IEEE INFOCOM 2001

Tutorial T4

TCP Congestion Controls: Algorithms and Models

Steven Low (Caltech)
Matthew Roughan (AT&T Labs - Research)

Sunday, 22 April, 2001 - Full Day

TCP Congestion Controls

Steven Low
CS & EE Depts, Caltech
netlab.caltech.edu

Matthew Roughan
AT&T Labs - Research
roughan@research.att.com

INFOCOM 2001, April 2001

Acknowledgments

- S. Athuraliya, D. Lapsley, V. Li, Q. Yin (UMelb)
- J. Doyle (Caltech), F. Paganini (UCLA)
- Darryl Veitch, D. Hayes, Ericsson-Melbourne University Lab (EMULab), Australia
- Ashok Erramilli (QNetworx)

TCP/IP

Primary protocols used in the Internet

- IP (Internet Protocol)
  - network layer
- TCP (Transmission Control Protocol)
  - transport layer
  - flow controlled
- TCP/IP refers to more than just TCP & IP
  - UDP: transport layer, not flow controlled
  - control & application protocols: ICMP, ARP, HTTP, ...

Why Flow Control?

- October 1986 Internet had its first congestion collapse
- Link LBL to UC Berkeley
  - 400 yards, 3 hops, 32 Kbps
  - throughput dropped to 40 bps
  - factor of ~1000 drop!
- 1988, Van Jacobson proposed TCP flow control

Outline

- Introduction
- TCP algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP models
  - Renewal model
  - Duality model
  - Feedback control model

Schedule

9:00 – 10:45	Introduction & TCP algorithms
10:45 – 11:00	Break
11:00 – 12:00	TCP Algorithms (Vegas, RED, …)
12:00 – 1:00	Lunch
1:00 – 2:00	TCP models (1/sqrt(p) law, fixed point)
2:00 – 2:20	Break
2:20 – 3:20	TCP models (duality)
3:20 – 3:40	Break
3:40 – 4:40	TCP models (dynamics)
4:40 – 5:00	Discussion

Steven

Matthew

Part 0
Introduction

IP

- Packet switched
- Datagram service
  - Unreliable (best effort)
  - Simple, robust
- Heterogeneous
- Dumb network, intelligent terminals

Compared with PSTN

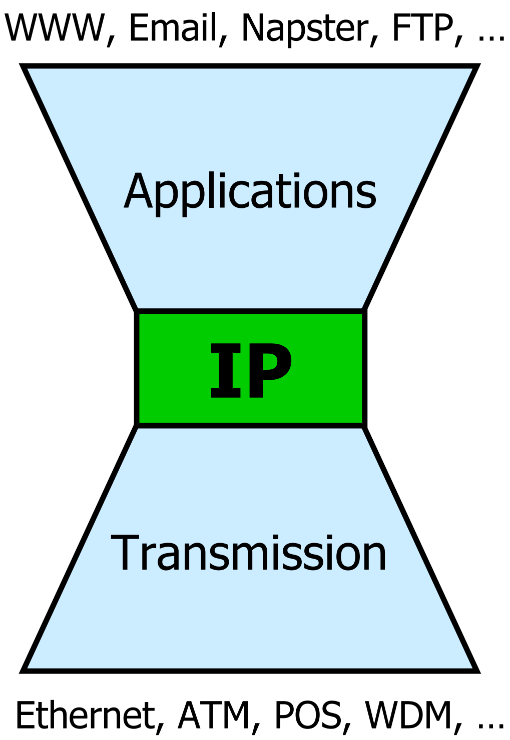
TCP

- Packet switched
- End-to-end (like a virtual circuit service)
  - Reliable, in order delivery of a byte stream
  - Reliability through ACKs
  - Multiplexing
- Flow control: use bandwidth efficiently
- Robustness Principle

  be conservative in what you do,

  be liberal in what you accept from others

Success of IP



Simple/Robust

- Robustness against failure
- Robustness against technological evolutions
- Provides a service to applications
  - Doesn't tell applications what to do

Quality of Service

- Can we provide QoS with simplicity?
- Not with current TCP…
- … but we can fix it!

IETF

- Internet Engineering Task Force
  - Standards organisation for Internet
  - Publishes RFCs - Requests For Comment
    - standards track: proposed, draft, Internet
    - non-standards track: experimental, informational
    - best current practice
    - poetry/humour (RFC 1149: Standard for the transmission of IP datagrams on avian carriers)
  - TCP should obey RFC
    - no means of enforcement
    - some versions have not followed RFC
- http://www.ietf.org/index.html

RFCs of note

- RFC 791: Internet Protocol
- RFC 793: Transmission Control Protocol
- RFC 1180: A TCP/IP Tutorial
- RFC 2581: TCP Congestion Control
- RFC 2525: Known TCP Implementation Problems
- RFC 1323: TCP Extensions for High Performance
- RFC 2026: Internet standards process

Other Key References

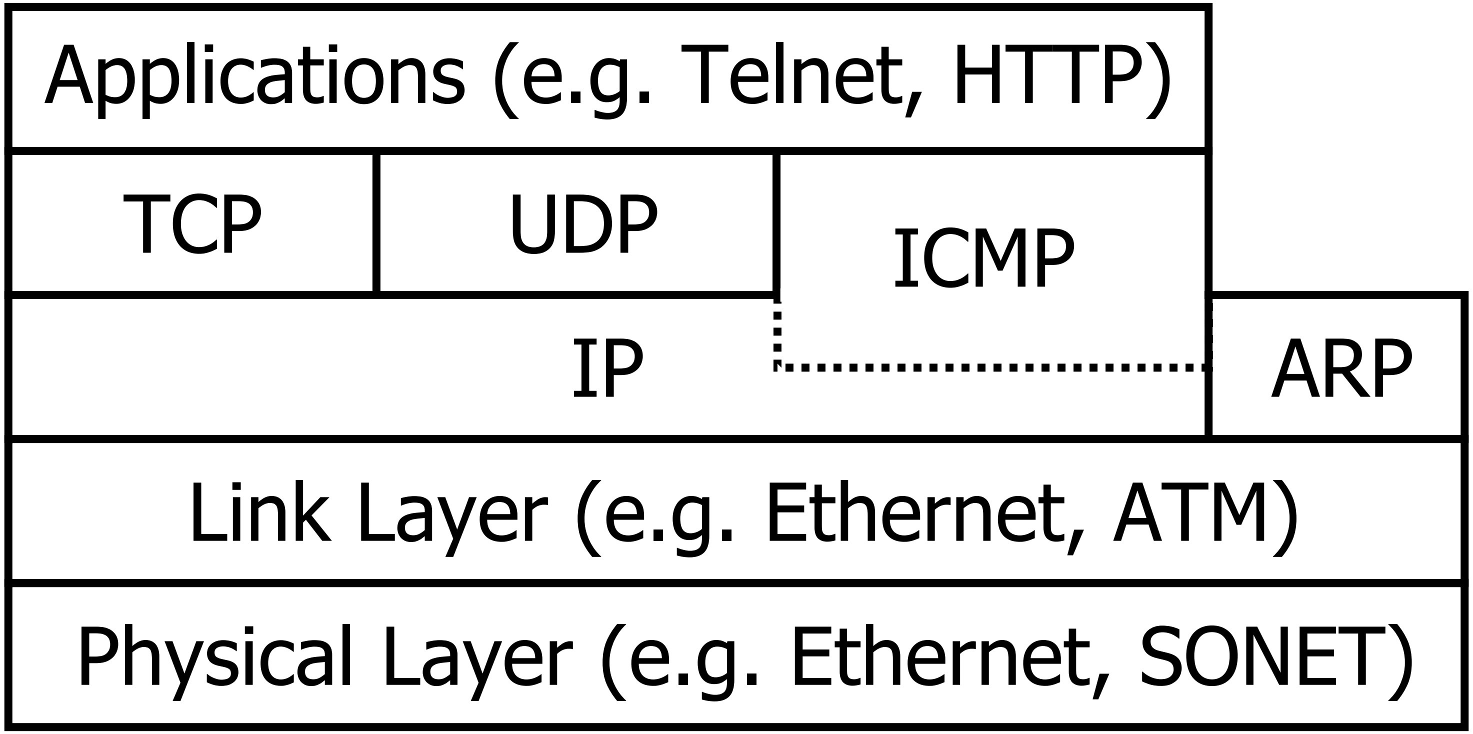
- W. Stevens (and Wright), “TCP/IP Illustrated”, Vol. 1-2

  Addison-Wesley, 1994
- Vern Paxson, “Measurements and Analysis of End-to-End Internet Dynamics”

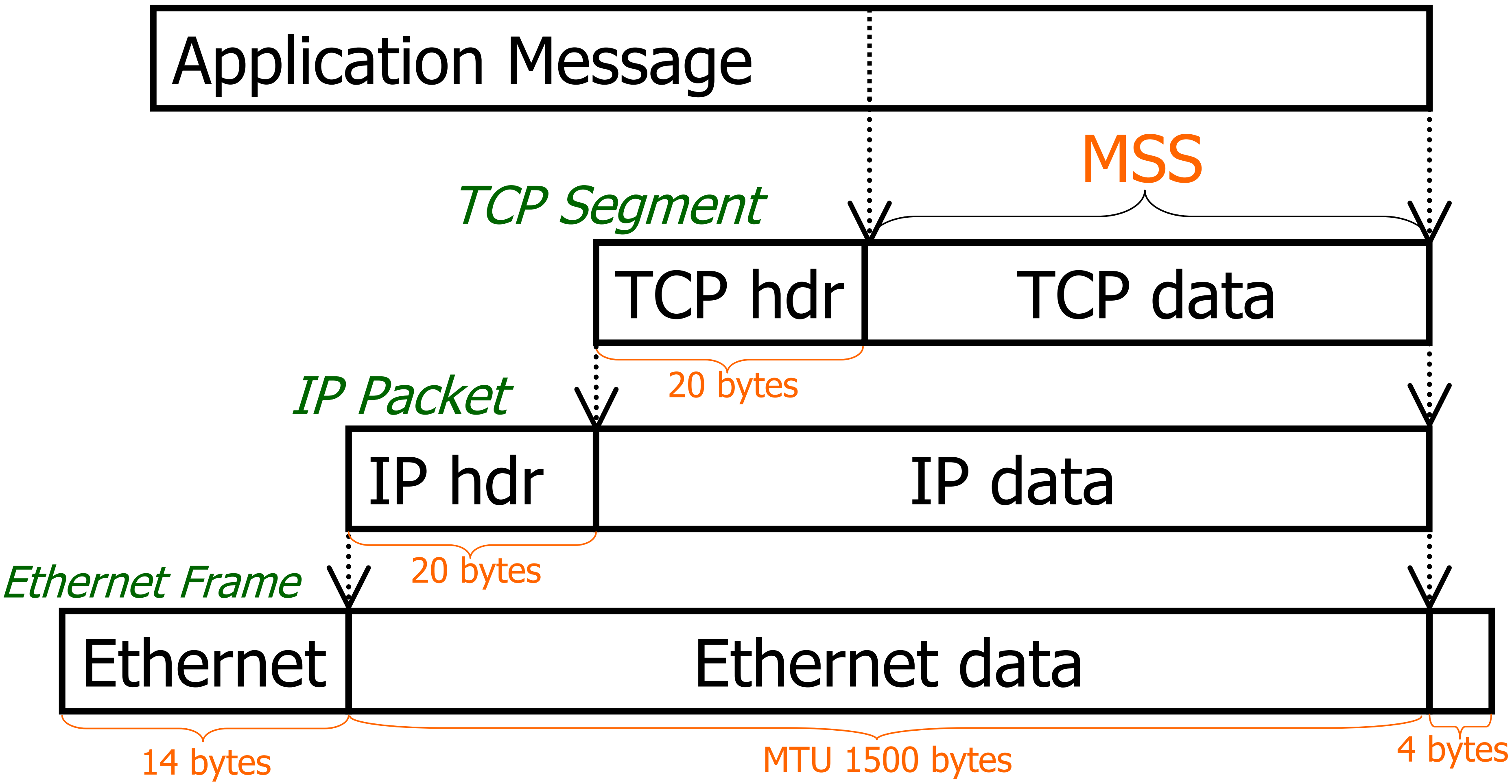
  PhD Thesis
- Van Jacobson, “Congestion Avoidance and Control”

  SIGCOMM’88

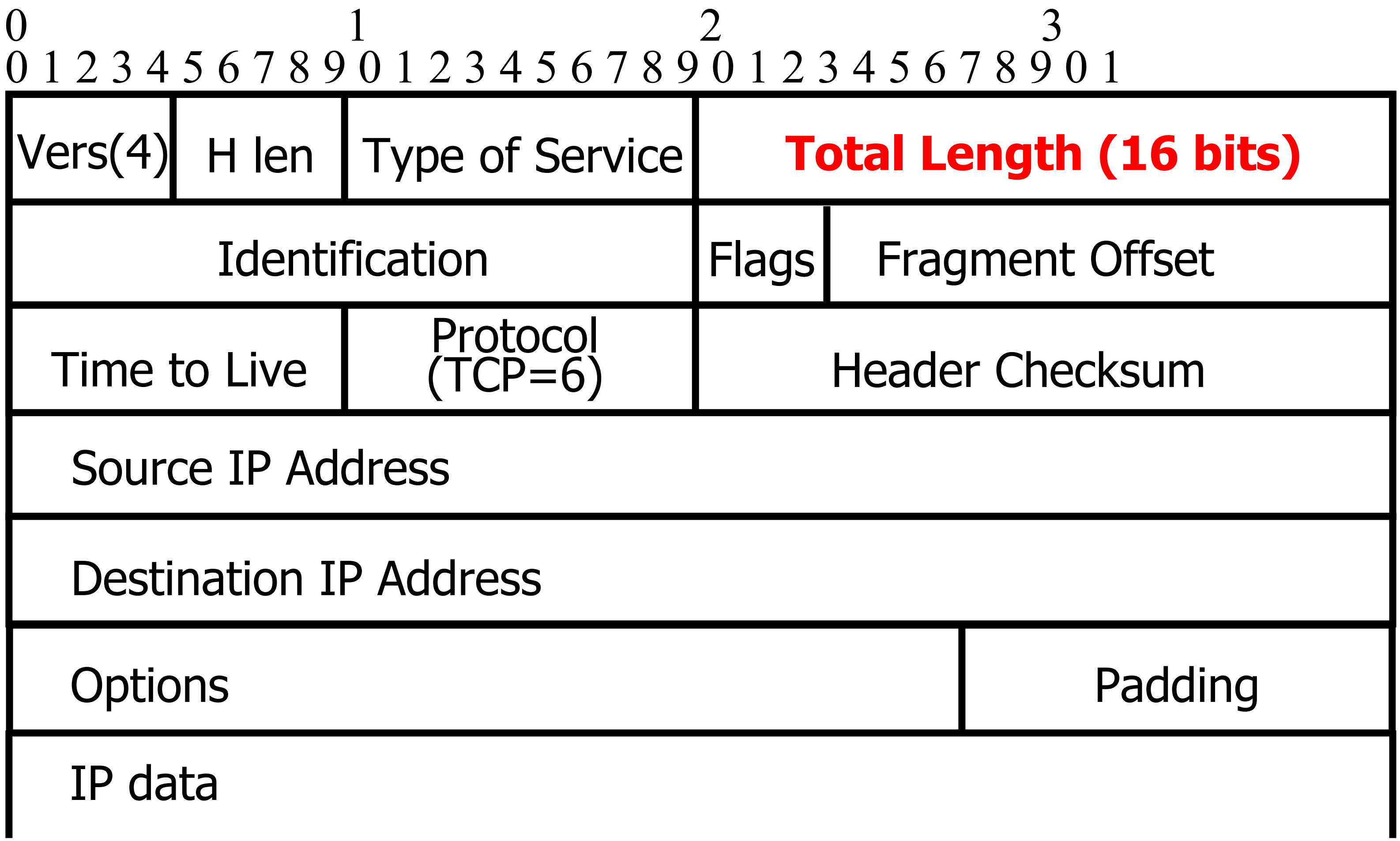
TCP/IP Protocol Stack



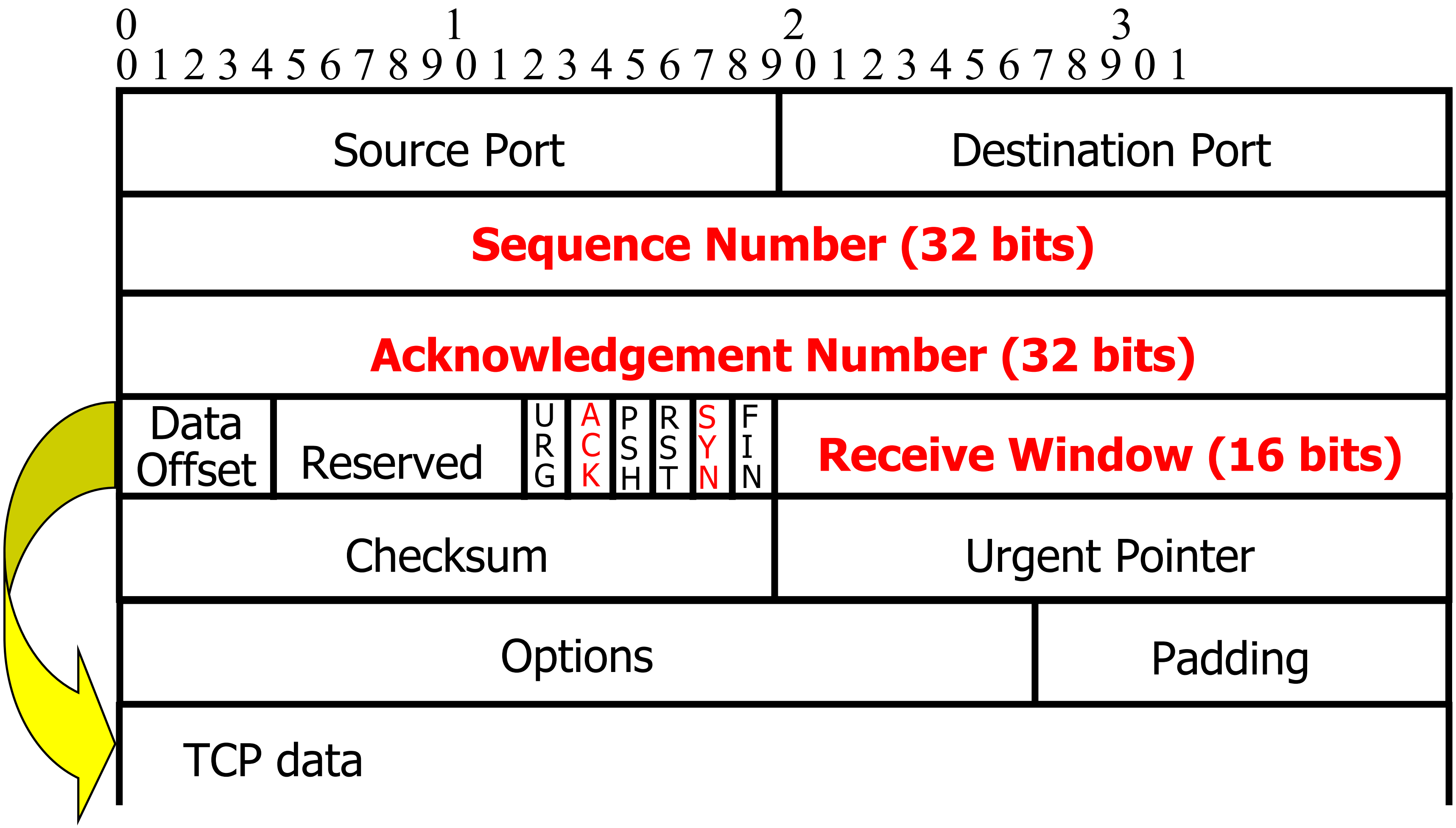
Packet Terminology



IP Header


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Vers(4)
H len
Type of Service
Total Length (16 bits)
Identification
Flags
Fragment Offset
Time to Live
Protocol (TCP=6)
Header Checksum
Source IP Address
Destination IP Address
Options
Padding
IP data

TCP Header



0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Source Port
Destination Port
Sequence Number (32 bits)
Acknowledgement Number (32 bits)
Data Offset
Reserved
URG
ACK
PSH
RST
SYN
FIN
Receive Window (16 bits)
Checksum
Urgent Pointer
Options
Padding
TCP data

TCP versions

- TCP is not perfectly homogenous (200+)

4.2 BSD first widely available release of TCP/IP (1983)

4.3 BSD (1986)

4.3 BSD **Tahoe** (1988)

SunOS 4.1.3,4.1.4

4.3 BSD **Reno** (1990)

IRIX

HP/UX

Digital OSF

4.4 BSD (1993)

NewReno (1999)

.....

Windows 95

Windows NT

Solaris 2.3,2.4

Linux 1.0

Vegas (1994)

Simulation

- ns-2: http://www.isi.edu/nsnam/ns/index.html
  - Wide variety of protocols
  - Widely used/tested
- SSFNET: http://www.ssfnet.org/homePage.html
  - Scalable to very large networks
- Care should be taken in simulations!
  - Multiple independent simulations
    - confidence intervals
    - transient analysis – make sure simulations are long enough
  - Wide parameter ranges
- All simulations involve approximation

Other Tools

- `tcpdump`
  - Get packet headers from real network traffic
- `tcpanaly` (V.Paxson, 1997)
  - Analysis of TCP sessions from `tcpdump`
- `traceroute`
  - Find routing of packets
- RFC 2398
- `http://www.caida.org/tools/`

Part I
Algorithms

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model
  - Feedback control model

Early TCP

- Pre-1988
- Go-back-N ARQ
  - Detects loss from timeout
  - Retransmits from lost packet onward
- Receiver window flow control
  - Prevent overflows at receive buffer
- Flow control: self-clocking

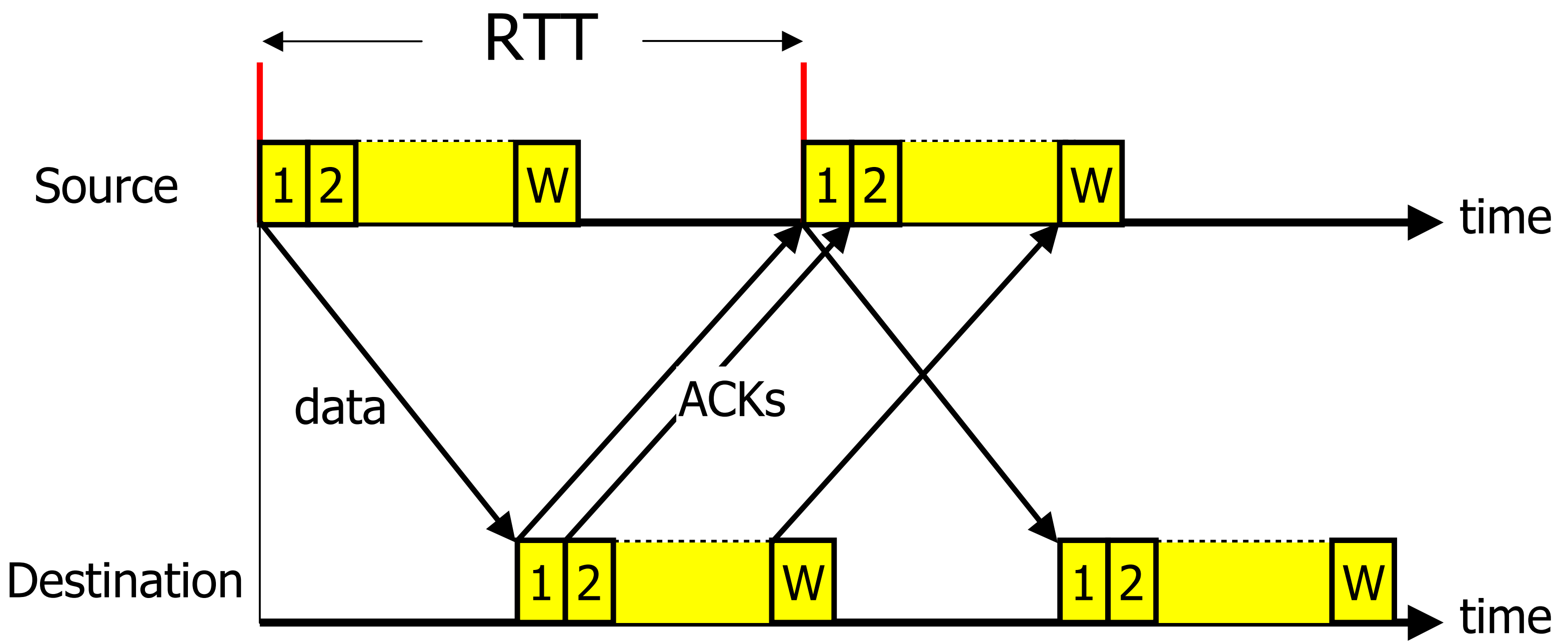
Why Flow Control?

- October 1986, Internet had its first congestion collapse
- Link LBL to UC Berkeley
  - 400 yards, 3 hops, 32 Kbps
  - throughput dropped to 40 bps
  - factor of ~1000 drop!
- 1988, Van Jacobson proposed TCP flow control

Flow Control Issues

	TCP (Reno)
Efficiency	reasonable (big buffers)
Stability	"yes"
Convergence	
Responsiveness	reasonable (after packets lost)
Smoothness	no
Fairness	no
Distribution	
Centralized/distributed	distributed
End-to-end/network	end-to-end

Window Flow Control



- ~ W packets per RTT
- Lost packet detected by missing ACK

Source Rate

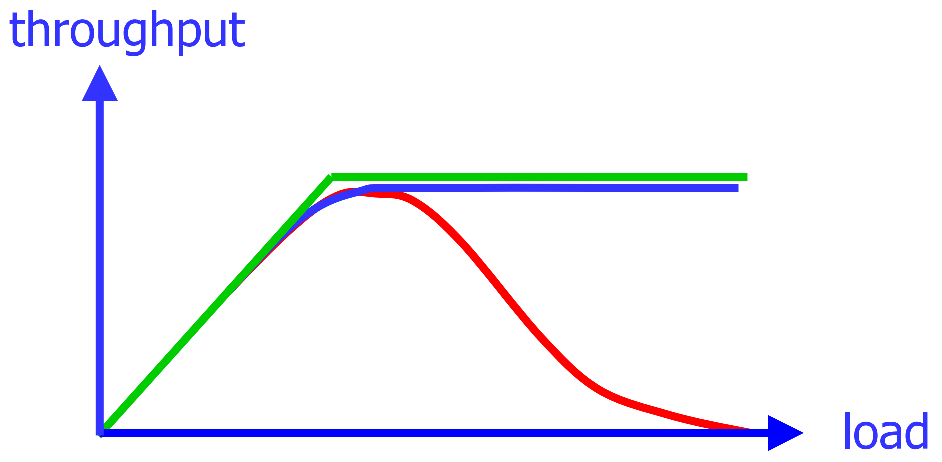
- Limit the number of packets in the network to window W
- Source rate = $\dfrac{W \times MSS}{RTT}$ bps
- If W too small then rate « capacity

  If W too big then rate > capacity

  => congestion

Effect of Congestion

- Packet loss
- Retransmission
- Reduced throughput
- Congestion collapse due to
  - Unnecessarily retransmitted packets
  - Undelivered or unusable packets
- Congestion may continue after the overload!



Congestion Control

- TCP seeks to
  - Achieve high utilization
  - Avoid congestion
  - Share bandwidth
- Window flow control
  - Source rate = $\dfrac{W}{RTT}$ packets/sec
  - Adapt W to network (and conditions)

    $W = BW \times RTT$

Example Networks

Network	Bandwidth	RTT	BW x delay
56k dial up	56 kbps	100 ms	700 B
10baseT Ethernet	10 Mbps	3 ms	3,750 B
T1 (satellite)	1.54 Mbps	500 ms	96 kB
OC48 (point-to-point)	2.5 Gbps	20 ms	6 MB
OC192 (transcontinental)	10 Gbps	60 ms	75 MB

Range covers 8 orders of magnitude

TCP Window Flow Controls

- Receiver flow control
  - Avoid overloading receiver
  - Set by receiver
  - awnd: receiver (advertised) window
- Network flow control
  - Avoid overloading network
  - Set by sender
  - Infer available network capacity
  - cwnd: congestion window
- Set W = min (cwnd, awnd)

Receiver Flow Control

- Receiver advertises awnd with each ACK
- Window awnd
  - closed when data is received and ack'd
  - opened when data is read
- Size of awnd can be *the* performance limit (e.g. on a LAN)
  - sensible default ~16kB

Network Flow Control

- Source calculates cwnd from indication of network congestion
- Congestion indications
  - **Losses**
  - Delay
  - Marks
- Algorithms to calculate cwnd
  - Tahoe, Reno, Vegas, RED, REM …

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model
  - Feedback control model

TCP Congestion Controls

- Tahoe (Jacobson 1988)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- Reno (Jacobson 1990)
  - Fast Recovery
- Vegas (Brakmo & Peterson 1994)
  - New Congestion Avoidance
- RED (Floyd & Jacobson 1993)
  - Probabilistic marking
- REM (Athuraliya & Low 2000)
  - Clear buffer, match rate

Variants

- Tahoe & Reno
  - NewReno
  - SACK
  - Rate-halving
  - Mod.s for high performance
- AQM
  - RED, ARED, FRED, SRED
  - BLUE, SFB
  - REM

TCP Congestion Control

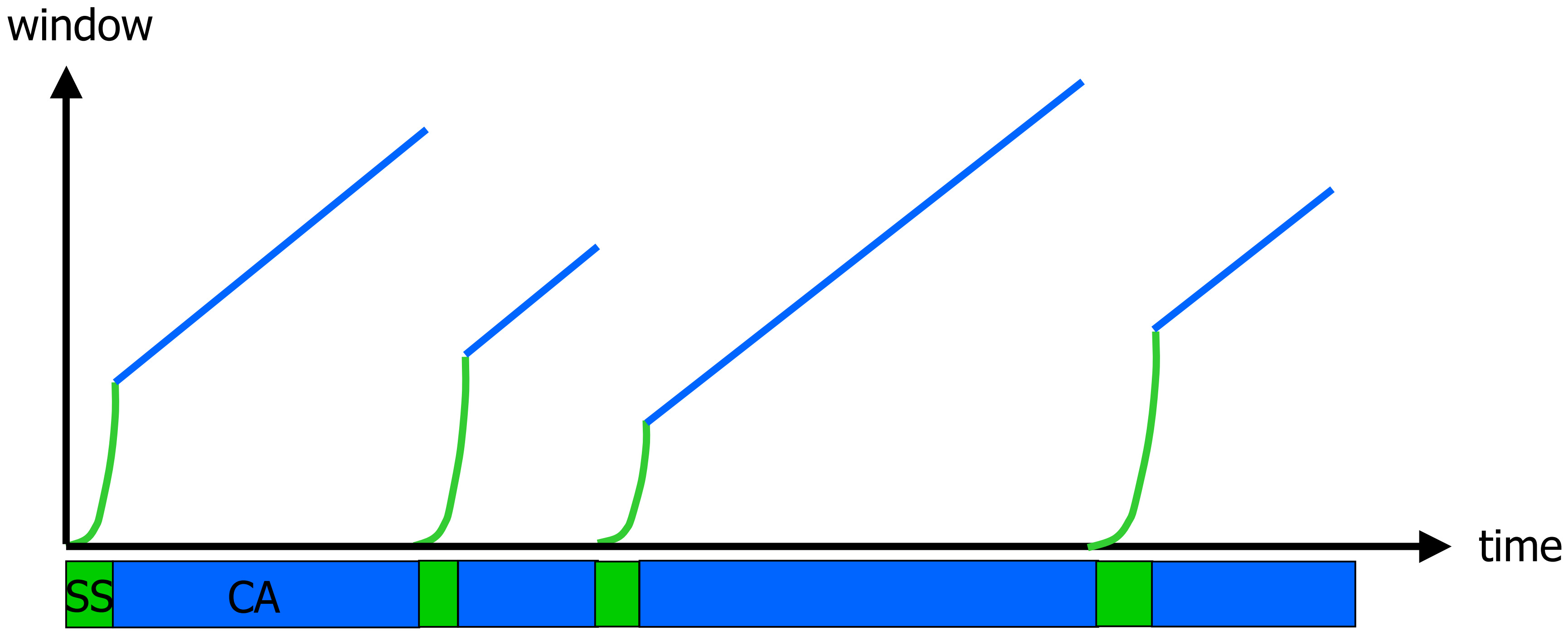
- Has four main parts
  - Slow Start (SS)
  - Congestion Avoidance (CA)
  - Fast Retransmit
  - Fast Recovery

Tahoe

Reno

- ssthresh: slow start threshold determines whether to use SS or CA
- Assume packet losses are caused by congestion

TCP Tahoe (Jacobson 1988)



SS: Slow Start

CA: Congestion Avoidance

Slow Start

- Start with cwnd = 1 (slow start)
- On each successful ACK increment cwnd

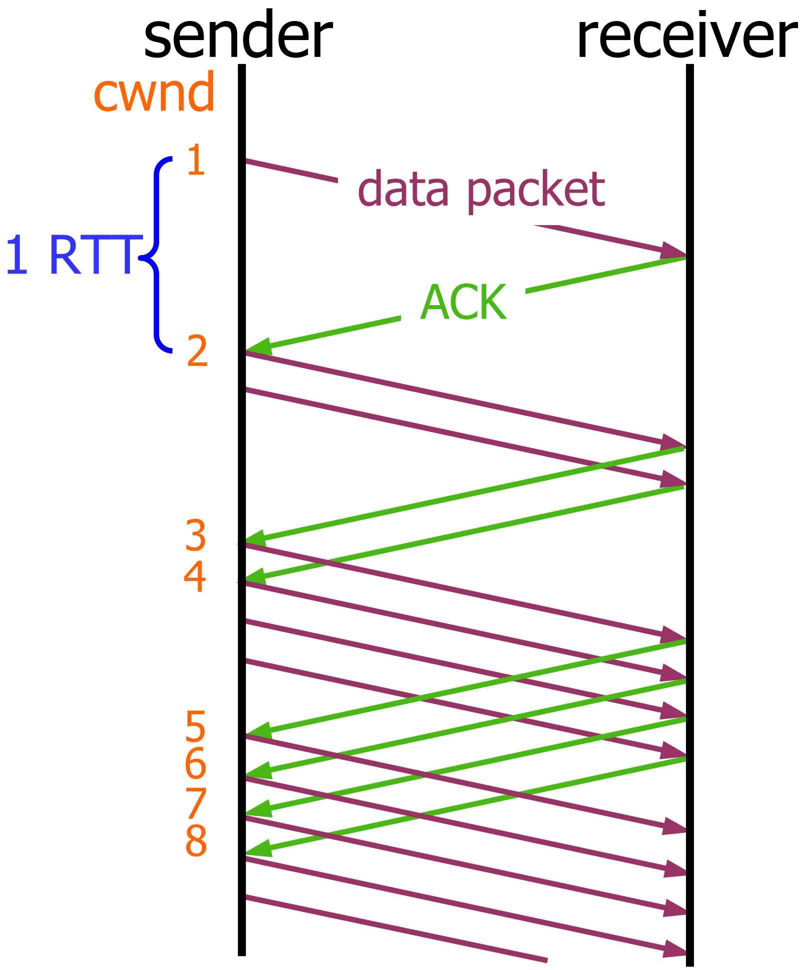
  $$\text{cwnd} \leftarrow \text{cnwd} + 1$$

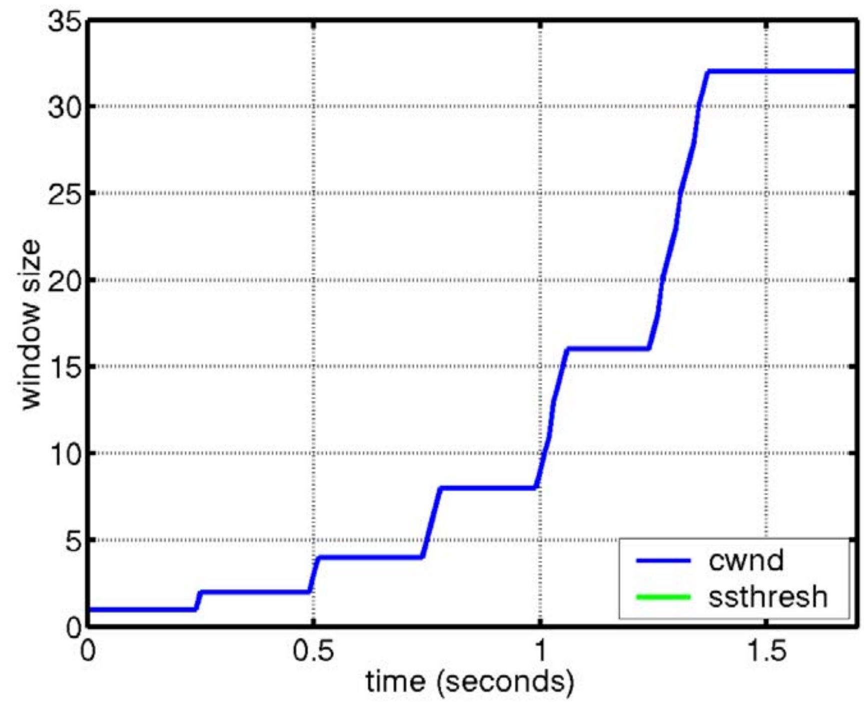
- Exponential growth of cwnd

  each RTT: $\text{cwnd} \leftarrow 2 \times \text{cwnd}$

- Enter CA when cwnd >= ssthresh

Slow Start





cwnd ← cwnd + 1 (for each ACK)

Congestion Avoidance

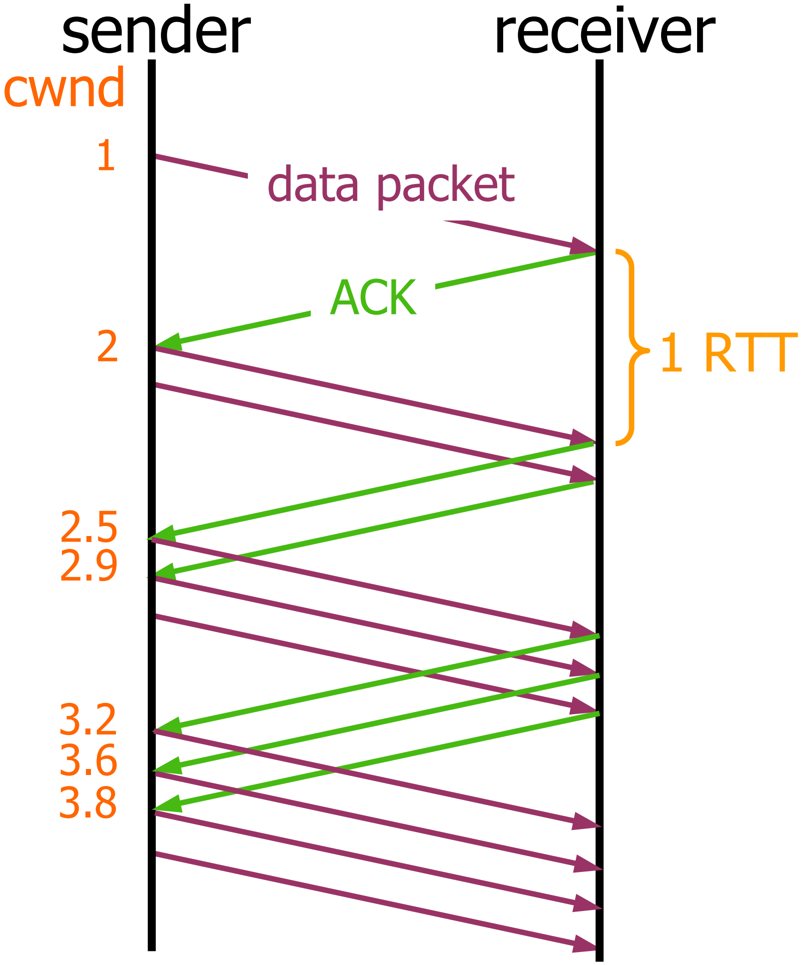
- Starts when $cwnd \geq ssthresh$
- On each successful ACK:

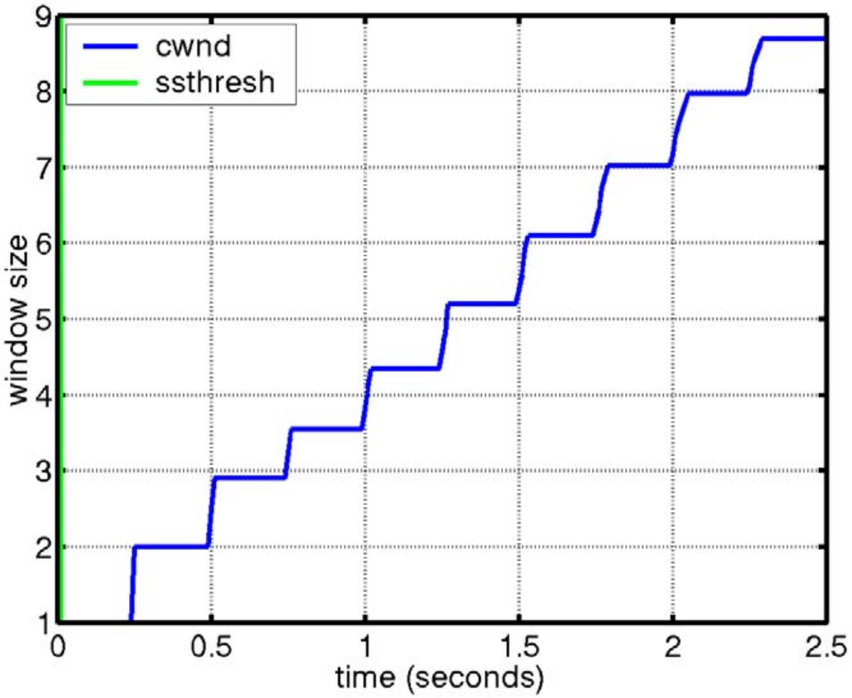
  $cwnd \leftarrow cwnd + 1/cwnd$

- <u>Linear growth</u> of cwnd

  each RTT: $cwnd \leftarrow cwnd + 1$

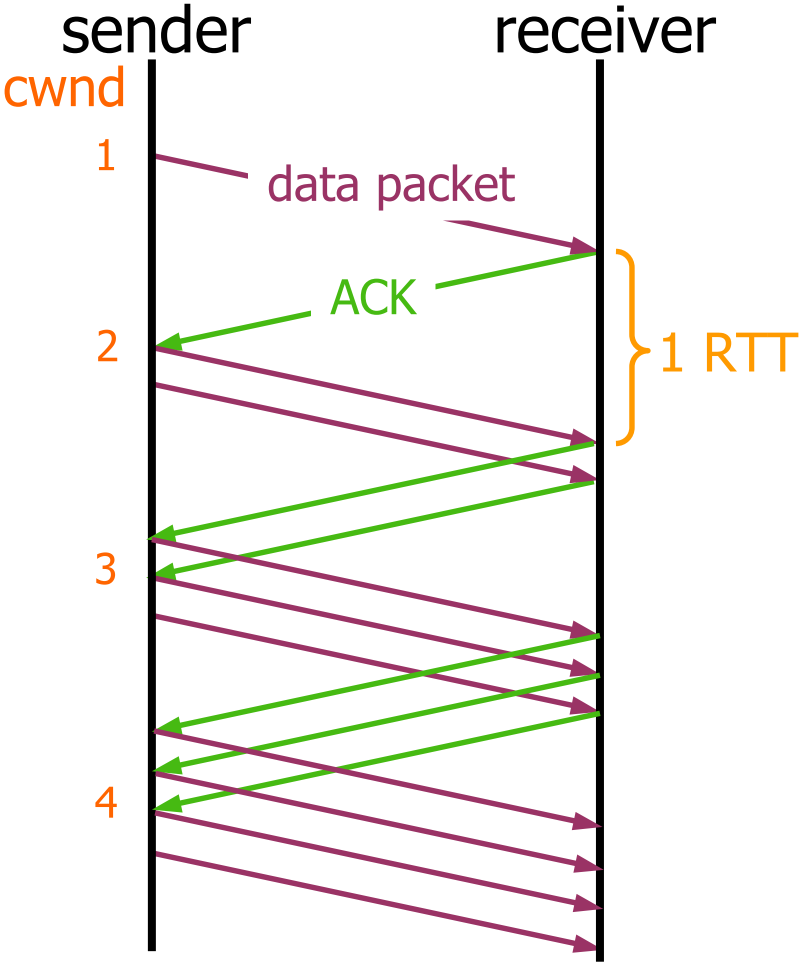
Congestion Avoidance

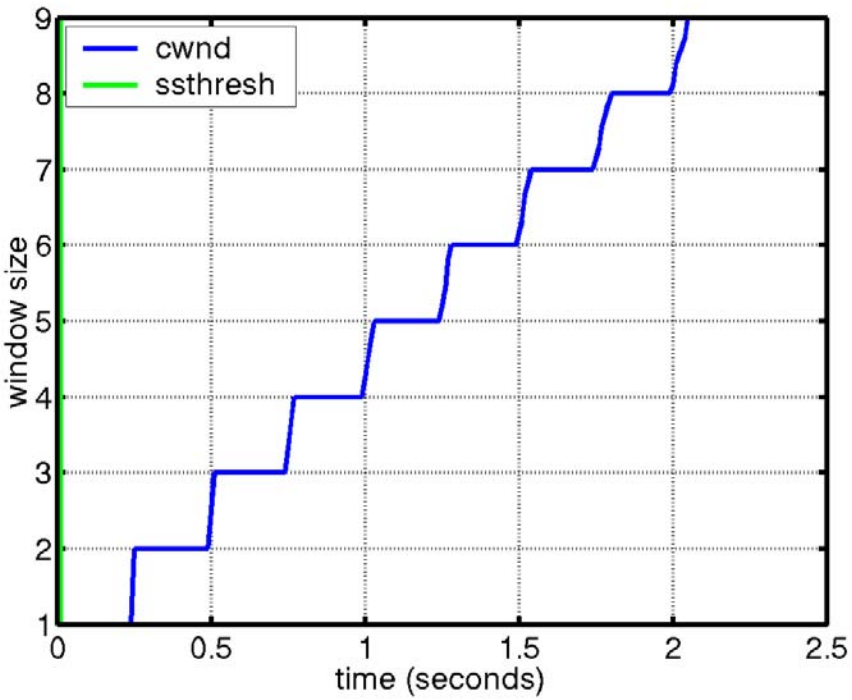




cwnd ← cwnd + 1/cwnd (for each ACK)

Congestion Avoidance

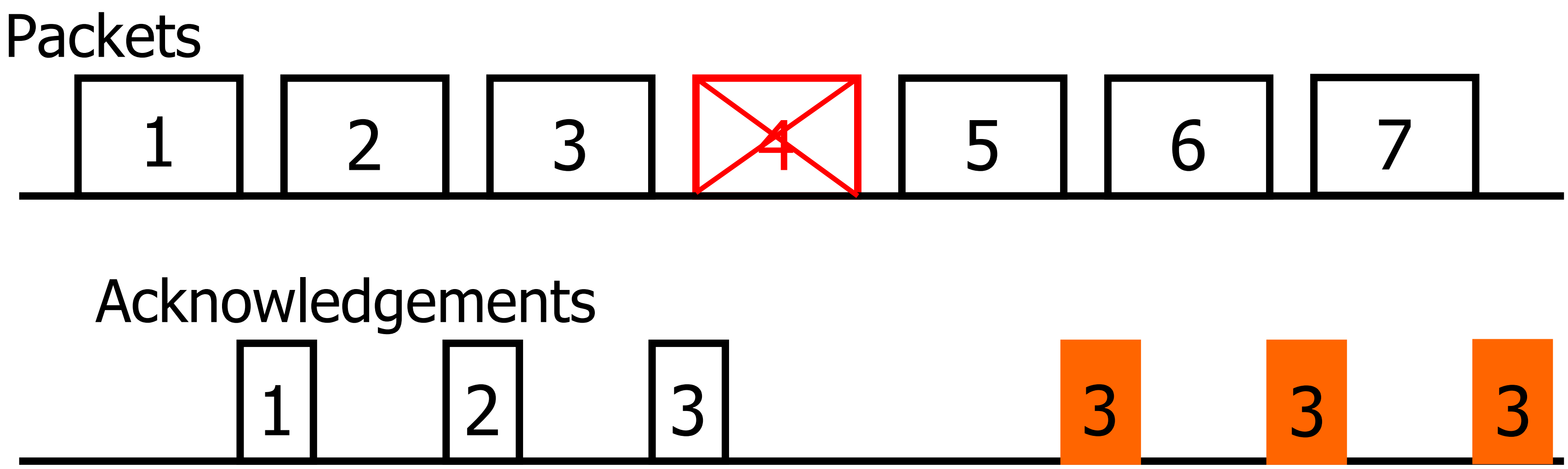




$cwnd \leftarrow cwnd + 1$ (for each cwnd ACKS)

Packet Loss

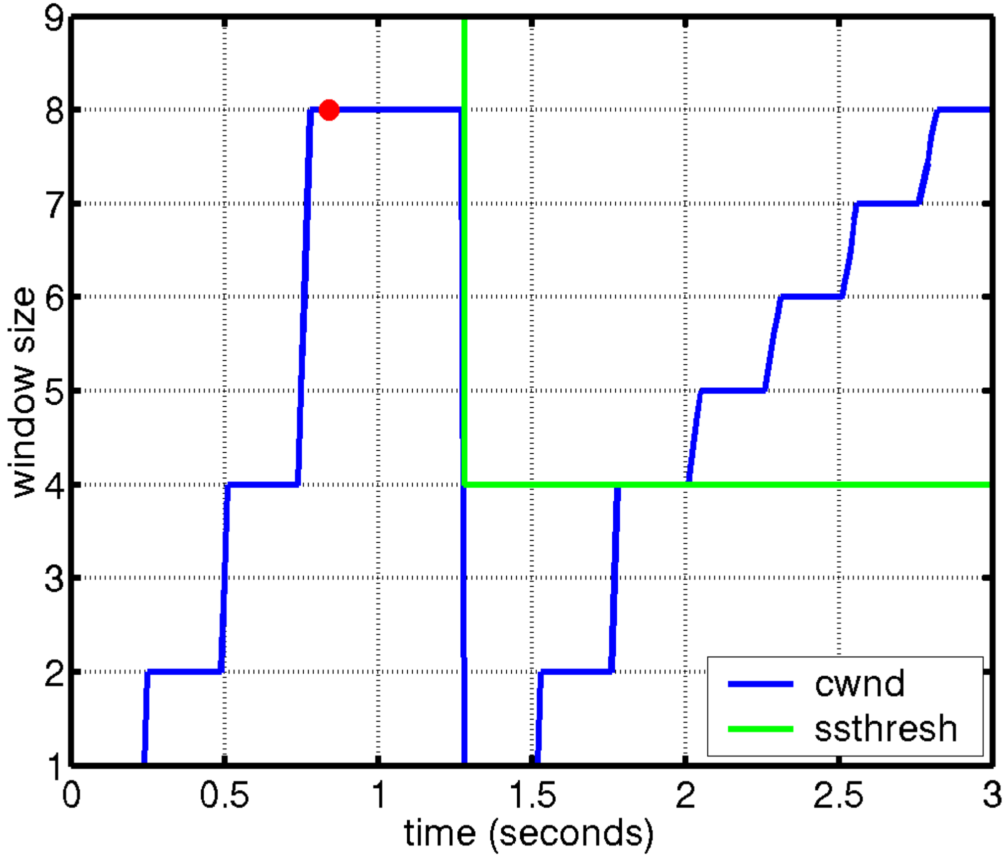
- Assumption: loss indicates congestion
- Packet loss detected by
  - Retransmission TimeOuts (RTO timer)
  - Duplicate ACKs (at least 3)



Timeout

ssthresh ← cwnd/2

cwnd = 1



Fast Retransmit

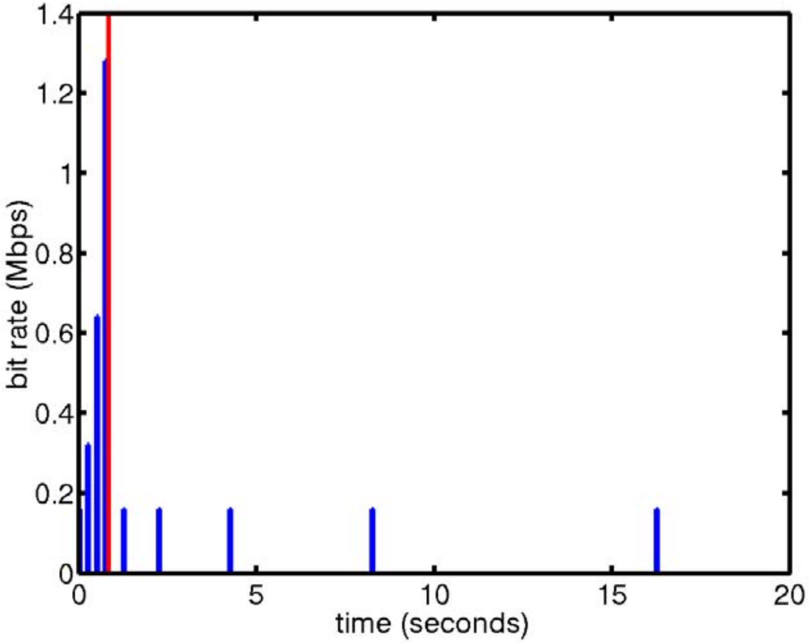
- Wait for a timeout is quite long
- Immediately retransmits after 3 dupACKs without waiting for timeout
- Adjusts ssthresh

  flightsize = min(awnd, cwnd)

  ssthresh $\leftarrow$ max(flightsize/2, 2)
- Enter Slow Start (cwnd = 1)

Successive Timeouts

- When there is a timeout, double the RTO
- Keep doing so for each lost retransmission
  - Exponential back-off
  - Max 64 seconds[1]
  - Max 12 restransmits[1]



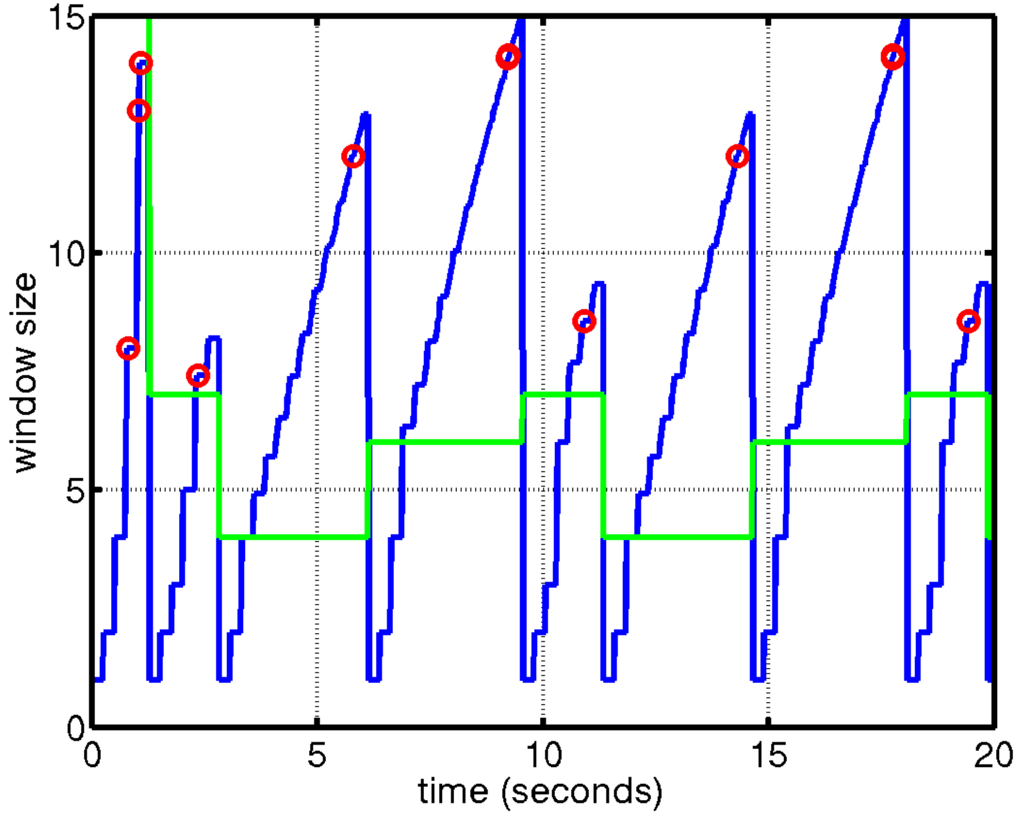
1 - Net/3 BSD

Summary: Tahoe

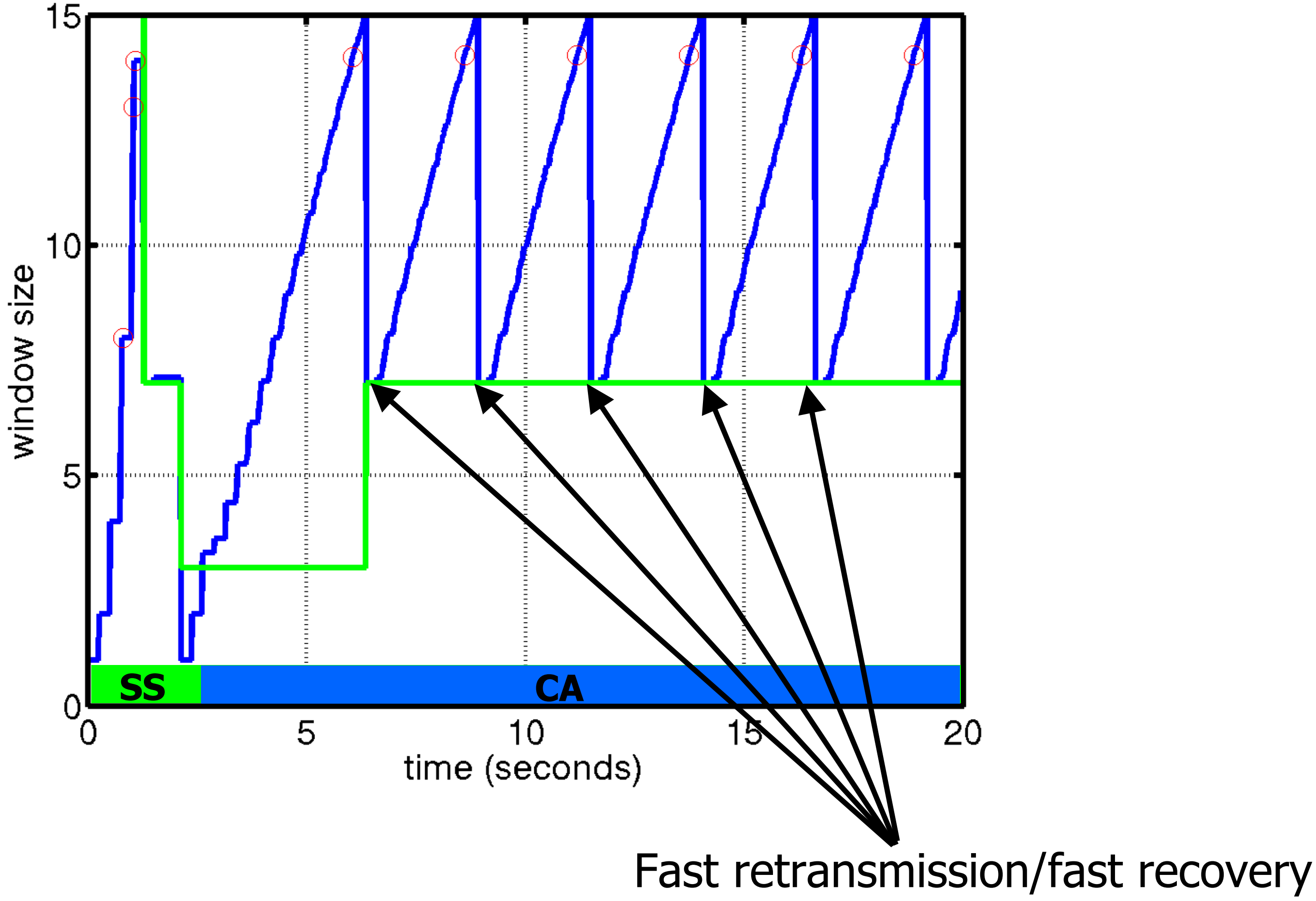
- Basic ideas
  - Gently probe network for spare capacity
  - Drastically reduce rate on congestion
  - Windowing: self-clocking
  - Other functions: round trip time estimation, error recovery

```
for every ACK {
    if (W < ssthresh) then W++         (SS)
    else  W += 1/W                     (CA)
}
for every loss {
    ssthresh = W/2
          W  = 1
}
```

TCP Tahoe



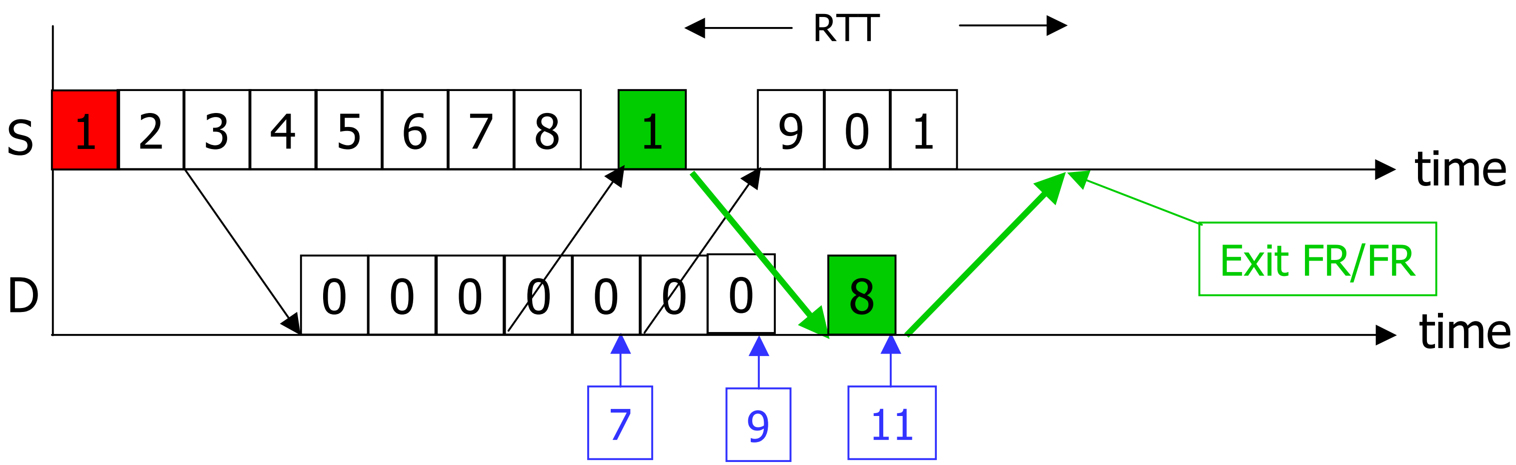
TCP Reno (Jacobson 1990)



Fast recovery

- Motivation: prevent `pipe' from emptying after fast retransmit
- Idea: each dupACK represents a packet having left the pipe (successfully received)
- Enter FR/FR after 3 dupACKs
  - Set ssthresh ← max(flightsize/2, 2)
  - Retransmit lost packet
  - Set cwnd ← ssthresh + ndup (window inflation)
  - Wait till W=min(awnd, cwnd) is large enough; transmit new packet(s)
  - On non-dup ACK (1 RTT later), set cwnd ← ssthresh (window deflation)
- Enter CA

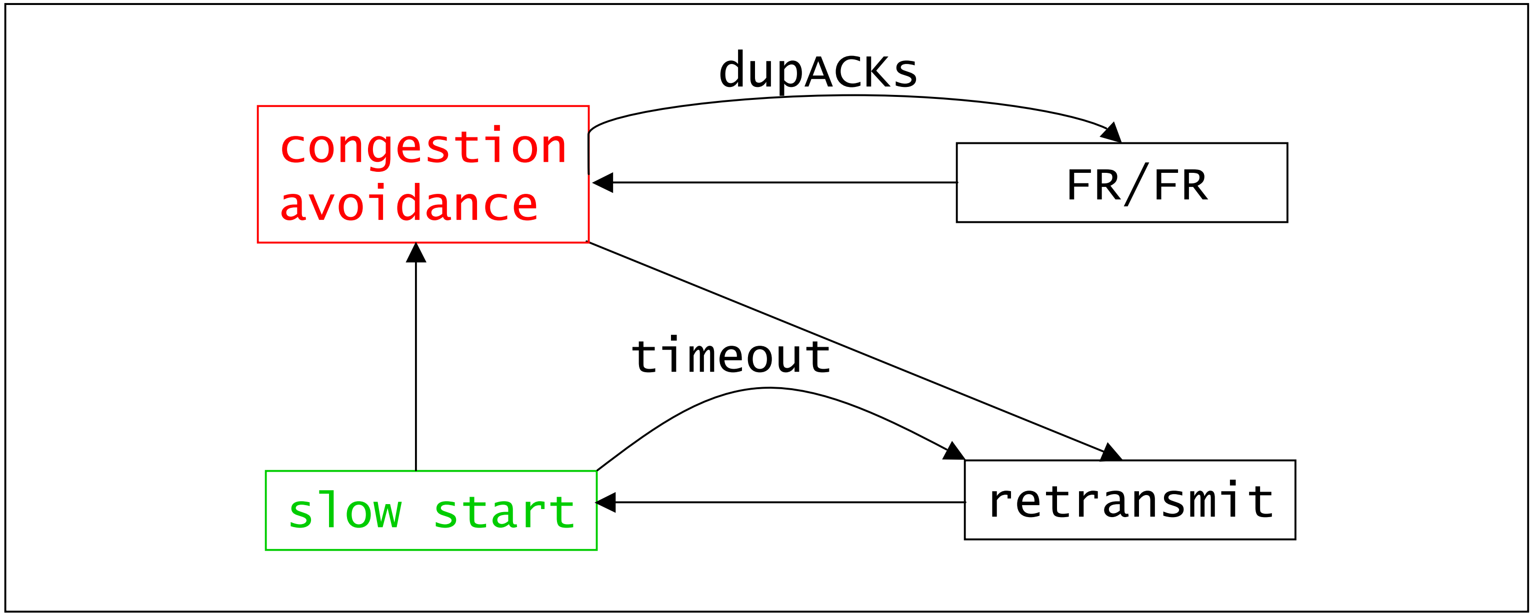
Example: FR/FR



- Fast retransmit
  - Retransmit on 3 dupACKs
- Fast recovery
  - Inflate window while repairing loss to fill pipe

Summary: Reno

- Basic ideas
  - Fast recovery avoids slow start
  - dupACKs: fast retransmit + fast recovery
  - Timeout: fast retransmit + slow start



RTO Calculation

- An accurate RTT measure is required to judge timeouts
- We can measure RTT by measuring the time to receive a packets ACK
- Use a smoothed RTT, S_{RTT} and the smoothed mean deviation D_{RTT}

$$RTO = S_{RTT} + 4\ D_{RTT}$$

- Initial RTT should be > 3 seconds
  - Avoid spurious retransmission

Round Trip Time Estimation

- RTT is not known
  - From <1 ms up to >1 second
- Need to know RTT to calculate RTO
- The measurement of RTT

  $$S_{RTT} = S_{RTT} + g\ (M_{RTT} - S_{RTT})$$
  $$D_{RTT} = D_{RTT} + h\ (\ |M_{RTT} - S_{RTT}| - D_{RTT})$$

- Need to minimize processing requirements
  - Only 1 counter (regardless of how many packets are extant)
  - Counter granularity is typically 500 ms
- Measurement equations have gain

Timers on a Packet Loss

- Ignore RTT for retransmitted packets (Karn)
- If a timeout occurs, double the RTO and retransmit the lost packet
  - results in exponential back-off
  - recalculate S_{RTT} only when a packet gets through
- RTT is lost if several packets are lost

Delayed Acknowledgements

- ACKs may be delayed to ‘piggy-back’ on returning data packets
  - by no more than 500ms, typically 200ms
  - Out of order segments are ACK’d immediately
  - Segments which fill a gap are ACK’d immediately
- While waiting
  - More data packets may arrive
  - A delayed ACK may ack. up to 2 MSS packets
- SS and CA increment cwnd per ACK
  - *Not* per ACK’d packet
  - Window size increases more slowly

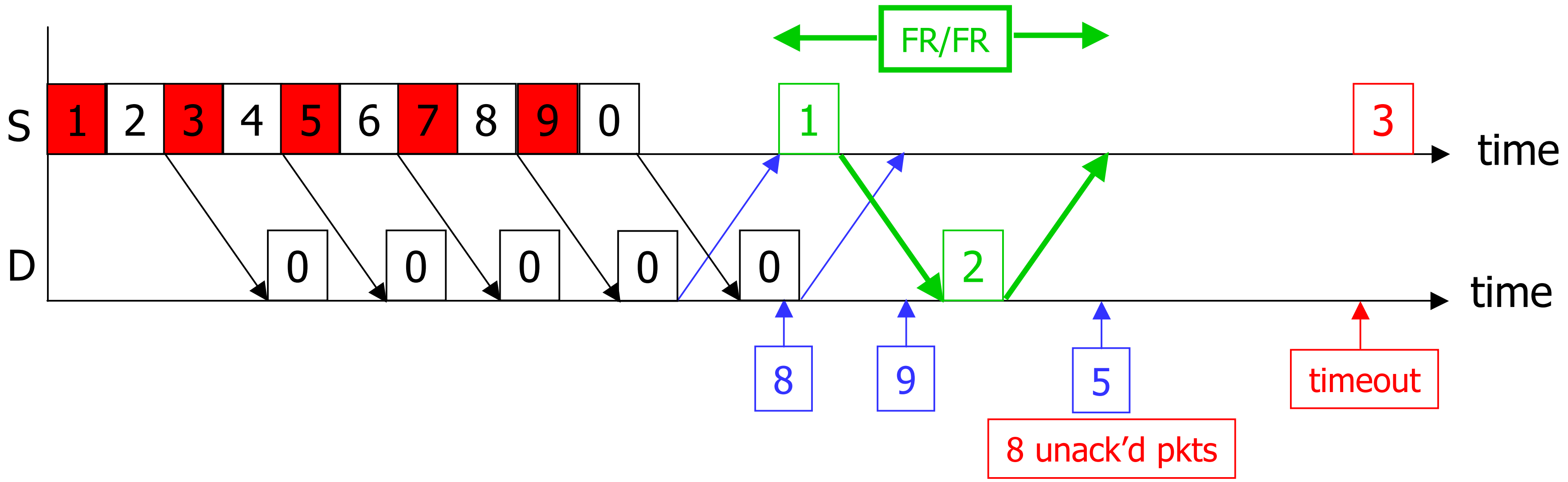
TCP Options for High-Perf.

- In high performance networks the counters may wrap
  - max sequence number is $2^{32} - 1 \cong 4$ GB
  - The maximum awnd is $2^{16} - 1 = 65{,}535$ B
  - Protection Against Wrapped Sequence Numbers (PAWS) (RFC 1323)
  - Window scaling (RFC 1323)
- Timestamps (RFC 1323)
- Larger initial window (RFC 2414, 2415, 2416)

Implementation Dependence

- ssthresh initialisation (not standardised)
  - Reno $ssthresh_{init} = \infty$
  - Solaris $ssthresh_{init} = 8$
  - Linux $ssthresh_{init} = 1$
- algorithm for incrementing cwnd in CA
- 1990 Reno had CA window increase
  - $\Delta W = MSS^2/cwnd + MSS/8$
- Sharing between TCP sessions (RFC 2140)
  - Over time (temporal) – caching window values
  - Between sessions (ensemble) – better RTT estimation
- Many possible bugs! (RFC 2525)

NewReno: Motivation



- On 3 dupACKs, receiver has packets 2, 4, 6, 8, cwnd=8, retransmits pkt 1, enter FR/FR
- Next dupACK increment cwnd to 9
- After a RTT, ACK arrives for pkts 1 & 2, exit FR/FR, cwnd=5, 8 unack'ed pkts
- No more ACK, sender must wait for timeout

NewReno Fall & Floyd '96, (RFC 2583)

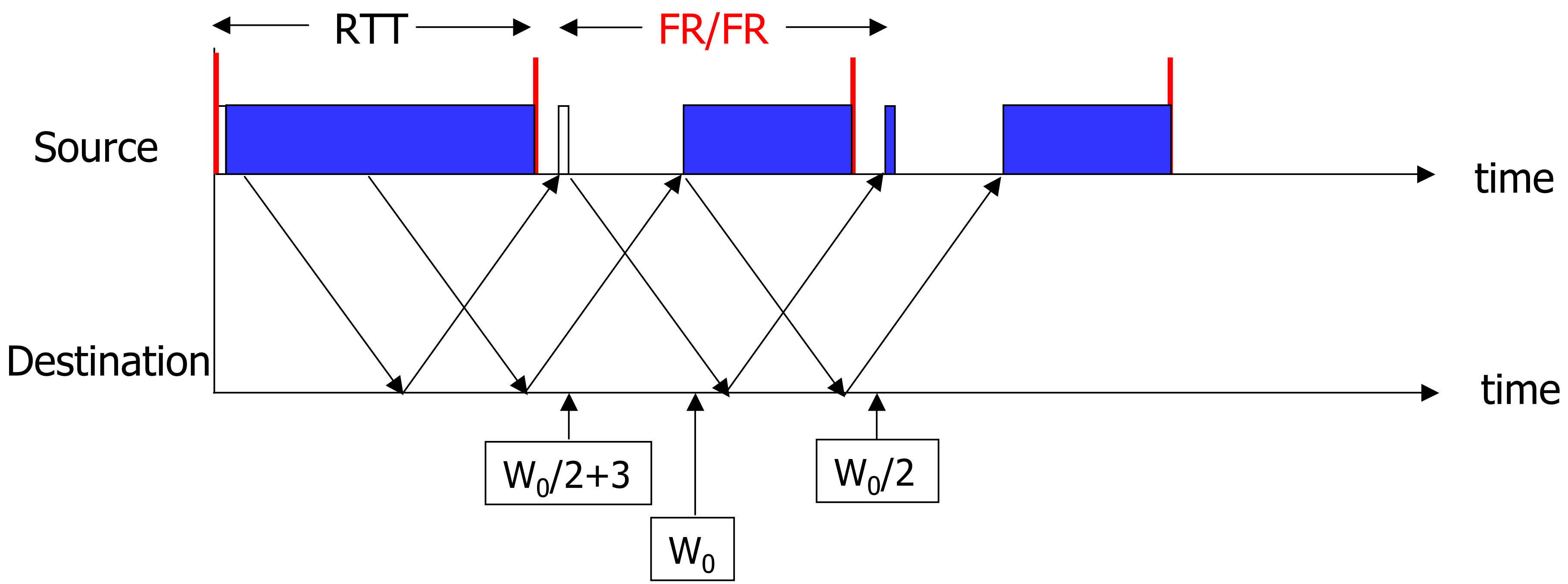
- Motivation: multiple losses within a window
  - Partial ACK acknowledges some but not all packets outstanding at start of FR
  - Partial ACK takes Reno out of FR, deflates window
  - Sender may have to wait for timeout before proceeding
- Idea: partial ACK indicates lost packets
  - Stays in FR/FR and retransmits immediately
  - Retransmits 1 lost packet per RTT until all lost packets from that window are retransmitted
  - Eliminates timeout

SACK Mathis, Mahdavi, Floyd, Romanow ’96 (RFC 2018, RFC 2883)

- Motivation: Reno & NewReno retransmit at most 1 lost packet per RTT
  - Pipe can be emptied during FR/FR with multiple losses
- Idea: SACK provides better estimate of packets in pipe
  - SACK TCP option describes received packets
  - On 3 dupACKs: retransmits, halves window, enters FR
  - Updates pipe = packets in pipe
    - Increment when lost or new packets sent
    - Decrement when dupACK received
  - Transmits a (lost or new) packet when pipe < cwnd
  - Exit FR when all packets outstanding when FR was entered are acknowledged

Variant: Rate-halving

- Motivation: in and after FR, cwnd packets sent in second half of RTT

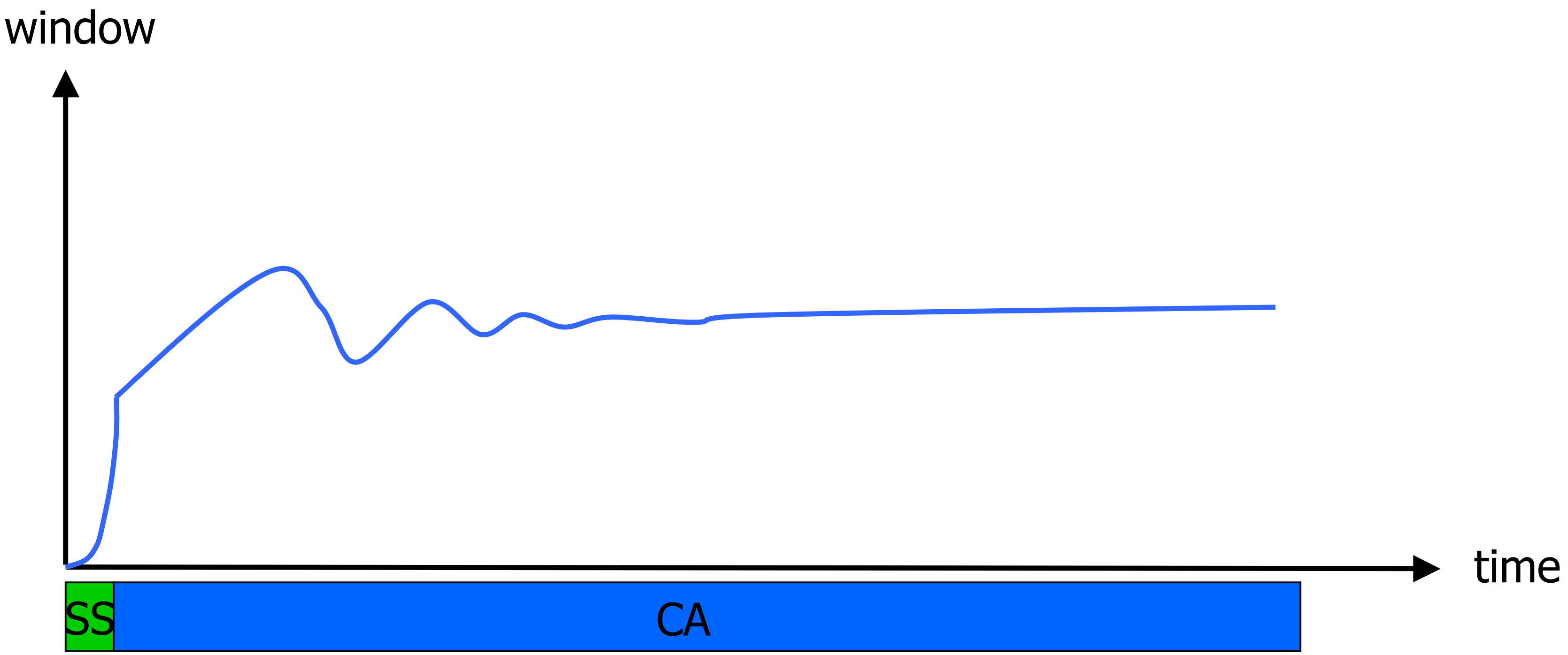


- Idea: send 1 packet every 2 ACKs for 1 RTT
  - Smooth burst
  - Reduce chance of timeout

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model
  - Feedback control model

TCP Vegas (Brakmo & Peterson 1994)



- Reno with a new congestion avoidance algorithm
- Converges (provided buffer is large) !

Congestion avoidance

- Each source estimates number of its own packets in pipe from RTT
- Adjusts window to maintain estimate between αd and βd

$$
\begin{array}{l}
\texttt{for every RTT} \\
\{ \\
\quad \texttt{if } W/RTT_{min} - W/RTT < \alpha\, RTT_{min} \texttt{ then W ++} \\
\quad \texttt{if } W/RTT_{min} - W/RTT > \beta\, RTT_{min} \texttt{ then W --} \\
\} \\
\texttt{for every loss} \\
\qquad \texttt{W := W/2}
\end{array}
$$

Implications

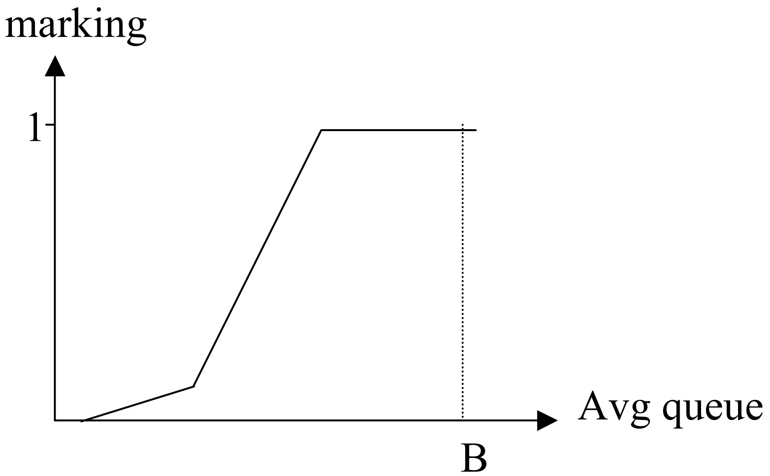
- Congestion measure = end-to-end queueing delay
- At equilibrium
  - Zero loss
  - Stable window at full utilization
  - Approximately weighted proportional fairness
  - Nonzero queue, larger for more sources
- Convergence to equilibrium
  - Converges if sufficient network buffer
  - Oscillates like Reno otherwise

Outline

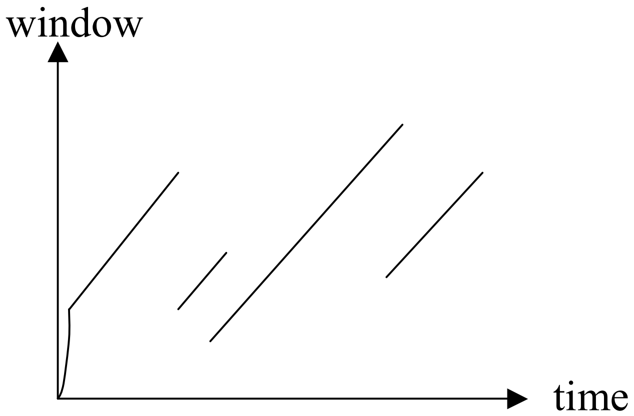
- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model
  - Feedback control model

RED (Floyd & Jacobson 1993)

- Idea: warn sources of incipient congestion by probabilistically marking/dropping packets
- Link algorithm to work with source algorithm (Reno)
- Bonus: desynchronization
  - Prevent bursty loss with buffer overflows



router



host

RED

- Implementation
  - Probabilistically drop packets
  - Probabilistically mark packets
  - Marking requires ECN bit (RFC 2481)
- Performance
  - Desynchronization works well
  - Extremely sensitive to parameter setting
  - Fail to prevent buffer overflow as #sources increases

Variant: ARED (Feng, Kandlur, Saha, Shin 1999)

- Motivation: RED extremely sensitive to #sources
- Idea: adapt $\mathtt{max}_p$ to load
  - If avg. queue < $\mathtt{min}_{th}$, decrease $\mathtt{max}_p$
  - If avg. queue > $\mathtt{max}_{th}$, increase $\mathtt{max}_p$
- No per-flow information needed

Variant: FRED (Ling & Morris 1997)

- Motivation: marking packets in proportion to flow rate is unfair (e.g., adaptive vs unadaptive flows)
- Idea
  - A flow can buffer up to $\mathtt{min}_q$ packets without being marked
  - A flow that frequently buffers more than max_q packets gets penalized
  - All flows with backlogs in between are marked according to RED
  - No flow can buffer more than `avgcq` packets persistently
- Need per-active-flow accounting

Variant: SRED (Ott, Lakshman & Wong 1999)

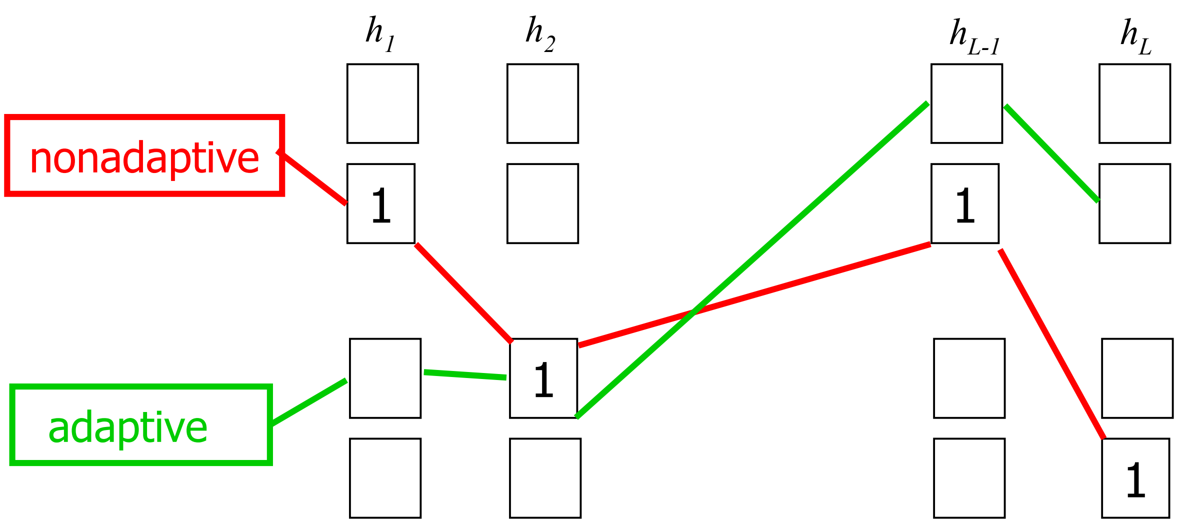
- Motivation: wild oscillation of queue in RED when load changes
- Idea:
  - Estimate number N of active flows
    - An arrival packet is compared with a randomly chosen active flows
    - $N \sim \text{prob(Hit)}^{-1}$
  - cwnd$\sim p^{-1/2}$ and $Np^{-1/2} = Q_0$ implies $p = (N/Q_0)^2$
  - Marking prob = $m(q)$ min($1, p$)
- No per-flow information needed

Variant: BLUE (Feng, Kandlur, Saha, Shin 1999)

- Motivation: wild oscillation of RED leads to cyclic overflow & underutilization
- Algorithm
  - On buffer overflow, increment marking prob
  - On link idle, decrement marking prob

Variant: SFB

- Motivation: protection against nonadaptive flows
- Algorithm
  - L hash functions map a packet to L bins (out of NxL)
  - Marking probability associated with each bin is
    - Incremented if bin occupancy exceeds threshold
    - Decremented if bin occupancy is 0
  - Packets marked with min $\{p_1, \ldots, p_L\}$



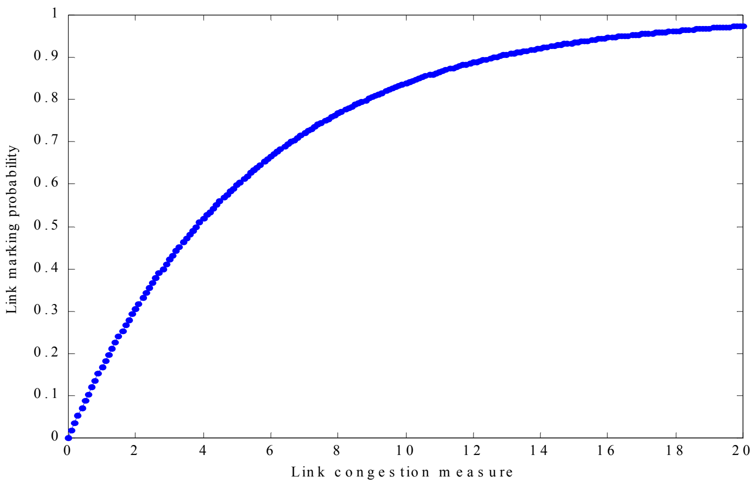
Variant: SFB

- Idea
  - A nonadaptive flow drives marking prob to 1 at all L bins it is mapped to
  - An adaptive flow may share some of its L bins with nonadaptive flows
  - Nonadaptive flows can be identified and penalized

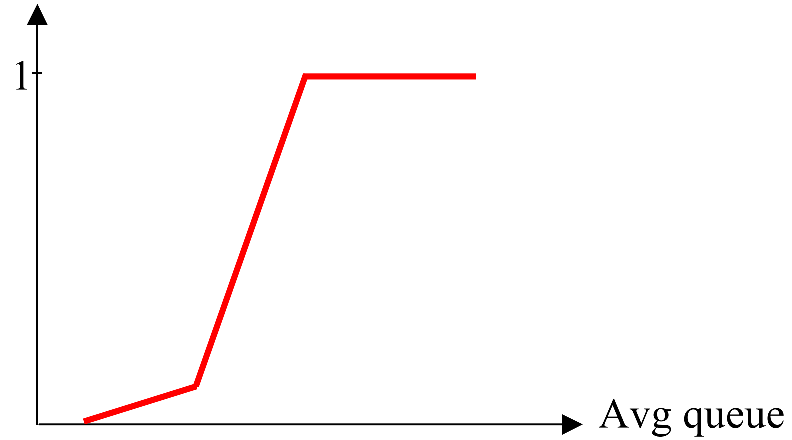
REM Athuraliya & Low 2000

- Main ideas
  - Marking probability exponential in `price'
  - Price adjusted to match rate and clear buffer
  - `Congestion' = `demand>supply'
  - … but performance remains good!

REM



RED



Part II
Models

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
    - $1/\sqrt{p}$ law
    - Fixed-point models
    - Finite source models
  - Duality model
  - Feedback control model

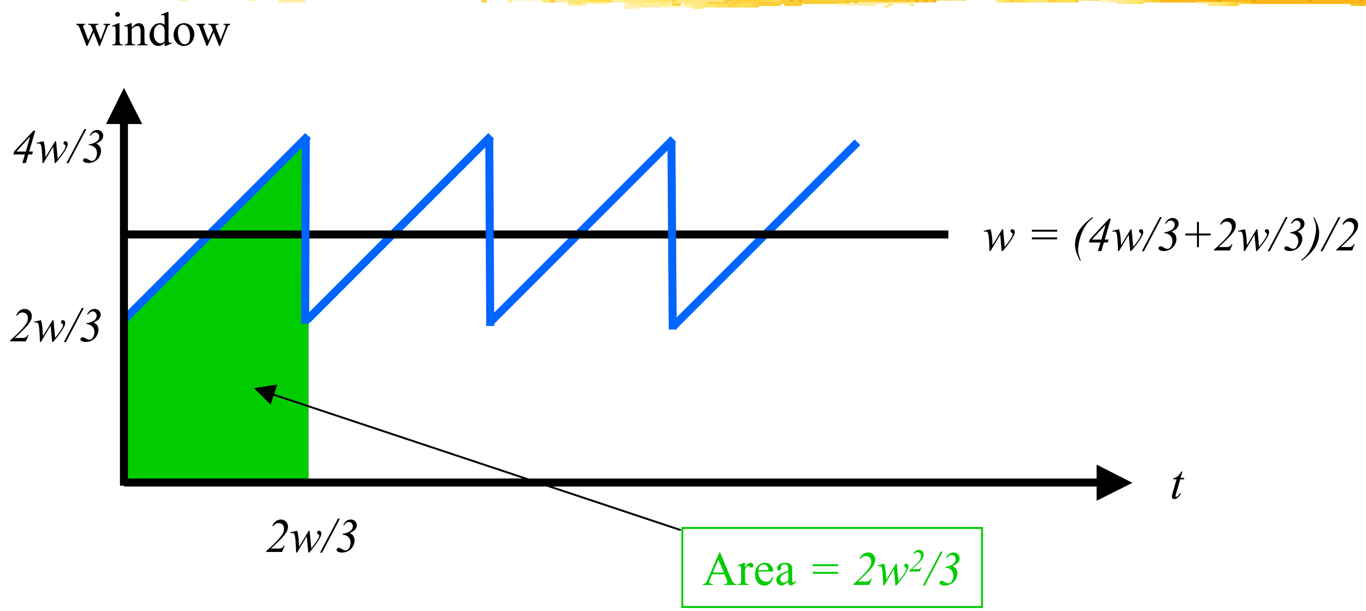
$1/\sqrt{p}$ Law

- Equilibrium window size $w_s = \dfrac{a}{\sqrt{p}}$
- Equilibrium rate $x_s = \dfrac{a}{D_s\sqrt{p}}$
- Empirically constant $a \sim 1$
- Verified extensively through simulations and on Internet
- References
    - T.J.Ott, J.H.B. Kemperman and M.Mathis (1996)
    - M.Mathis, J.Semke, J.Mahdavi, T.Ott (1997)
    - T.V.Lakshman and U.Mahdow (1997)
    - J.Padhye, V.Firoin, D.Towsley, J.Kurose (1998)
    - J.Padhye, V.Firoin, D.Towsley (1999)

Implications

- Applicability
  - Additive increase multiplicative decrease (Reno)
  - Congestion avoidance dominates
  - No timeouts, e.g., SACK+RH
  - Small losses
  - Persistent, greedy sources
  - Receiver not bottleneck
- Implications
  - Reno equalizes window
  - Reno discriminates against long connections

Derivation (I)

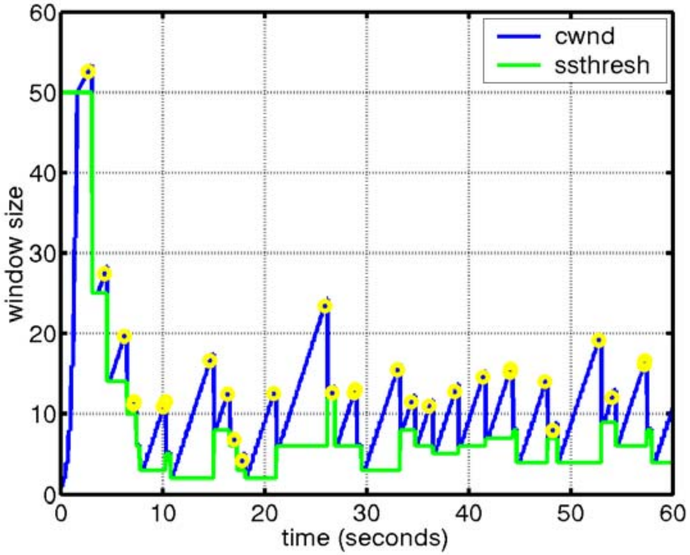


- Each cycle delivers $2w^2/3$ packets
- Assume: each cycle delivers $1/p$ packets
  - Delivers 1/p packets followed by a drop
  - Loss probability = $p/(1+p) \sim p$ if p is small.
- Hence $w = \sqrt{3/2p}$

Derivation (II)

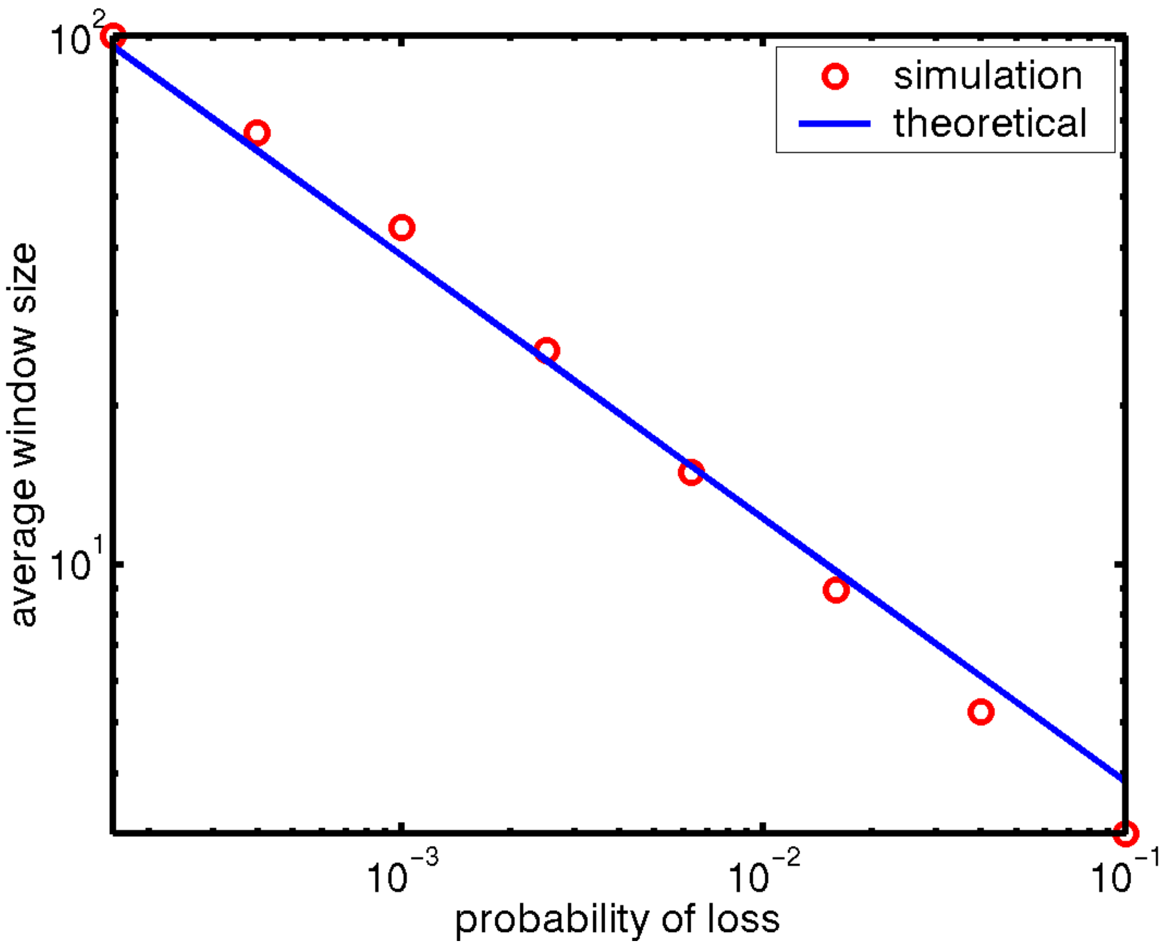
- Assume: loss occurs as Bernoulli process rate p
- Assume: spend most time in CA
- Assume: p is small
- w_n is the window size after n

$$w_{n+1} = \begin{cases} w_n/2, & \text{if a packet is lost (prob. } pw_n) \\ w_n + 1, & \text{if no packet is lost (prob. } (1-pw_n)) \end{cases}$$



$$\begin{aligned} \overline{w} &= \frac{\overline{w}}{2} p\overline{w} + (\overline{w}+1)(1-p\overline{w}) \\ \overline{w}^2 &\approx 2/p \\ \overline{w} &\approx \sqrt{2/p} \end{aligned}$$

Simulations



Refinement (Padhye, Firoin, Towsley & Kurose 1998)

- Renewal model including
  - FR/FR with Delayed ACKs (b packets per ACK)
  - Timeouts
  - Receiver awnd limitation
- Source rate

$$x_s = \min\left(\frac{W_r}{D_s}, \frac{1}{D_s\sqrt{\frac{2bp}{3}} + T_o \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right)$$

- When p is small and W_r is large, reduces to

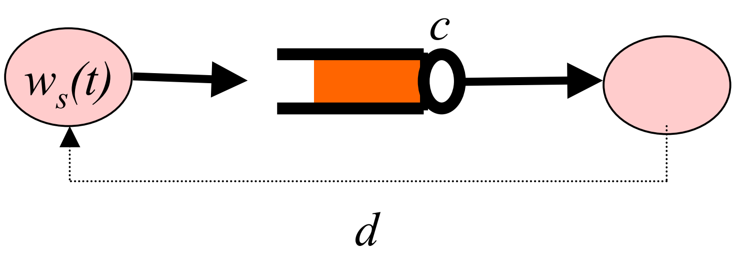
$$x_s = \frac{a}{D_s\sqrt{p}}$$

Further Refinements

- Further refinements of previous formula
  - Padhye, Firoin, Towsley and Kurose (1999)
- Other loss models
  - E.Altman, K.Avrachenkov and C.Barakat (Sigcomm 2000)
  - Square root p still appears!
- Dynamic models of TCP
  - E.G. RTT evolves as window increases

Dynamic model (Bonald 1998)

- Single source model
  - Single link
  - Ignore slow start
  - Instantaneous loss detection
  - RTT $D(t) = \max\{d,\ w(t)/c\}$

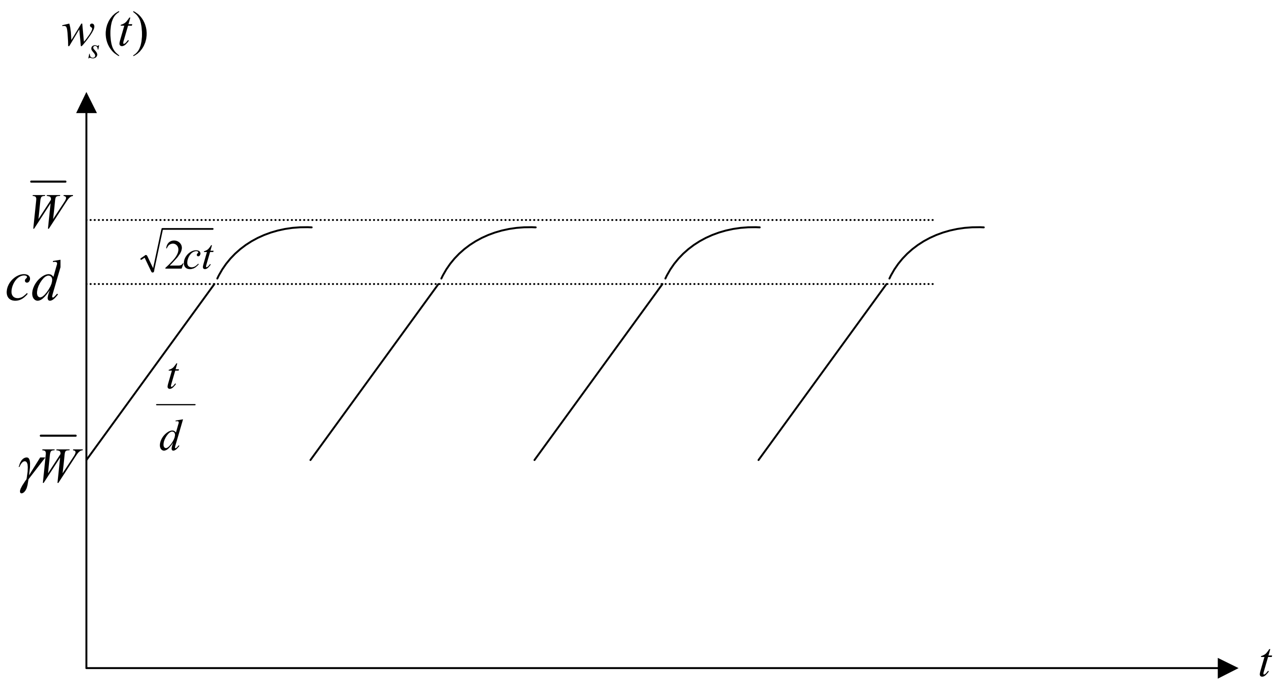


- Key: window process (CA) increases at rate $1/D(t)$

$$
\begin{aligned}
\dot{w}(t) &= \frac{1}{d}, && \text{if } w(t) \le cd \\
\dot{w}(t) &= \frac{c}{w(t)}, && \text{if } cd < w(t) < \overline{W} \\
w(t^{+}) &= \gamma w(t), && \text{if } w(t) = \overline{W}
\end{aligned}
$$

Dynamic Model

- Periodic solution (single source)



Application (TCP Over Wireless)

- TCP uses loss as a congestion indication
- In wireless, packet loss may occur due to
  - Fading
  - Interference
  - Handover
- $1/\sqrt{p}$ law provides a quick method for estimating the effect of wireless losses
- Some method is required to avoid performance degradation (see RFC 2757 and the references therein)

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
    - $1/\sqrt{p}$ law
    - Fixed-point models
    - Finite source models
  - Duality model
  - Feedback control model

Calculating Performance

- Single link, capacity C, buffer B
  - Window size: $w = f(p)$
  - Loss rate: $p = g(w; C,B)$
  - Find w*: $w^* = f(g(w^*; C,B))$
- Example:
  - Window size: $w = 1/\sqrt{p}$
  - Loss rate from bufferless approx. $p = \frac{[w-C]^+}{w}$

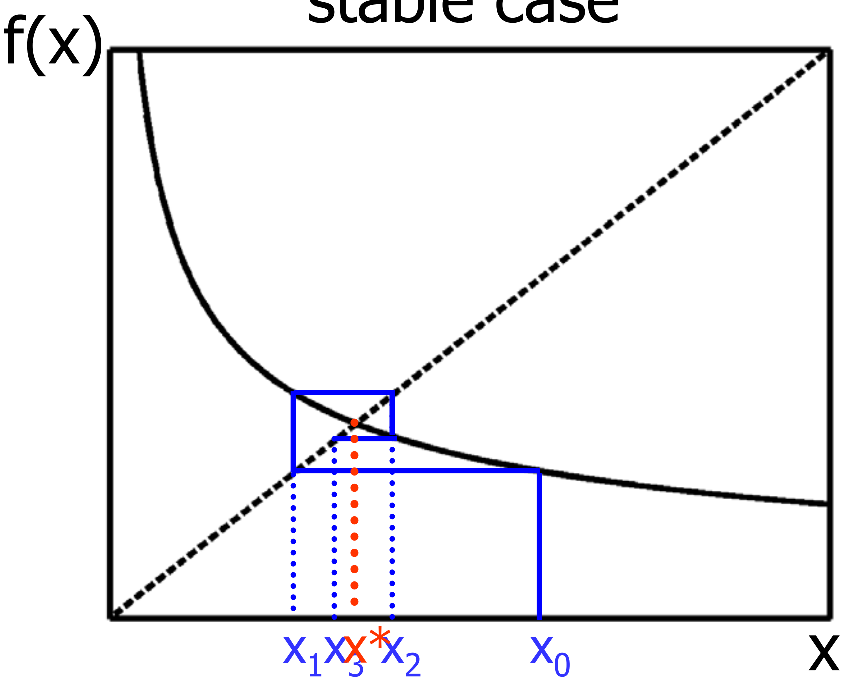
$$w^* = \frac{C + \sqrt{C^2 + 4}}{2}$$

Fixed Point Models

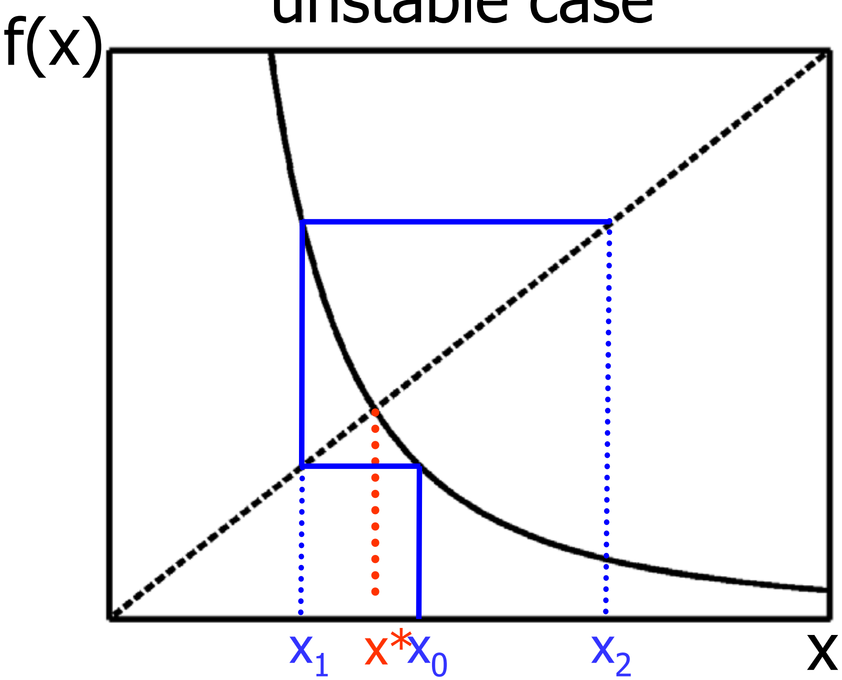
- Mean field theory
  - Solve for a particular source given the mean field
  - Use single source to approximate the mean field
- Generalize previous example
  - Multiple sources
  - Network
    - various routes, RTTs, capacities, …
  - Arbitrary functions f, and g
- Solve using
  - Repeated substitution
  - Newton-Raphson

Repeated Substitution

stable case



unstable case



$x_{n+1} = f(x_n)$

$x^* = f(x^*)$

$|f'(x)|<1$

$x^* = f(x^*)$

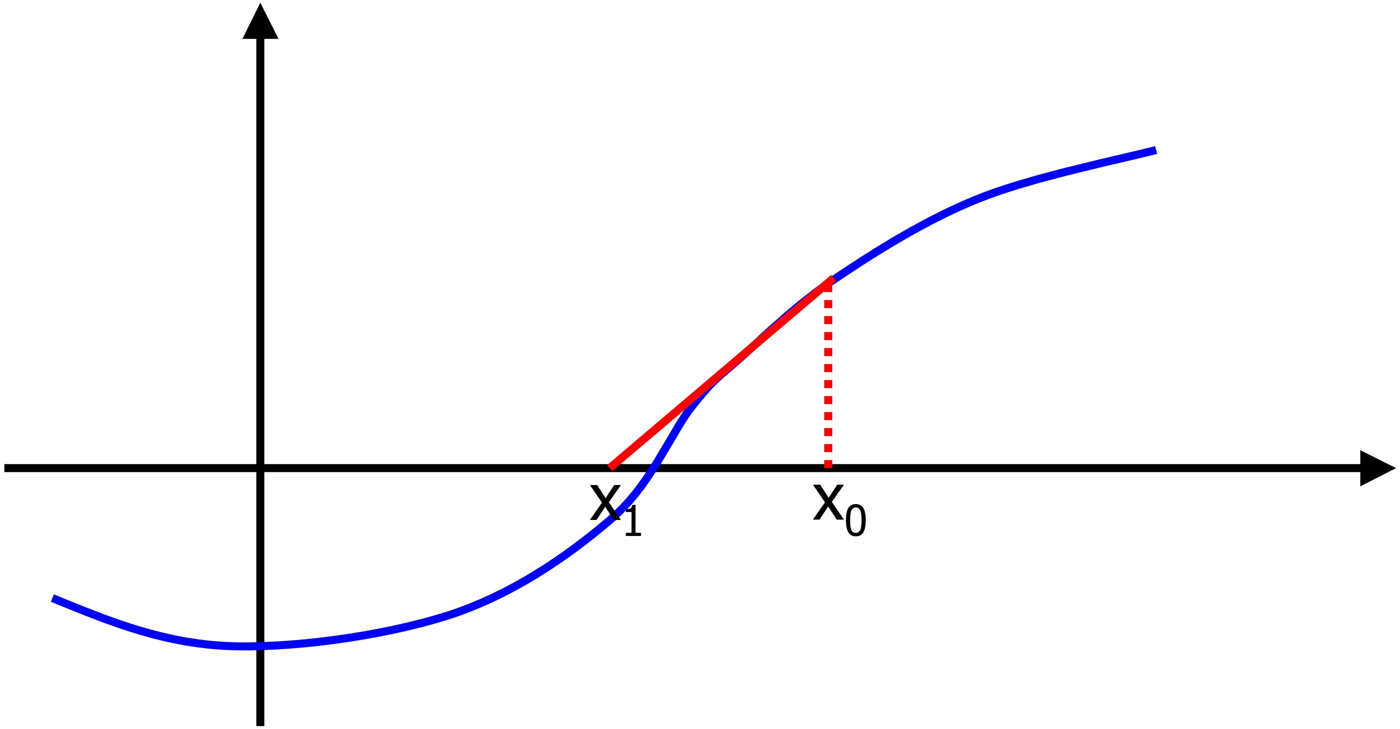
$|f'(x)|>1$

Newton-Raphson

x=f(x) becomes F(x)=f(x)-x=0

use the slope F′ to form a tangent

$$x_{n+1} = x_n - F(x_n)/F'(x_n)$$



Network Formulation

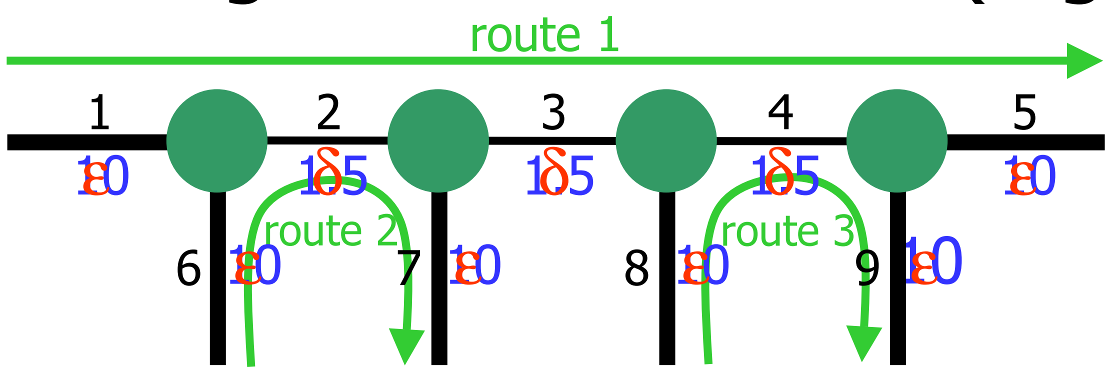
- N links, R routes
- Capacity $\mathbf{c} = \{c_j\}$ j=1,..,N
- Propagation time $\mathbf{t} = \{t_j\}$ j=1,..,N
- Routing matrix $\mathbf{A} = \{a_{ij}\}$ j=1,..,N, i=1,..,R

  $a_{ij} = 1$, if link j is in route i

  $a_{ij} = 0$, if link j isn't in route j
- Sources per route $\mathbf{n} = \{n_i\}$ i=1,..,R
- MSS per route $\mathbf{m} = \{m_j\}$ i=1,..,R
- Route send rate $\mathbf{s} = \{s_j\}$ i=1,..,R
- Link loss rate $\mathbf{q} = \{d_i\}$ j=1,..,N
- Route loss rate $\mathbf{p} = \{p_i\}$ i=1,..,R

Example Network

- N congested bottlenecks (e.g. 2)



$$\mathbf{t} = (\varepsilon,\ \delta,\ \delta,\ \delta,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon)^t$$

$$\mathbf{c} = (10,\ 1.5,\ 1.5,\ 1.5,\ 10,\ 10,\ 10,\ 10,\ 10)^t$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Solution

- Estimate RTT delay from propagation time

  $$d = 2At$$ (can use queueing delays)

- Route send rates

  $$x(w) = (w .* n .* m) ./ d$$

- Link rates

  $$b(w) = A^t x$$

- Link loss rate

  $$q(w;c) = [b - c]^+ ./ b$$ (can use queueing losses)

- Route loss rate

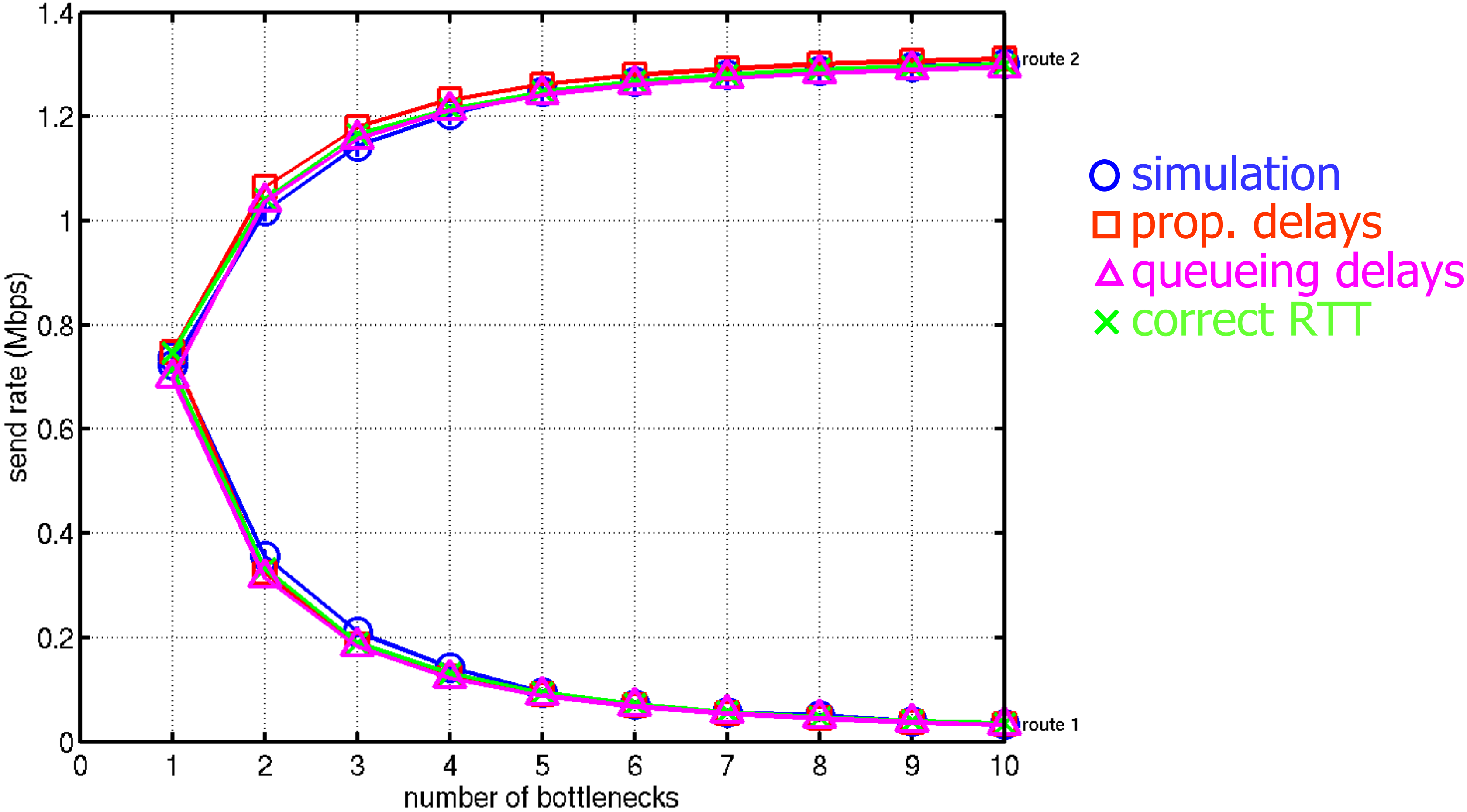
  $$p(w;c) = 1 - e^{A\ln(1-q(w;c))}$$

- Window size

  $$W^2\, p(w;c) - a = 0$$ (could use refined model, or a transient model)

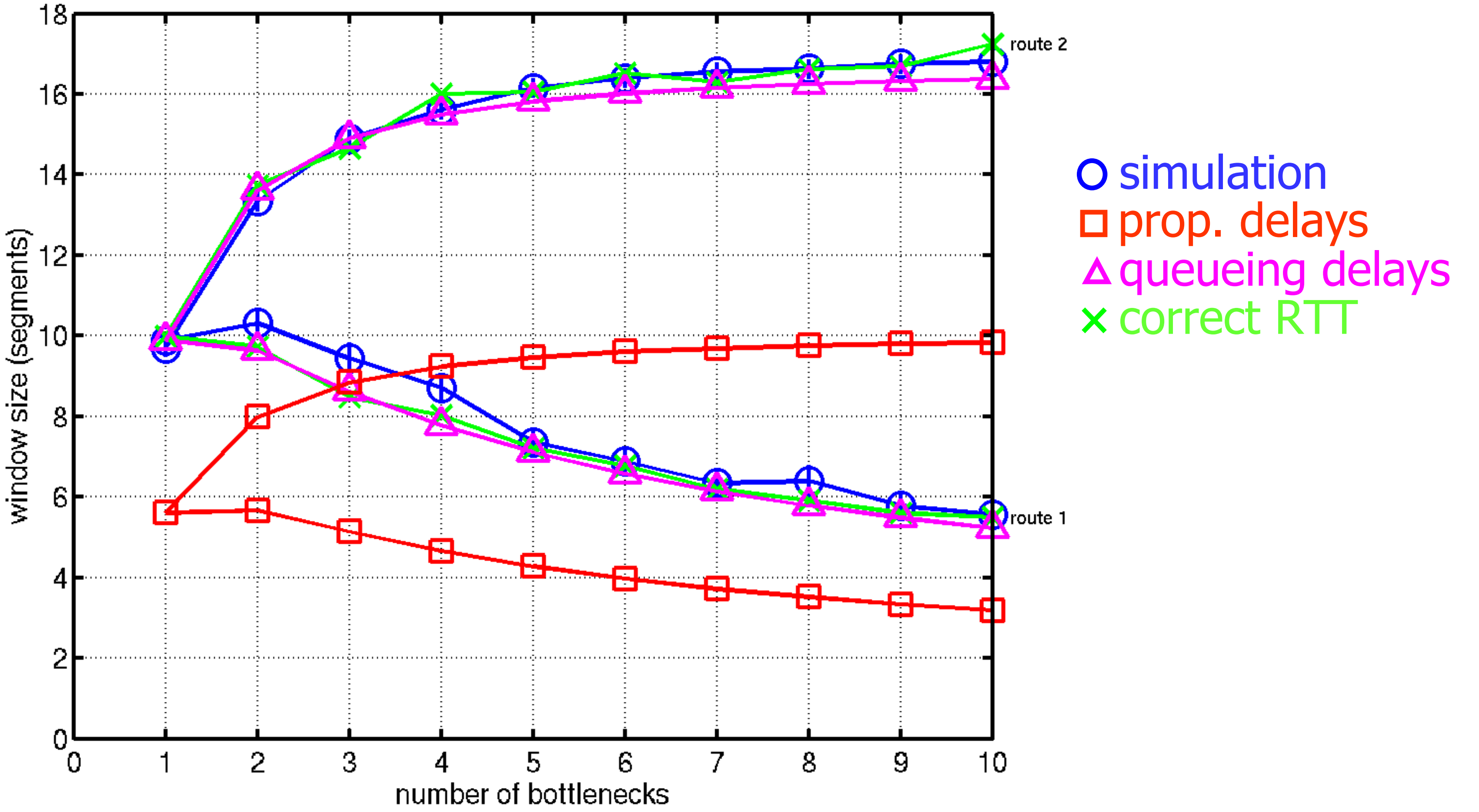
Numerical Example

- Send rates



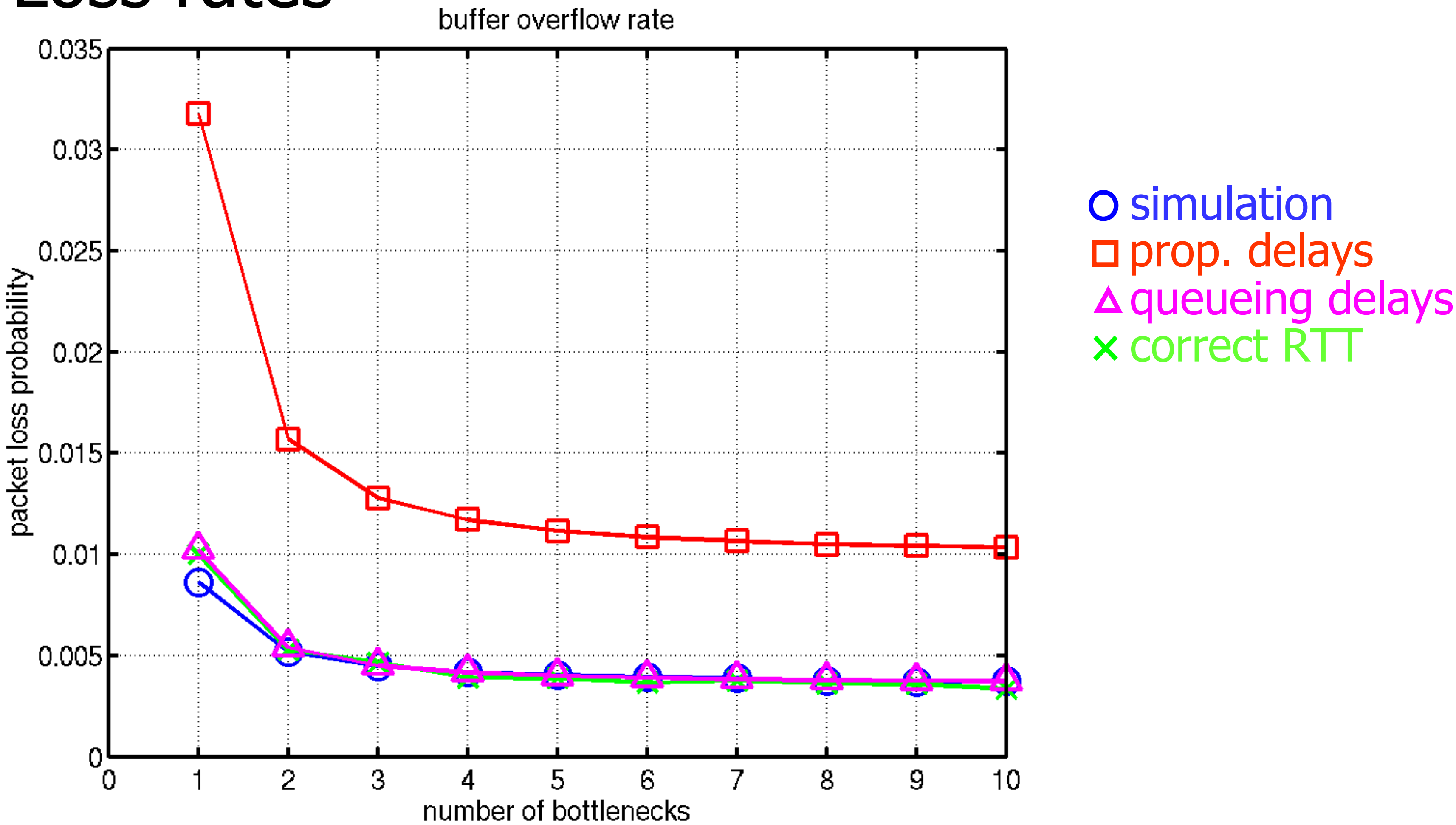
Numerical Example

- Window sizes



Numerical Example

- Loss rates



Unfairness in TCP

- Rates along either route are skewed
- TCP Tahoe/Reno are inherently unfair
  - biased against long RTT
  - biased against multiply congested paths (S.Floyd, 1991)
- TCP Vegas is proportionally fair (see later)

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
    - $1/\sqrt{p}$ law
    - Fixed-point models
    - Finite source models
  - Duality model
  - Feedback control model

Importance of Finite Sources

- Infinite sources are unrealistic
  - WWW traffic has median response size 3-4kB
- Infinite sources lead to possibly spurious conclusions
  - Synchronization
  - LRD (Veres and Boda, Infocom 2000)
- Performance of finite sources is **profoundly** different
  - SS rather than CA

Finite Source Models

- Processor sharing models
  - D.P. Heyman and T.V. Lakshman and A. Neidhardt, “A New Method for Analysing Feedback-Based Protocols with Applications to Engineering Web Traffic over the Internet”, SIGMETRICS 1997.
  - others
- E.G. M/G/1 with processor sharing
  - Assumes: TCP sessions arrive as Poisson process
  - Assumes: some file transfer size distribution G (heavy-tail)
  - Assumes: TCP sources share bandwidth evenly
- These types of model are not quite good enough
  - Don’t take dynamics of TCP into account

Outline

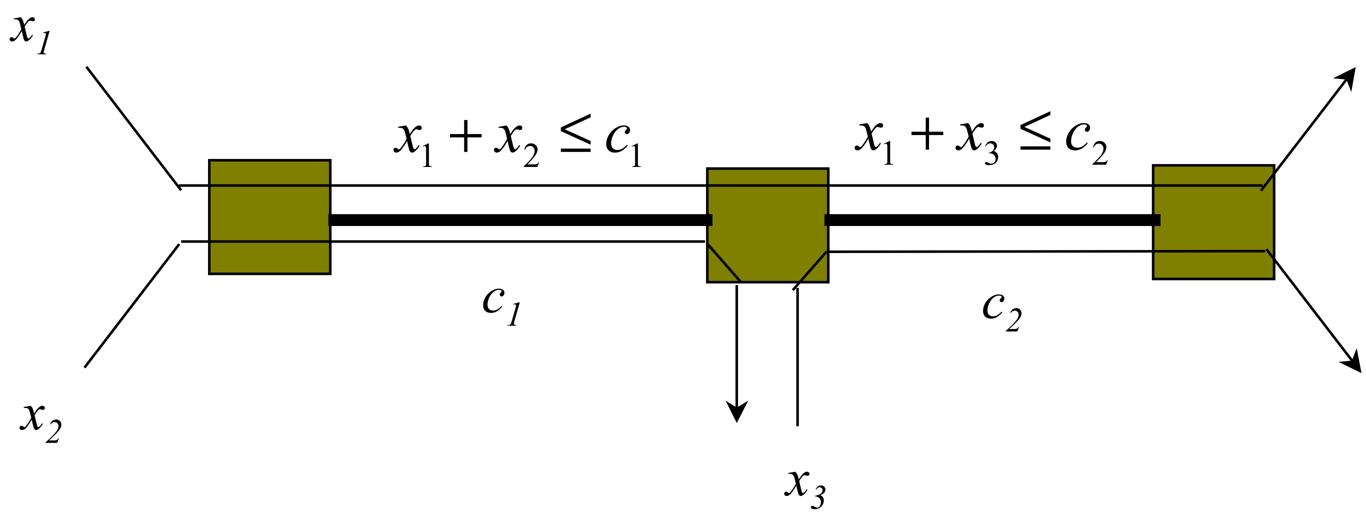
- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model *(F, G, U)*
    - Queue management G : RED, REM
    - TCP G and U : Reno, Vegas
    - Performance of REM
  - Feedback control model

Flow control

- Interaction of source rates $x_s(t)$ and congestion measures $p_l(t)$
- Duality theory
  - They are primal and dual variables
  - Flow control is optimization process
- Example congestion measure
  - Loss (Reno)
  - Queueing delay (Vegas)
  - Queue length (RED)
  - Price (REM)

Model

- Sources s
  - $L(s)$ - links used by source s
  - $U_s(x_s)$ - utility if source rate = x_s
- Network
  - Links l of capacities c_l



Primal problem

$$\max_{x_s \geq 0} \quad \sum_s U_s(x_s)$$

$$\text{subject to} \quad x^l \leq c_l, \quad \forall l \in L$$

- Assumptions
  - Strictly concave increasing U_s
- Unique optimal rates x_s exist
- Direct solution impractical

Prior Work

- Formulation
  - Kelly 1997
- Penalty function approach
  - Kelly, Maulloo and Tan 1998
  - Kunniyur and Srikant 2000
- Duality approach
  - Low and Lapsley 1999
  - Athuraliya and Low 2000, Low 2000
- Extensions
  - Mo & Walrand 1998
  - La & Anantharam 2000

Prior Work

- Formulation
  - Kelly 1997
- Penalty function approach
  - Kelly, Maulloo and Tan 1998
  - Kunniyur and Srikant 2000
- Duality approach
  - Low and Lapsley 1999
  - Athuraliya and Low 2000, Low 2000
- Extensions
  - Mo & Walrand 1998
  - La & Anantharam 2000

Duality Approach

$$\text{Primal:} \quad \max_{x_s \geq 0} \sum_s U_s(x_s) \quad \text{subject to } x^l \leq c_l, \quad \forall l \in L$$

$$\text{Dual:} \quad \min_{p \geq 0} \quad D(p) = \left(\max_{x_s \geq 0} \sum_s U_s(x_s) + \sum_l p_l(c_l - x^l) \right)$$

Primal-dual algorithm:

$$x(t+1) = F(p(t), x(t))$$

$$p(t+1) = G(p(t), x(t))$$

Duality Model of TCP

- Source algorithm iterates on rates
- Link algorithm iterates on prices
- With different utility functions

Primal-dual algorithm:

$$x(t+1) = F(p(t), x(t)) \longleftarrow \boxed{\text{Reno, Vegas}}$$

$$p(t+1) = G(p(t), x(t)) \longleftarrow \boxed{\text{DropTail, RED, REM}}$$

Example

- Basic algorithm

$$\text{source :} \quad x_s(t+1) = U_s'^{-1}(p^s(t))$$

$$\text{link :} \quad p_l(t+1) = [p_l(t) + \gamma(x^l(t) - c_l)]^+$$

Theorem (ToN'99)

Converge to optimal rates in asynchronous environment

TCP schemes are smoothed versions of source algorithm …

Summary

- Flow control problem

$$\max_{x_s \ge 0} \quad \sum_s U_s(x_s)$$

$$\text{subject to} \quad x^l \le c_l, \quad \forall l \in L$$

- Primal-dual algorithm

$$x(t+1) = F(p(t),\ x(t))$$

$$p(t+1) = G(p(t),\ x(t))$$

- Major TCP schemes
  - Maximize aggregate source utility
  - With different utility functions

Summary

- What are the (F, G, U) ?
- Derivation
  - Derive (F, G) from protocol description
  - Fix point $(x, p) = (F, G)$ gives equilibrium
  - Derive U
    - regard fixed point as Kuhn-Tucker condition

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model (F, G, U)
    - Queue management G : RED, REM
    - TCP G and U : Reno, Vegas
    - Performance of REM
  - Feedback control model

Active queue management

- Idea: provide congestion information by probabilistically marking packets
- Issues
  - How to measure congestion (p and G)?
  - How to embed congestion measure?
  - How to feed back congestion info?

$$x(t+1) = F(\,p(t), x(t)\,) \longleftarrow \boxed{\text{Reno, Vegas}}$$

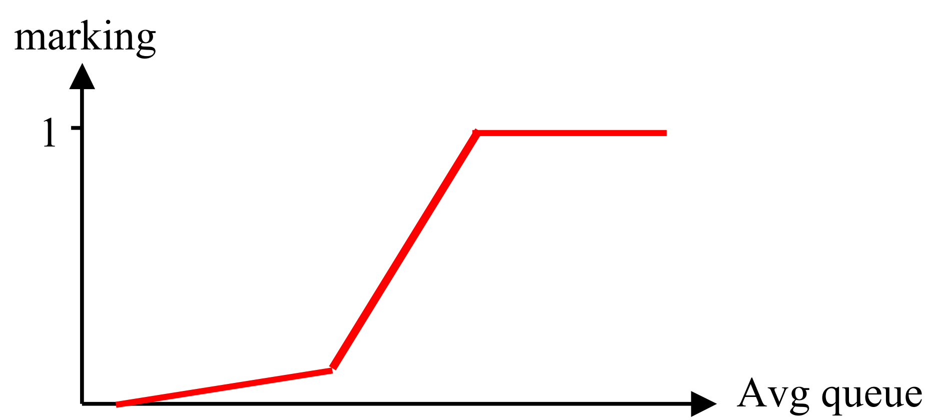
$$p(t+1) = G(\,p(t), x(t)\,) \longleftarrow \boxed{\text{DropTail, RED, REM}}$$

RED (Floyd & Jacobson 1993)

- Congestion measure: average queue length

$$p_l(t+1) = [p_l(t) + x^l(t) - c_l]^+$$

- Embedding: p-linear probability function



- Feedback: dropping or ECN marking

REM (Athuraliya & Low 2000)

- Congestion measure: price

$$p_l(t+1) = [p_l(t) + \gamma(\alpha_l\, b_l(t) + x^l\,(t) - c_l)]^+$$

- Embedding:

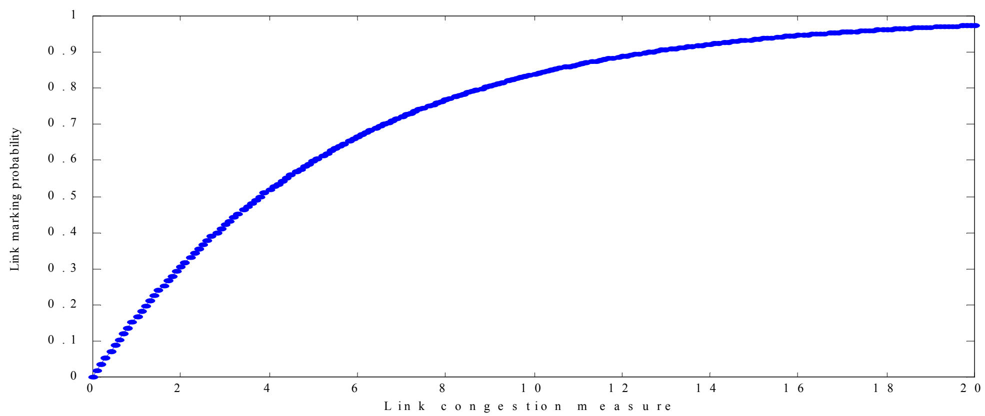
- Feedback: dropping or ECN marking

REM (Athuraliya & Low 2000)

- Congestion measure: price

$$p_l(t+1) = [p_l(t) + \gamma(\alpha_l\, b_l(t) + x^l(t) - c_l)]^+$$

- Embedding: exponential probability function



- Feedback: dropping or ECN marking

Key features

- Clear buffer and match rate

$$p_l(t+1) = [p_l(t) + \gamma(\underbrace{\alpha_l b_l(t)}_{\text{Clear buffer}} + \underbrace{\hat{x}^l(t) - c_l}_{\text{Match rate}})]^+$$

- Sum prices

$$1 - \phi^{-p_l(t)} \quad \Rightarrow \quad 1 - \phi^{-p^s(t)}$$

Theorem (Paganini 2000)

Global asymptotic stability for general utility function (in the absence of delay)

Active Queue Management

	$p_l(t)$	$G(p(t), x(t))$
DropTail	loss	$[1 - c_l/x^l(t)]^+$ (?)
RED	queue	$[p_l(t) + x^l(t) - c_l]^+$
Vegas	delay	$[p_l(t) + x^l(t)/c_l - 1]^+$
REM	price	$[p_l(t) + \gamma(\alpha_l\, b_l(t) + x^l(t) - c_l)]^+$

$x(t+1) = F(\,p(t), x(t)\,)$ ← Reno, Vegas

$p(t+1) = G(\,p(t), x(t)\,)$ ← DropTail, RED, REM

Congestion & performance

	$p_l(t)$	$G(p(t), x(t))$
Reno	loss	$[1 - c_l/x^l(t)]^+$ (?)
Reno/RED	queue	$[p_l(t) + x^l(t) - c_l]^+$
Reno/REM	price	$[p_l(t) + \gamma(\alpha_l b_l(t) + x^l(t) - c_l)]^+$
Vegas	delay	$[p_l(t) + x^l(t)/c_l - 1]^+$

- Decouple congestion & performance measure
  - RED: \`congestion' = \`bad performance'
  - REM: \`congestion' = \`demand exceeds supply'
    But performance remains good!

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model (F, G, U)
    - Queue management G : RED, REM
    - TCP G and U : Reno, Vegas
    - Performance of REM
  - Feedback control model

Utility functions

- Reno

$$U_s^{reno}(x_s) = \frac{\sqrt{2}}{D_s}\tan^{-1}\left(\frac{x_s D_s}{2}\right)$$

- Reno/RED

$$U_s^{reno/red}(x_s) = \begin{cases} b_1 x_s + \rho_1 \frac{\sqrt{2}}{D_s}\tan^{-1}\left(\frac{x_s D_s}{2}\right), & x_s \text{ large} \\ b_2 x_s + \rho_2 \frac{\sqrt{2}}{D_s}\tan^{-1}\left(\frac{x_s D_s}{2}\right), & x_s \text{ small} \end{cases}$$

- Reno/REM

$$U_s^{reno/rem}(x_s) = (\log \phi)^{-1}\left(x \log\left(1 + \frac{2}{x_s^2 D_s^2}\right) + \frac{2\sqrt{2}}{D_s}\tan^{-1}\left(\frac{x_s D_s}{2}\right)\right)$$

- Vegas, Vegas/REM

$$U_s^{vegas}(x_s) = \alpha_s d_s \log x_s$$

Reno: F

```
for every ack (ca)
{   W += 1/W   }
for every loss
{   W := W/2   }
```

$$\Delta w_s(t) \;=$$

Primal-dual algorithm:

$$x(t+1) = F(\,p(t), x(t)\,)$$ ← Reno, Vegas

$$p(t+1) = G(\,p(t), x(t)\,)$$ ← DropTail, RED, REM

Reno: F

```
for every ack (ca)
{   w += 1/w    }
for every loss
{   w := w/2    }
```

$$\Delta w_s(t) = \frac{x_s(t)(1 - p(t))}{w_s}$$

Primal-dual algorithm:

$x(t+1) = F(\, p(t), x(t)\,)$ ← Reno, Vegas

$p(t+1) = G(\, p(t), x(t)\,)$ ← DropTail, RED, REM

Reno: F

```
for every ack (ca)
{   w += 1/w   }
for every loss
{   w := w/2   }
```

$$\Delta w_s(t) = \frac{x_s(t)(1-p(t))}{w_s} - \frac{w_s(t)}{2} x_s(t) p(t)$$

$$F_s\big(p(t), x(t)\big) = x_s(t) + \frac{(1-p(t))}{D_s^2} - \frac{x_s^2(t)}{2} p(t)$$

Primal-dual algorithm:

$x(t+1) = F(\, p(t), x(t)\,)$ ← Reno, Vegas

$p(t+1) = G(\, p(t), x(t)\,)$ ← DropTail, RED, REM

Reno: Utility Function

$$F_s\big(p(t), x(t)\big) = x_s(t) + \frac{(1 - p(t))}{D_s^2} - \frac{x_s^2(t)}{2} p(t)$$

$$x_s = F_s(p, x)$$

$$\Rightarrow \quad \frac{(1 - p)}{D_s^2} = \frac{x_s^2}{2} p$$

$$\Rightarrow \quad \frac{2}{2 + x_s^2 D_s^2} = p \qquad \Rightarrow \quad U_s^{reno}(x_s) = \frac{\sqrt{2}}{D_s} \tan^{-1}\left(\frac{x_s D_s}{2}\right)$$

Reno: summary

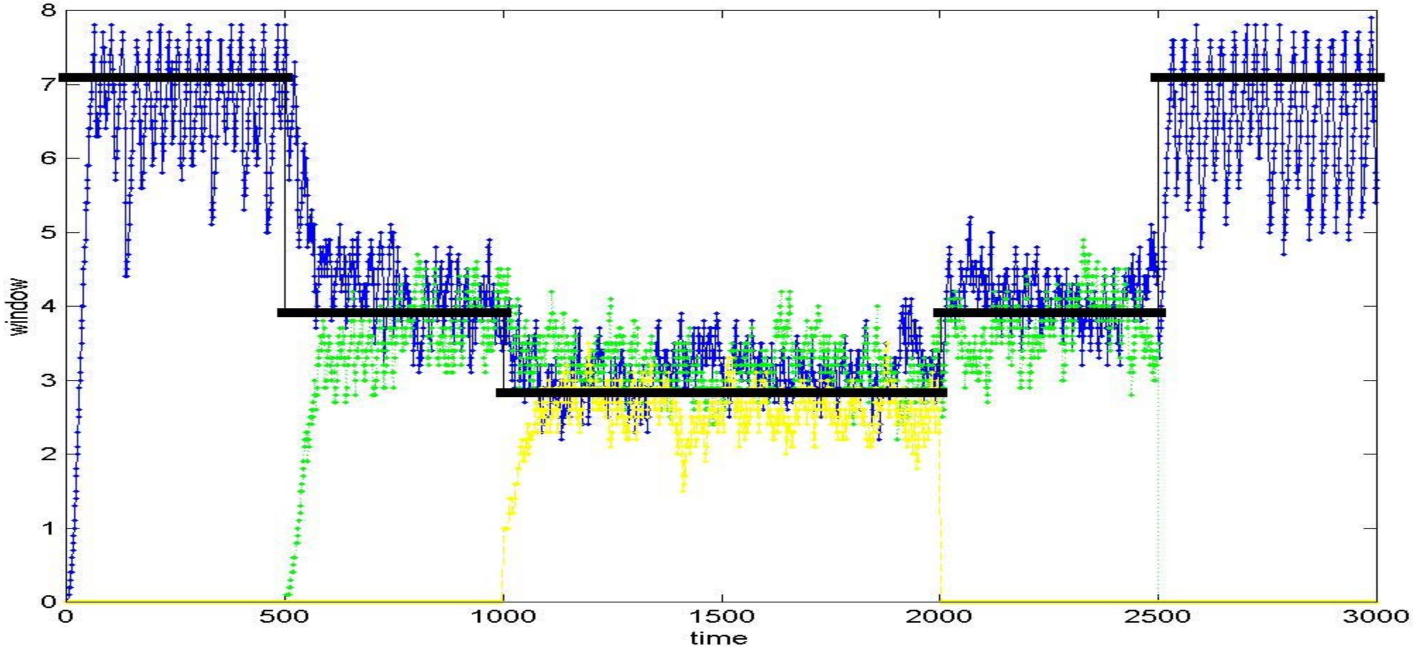
- Equilibrium characterization

$$\frac{2}{2 + x_s^2 D_s^2} = p \qquad \Rightarrow \qquad x_s \approx \frac{\sqrt{2}}{D_s \sqrt{p}}$$

Duality $\Rightarrow \quad U_s^{reno}(x_s)$

- Congestion measure p = loss
- Implications
  - Reno equalizes window $w = D_s x_s$
  - inversely proportional to delay D_s
  - $1/\sqrt{p}$ dependence for small p

Validation - Reno



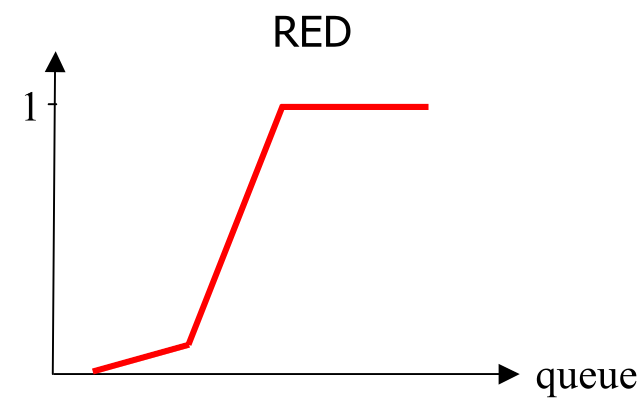
- 30 sources, 3 groups with RTT = 3, 5, 7ms + 6ms (queueing delay)
  Link capacity = 64 Mbps, buffer = 50 kB
- Measured windows equalized, match well with theory (black line)

Reno/RED

- Algorithm model

$$F_s\big(p(t), x(t)\big) = x_s(t) + \frac{(1 - m(p(t)))}{D_s^2} - \frac{x_s^2(t)}{2} m(p(t))$$

$$G(p(t), x(t)) = \left[p(t) + \sum_s x_s(t) - c \right]^+$$



Reno/RED

- Algorithm model

$$F_s\big(p(t), x(t)\big) = x_s(t) + \frac{(1 - m(p(t)))}{D_s^2} - \frac{x_s^2(t)}{2} m(p(t))$$

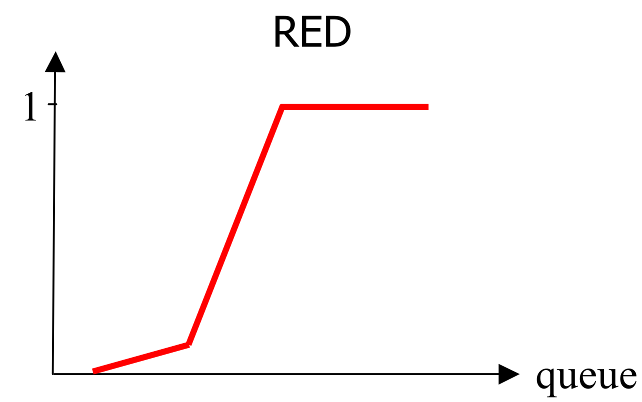
$$G(p(t), x(t)) = \left[p(t) + \sum_s x_s(t) - c \right]^+$$

- Equilibrium characterization

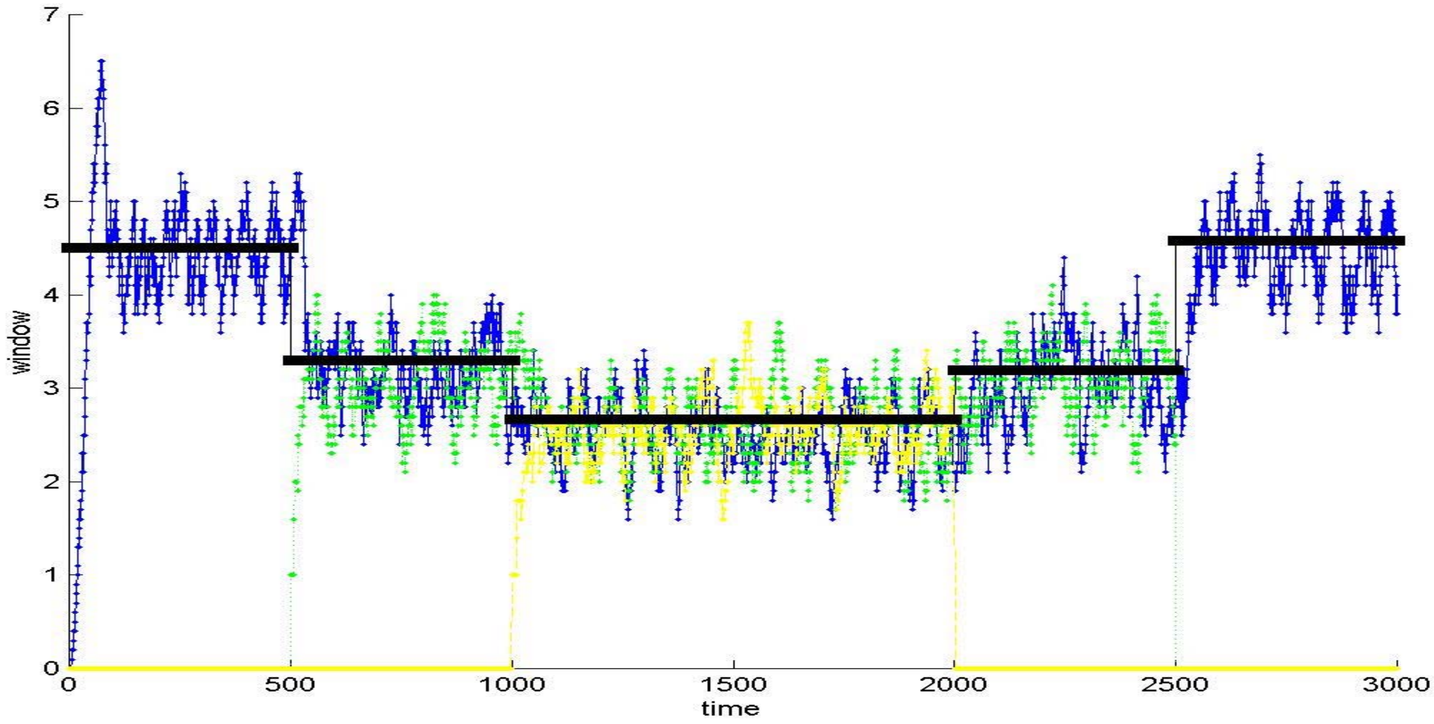
$$\frac{2}{2 + x_s^2 D_s^2} = m(p)$$

Duality $\Rightarrow U_s^{reno/red}(x_s)$

- Congestion measure p = queue
  - Queue increases with load



Validation – Reno/RED



- 30 sources, 3 groups with RTT = 3, 5, 7 ms + 6 ms (queueing delay)
- Link capacity = 64 Mbps, buffer = 50 kB

Reno/REM

- Algorithm model

$$F_s\left(p(t), x(t)\right) = x_s(t) + \frac{(1 - m(p(t)))}{D_s^2} - \frac{x_s^2(t)}{2} m(p(t))$$

$$G(p(t), x(t)) = \left[p(t) + \gamma(\alpha(b(t) - b^*) + \sum_s x_s(t) - c \right]^+$$

Reno/REM

- Algorithm model

$$F_s\big(p(t), x(t)\big) = x_s(t) + \frac{(1 - m(p(t)))}{D_s^2} - \frac{x_s^2(t)}{2} m(p(t))$$

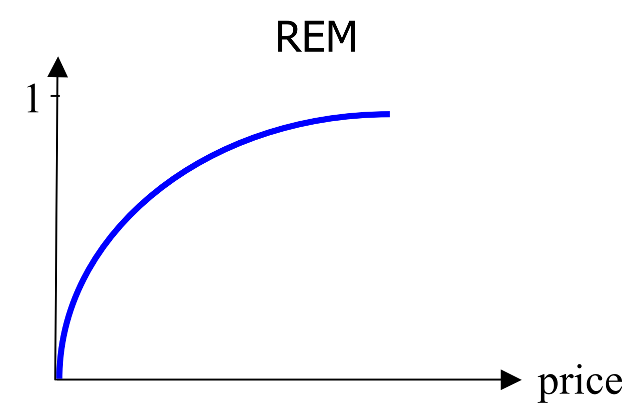
$$G(p(t), x(t)) = \left[p(t) + \gamma\Big(\alpha(b(t) - b^*) + \sum_s x_s(t) - c\Big) \right]^+$$

- Equilibrium characterization

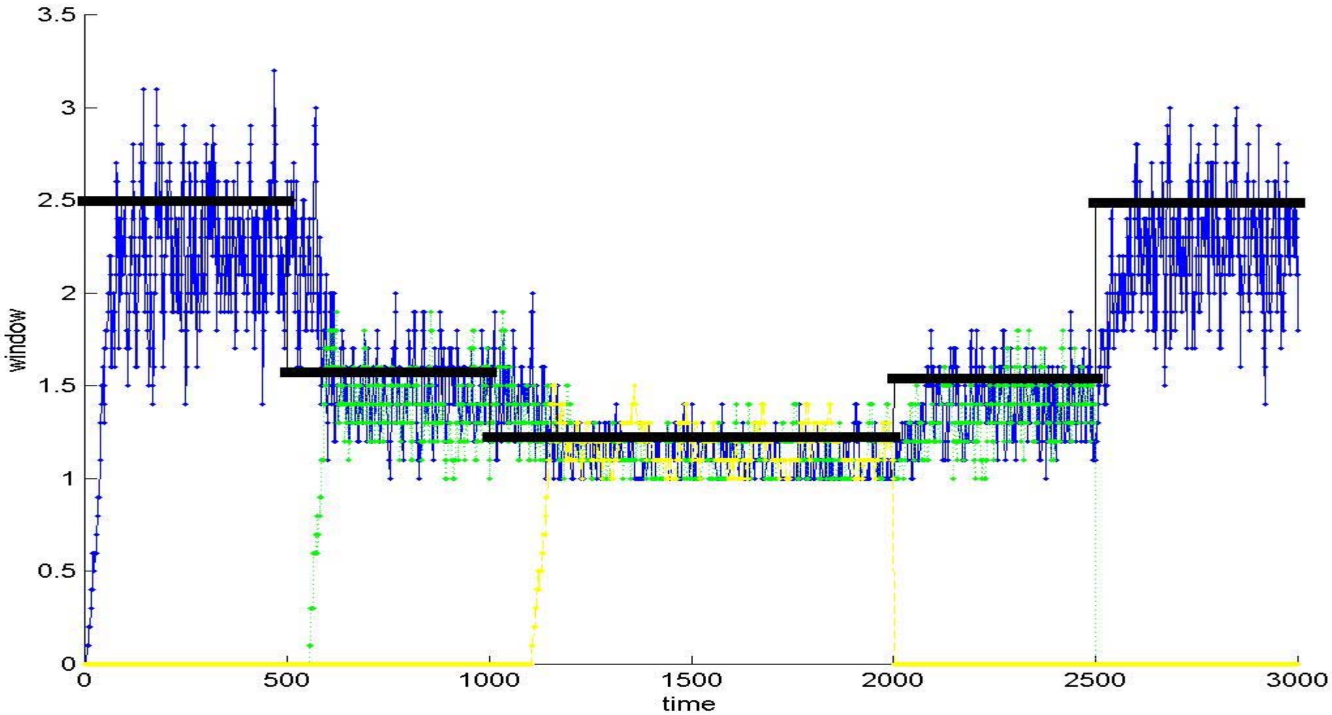
$$\frac{2}{2 + x_s^2 D_s^2} = m(p)$$

Duality $\Rightarrow U_s^{reno/rem}(x_s)$

- Congestion measure p = price
  - Match queue and rate
  - Sum prices

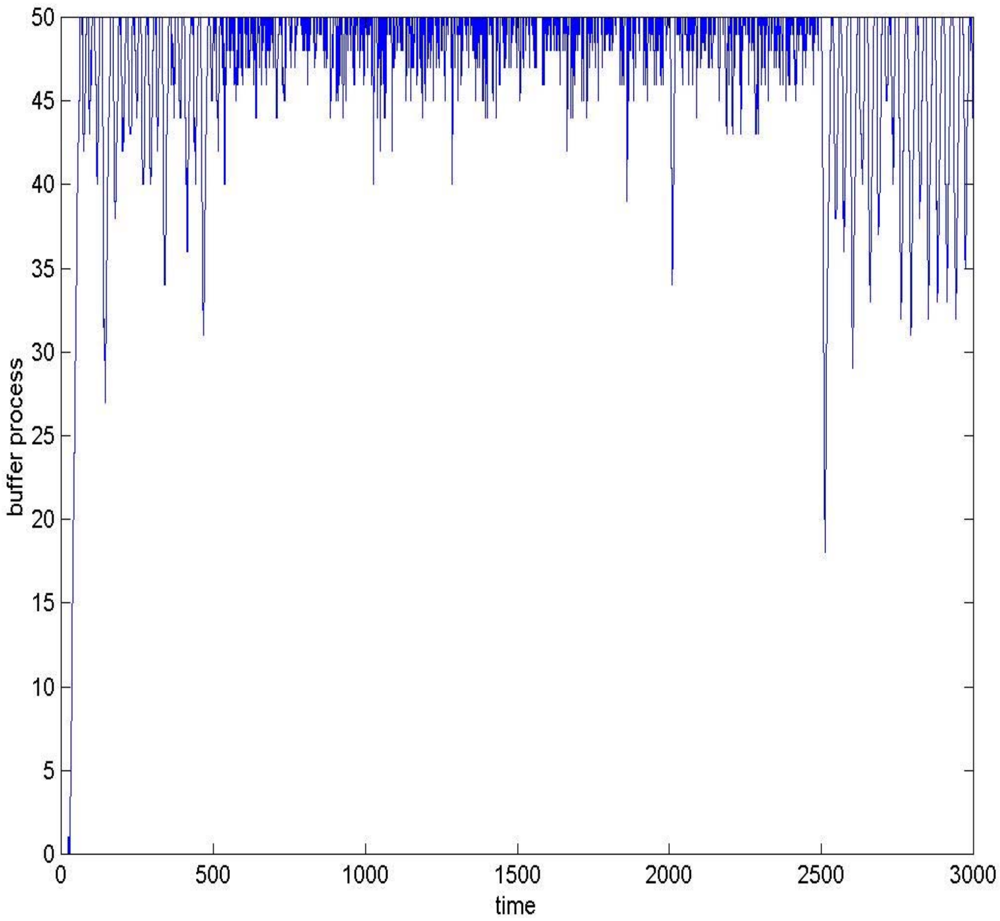


Validation – Reno/REM



- 30 sources, 3 groups with RTT = 3, 5, 7 ms
- Link capacity = 64 Mbps, buffer = 50 kB
- Smaller window due to small RTT (~0 queueing delay)

Queue – Reno/DropTail



mean queue = 47 pkts

buffer capacity = 50 pkts

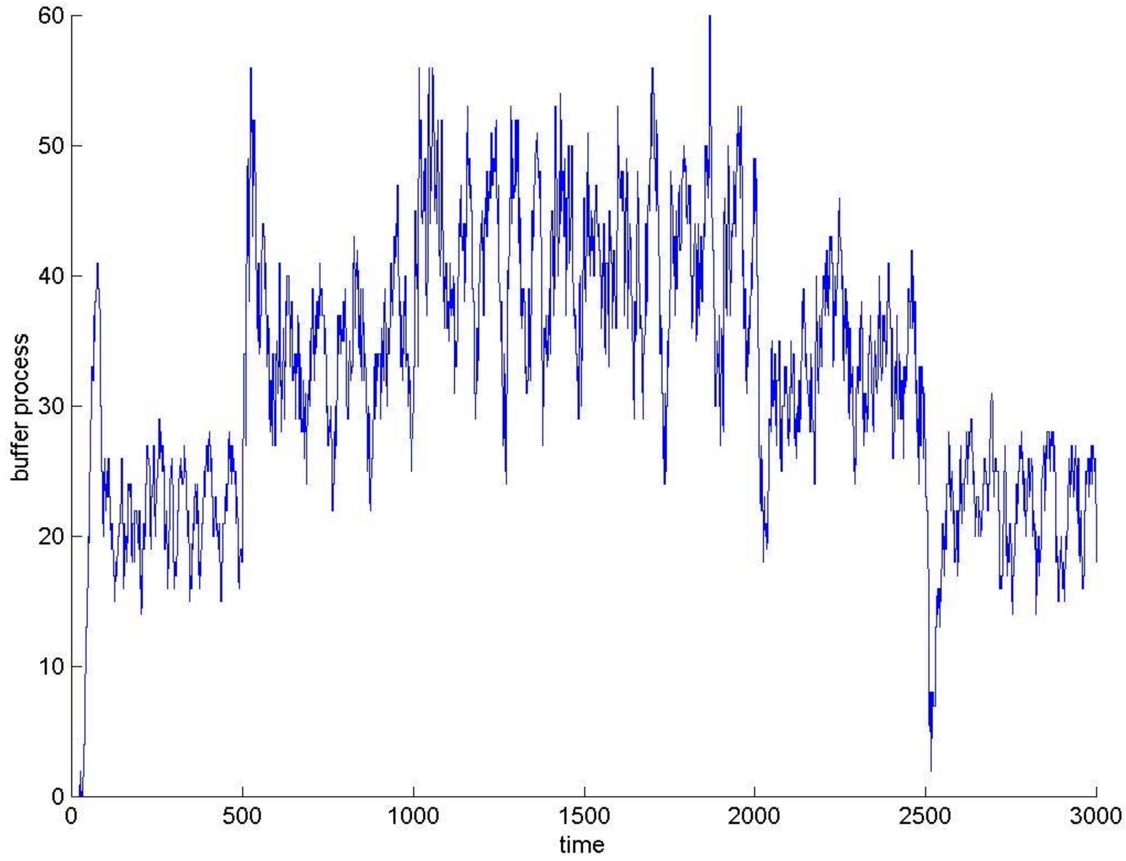
Queue close to full if

- many sources

If buffer capacity is small

- wild oscillation of queue and windows

Queue – Reno/RED

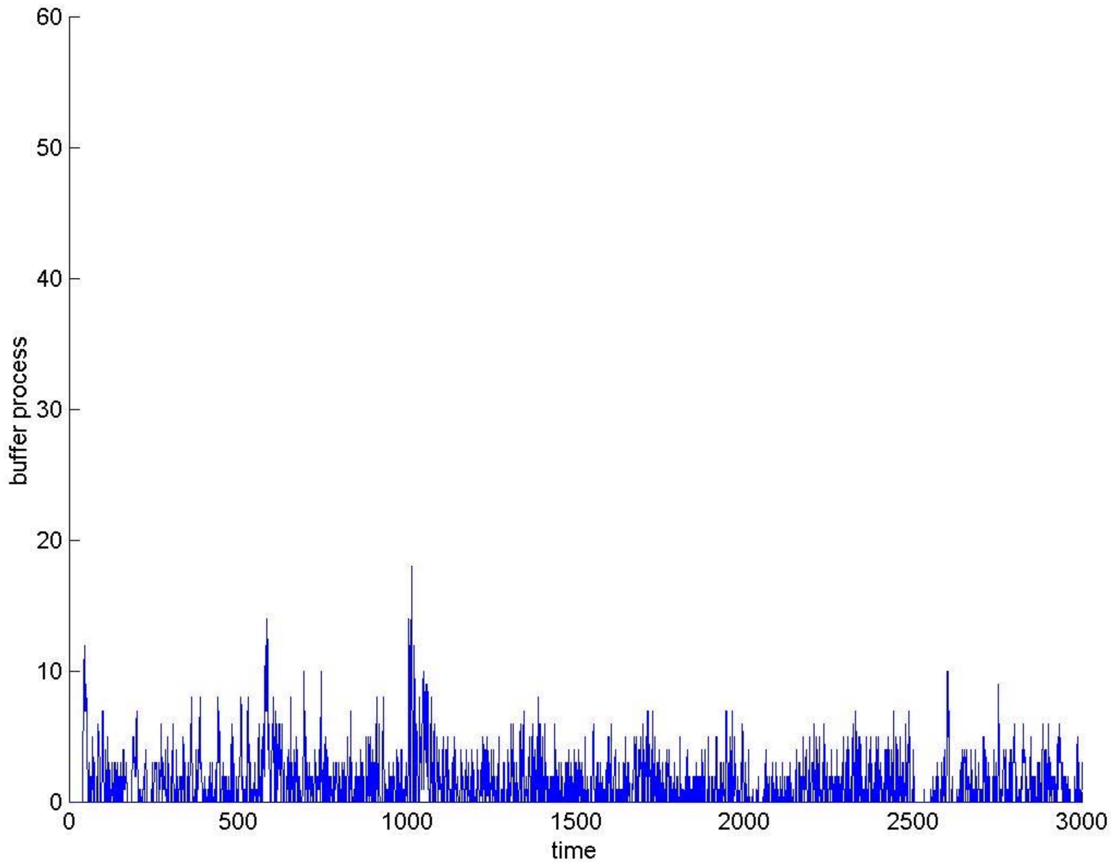


Queue increases as sources activate

RED parameters:

min_th = 10 pkts, max_th = 40 pkts, max_p = 0.1

Queue – Reno/REM



- Very small queue
  - mean = 1.5 pkts
- Yet, utilization = 92%

REM parameters: $\gamma = 0.05$, $\alpha = 0.4$, $\phi = 1.15$

Reno & Basic Algorithm

- Basic algorithm

  source $\bar{x}_s(t+1) = U_s'^{-1}(p(t))$

- TCP smoothed version of Basic Algorithm …

Reno & Basic Algorithm

- Basic algorithm

source $\quad \bar{x}_s(t+1) = U_s'^{-1}(p(t))$

- TCP smoothed version of Basic Algorithm …

- Reno/DropTail, Reno/RED, Reno/REM

$$x_s(t+1) = \left[x_s(t) + \frac{m(p(t))}{2} (\underbrace{\bar{x}_s^2(t)}_{U_s'^{-1}(p(t))} - x_s^2(t)) \right]^+$$

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model (F, G, U)
    - Queue management G : RED, REM
    - TCP G and U : Reno, Vegas
    - Performance of REM
  - Feedback control model

Vegas model

```
for every RTT
{    if W/RTTmin – W/RTT < α then W ++
     if W/RTTmin – W/RTT > α then W --    }
for every loss
        W := W/2
```

queue size

$$F: \quad x_s(t+1) = \left\{ x_s(t) + \frac{1}{D_s^2} \right. \qquad \text{if} \quad w_s(t) - d_s x_s(t) < \alpha_s d_s$$

$$x_s(t+1) = \left\{ x_s(t) - \frac{1}{D_s^2} \right. \qquad \text{if} \quad w_s(t) - d_s x_s(t) > \alpha_s d_s$$

$$x_s(t+1) = x_s(t) \qquad \text{else}$$

$$G: \quad p_l(t+1) = [p_l(t) + x^l(t)/c_l - 1]^+$$

Vegas Utility

- Equilibrium $(x, p) = (F, G)$

$$w_s - d_s x_s = \alpha_s d_s$$

$$U_s^{reno}(x_s) = \alpha_s d_s \log x_s$$

Vegas & Basic Algorithm

- Basic algorithm

  source $\quad \bar{x}_s(t+1) = U_s'^{-1}(p(t))$

- TCP smoothed version of Basic Algorithm …

Vegas & Basic Algorithm

- Basic algorithm

  source $\quad \bar{x}_s(t+1) = U_s'^{-1}(p(t))$

- TCP smoothed version of Basic Algorithm …

- Vegas

$$x_s(t+1) = \left\{ x_s(t) + \frac{1}{D_s^2} \right. \qquad \text{if} \quad x_s(t) < \bar{x}_s(t)$$

$$x_s(t+1) = \left\{ x_s(t) - \frac{1}{D_s^2} \right. \qquad \text{if} \quad x_s(t) > \bar{x}_s(t)$$

$$x_s(t+1) = x_s(t) \qquad \text{else}$$

$$\boxed{U_s'^{-1}(p(t))}$$

Implications

- Delay
  - Congestion measures $\frac{q_l(t)}{c_l}$ = end to end *queueing* delay
  - Sets rate $x_s(t) = \alpha_s \frac{d_s}{q^s(t)}$
  - Equilibrium condition: Little's Law
- Fairness
  - Weighted proportional fairness
- Loss
  - No loss if buffers are sufficiently large
  - Otherwise: equilibrium not attainable, loss unavoidable (revert to Reno)

Validation - Vegas

	Source 1	Source 3	Source 5
RTT (ms)	17.1 (17)	21.9 (22)	41.9 (42)
Rate (pkts/s)	1205 (1200)	1228 (1200)	1161 (1200)
Window (pkts)	20.5 (20.4)	27 (26.4)	49.8 (50.4)
Avg backlog (pkts)	9.8 (10)		

measured

theory

- Single link, capacity = 6 pkts/ms
- 5 sources with different propagation delays, α_s = 2 pkts/RTT

Persistent congestion

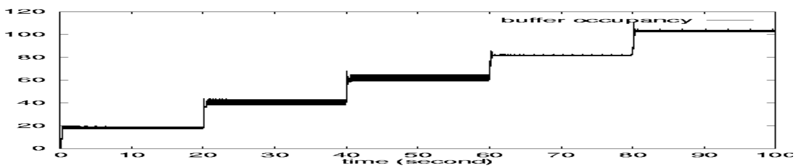
- Vegas exploits buffer process to compute prices (queueing delays)
- Persistent congestion due to
  - Coupling of buffer & price
  - Error in propagation delay estimation
- Consequences
  - Excessive backlog
  - Unfairness to older sources

Theorem

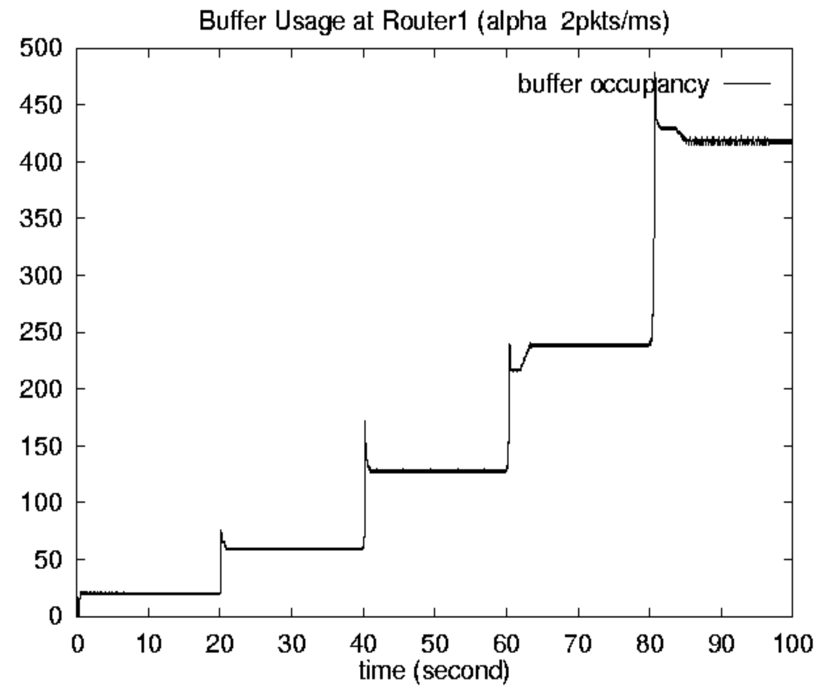
A relative error of ε_s in propagation delay estimation distorts the utility function to

$$\hat{U}_s(x_s) = (1 + \varepsilon_s)\alpha_s d_s \log x_s + \varepsilon_s d_s x_s$$

Evidence



Without estimation error



With estimation error

- Single link, capacity = 6 pkt/ms, α_s = 2 pkts/ms, d_s = 10 ms
- With finite buffer: Vegas reverts to Reno

Evidence

Source rates (pkts/ms)

#	src1	src2	src3	src4	src5
1	5.98 (6)				
2	2.05 (2)	3.92 (4)			
3	0.96 (0.94)	1.46 (1.49)	3.54 (3.57)		
4	0.51 (0.50)	0.72 (0.73)	1.34 (1.35)	3.38 (3.39)	
5	0.29 (0.29)	0.40 (0.40)	0.68 (0.67)	1.30 (1.30)	3.28 (3.34)

#	queue (pkts)	baseRTT (ms)
1	19.8 (20)	10.18 (10.18)
2	59.0 (60)	13.36 (13.51)
3	127.3 (127)	20.17 (20.28)
4	237.5 (238)	31.50 (31.50)
5	416.3 (416)	49.86 (49.80)

Vegas/REM

- To preserve Vegas utility function & rates

$$x_s = \alpha_s \frac{d_s}{p^s}$$

end2end queueing delay

Vegas/REM

- To preserve Vegas utility function & rates

$$x_s = \alpha_s \frac{d_s}{p^s}$$

end2end price

- REM
  - Clear buffer : estimate of d_s
  - Sum prices : estimate of p^s

Vegas/REM

- To preserve Vegas utility function & rates

$$x_s = \alpha_s \frac{d_s}{p^s}$$

end2end price

- REM
  - Clear buffer : estimate of d_s
  - Sum prices : estimate of p^s
- Vegas/REM

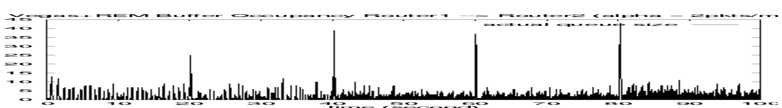
$$x_s(t+1) = \left\{ x_s(t) + \frac{1}{D_s^2} \right. \qquad \text{if} \quad x_s(t) < \hat{x}_s(t)$$

$$x_s(t+1) = \left\{ x_s(t) - \frac{1}{D_s^2} \right. \qquad \text{if} \quad x_s(t) > \hat{x}_s(t)$$

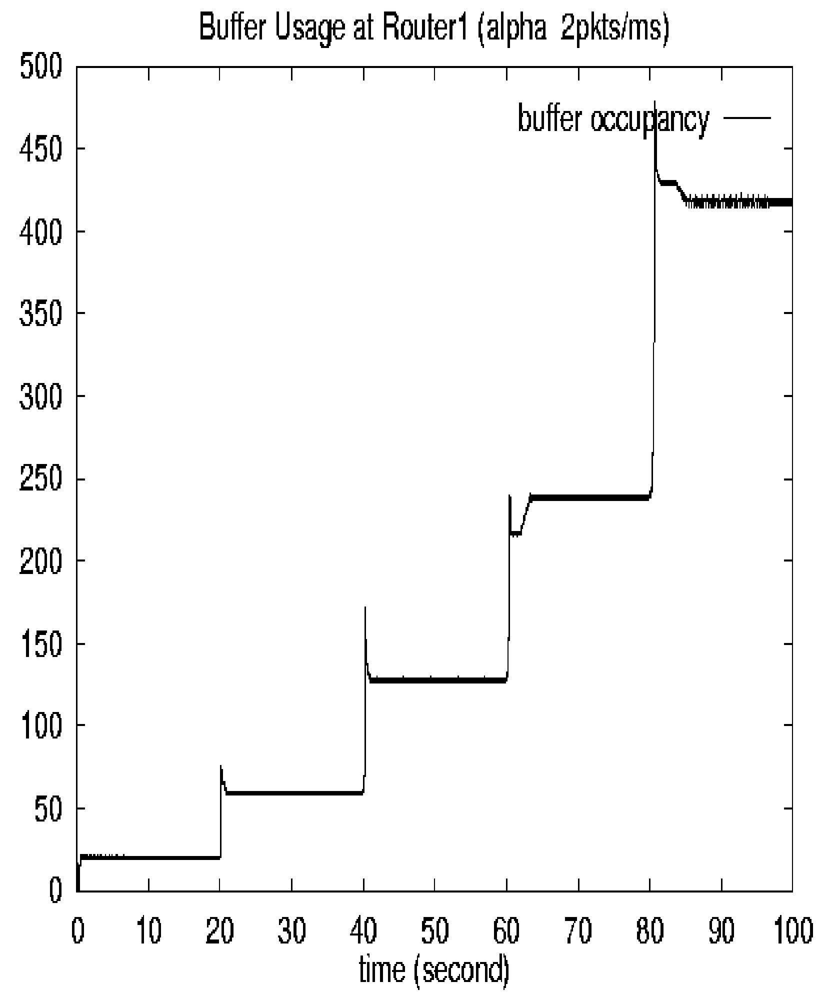
$$x_s(t+1) = x_s(t) \qquad \text{else}$$

Performance

peak = 43 pkts
utilization : 90% - 96%



Vegas/REM



Vegas

Conclusion

Duality model of TCP: ***(F, G, U)***

$$x(t+1) = F(p(t), x(t))$$

$$p(t+1) = G(p(t), x(t))$$

Reno, Vegas

- Maximize aggregate utility
- With different utility functions

DropTail, RED, REM

- Decouple congestion & performance
- Match rate, clear buffer
- Sum prices

Food for thought

- How to tailor utility to application?
  - Choosing congestion control automatically fixes utility function
  - Can use utility function to determine congestion control

Outline

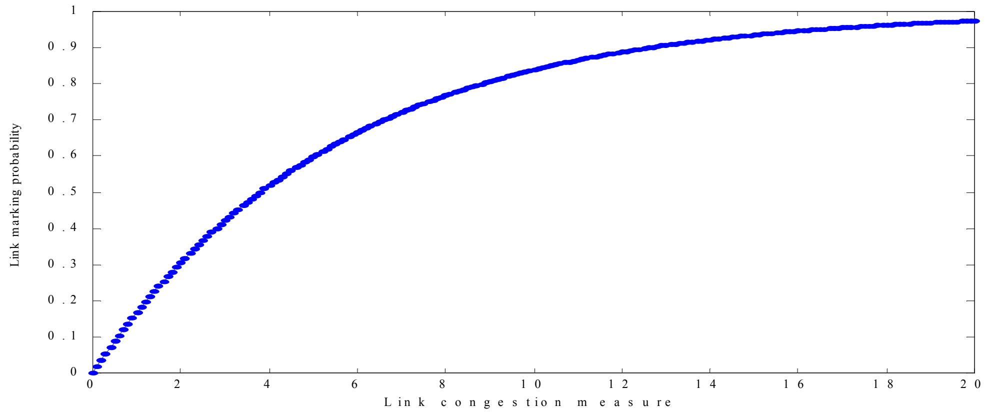
- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model (F, G, U)
    - Queue management G : RED, REM
    - TCP G and U : Reno, Vegas
    - Performance of REM
  - Feedback control model

REM (Athuraliya & Low 2000)

- Congestion measure: price

$$p_l(t+1) = [p_l(t) + \gamma(\alpha_l\, b_l(t) + x^l(t) - c_l)]^+$$

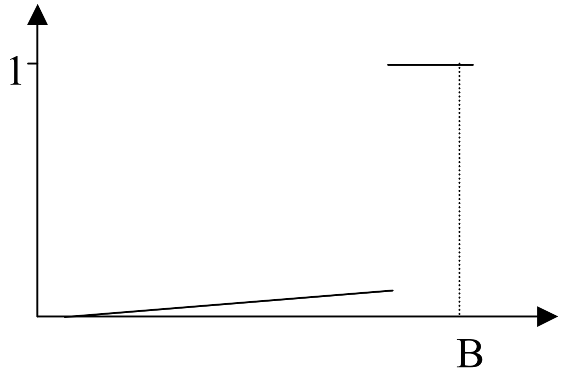
- Embedding: exponential probability function



- Feedback: dropping or ECN marking

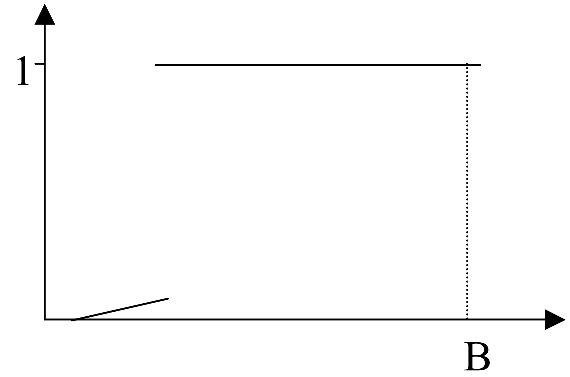
Performance Comparison

- RED



High utilization

OR



Low delay/loss

- REM

match rate

AND

clear buffer

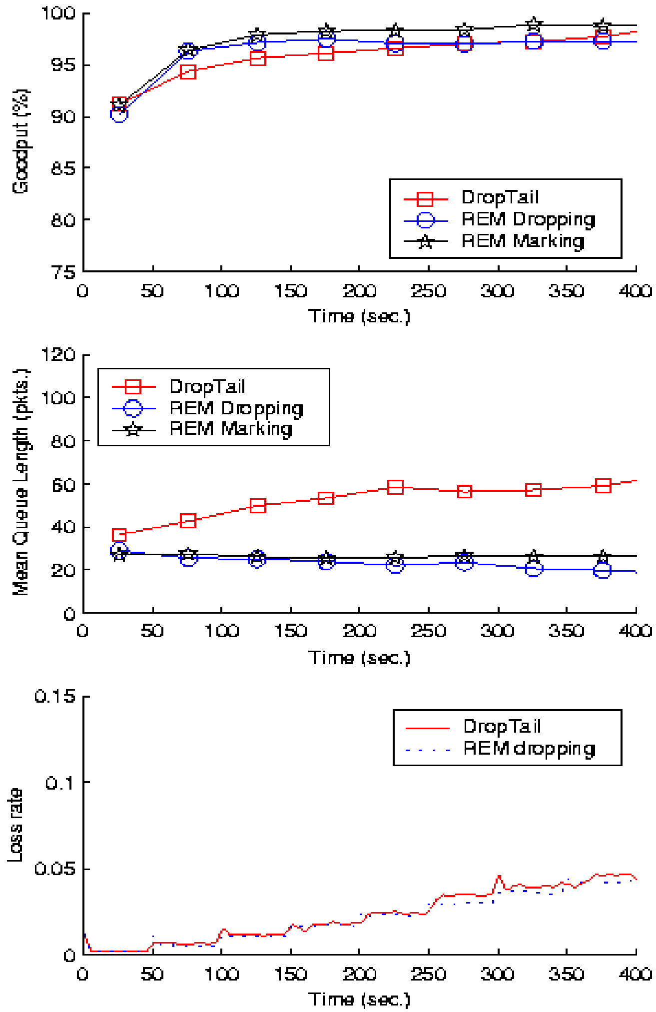
High utilization

Low delay/loss

Comparison with RED

REM

- Goodput
- Queue
- Loss

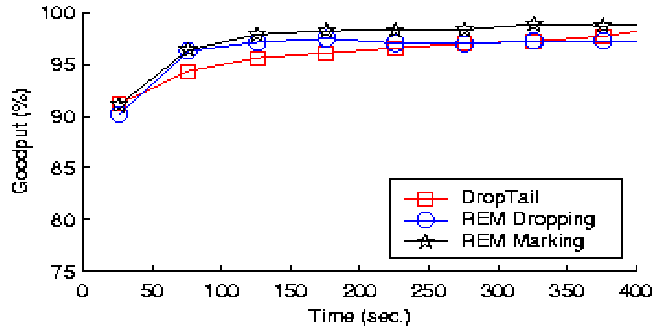


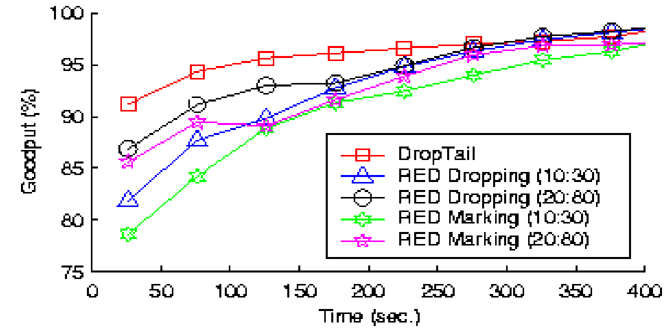
Comparison with RED

REM

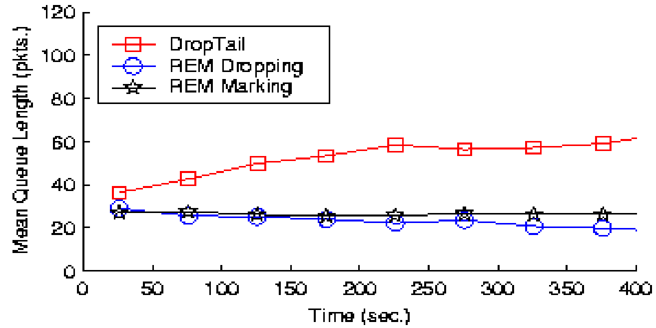
RED

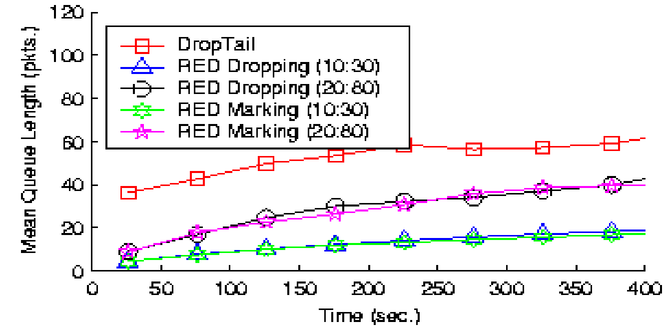
- Goodput



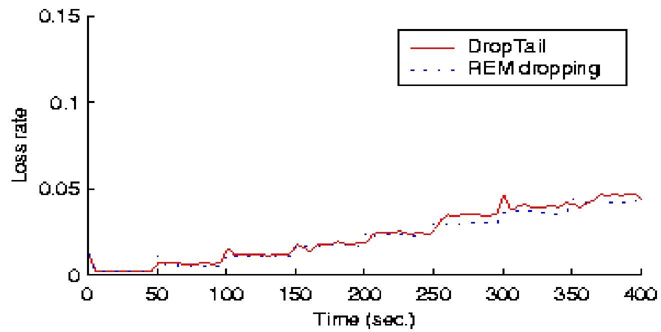


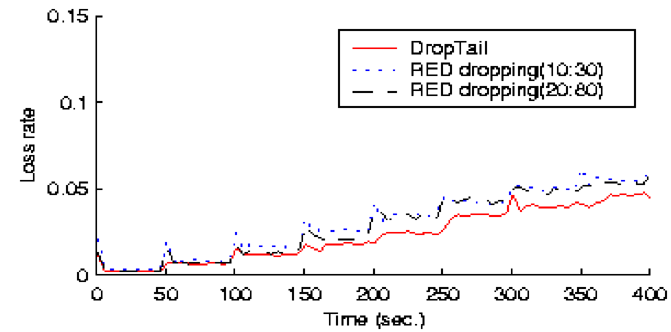
- Queue





- Loss





Application: Wireless TCP

- Reno uses loss as congestion measure
- In wireless, significant losses due to
  - Fading
  - Interference
  - Handover
  - Not buffer overflow (congestion)
- Halving window too drastic
  - Small throughput, low utilization

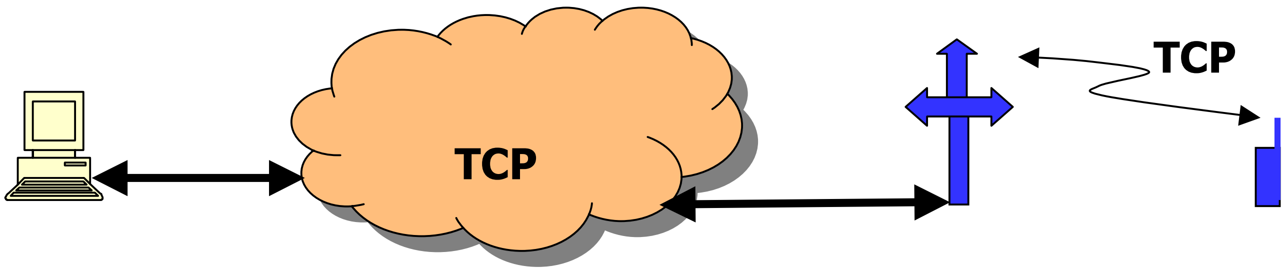
Proposed solutions

- Ideas
  - Hide from source noncongestion losses
  - Inform source of noncongestion losses
- Approaches
  - Link layer error control
  - Split TCP
  - Snoop agent
  - SACK+ELN (Explicit Loss Notification)

Link layer protocols

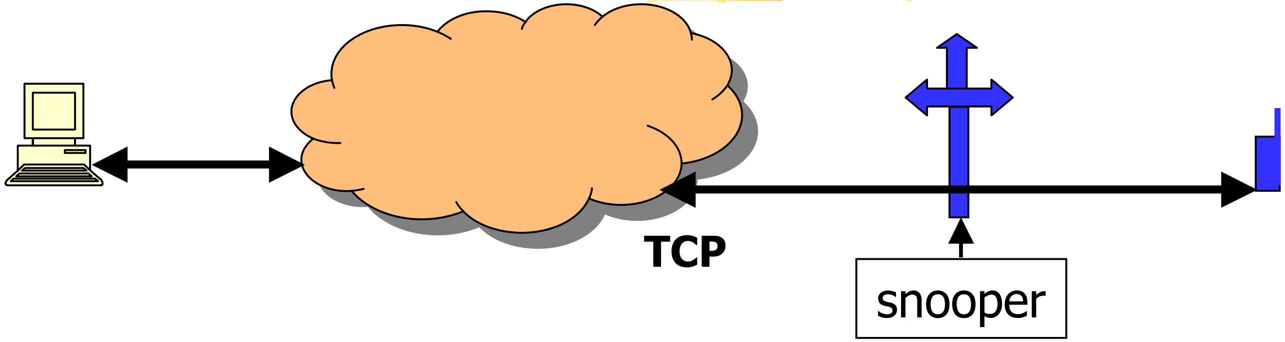
- Interference suppression
  - Reduces link error rate
  - Power control, spreading gain control
- Forward error correction (FEC)
  - Improves link reliability
- Link layer retransmission
  - Hides loss from transport layer
  - Source may timeout while BS retransmits

Split TCP



- Each TCP connection is split into two
  - Between source and BS
  - Between BS and mobile
- Disadvantages
  - TCP not suitable for lossy link
  - Overhead: packets TCP-processed twice at BS (vs. 0)
  - Violates end-to-end semantics
  - Per-flow information at BS complicates handover

Snoop protocol



- Snoop agent
  - Monitors packets in both directions
  - Detects loss by dupACKs or local timeout
  - Retransmits lost packet
  - Suppresses dupACKs
- Disadvantages
  - Cannot shield all wireless losses
  - One agent per TCP connection
  - Source may timeout while BS retransmits

Explicit Loss Notification

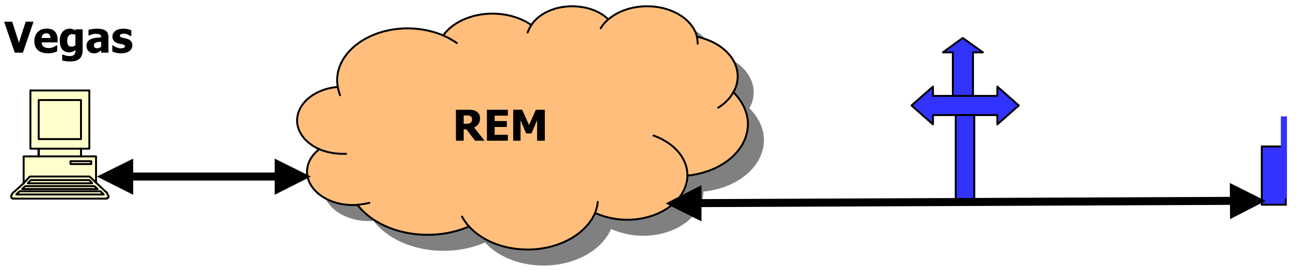
- Noncongestion losses are marked in ACKs
- Source retransmits but do not reduce window
- Effective in improving throughput
- Disadvantages
  - Overhead (TCP option)
  - May not be able to distinguish types of losses, e.g., corrupted headers

Third approach

- Problem
  - Reno uses loss as congestion measure
  - Two types of losses
    - Congestion loss: retransmit + reduce window
    - Noncongestion loss: retransmit
  - Previous approaches
    - Hide noncongestion losses
    - Indicate noncongestion losses
  - Our approach
    - Eliminates congestion losses (buffer overflows)

Third approach

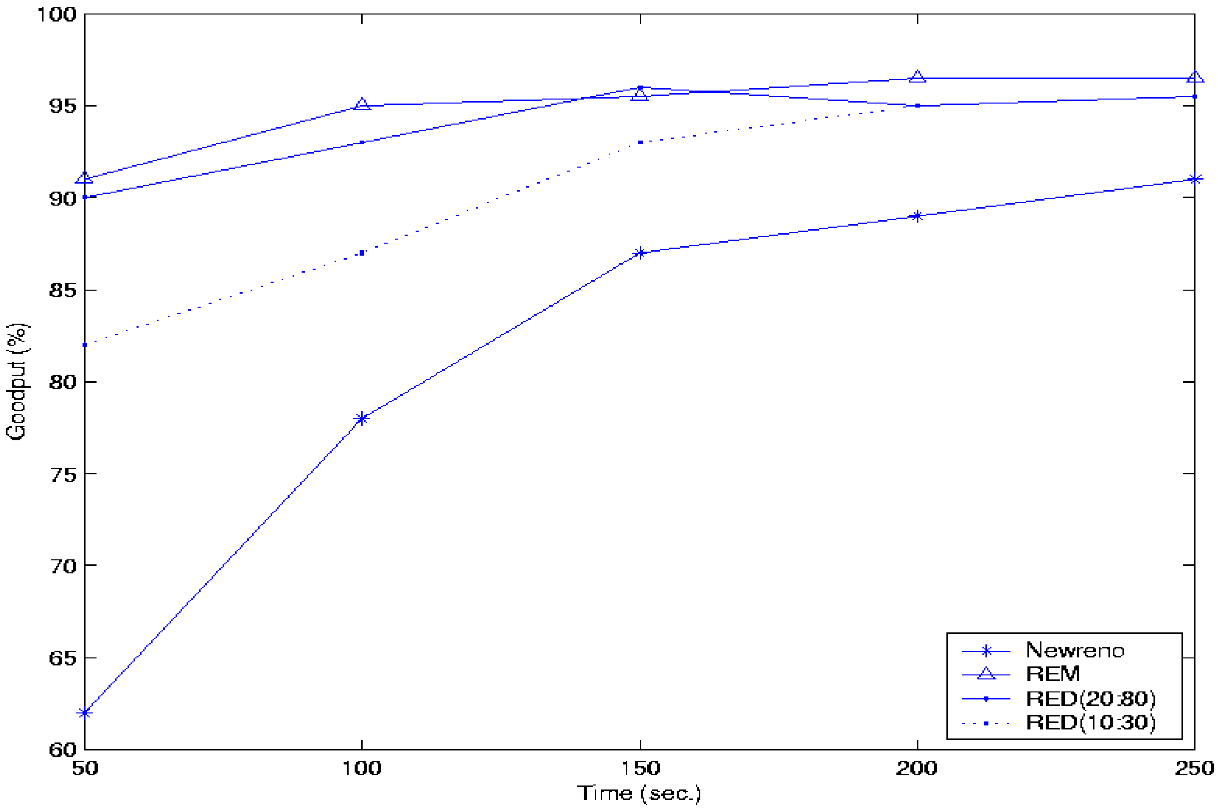
- Router
  - REM capable
- Host
  - Do not use loss as congestion measure



- Idea
  - REM clears buffer
  - Only noncongestion losses
  - Retransmits lost packets without reducing window

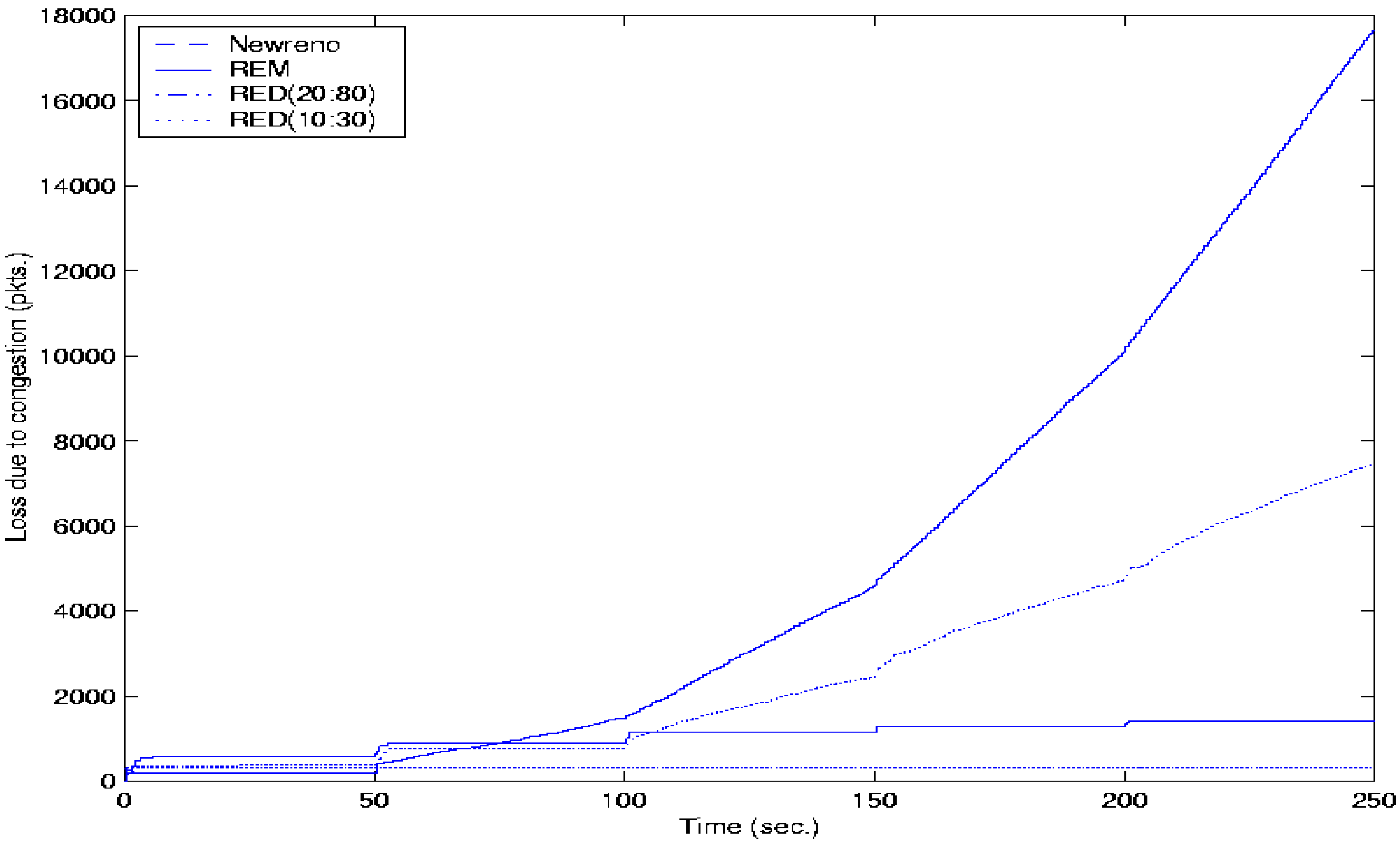
Performance

- Goodput



Performance

- Goodput



Food for thought

- How to tailor utility to application?
  - Choosing congestion control automatically fixes utility function
  - Can use utility function to determine congestion control
- Incremental deployment strategy?
  - What if some, but not all, routers are ECN-capable

Outline

- Introduction
- TCP Algorithms
  - Window flow control
  - Source algorithm: Tahoe, Reno, Vegas
  - Link algorithm: RED, REM, variants
- TCP Models
  - Renewal model
  - Duality model *(F, G, U)*
  - Feedback control model

Motivation

- Duality model
  - Equilibrium properties
    - Rate, loss, queue, delay, fairness
    - Optimality (utility function)
    - Interaction, TCP-friendliness
- Dynamic model
  - Stability & robustness
  - Transient behavior

Strategy

- Start with duality model
- Linearize around equilibrium point
  - Local stability & robustness
- Apply linear control & robustness theory
- Conclusions
  - TCP stability does not scale
  - How to scale

…… the rest are details

Model assumptions

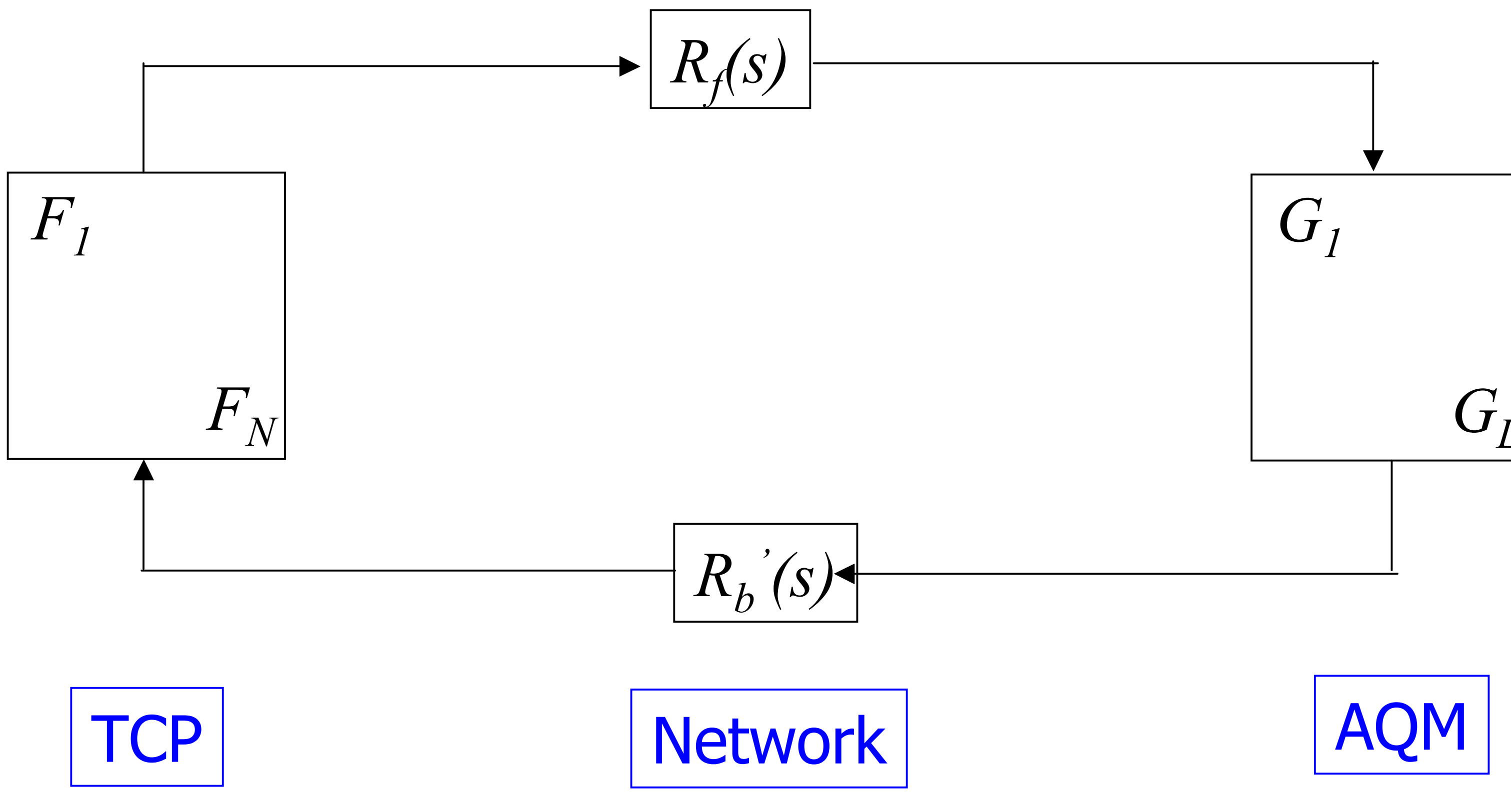
- Small marking probabilities
  - End to end marking probability

$$1-\prod_{l}(1-p_l)\approx\sum_{l}p_l$$

- Congestion avoidance dominates
- Receiver not limiting
- Decentralized
  - TCP algorithm depends only on end-to-end measure of congestion
  - AQM algorithm depends only on local & aggregate rate or queue
- Constant (equilibrium) RTT

Model structure

Multi-link multi-source network



Duality model - AIMD

AI MD

$$\dot{x}_i = \boxed{x_i(t-\tau_i)(1-q_i(t))\frac{1}{\tau_i^2 x_i(t)}} - \boxed{x_i(t-\tau_i)q_i(t)\frac{x_i(t)}{2}}$$

source rate

e2e prob

$$q_i(t) = \sum_l m_l(t-\tau_{li}^b)$$

Duality model - AIMD

AI MD

$$\dot{x}_i = \boxed{x_i(t-\tau_i)(1-q_i(t))\frac{1}{\tau_i^2 x_i(t)}} - \boxed{x_i(t-\tau_i)q_i(t)\frac{x_i(t)}{2}}$$

Linearize around equilibrium

$$\dot{x}_i = -x_i q_i x_i(t) - \frac{1}{\tau_i^2 q_i} q_i(t)$$

In Laplace domain

$$x_i(s) = -\frac{1}{\tau_i^2 q_i}\frac{1}{s + x_i q_i} q_i(s)$$

Duality model - AIMD

$$\dot{p}_l = y_l(t) - c_l$$

$$m_l(t) = m_l(p_l(t))$$

congestion measure

marking prob

Duality model - AIMD

$$\dot{p}_l = y_l(t) - c_l$$

$$m_l(t) = m_l(p_l(t))$$

Aggregate rate

$$y_l(t) = \sum_i x_i(t - \tau_i^f)$$

Linearize around equilibrium

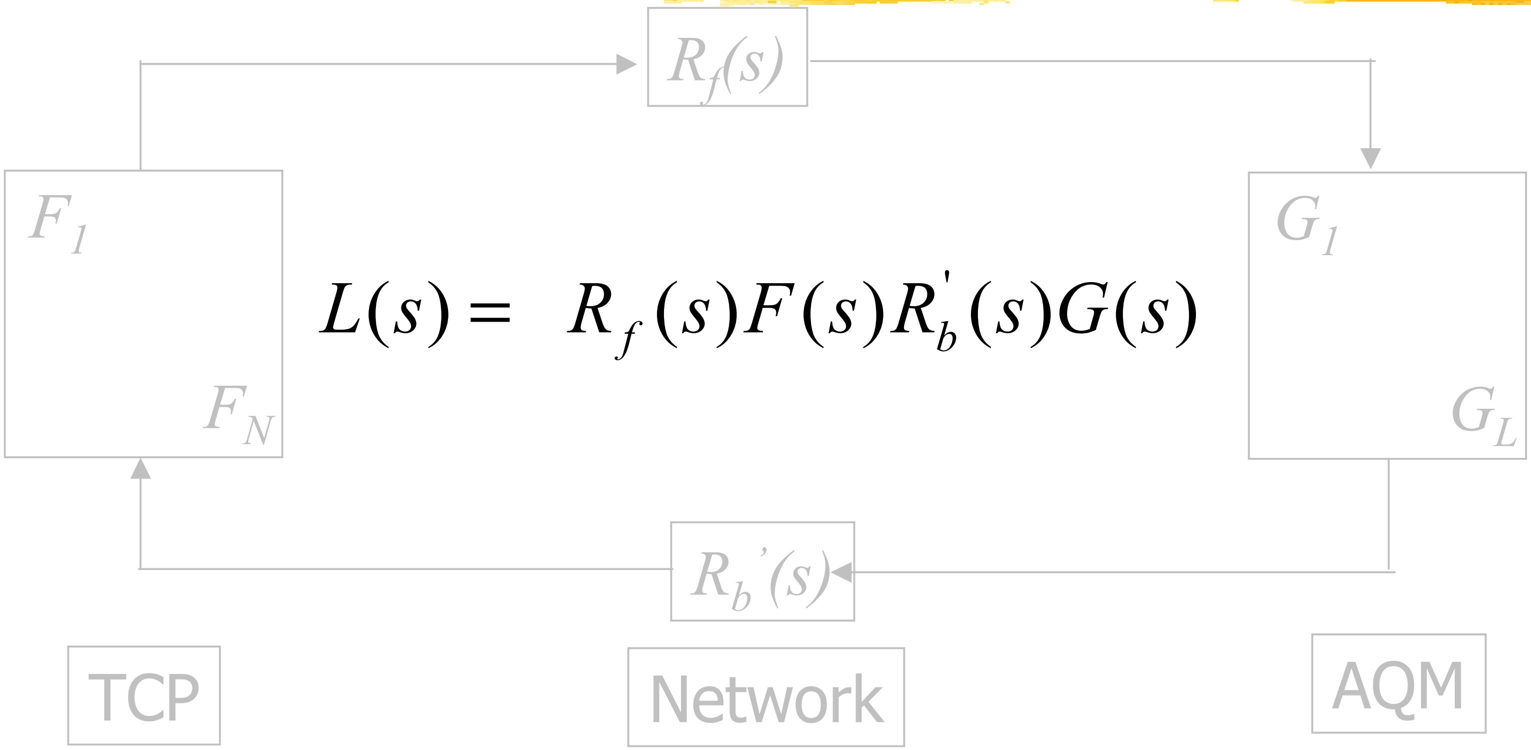
$$\dot{p}_l = y_i(t)$$

$$\dot{m}_l = m_l^{'}(p_l)\, y_l(t)$$

In Laplace domain

$$m_l(s) = m_l^{'}(p_l)\, y_l(s)$$

Loop function



$$L(s) = R_f(s)F(s)R_b'(s)G(s)$$

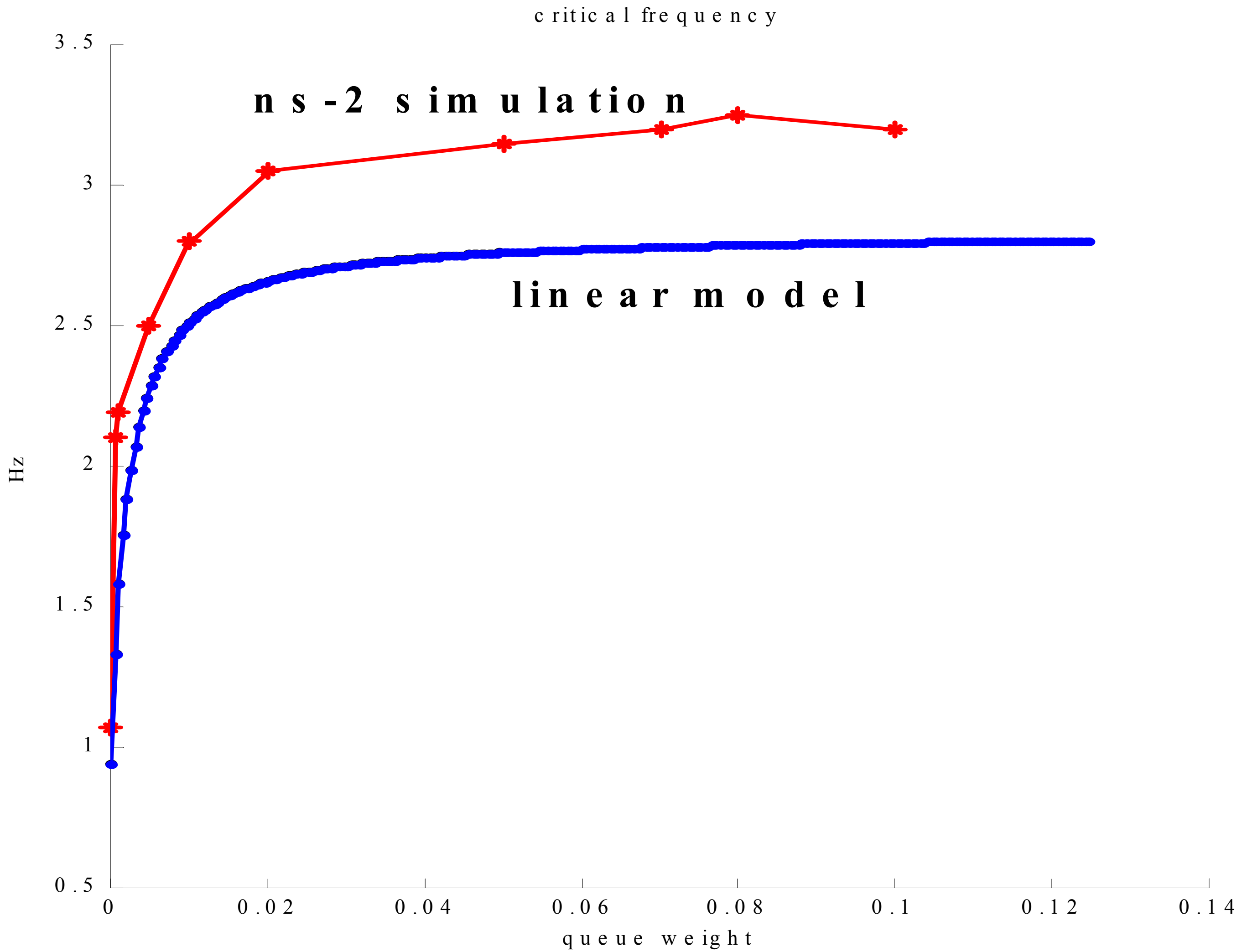
Theorem

Closed loop system is stable if and only if

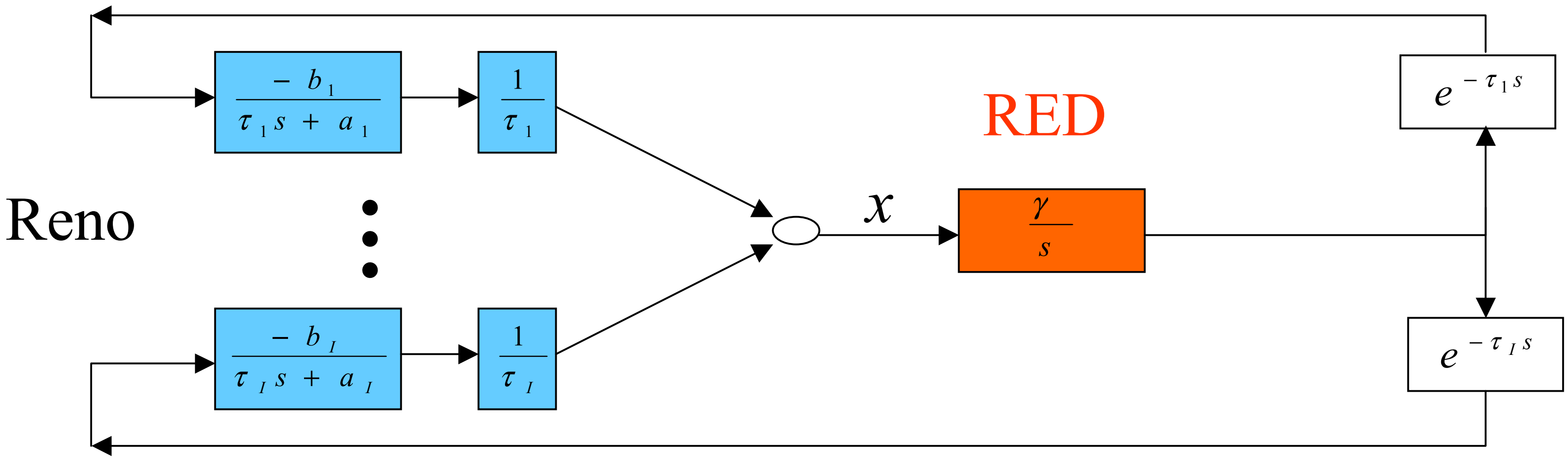
$$\det(I + L(s)) = 0$$

for no s in closed RHP

Validation


critical frequency
ns-2 simulation
linear model
Hz
queue weight
3.5
3
2.5
2
1.5
1
0.5
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14

Single link



$$P(s) = \frac{k}{s}\left(\frac{c^3\tau^2}{N^2}\right)\frac{e^{-\tau s}}{2\left(\frac{c\tau^2 s}{N}+2\right)}$$

The lag introduced by Reno is more of a problem than time delay of network.

This control scheme is unstable in many conditions, particular for large c!

Robustness of AIMD

- Robustness = stability as network scales
- Unstable as
  - Delay increases
  - Capacity increase
  - #sources decreases
- Stable when window size is small
- Unstable for future networks

…… is strong robustness possible?

The End

Discussion

Acronyms

ACK	Acknowledgement
AQM	Active Queue Management
ARP	Address Resolution Protocol
ARQ	Automatic Repeat reQuest
ATM	Asynchronous Transfer Mode
BSD	Berkeley Software Distribution
B	Byte (or octet) = 8 bits
bps	bits per second
CA	Congestion Avoidance
ECN	Explicit Congestion Notification
FIFO	First In First Out
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISOC	Internet Society
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
POS	Packet Over SONET
QoS	Quality of Service
RED	Random Early Detection/Discard
RFC	Request for Comment
RTT	Round Trip Time
RTO	Retransmission TimeOut
SACK	Selective ACKnowledgement
SONET	Synchronous Optical NETwork
SS	Slow Start
SYN	Synchronization Packet
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VQ	Virtual Queue
WWW	World Wide Web