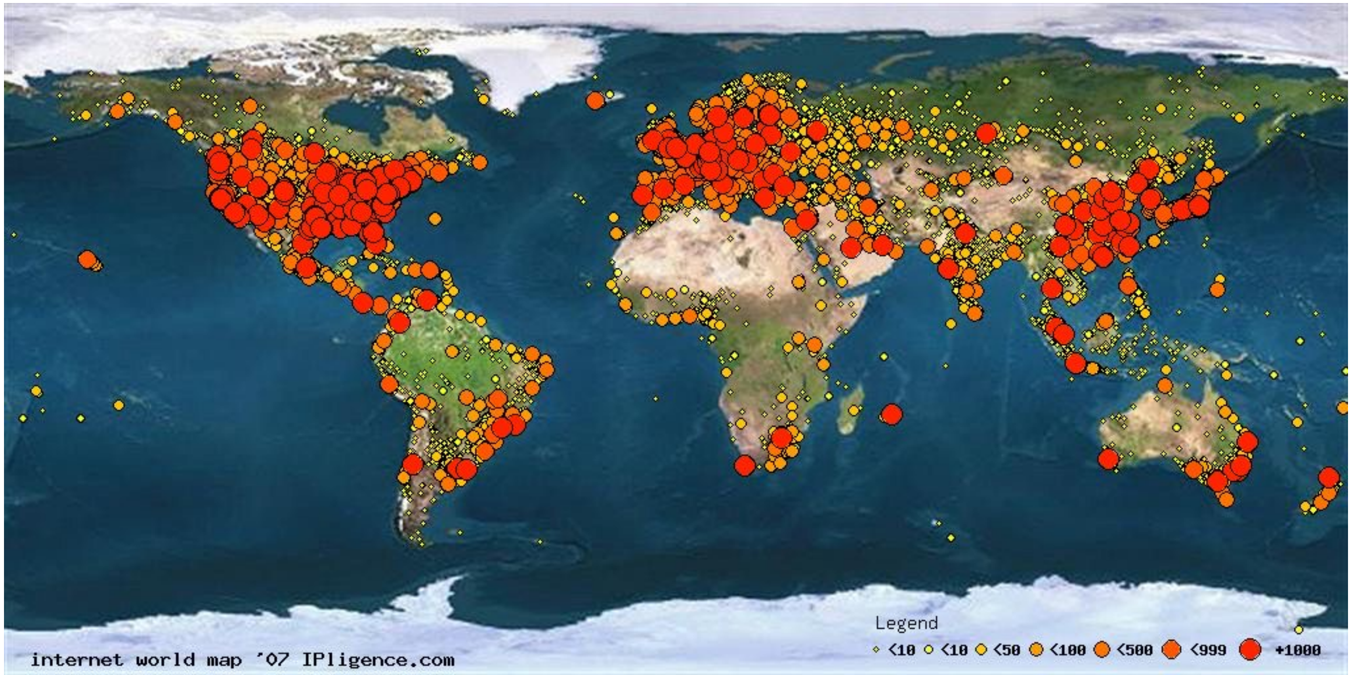
Advanced TCP Congestion Control Algorithms

Sangtae Ha
University of Colorado at Boulder

Current Network Trends

- The Internet evolves by including many high-speed and long distance networks.
- Many multi-national companies centralize their data centers for economical reasons.



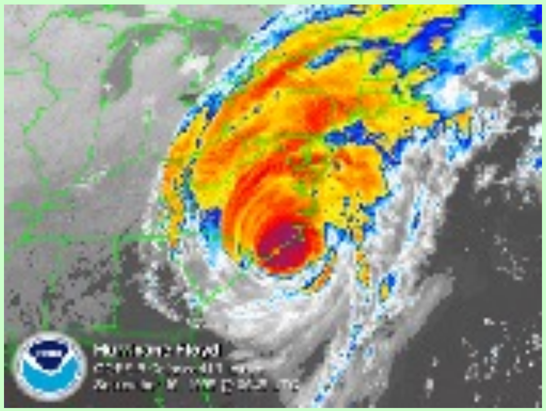
North America: 56%
Europe: 22%
Asia: 14%

Credit: http://www.ipligence.com

High-Speed Applications

High-Speed Applications

Weather Simulation



Video Conference

Telemedicine

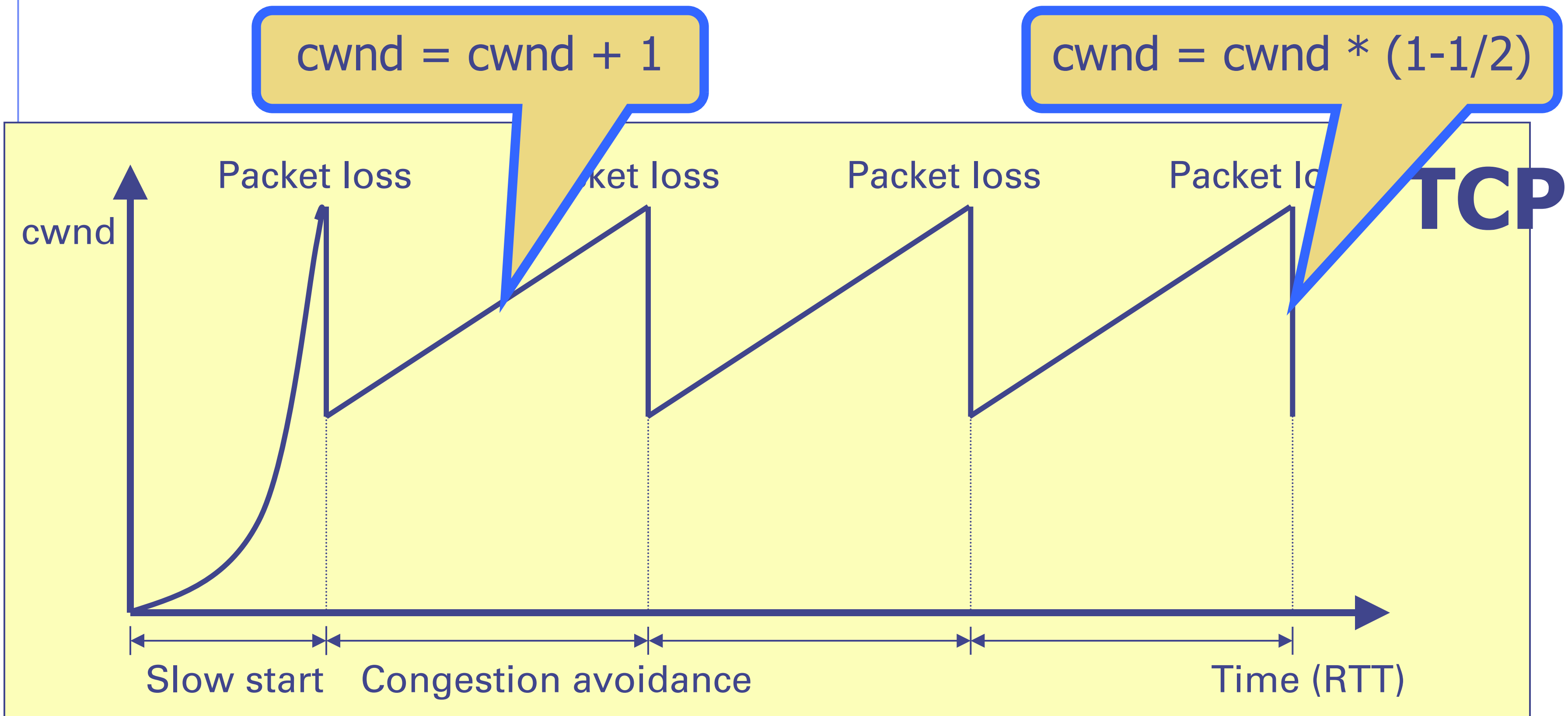
Transport Protocols

be able to transfer a large amount of data over a long distance within a short amount of time

High-Speed Networks

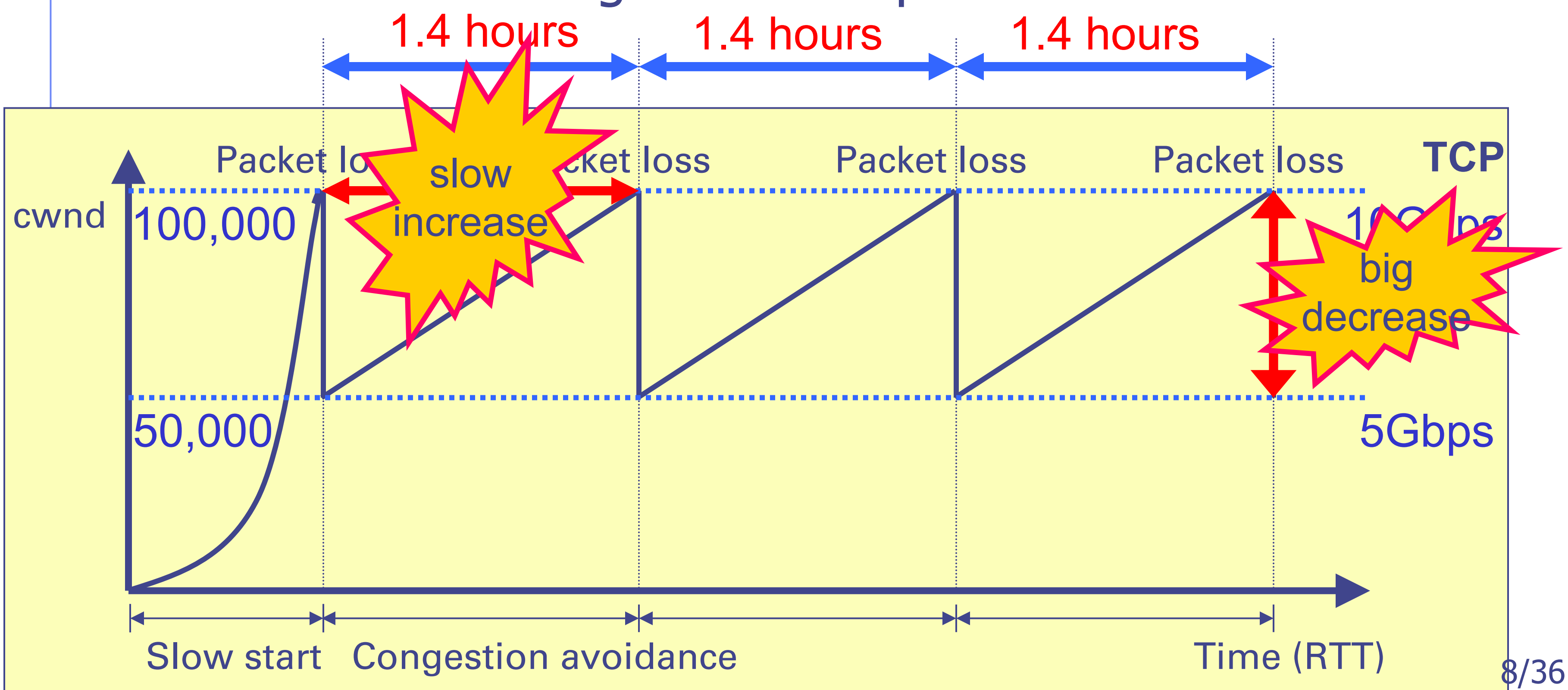
TCP Congestion Control

- The instantaneous throughput of TCP is controlled by a variable *cwnd*,
- TCP transmits approximately a *cwnd* number of packets per RTT (Round-Trip Time).



Performance Problem – Closer look on what's happening inside

- Slow congestion window growth during CA
- A TCP connection with 1250-byte packet size and 100m RTT running over 10Gbps link.



Existing work

Distinguishing losses	Delay and rate-based TCPs	A hybrid delay and loss based TCPs	Forward Error Correction
* Jain's Congestion Predictor * TCP Veno * ECN	* Vegas, FAST * Westwood * WTCP	* H-TCP * TCP-Illinois * TCP-Africa * Yeah-TCP * Compound-TCP	* LT-TCP * Maelstrom

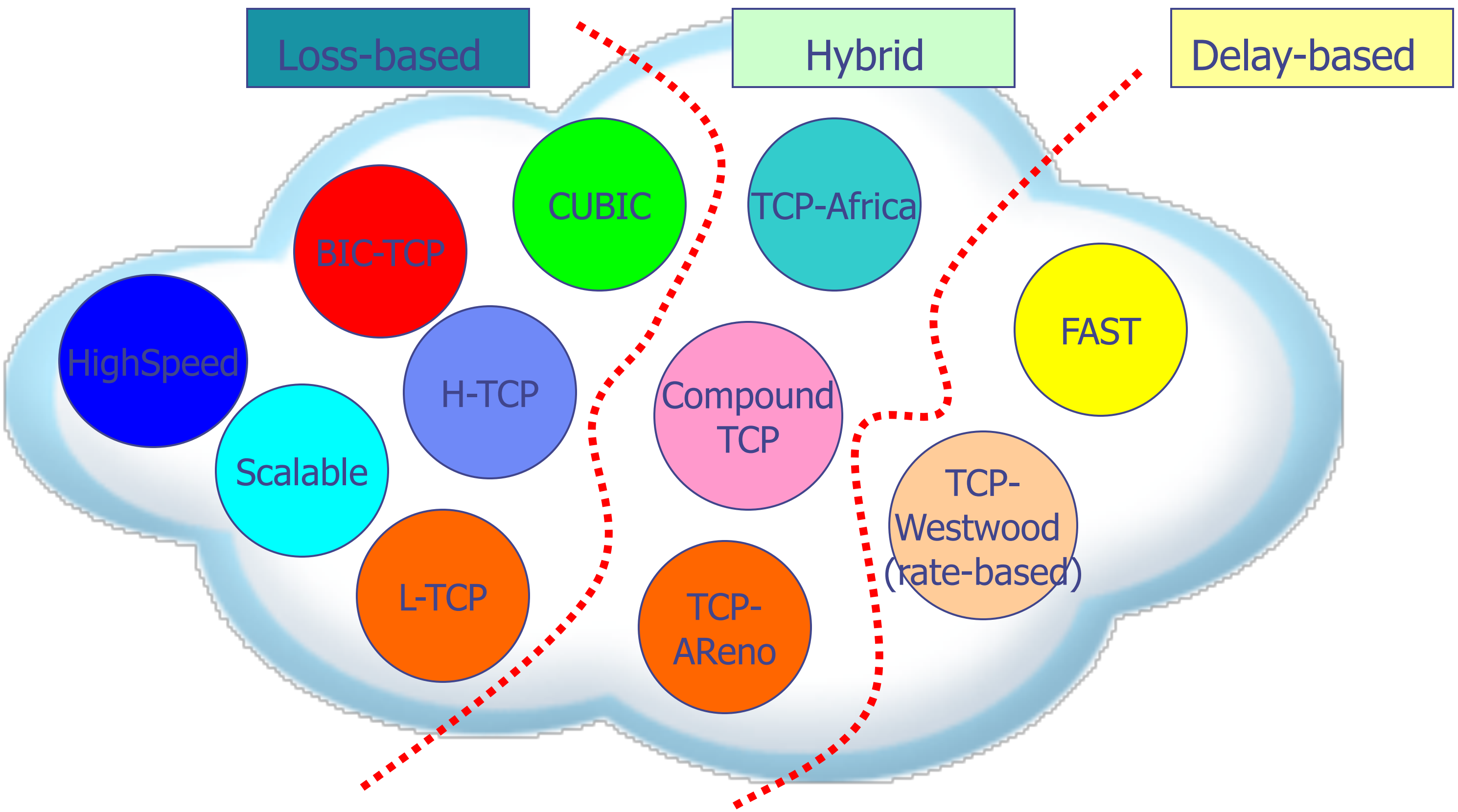
Proposed High-Speed Protocols

- Window-Based Protocols
  - AIMD (Additive Increase Multiplicative Decrease)
    - Jumbo Frame, GridFTP, PFTP, PSockets
  - HSTCP (High-Speed TCP) by Sally Floyd at ICIR, Berkeley
  - STCP (Scalable TCP) by Tom Kelly at Cambridge University
  - FAST (Fast AQM Scalable TCP) by Steven Low at California Institute of Technology

- Rate-Based Protocols
  - SABUL (Simple Available Bandwidth Utilization Library) by Robert Grossman at University of Illinois at Chicago

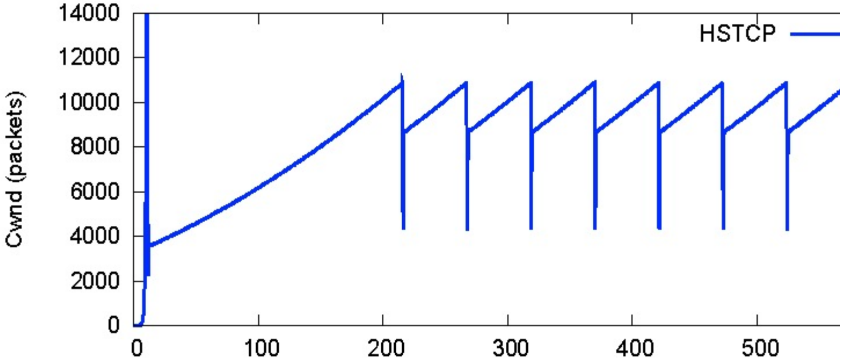
window-based protocols are known for safer incremental deployment.
D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic behavior of slowly responsive congestion controls", In Proceedings of SIGCOMM 2001, San Diego, California.

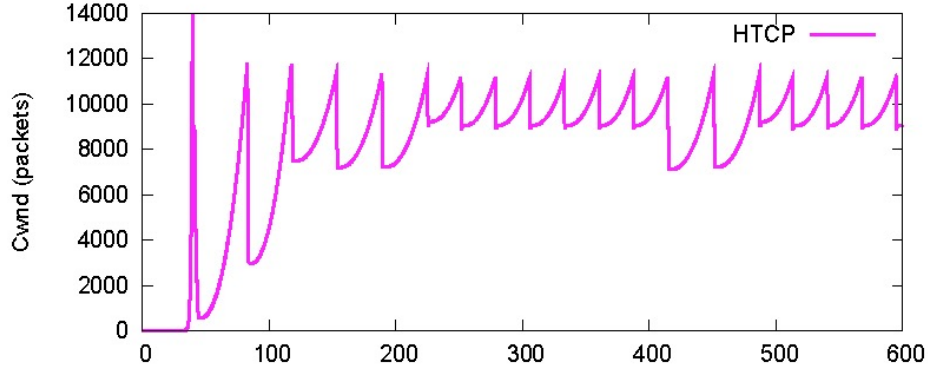
High-Speed Protocols

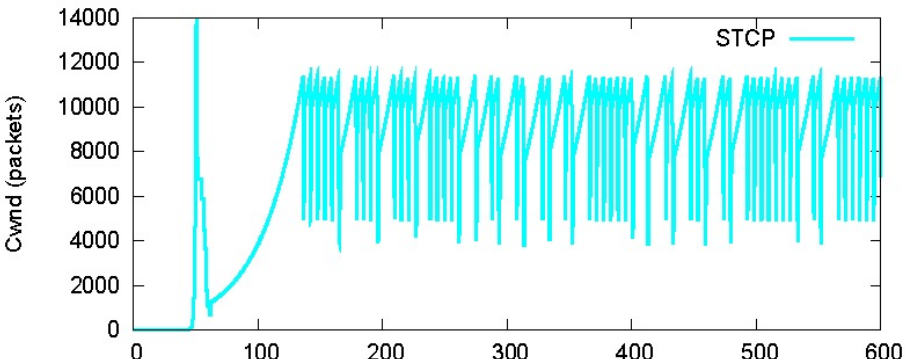


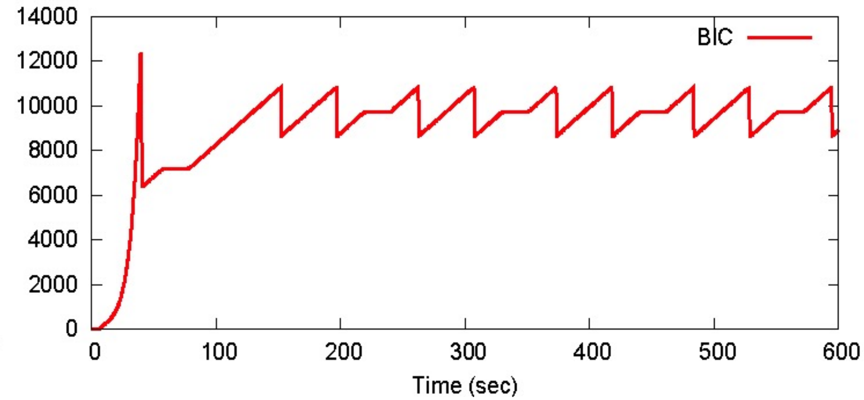
- The difference among the protocols lie in how they search an available bandwidth of a path.

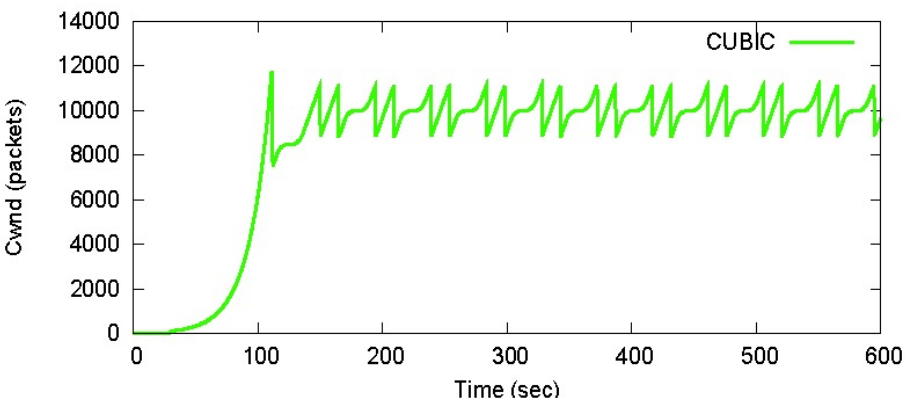
Examples of bandwidth search functions

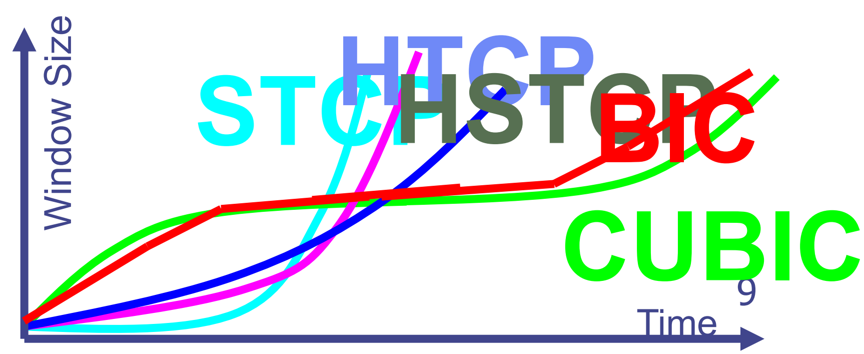




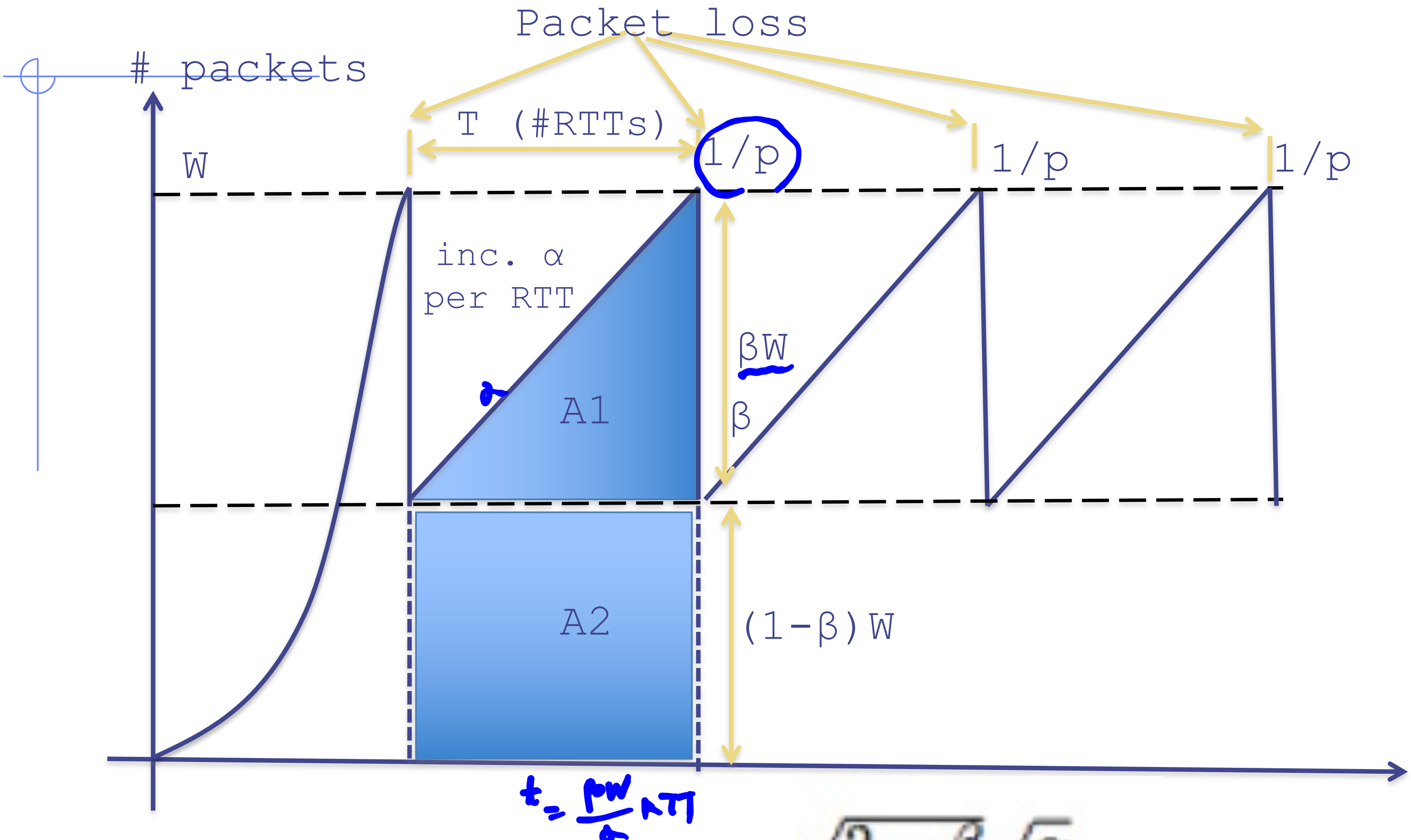








AIMD (α, β) Throughput



$$E\{W_{aimd}\} = \frac{\sqrt{2-\beta}\sqrt{\alpha}}{\sqrt{2\beta}RTT\sqrt{p}}$$

AIMD(1, 0.5)

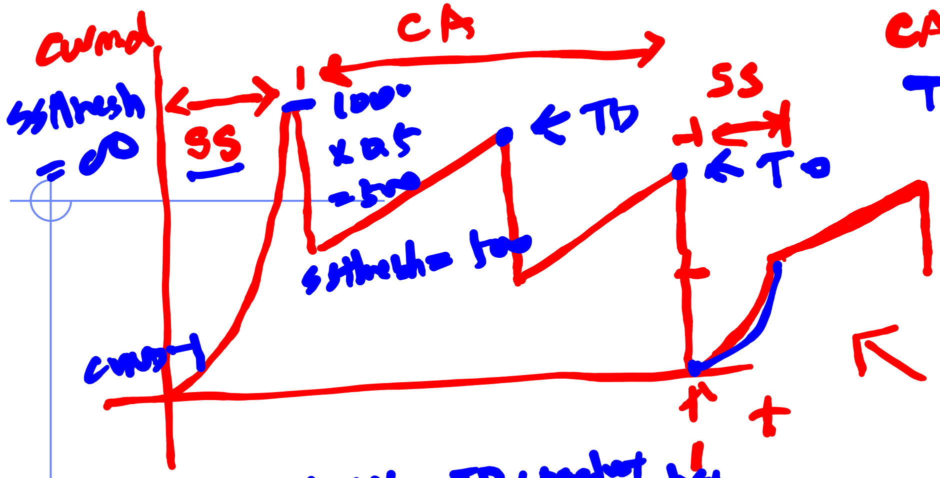
SS: Slow Start

Ack= Quick Ack

CA= Congestion Avoidance Delay Ack

TD= Triple Duplicate Acks

TO: Time out.



CWND : # of packets (the window)

ssthresh : slow start threshold ← ∞

if cwnd < ssthresh (SS)

 cwnd = cwnd + 1 (per Ack)

else (CA)

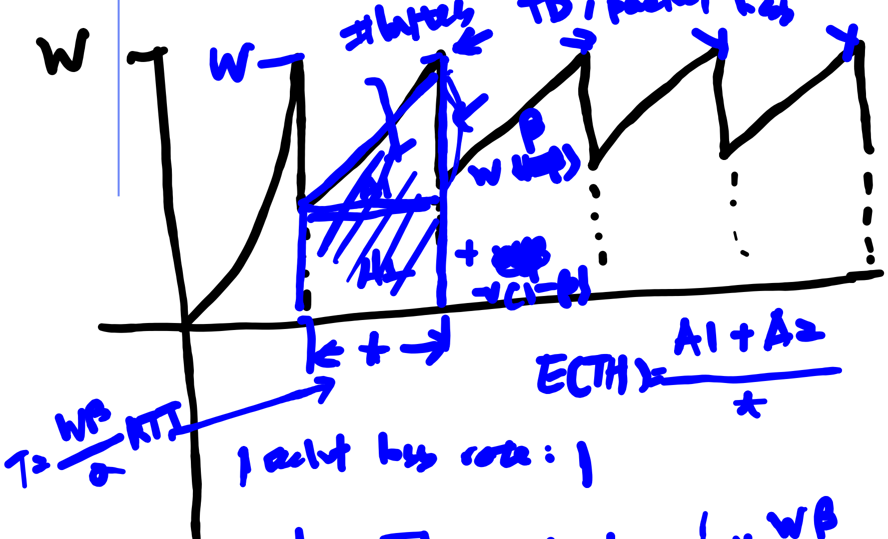
 $CWND = CWND + \boxed{\frac{a}{CWND}}$ (per Acks) $\frac{1}{cwnd}$

 ↑ CWND= CWND + 1

#upon packet loss

 CWND = CWND × β (0.5)

 ssthresh = CWND



$$\frac{1}{p} = \boxed{A}(A1+A2) = \frac{\frac{1}{2} \times \frac{W\beta}{a} \cdot W\beta}{A1}$$

$$\parallel$$

$$\frac{W^2(2-\beta)\beta}{2a} \Leftarrow + \left(W(1-\beta)\frac{W\beta}{a}\right)$$

a=1

□ΔΔΔ → 4 Acks

$4 + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 5$

TO: CWND = 1

AIMD (α, β) Throughput (Cont'd)

W: The window size immediately before a loss event

α: ~~Addictive increase~~ factor. The window size increases α every RTT round

β: The multiplicative decrease factor. After a loss event, the window size is reduced to (1-β)W

RTT: The RTT of a flow

P: Loss event rate

The total number of packets sent in a loss epoch is A1+A2, then 1/p = A1+A2

$$\frac{1}{p} = A = \left(W\beta\frac{W\beta}{\alpha}\right)\frac{1}{2} + \left(W(1-\beta)\frac{W\beta}{\alpha}\right) = \frac{W^2(2-\beta)\beta}{2\alpha}$$

By solving the above equation for W will be $W = \frac{\sqrt{2\alpha}}{\sqrt{\beta(2-\beta)p}}$

The number of RTTs (T) during one loss epoch is $T = \frac{W\beta}{\alpha}RTT$

The average window size (throughput) is A/T

$$E\{W_{aimd}\} = \frac{A}{T} = \frac{\frac{W^2(2-\beta)\beta}{2\alpha}}{\frac{W\beta}{\alpha}RTT} = \frac{\sqrt{2-\beta}\sqrt{\alpha}}{\sqrt{2\beta}RTT\sqrt{p}} = AIMD(\alpha,\beta)$$

Response Function of TCP

- Response function of TCP is the average throughput of a TCP connection in terms of the packet loss probability, the packet size, and the round-trip time.
- Response Function of TCP is :

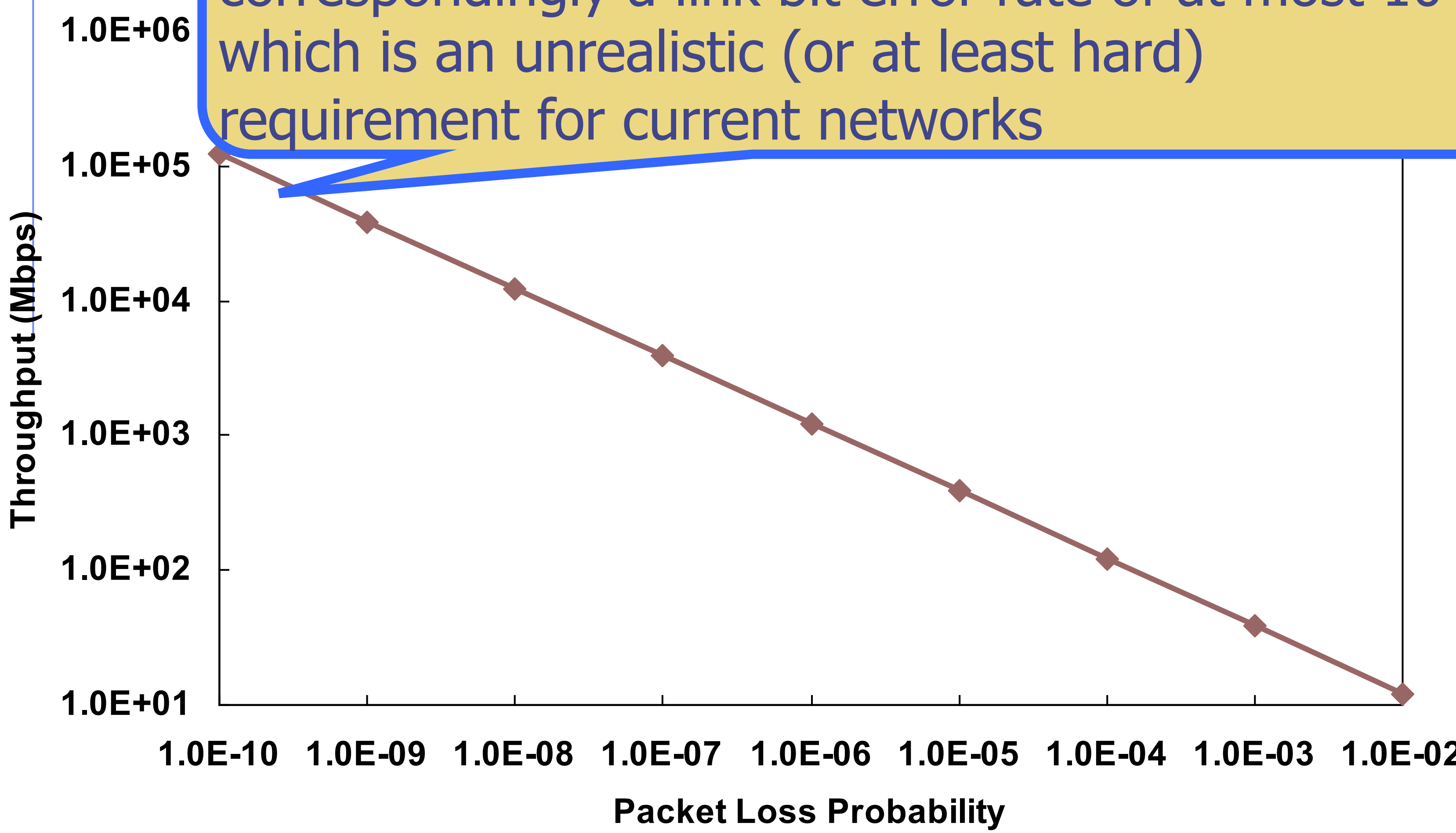
$$R = \frac{MSS}{RTT}\frac{1.2}{p^{0.5}}$$

  - R : Average Throughput
  - MSS: Packet Size
  - RTT: Round-Trip Time
  - P : Packet Loss Probability

J. Padhye, V. Firoio, D. Towsley, J. Kurose, "Modeling TCP Throughput: a Simple Model and its Empirical Validation", Proceedings of SIGCOMM 98

Response Function of TCP

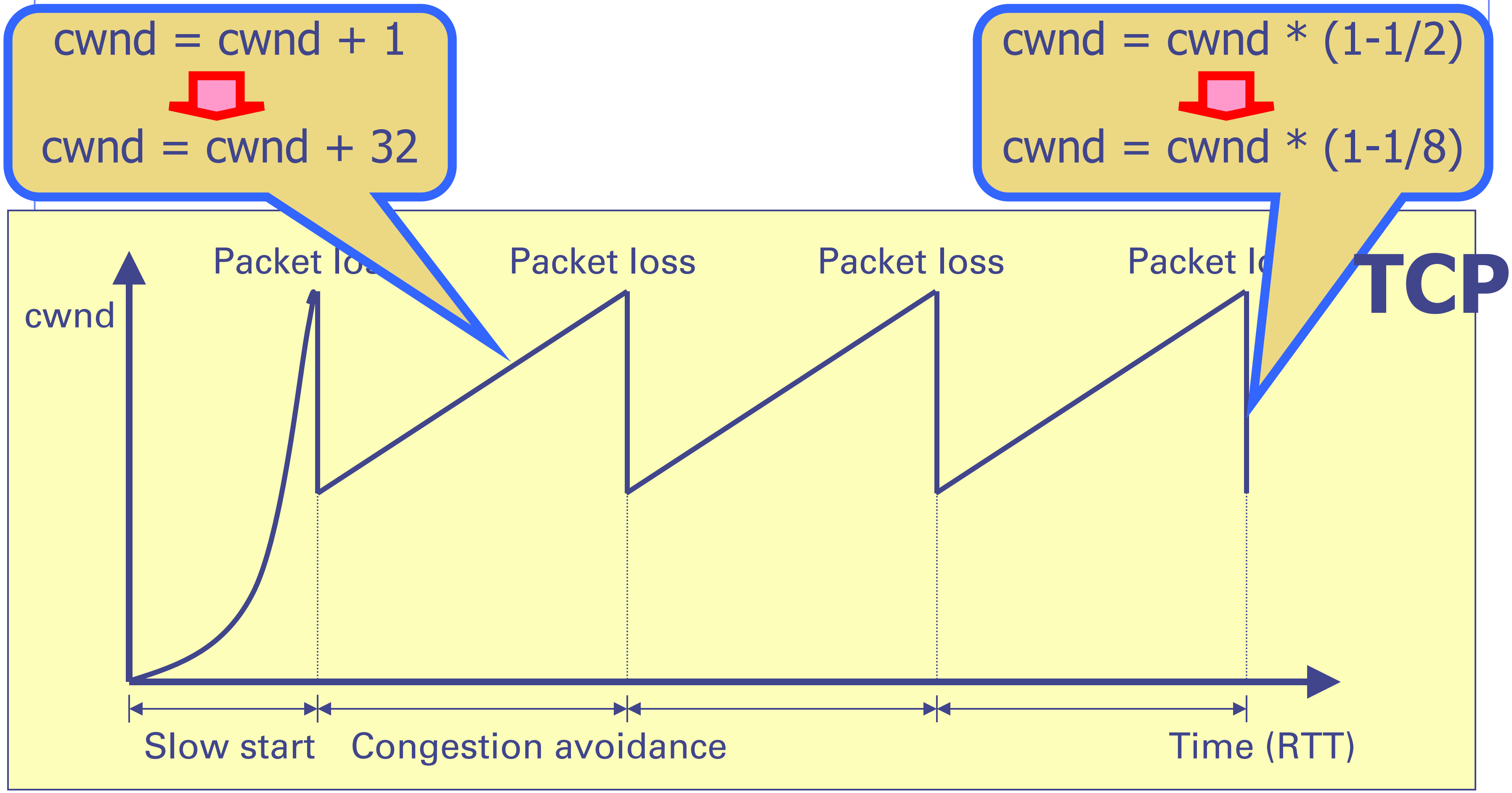
10Gbps requires a packet loss rate of 10^{-10}, or correspondingly a link bit error rate of at most 10^{-14}, which is an unrealistic (or at least hard) requirement for current networks



Assuming 1250-Byte packet size, and 100ms RTT

AIMD (Additive Increase Multiplicative Decrease)

- AIMD increases *cwnd* by a larger number, say 32, instead of 1 per RTT.
- After a packet loss, AIMD decreases *cwnd* by 1/8, instead of 1/2

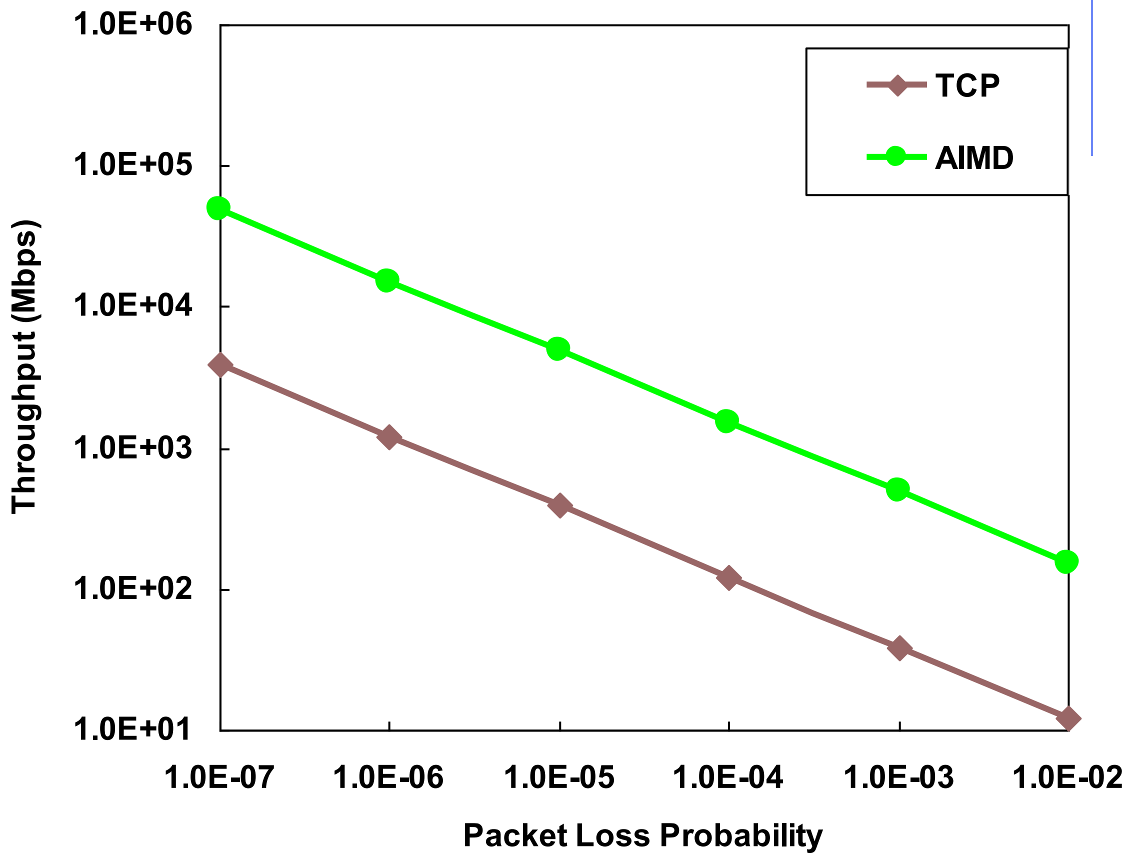


Response Function of AIMD

- TCP: $R = \dfrac{MSS}{RTT}\dfrac{1.2}{p^{0.5}}$

- AIMD: $R = \dfrac{MSS}{RTT}\dfrac{15.5}{p^{0.5}}$

The throughput of AIMD is always about 13 times larger than that of TCP



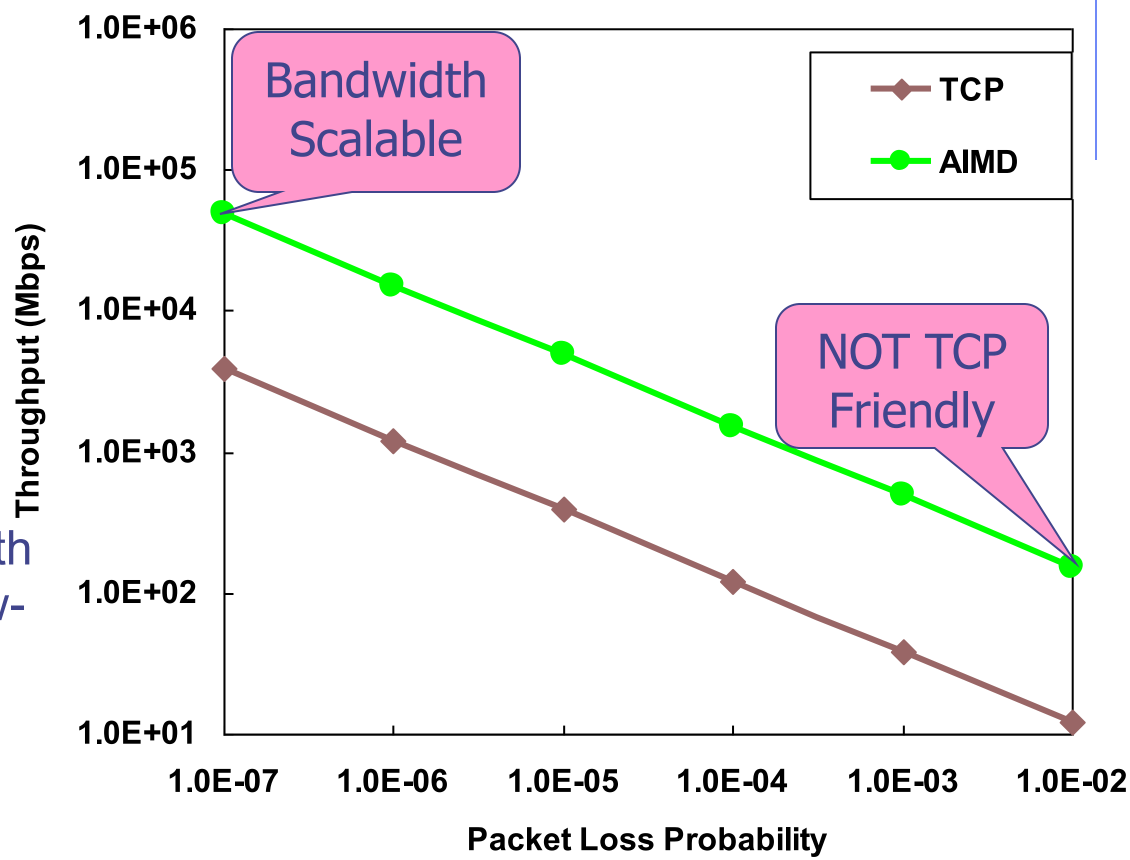
Properties of AIMD

- **Bandwidth Scalability**

  The ability to achieve 10Gbps with a reasonable packet loss probability

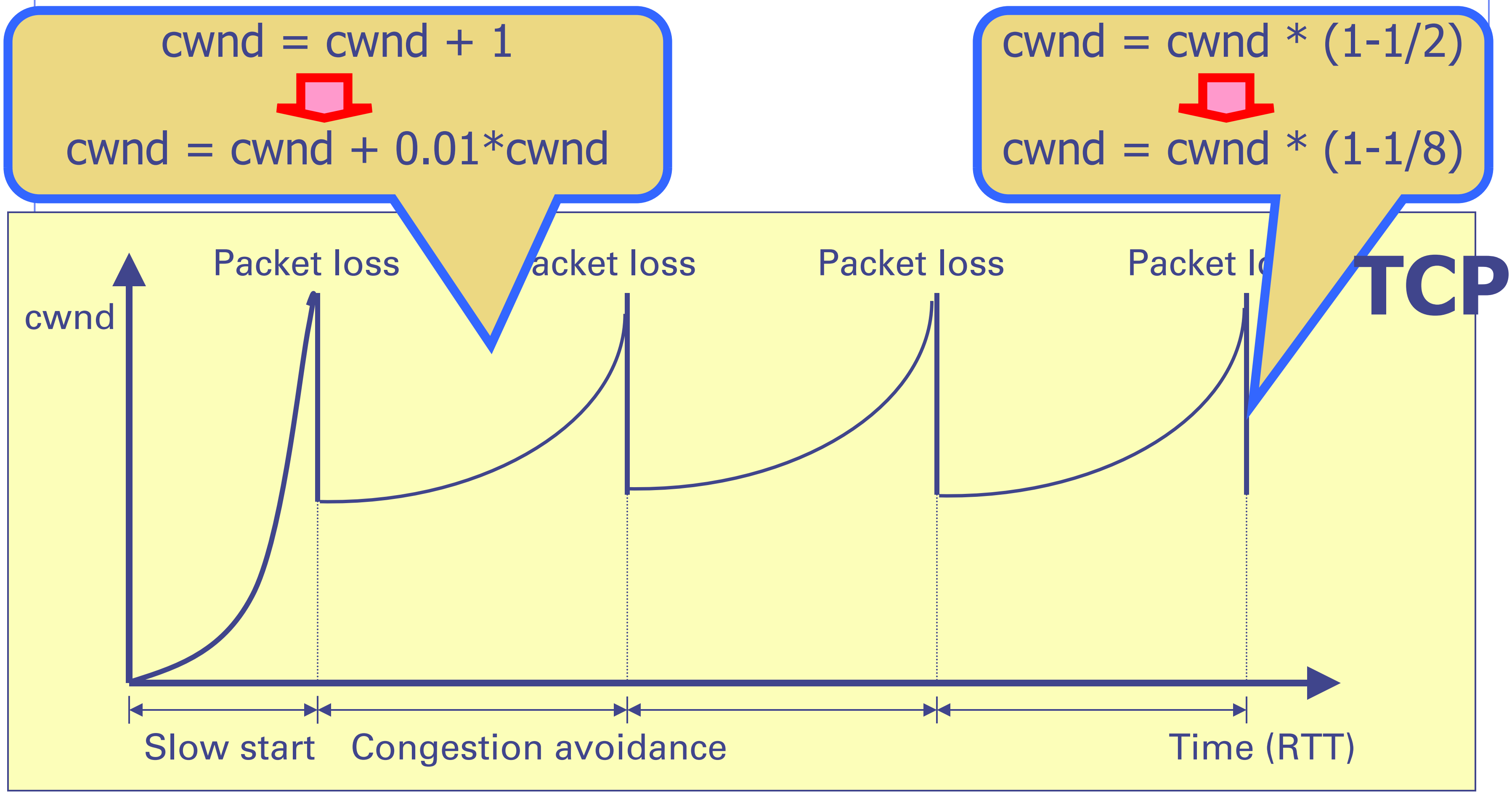
- **TCP-Friendliness**

  The ability to share bandwidth with TCP connections on low-speed networks



STCP (Scalable TCP)

- STCP adaptively increases *cwnd*, and decreases *cwnd* by 1/8.



HSTCP (High Speed TCP)

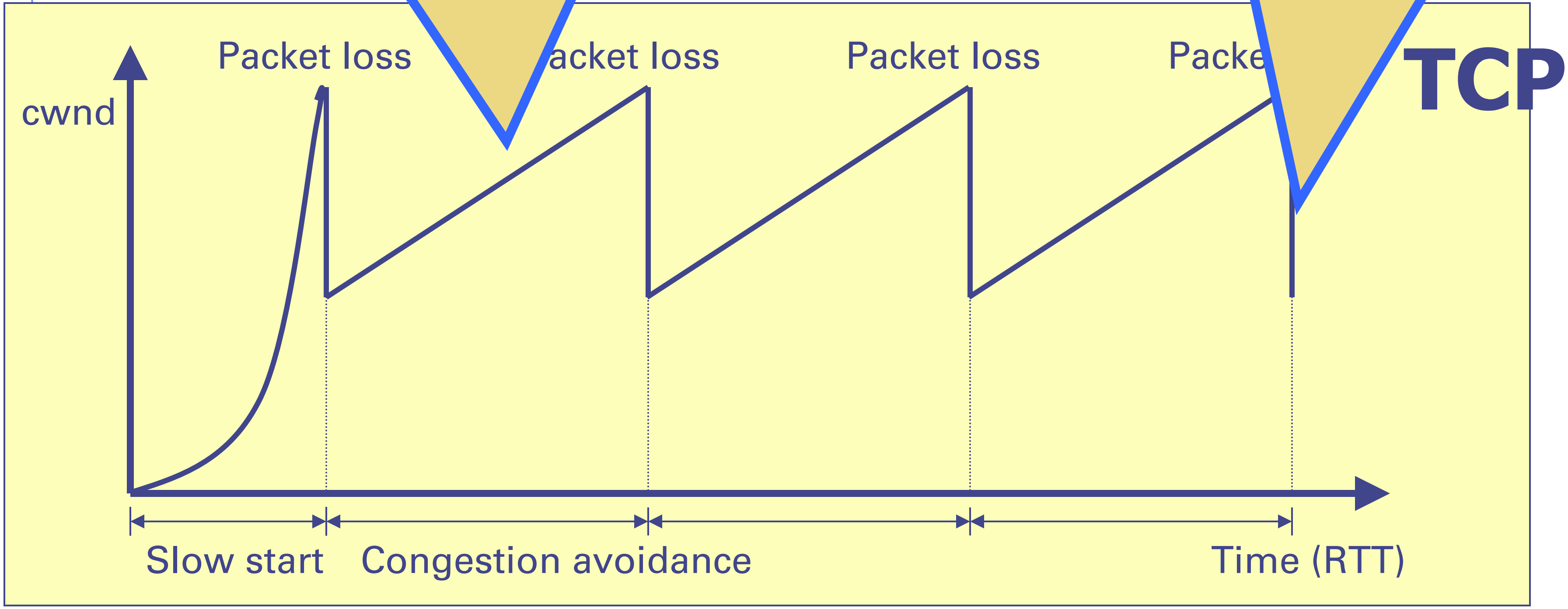
- HSTCP adaptively increases *cwnd*, and adaptively decreases *cwnd*.
- The larger the *cwnd*, the larger the increment, and the smaller the decrement.

cwnd = cwnd + 1

cwnd = cwnd + inc(cwnd)

cwnd = cwnd * (1-1/2)

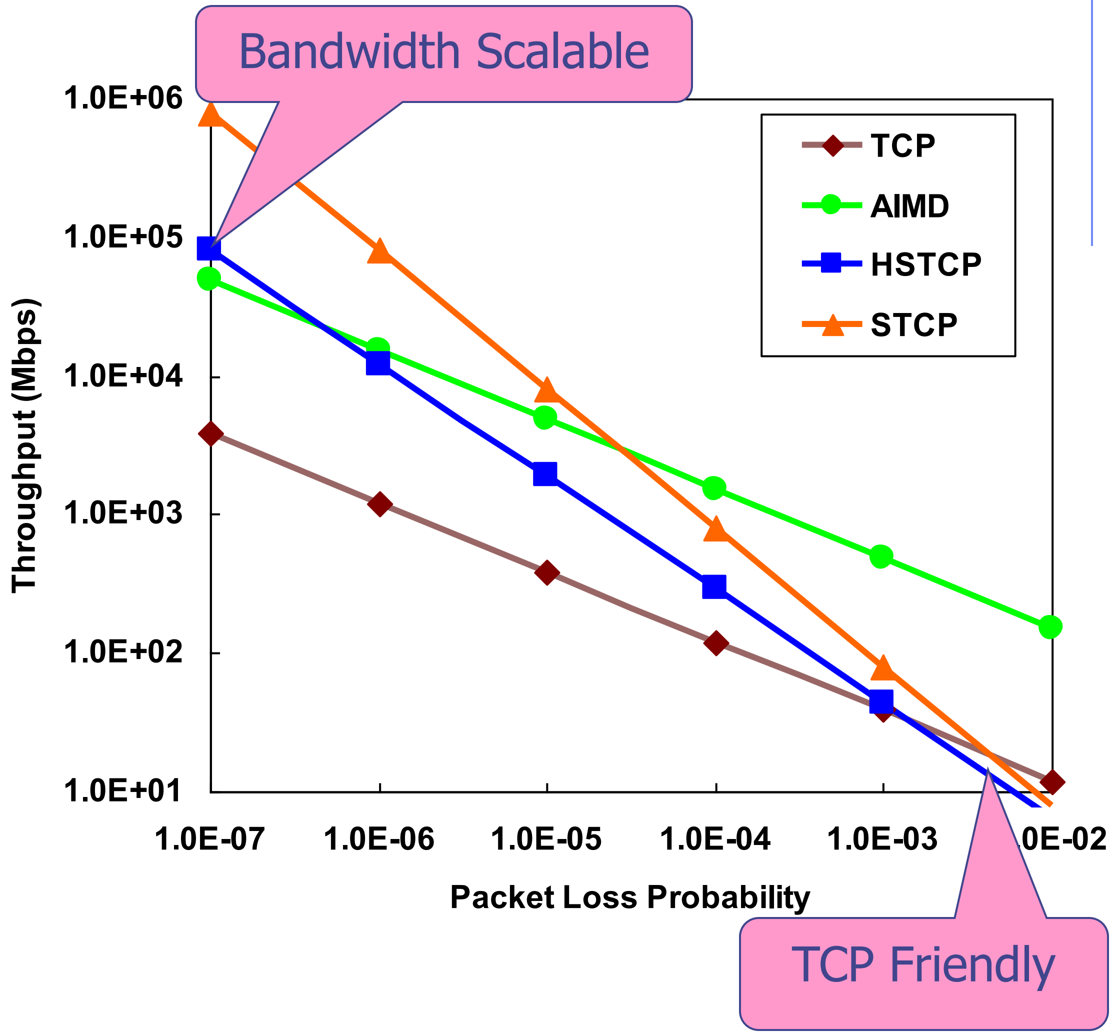
cwnd = cwnd * (1-dec(cwnd))



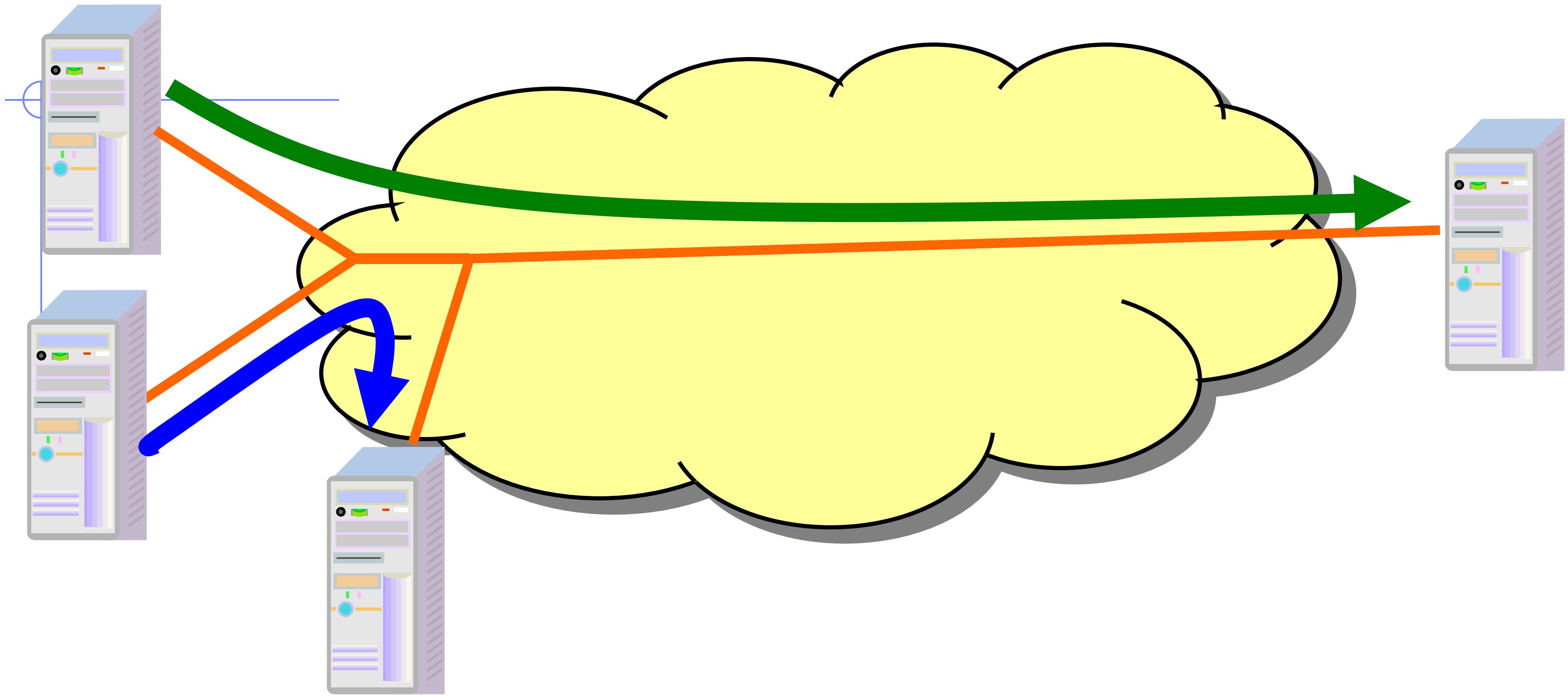
Response Functions of HSTCP and STCP

- HSTCP: $R = \dfrac{MSS}{RTT}\dfrac{0.12}{p^{0.835}}$
- STCP: $R = \dfrac{MSS}{RTT}\dfrac{0.08}{p}$

HSTCP and STCP are both bandwidth scalable and TCP friendly



RTT Fairness



- Different connections may have quite different round-trip times, and a good protocol should allocate bandwidth fairly among those connections
- RTT fairness index = throughout ratio of two flows with different RTTs

RTT Fairness on Low-Speed Networks

- For a protocol with the following response function, where *c* and *d* are protocol-related constants.

$$R = \frac{MSS}{RTT} \frac{c}{p^d}$$

- The RTT Fairness Index (or the throughput ratio of two flows) on low-speed networks is

$$\left(\frac{RTT_2}{RTT_1} \right)$$

- On low speed networks, different protocols have the same RTT fairness

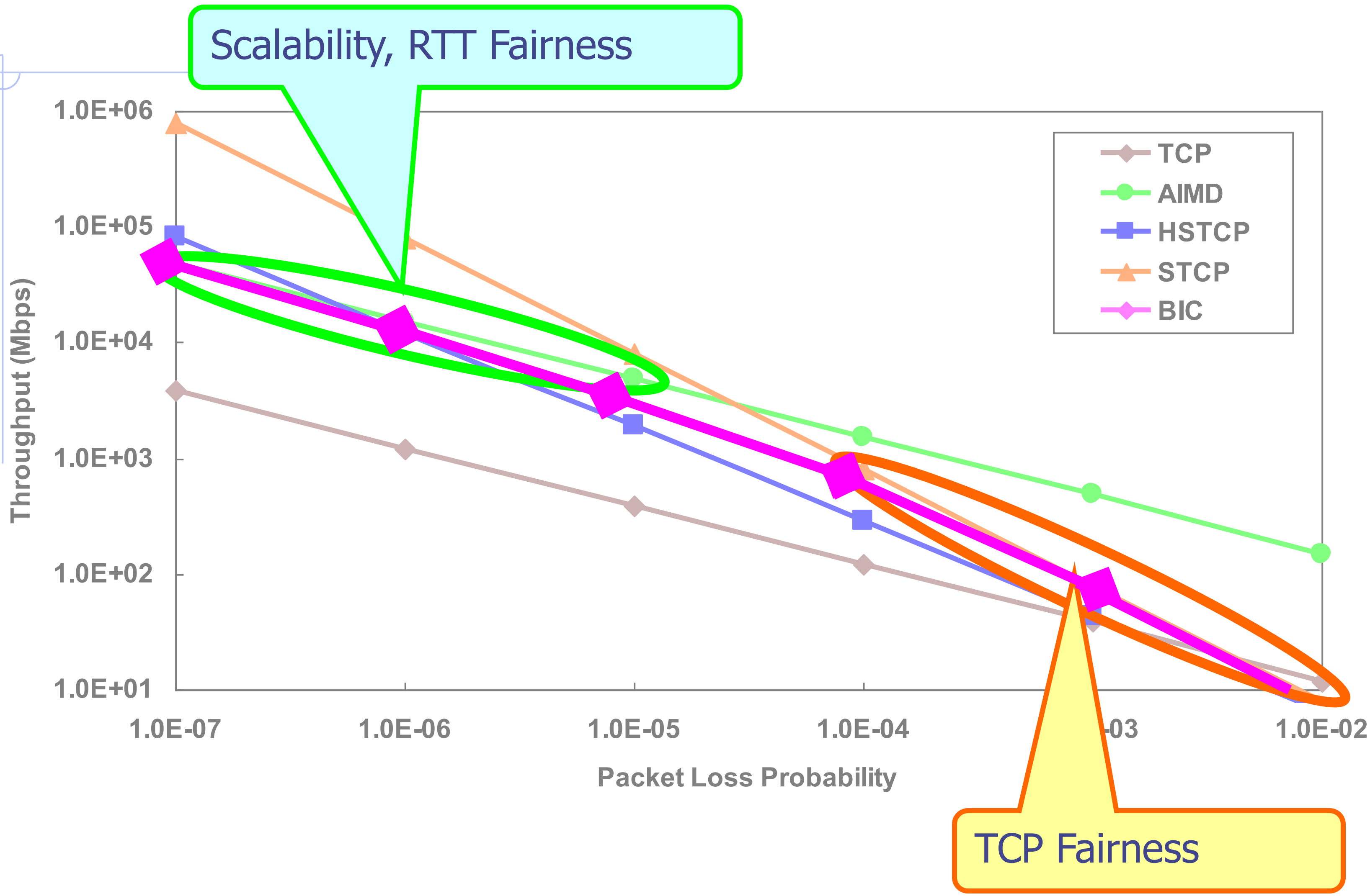
Simulation Results of RTT Fairness

- Throughout ratio of two flows on a 2.5Gbps Link

Inverse RTT Ratio	1	3	6
AIMD	1	6	22
HSTCP	1	29	107
STCP	1	127	389

Simulation setup: BDP Buffer, Drop Tail, Reverse Traffic, Forward Background Traffic (short-lived TCP, Web Traffic)

Design Goal



Scalability, RTT Fairness
1.0E+06
1.0E+05
1.0E+04
1.0E+03
1.0E+02
1.0E+01
Throughput (Mbps)
1.0E-07
1.0E-06
1.0E-05
1.0E-04
1.0E-02
Packet Loss Probability
TCP
AIMD
HSTCP
STCP
BIC
TCP Fairness

Linux CUBIC Protocol

BIC (Binary Increase Congestion control)

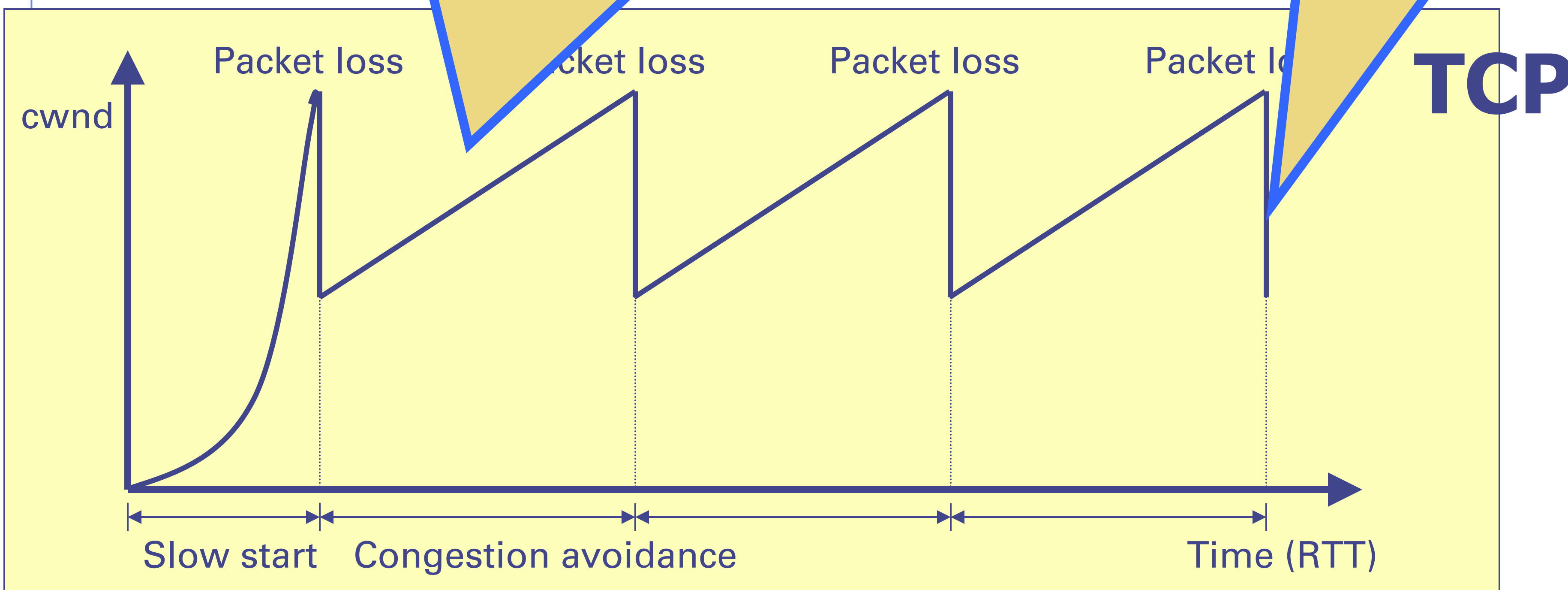
- BIC adaptively increase cwnd, and decrease cwnd by 1/8

cwnd = cwnd + 1

cwnd = cwnd + f(cwnd, history)

cwnd = cwnd * (1-1/2)

cwnd = cwnd * (1-1/8)



A Search Problem

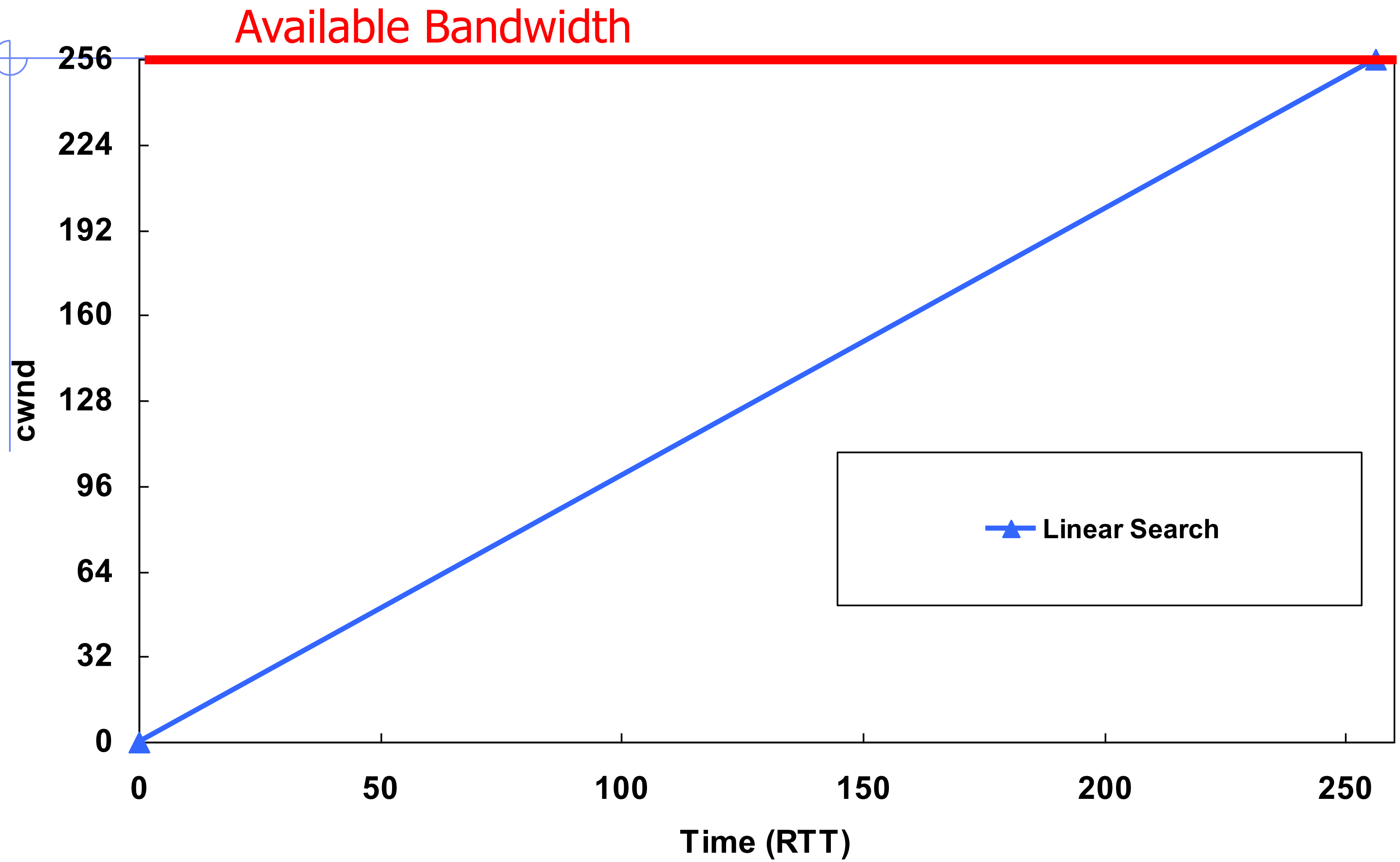
- A Search Problem
  - We consider the increase part of congestion avoidance as a search problem, in which a connection looks for the available bandwidth by comparing its current throughput with the available bandwidth, and adjusting cwnd accordingly.

- Q: How to compare R with A?
  - R = current throughput
  - = cwnd/RTT
  - A = available bandwidth
- A: Check for packet losses
  - No packet loss: R <= A
  - Packet losses : R > A

- How does TCP find the available bandwidth?
- Linear search

```
while (no packet loss){
    cwnd++;
}
```

Linear Search



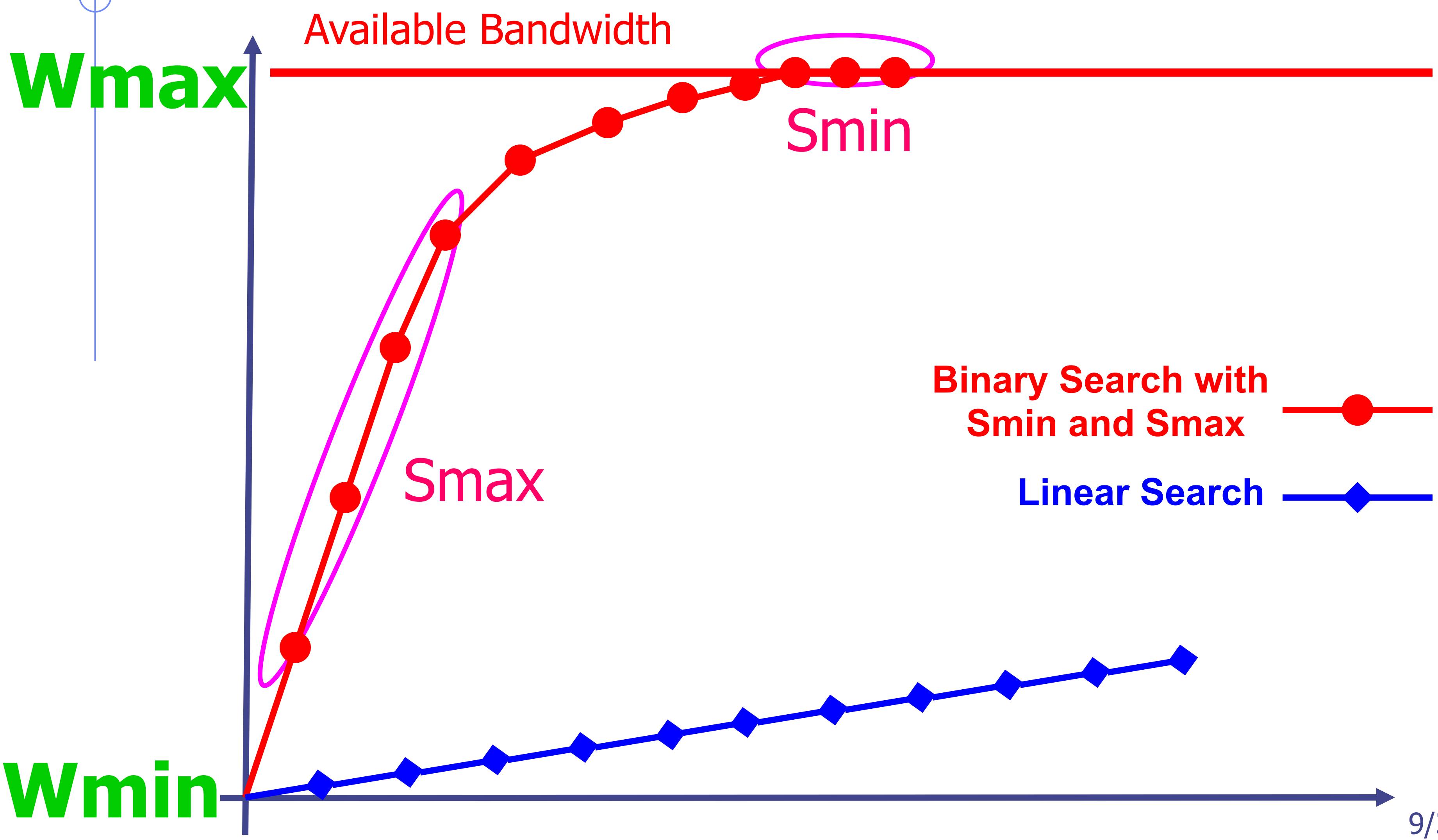
BIC: Binary Search with Smax and Smin

- BIC - Binary search

```
while (Wmin <= Wmax){
    inc = (Wmin+Wmax)/2 - cwnd;
    if (inc > Smax)
            inc = Smax;
    else if (inc < Smin)
            inc = Smin;
    cwnd = cwnd + inc;
    if (no packet losses)
            Wmin = cwnd;
    else
            break;
}
```

- Wmax: Max Window
- Wmin: Min Window
- Smax: Max Increment
- Smin: Min Increment

BIC-TCP
Binary Search with Smax and Smin

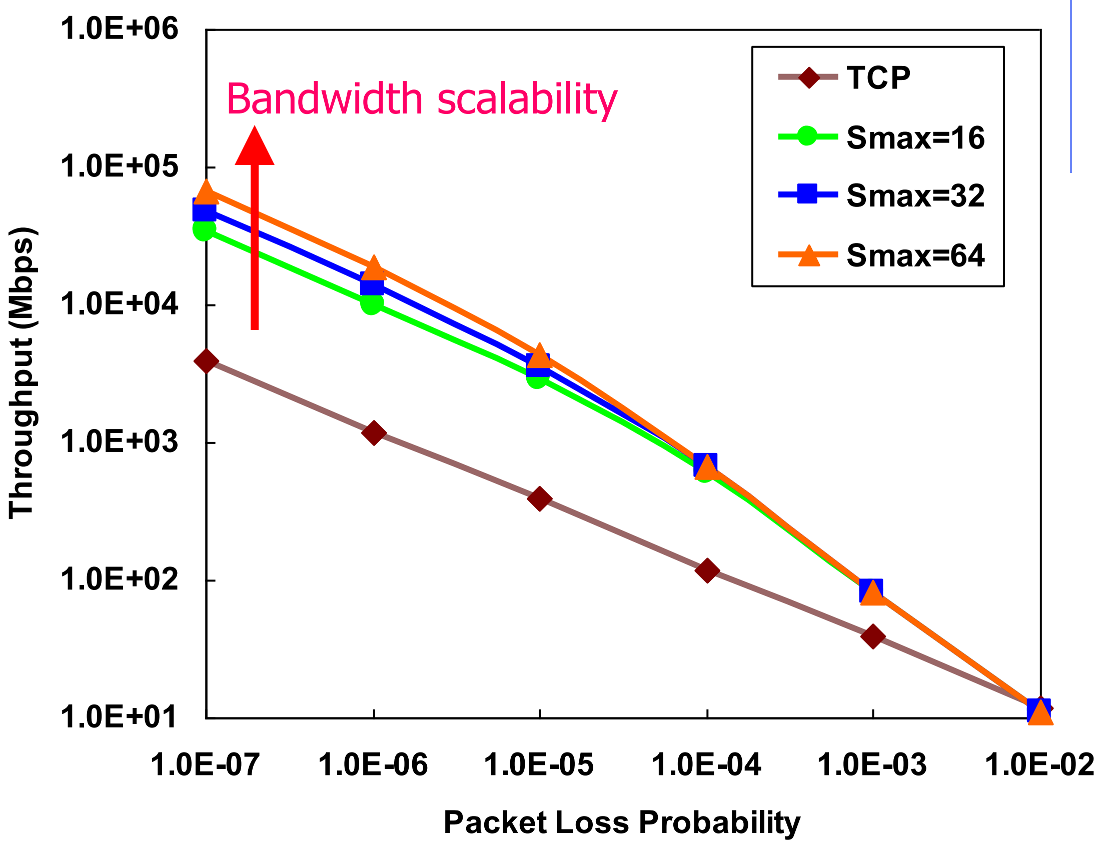


Setting Smax

- Response Function of BIC on high-speed networks

$$R = \frac{MSS}{RTT} \frac{2.7\sqrt{S_{\max}}}{p^{0.5}}$$

- Bandwidth scalability of BIC depends only on Smax
- RTT Fairness of BIC on high-speed networks is the same as that of AIMD

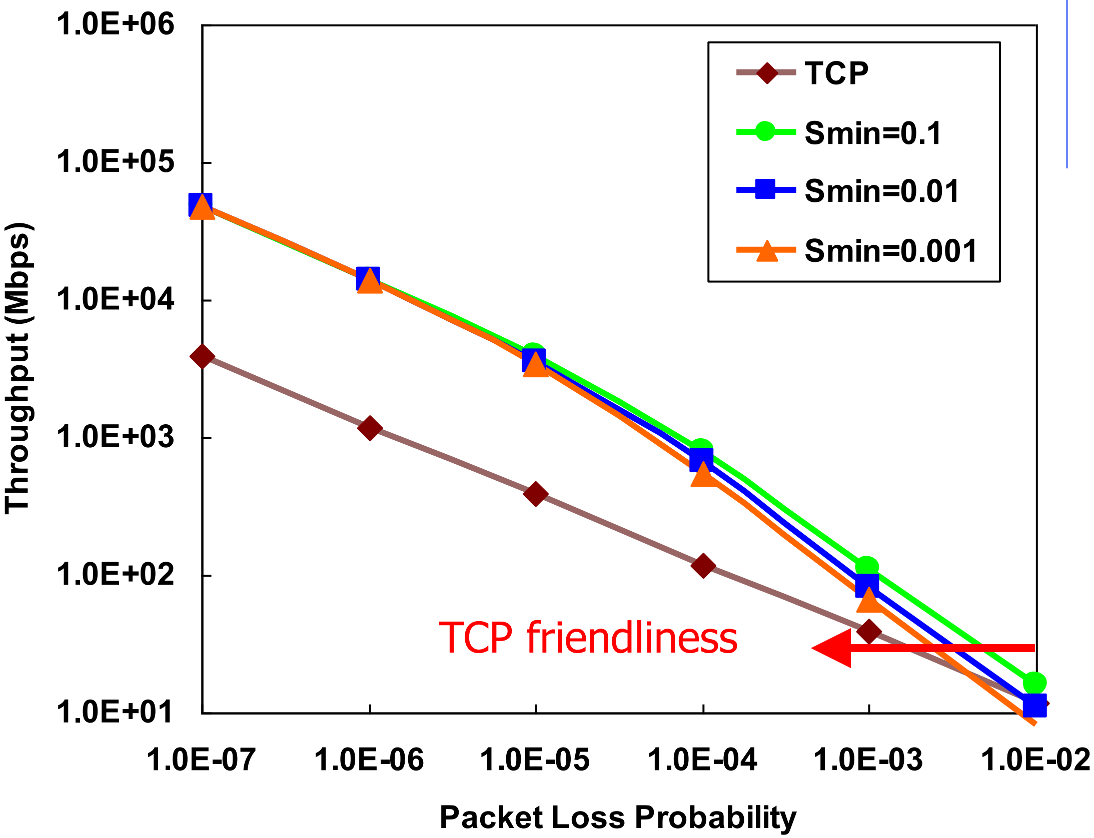


Setting Smin

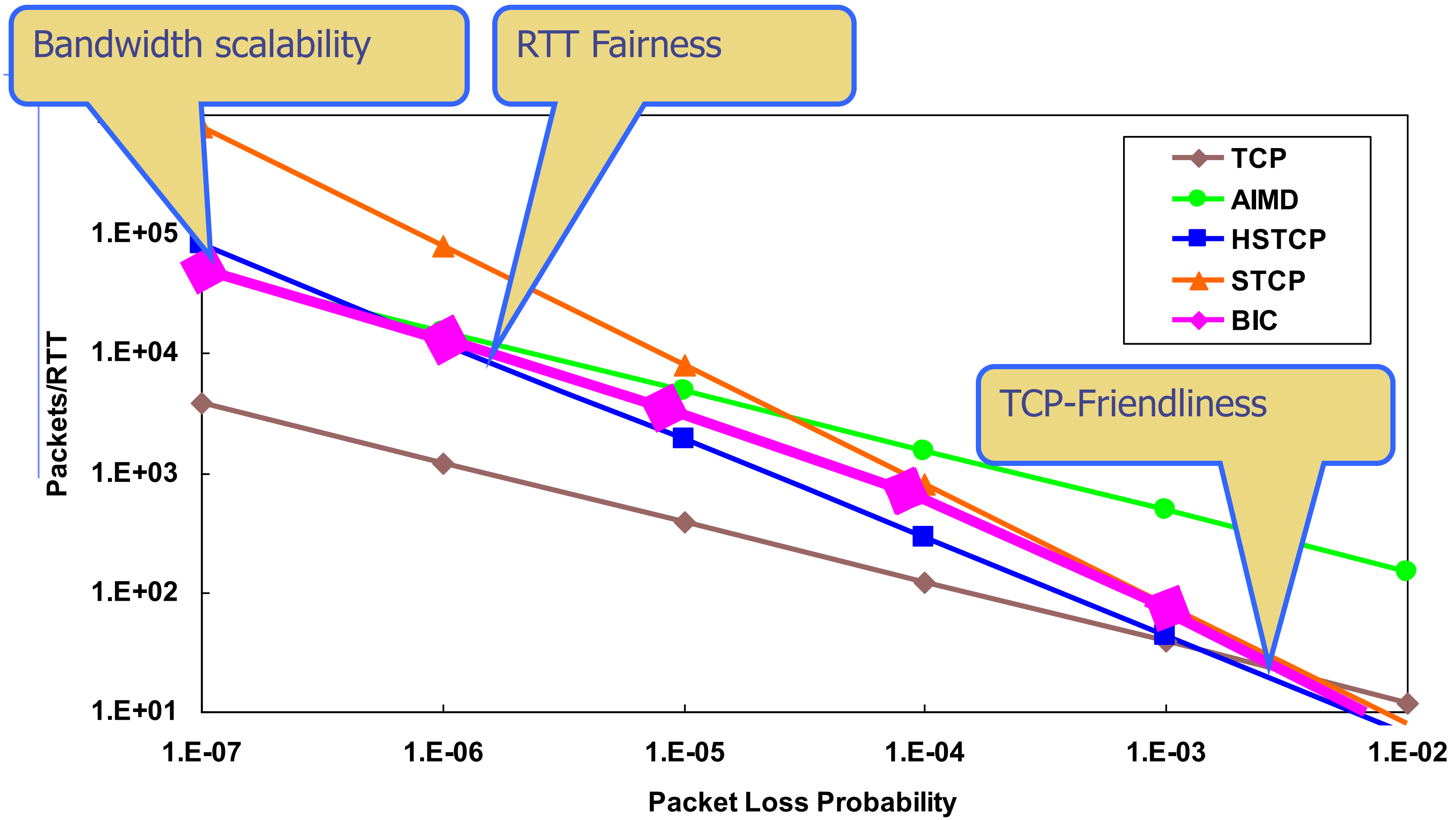
- Response Function of BIC on low-speed networks

$$R = \frac{MSS}{RTT} f\left(p, S_{\min}\right)$$

- TCP-friendliness of BIC depends only on Smin



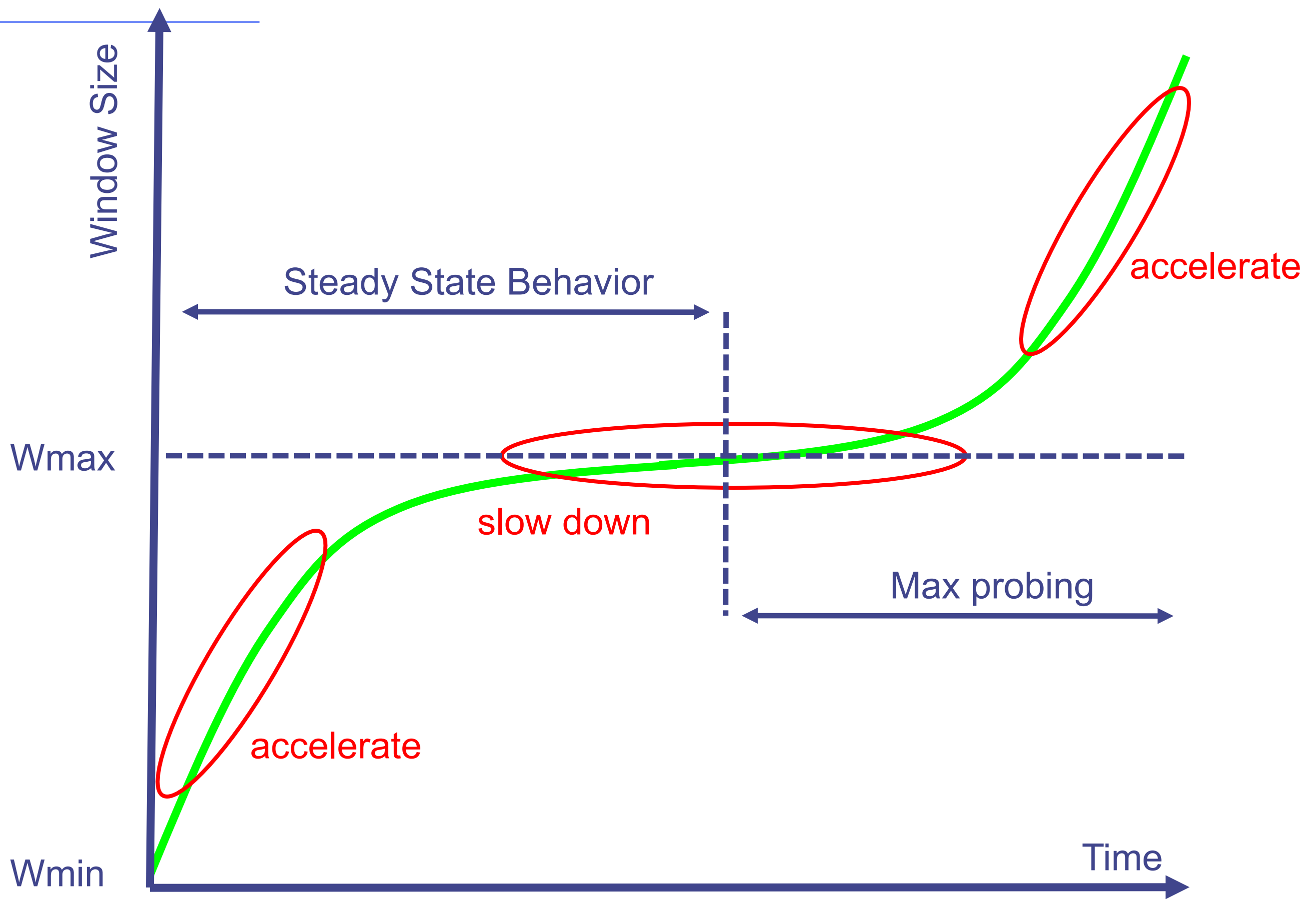
Response Functions



CUBIC – A new TCP variant

- Why a new protocol?
  - While the window growth of new TCP protocols is scalable, their fairness issue has remained as a major challenge .
  - BIC-TCP shows good utilization and stability, but it lacks TCP friendliness and RTT fairness.
- CUBIC is an enhanced version of BIC
  - Simplifies the BIC window control using a cubic function.
  - Improves its TCP friendliness & RTT fairness.
  - The window growth function of CUBIC is based on real-time (the elapsed time since the last loss event), so that it is independent of RTT.

CUBIC function



$$W_{cubic} = C(t-K)^3 + W_{max} \qquad K = \sqrt[3]{W_{max}\beta/C}$$

where ***C*** is a scaling factor, ***t*** is the elapsed time from the last window reduction, and ***β*** is a constant multiplication decrease factor.

CUBIC – New TCP Mode

- In short RTT networks, the window growth of CUBIC is slower than TCP since CUBIC is independent of RTT. We emulate the TCP window algorithm after a packet loss event.

Average sending rate of AIMD = $\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\frac{1}{p}}$

$\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}}$ (TCP). Thus, $\alpha = 3 \times \frac{1-\beta}{1+\beta}$

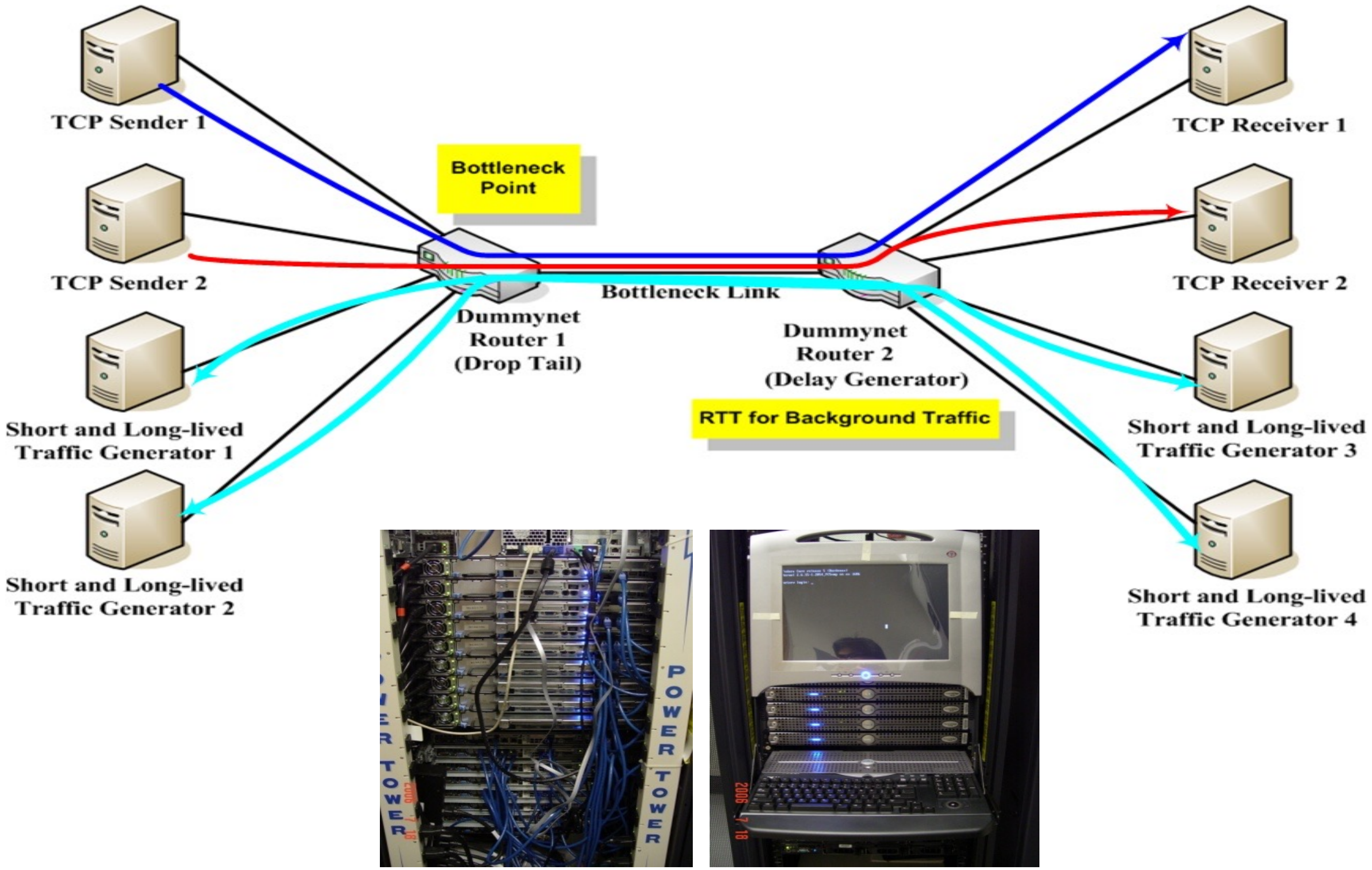
$$W_{tcp} = W_{\max}\beta + 3\frac{1-\beta}{1+\beta}\frac{t}{RTT}$$

The size of TCP window after time t from window reduction.

if W_{tcp} > W_{cubic} : window size = W_{tcp}

Otherwise : window size = W_{cubic}

Experimental Testbed

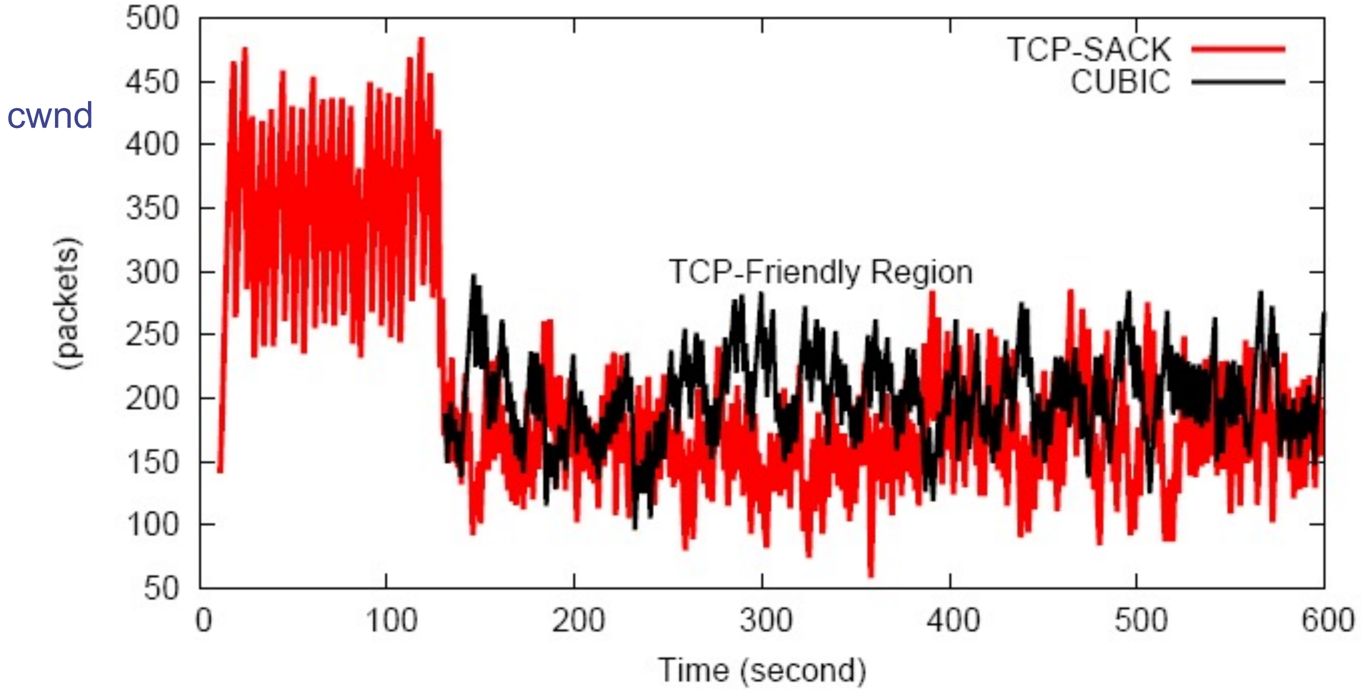


Testbed Setup : Background Traffic Generation

- We use the Internet measurement studies, which have shown complex behaviors and characteristics of Internet traffic.
- TCP RTTs observed in edge routers
  - The exponential mean is set to 66ms (one-way delay), then the CDF is very similar to the CDF of RTT samples shown in the paper, "Variability in TCP Round-trip Times" by J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay in SIGCOMM IMC, 2003.
- Inter-arrival time between two successive TCP connections: Exponential distribution (observed from Floyd and Paxson).
- Short-lived flows (web traffic flows) follow Lognormal (Body) and Pareto (Tail) Distribution.
  - Using the parameters from the paper "Generating Representative Web Workloads for Network and Server Performance Evaluation" by Paul Barford, Mark Crovella in SigMetric 1998.

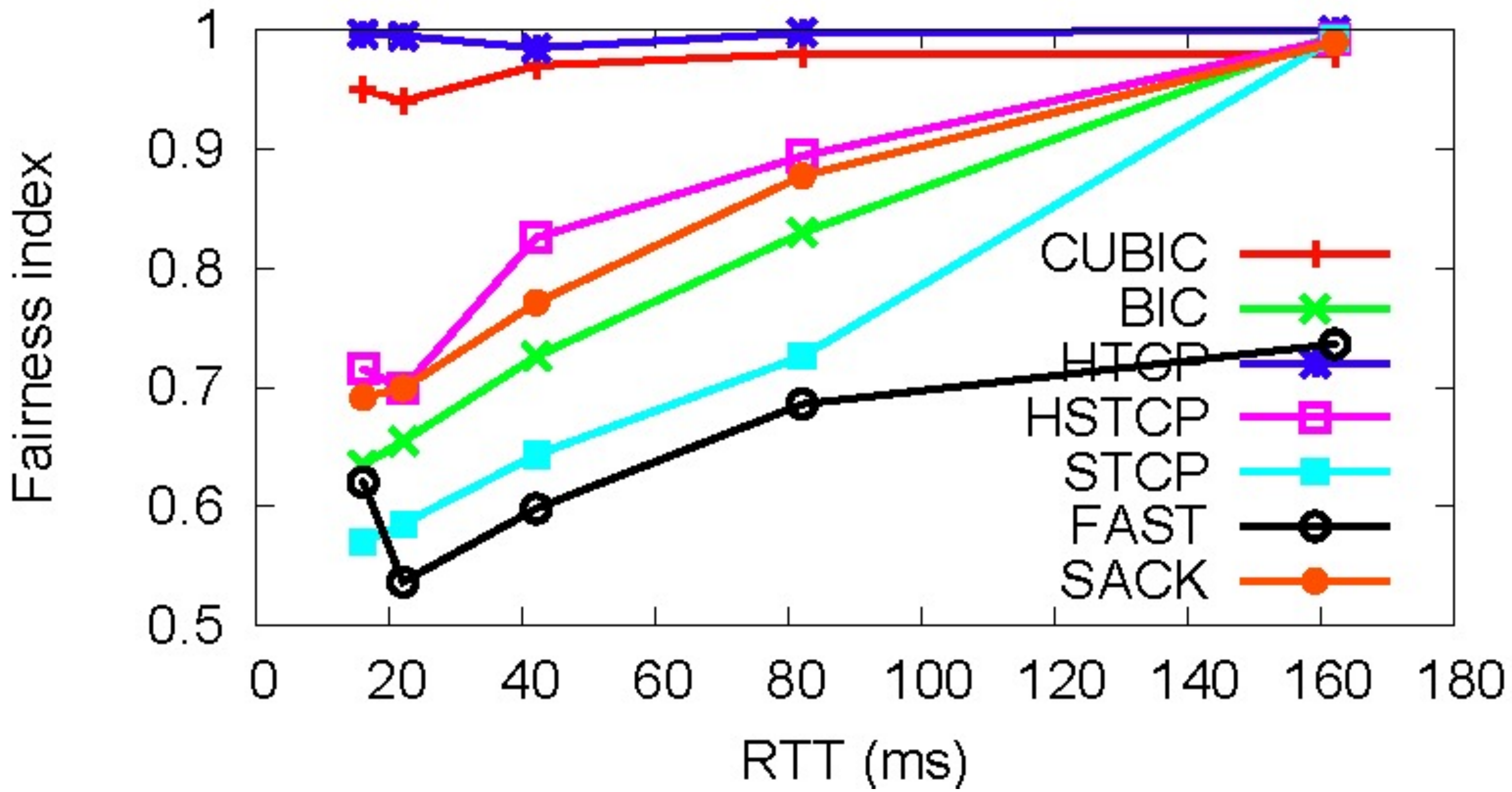
TCP Friendliness

- Dummynet Testbed: 400Mbps, RTT 10ms, 100% router buffer, and moderate background traffic



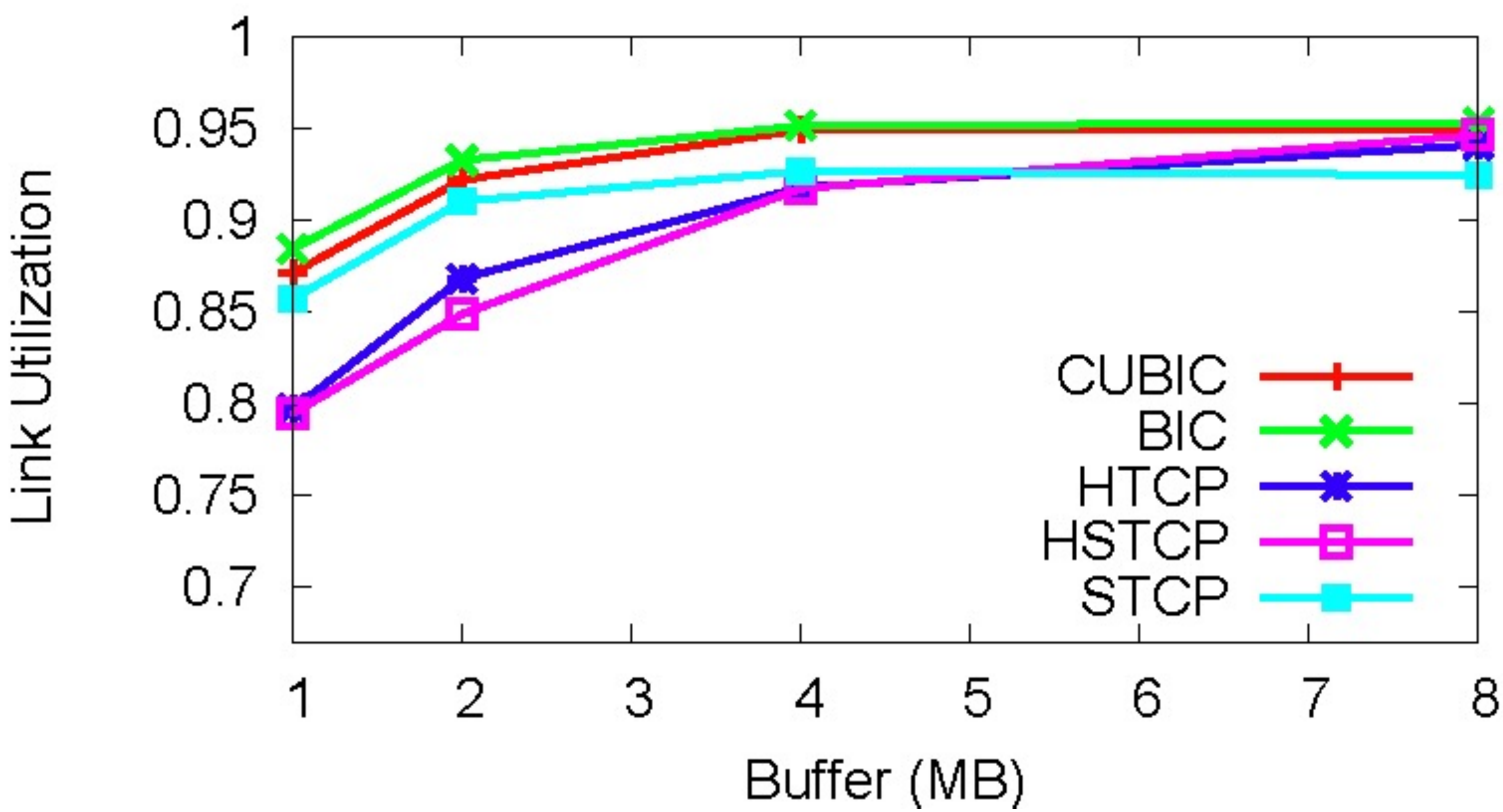
RTT Fairness

- Dummynet Testbed: 400Mbps, 1MB buffer size, background traffic

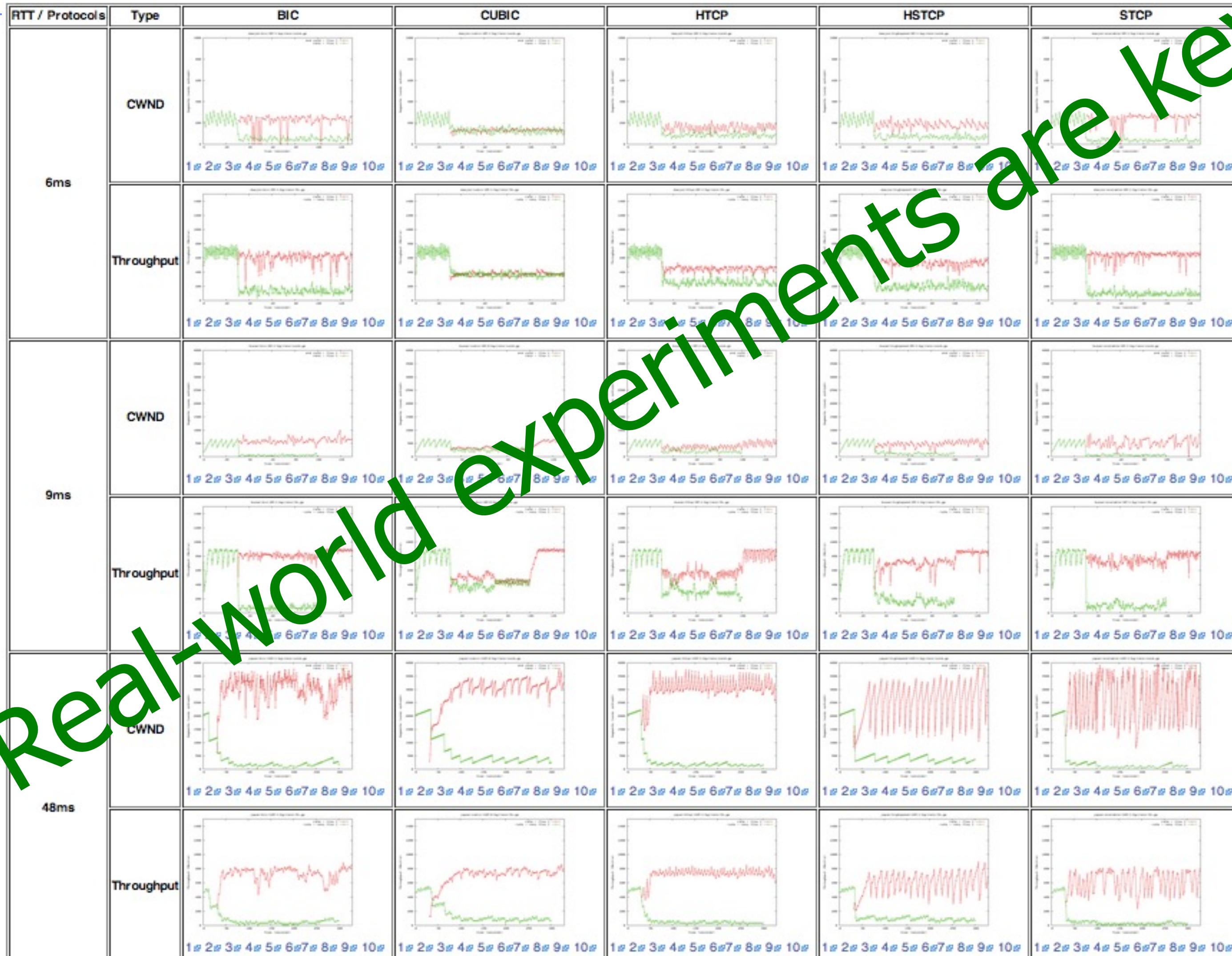


Link Utilization

- Dummynet Testbed : 400Mbps, 4 high-speed TCP flows, background traffic, and vary the buffer size from 1MB to 8MB



Internet Experiment



Real-world experiments are key!

History of BIC/CUBIC in Linux

- BIC had been the default TCP congestion control algorithm from 2004 to 2006.
- CUBIC replaced BIC and has been the default TCP congestion control algorithm in Linux since 2006, thanks to its improved fairness while retaining the strength of BIC-TCP.
- CUBIC has improved its growth functions several times for scalability and stability, based on the feedback from the researchers and users.
- Since 2008, HyStart has been part of CUBIC, which improves the throughput of CUBIC especially for high-BDP networks, by preventing SACK processing overhead in Linux.

Linux CUBIC Implementation

- Practical considerations
  - Computation overhead of cubic root calculation has been significantly improved.
  - Original implementation with bisection method took 1032 clocks.
  - Newton-Raphson method with table lookups for small values took 79 clocks.

13 times improvement

cubic_update(): .... **(3.2)**

begin

$ack\_cnt \longleftarrow ack\_cnt + 1$

if $epoch\_start \leq 0$ **then**

$epoch\_start \longleftarrow tcp\_time\_stamp$

if $cwnd < W_{last\_max}$ **then**

$K \longleftarrow \sqrt[3]{\frac{W_{last\_max} - cwnd}{C}}$

$origin\_point \longleftarrow W_{last\_max}$

else

$K \longleftarrow 0$

$origin\_point \longleftarrow cwnd$

$ack\_cnt \longleftarrow 1$

$W_{tcp} \longleftarrow cwnd$

$\longleftarrow tcp\_time\_stamp + dMin - epoch\_start$

$origin\_point + C(t - K)^3$

$> cwnd$ **then** $cnt \longleftarrow \frac{cwnd}{target - cwnd}$ .. **(3.4,3.5)**

$\longleftarrow 100 * cwnd$

cp_friendliness **then** $cubic\_tcp\_friendliness()$

end

cubic_tcp_friendliness(): .... **(3.3)**

begin

$W_{tcp} \longleftarrow W_{tcp} + \frac{3\beta}{2-\beta} * \frac{ack\_cnt}{cwnd}$

$ack\_cnt \longleftarrow 0$

if $W_{tcp} > cwnd$ **then**

$max\_cnt \longleftarrow \frac{cwnd}{W_{tcp} - cwnd}$

if $cnt > max\_cnt$ **then** $cnt \longleftarrow max\_cnt$

end

Lessons Learned

- Go beyond paper – close the loop from theory to practice (it's fun!)
- Get help from domain experts (TCP experts and kernel developers)
- Construct a testbed to evaluate your work with others
- Be ready to defend your work and constantly improve it

Questions?