# FAST TCP

- Design, architecture, algorithms
- Experimental evaluations

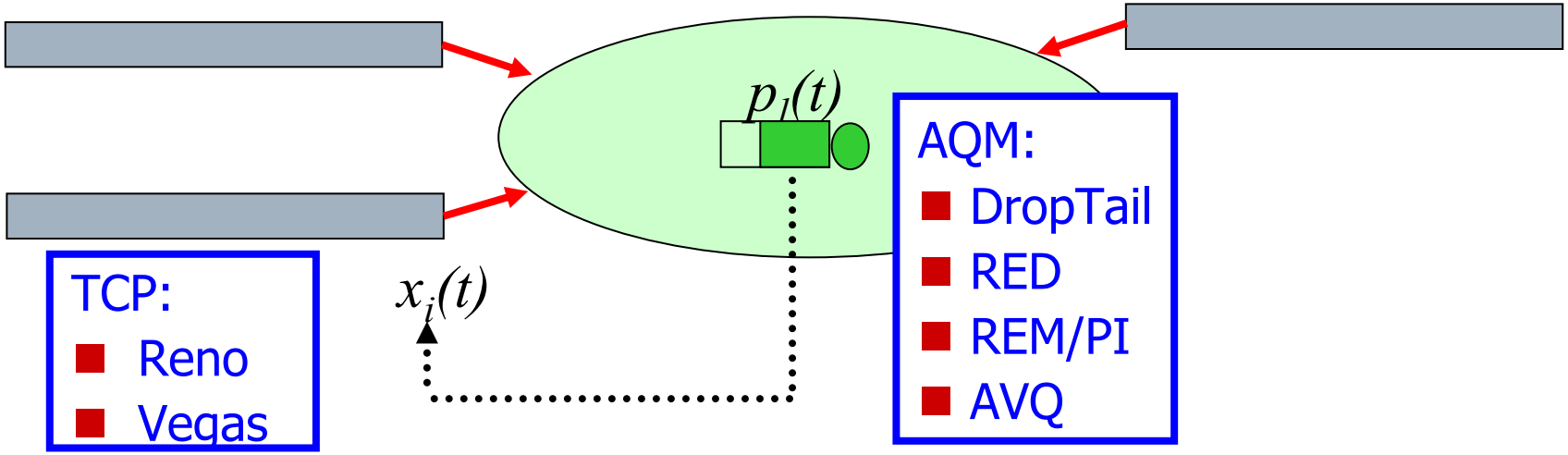
Steven Low
netlab.CALTECH.edu

# Acks & Collaborators

- Caltech
  - Bunn, Choe, Doyle, Hegde, Jin, Li, Low Newman, Papadoupoulous, Ravot, Singh, Tang, J. Wang, Wei, Wydrowski, Xia
- UCLA
  - Paganini, Z. Wang
- StarLight
  - deFanti, Winkler
- CERN
  - Martin
- SLAC
  - Cottrell
- PSC
  - Mathis
- Internet2
  - Almes, Shalunov
- Abilene GigaPoP's
  - GATech, NCSU, PSC, Seattle, Washington
- Cisco
  - Aiken, Doraiswami, McGugan, Smith, Yip
- Level(3)
  - Fernes
- LANL
  - Wu



# TCP/AQM




- Congestion control is a distributed asynchronous algorithm to share bandwidth
- It has two components
  - TCP: adapts sending rate (window) to congestion
  - AQM: adjusts & feeds back congestion information
- They form a distributed feedback control system
  - Equilibrium & stability depends on both TCP and AQM
  - And on delay, capacity, routing, #connections

# Packet & flow level

## Reno TCP

- Packet level

```
ACK:  W  ←  W + 1/W
Loss: W  ←  W – 0.5W
```

- Flow level
  - Equilibrium $$x_i T_i = \frac{\sqrt{1.5}}{\sqrt{p_i}}$$ pkts (Mathis formula)
  - Dynamics $$\dot{w}_i(t) = \frac{1}{T_i}\left(1 - \frac{2}{3} \cdot w_i^2(t) p_i(t)\right)$$

# Reno TCP

- Packet level
  - Designed and implemented first
- Flow level
  - Understood afterwards
- Flow level dynamics determines
  - Equilibrium: performance, fairness
  - Stability

- Design flow level equilibrium & stability
- Implement flow level goals at packet level

# Reno TCP

- Packet level
  - Designed and implemented first
- Flow level
  - Understood afterwards
- Flow level dynamics determines
  - Equilibrium: performance, fairness
  - Stability

Packet level design of FAST, HSTCP, STCP guided by flow level properties

# Flow level: Reno, HSTCP, STCP, FAST

- **Similar** flow level equilibrium

$$\text{Reno} \qquad x_i = \frac{1}{T_i} \cdot \frac{\alpha}{p_i^{0.5}} \qquad \text{pkts/sec}$$

$$\text{HSTCP} \qquad x_i = \frac{1}{T_i} \cdot \frac{\alpha}{p_i^{0.84}}$$

$$\text{STCP} \qquad x_i = \frac{1}{T_i} \cdot \frac{\alpha}{p_i}$$

$$\text{FAST} \qquad x_i = \frac{\alpha}{p_i}$$

$\alpha$ = 1.225 (Reno), 0.120 (HSTCP), 0.075 (STCP)

# Flow level: Reno, HSTCP, STCP, FAST

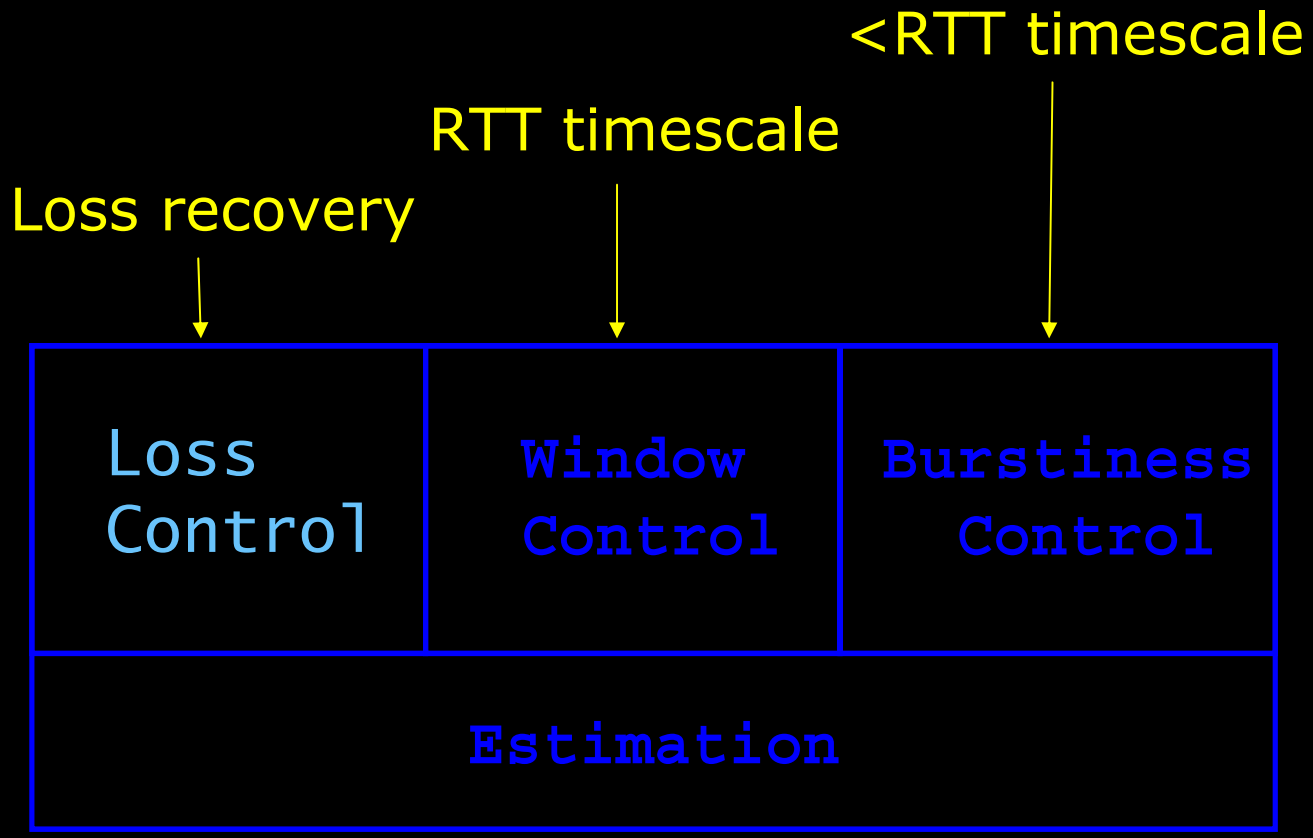
- **Common** flow level dynamics!

$$\dot{w}_i(t) = \kappa(t) \cdot \left(1 - \frac{p_i(t)}{U_i'(t)}\right)$$

window adjustment = control gain × flow level goal

- **Different** gain $\kappa$ and utility $U_i$
  - They determine equilibrium and stability
- **Different** congestion measure $p_i$
  - Loss probability (Reno, HSTCP, STCP)
  - Queueing delay (Vegas, FAST)

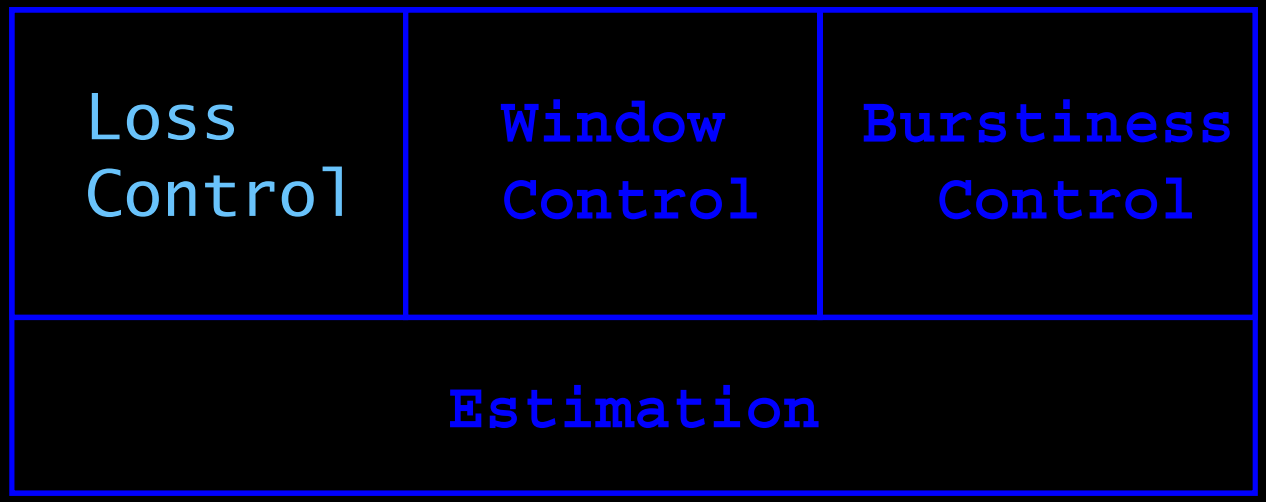
# FAST Architecture



# FAST Architecture

Each component

- designed independently
- upgraded asynchronously



# Window control algorithm

$$w \leftarrow w \cdot \frac{\text{baseRTT}}{\text{RTT}} + \alpha$$

- Full utilization
  - regardless of bandwidth-delay product
- Globally stable
  - exponential convergence
- Fairness
  - weighted proportional fairness
  - parameter $\alpha$

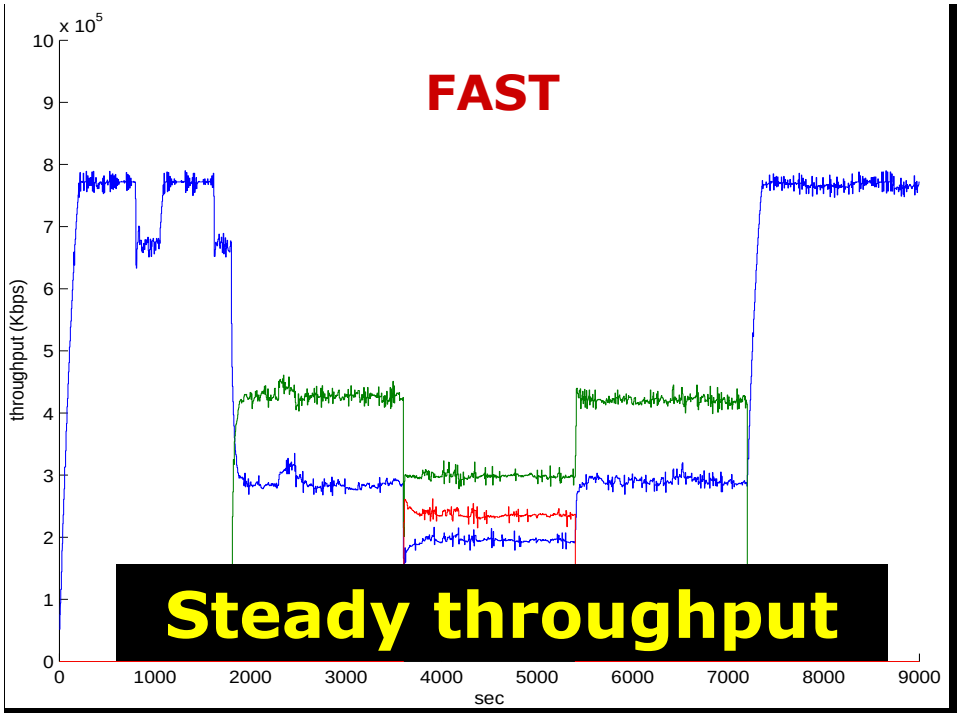
# Window control algorithm

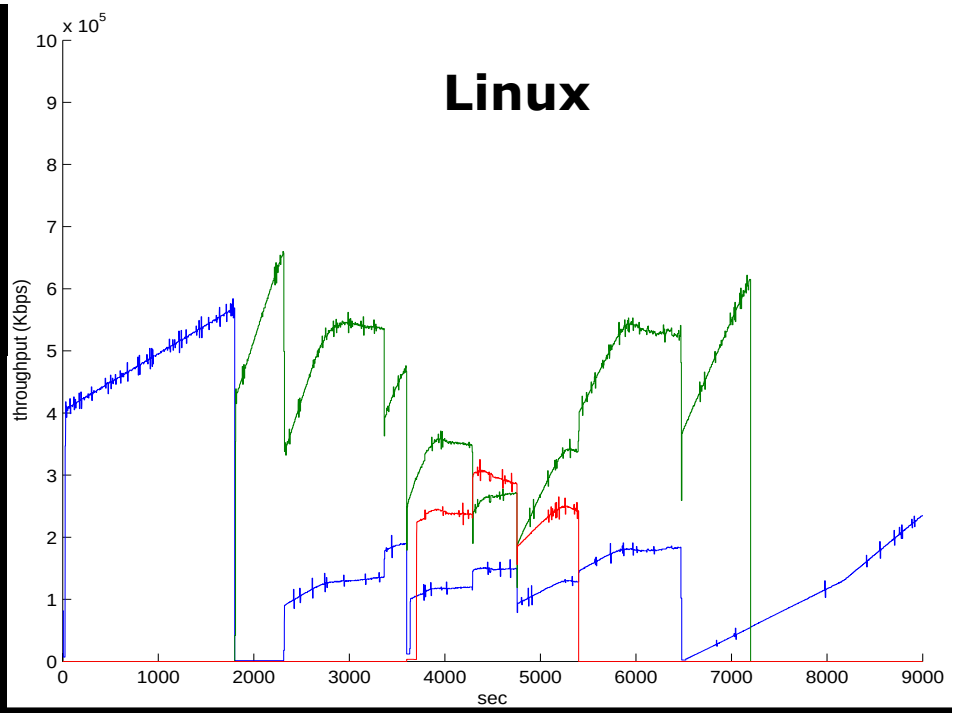
$$\text{window update}: \quad w_i(t+1) = w_i(t) + \gamma\left(\alpha_i - q(t)x_i(t)\right)$$

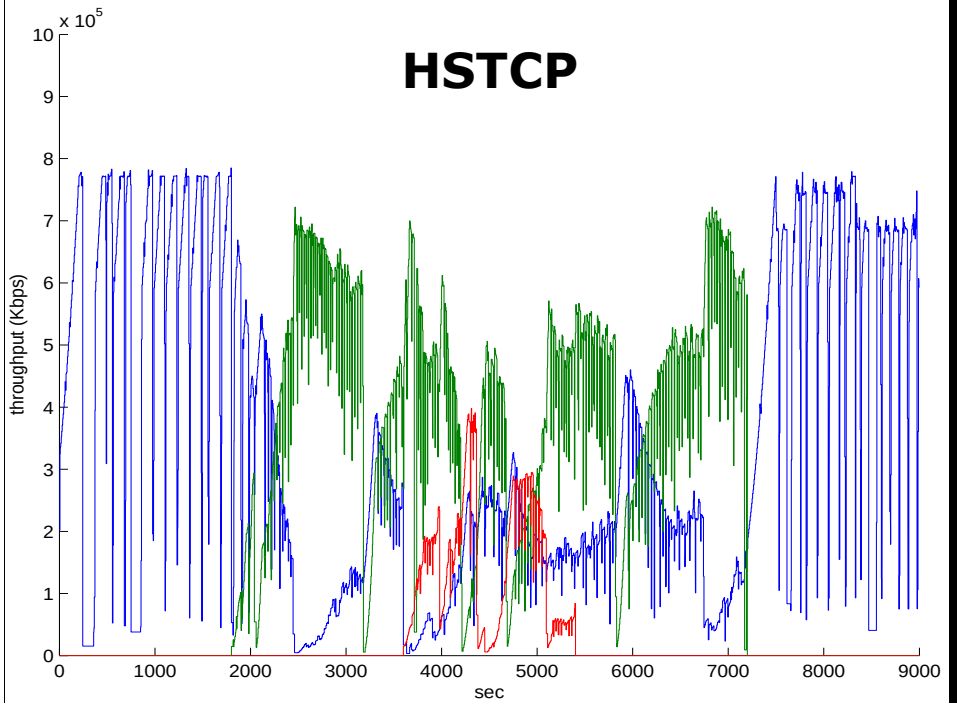
$$\text{self-clocking}: \quad \sum \frac{w_i(t)}{d_i + q_i(t)} = c_l$$

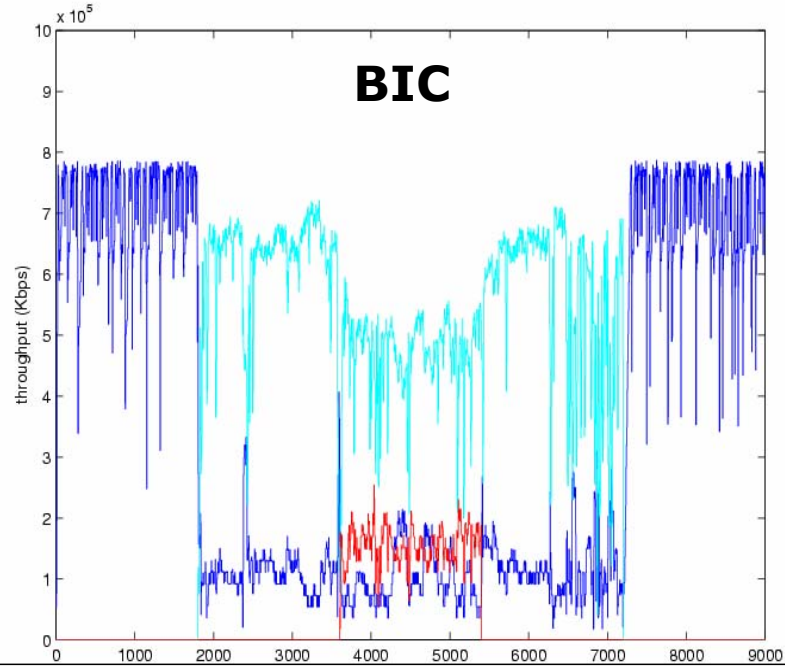
**Theorem** (Infocom04, CDC04, Infocom05)

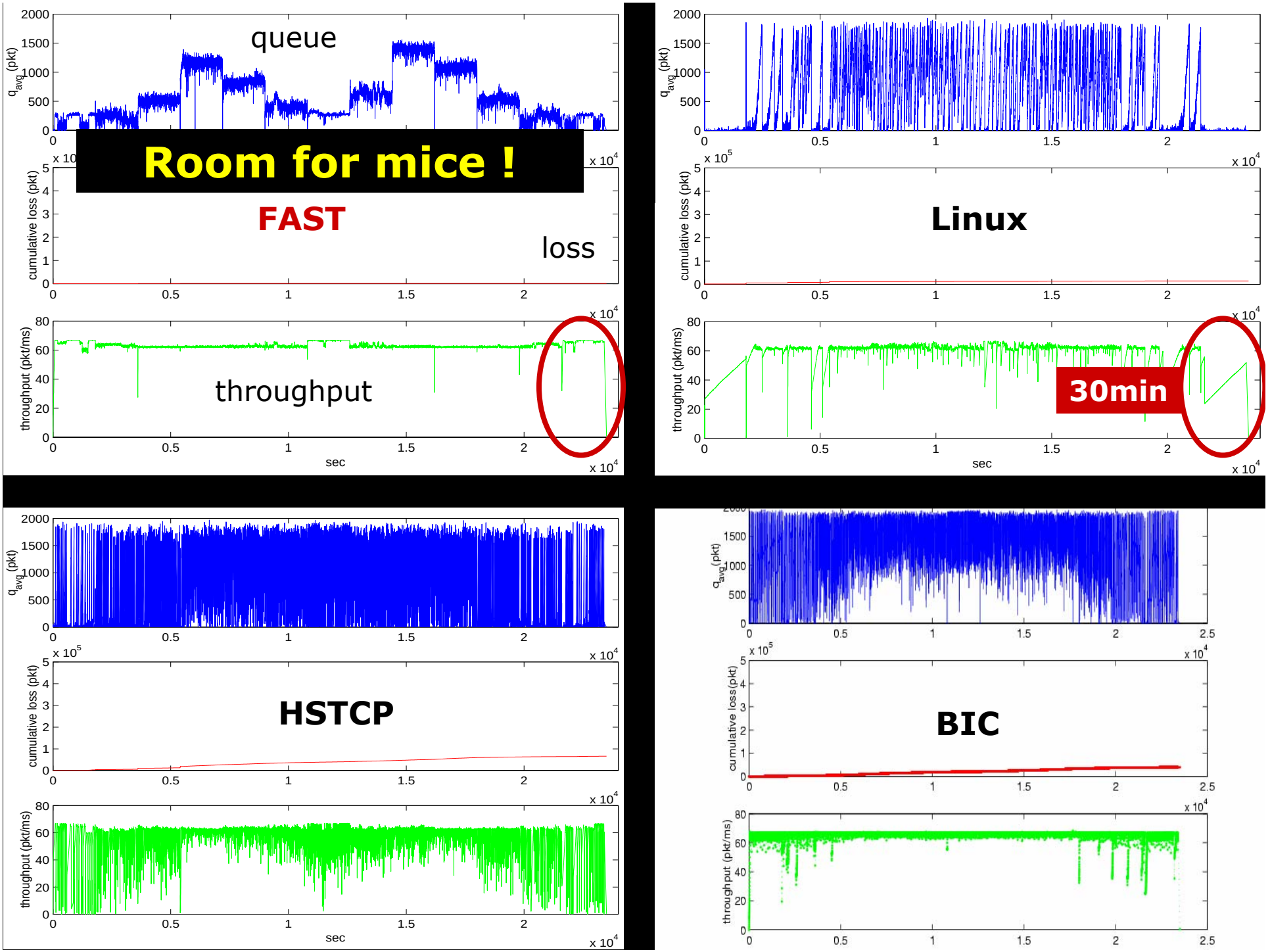
- Mapping from $w(t)$ to $w(t+1)$ is contraction
- Global exponential convergence
- Full utilization after finite time
- Utility function: $\alpha_i$ log $x_i$ (proportional fairness)

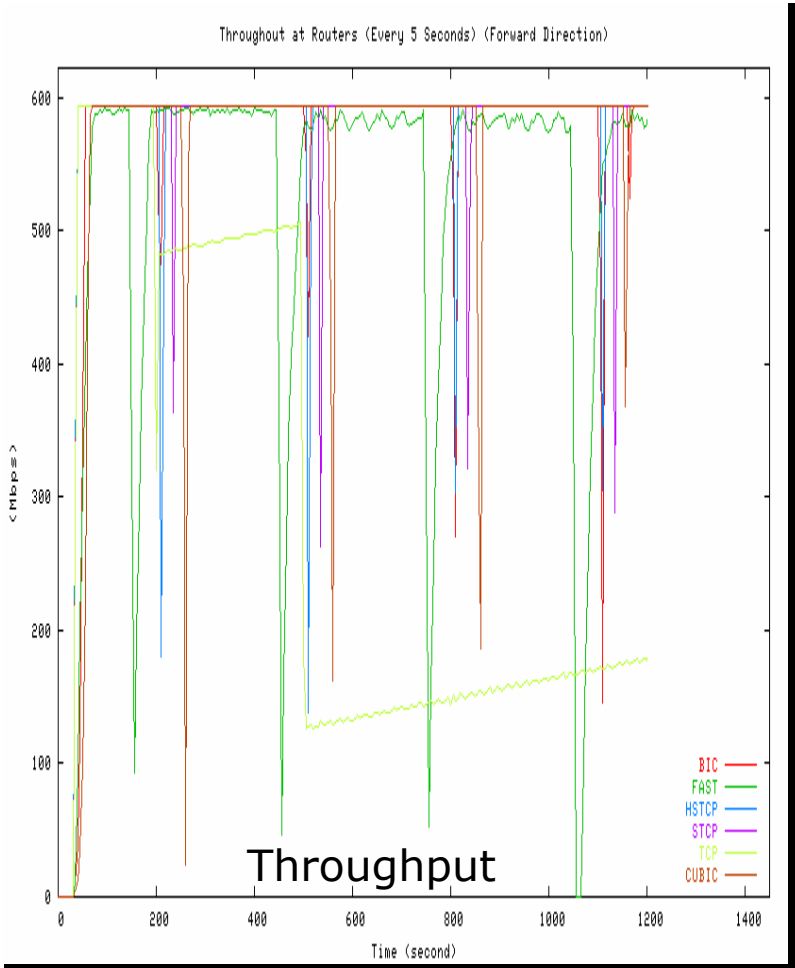

FAST
x 10^5
throughput (Kbps)
sec
Steady throughput

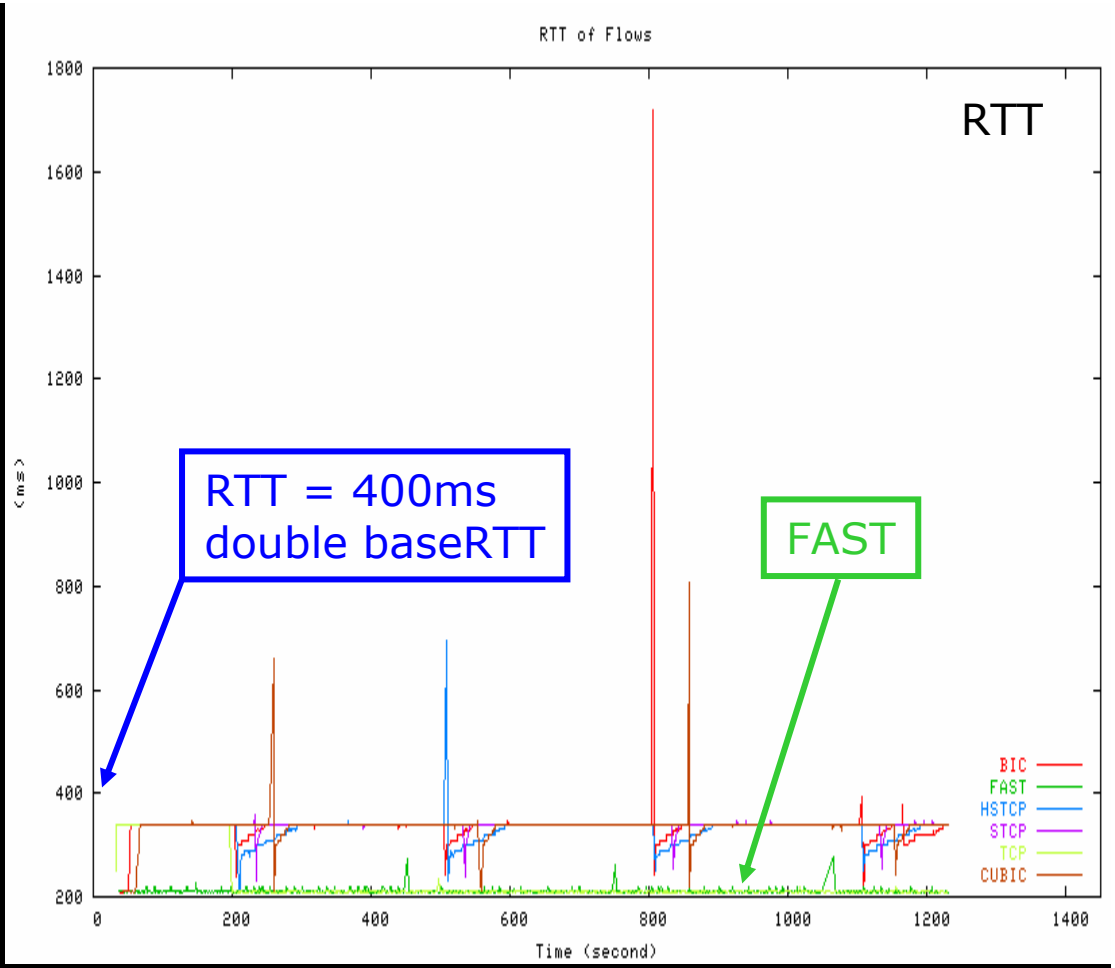

Linux
x 10^5
throughput (Kbps)
sec


HSTCP
x 10^5
throughput (Kbps)
sec


BIC
x 10^5
throughput (Kbps)


queue
Room for mice !
FAST
loss
throughput
sec
Linux
30min
HSTCP
BIC
cumulative loss (pkt)
cumulative loss(pkt)
throughput (pkt/ms)

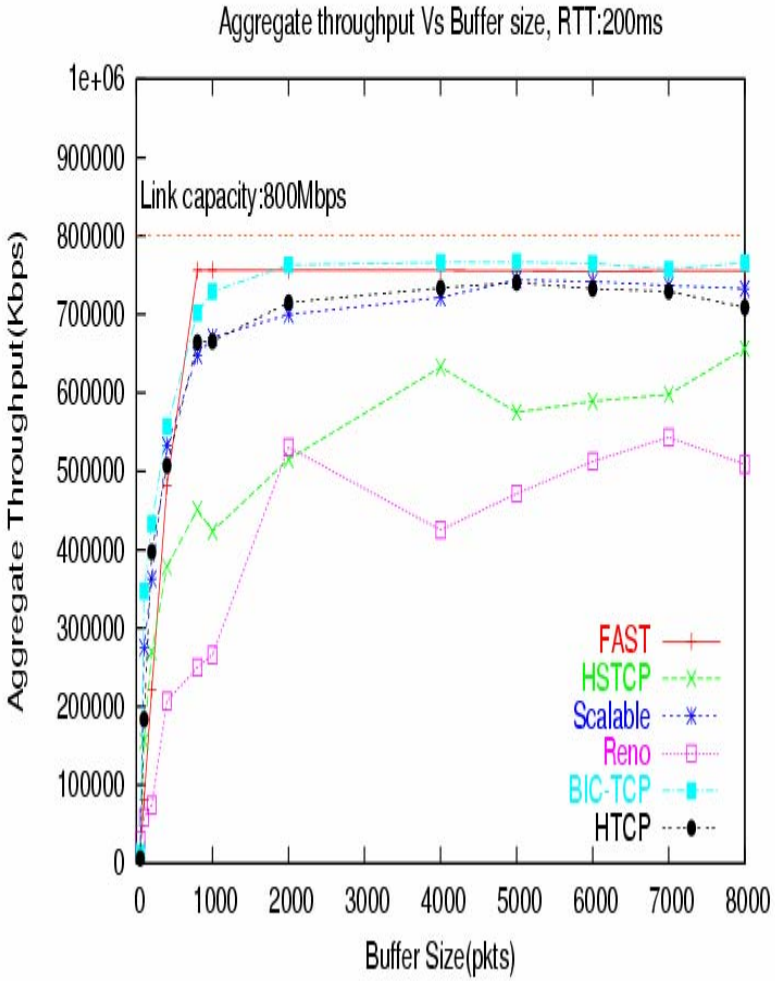


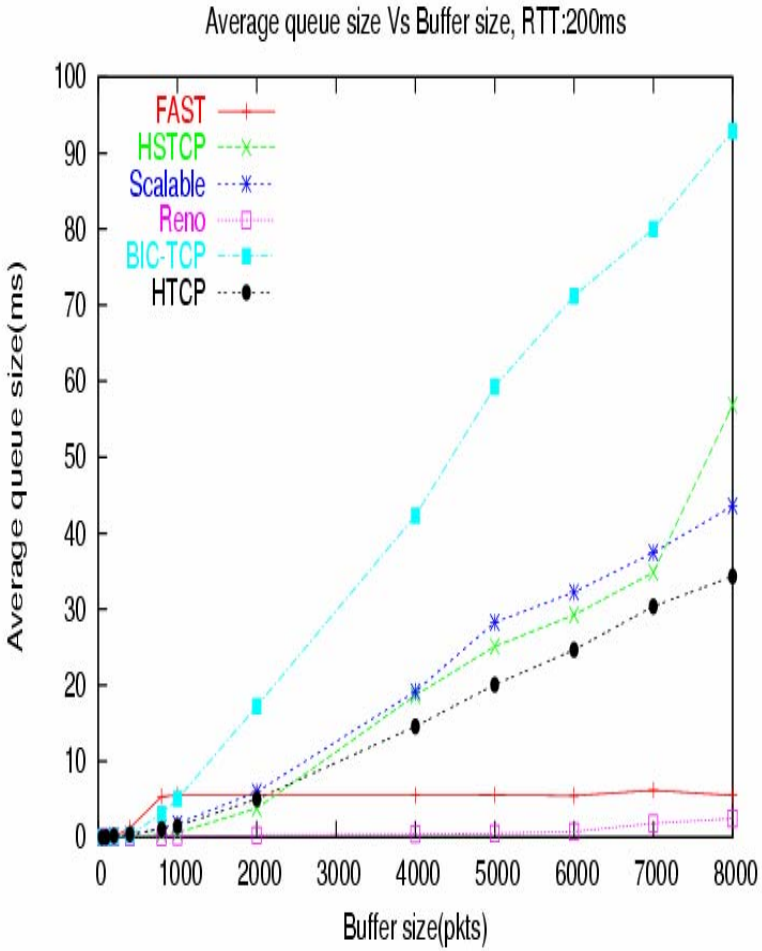



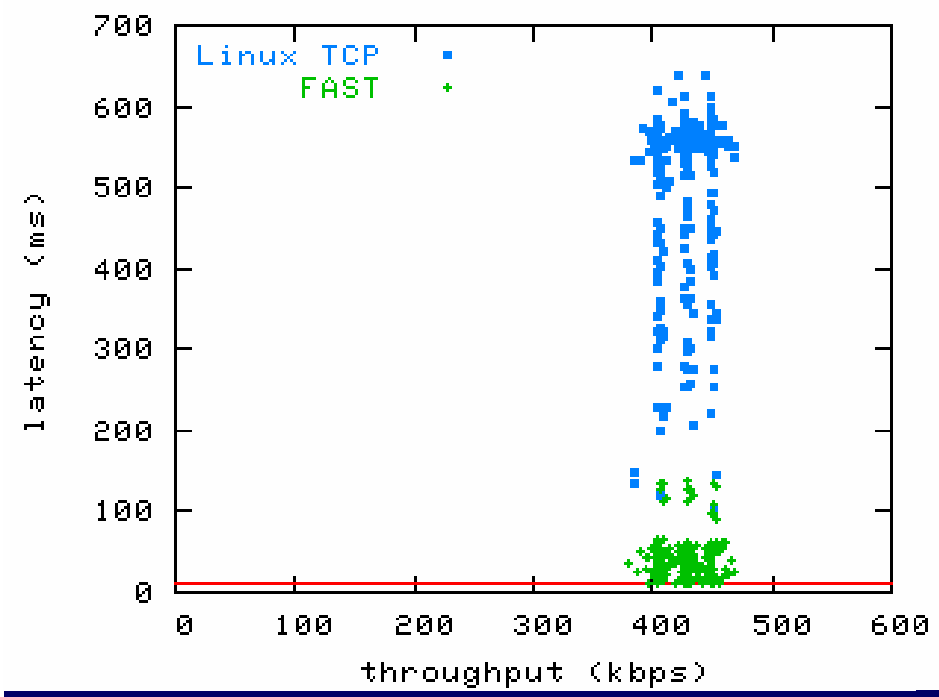

Yusung Kim, KAIST, Korea 10/2004

- All can achieve high throughput except Reno
- FAST adds negligible queueing delay
- Loss-based control (almost) fills buffer …
- adding delay and reducing ability to absorb bursts

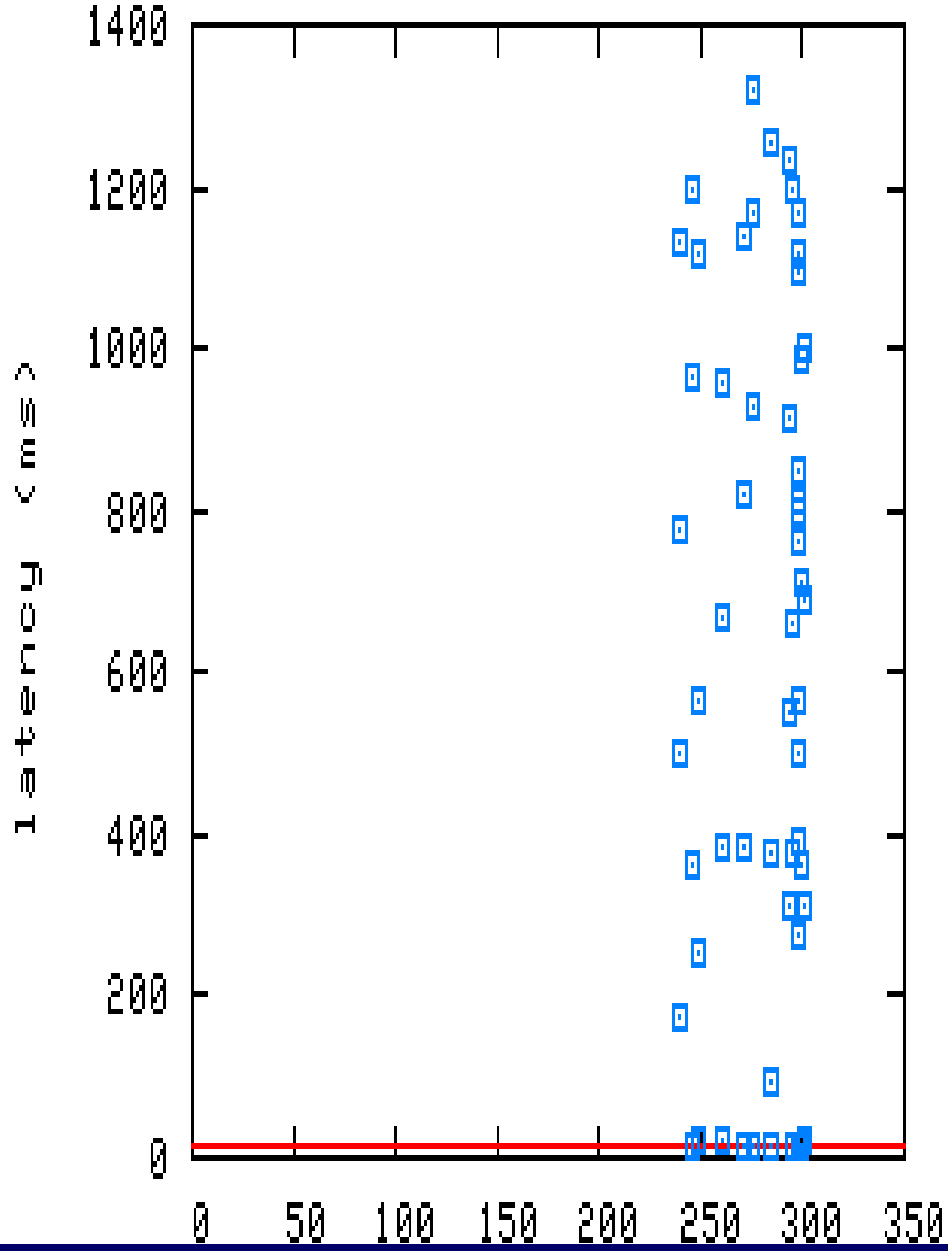
# Is large queue necessary for high throughput?






- DSL upload (6Mbps/512kbps), 5/2005
- Min RTT: 10ms




- DSL upload, 4/28/2005 11:15-11:38am
- Min RTT: 18ms (File size: 16.6MB)

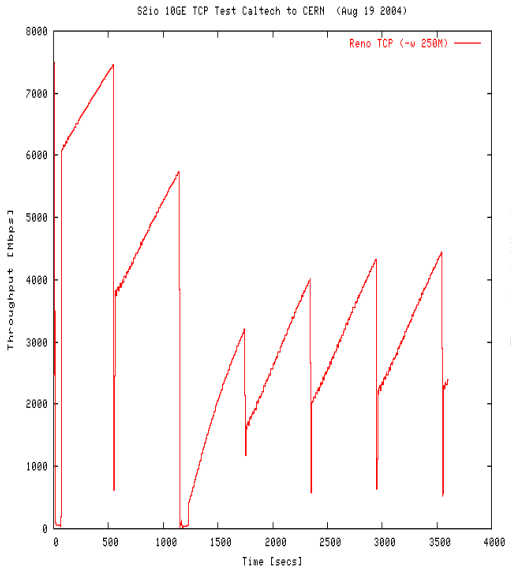
# "Ultrascale" protocol development: FAST TCP

## FAST TCP

- Based on TCP Vegas
- Uses end-to-end delay and loss to dynamically adjust the congestion window
- Defines an explicit equilibrium

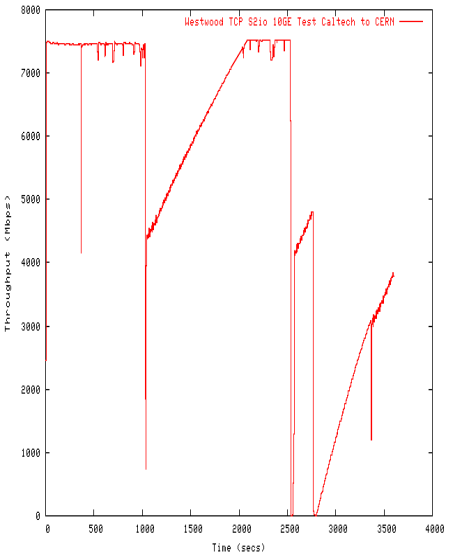
*Capacity = OC-192 9.5Gbps; 264 ms round trip latency; 1 flow*

**BW use 30%**



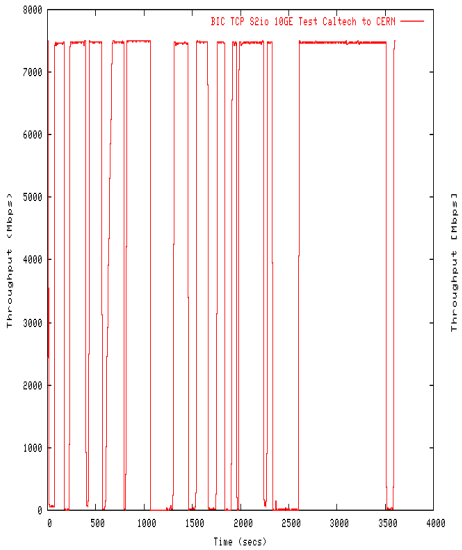

**Linux TCP**

**BW use 40%**



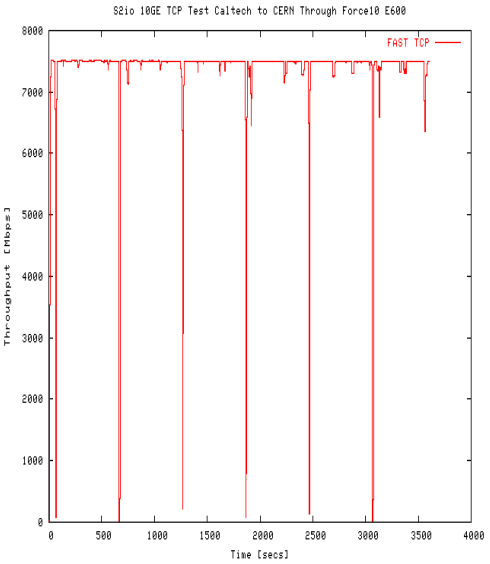

**Westwood+**

**BW use 50%**



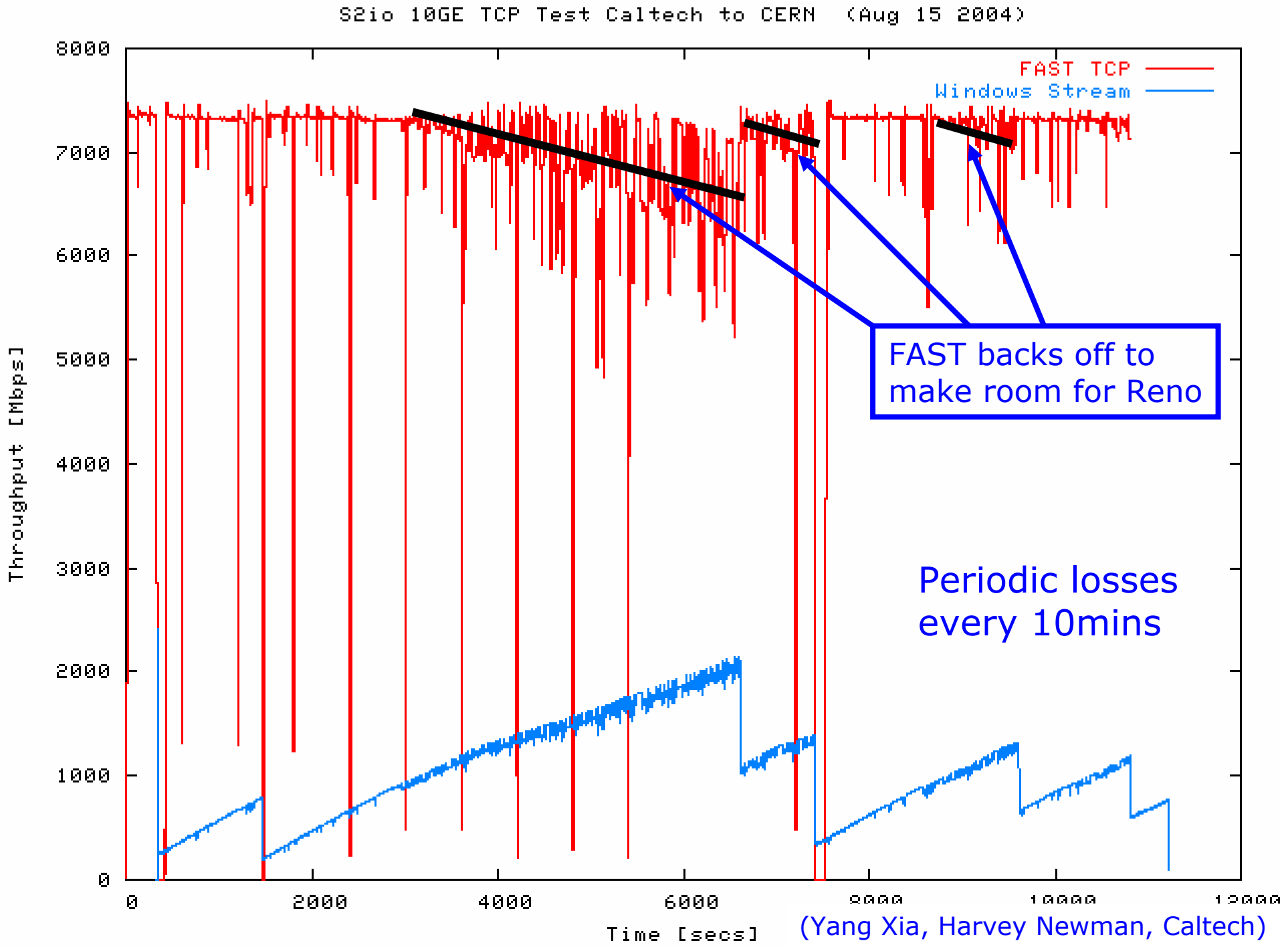

**BIC TCP**

**BW use 79%**

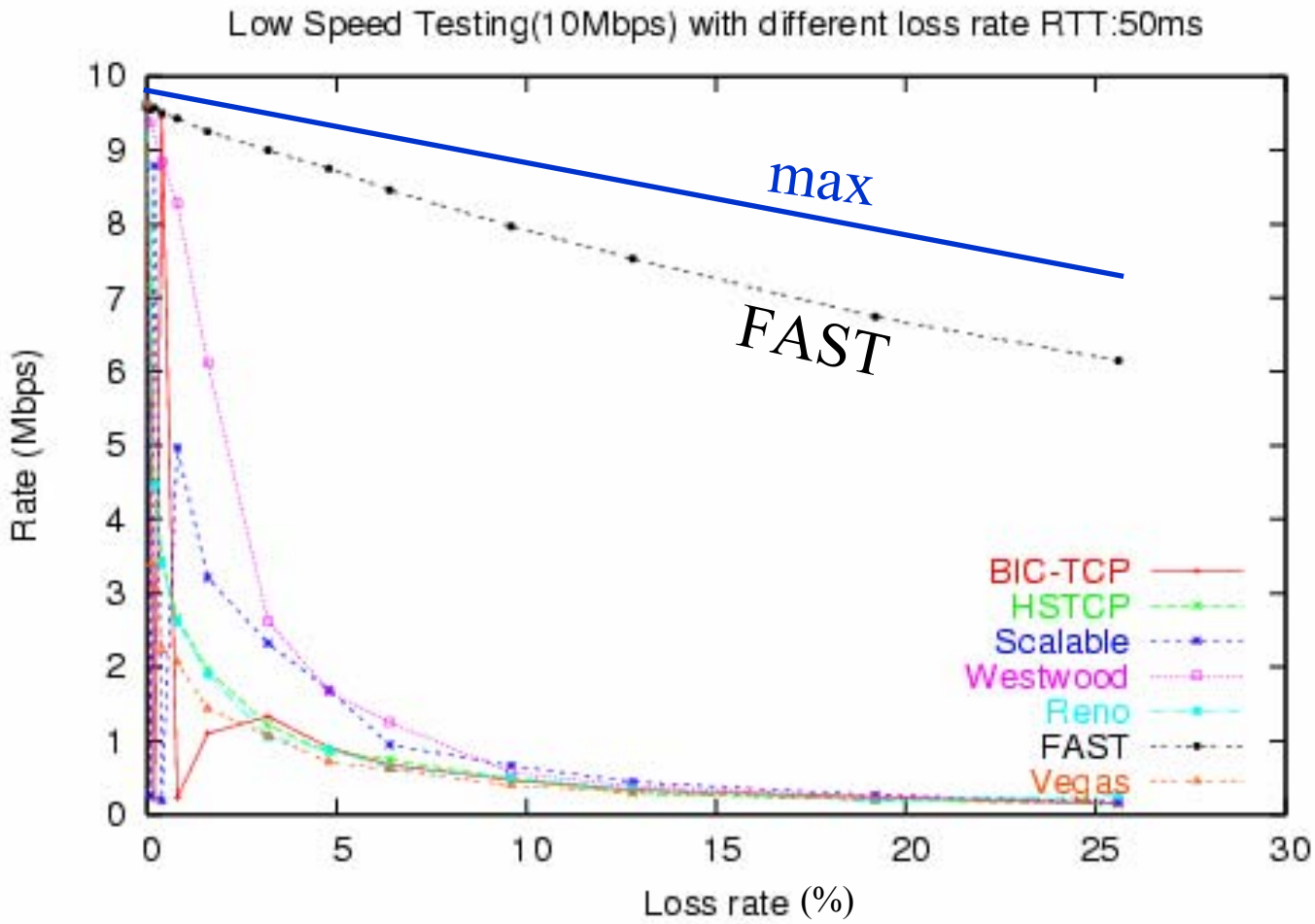



**FAST**

**(Yang Xia, Caltech)**

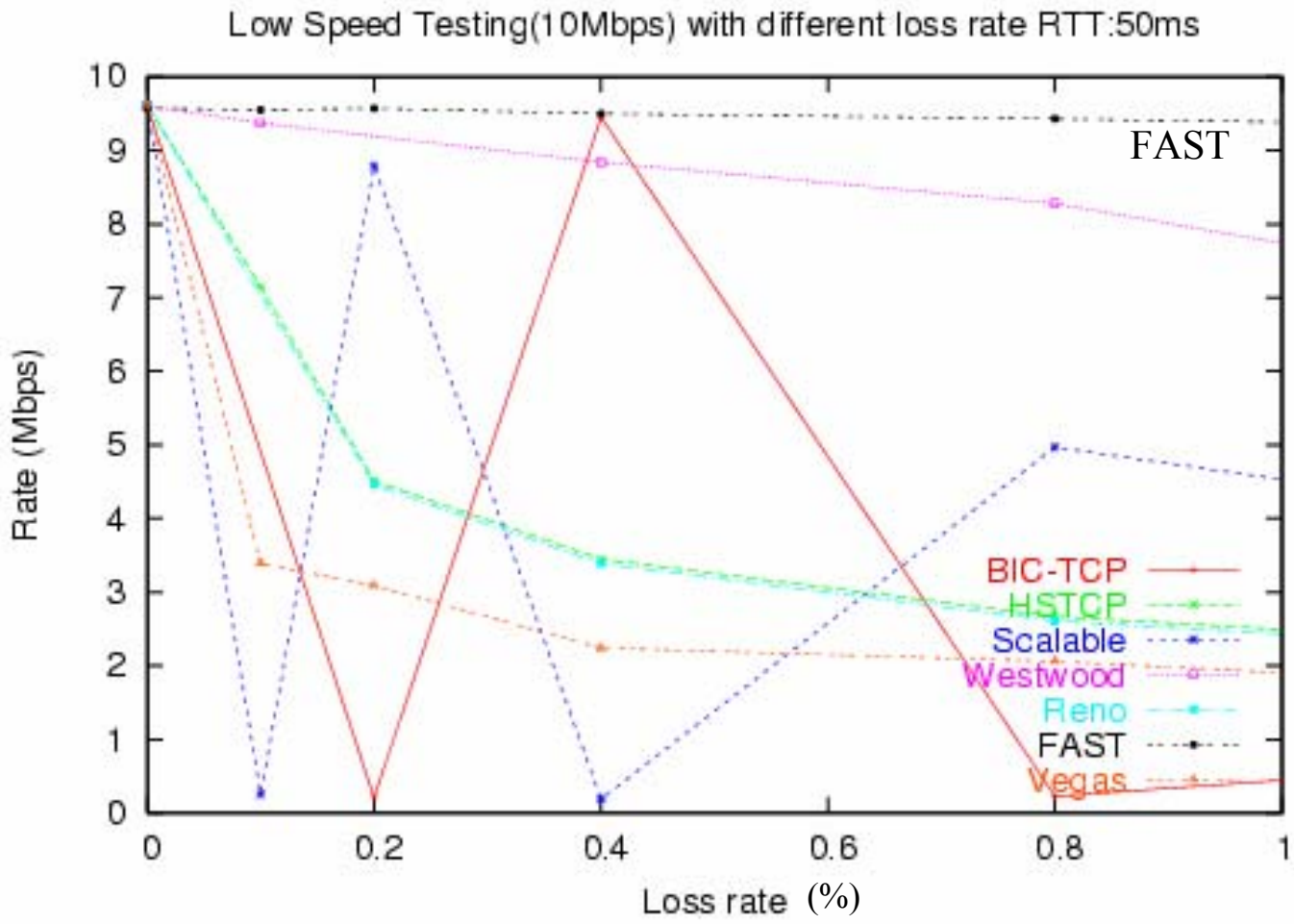

S2io 10GE TCP Test Caltech to CERN (Aug 15 2004)
FAST TCP
Windows Stream
FAST backs off to make room for Reno
Periodic losses every 10mins
Throughput [Mbps]
Time [secs]
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
(Yang Xia, Harvey Newman, Caltech)

# Wireless Networking



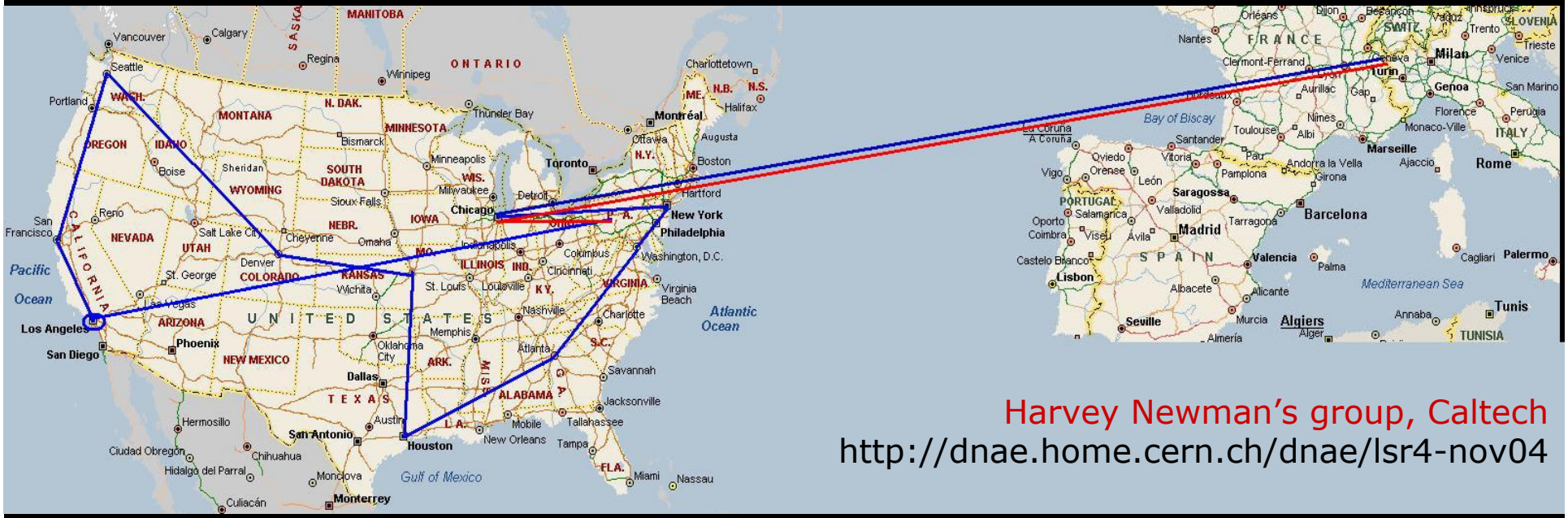

B. Wydrowski
S. Hedge
Caltech, April 2005

# Wireless Networking




B. Wydrowski
S. Hedge
Caltech, April 2005

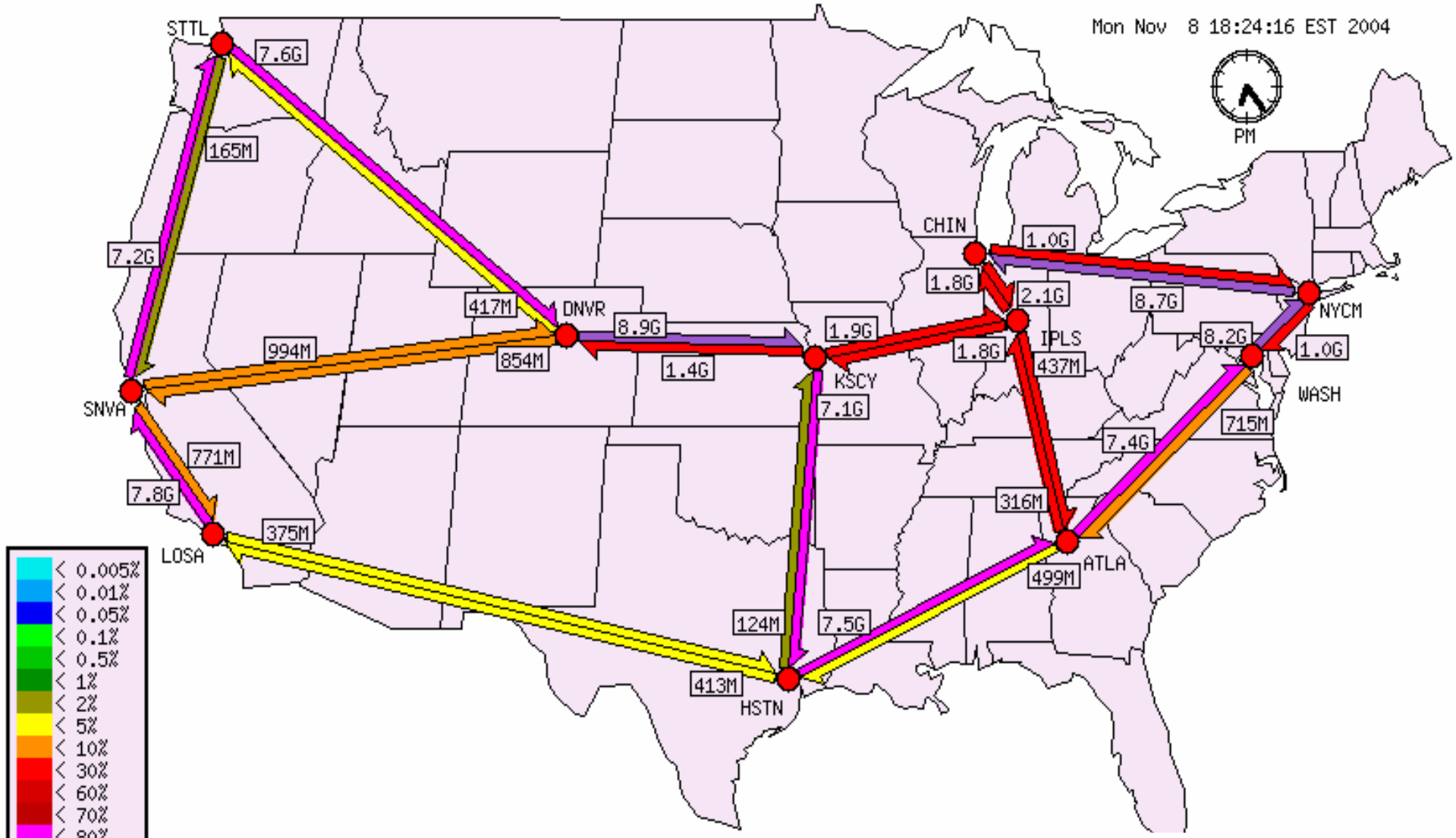
# I2LSR, SC2004 Bandwidth Challenge




November 8, 2004 Caltech and CERN transferred

- 2,881 GBytes in one hour (6.86Gbps)
- between Geneva - US - Geneva (25,280 km)
- through LHCnet/DataTag, Abilene and CENIC backbones
- using 18 FAST TCP streams
- on Linux 2.6.9 kernel with 9000KB MTU
- at 174 Pbm/s

# Internet2 Abilene Weather Map




7.1G: GENV-PITS-LOSA-SNVA-STTL-DNVR-KSCY-HSTON-ATLA-WASH-NYCM-CHIN-GENV

Newman's group, Caltech

**Theory**

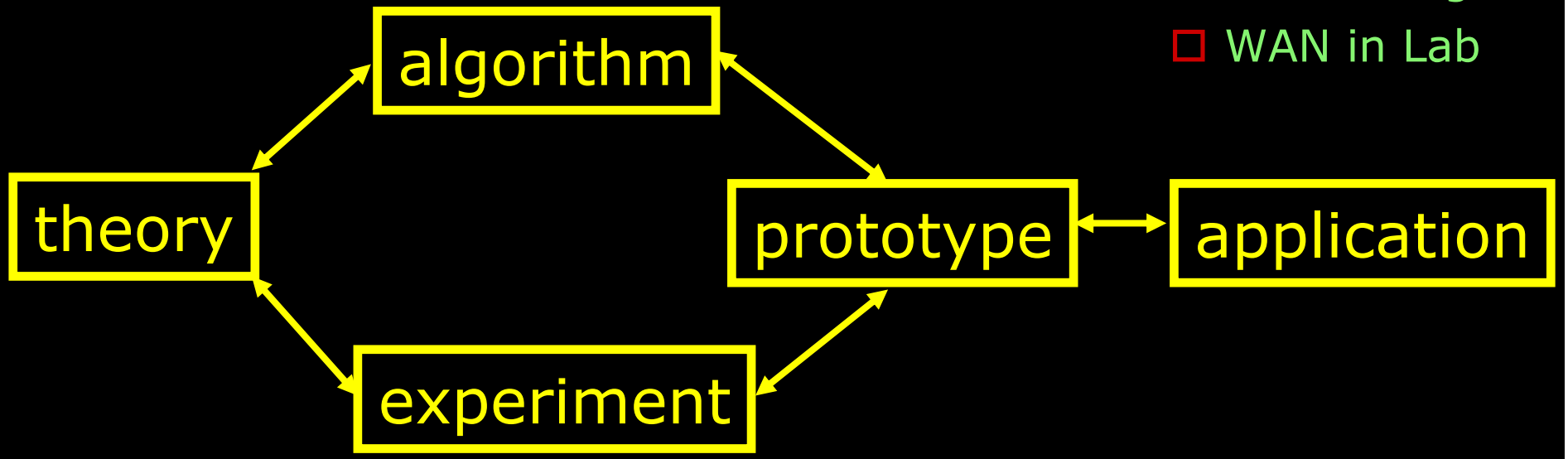
- general large scale network
- performance, fairness, dynamics

**Algorithm, prototype**

- window control
- loss control
- burstiness control

**Experiment**

- HEP
- networking
- WAN in Lab




**Theory**

- heterogeneous protocols
- TCP/IP interactions

**Alg, prototype**

- $\alpha$ –tuning
- loadable kernel module

**Application**

- make robust
- support deployment