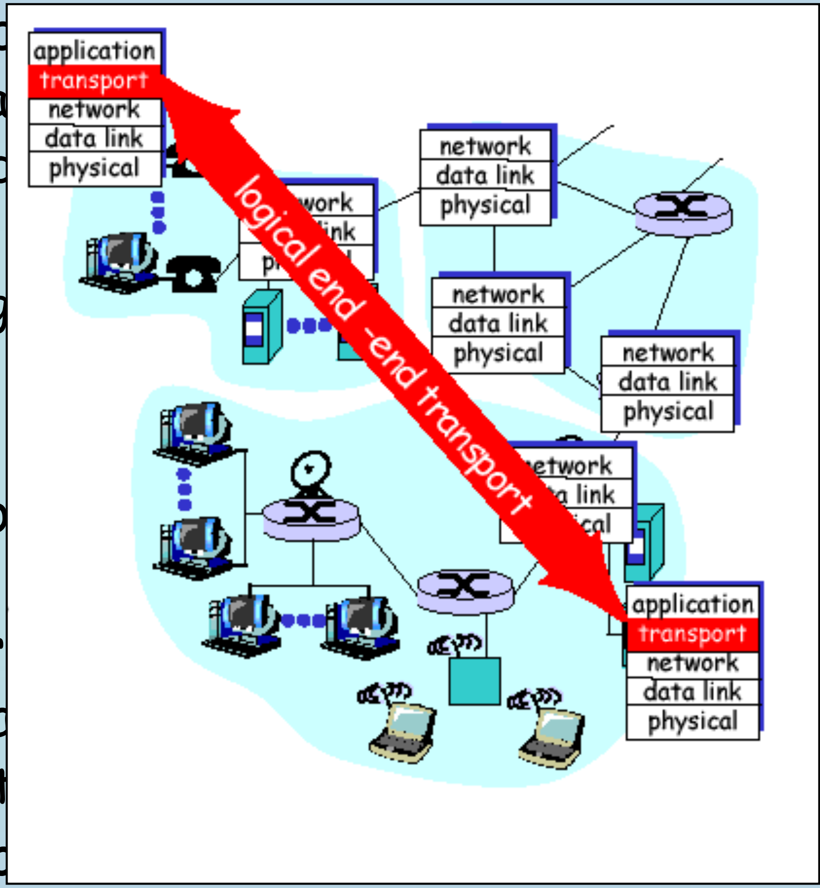
# The dark side of TCP

## understanding TCP on very high-speed networks

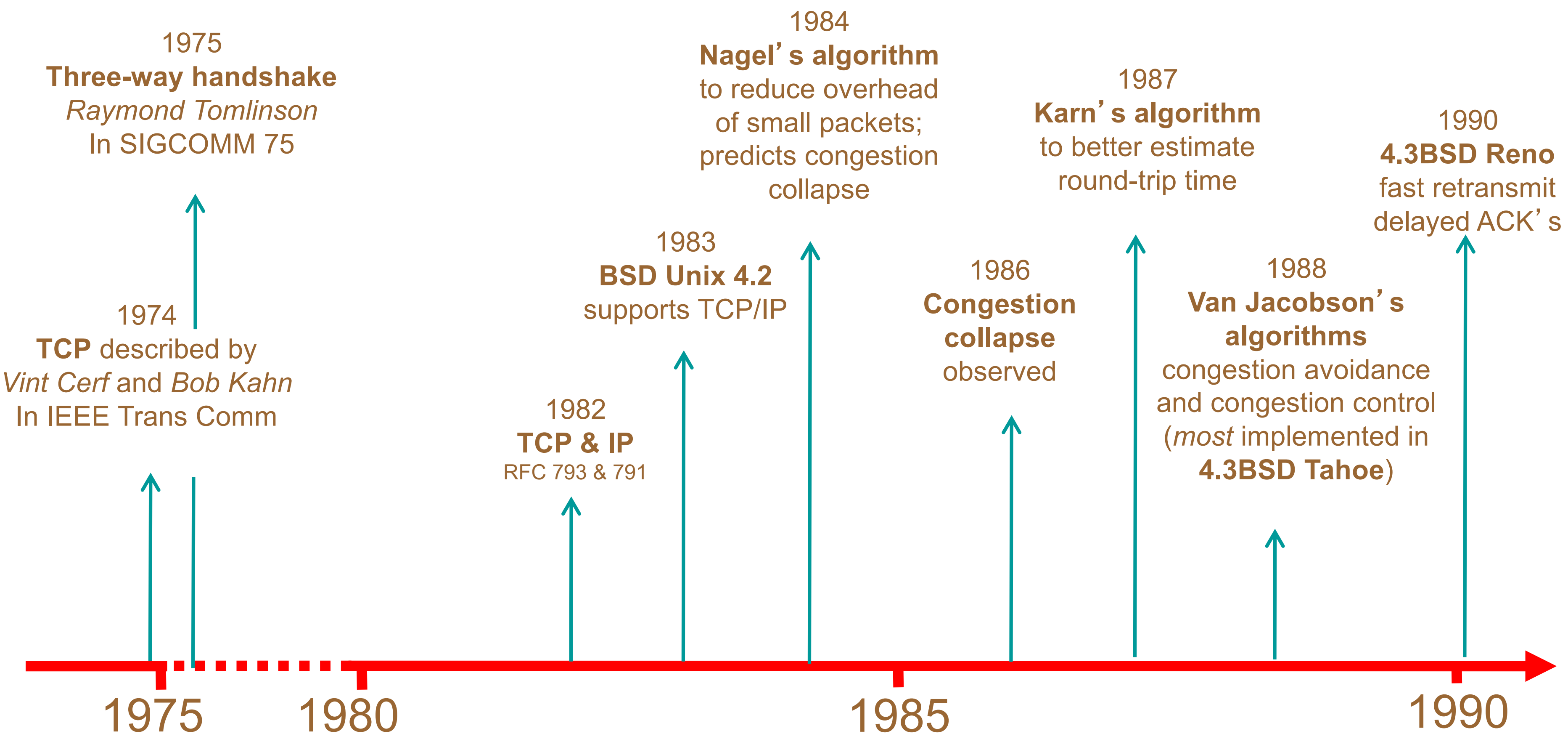
C. Pham
http://www.univ-pau.fr/~cpham
University of Pau, France
LIUPPA laboratory

# What TCP brings

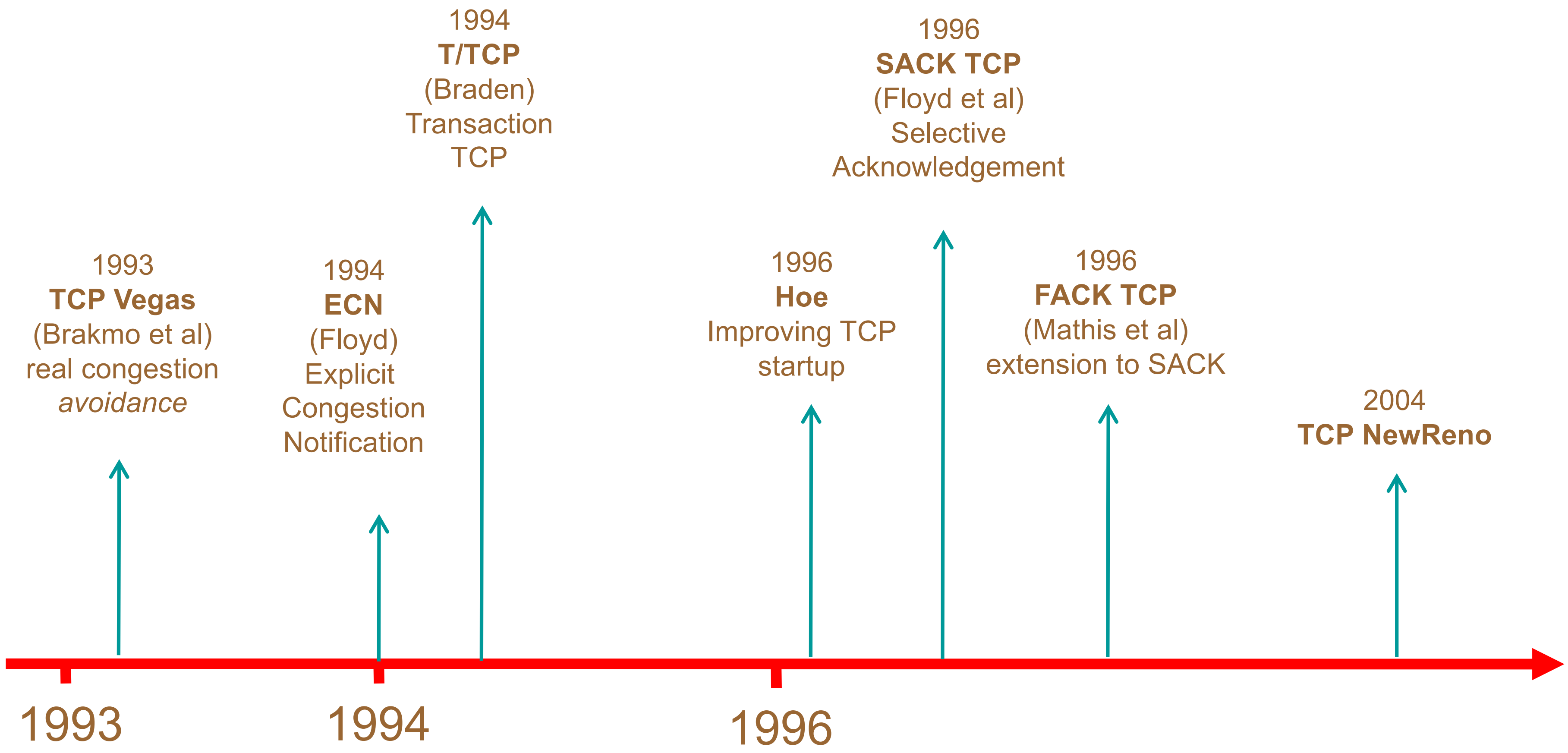
- stream-base
  - segments a … umbers
  - only conse
- reliability
  - missing seg … g) and retransmitted
- flow control
  - receiver is … based)
- congestion co
  - network is … based)
  - protocol tr
- connection h
  - explicit est
- full-duplex c
  - an ACK can be a data segment at the same time (piggybacking)



# A brief history of TCP



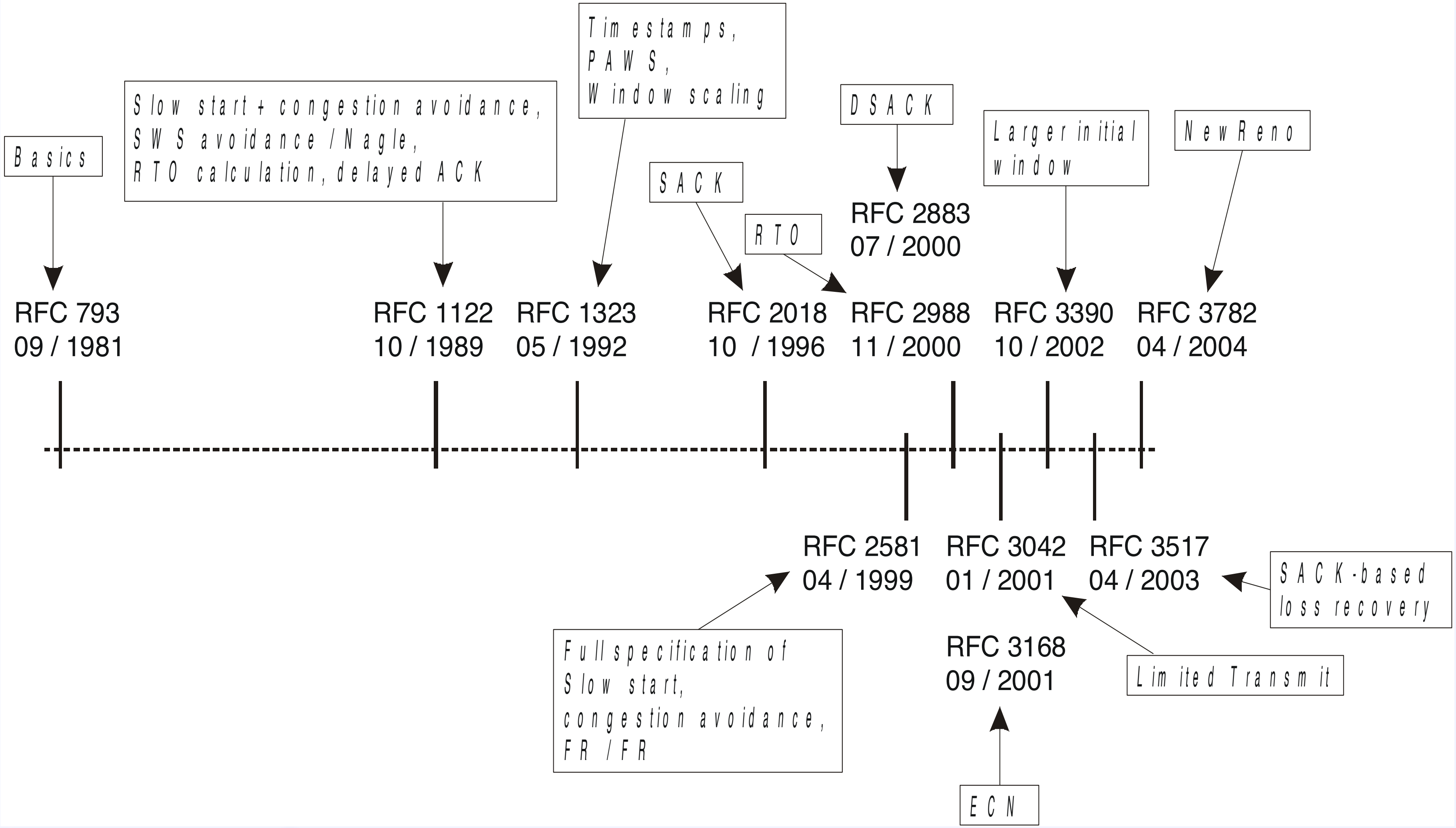
# …in the nineties



From Mickeal Welzt

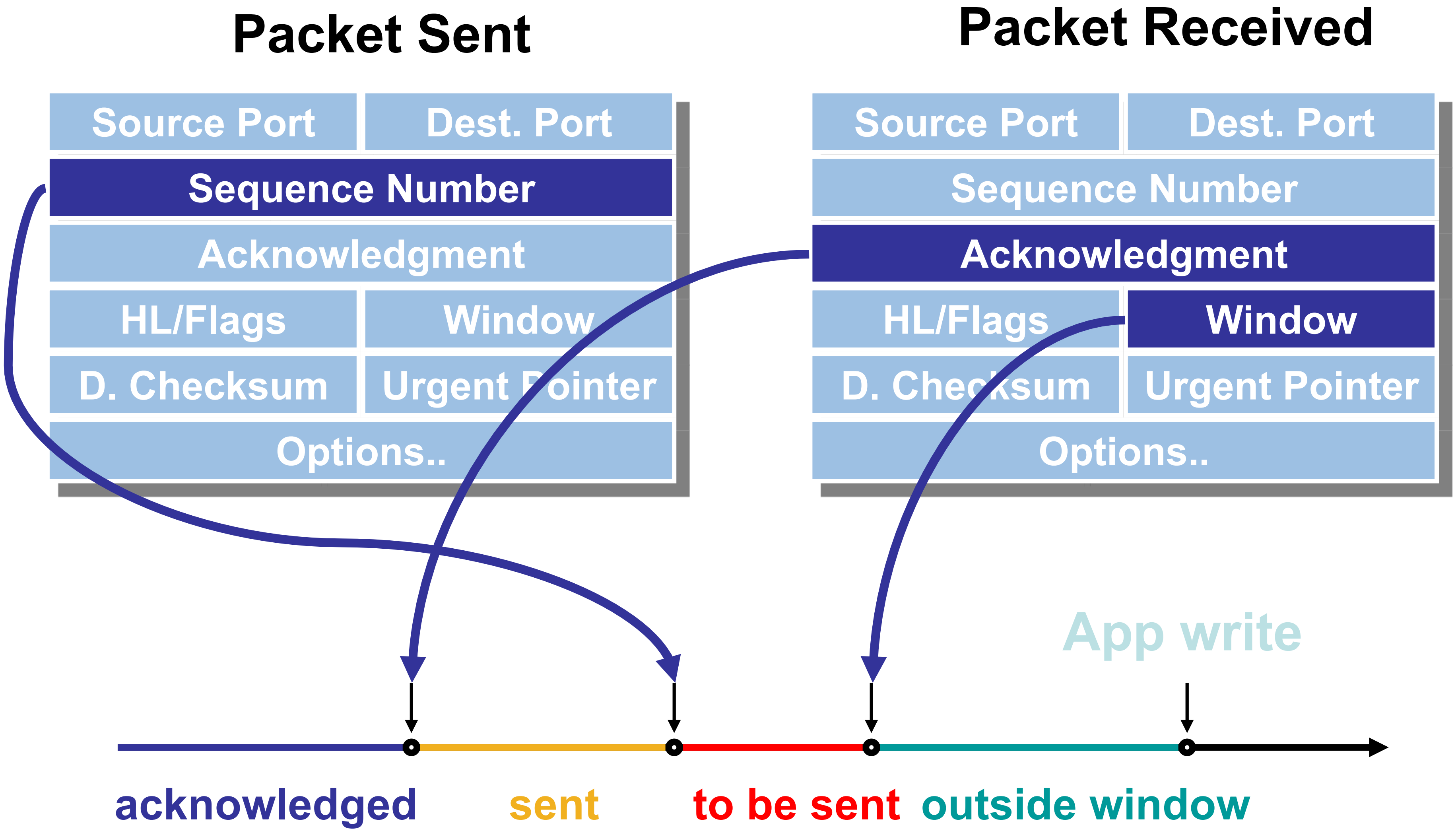
# TCP History in RFC

Standards track TCP RFCs which influence when a packet is sent
(status: early 2005)

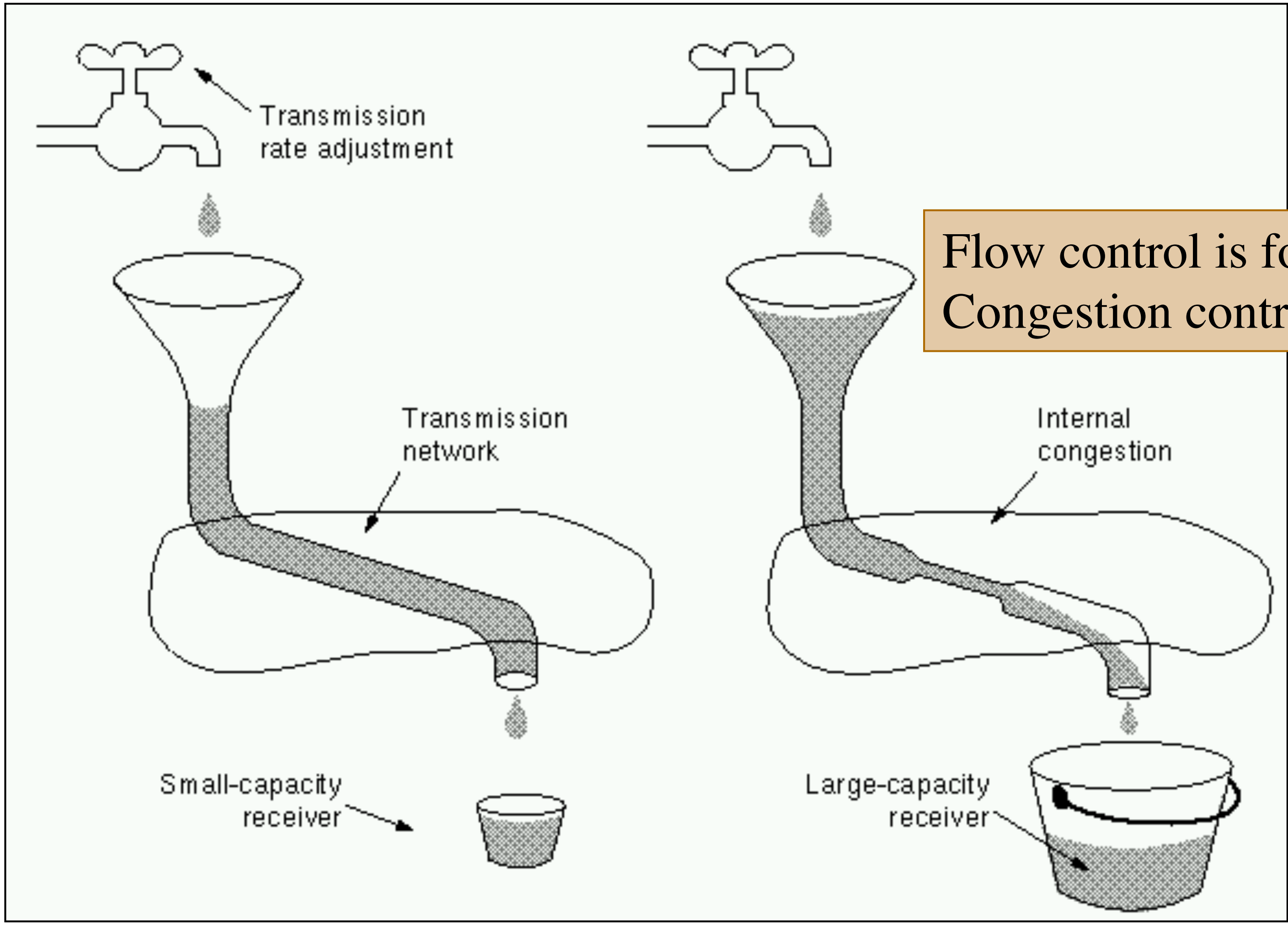


# Flow control

## prevents receiver's buffer overfow



# Congestion control vs flow control




From Computer Networks, A. Tanenbaum

Flow control is for receivers
Congestion control is for the network

Congestion collapse was first observed in 1986 by V. Jacobson. Congestion control was added to TCP (TCP Reno) in 1988.

From Mickeal Welzt

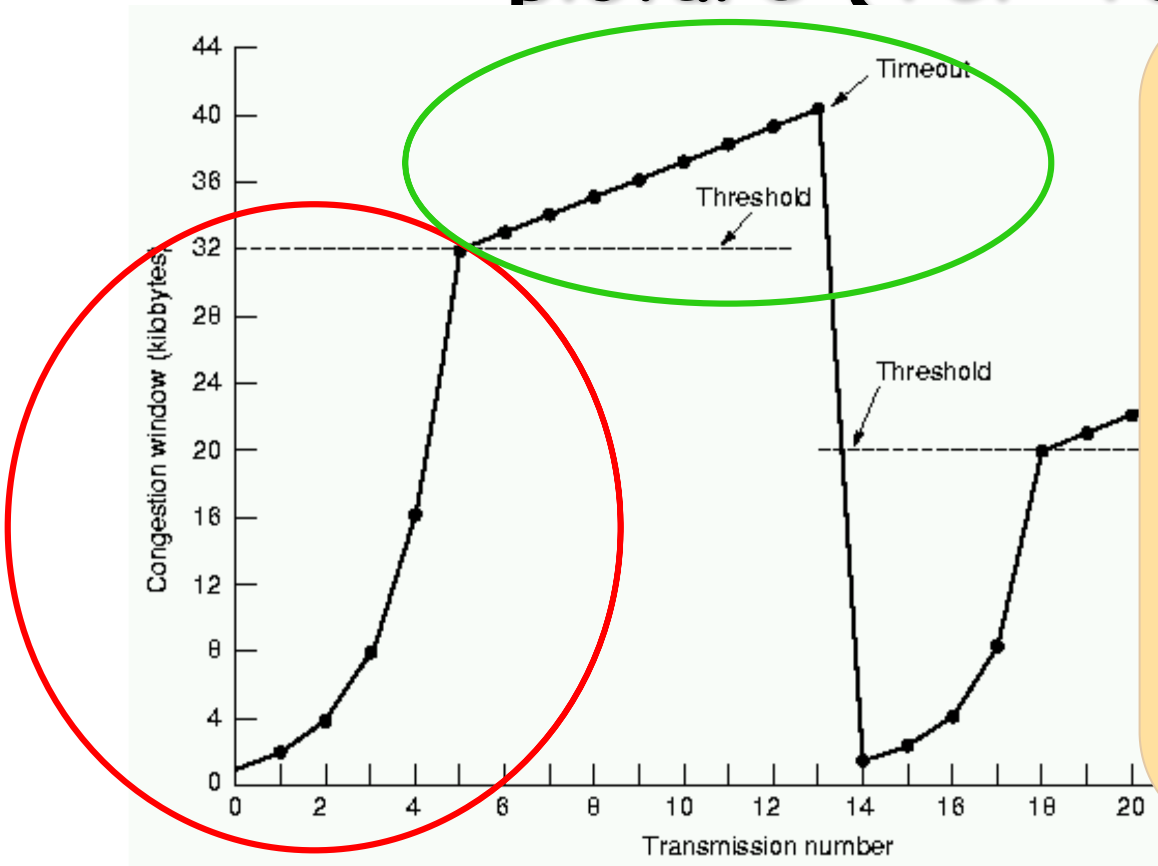
# Internet congestion control: History

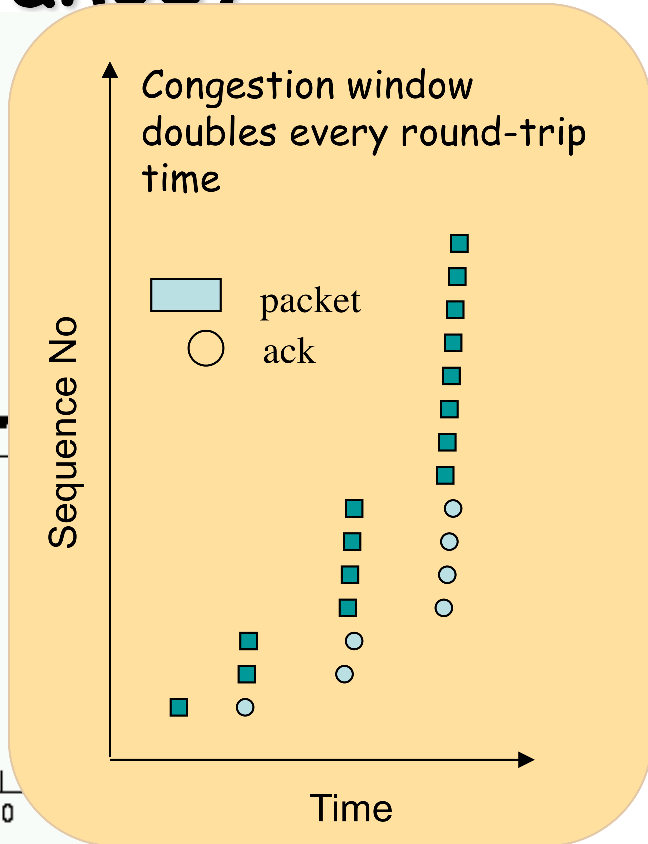
- **1968/69:** dawn of the Internet
- **1986:** first congestion collapse
- **1988:** "Congestion Avoidance and Control" (Jacobson)
  Combined congestion/flow control for TCP
  (also: variation change to RTO calculation algorithm)

- Goal: stability - in equilibrum, no packet is sent into the network until an old packet leaves
  - ack clocking, "conservation of packets" principle
  - made possible through window based stop+go - behaviour

- Superposition of stable systems = stable → network based on TCP with congestion control = stable

# TCP congestion control: the big picture (TCP Tahoe)




- cwnd grows exponentially (slow start), then linearly (*congestion avoidance*) with 1 more segment per RTT
- If loss, divides threshold by 2 (multiplicative decrease) and restart with cwnd=1 packet

From Mickeal Welzt

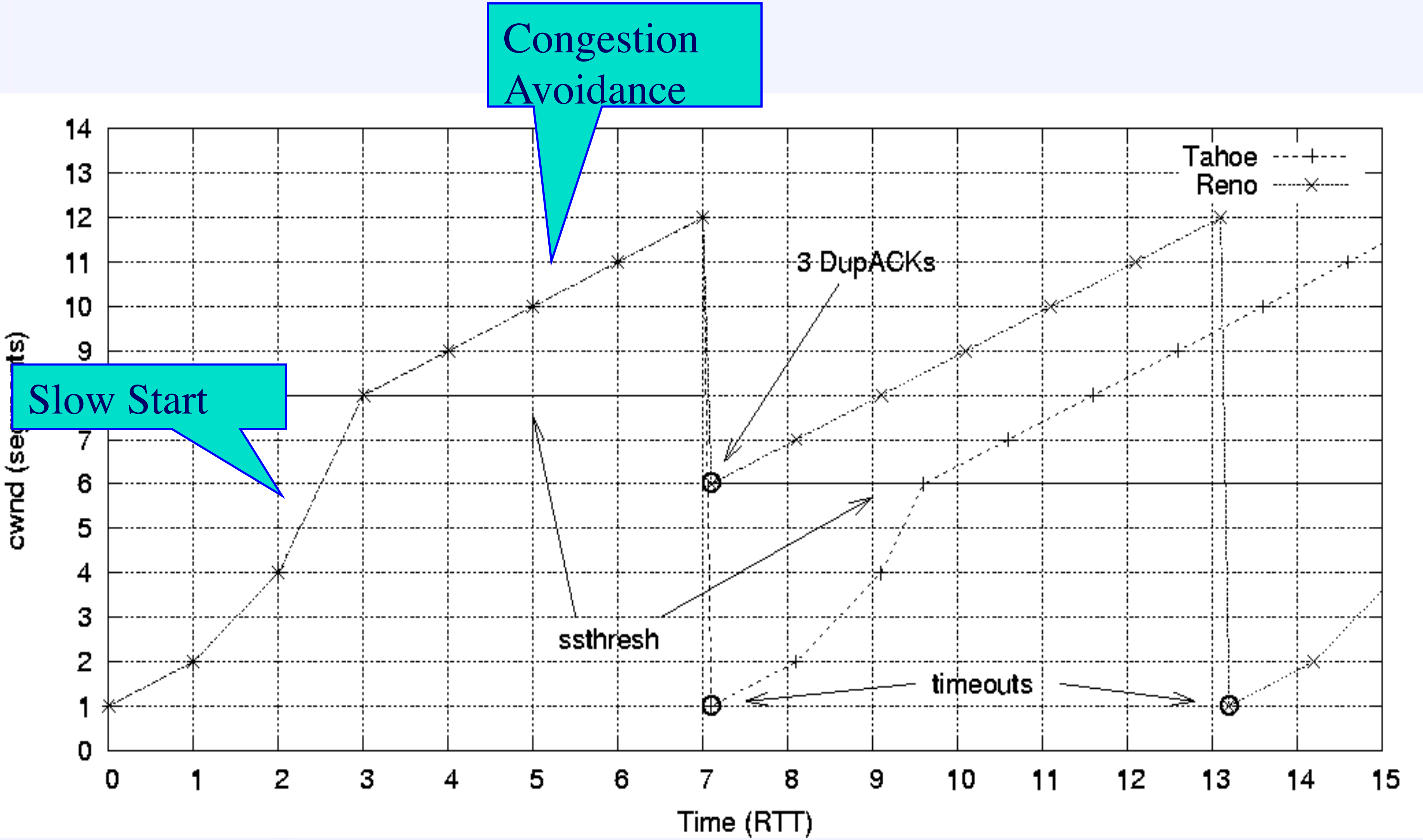
# Fast Retransmit / Fast Recovery (Reno)

Reasoning: slow start = restart; assume that network is empty
But even similar incoming ACKs indicate that packets arrive at the receiver!
Thus, slow start reaction = too conservative.

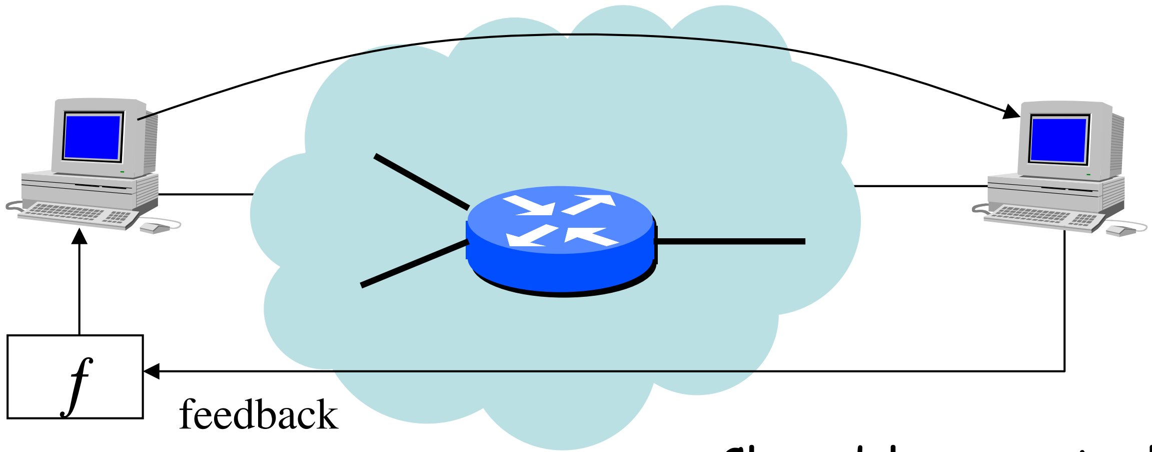
1. Upon reception of third duplicate ACK (DupACK): ssthresh = FlightSize/2
2. Retransmit lost segment (fast retransmit);
   cwnd = ssthresh + 3*SMSS
   ("inflates" cwnd by the number of segments (three) that have left the network and which the receiver has buffered)
3. For each additional DupACK received: cwnd += SMSS
   (inflates cwnd to reflect the additional segment that has left the network)
4. Transmit a segment, if allowed by the new value of cwnd and rwnd
5. Upon reception of ACK that acknowledges new data (“full ACK“):
   "deflate" window: cwnd = ssthresh (the value set in step 1)

From Mickeal Welzt

# Tahoe vs. Reno



# From the control theory point of view



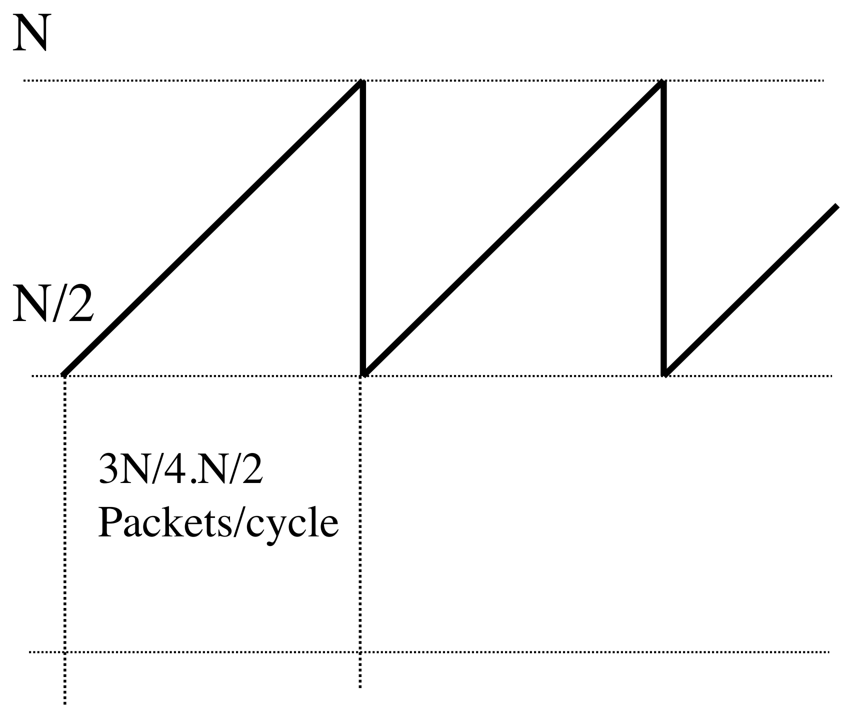

Closed-loop control

- Feedback should be frequent, but not too much otherwise there will be oscillations
- Can not control the behavior with a time granularity less than the feedback period

# The TCP saw-tooth curve

**TCP behavior in steady state**

Isolated packet losses trigger the fast recovery procedure instead of the slow-start.

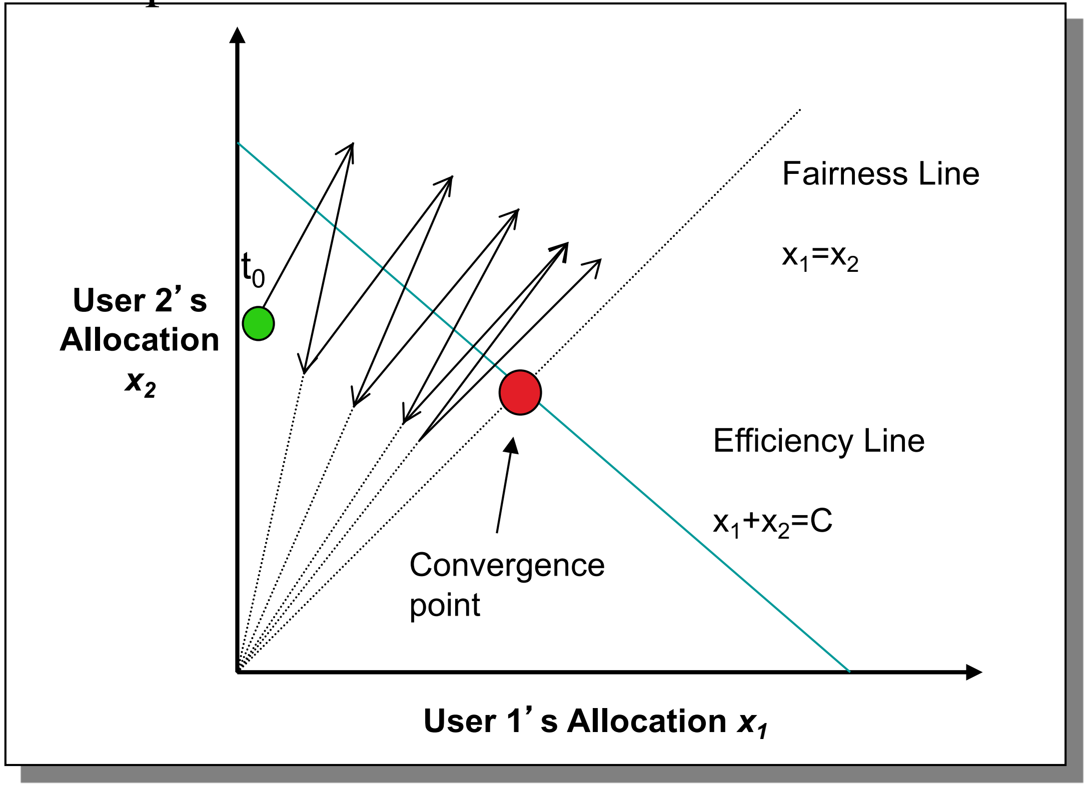



- The TCP steady-state behavior is referred to as the Additive Increase-Multiplicative Decrease process

no loss:
cwnd = cwnd + 1
loss:
cwnd = cwnd*0.5

# AIMD

Phase plot

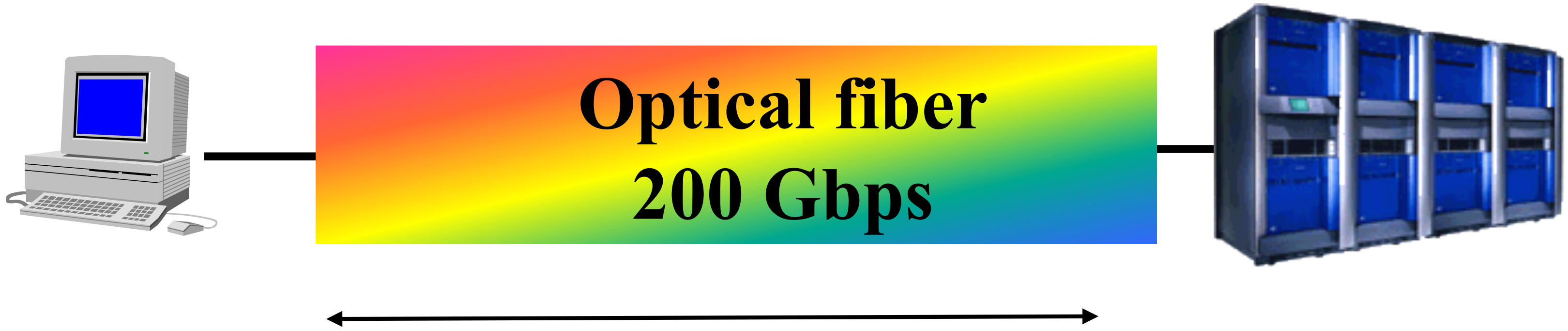



Fairness is preserved under Multiplicative Decrease since the user's allocation ratio remains the same

Ex: $$\frac{x_2}{x_1} = \frac{x_2 b}{x_1 b}$$

- Assumption: decrease policy must (at minimum) reverse the load increase over-and-above efficiency line
- Implication: decrease factor should be conservatively set to account for any congestion detection lags etc

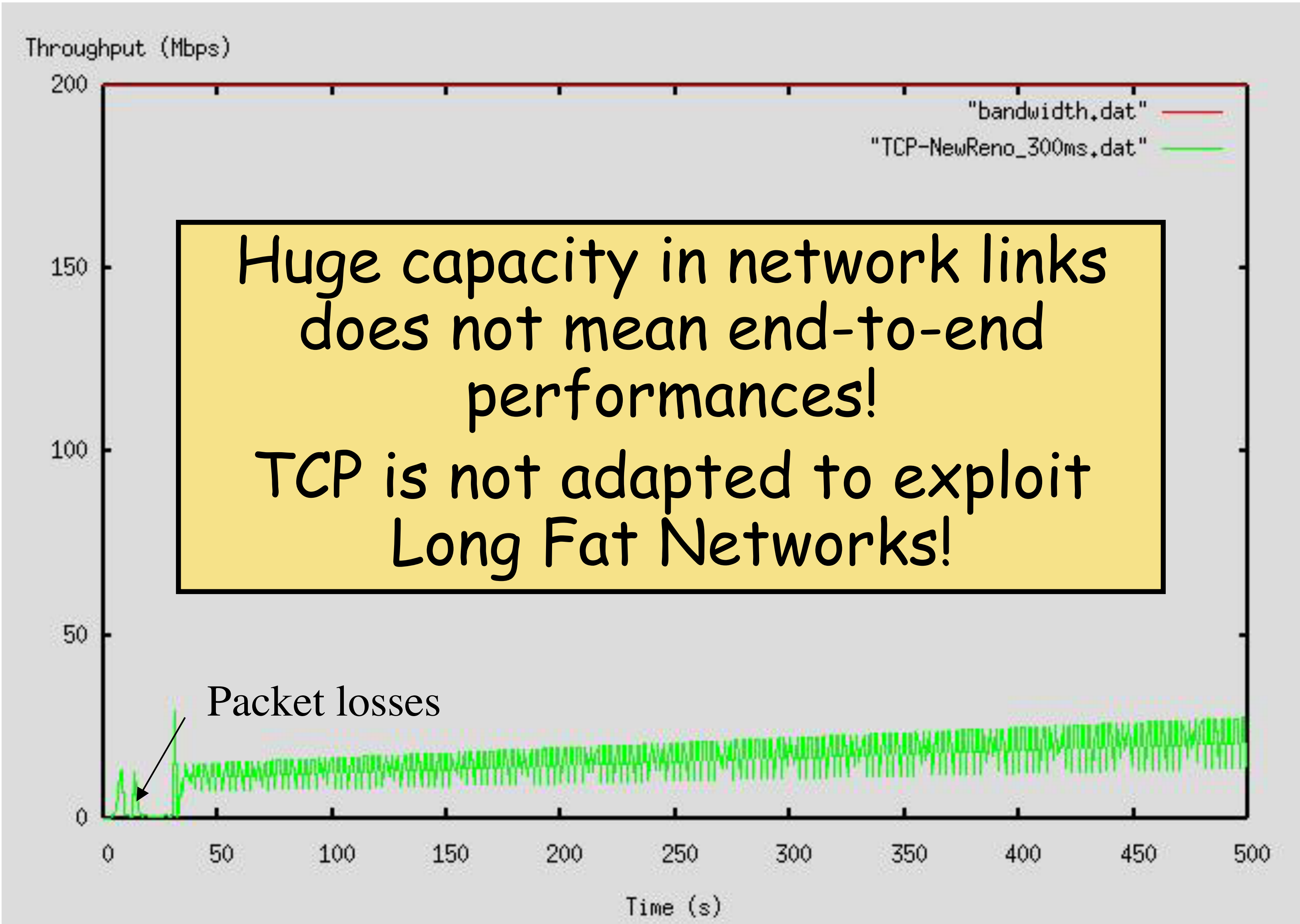
# Very High-Speed Networks



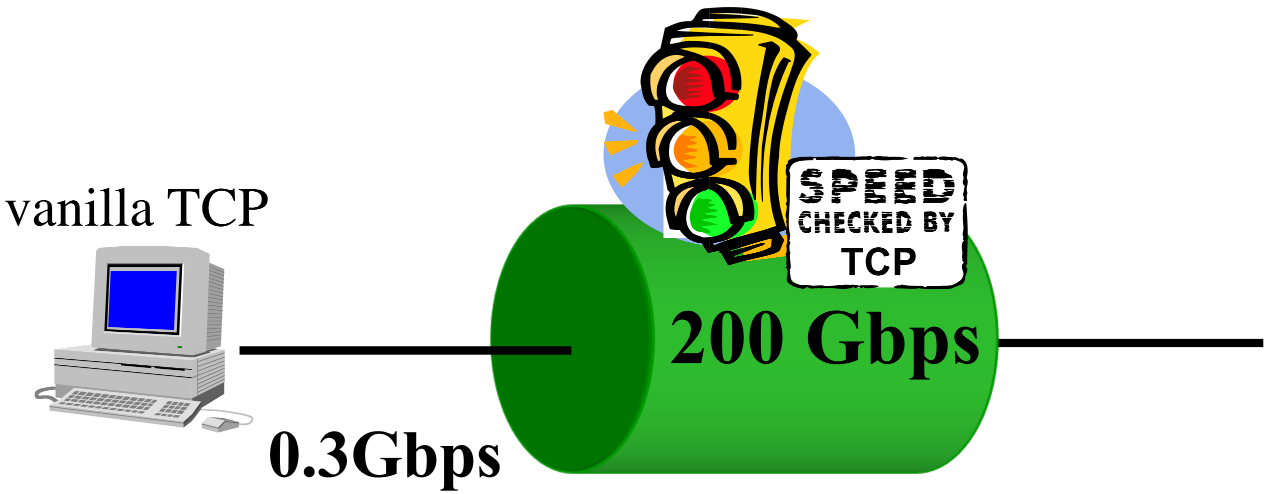

200000km/s, delay of 5ms every 1000km

- Today's backbone links are optical, DWDM-based, and offer gigabit rates
- Transmission time <<< propagation time
- Duplicating a 10GB database should not be a problem anymore

# The reality check: TCP on a 200Mbps link



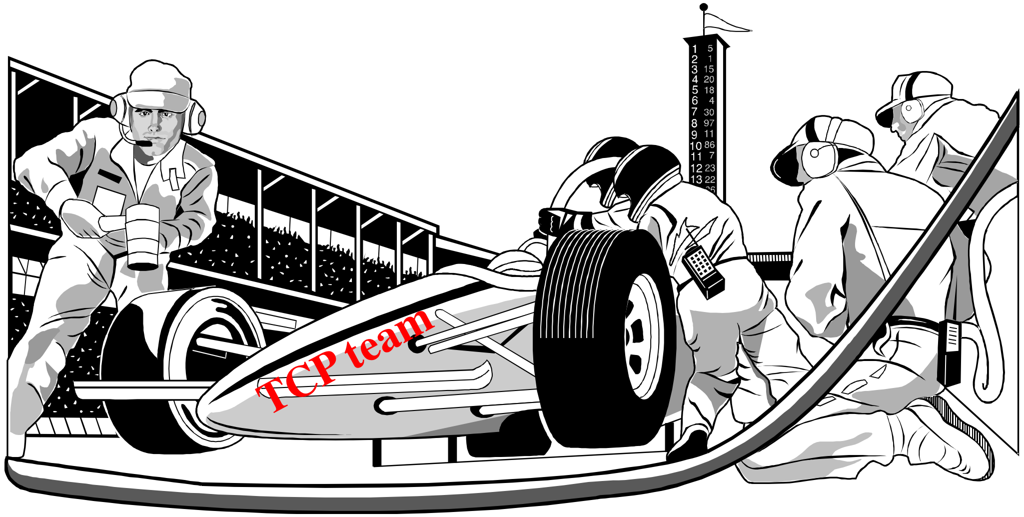
# The things about TCP your mother never told you!




- If you want to transfer a 1Go file with a standard TCP stack, you will need minutes even with a 200Gbps (how much in $?) link!

# Tuning stand for TCP

## the dark side of speed!



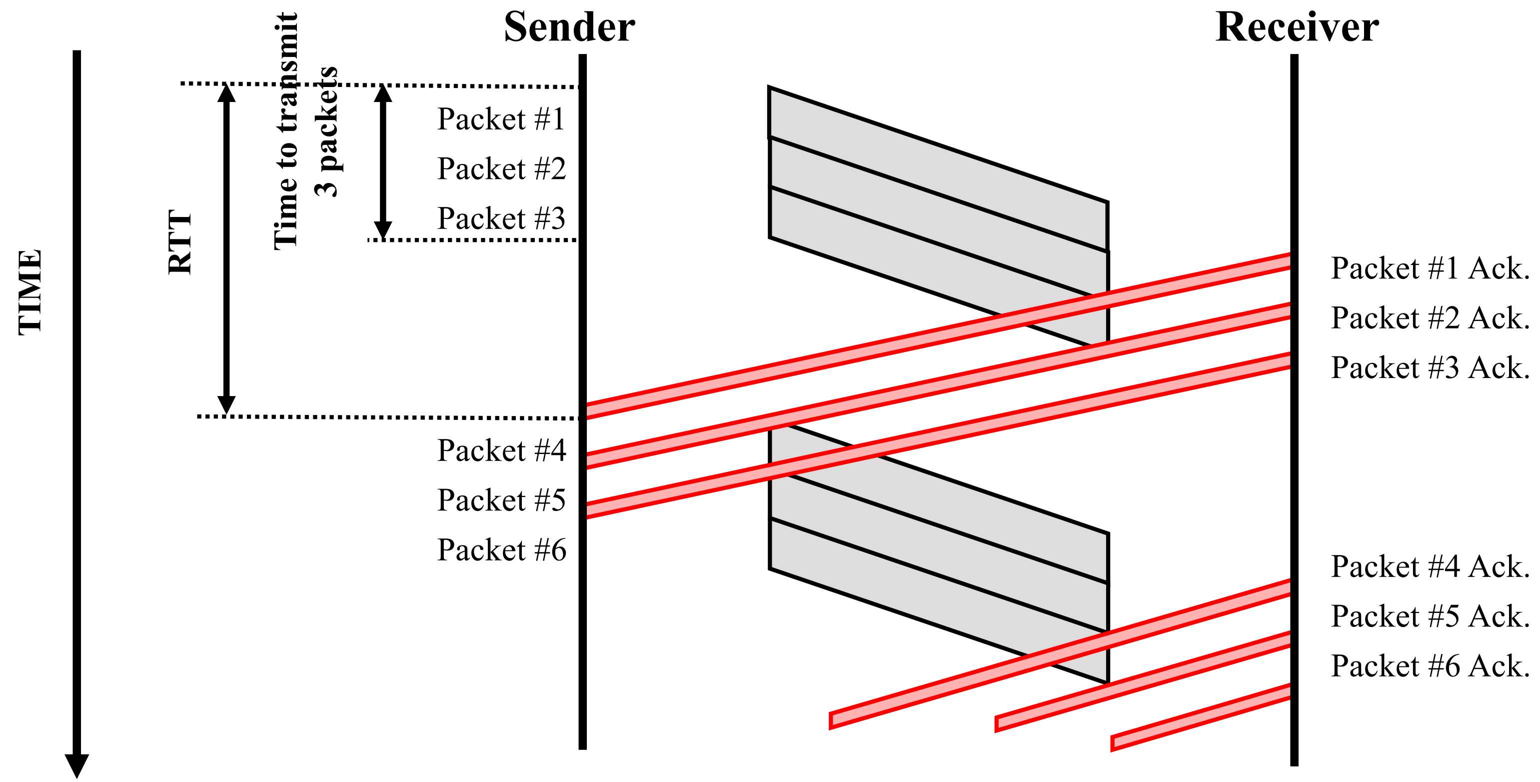

**TCP performances depend on**

- TCP & network parameters
  - Congestion window size, *ssthresh* (threshold)
  - RTO timeout settings
  - SACKs
  - Packet size
- System parameters
  - TCP and OS buffer size (in comm. subsys., drivers…)

NEED A SPECIALIST!

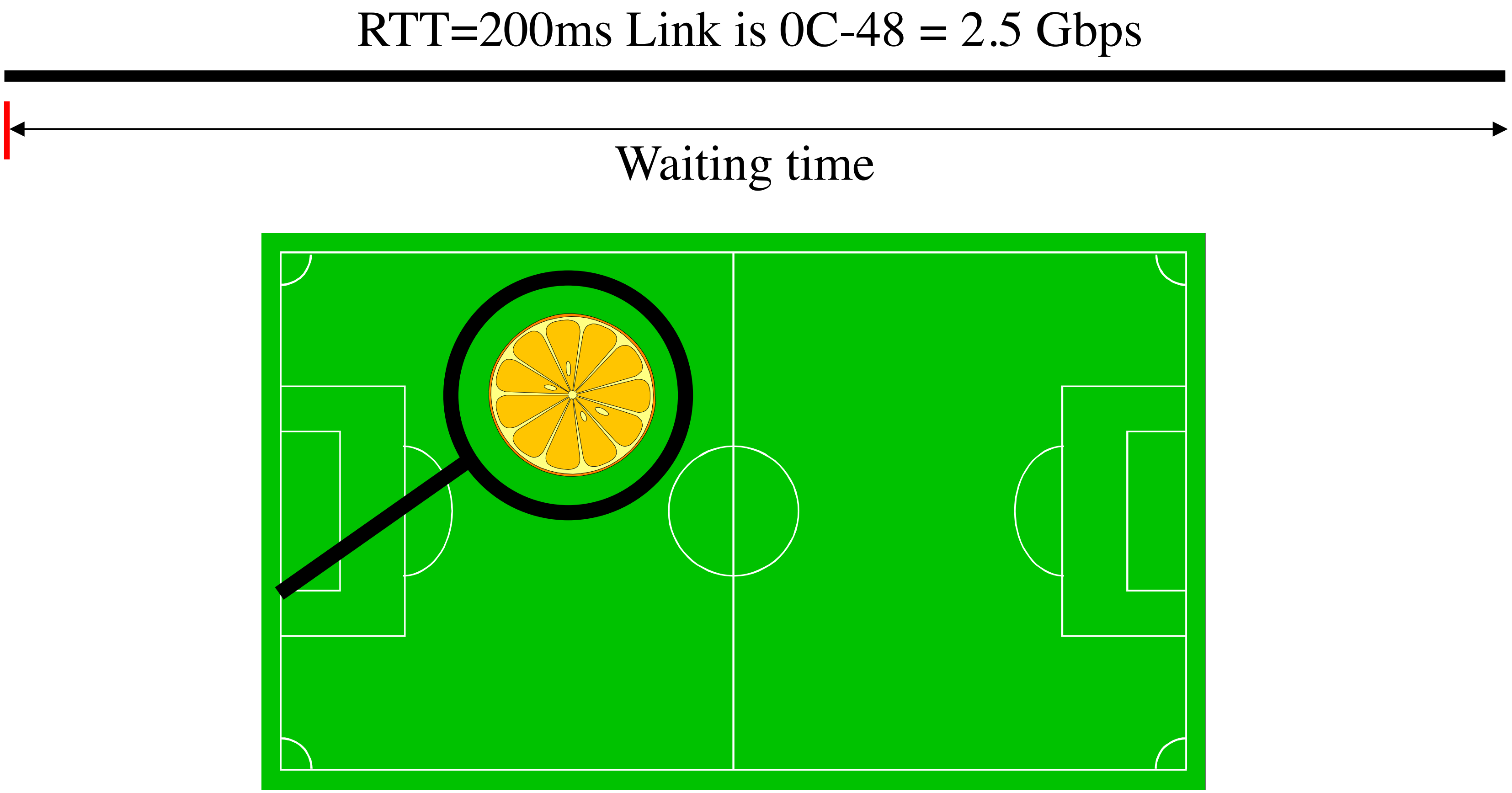
# First problem: window size

- The default maximum window size is 64Kbytes. Then the sender has to wait for acks.



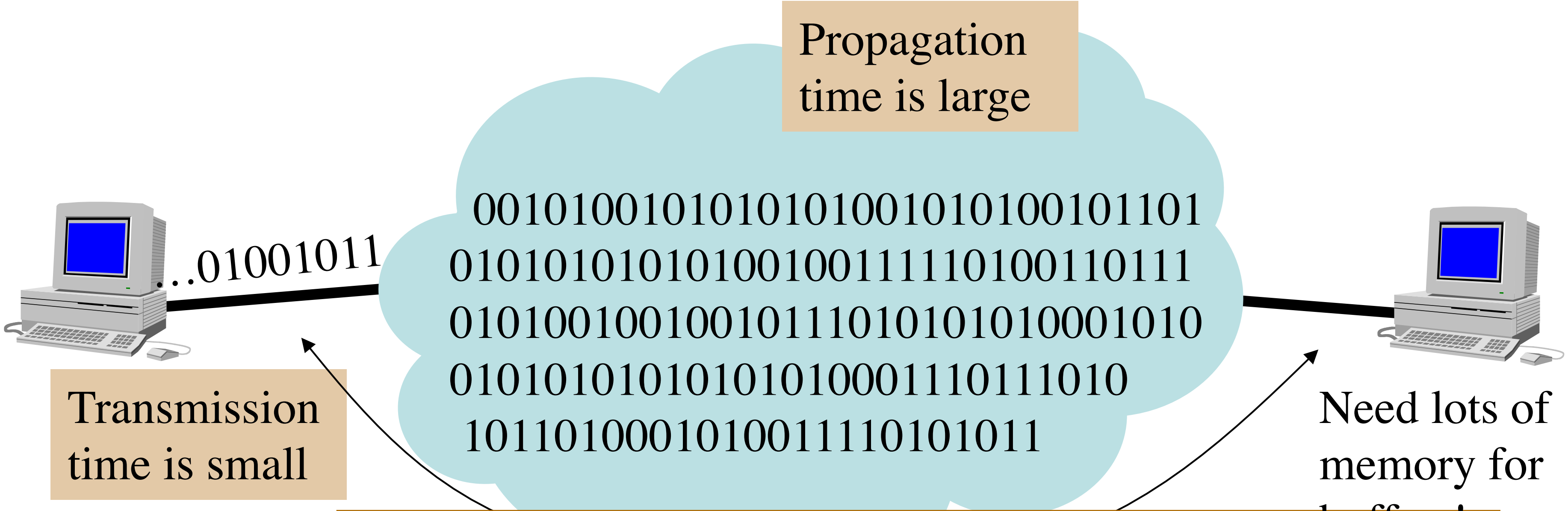
# First problem: window size

- The default maximum window size is 64Kbytes. Then the sender has to wait for acks.



# Rule of thumb on Long Fat Networks

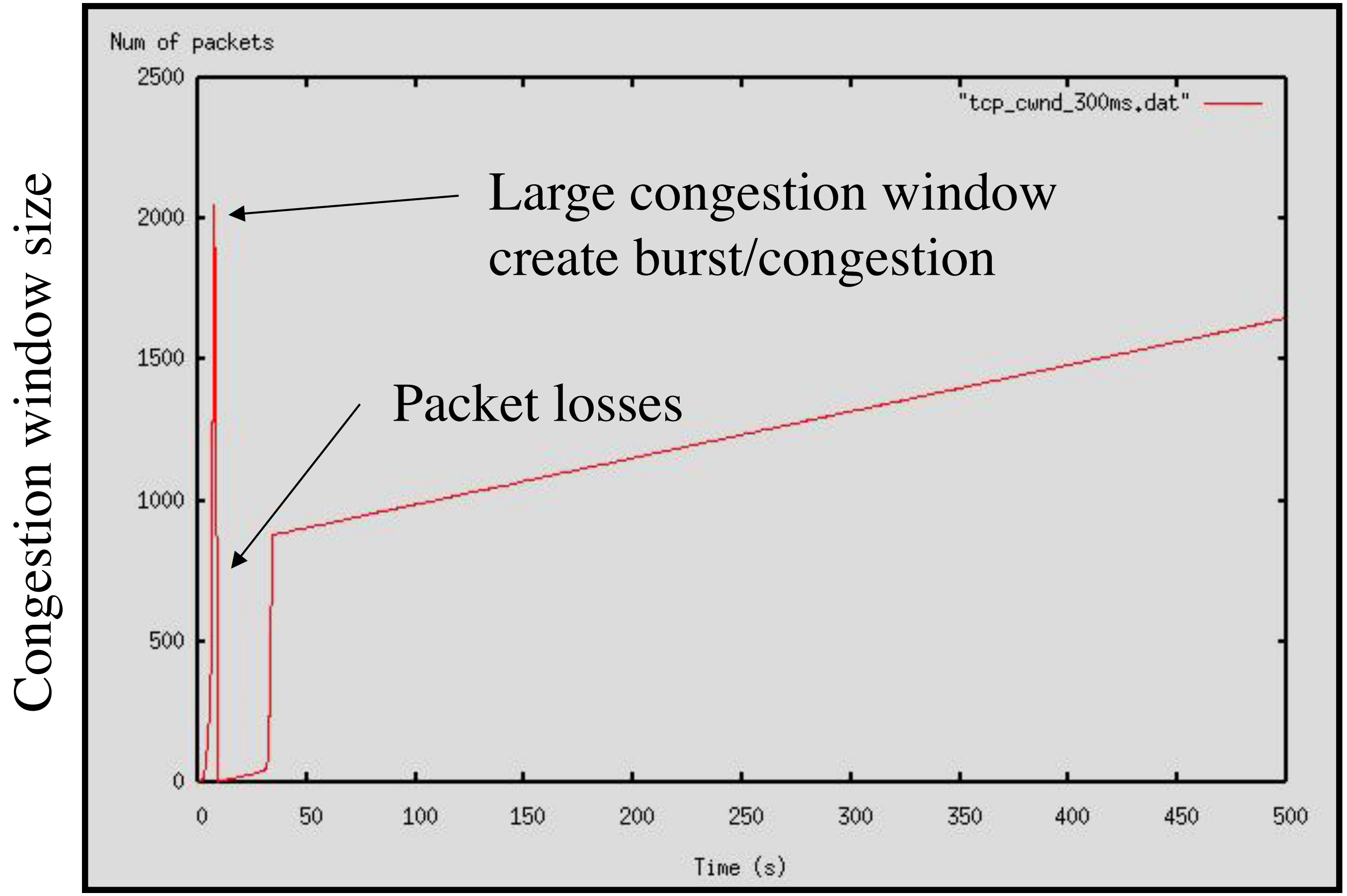
- High-~~speed~~ capacity network




The optimal window size should be set to the bandwidthxRTT product to avoid blocking at the sender side

# Side effect of large windows

TCP becomes very sensitive to packet losses on LFN

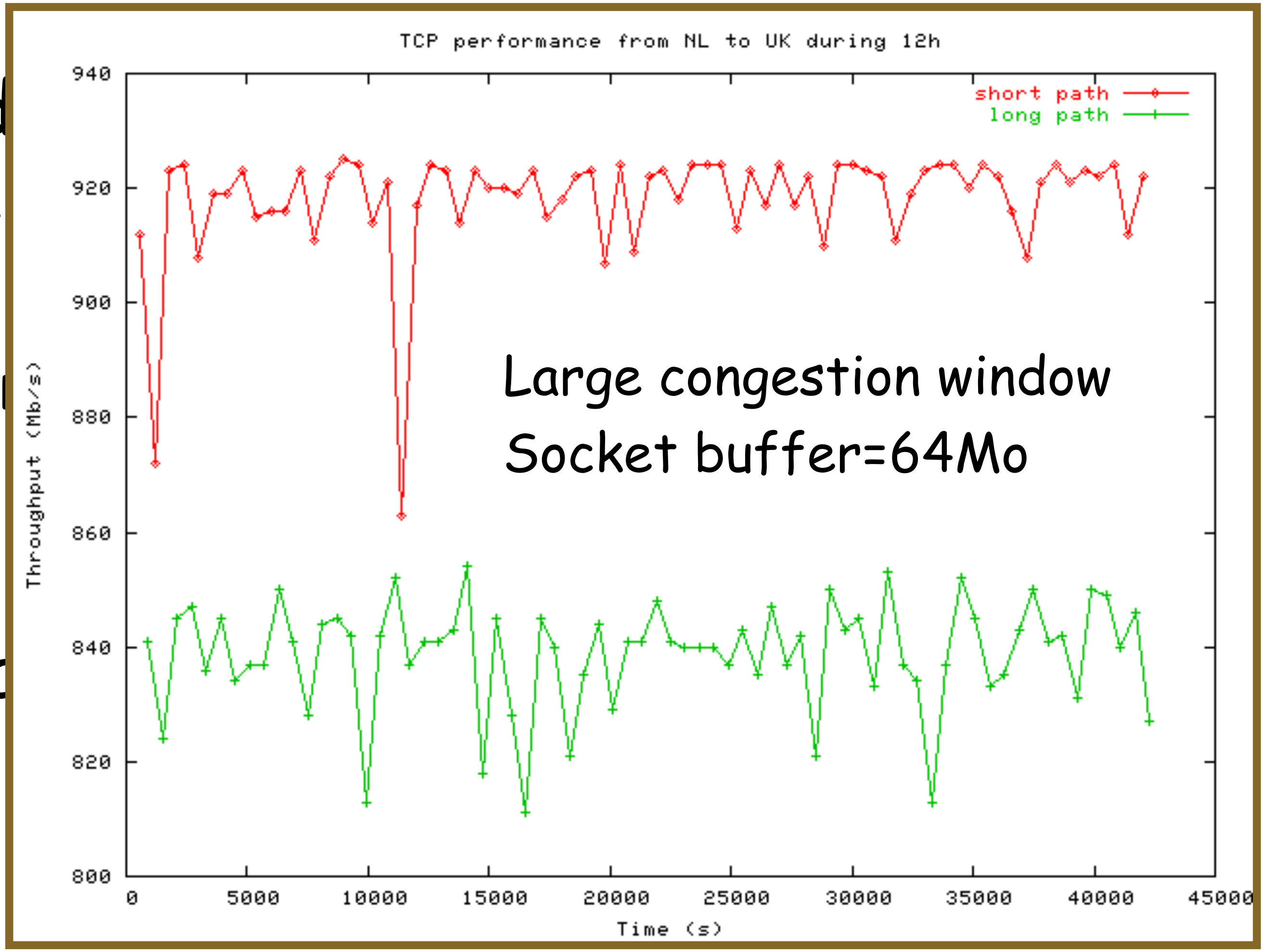


# Pushing the limits of TCP

- Standard configuration (vanilla TCP) is not adequate on many OS, everything is under-sized
  - Receiver buffer
  - System buffer
  - Default block size
- Will manage to get near 1Gbps if well-tuned

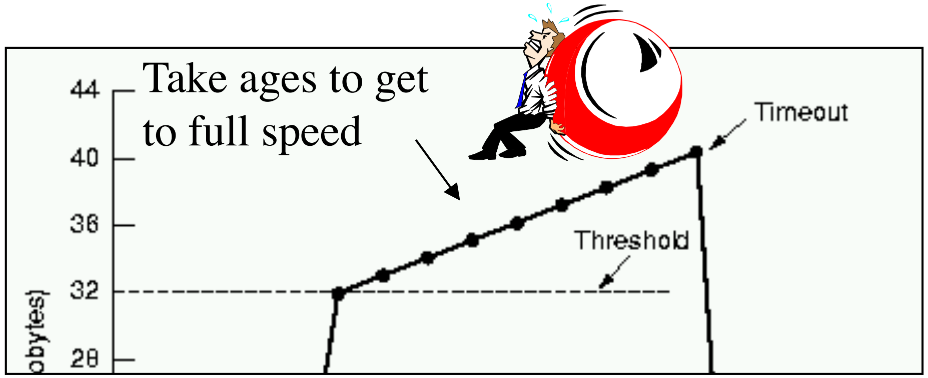
# Pushing the limits of TCP

- Standard
  adequate
  sized
  - Receive
  - System
  - Default
- Will mano




Source: M. Goutelle, GEANT test campaign

# Problem on high capacity link? Additive increase is still too slow!




With 100ms of round trip time, a connection needs 203 minutes (3h23) to send at 10Gbps starting from 1Mbps!

**Once you get high throughput, maintaining it is difficult too!**

- Sustaining high congestion windows:

A Standard TCP connection with:

- 1500-byte packets;
- a 100 ms round-trip time;
- a steady-state throughput of 10 Gbps;

would require:

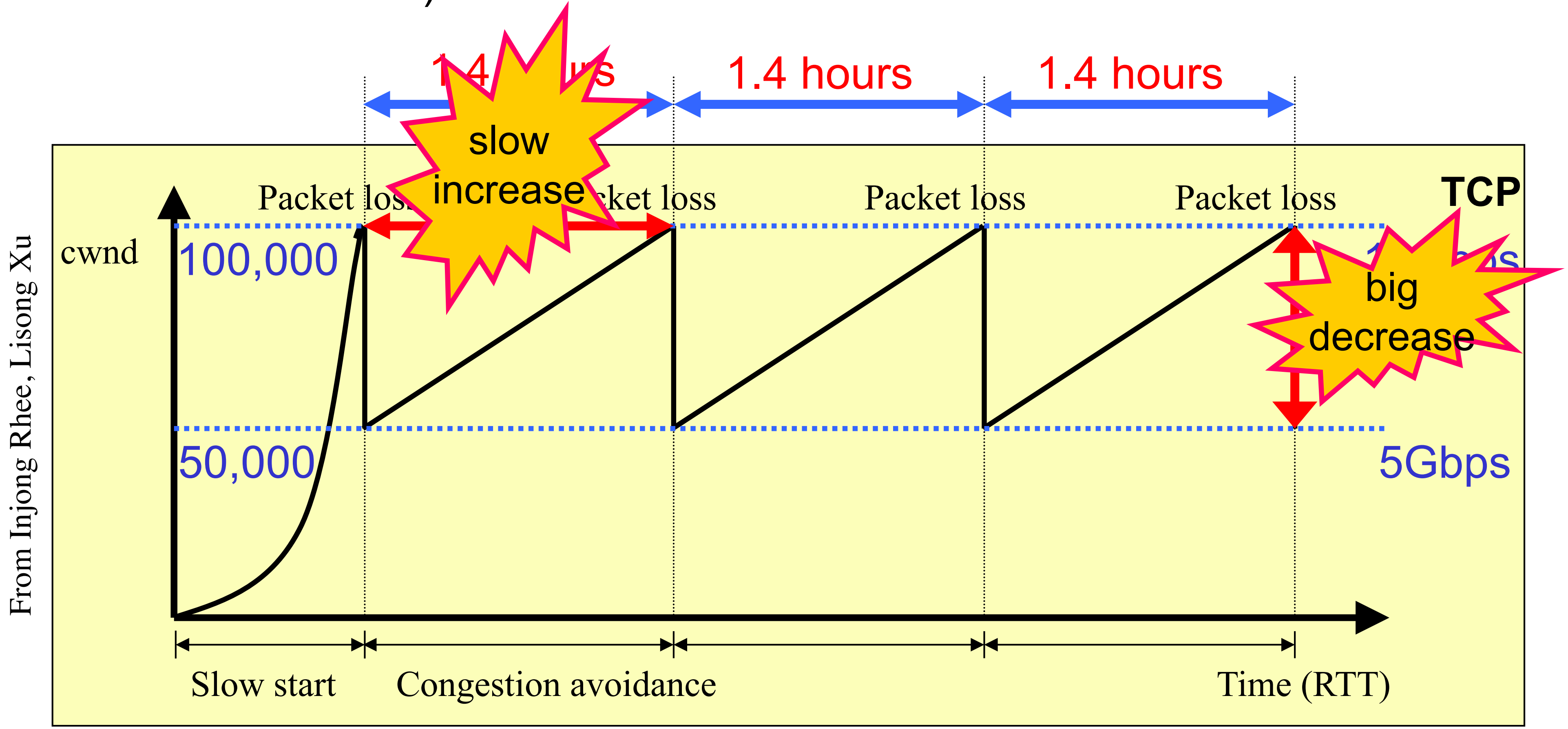
- an average congestion window of 83,333 segments;
- and at most one drop (or mark) every 5,000,000,000 packets (or equivalently, at most one drop every 1 2/3 hours).

This is not realistic.

From S. Floyd

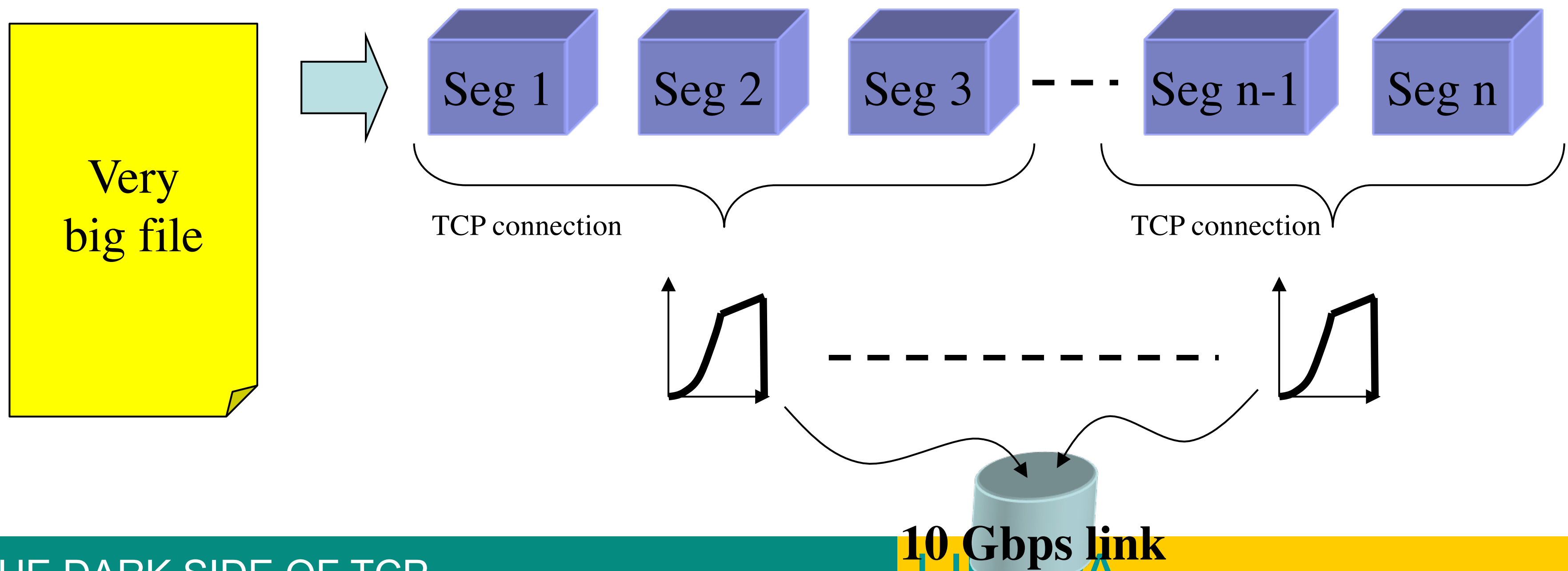
# TCP rules: slow increase, big decrease

A TCP connection with 1250-Byte packet size and 100ms RTT is running over a 10Gbps link (assuming no other connections, and no buffers at routers)

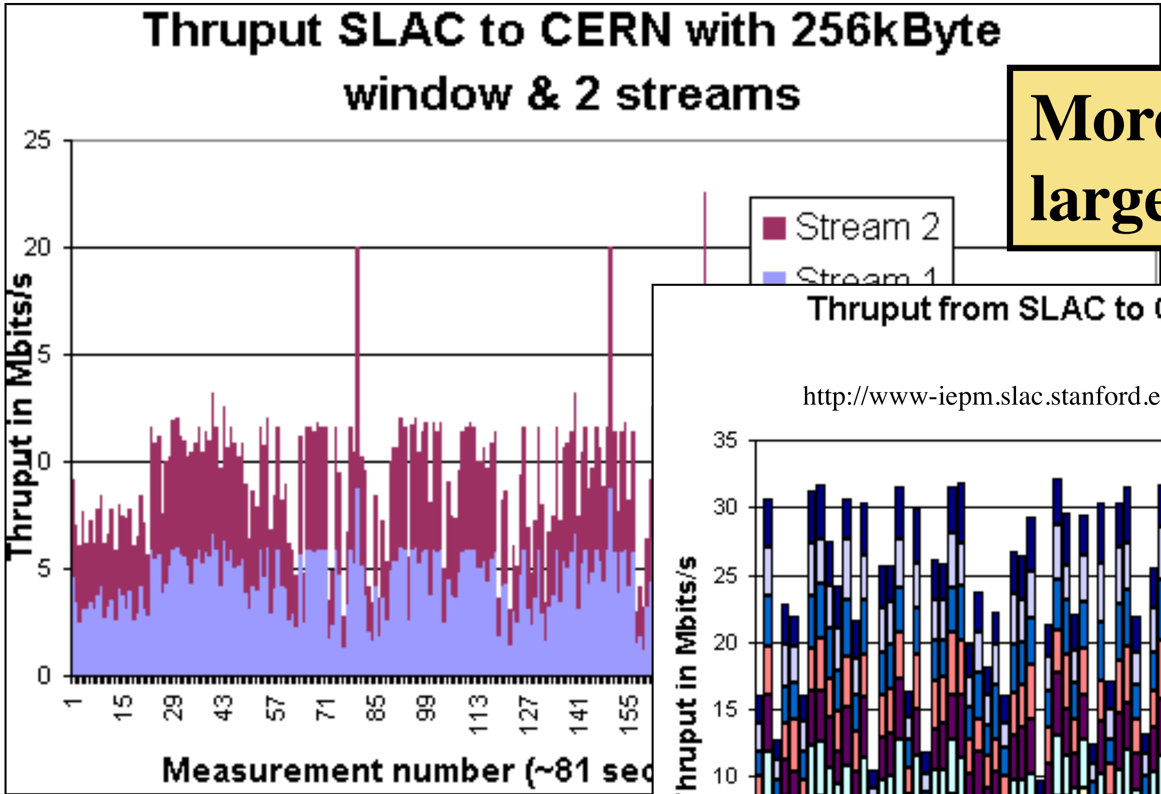


# Going faster (cheating?)
## *n* flows is better than 1

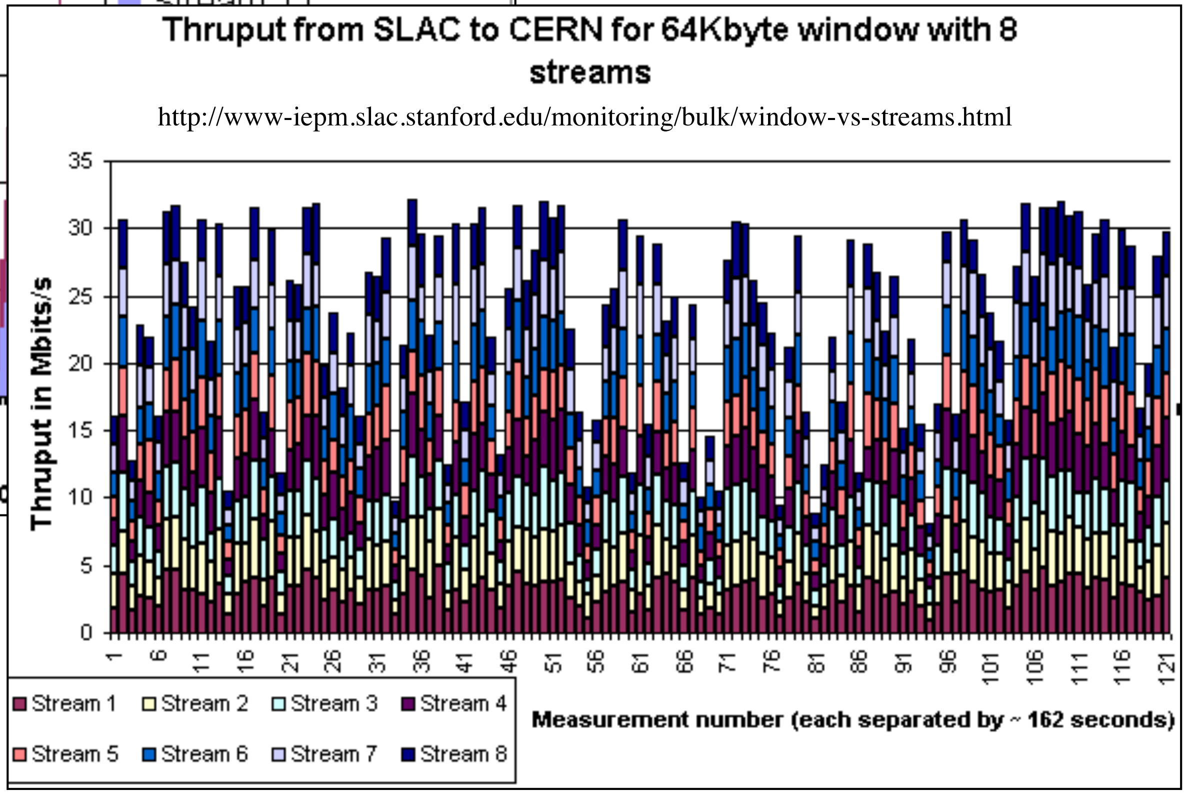
- The CC limits the throughput of a TCP connection: so why not use more than 1 connection for the same file?



# Some results from IEPM/SLAC




**More streams is better than larger congestion windows**



# Multiple streams

- No/few modifications to transport protocols (i.e. TCP)
  - Parallel socket libraries
  - GridFTP (http://www.globus.org/datagrid/gridftp.html)
  - bbFTP (http://doc.in2p3.fr/bbftp/)

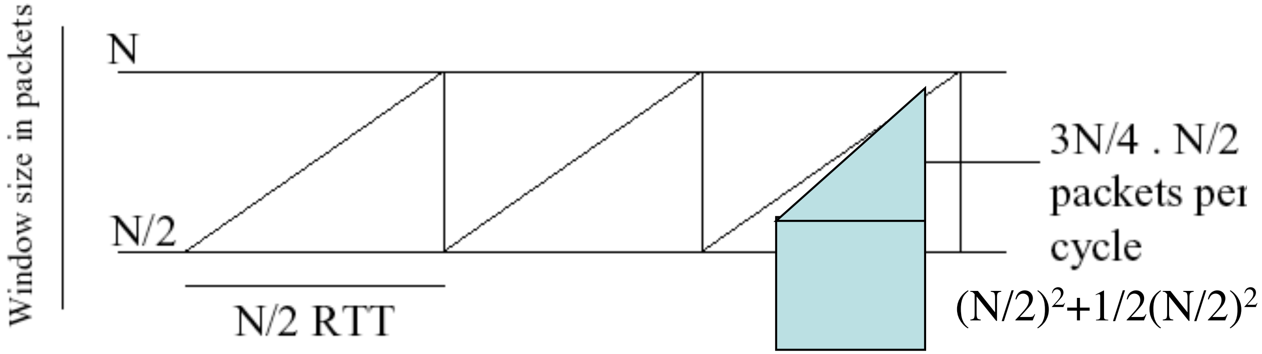
# New transport protocols

- New transport protocols are those that are not only optimizations of TCP
- New behaviors, new rules, new requirements! Everything is possible!
- New protocols are then not necessarily TCP compatible!

# The new transport protocol strip

# Response function

- Throughput = f(p, RTT)
- TCP's response function




Average window size (in packets) = W = 3N/4 , from (N+N/2)/2

Number of packets per cycle = 3N/4 . N/2 = $3N^2/8$ = 1/ p

- Where p is the packet loss ratio (which should remain small enough)
- So $N = \sqrt{8/3p}$

Average throughput (in packets/sec) = B = W / RTT = 3N / 4 RTT

$$= \sqrt{3/2}\,\frac{1}{RTT\sqrt{p}}$$

$$Throughput = \frac{W}{RTT} = \sqrt{\frac{3}{2}}\,\frac{MTU}{RTT\sqrt{p}}$$

# TCP's response function in image

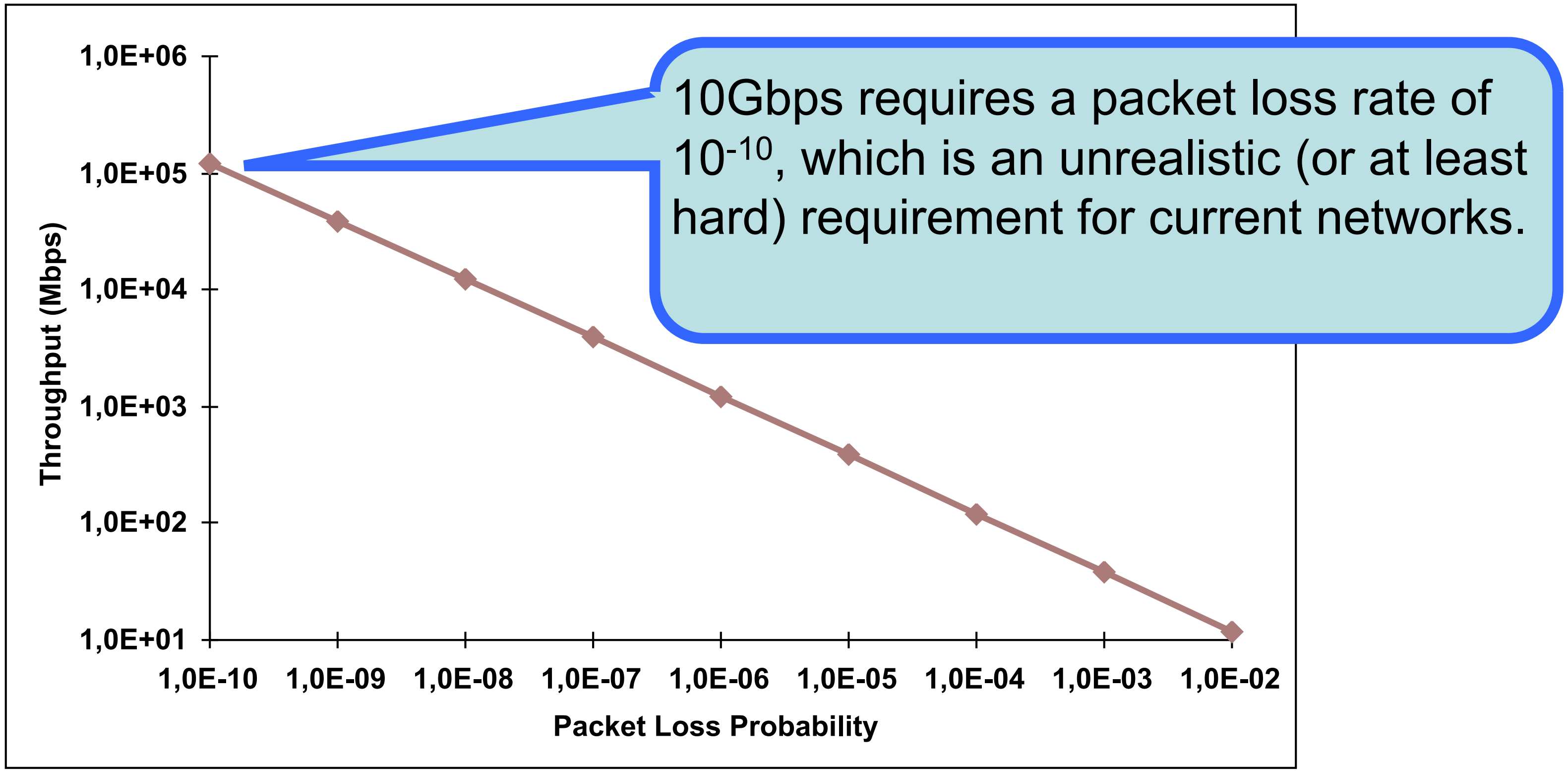
$$Throughput = \frac{W}{RTT} = \sqrt{\frac{3}{2}}\frac{MTU}{RTT\sqrt{p}}$$

$MTU$: Packet Size

$RTT$: Round-Trip Time

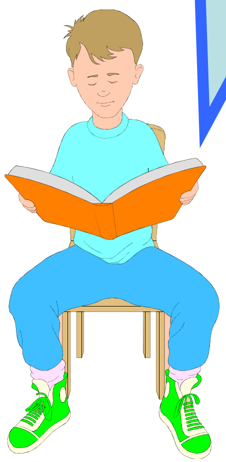
$P$ : Packet Loss Probability

From Injong Rhee, Lisong Xu



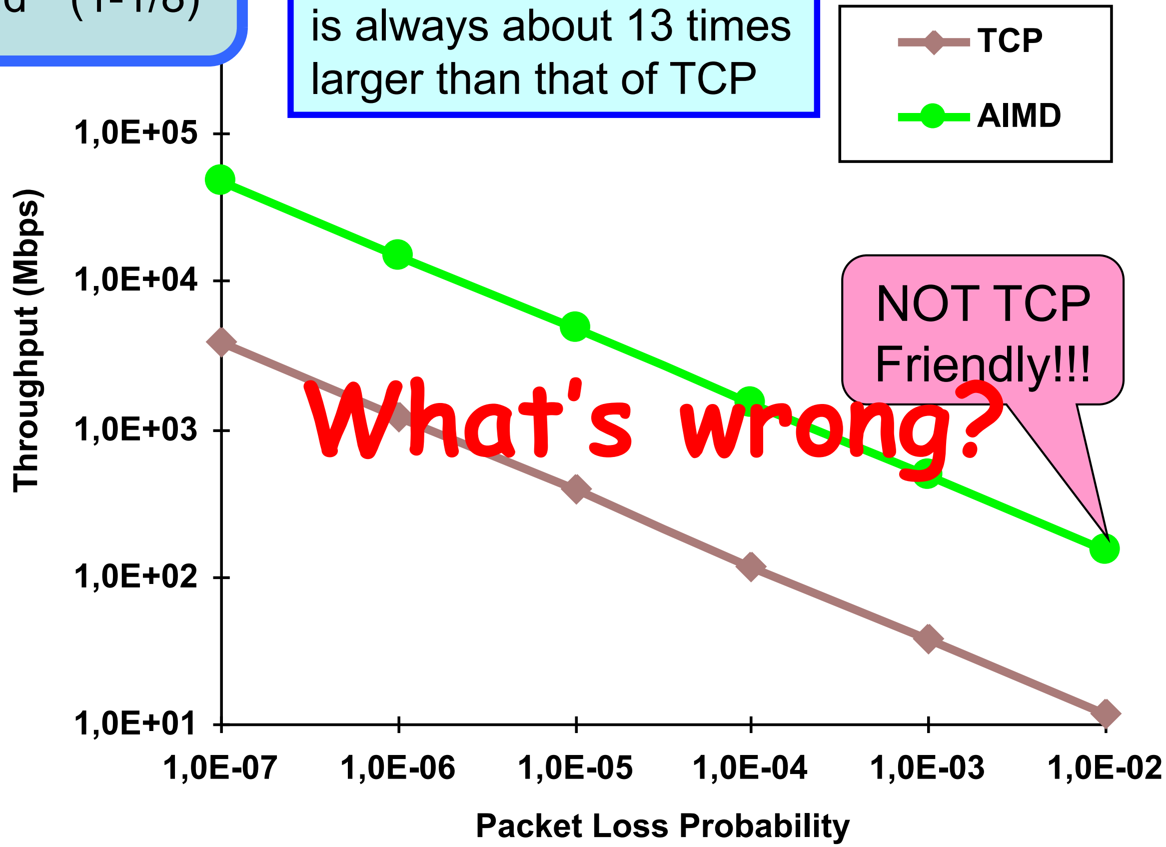
# AIMD, general case

cwnd = cwnd + 1
⇩
cwnd = cwnd + 32



cwnd = cwnd * (1-1/2)
⇩
cwnd = cwnd * (1-1/8)

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




- TCP: $R = \frac{MSS}{RTT}\frac{1.2}{p^{0.5}}$
- AIMD: $R = \frac{MSS}{RTT}\frac{15.5}{p^{0.5}}$

Inspired from Injong Rhee, Lisong Xu

# High Speed TCP [Floyd]

- Modifies the response function to allow for more link utilization in current high-speed networks where the loss rate is smaller than that of the networks TCP was designed for (at most $10^{-2}$)

| TCP Throughput (Mbps) | RTTs Between Losses | W | P |
|---|---|---|---|
| 1 | 5.5 | 8.3 | 0.02 |
| 10 | 55.5 | 83.3 | 0.0002 |
| 100 | 555.5 | 833.3 | 0.000002 |
| 1000 | 5555.5 | 8333.3 | 0.00000002 |
| 10000 | 55555.5 | 83333.3 | 0.0000000002 |

Table 1: RTTs Between Congestion Events for Standard TCP, for 1500-Byte Packets and a Round-Trip Time of 0.1 Seconds.

From draft-ietf-tsvwg-highspeed-01.txt

# Modifying the response

| Packet Drop Rate P | Congestion Window W | RTTs Between Losses |
|---|---|---|
| 10^-2 | 12 | 8 |
| 10^-3 | 38 | 25 |
| 10^-4 | 120 | 80 |
| 10^-5 | 379 | 252 |
| 10^-6 | 1200 | 800 |
| 10^-7 | 3795 | 2530 |
| 10^-8 | 12000 | 8000 |
| 10^-9 | 37948 | 25298 |
| 10^-10 | 120000 | 80000 |

Table 2: TCP Response Function for Standard TCP. The average congestion window W in MSS-sized segments is given as a function of the packet drop rate P.

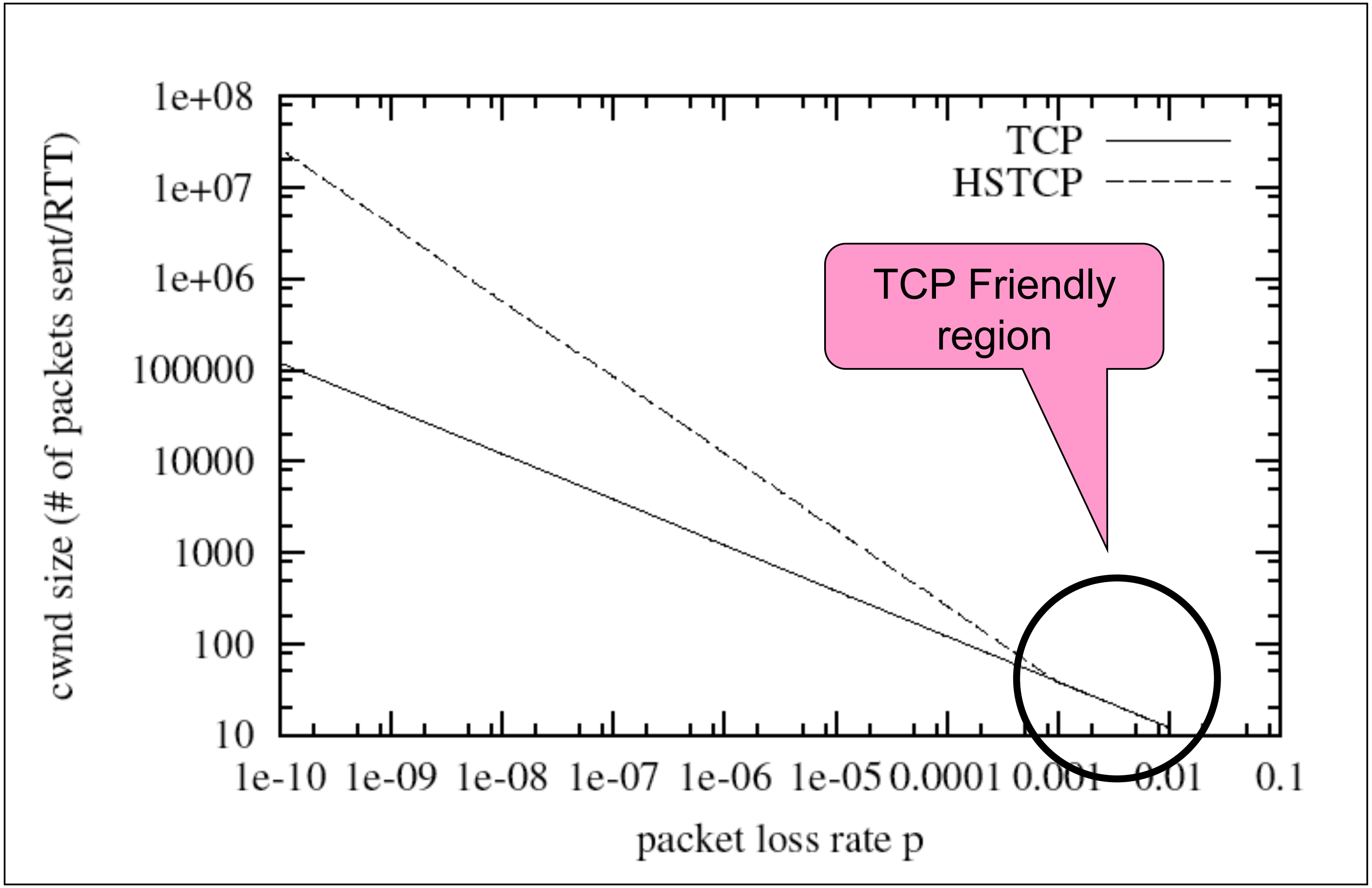
To specify a modified response function for HighSpeed TCP, we use three parameters, Low_Window, High_Window, and High_P. To Ensure TCP compatibility, the HighSpeed response function uses the same response function as Standard TCP when the current congestion window is at most Low_Window, and uses the HighSpeed response function when the current congestion window is greater than Low_Window. In this document we set Low_Window to 38 MSS-sized segments, corresponding to a packet drop rate of 10^-3 for TCP.

From draft-ietf-tsvwg-highspeed-01.txt

| Packet Drop Rate P | Congestion Window W | RTTs Between Losses |
|---|---|---|
| 10^-2 | 12 | 8 |
| 10^-3 | 38 | 25 |
| 10^-4 | 263 | 38 |
| 10^-5 | 1795 | 57 |
| 10^-6 | 12279 | 83 |
| 10^-7 | 83981 | 123 |
| 10^-8 | 574356 | 180 |
| 10^-9 | 3928088 | 264 |
| 10^-10 | 26864653 | 388 |

**Table 3: TCP Response Function for HighSpeed TCP. The average congestion window W in MSS-sized segments is given as a function of the packet drop rate P.**

# See it in image



# Relation with AIMD

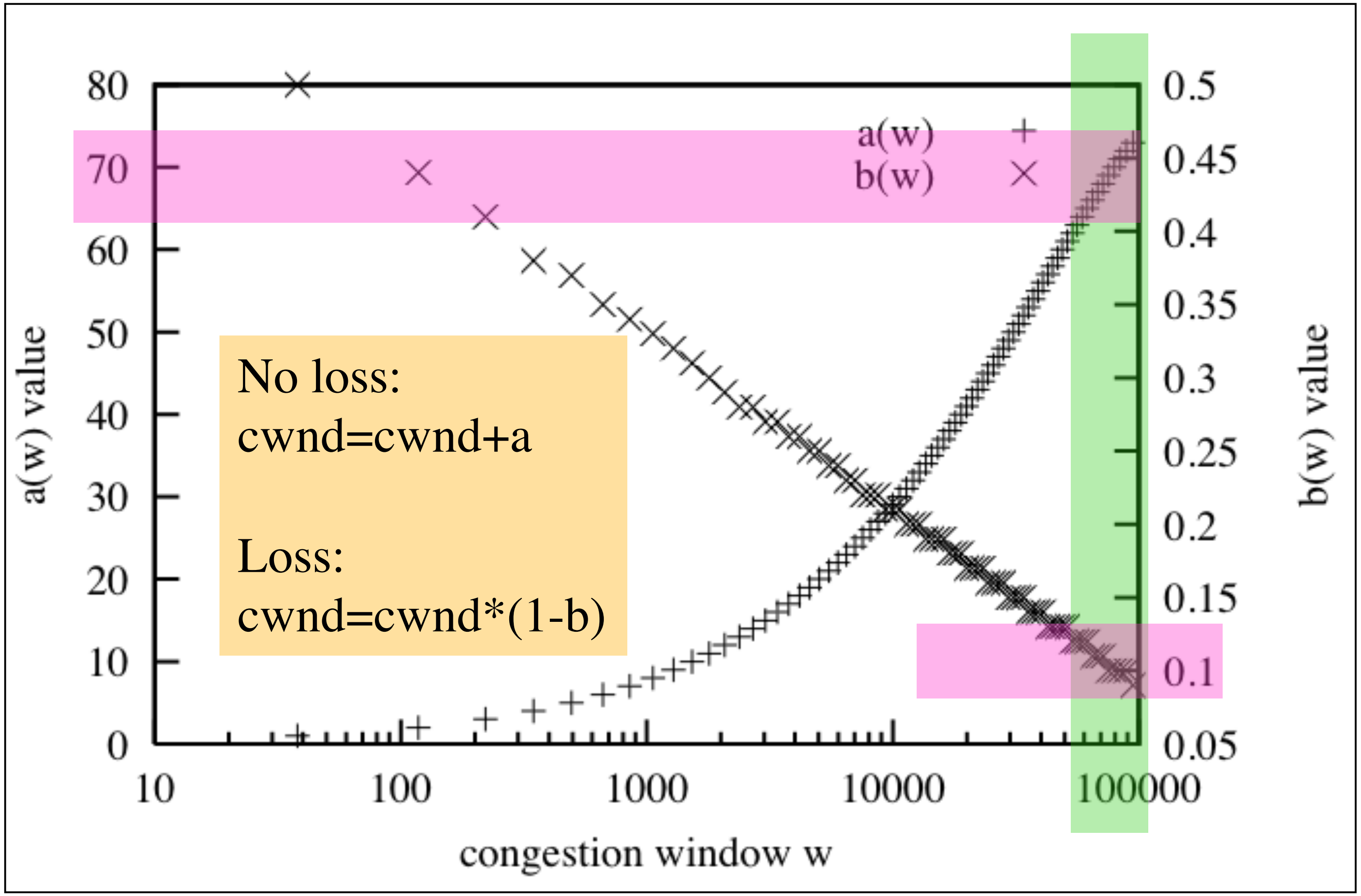
- TCP-AIMD
  - Additive increase: a=1
  - Multiplicative decrease: b=1/2

no loss:
cwnd = cwnd + 1

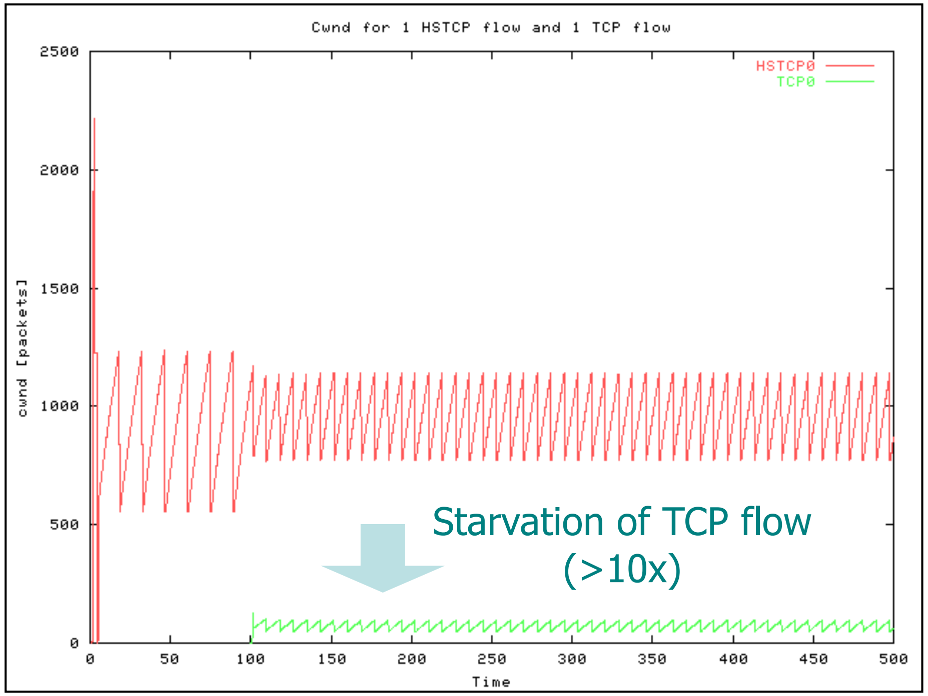
loss:
cwnd = cwnd*0.5

- HSTCP-AIMD
  - Link a & b to congestion window size
  - a = a(cwnd), b=b(cwnd)
  - General rules
    - the larger cwnd, the larger the increment
    - The larger cwnd, the smaller the decrement

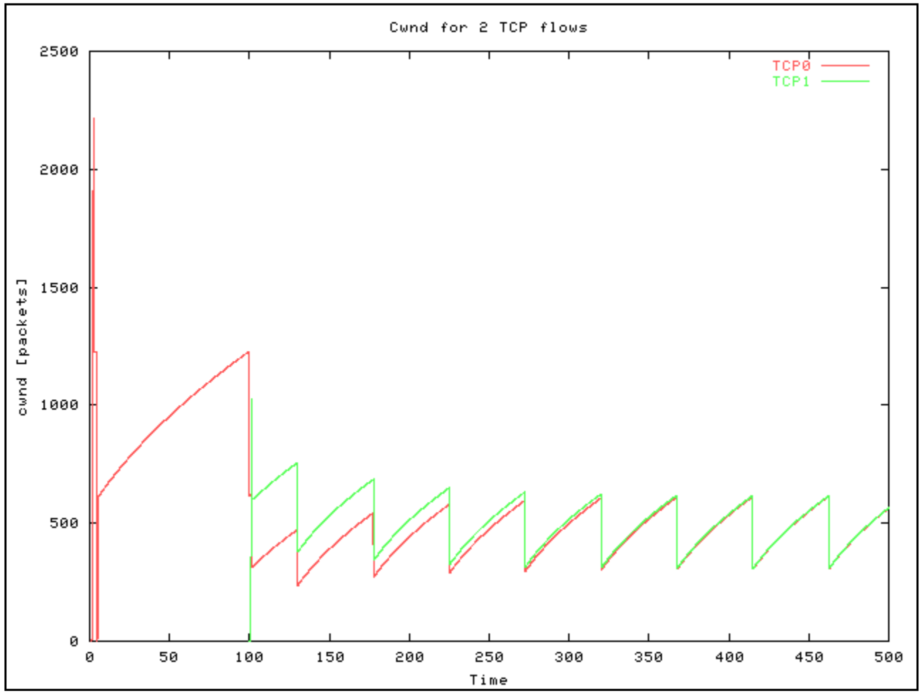
# Quick to grab bandwidth, slow to give some back!



# Talking about dark side…




**1 HSTCP and 1 TCP flow**

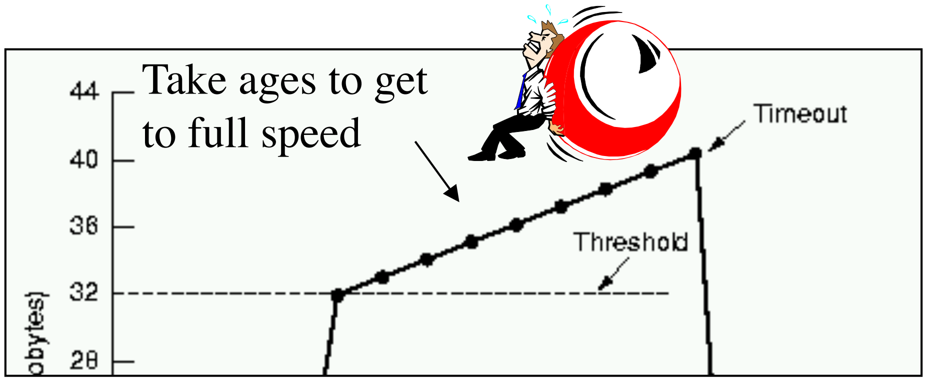



**2 TCP flows**

**SETUP** RTT=100ms
Bottleneck BW=50Mbps
Qsize=BW*RTT
Qtype=DropTail

# It's a search problem!

- Get to the available bandwidth: how to get there efficiently?




Linear increase not optimal

« Small jumps » strategy

# Binary Search with Smax and Smin

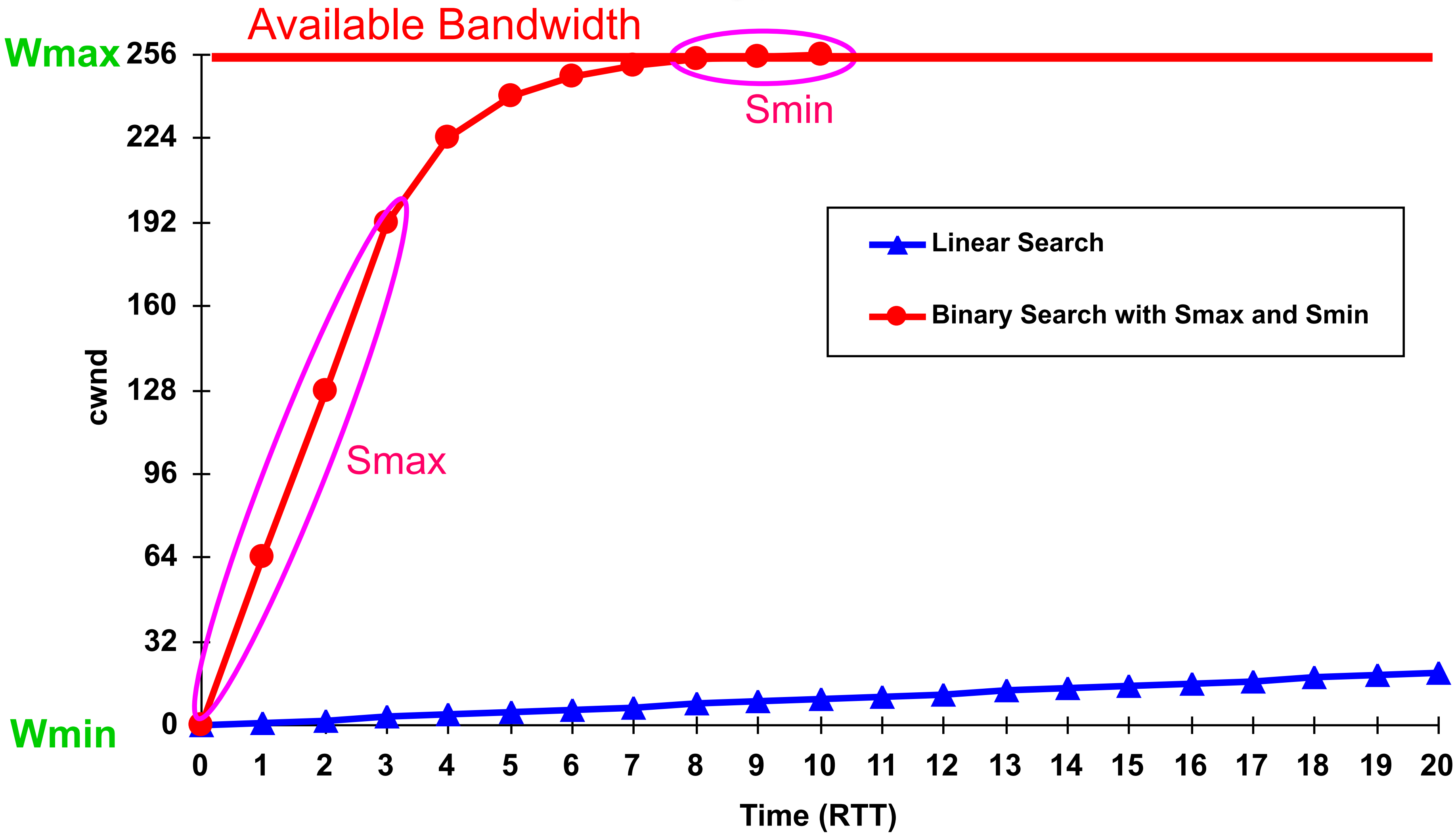
- Binary search

```
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
  - Usually the last cwnd value before packet drops (last fast recovery)
- Wmin: Min Window
- Smax: Max Increment
- Smin: Min Increment

Source Injong Rhee, Lisong Xu

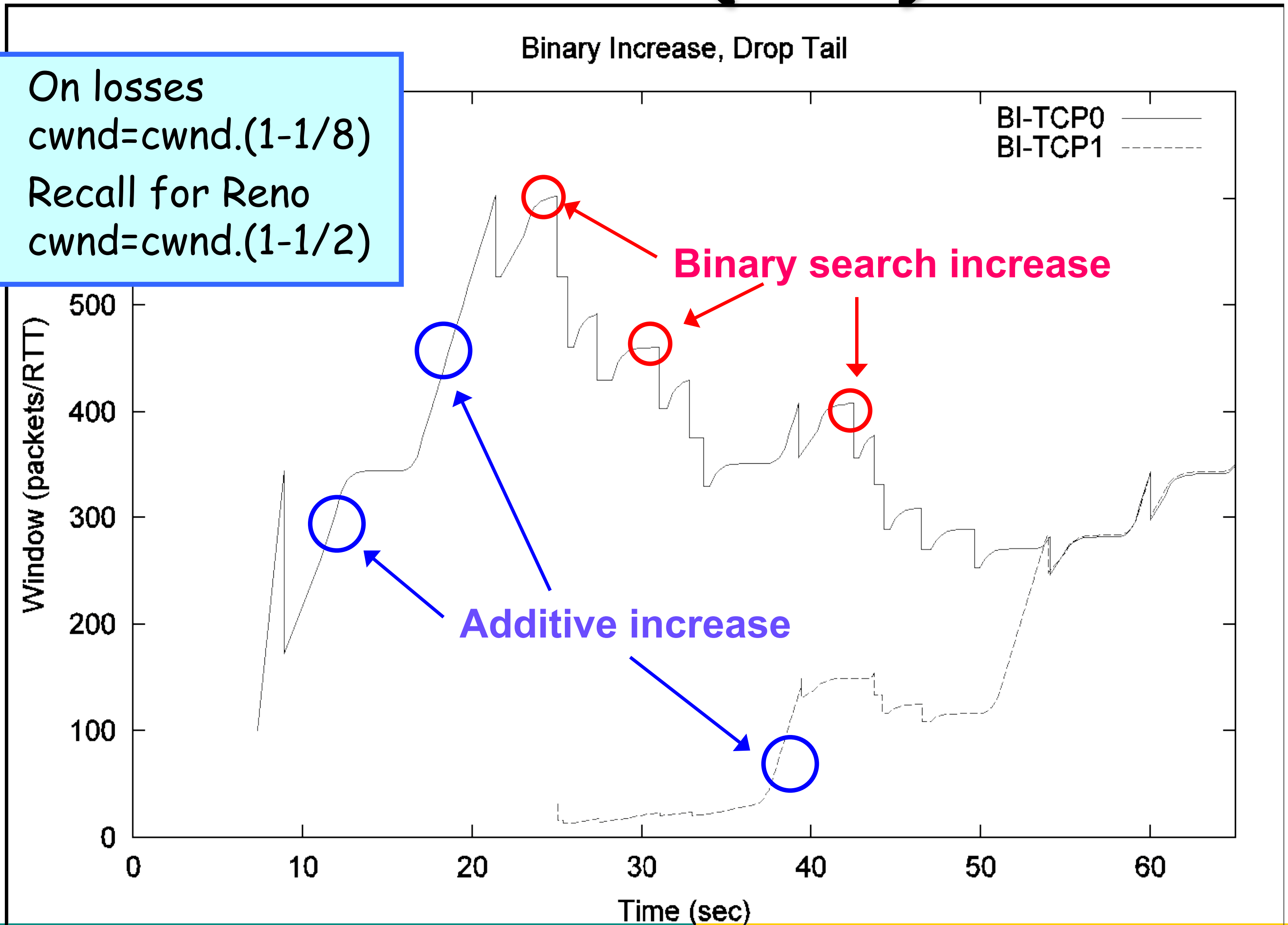
# Binary Search with Smax and Smin




Source Injong Rhee, Lisong Xu

# Binary Increase Congestion Control (BIC)

- On losses cwnd=cwnd.(1-1/8)
- Recall for Reno cwnd=cwnd.(1-1/2)



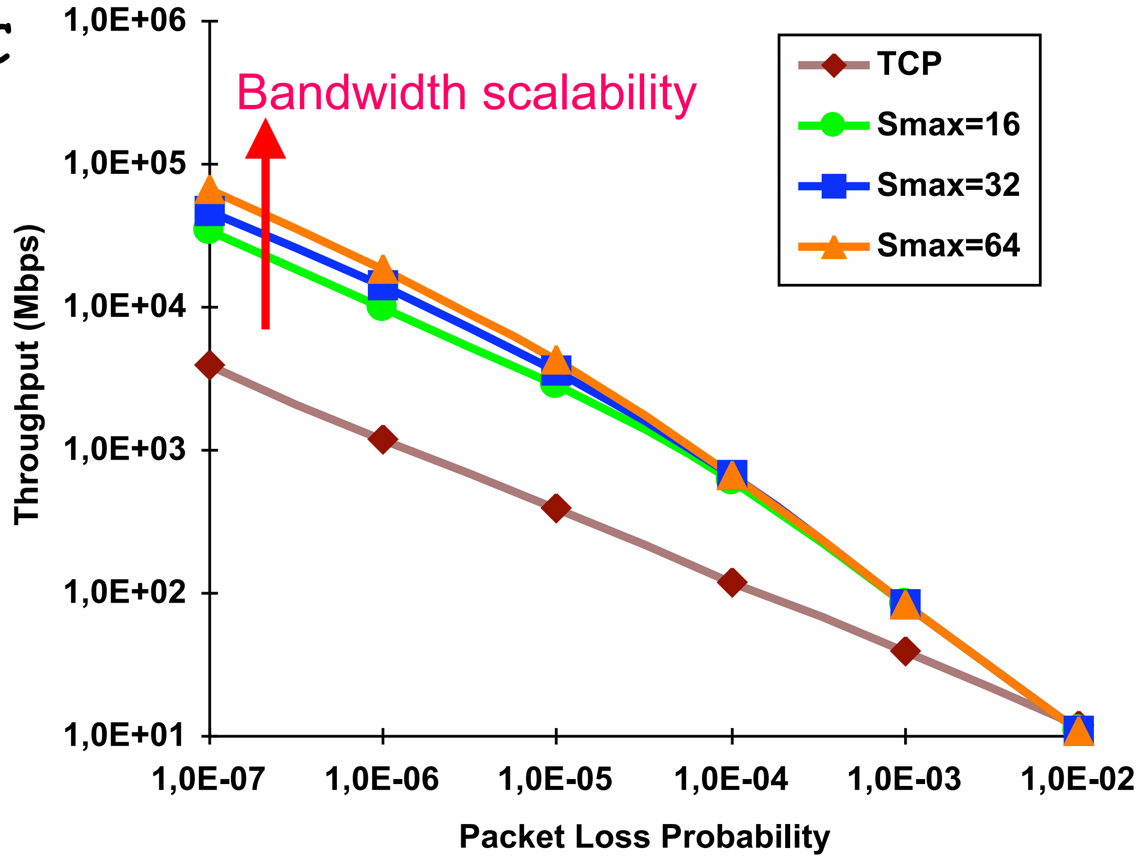

Source Injong Rhee, Lisong Xu

# 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



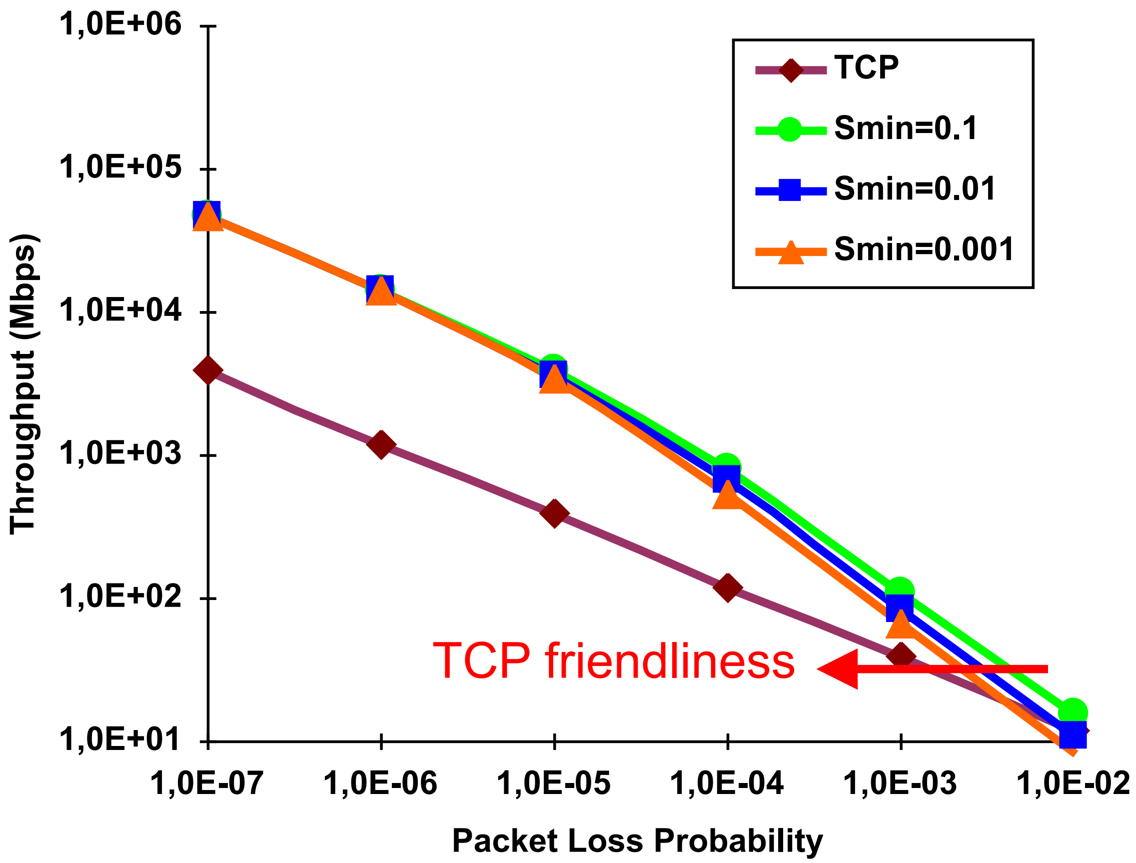

Source Injong Rhee, Lisong Xu

# Setting Smin

- Response Function of BIC on low-speed networks

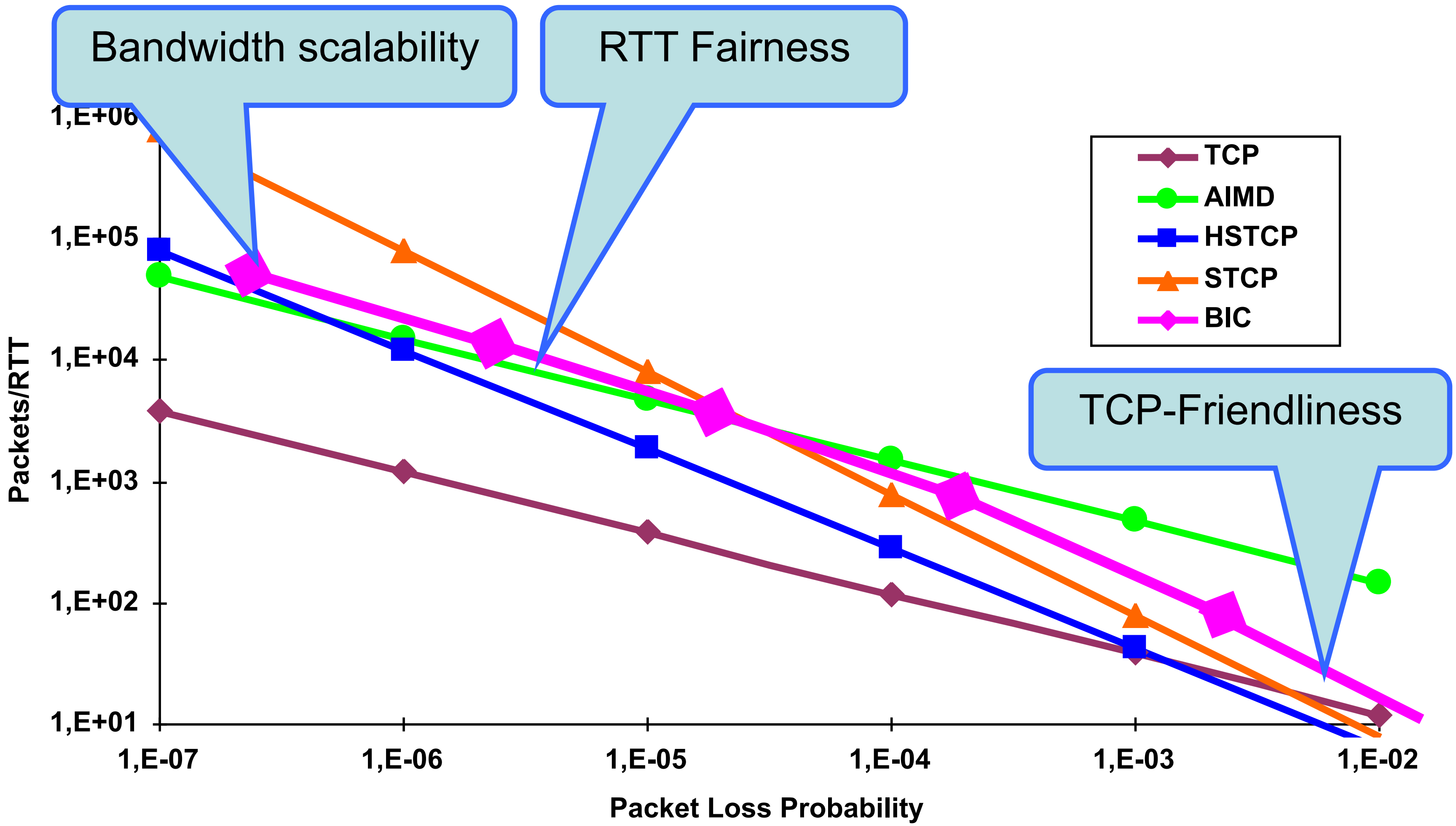
$$R = \frac{MSS}{RTT} f(p, S_{\min})$$

- TCP-friendliness of BIC depends only on Smin




Source Injong Rhee, Lisong Xu

# Response Functions

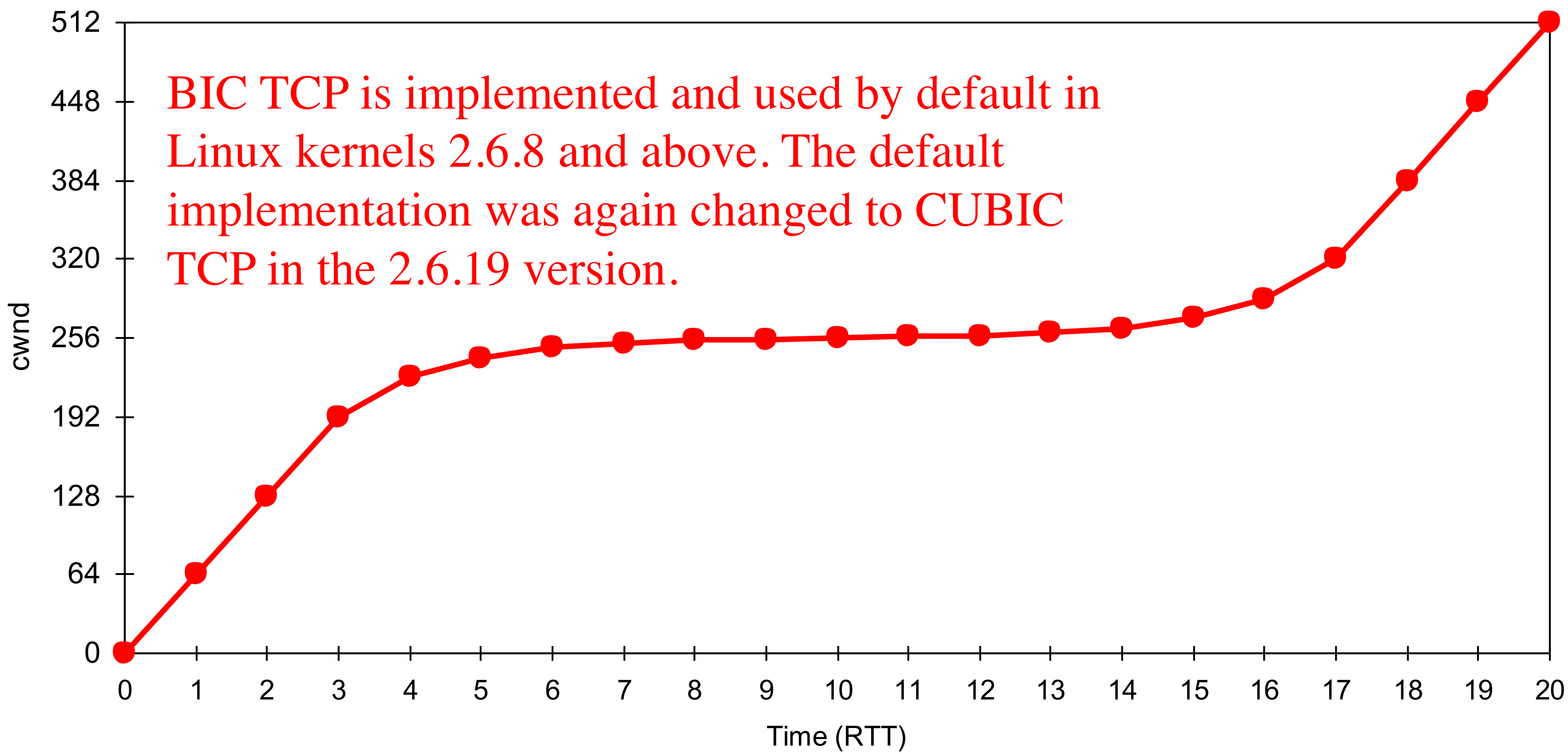



Source Injong Rhee, Lisong Xu

# CUBIC

$$Cwnd = W_{\max} + C(t-K)^3$$

BIC TCP is implemented and used by default in Linux kernels 2.6.8 and above. The default implementation was again changed to CUBIC TCP in the 2.6.19 version.

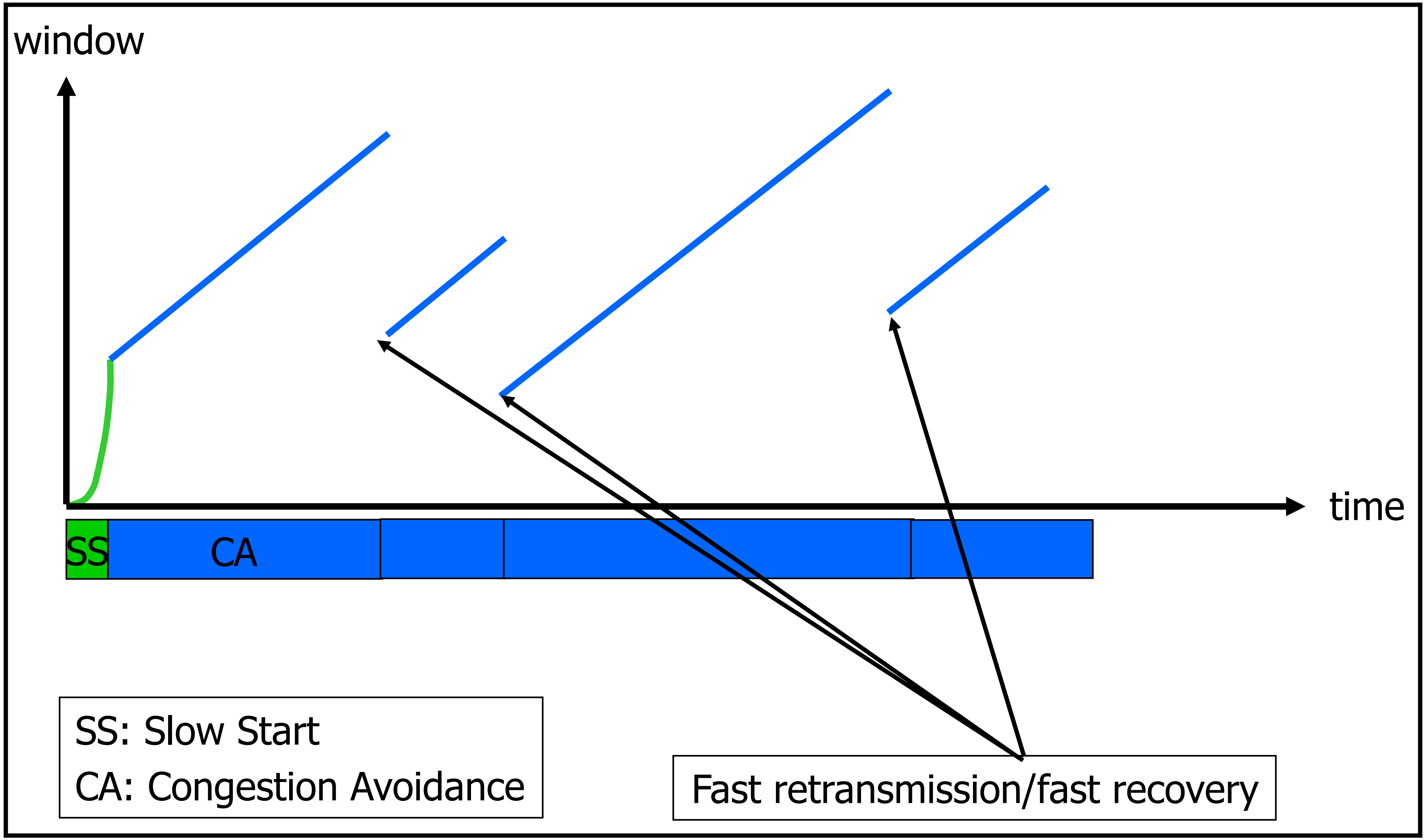



Source Injong Rhee, Lisong Xu

# Loss-based vs Delay-based

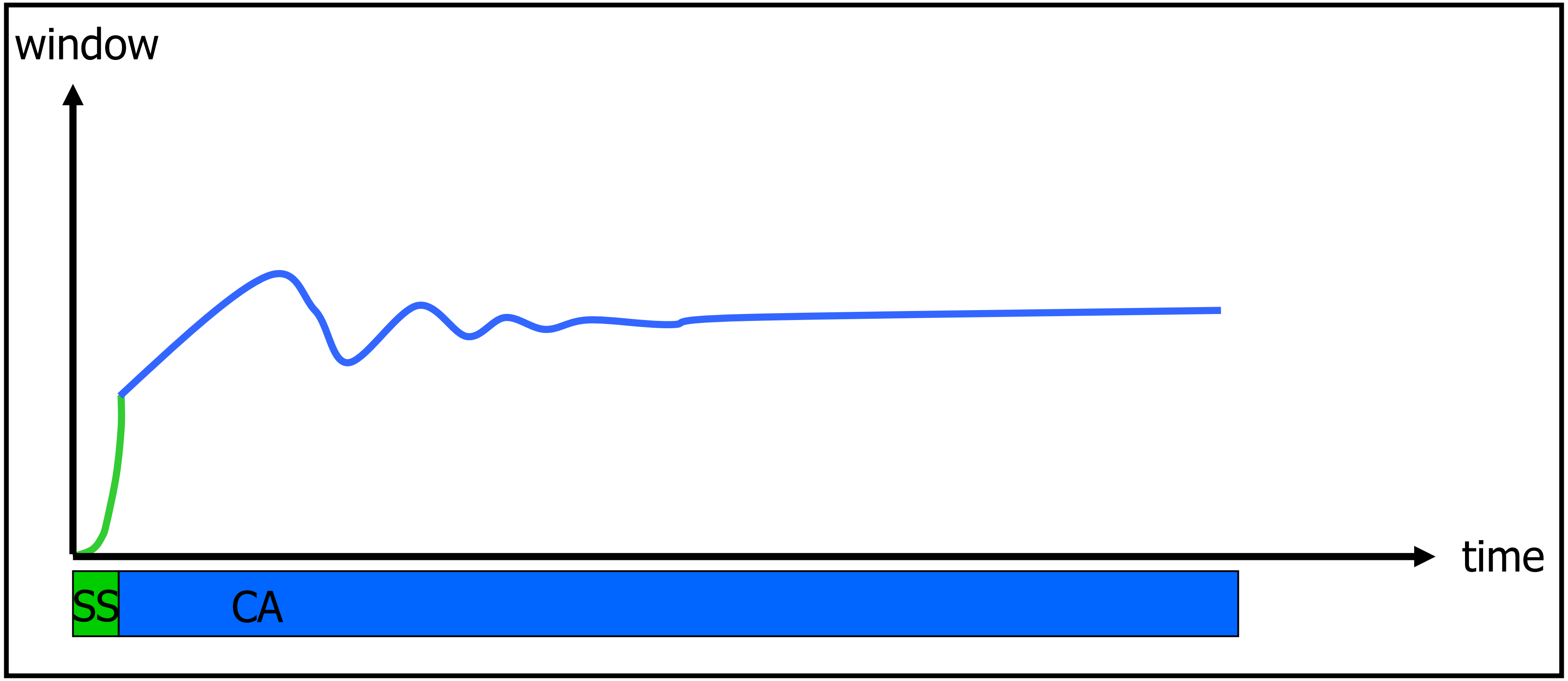
- Most of TCP approaches uses loss-based factor to control cwnd's growth (TCP, HSTCP, BIC)
- A delay-based approach typically uses the RTT increases/decrease to decrease/increase cwnd
- When RTT increases, there is a high probability that packets are backlogged in router's buffer, indicating congestion in a near future

# Loss-Based: TCP Reno



# Delay-based: TCP Vegas

(Brakmo & Peterson 1994)




- Converges, no retransmission
- … provided buffer is large enough

# Compound TCP

- Compound TCP incorporates a delay-based factor in addition to the loss-based factor
- 2 window state variables
  - Cwnd
  - Dwnd: delay window
- Win=min(cwnd+dwnd, $a_{dvertised}$wnd)
- Cwnd updated as standard TCP

# Congestion Control in CTCP (1)

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

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

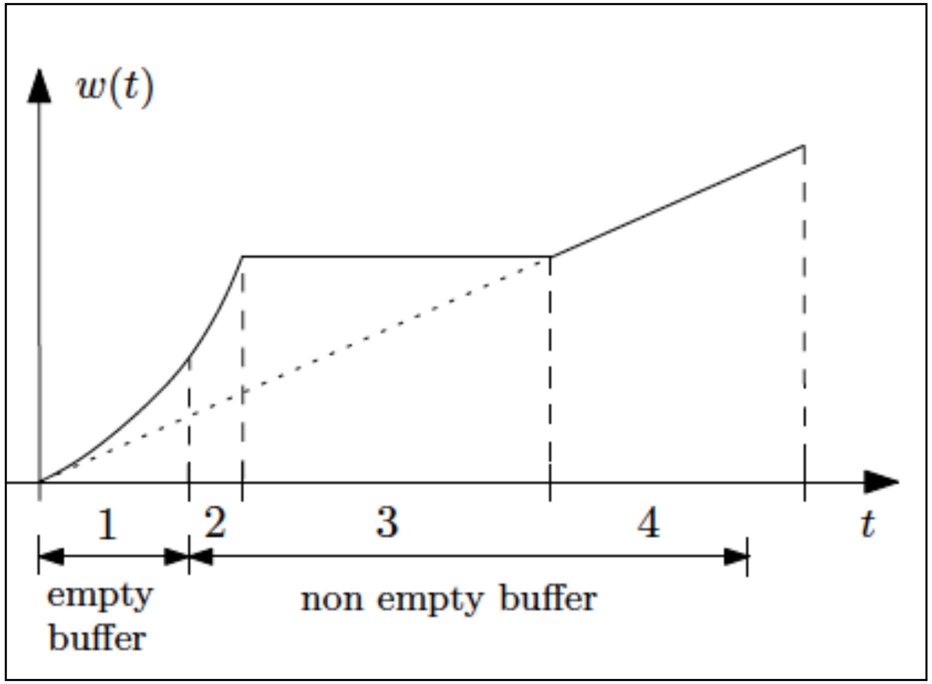
- Control functions

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

$(.)^+ = \max(.,0)$

# Congestion Control in CTCP (2)

- Reno
  - $W_{i+1}=W_i+1$
- CTCP ($\xi=1$)
  - $W_{i+1}=W_i+\alpha W_i^k$, ❶❷
  - $W_{i+1}=W_i$, ❸
  - $W_{i+1}=W_i+1$, ❹
- $\Delta_i$: queue size estimation
- If $\Delta_i > \gamma$, move from ❷ to ❸.



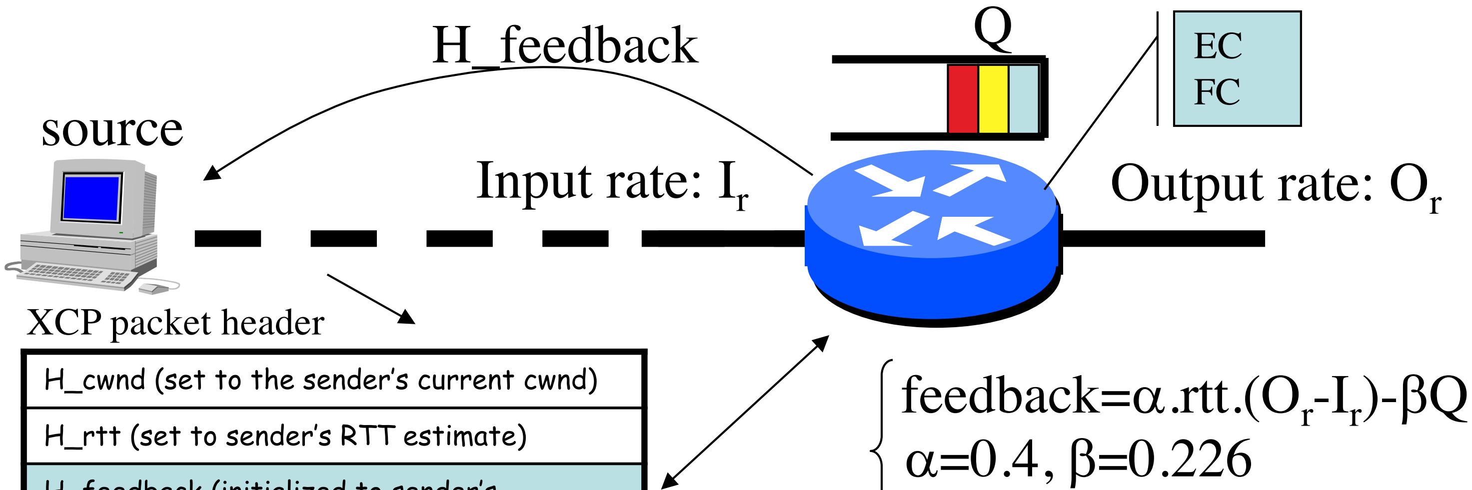
# CTCP and Windows Vista

- CTCP is enabled by default in computers running beta versions of Windows Server 2008 and disabled by default in computers running Windows Vista. CTCP can be enabled with the command

```
netsh interface tcp set global congestionprovider=ctcp
```

# XCP [Katabi02]

- XCP is a router-assisted solution, generalized the ECN concepts (FR, TCP-ECN)
- XCP routers can compute the available bandwidth by monitoring the input rate and the output rate
- Feedback is sent back to the source in special fields of the packet header




| |
|---|
| H_cwnd (set to the sender's current cwnd) |
| H_rtt (set to sender's RTT estimate) |
| H_feedback (initialized to sender's demands) |

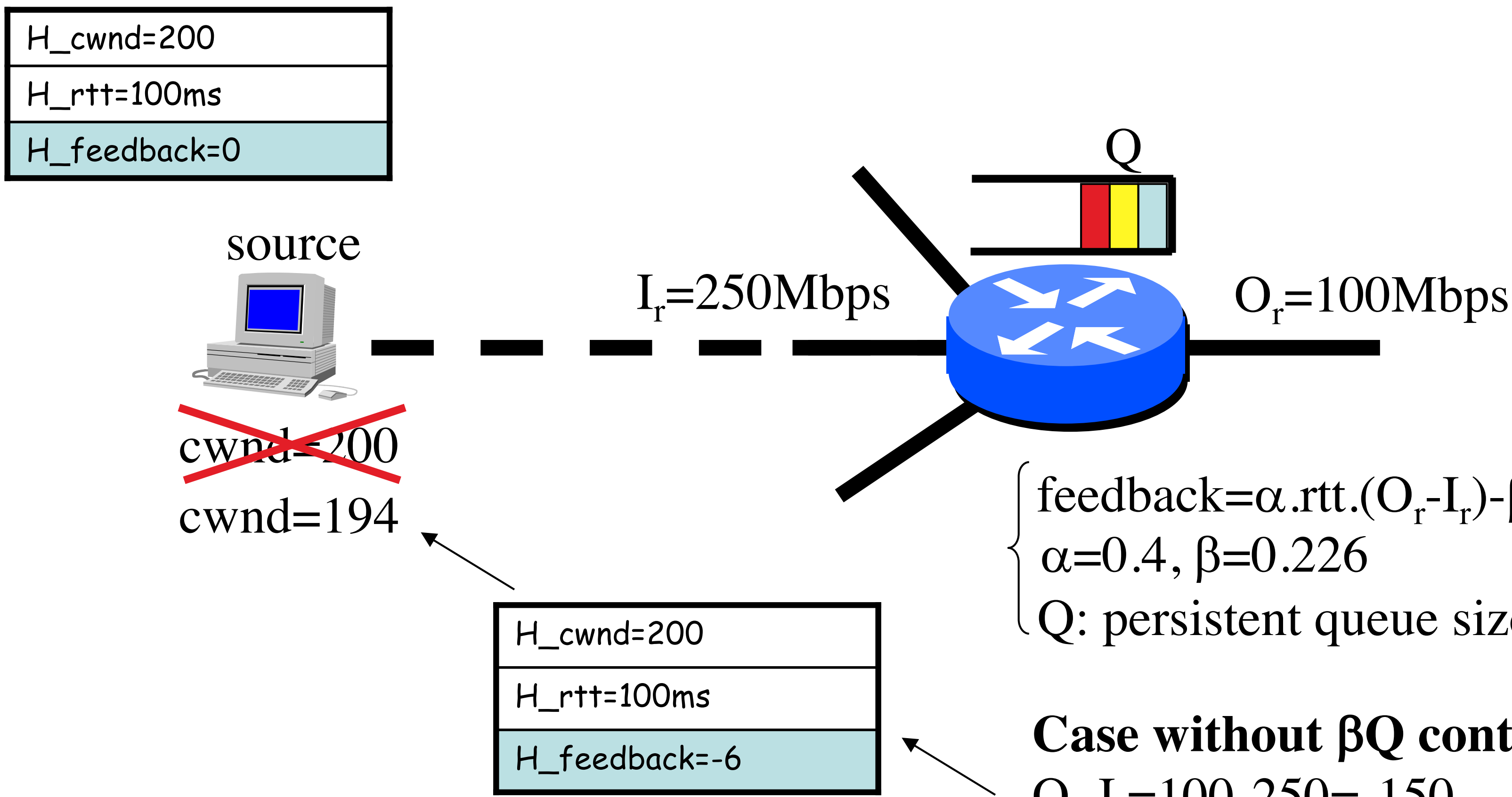
$$feedback = \alpha.rtt.(O_r - I_r) - \beta Q$$

$\alpha=0.4$, $\beta=0.226$

Q: persistent queue size

# XCP in action

Feedback value represents a window increment/decrement




$feedback=\alpha.rtt.(O_r-I_r)-\beta Q$

$\alpha=0.4$, $\beta=0.226$

Q: persistent queue size

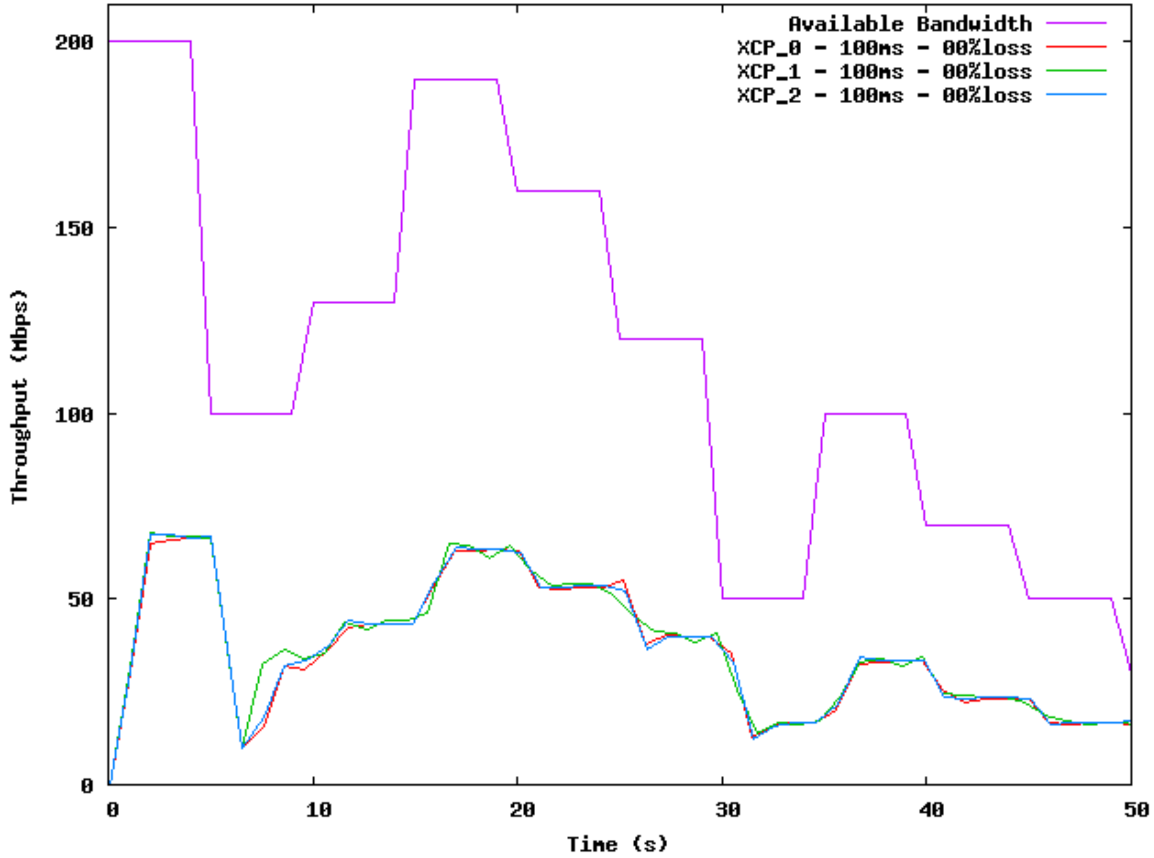
**Case without βQ contribution**

$O_r-I_r=100-250=-150$

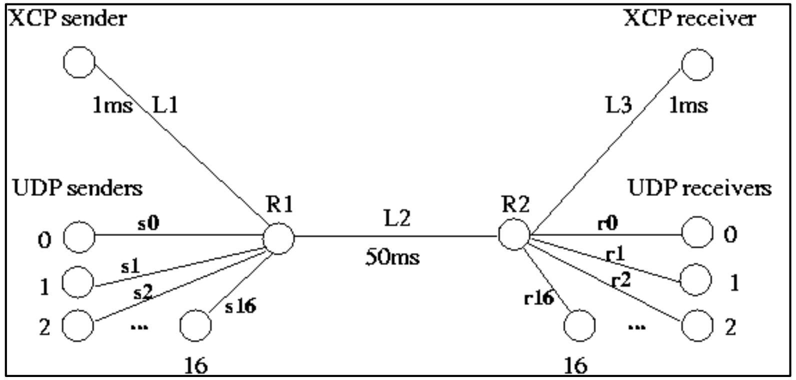
feedback=-6

# XCP

## Variable bandwidth environments

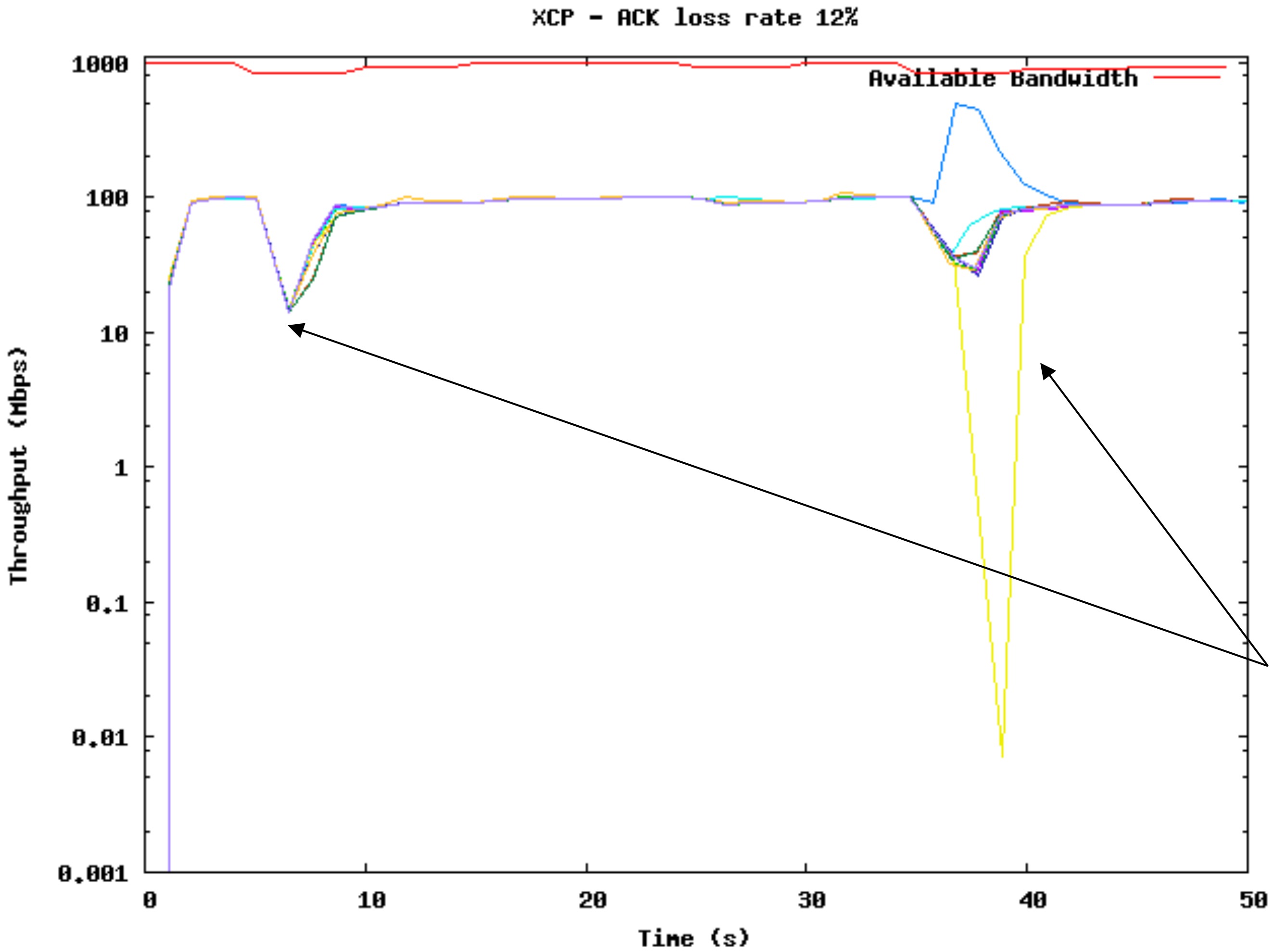



Good fairness and stability even in variable bandwidth environments



# XCP-r [Pacheco&Pham05]

## A more robust version of XCP

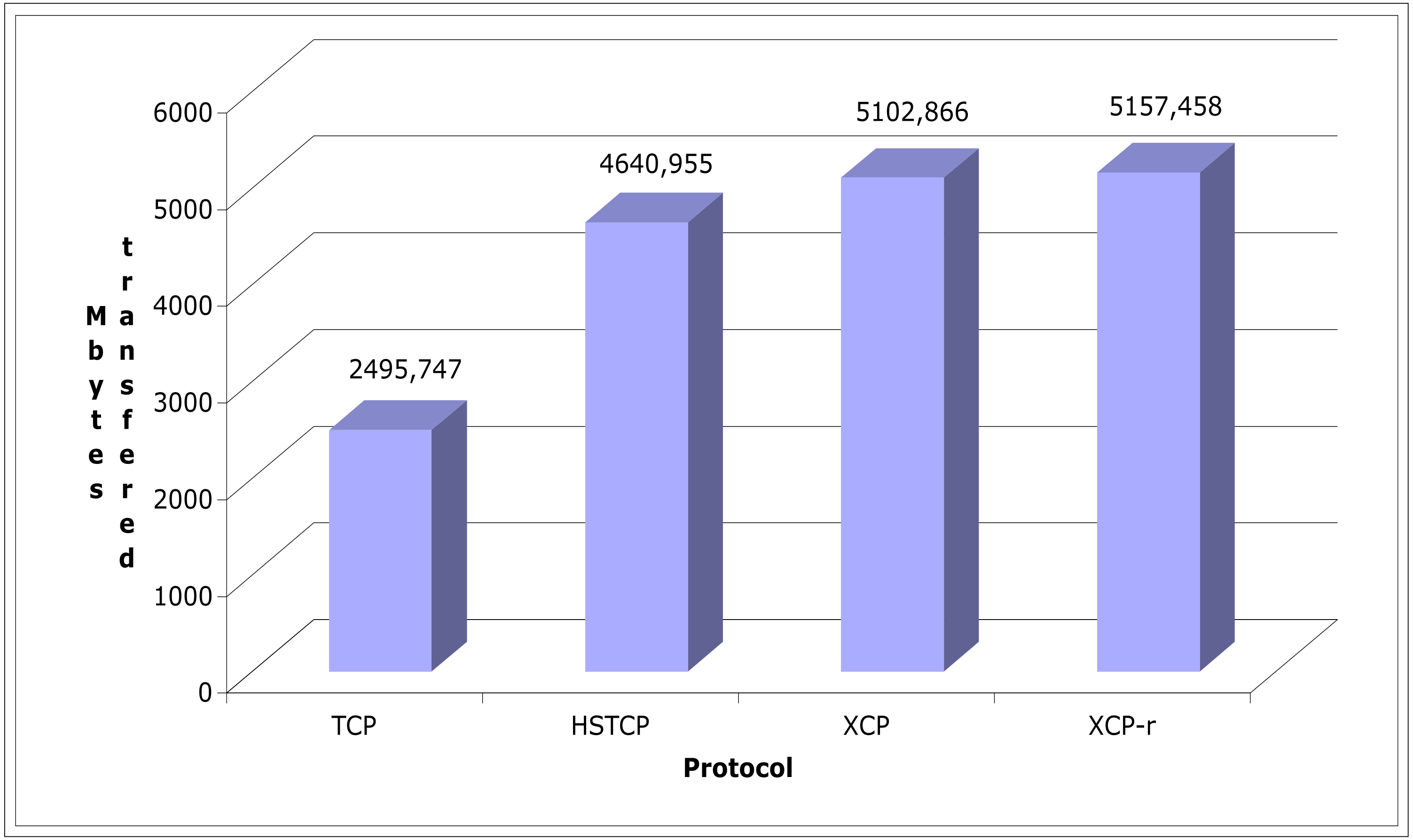



10 flows sharing a 1Gbps link

Fast recovery after the timeouts and better fairness level

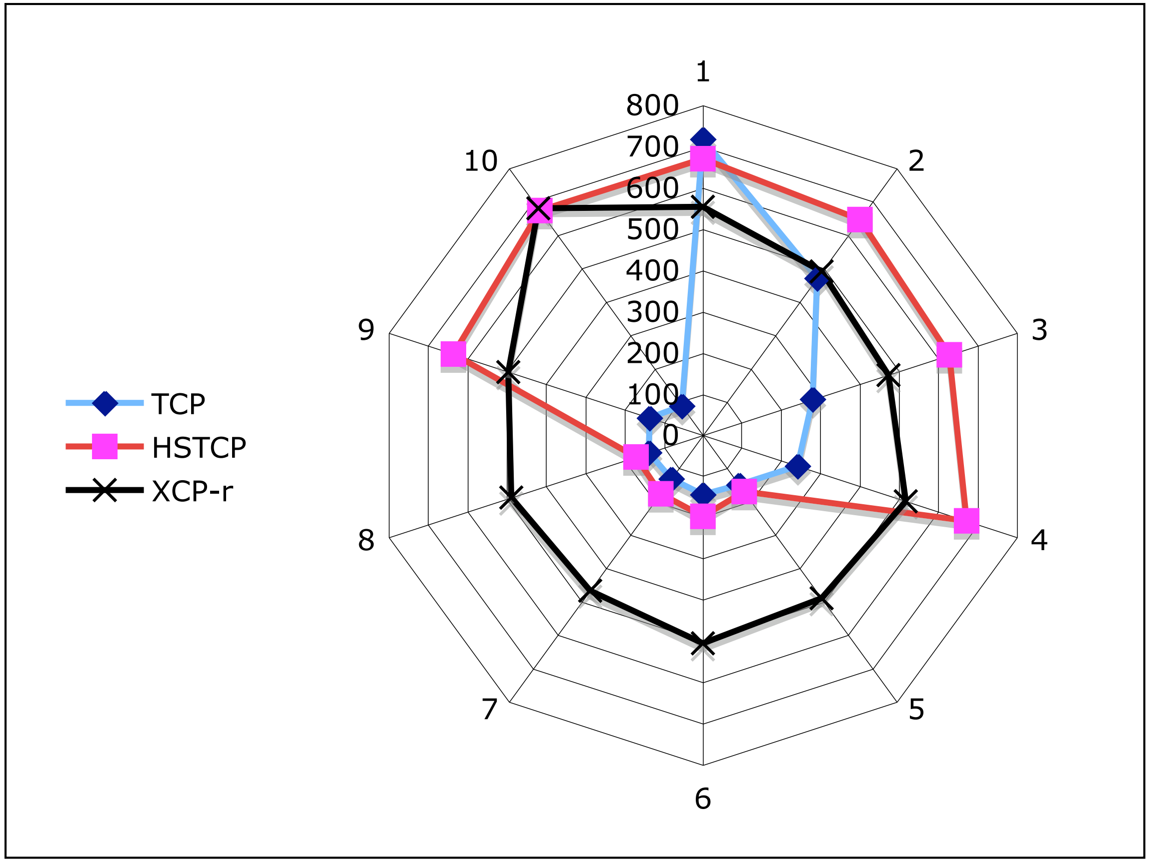
# XCP-r performance

Amount of data transfered in 50s, 10 flows, 1Gbps link, 200ms RTT



# XCP-r fairness

TCP and HSTCP are not really fair...



# Nothing is perfect :-(

- Multiple or parallel streams
  - How many streams?
  - OS high overheads
  - Tradeoff between window size and number of streams
- New protocol
  - Fairness issues?
  - Deployment issues?
  - Still too early to know the side effects

# Hostile environments

- Asymetric networks
  - Satellite links & terrestrial links
- Wireless (WiFi, WiMax, 5G)
  - High loss probability
  - Losses ≠congestions
- Ad-Hoc
  - Small capacity
  - High mobility
- Wireless Sensor Networks/IoT
  - **High resource constraints**

# Conclusions

- Understanding the dark side allows to move forwards!
- However…