TCP Westwood: Efficient Transport for High-speed wired/wireless Networks

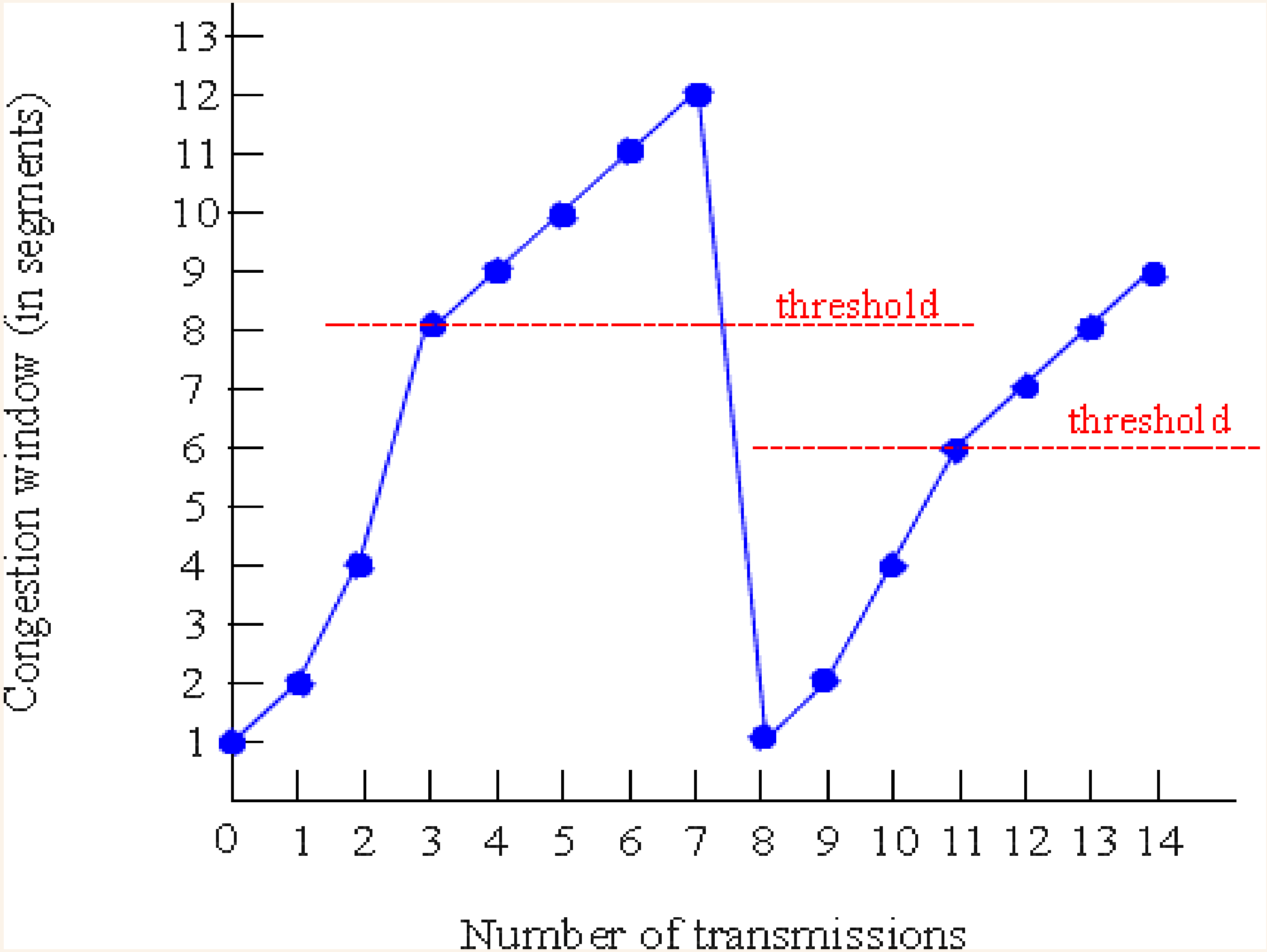
Mario Gerla, Medy Sanadidi, Ren Wang and Massimo Valla

UCLA Computer Science Department

Outline

1. TCP Overview
2. Bandwidth Estimation and TCP Westwood
3. Bandwidth Estimation vs. Rate Estimation
4. Performance Evaluation

TCP Congestion Control

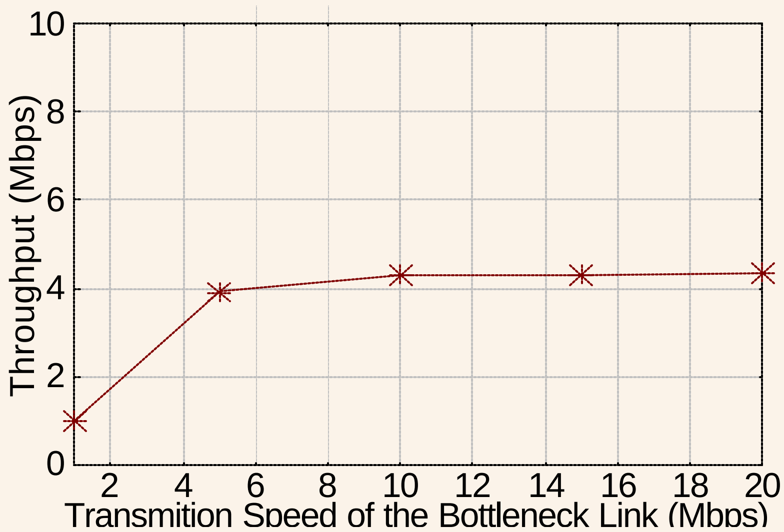


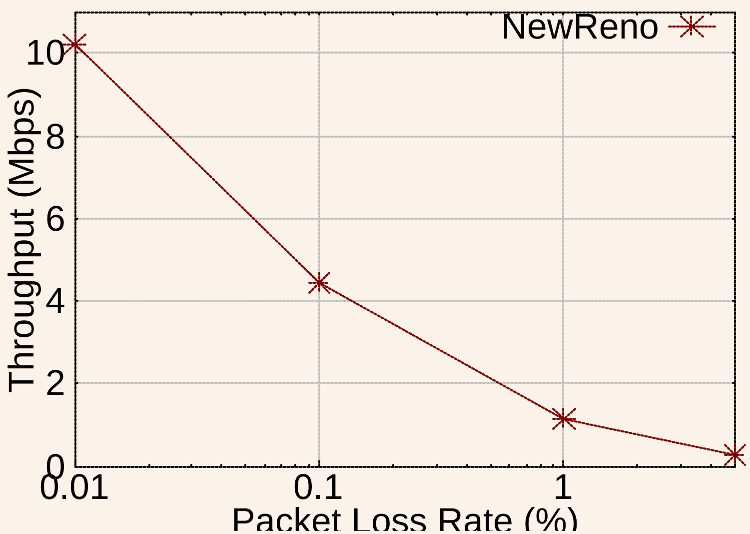
TCP Congestion Control Overview

- Evolved over the years: *Tahoe*, *Reno*, *NewReno*, *SACK*
- *Window* based, window size => offered traffic rate
- *Probing*: a connection “probes” for available bandwidth, when perceives packet loss, backs down to slower rate
- Two *phases* with differing probing behavior
- Initial design assumes packet losses are almost all due to *buffer overflow*
- Network layer assistance (*RED*, *ECN*, *XCP*, *BA-TCP*) for better efficiency, fairness and stability
- Link layer assistance has been suggested for hybrid networks where packets loss can be caused by both (1) *random error*, and (2) *buffer overflow*

TCP Reno Limitations (Ren, please clarify…)

- In wireless (lossy) networks random packet loss causes unnecessary window reduction and thus *inefficiency*
- In High speed networks blind halving of cwnd also results in *inefficiency* (why?)





Outline

1. TCP Overview
2. TCP Westwood and Rate Estimation
3. Bandwidth and Rate Estimation; adptive filter
4. Performance Evaluation

TCP Westwood (2000)

Key Idea:

- Enhance congestion control via the Rate Estimate (*RE*)
  - Estimate is computed at the sender by *sampling* and *exponential filtering*
  - Samples are determined from ACK *inter-arrival times* and info in ACKs regarding amounts of *bytes delivered*
- RE is used by sender to properly set *cwnd* and ssthresh after packet loss (indicated by 3 DUPACKs, or Timeout)

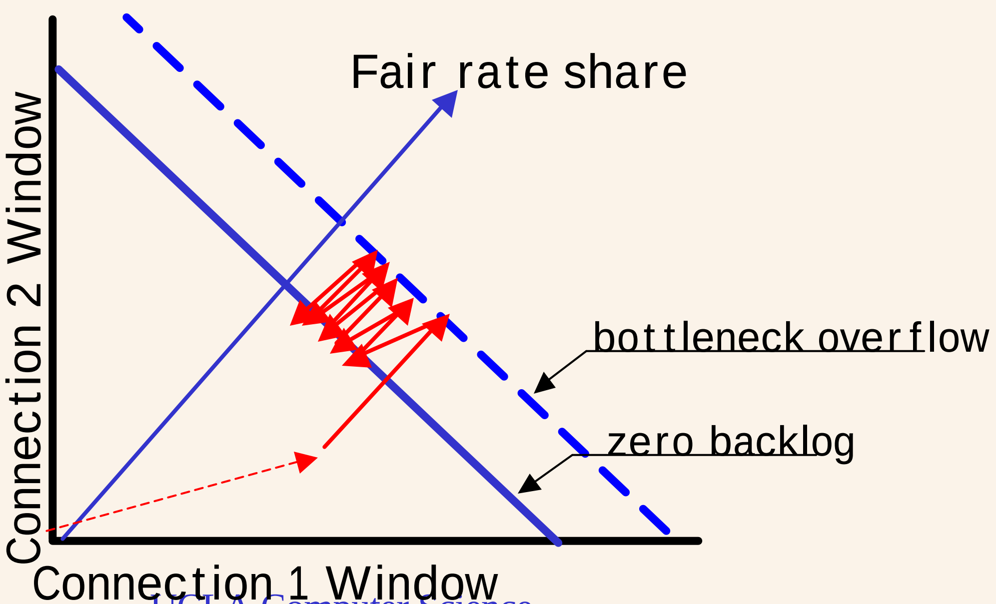
TCP Westwood: the control algorithm

- TCPW Algorithm Outline:
  - When three duplicate ACKs are detected:
    - set `ssthresh=RE*RTTmin` (instead of `ssthresh=cwin/2` as in Reno)
    - if `(cwin > ssthresh)` set `cwin=ssthresh`
  - When a TIMEOUT expires:
    - set `ssthresh=RE*RTTmin` (instead of `ssthresh=cwnd/2` as in Reno) and `cwin=1`

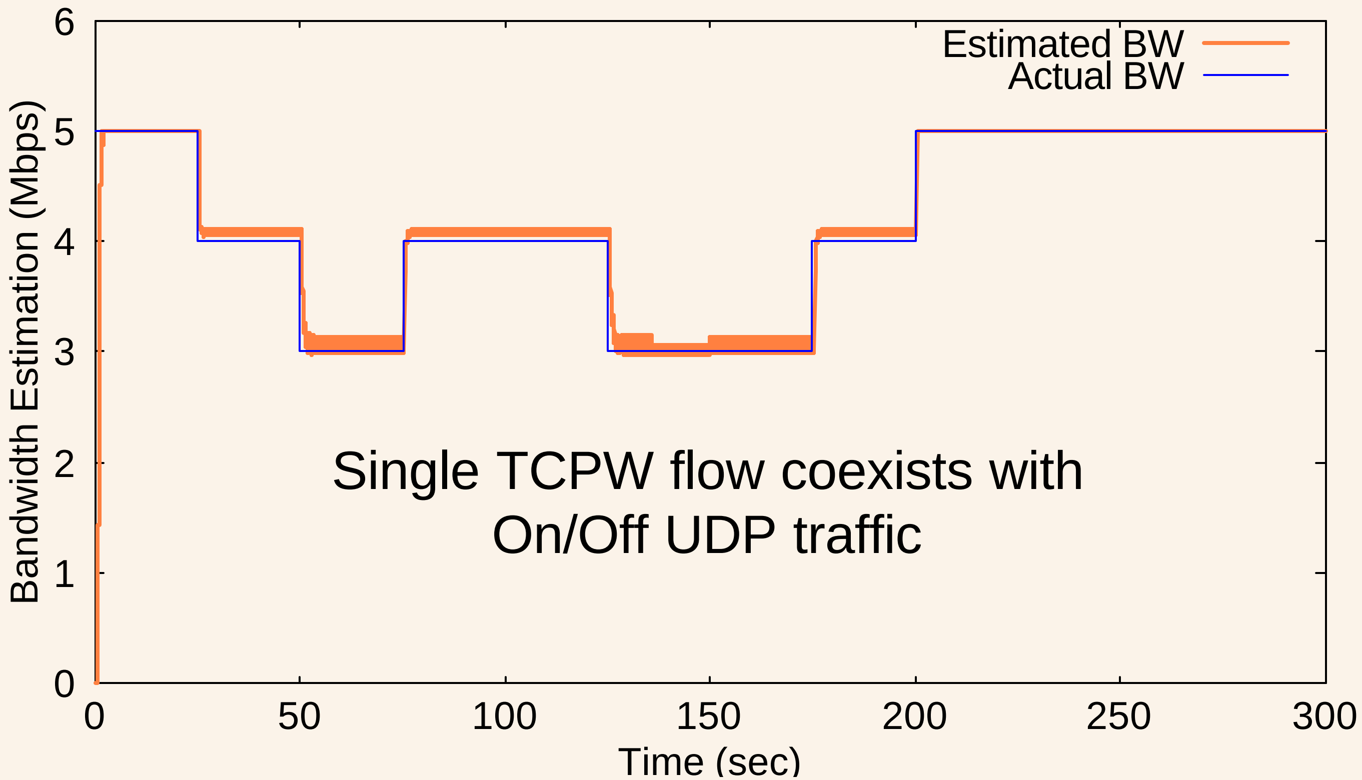
Note: RTTmin = min round trip delay experienced by the connection

At equilibrium, RE -> Fair RE

- Initially, two connections have different Wi and Ri.
- In the increase phase windows grow at the same rate
- Just before overflow : **Wi = Ri (Buf/Cap + RTTm) for i = 1,2**
- At overflow, RE estimate reduces windows back to "zero backlog" line, ie: **Wi = RE RTTm = Ri RTTm**



Fair RE = "Residual Bandwidth" Estimate?



Single TCPW flow at equilibrium does estimate "residual bottleneck bandwidth"

Related TCP + Bdw estimation work

- Note: the concept of using the bandwidth estimate to control the TCP flow is not new
- **TCP Vegas** monitors Bdw and RTT to infer the *bottleneck backlog*; then, from backlog it derives feedback to congestion window
- **Keshav's Packet Pair** scheme also monitors bandwidth to estimate the *bottleneck backlog* and compare to common target; it adjusts source rate

Related Works (cont)

- TCP Vegas
  - Sender watches for some sign that router's queue is building up and congestion will happen; e.g.,
    - RTT grows
    - sending rate flattens
  - Sender adjust sending rate to avoid filling the buffer
  - Fairness problem has been reported
- Packet Pair Flow Control
  - Using Packet Pair method to estimate bottleneck service rate to a connection
  - Adjusts the **transmitting rate** to maintain the TCP connection bottleneck queue equal to a target called **setpoint** (B)
  - Under round-robin, packet pair measures fair share; otherwise measure is inaccurate, and can overestimate fair share, up to link capacity

TCPW Benefits

What do we gain by using **RE** “feedback” in addition to **packet loss**)?

(a) better performance with **random loss** (ie, loss caused by random errors as opposed to overflow)

(b) ability to distinguish random loss from buffer loss

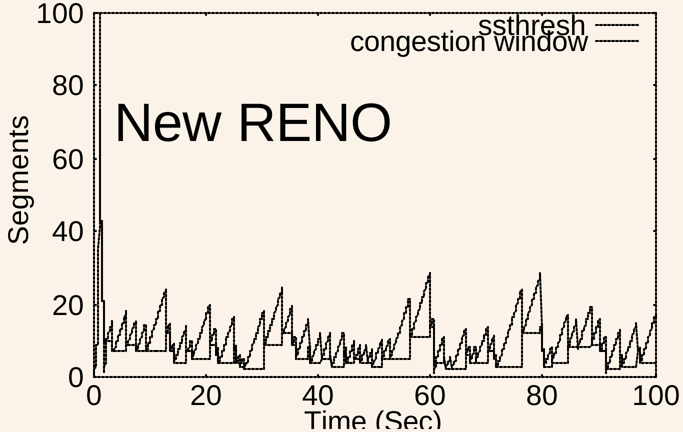
(c) using RE to **estimate** bottleneck bdw during slow start

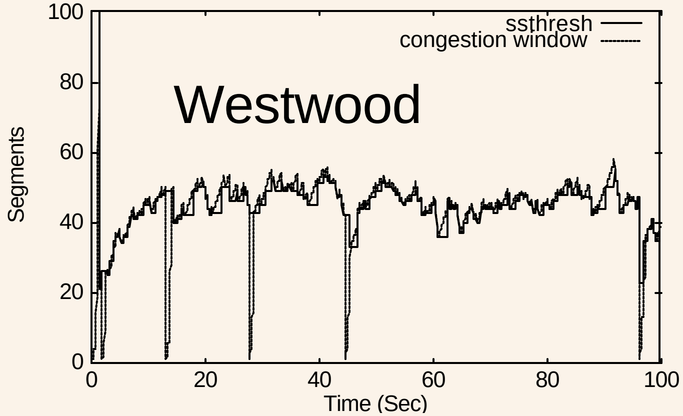
TCPW and random loss

- Reno overreacts to random loss (**cwin** cut by half)
- TCPW less sensitive to random loss
- a small fraction of "randomly" lost packets minimally impacts the rate estimate RE
- Thus, **cwin** = RE x RTT remains unchanged
- As a result, TCPW throughput is higher than Reno and SACK

TCPW And Random Loss

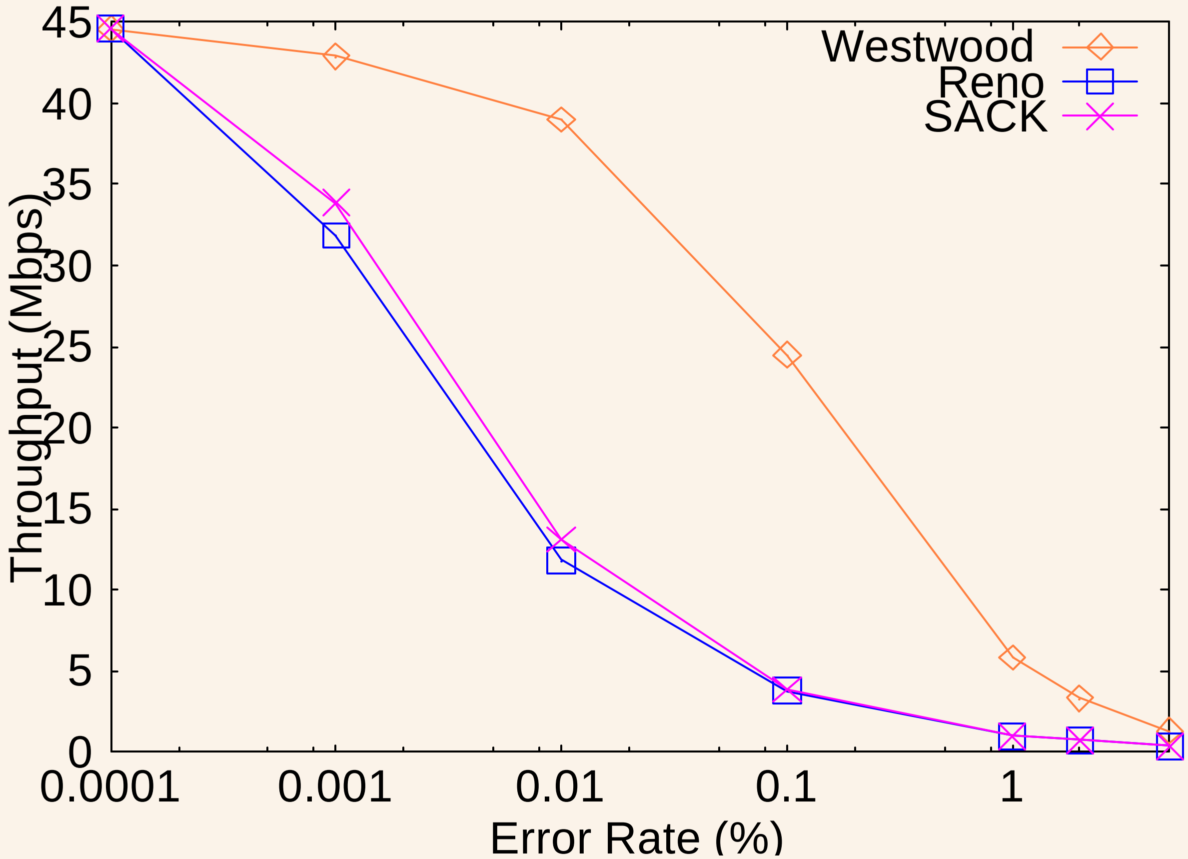
Cwnd and ssthresh of TCPW and NewReno under random losses:





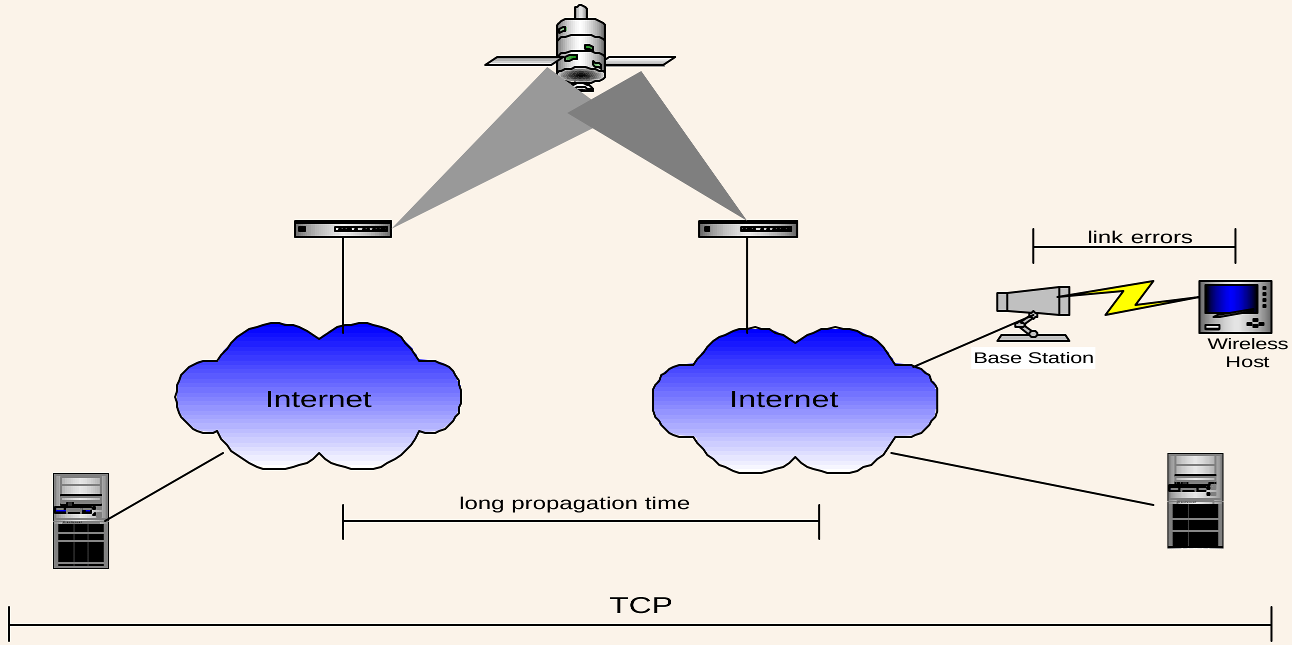
- NewReno overreacts to random loss (cwin cut by half)
- A small fraction of isolated "randomly" lost packets does not impact the RE estimate
- Thus, *cwnd = RE * RTTmin* remains unchanged
- As a result, TCPW efficiency is higher than NewReno and SACK

TCPW in “lossy” environment



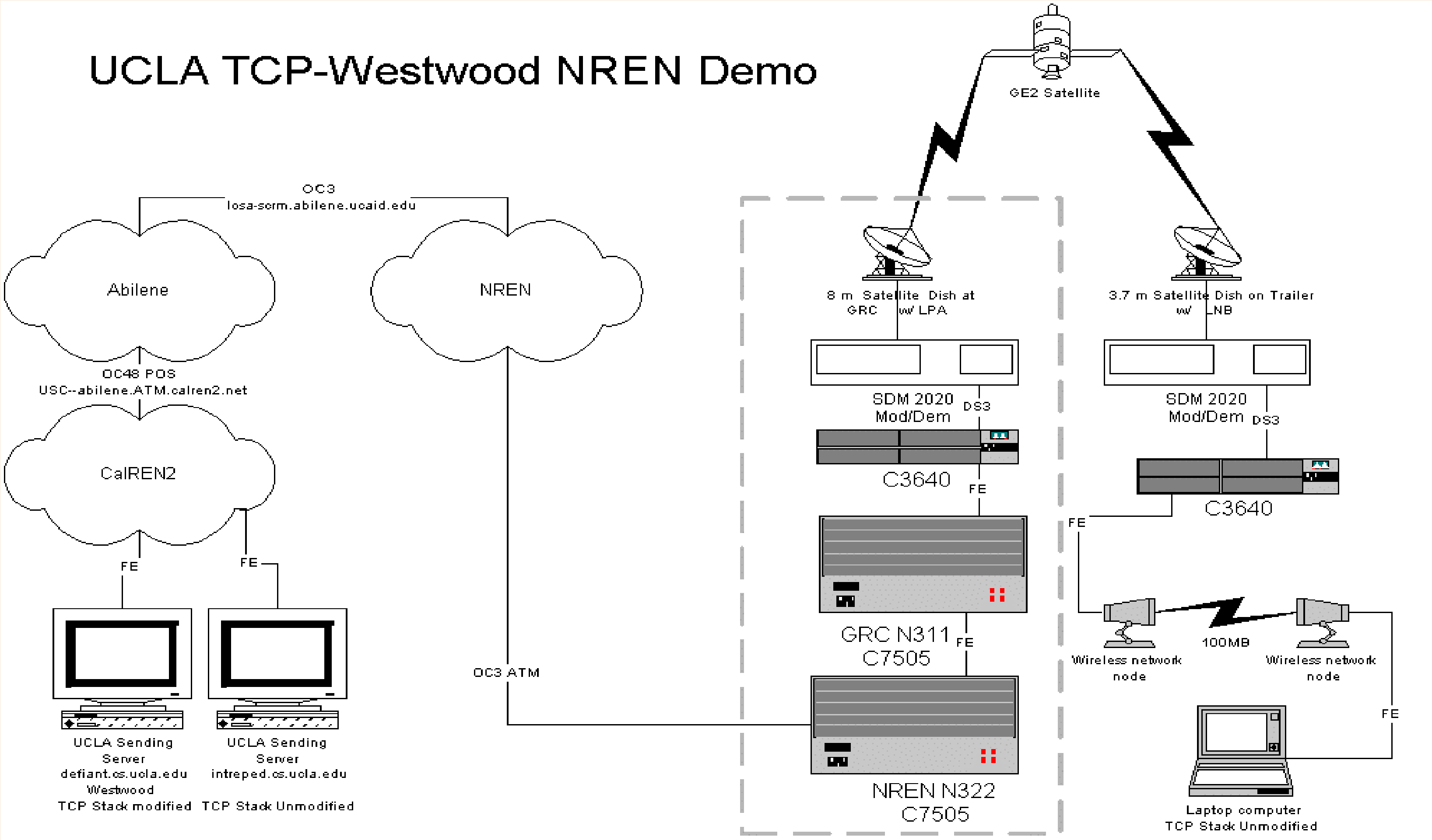
TCPW in a wireless lossy environment

- *Efficiency*: Improvement significant on high (Bdw x Length) paths

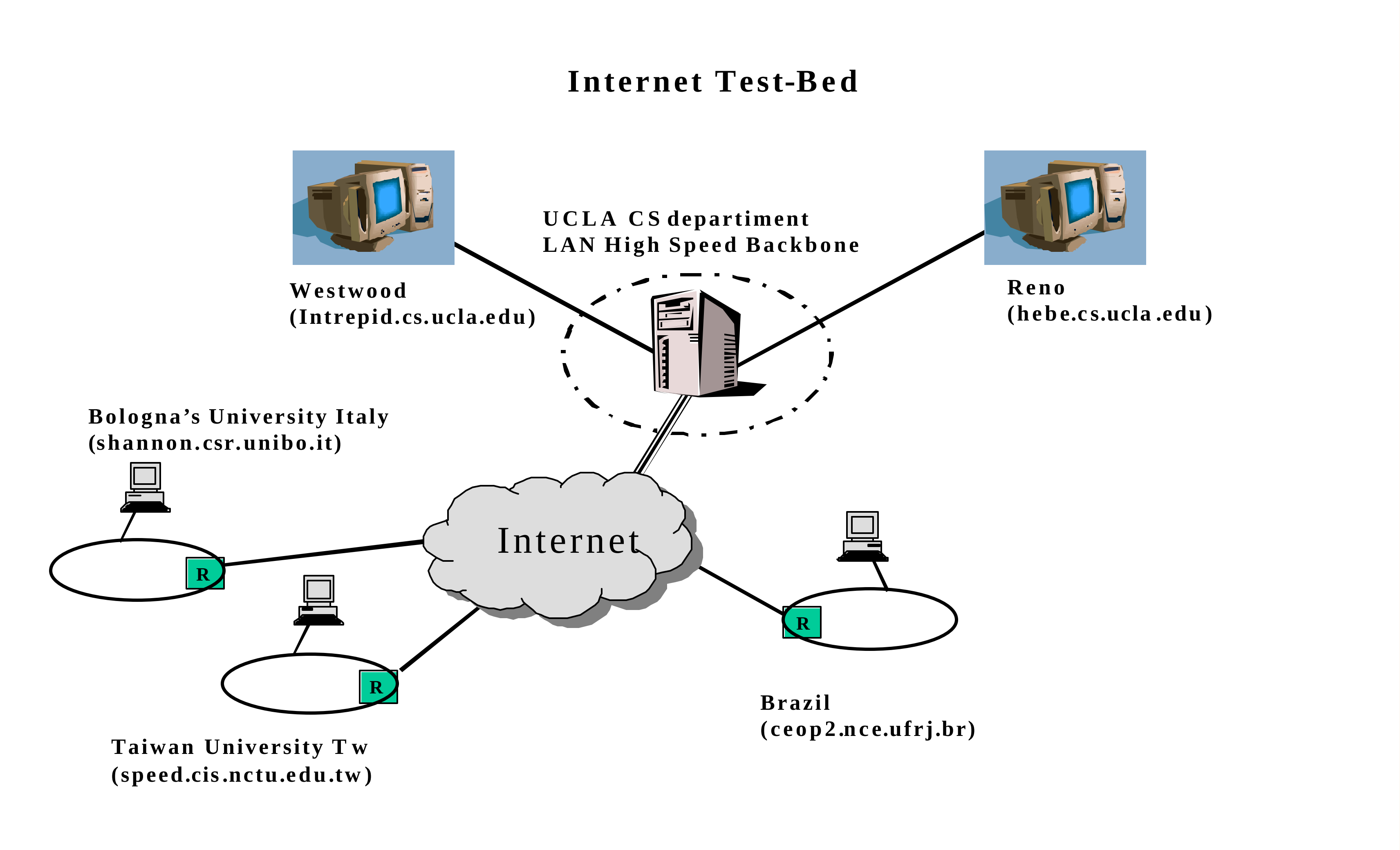


- *Fairness*: better fairness than RENO under varying RTT
- *Friendliness*: TCPW is friendly to TCP Reno

NASA Workshop Demo
(From Steve Schultz, NASA)

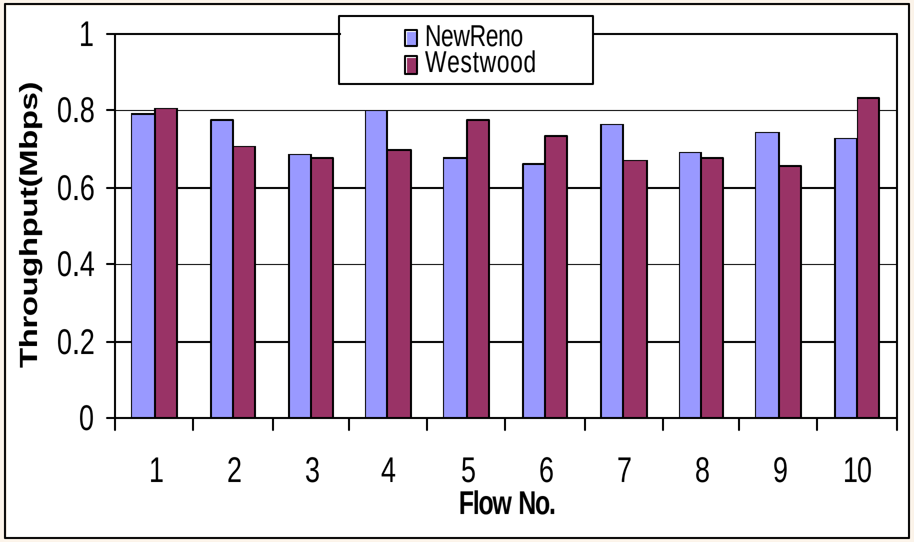


Internet Measurements Testbed



TCPW Fairness

- **Fairness**: how equitably is bandwidth shared among same flavor TCP flows?
  - Internet experiment with 10 TCPW and 10 TCP NR
  - Jain's index for this experiment is ???

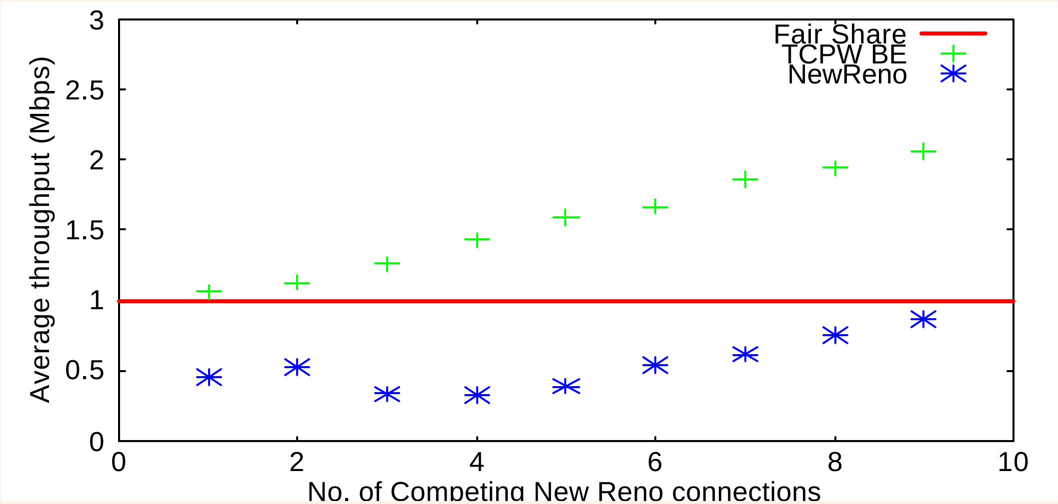


TCPW Friendliness

- **Friendliness**: fairness across different TCP flavors

  "Friendly share" principle: *TCPW is allowed to recover the bandwidth wasted by NewReno because of "blind" window reduction*

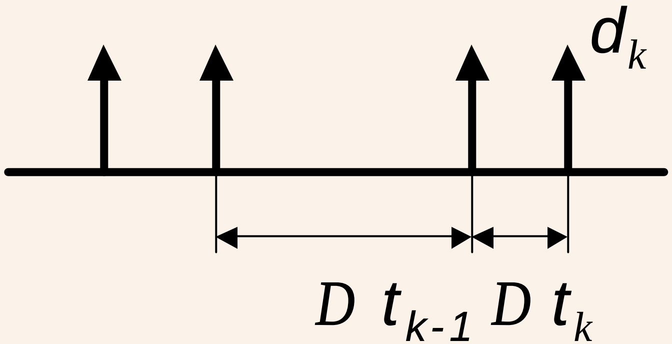
- TCPW **original RE filter** has Friendliness Problem….
  - 10 connections total (TCPW + RENO) ; No random errors
  - Average throughput per connection is shown below:



Outline

1. TCP Overview
2. TCP Westwood and Bandwidth Estimation
3. Bandwidth & Rate Estimation; Adaptive Filter
4. Performance Evaluation

TCPW original estimation (BE)



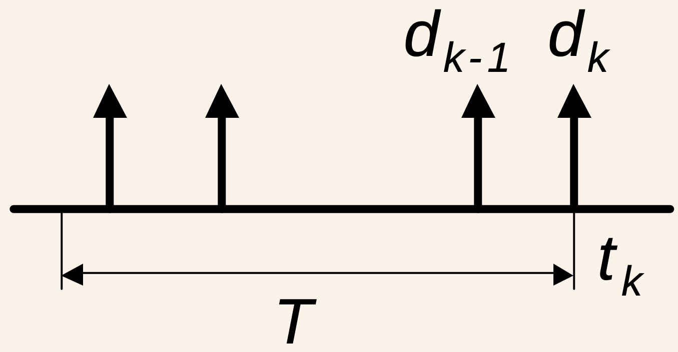
- First TCPW version (referred to as: TCPW BE) used a “bandwidth like” estimator (BE) given by:

$$b_k = d_k / (t_k - t_{k-1})$$ sample

$$BE_k = a_k BE_{k-1} + (1 - a_k)\left(\frac{b_k + b_{k-1}}{2}\right)$$ exponential filter

$$a_k = \frac{2t - \Delta t_k}{2t + \Delta t_k}$$ filter gain

TCPW Rate Estimation (TCP RE)



T is the sample interval

- Rate estimate (RE) is obtained by aggregating the data ACKed during the interval T (typically = RTT):

$$b_k = \frac{\sum_{t_j > t_k - T} d_j}{T}$$ sample

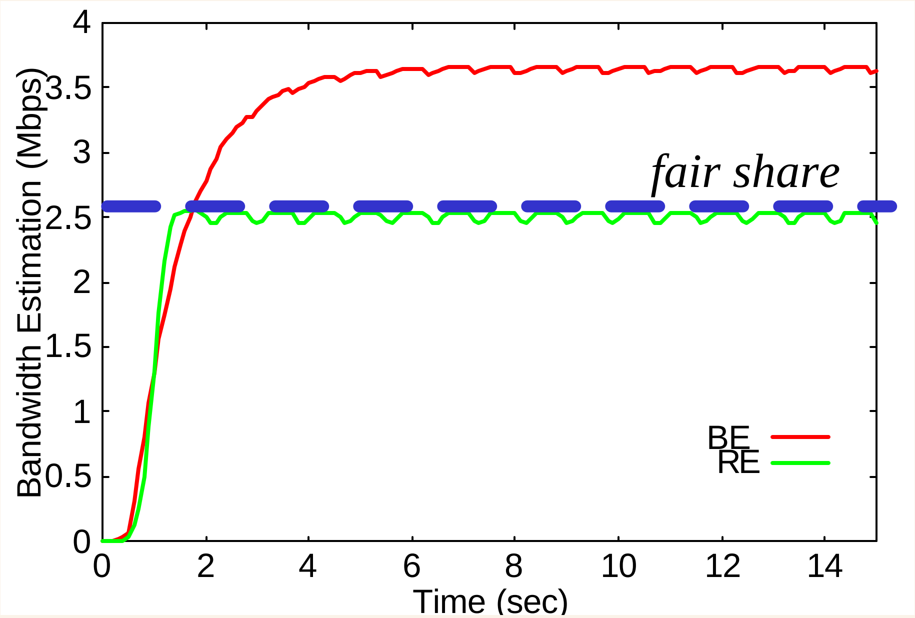
$$RE_k = a_k RE_{k-1} + (1 - a_k)\left(\frac{b_k + b_{k-1}}{2}\right)$$ exponential filter

$$a_k = \frac{2t - \Delta t_k}{2t + \Delta t_k}$$ filter gain

TCPW RE/BE interaction with RENO

One TCPW RE/BE and one Reno share a 5Mbps bottleneck

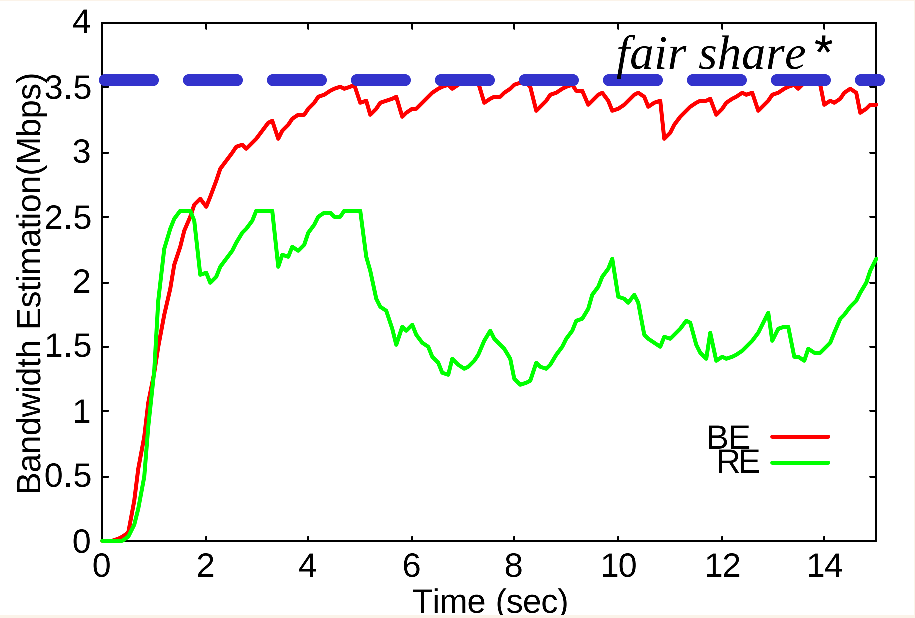
No errors (bottleneck gets saturated)



BE overestimates fair rate (= 2.5 Mbps)

TCPW BE Not friendly to NewReno!

Errors (0.5%), no congestion



RE underestimates fair rate (=3.6 Mbps)

TCPW RE does not improve thruput!

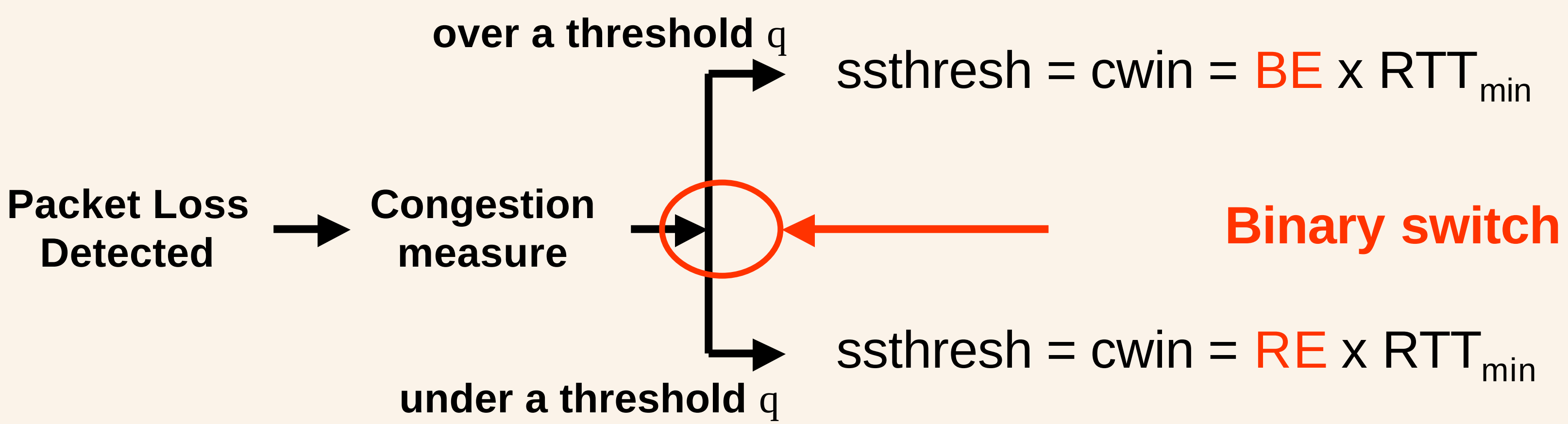
(*) TCPW fair share > 50% because NewReno is incapable of getting 50%

TCPW Adaptation

- Neither RE or BE estimator are optimal for all situations
  - BE is more effective in random loss
  - RE is more appropriate in congestion loss (ie, buffer overflow)
- KEY IDEA: dynamically select the aggressive estimate (BE) or the conservative estimate (RE) depending on current channel status (congestion or random loss?)
- NEEDED: a "congestion measure" that gives us an idea of the most probable cause of packet loss (congestion or random)

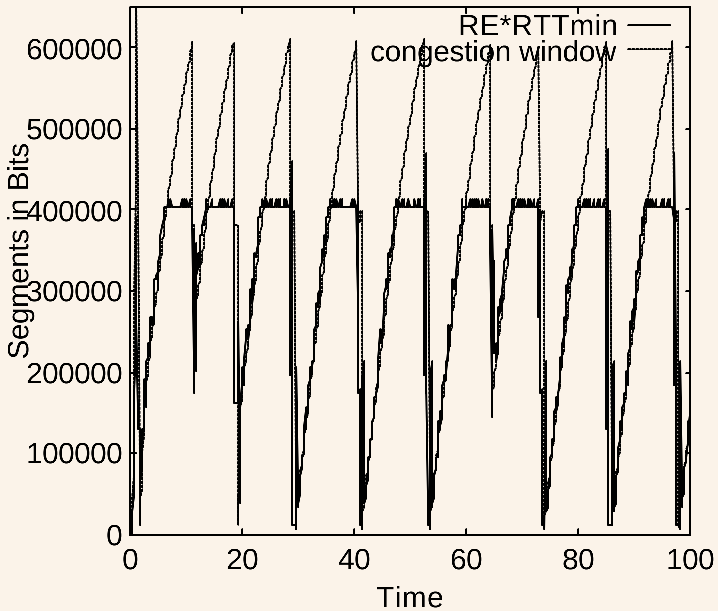
Combining Rate and Bandwidth estimations: TCPW CRB

- **TCPW CRB** chooses between RE or BE upon packet loss to set the ssthresh

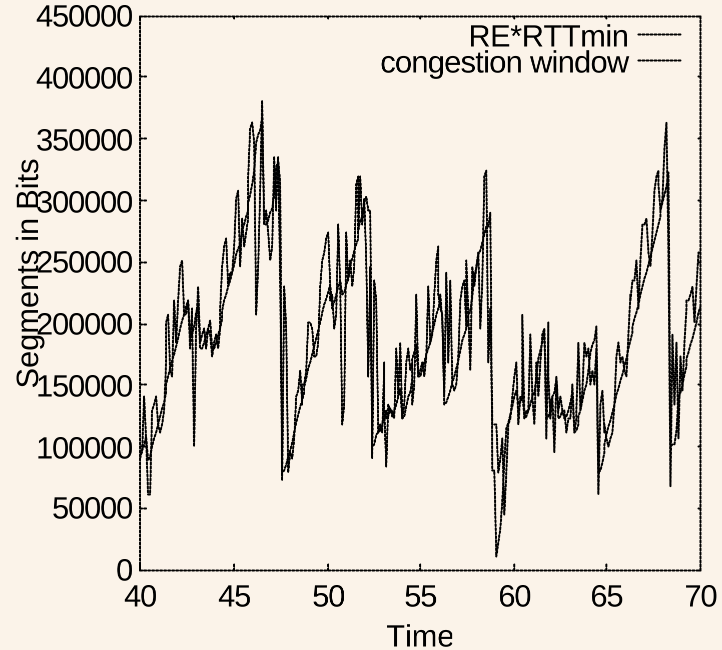


Congestion Measure

- IF cwnd/ RTTmin (ie, **max** achievable rate) is larger than vs. RE (**currently** achieved rate) the channel is congested;
- if **max** is equal to **current** rate, the loss is random loss



Congested Channel



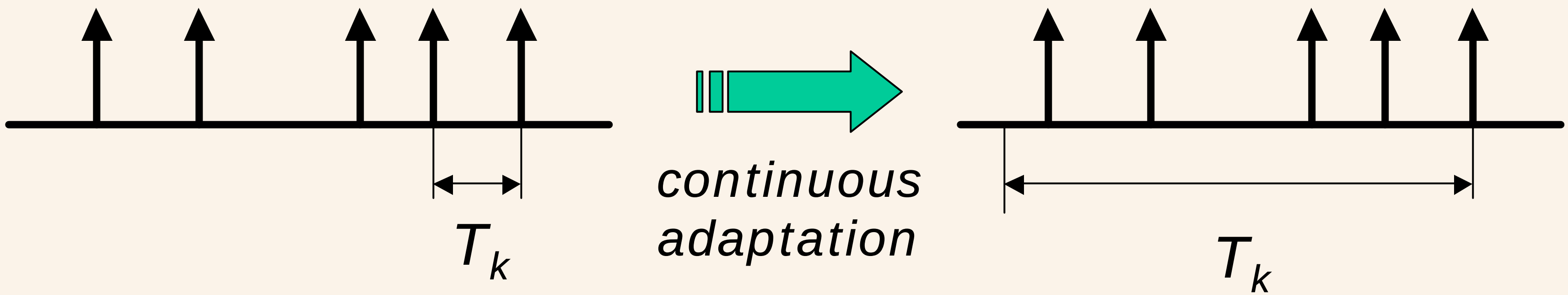
Random loss channel

TCPW with continuous filter adaptation

- Next step is to have a *continuous* instead of *switched* filter adaptation
- IDEA:
  - adapt continuously the sample size according to congestion level
  - adapt continuously the filter agility according to network instability
- In TCPW AF (Adaptive Filtering) we adapt the sample interval T_k according to current measured congestion level
  - T_k ranges from T_k = *inter ACK interval* to T_k = *RTT*
- Filter agility (more or less weight on history) must be limited so that it does not overreact to network jitter

TCPW AF: Sampling

- Adapting the size of sampling intervals to congestion level measure



No Congestion: T_k *= inter ACK*   *Congestion:* T_k *grows*

$$s_k = \frac{\sum_{t_j > t_k - T_k} d_j}{T_k}$$

Rate sample

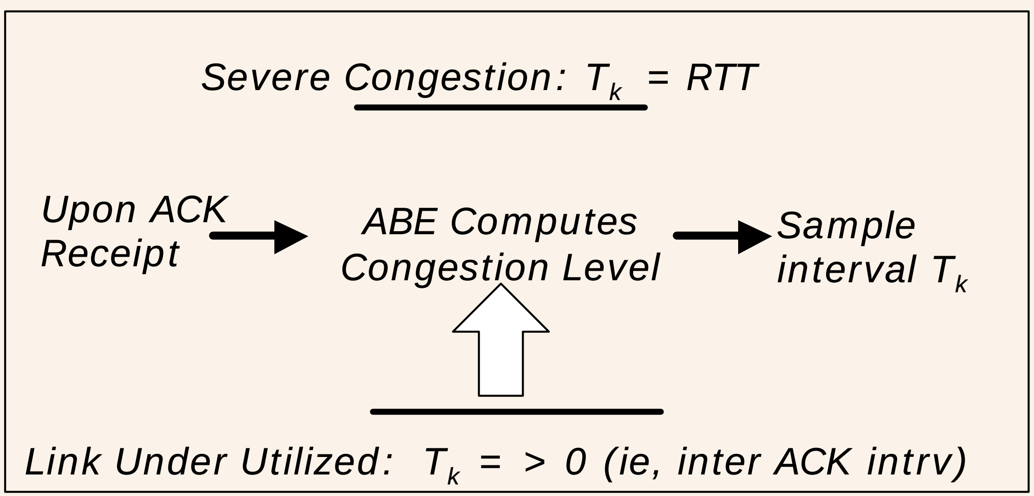
TCPW AF: Sampling (cont)

- The sample size T_k is continuously adjusted according to current congestion measure:

$$T_k = RTT * \left(\frac{cwin}{RTT_{min}} - \hat{Th}_k\right) / \frac{cwin}{RTT_{min}}$$

Max throughput assuming there is no congestion in the network

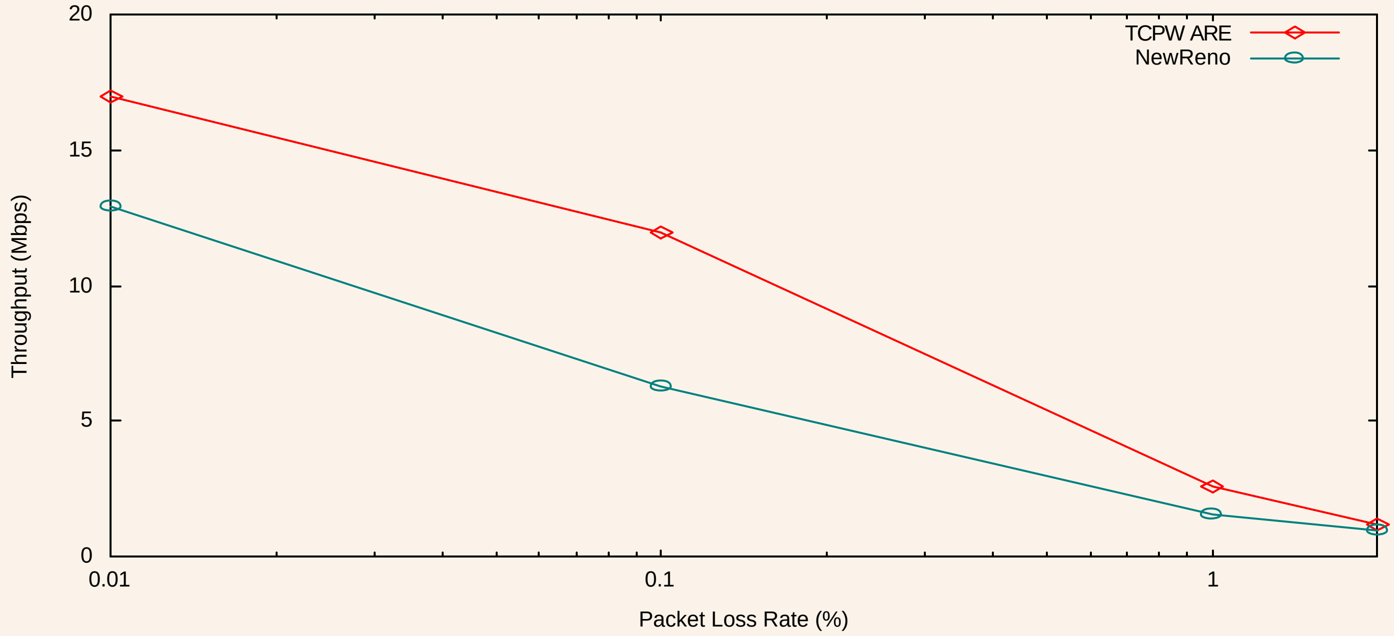
actual achieved throughput



Outline

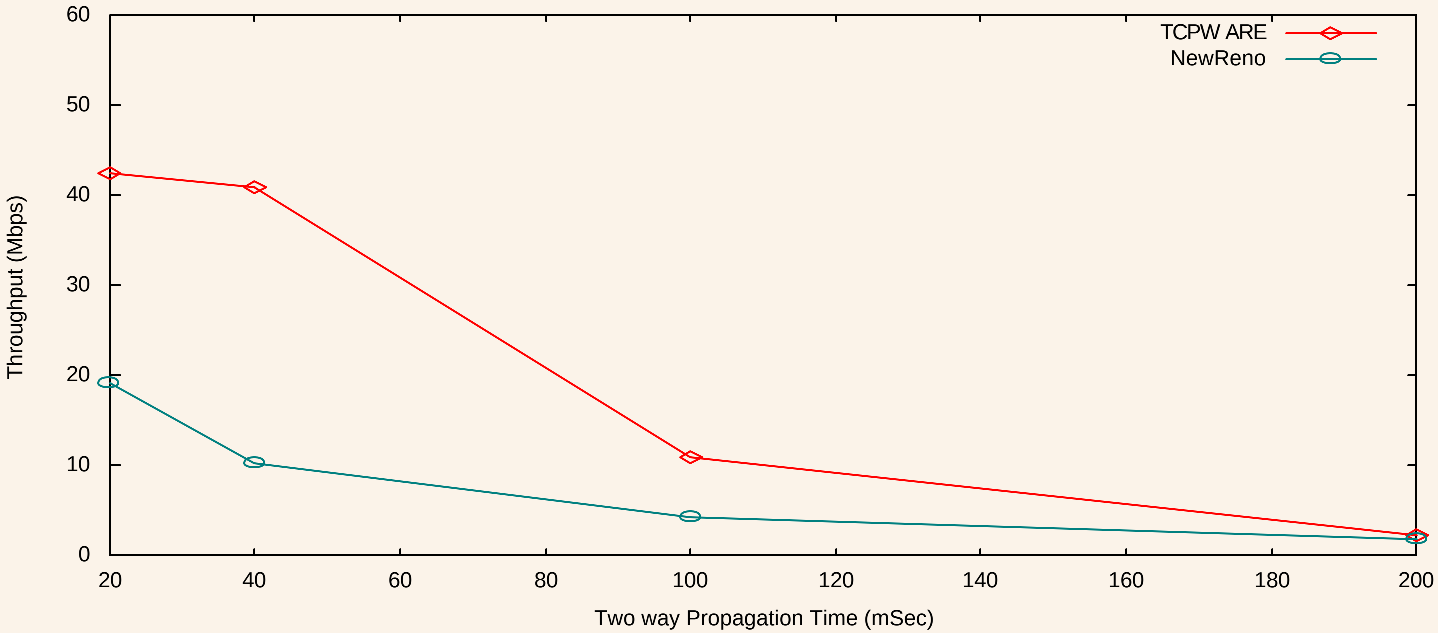
1. TCP Overview
2. TCP Westwood and Bandwidth Estimation
3. Bandwidth & Rate Estimation; Adaptive Filter
4. Performance Evaluation

TCPW AF Simulation Results (1)



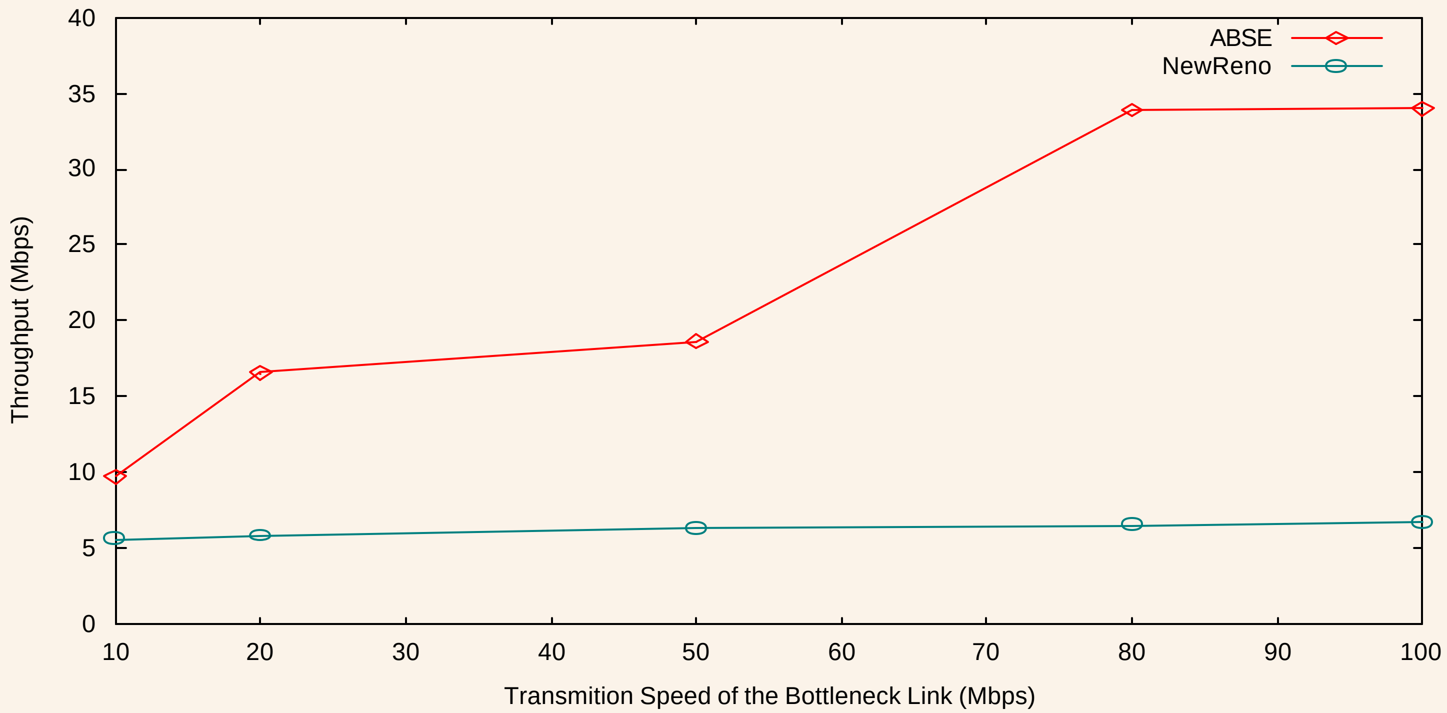
Throughput vs. packet loss rate

TCPW AF Simulation Results (2)



Throughput vs. Two-way Propagation Time
Is there loss?

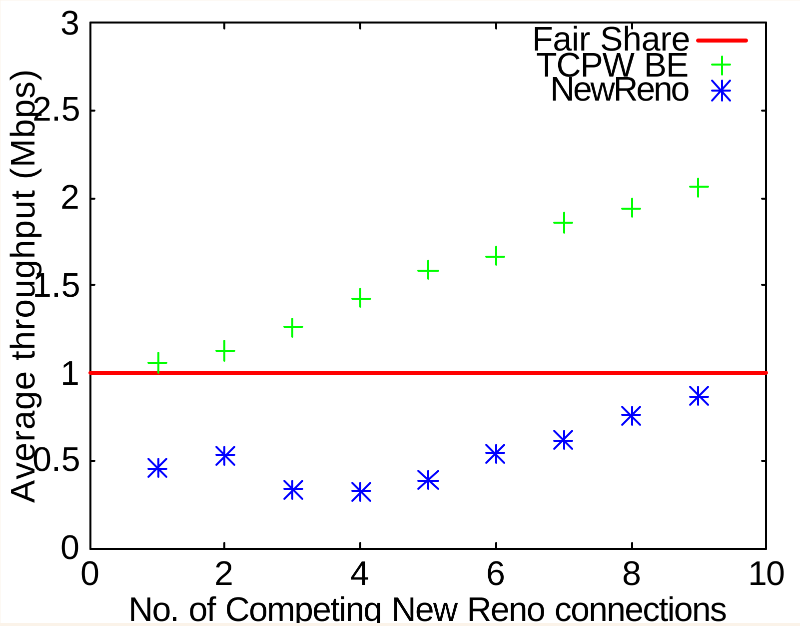
TCPW AF Simulation Results (3)

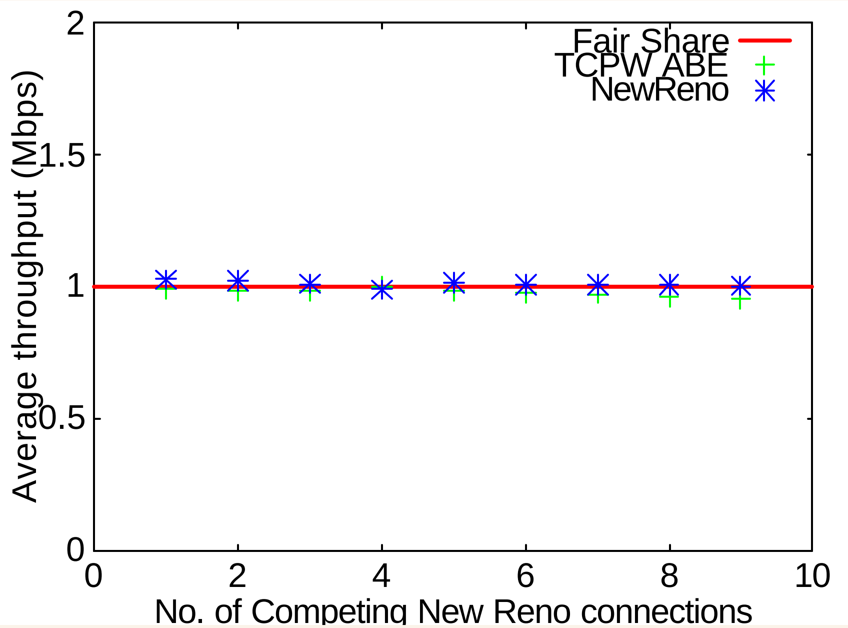


Throughput vs. Bottleneck Capacity

Is there loss?

TCPW AF Simulation results (4)





TCPW AF Is Friendly Towards NewReno !

Summary

- Introduced the concept of Rate Estimation and related work
- Reviewed end-to-end estimation based congestion control methods
- Presented TCP Westwood, and the evolution of "fair rate" estimate to improve the performance; showed simulation results to evaluate the method
- Compared TCPW with other methods

References

- The papers about TCP Westwood, TCP Westwood CRB and ABE can be found in the papers section of the TCP Westwood Web Page: http://www.cs.ucla.edu/NRL/hpi/tcpw/
- TCP Vegas: New Techniques for Congestion Detection and Avoidance. Lawrence Brakmo, Sean O'Malley, and Larry Peterson. In *ACM SIGCOMM*, pages 24-35, August 1994
- I. Akyildiz, G. Morabito, and S. Palazo. TCP-Peach: A new Congestion Control Scheme for Satellite IP Networks. IEEE/ACM Transaction on Networking, vol. 6, pp 307-21, 2001.
- S. Keshav “A Control-Theoretic Approach to Flow Control,” Proceeding of ACM SIGCOMM 1991
- K. Lai and M. Baker, “Measuring Link Bandiwdths Using a Deterministic Model of Packet Delay”, Sigcomm 2000
- M. Allman and Vern Paxson, “On Estimating End-to-End Network Path Properties”, ACM/Sigcomm 1999
- R. Carter and M. Crovella, “Measuring Bottleneck Link Speed in Packet-Switched Networks” Performance Evaluation, Vol 27,28, 1996
- **Dovrolis, Ramanathan, Moore, “What Do Packet Dispersion Techniques Measure?”, Infocom 2001.**