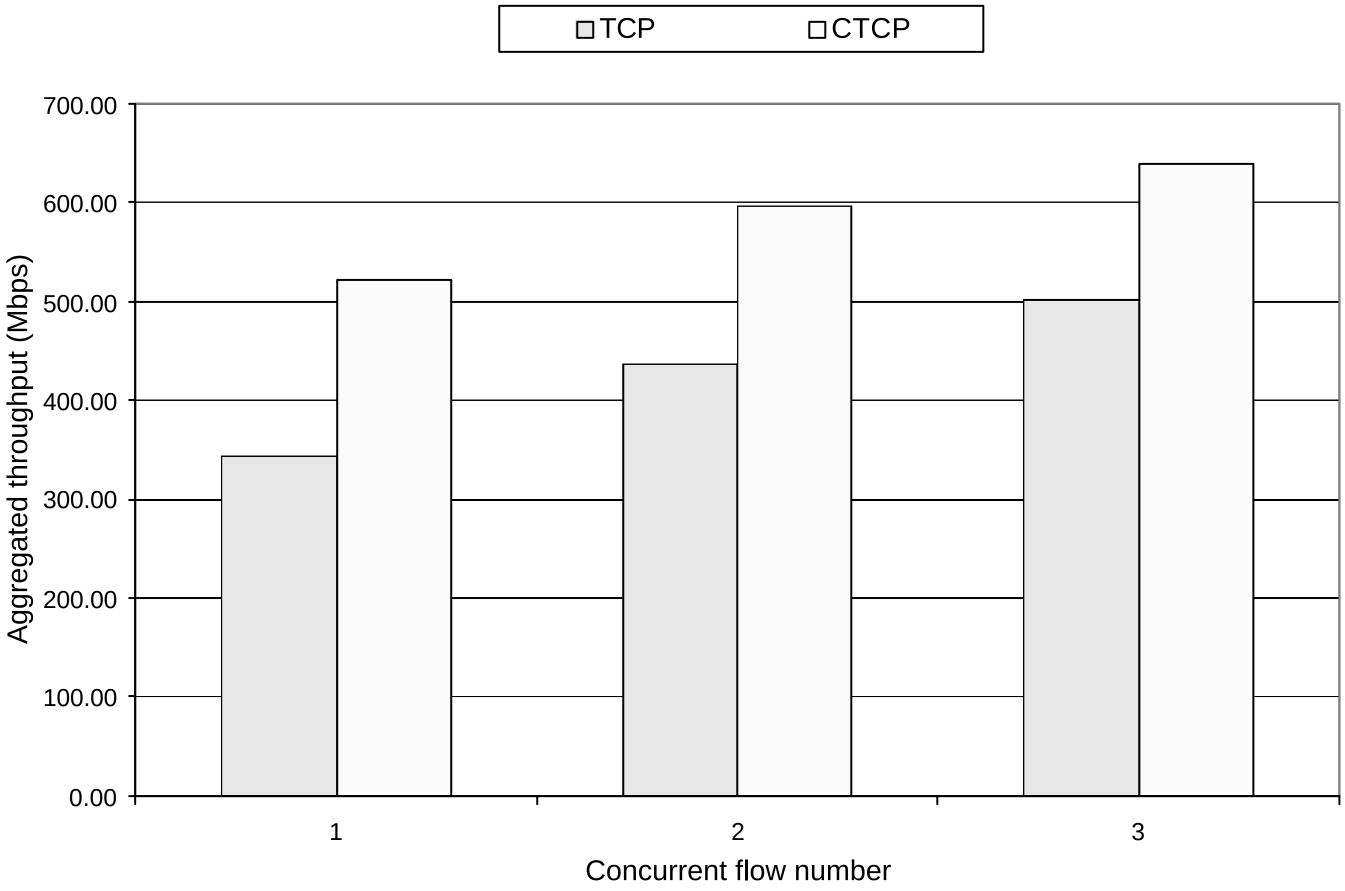
RF #	HSTCP			CTCP			New Reno		
	FW	R	Sum	FW	R	Sum	FW	R	Sum
1	818	338	1156	557	496	1053	491	531	1022
2	705	430	1136	397	662	1059	357	664	1021
4	653	442	1096	307	842	1133	291	827	1134
8	648	437	1085	272	850	1121	243	876	1119
16	480	619	1099	300	898	1198	271	900	1170

- *CTCP improves throughput regarding NewReno*
- *A tradeoff between forward and backward traffic*

Testing on MS production network

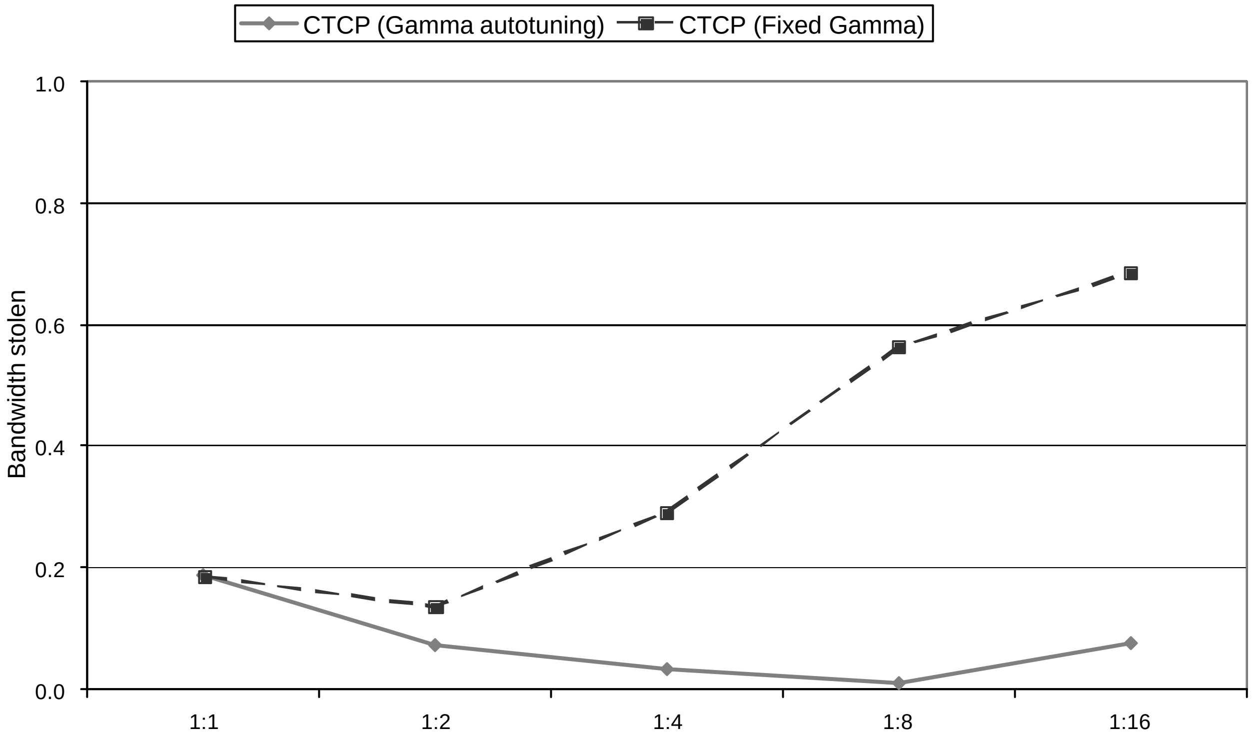
- MS high-speed intranet: Tukwila -> San Francisco
- Speed 1 Gbps, RTT = 30ms
- Light-loaded background traffic
- Low-buffer provision
- Windows implementation of CTCP

Results: throughput



Results: TCP fairness

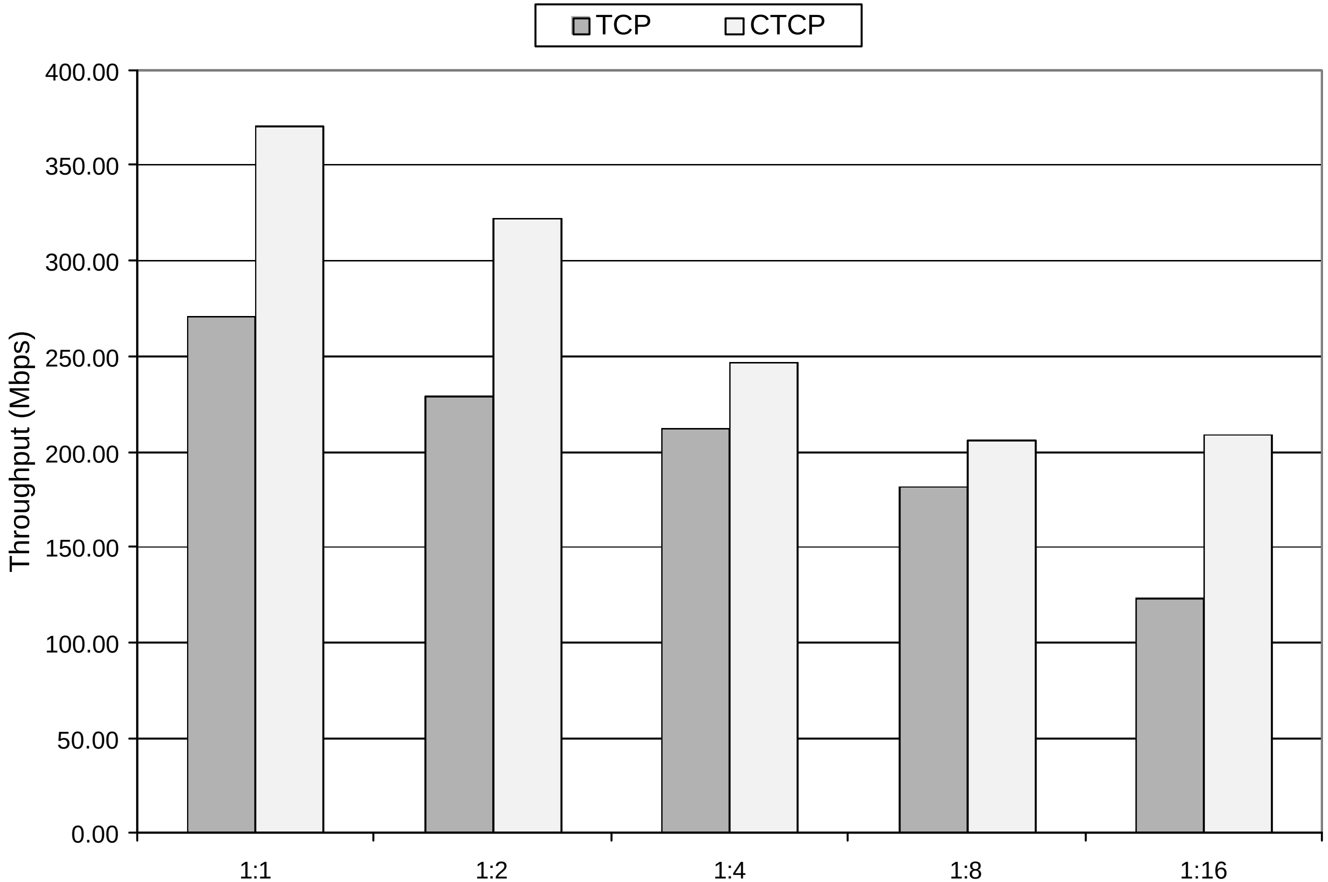
- 1 TCP vs. *n* CTCP



- *Fixed gamma CTCP steal more bandwidth from NewReno with the increase of flow number*

Results: reverse traffic

- 1 forward flow vs. *n* reverse flows



Conclusion

- CTCP is a synergy of loss-based and delay-based approach
- Effectively use the high-speed link bandwidth
- Maintain good TCP fairness
- Promising to safely progressively deploy

Thank you!

Questions?

Results – various buffer size

- Speed = 1Gbps, RTT = 100ms
- 4 high-speed flows vs 4 TCP NewReno