TCPW: Sender-Side-Only Handling of Leaky Large Dynamic Pipes

M. Y. "Medy" Sanadidi

http://www.cs.ucla.edu/~medy
http://www.cs.ucla.edu/NRL/hpi/tcpw/

Acknowledgment

This presentation includes research material produced at the UCLA Network Research Laboratory under the inspiring leadership of ***Professor Mario Gerla***

The research results were contributed to by the talented researchers: ***Claudio Casetti, Mario Gerla, Scott Lee, Saverio Mascolo, Giovanni Pau, Bryan Kwok-Fai Ng, Paulo Rodrigues, Hideyuki Simonisi, Massimo Valla, Ren Wang, Kenshin Yamada***

This presentation was prepared with the intelligent and generous assistance of ***Ren Wang***

Outline

- Problem Definition
- TCPW Approach
- TCPW Algorithms and Performance
- Alternative Approaches
- Summary
- On-going work
- References

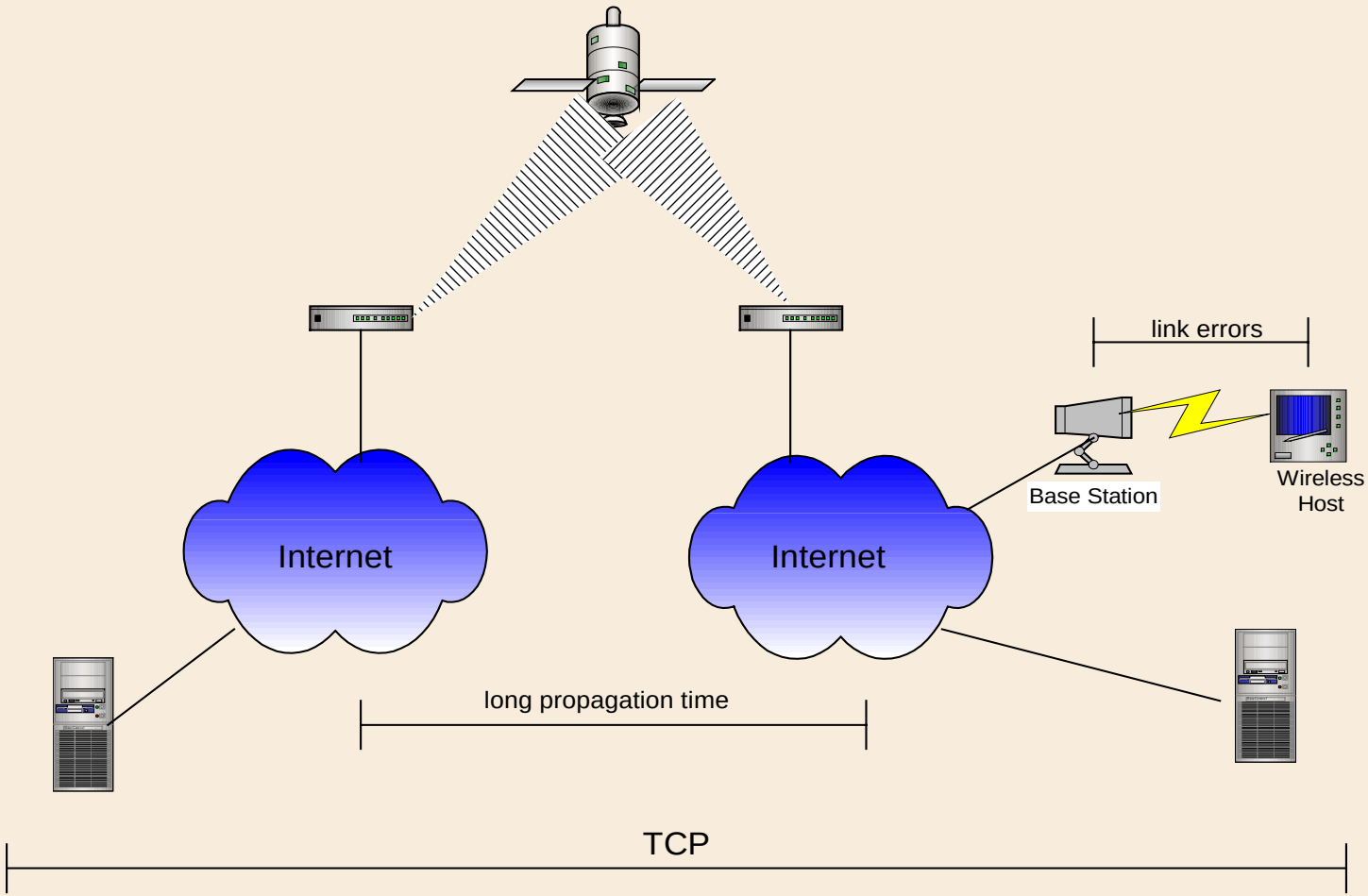
Outline

- **Problem Definition:** leaky large dynamic pipes
- **TCPW Approach:**
  - Sender-side-only; mining packet streams
  - Uses dispersion based bandwidth measures estimation techniques
- **TCPW Algorithms and Performance**
  - **Eligible Rate Estimation (ERE):**
    - Bandwidth Estimation (BE): efficiency gains, and friendliness problem
    - Rate Estimation (RE): friendliness gains, weakness under error loss
    - Efficiency/Friendliness Tradeoff
    - CRB and ABSE
  - **Agile Probe (Aprobe):** initially for solving initial ssthresh setting, but later generalized
  - **Persistent Non-Congestion Detection (PNCD):** for efficiency in dynamic large pipes
- **Alternative Approaches**: Snoop, XCP, FAST , S-TCP, High-Speed TCP
- **Summary:** Adaptive bandwidth estimation method produce effective ERE, leading to better efficiency/fairness/friendliness; and provide ability to control efficiency/friendliness tradeoff
- **Ongoing work:** control model analysis, measurements, use in transport service differentiation, comparison to alternatives, more dispersion techniques, video streaming
- **References**

Problem Definition

- **Leaky** Pipes: packet loss due to error
  - Unjustified cwnd cut
  - Premature exit to Congestion Avoidance during Slow Start
- **Large** Pipes: large capacity and long propagation time
  - Control schemes may not scale
- **Dynamic** Pipes: dynamic load
  - Linear increase in Congestion Avoidance limits efficiency

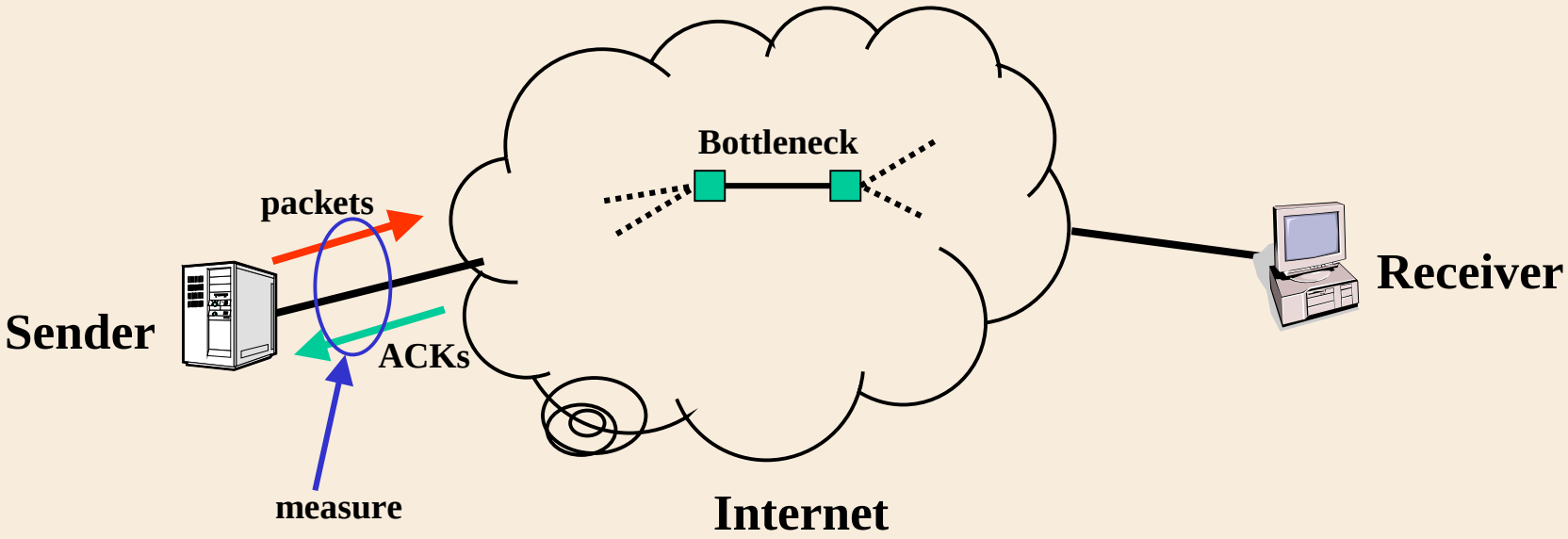
Example: Satellite/802.11 Networks



TCPW Approach

- Sender-side-only
- Sender uses bandwidth measures estimation techniques to determine *Eligible Rate Estimate* (ERE)
- Bandwidth samples are derived from ACK *arrival statistics*, and info in ACKs regarding *bytes delivered*
- ERE is used by sender to
  - (1) set *cwnd* and *ssthresh* after packet loss indication
  - *(2)* guide *"Agile Probing"*, a proposed more agile probing technique to be used instead of Slow Start and elsewhere

Mining Packet Streams

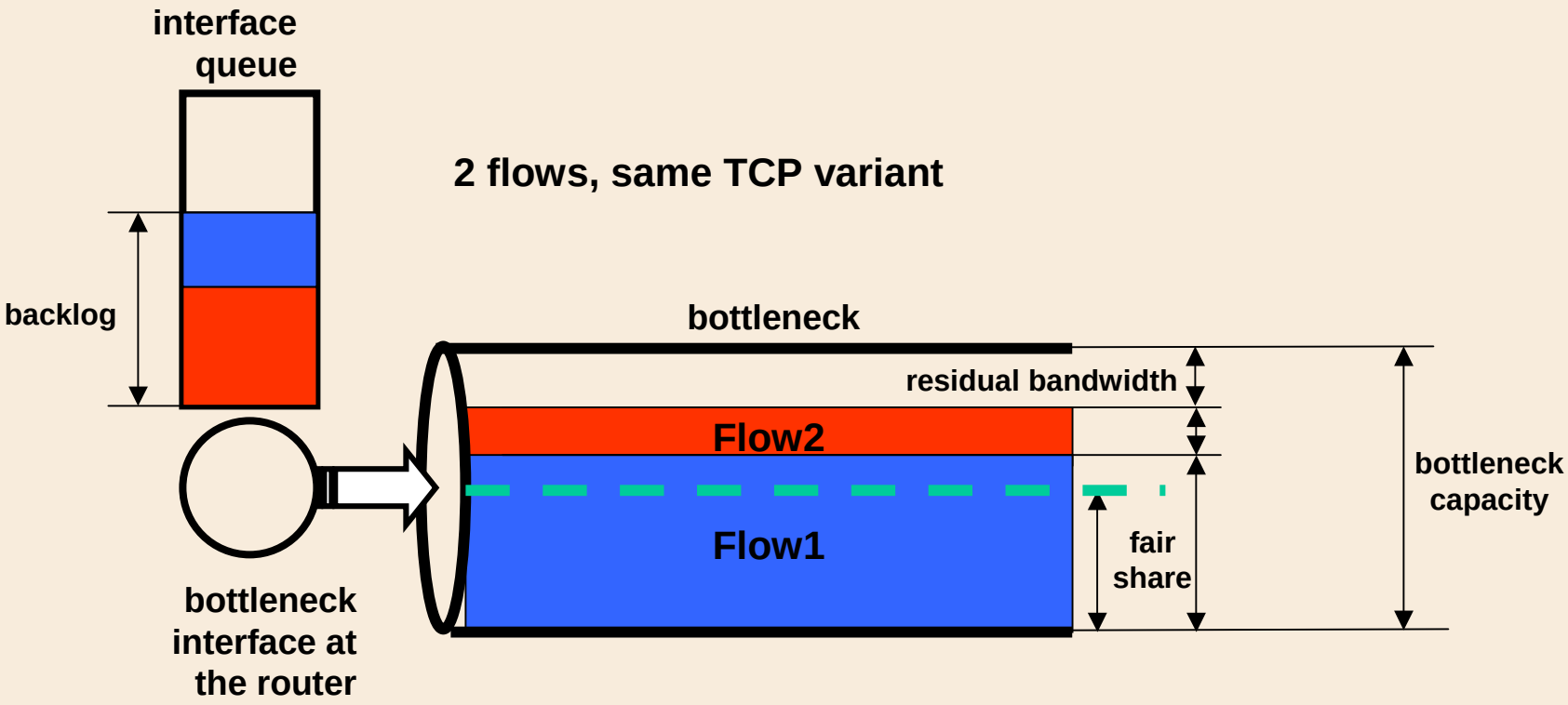


- Ideally, would like to determine the connection *fair share* of the network bandwidth
- Since *fair share* is difficult to define or determine, we instead determine an *Eligible Rate Estimate (ERE)*

Definitions

- **Capacity:** Physical bandwidth of a link
- **Bottleneck Link**: Link with minimum capacity on the path
- **Residual Bandwidth (or Available bandwidth):** Spare (or unused) bandwidth on a link
- **Connection Service Rate**: service rate provided at the bottleneck router to a connection, might also be called ‘bandwidth share’
- **Fair Share**: share of bandwidth according to some fairness criterion
- **Eligible Rate Estimate:** An estimate of the sending rate a connection is eligible for, that provides good efficiency, fairness and friendliness

Sharing a link



Background: Packet-Pair Flow Control (Keshav 1990)

- Sends packet pairs and measure their dispersion
- Exponentially average samples to determine *connection service rate*
- Use *connection service rate* to implement ‘Rate Flow Control’
- Works only if routers implement Round Robin (realized by Keshav)
- May suffer from packet “Compression” inaccuracy ?
- Attempts to set the backlog in router to some target => Vegas like co-existence problems ?

Packet Pair Dispersion

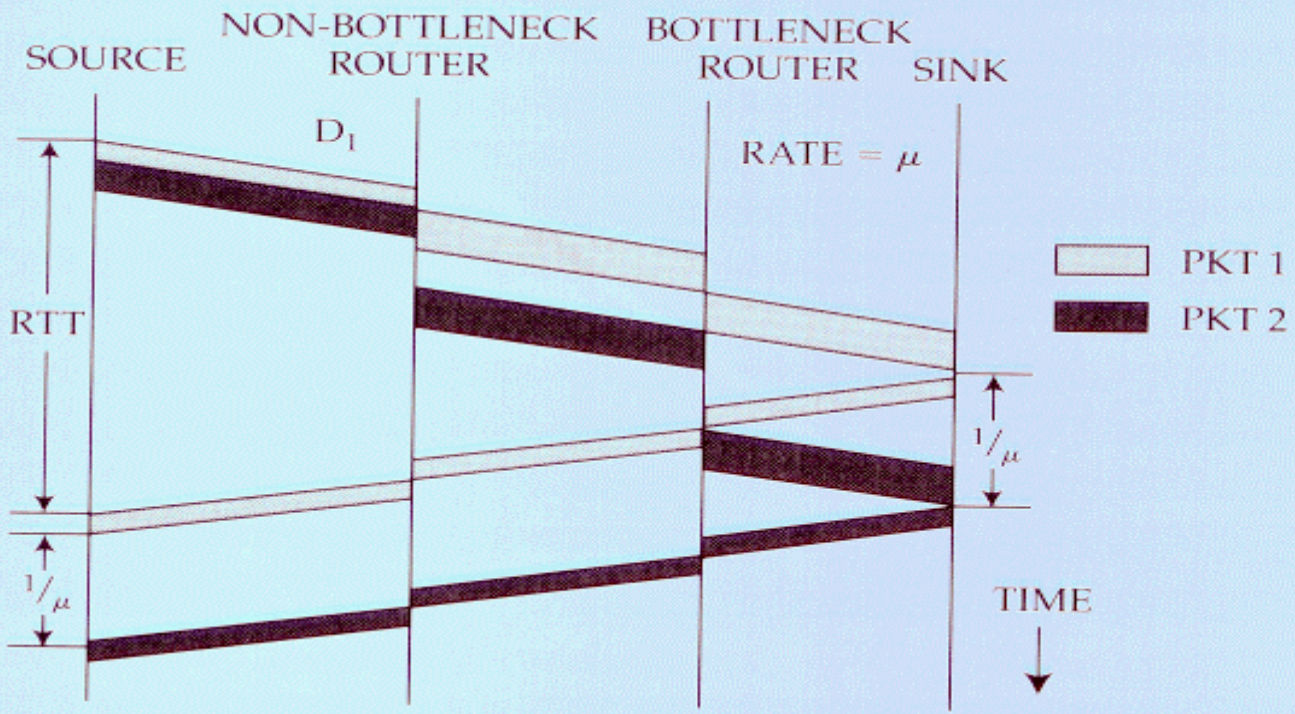
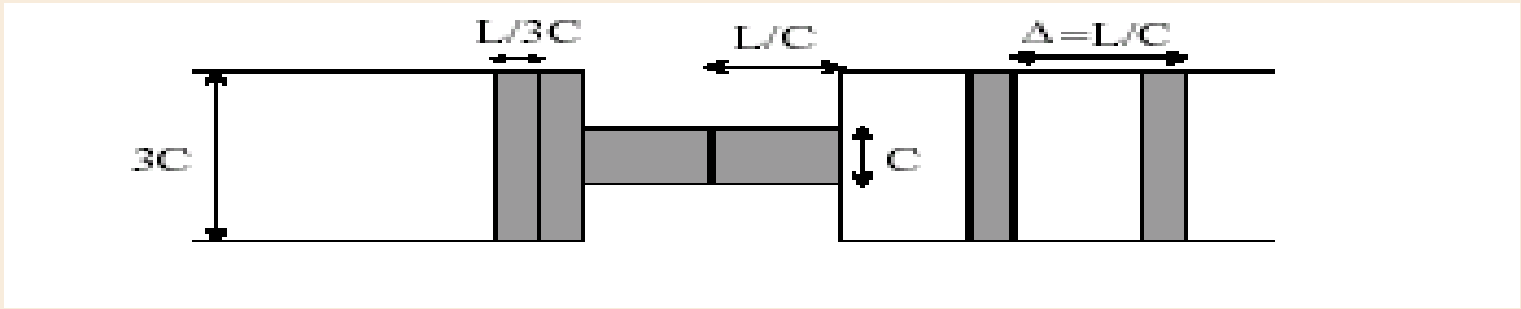


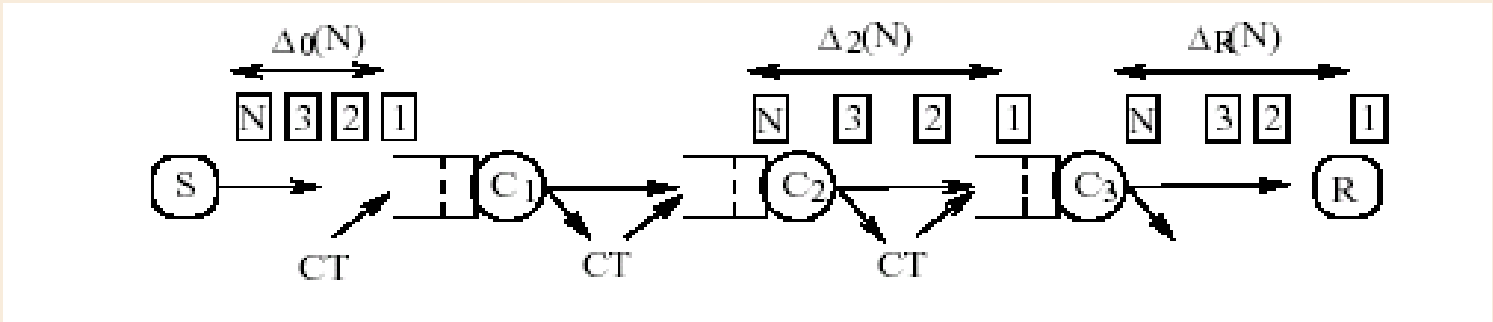
Figure 13.10: Packet pair. The source sends two packets back-to-back, and they are separated at every server because of cross traffic. If the servers obey a round-robin-like discipline, the largest separation of the pair measures the bottleneck service rate for this source. The packet separation is reflected in the inter-ack spacing, which can be measured at the source.

Pairs Dispersion => Trains Dispersion (Dovrolis 2001, Concurrent with TCPW)

- Generalize packet pairs to packet trains of increasing length
- Dispersion measurements do NOT directly provide estimates for Residual Bandwidth nor Capacity due to:
  - Cross traffic: expands packet pair/train dispersion
  - Post-bottleneck compression: reduces dispersion

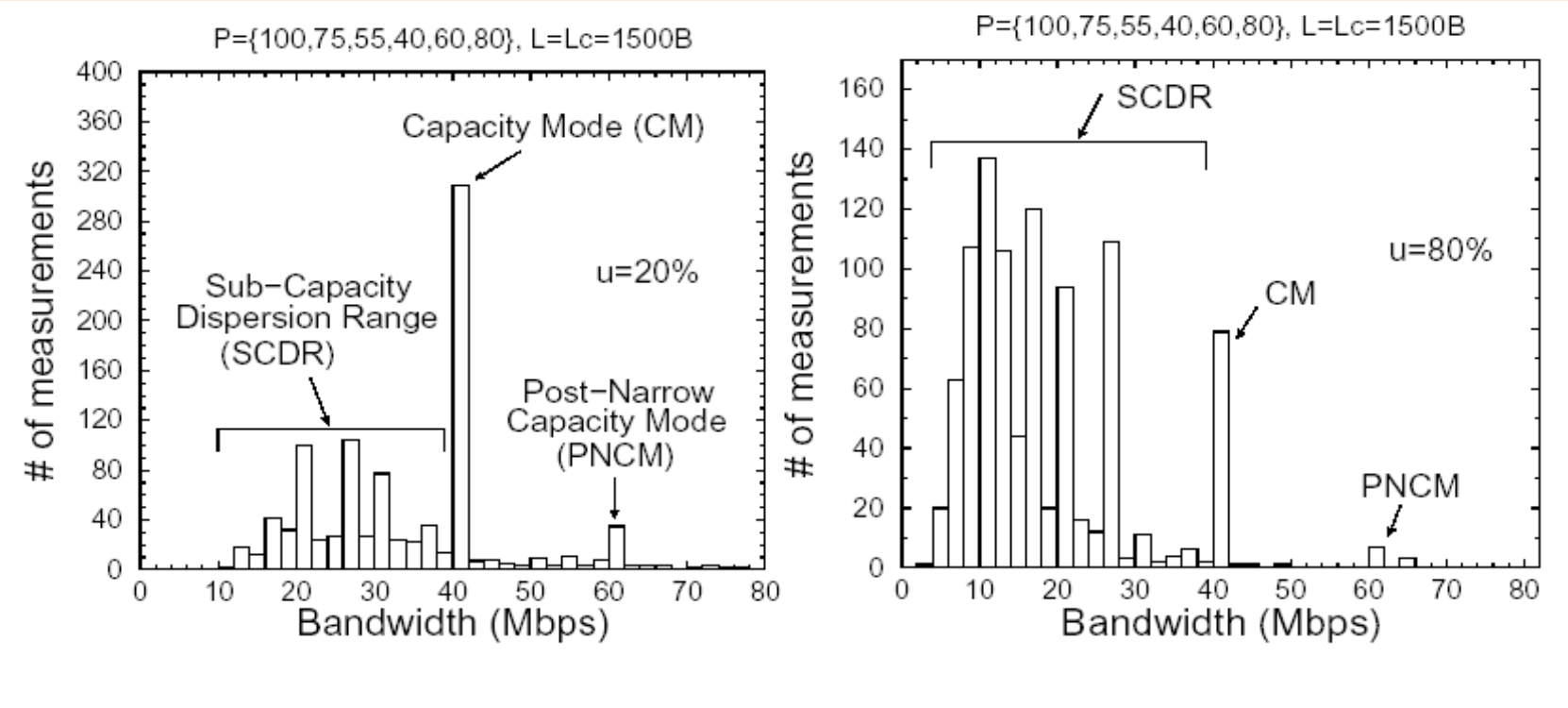
Packet Pair and Train Dispersion





Packet Pairs Bandwidth Histogram

- **Packet-pair** estimates: multimodality with cross traffic:

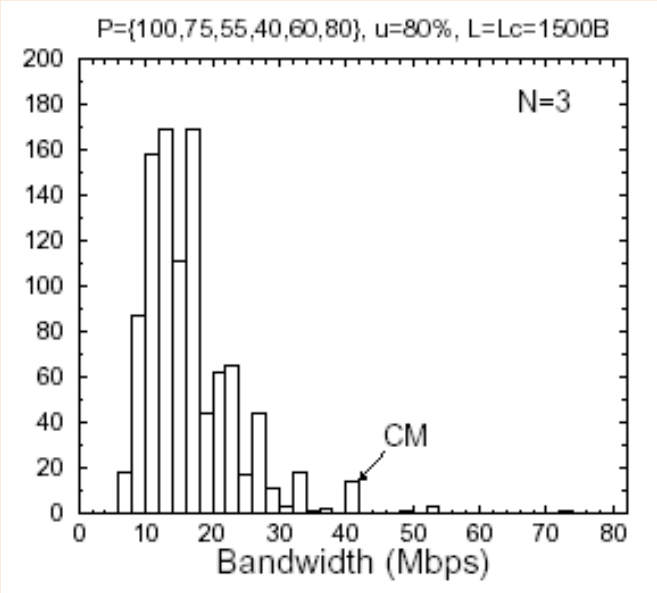


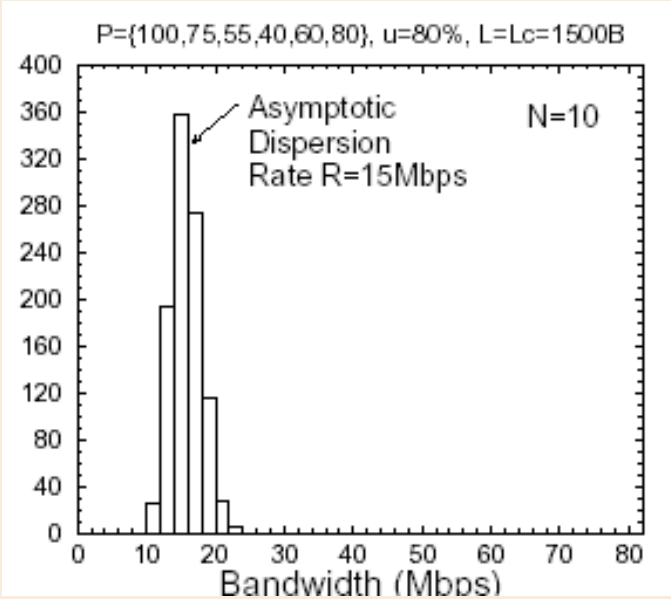
(a) Light load conditions(20%) (b) heavy load conditions(80%)

- SCDR is caused by cross traffic dispersion
- PNCM is caused by packet compression

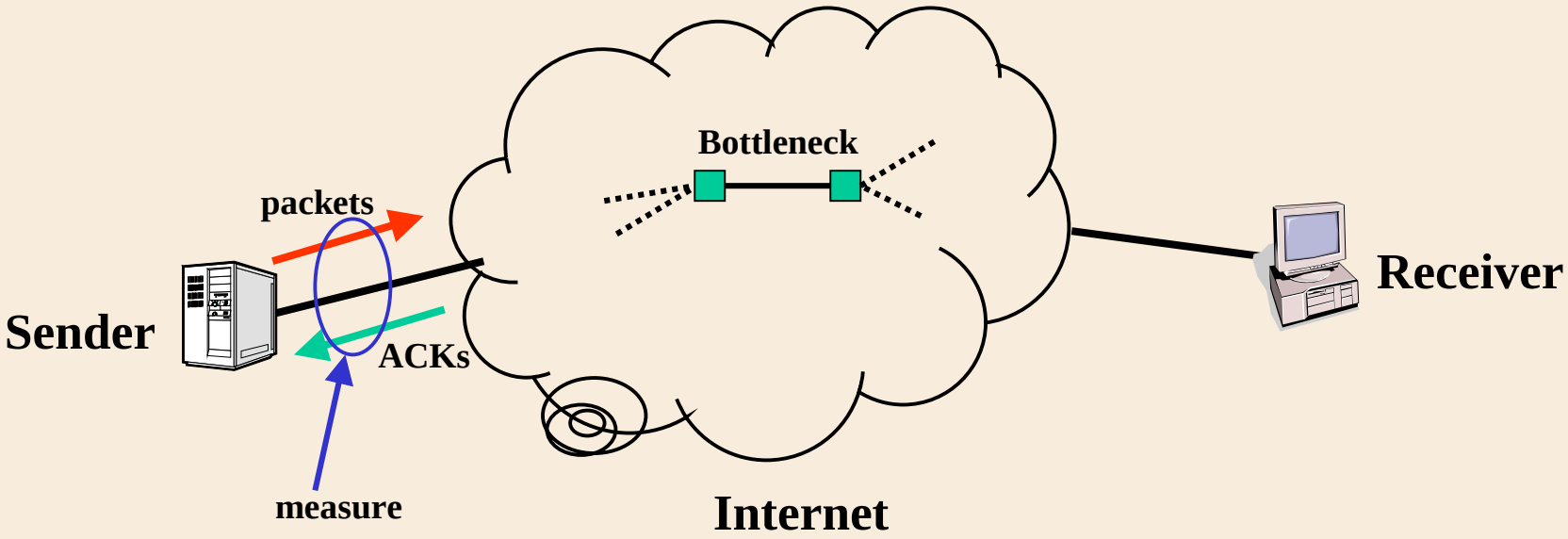
Packet Train Bandwidth Histogram

- As trains get longer, get Dovrolis' "Asymptotic Dispersion Rate" or ADR
- ADR is not equal to Residual (available) Bandwidth
- We found and proved a physical interpretation (to be published): ADR is the flow share proportional to merging links speeds
- All above is under "non-responsive traffic flows"





TCPW Approach



- Ideally, would like to determine the connection *fair share* of the network bandwidth
- Since *fair share* is difficult (to define or determine), we instead determine an *Eligible Rate Estimate (ERE)*, and use it in control algorithms as below

TCPW Control Algorithm

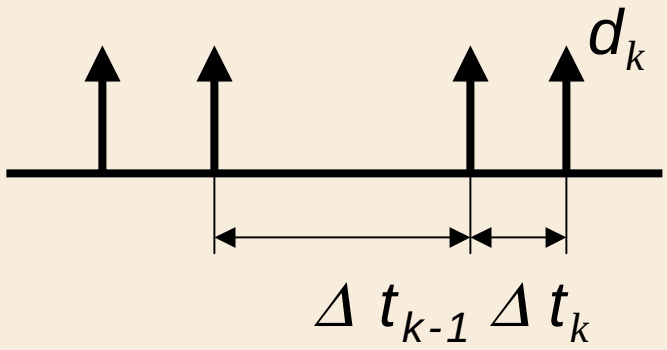
When three duplicate ACKs are received:

- set `ssthresh=ERE*RTTmin` (instead of `ssthresh=cwin/2` as in Reno and NewReno)
- if `(cwin > ssthresh)` set `cwin=ssthresh`

When a TIMEOUT expires:

- set `ssthresh=ERE*RTTmin` (instead of `ssthresh=cwnd/2` as in Reno) and
- set `cwin=1`

ERE Evolution: TCPW BE



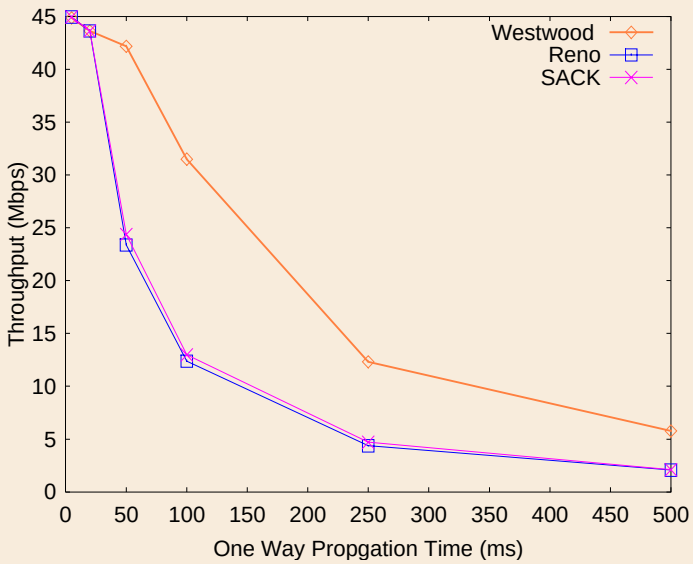
- First TCPW version (BE) used a Bandwidth Estimation (BE) given by:

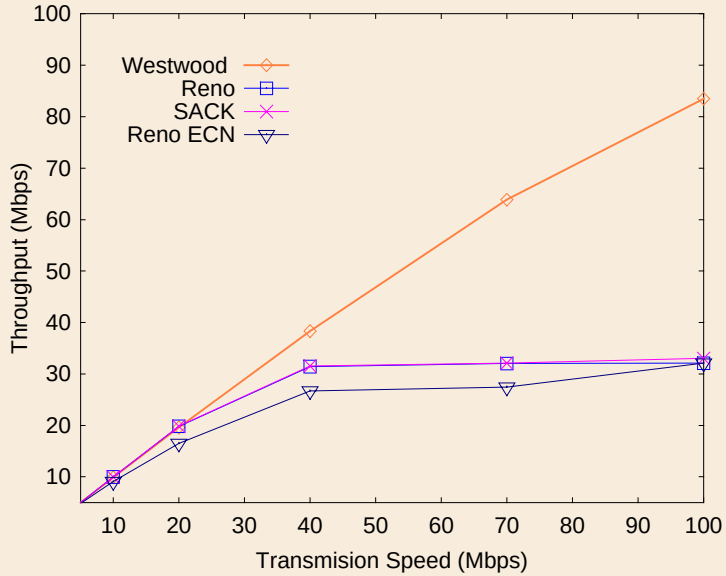
$$b_k = d_k / (t_k - t_{k-1})$$ sample

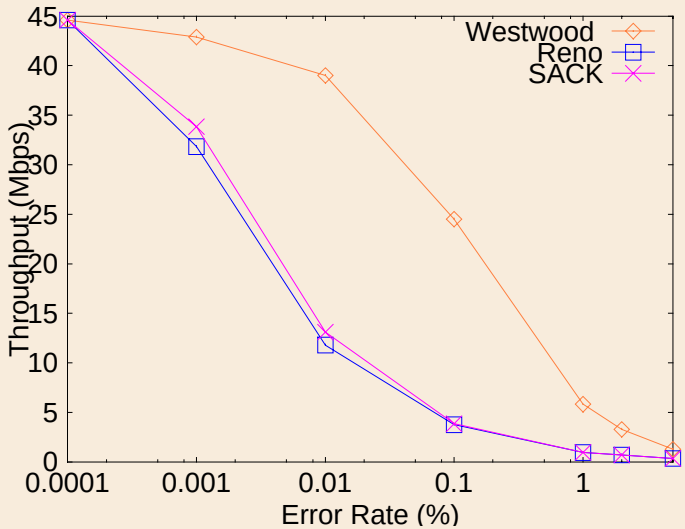
$$BE_k = \alpha_k BE_{k-1} + (1-\alpha_k)\left(\frac{b_k + b_{k-1}}{2}\right)$$ exponential filter

$$\alpha_k = \frac{2\tau - \Delta t_k}{2\tau + \Delta t_k}$$ filter gain

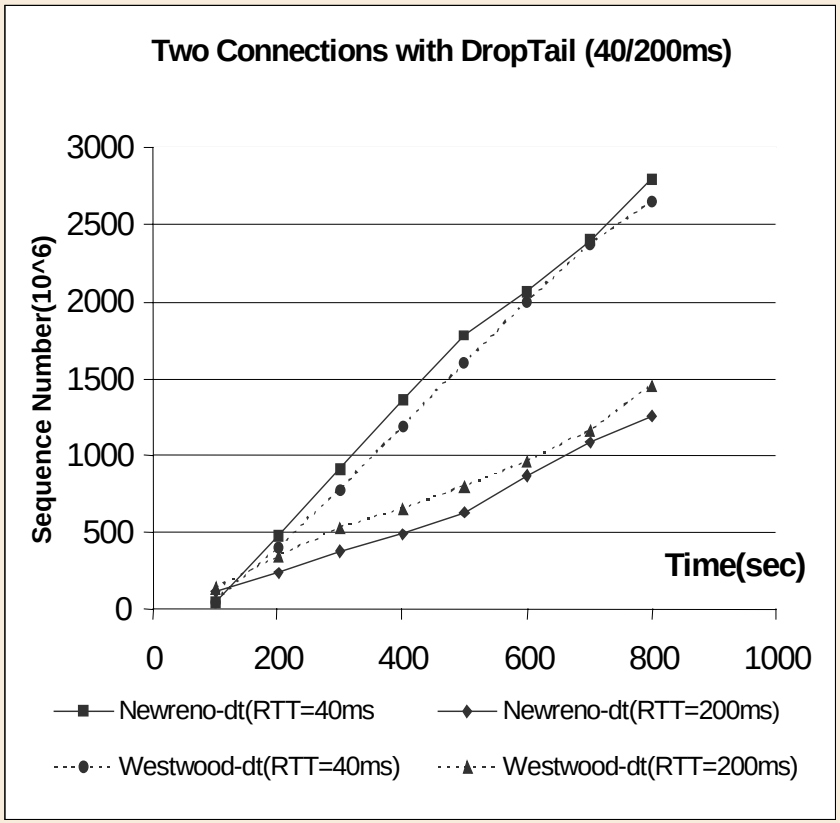
Efficiency Gains

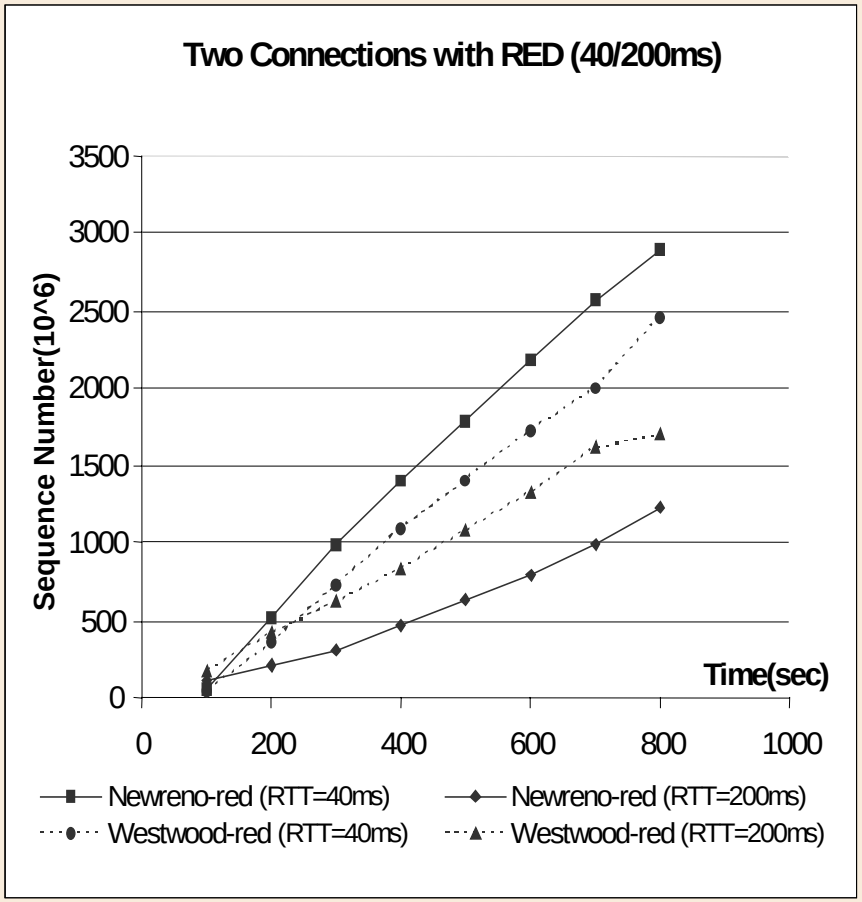






Improved Fairness

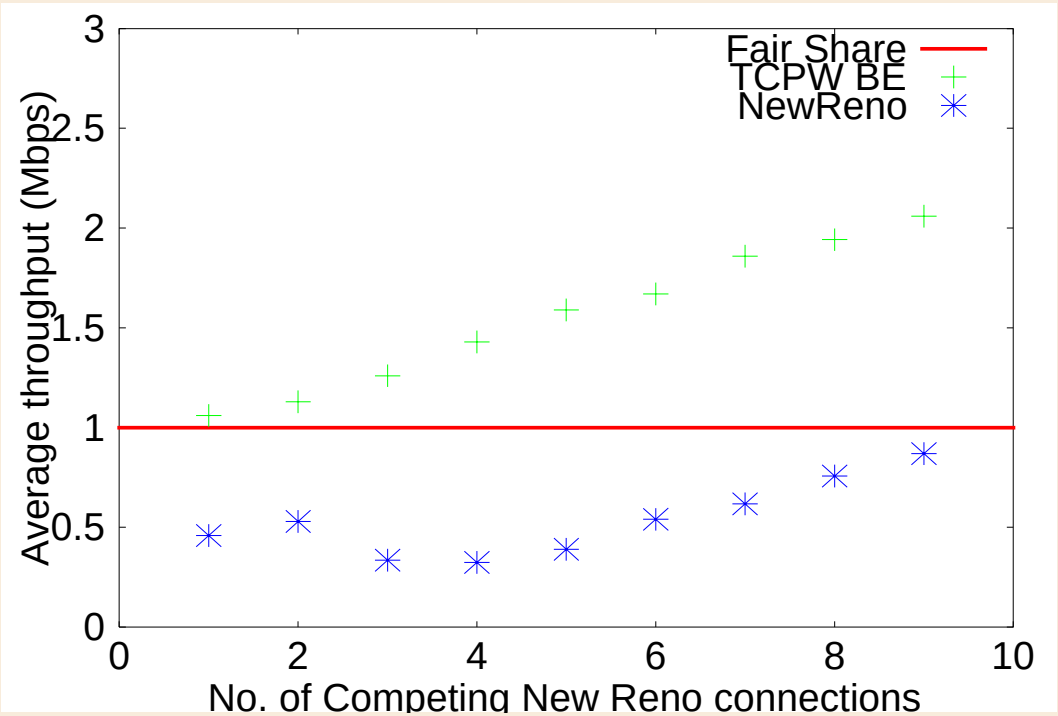




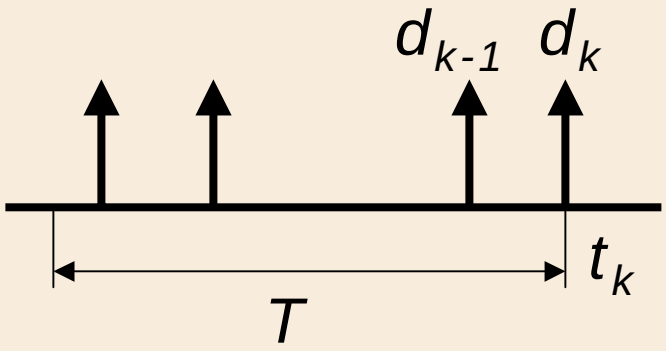
The results above are observed for buffer >= pipe size

BE Lacking in Friendliness!

- 10 total connections
- No random errors
- Average throughput per connection is provided below



TCPW Rate Estimation (RE)



T is the sample interval

- Allows more cross traffic interleaving with connection traffic, thus provides achieved Rate Estimation (RE):

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

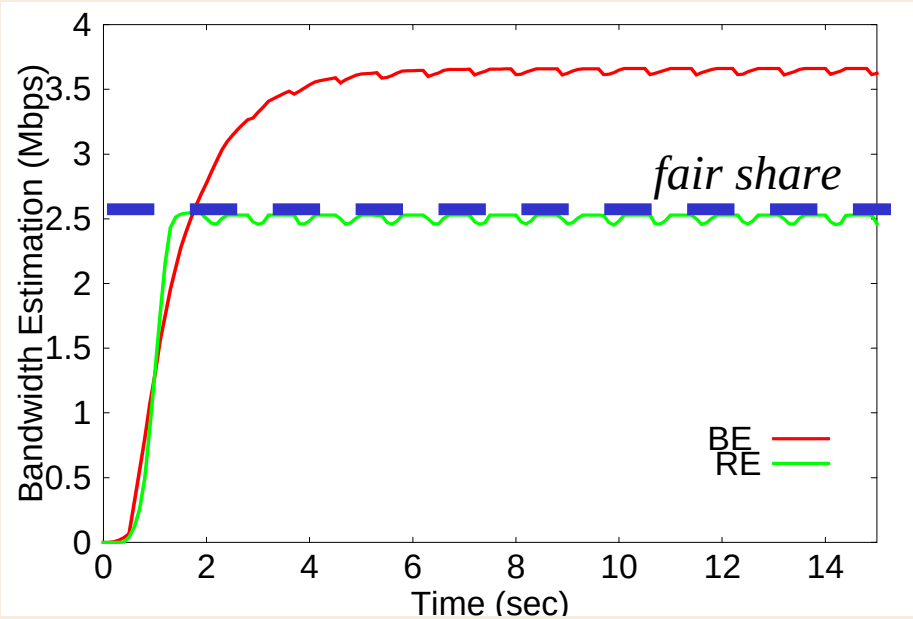
$$RE_k = \alpha_k RE_{k-1} + (1-\alpha_k)\left(\frac{b_k + b_{k-1}}{2}\right)$$ exponential filter

$$\alpha_k = \frac{2\tau - \Delta t_k}{2\tau + \Delta t_k}$$ filter gain

BE and RE Comparison

2 connections, sharing 5Mbps bottleneck

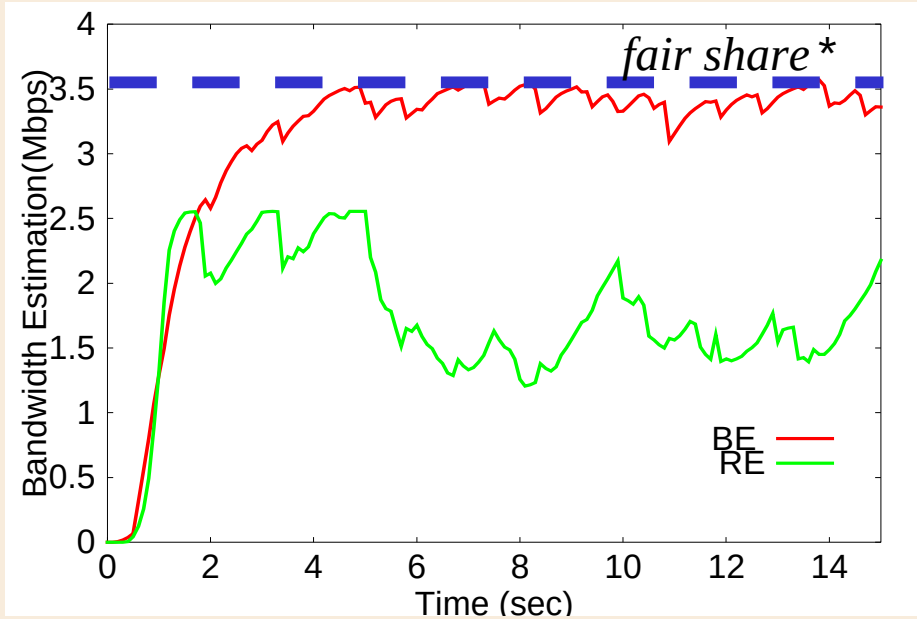
Congestion, no errors



BE overestimates fair share (2.5 Mbps)

Not friendly towards NewReno

Errors (0.5%), no congestion



RE under estimate fair share (3.6 Mbps)

No throughput improvement

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

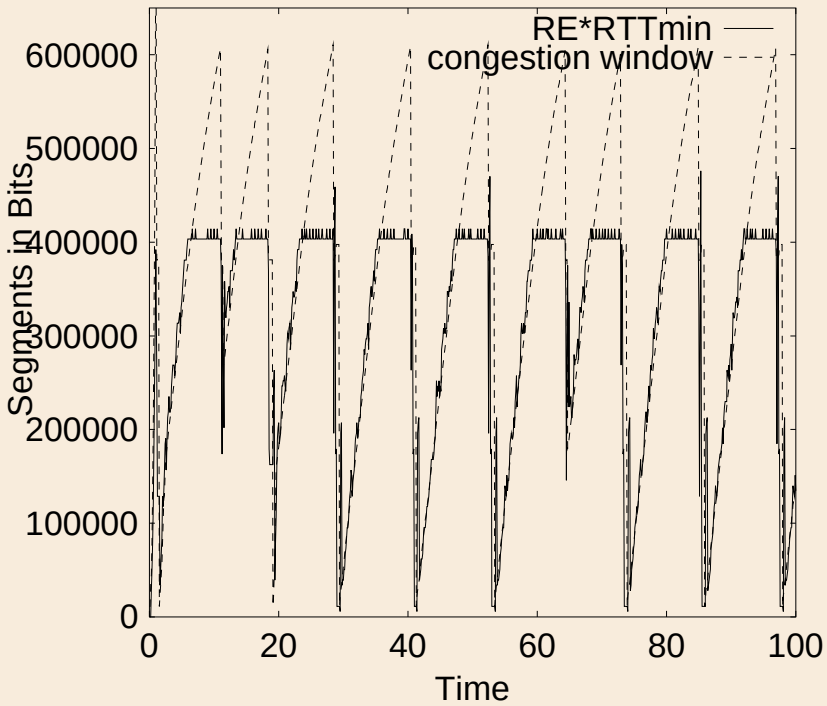
TCPW Adaptation

- Neither RE or BE are effective in all conditions
  - BE is more effective under random error loss
  - RE more effective under congestion loss (buffer overflow)
- We need ***adaptation***: choose between an aggressive estimate (like BE) and a conservative estimate (like RE) depending on current *congestion measure*
- High congestion => loss likely due to buffer overflow
- Low congestion => loss likely due to error

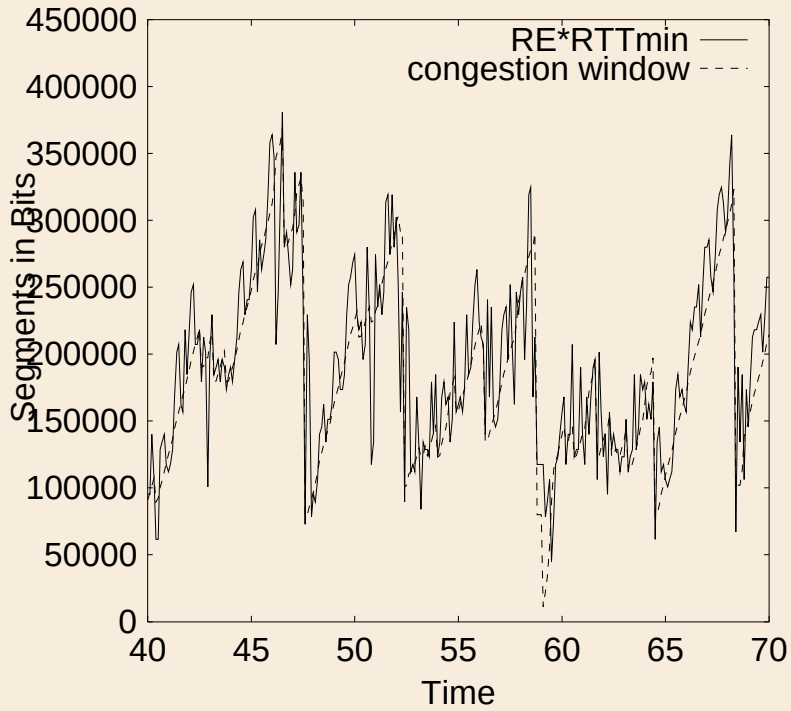
Congestion Measure

- Congestion measure is obtained comparing cwnd vs. RTTmin x RE;

i.e ‘expected rate’ vs. ‘achieved rate’; similar to Vegas’ congestion measure!



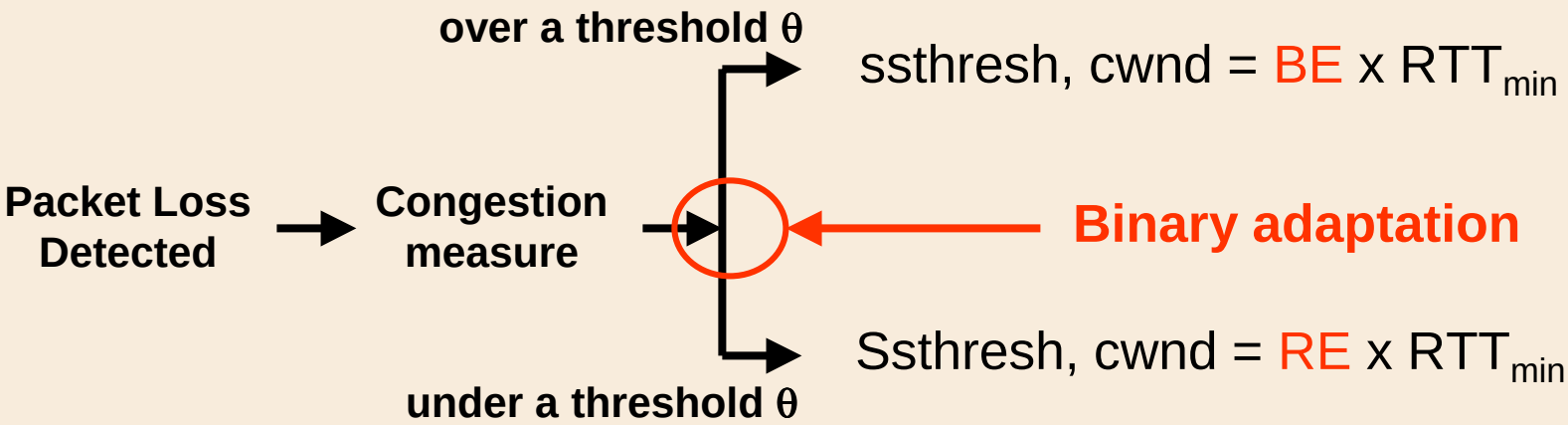
Congestion case



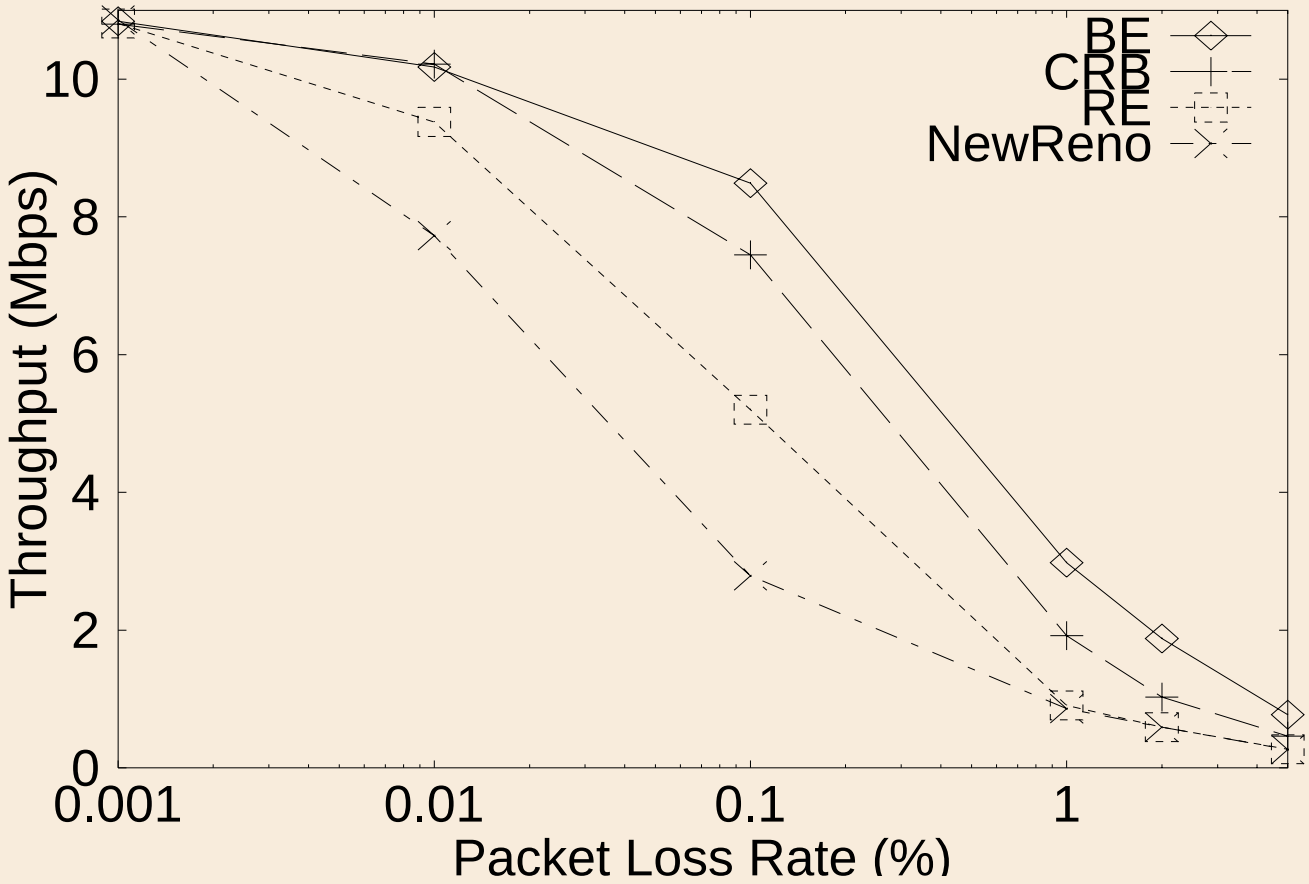
Link Error Case

TCPW CRB

- ***Combined Rate and Bandwidth*** estimation (**CRB**) uses a congestion measure to choose between RE or BE upon packet loss to set the ssthresh



CRB: More friendly, Less Efficient



Efficiency/Friendliness Profile

- Each point in the graph is obtained by:

a) Running a simulation with N NewReno flows, we obtain throughput t_{R1} and total link utilization U_1

b) Running a simulation with N/2

NewReno and N/2 TCP*x* flows,

we obtain throughput t_{R2} of

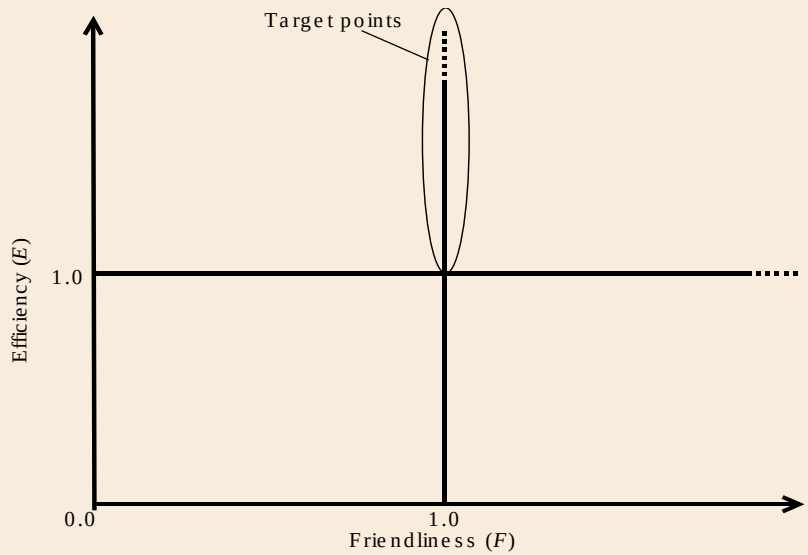
NR flows and total link utilization U_2

c) We define:

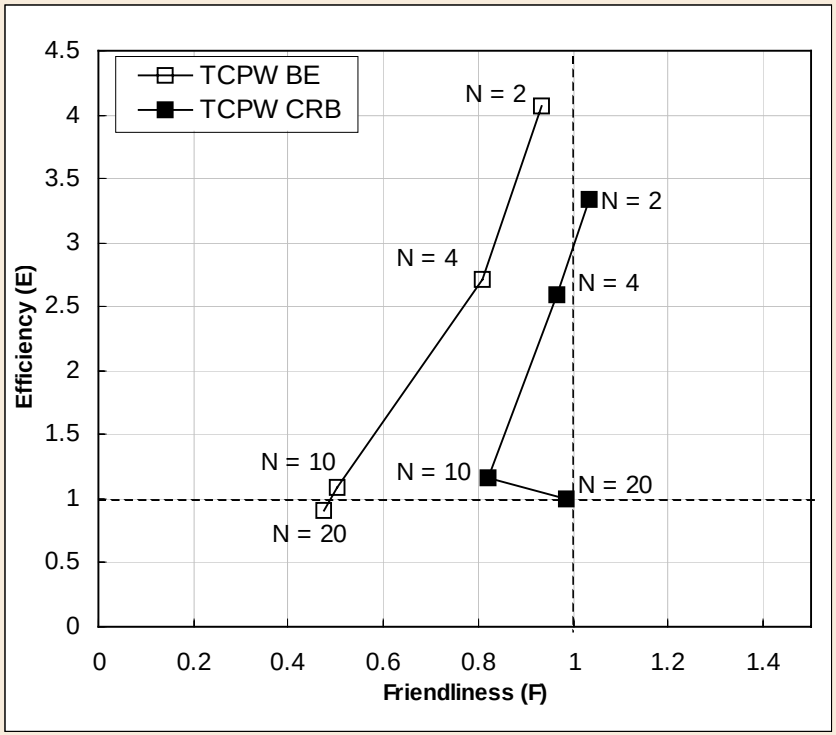
Efficiency Improvement $E = U_2 / U_1$

Friendliness $F = t_{R2} / t_{R1}$

And thus obtain a point (F,E) in the graph

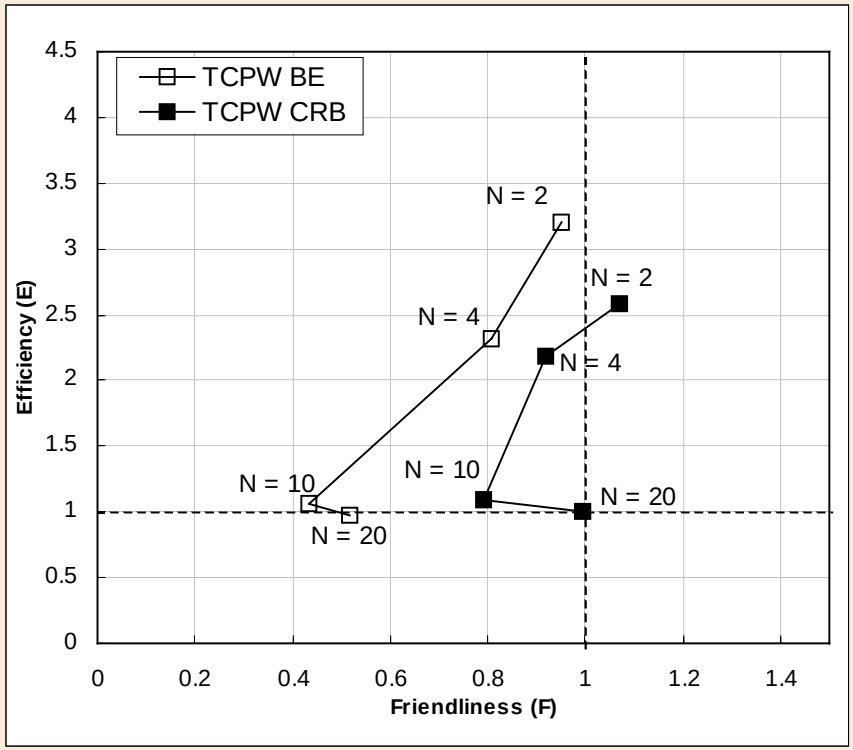


TCPW E/F Profiles for BE and CRB



(a) 45 Mbps, 74ms, 0.1% error

(b) 11 Mbps, 74ms, 2% error



- TCPW CRB provides less utilization gain but better friendliness

Better E/F Profile: TCPW ABSE

BE Sampling:

Packet *pair*,

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

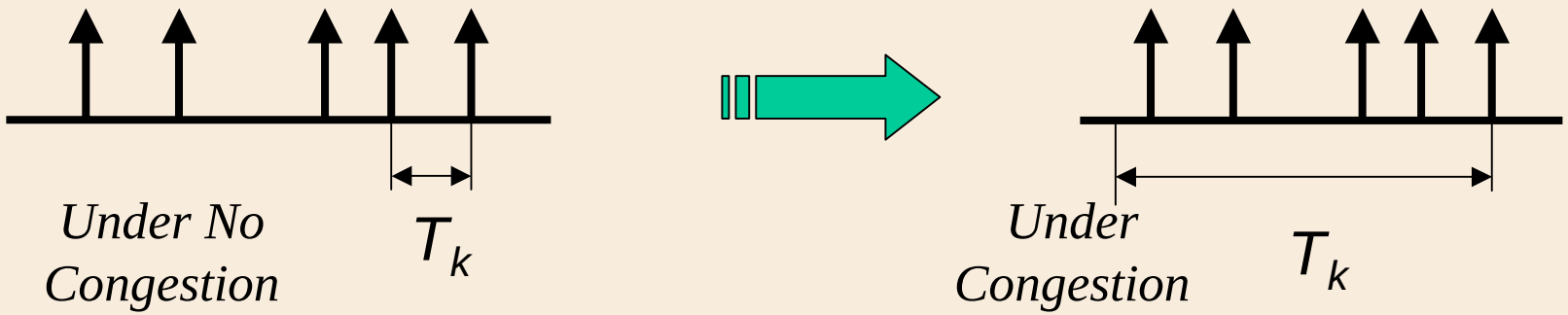
effective under random loss,
overestimates under congestion

RE Sampling:

Packet *train*,

$$S_k = \frac{\sum_{t_j > t_k - RTT} d_j}{RTT}$$

fair estimate under congestion,
underestimates under random loss



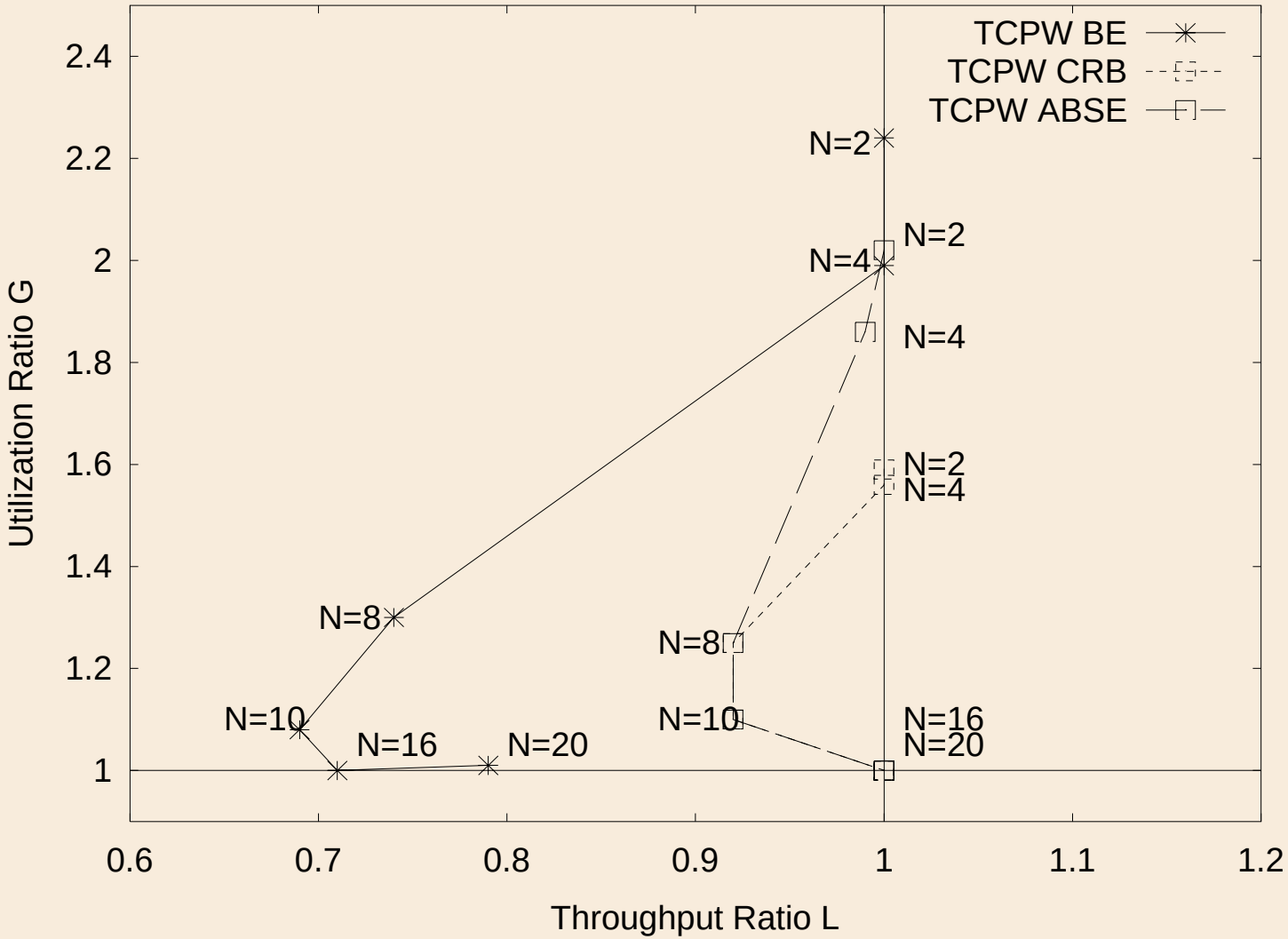
- To obtain *ERE*: adapt the sample interval T_k according to *congestion level*
- *Congestion level* is similar to that in Vegas: *Expected Rate-Achieved Rate*
- T_k ranges from *one 'interACK' interval* to *RTT*
- *Note: ERE is NOT the Residual Bandwidth*
  ERE is NOT the Capacity at the Bottleneck Link!

Sample Width Adaptation

$$let \quad \gamma = \left(\frac{cwin}{RTT_{\min}} - \hat{Th}_k\right) / \frac{cwin}{RTT_{\min}}$$

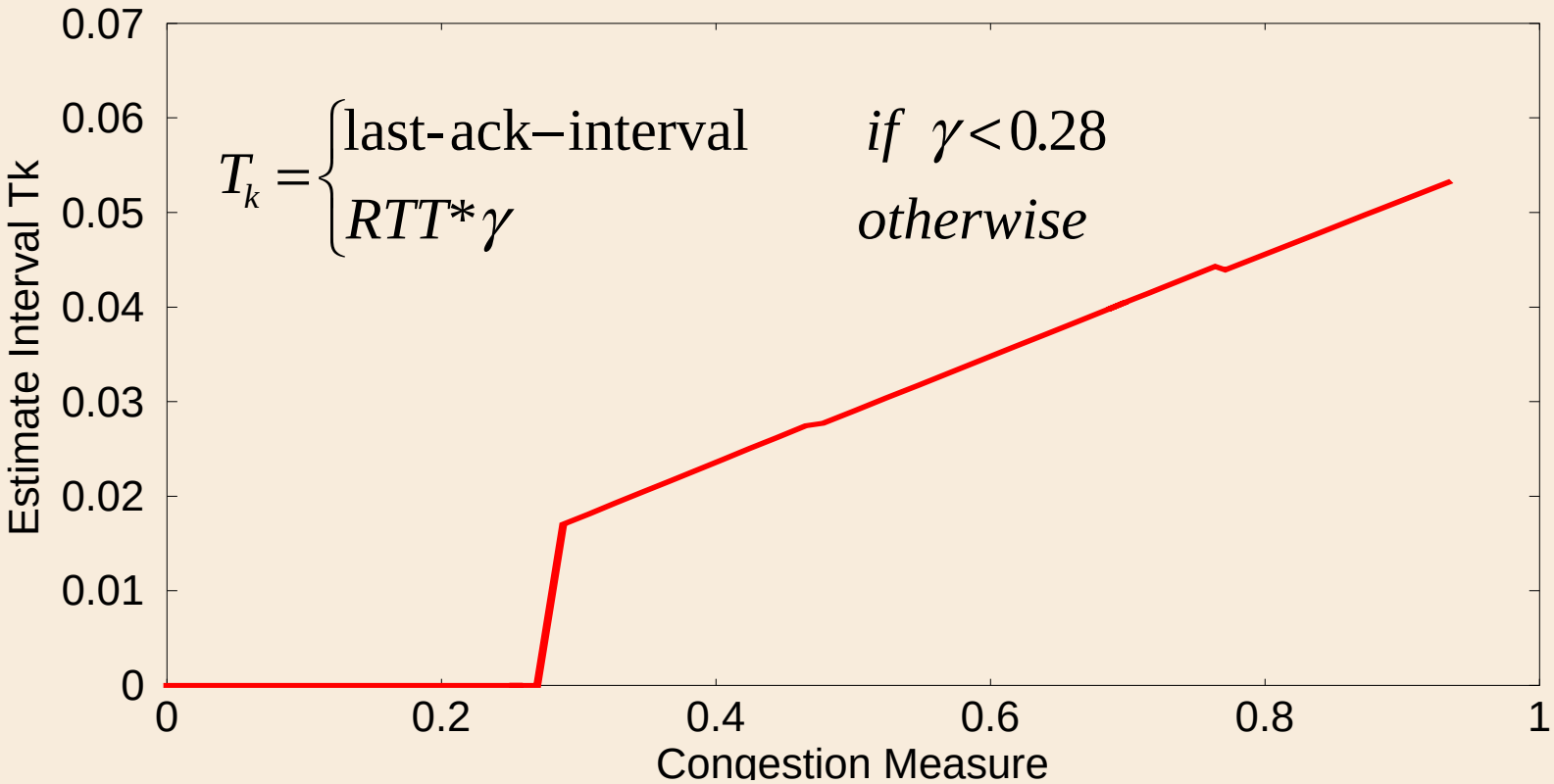
$$T_k = \begin{cases} \text{last-ack} - \text{interval} & if \quad \gamma < 0.28 \\ RTT * \gamma & otherwise \end{cases}$$

E/F Profiles of BE, CRB and ABSE

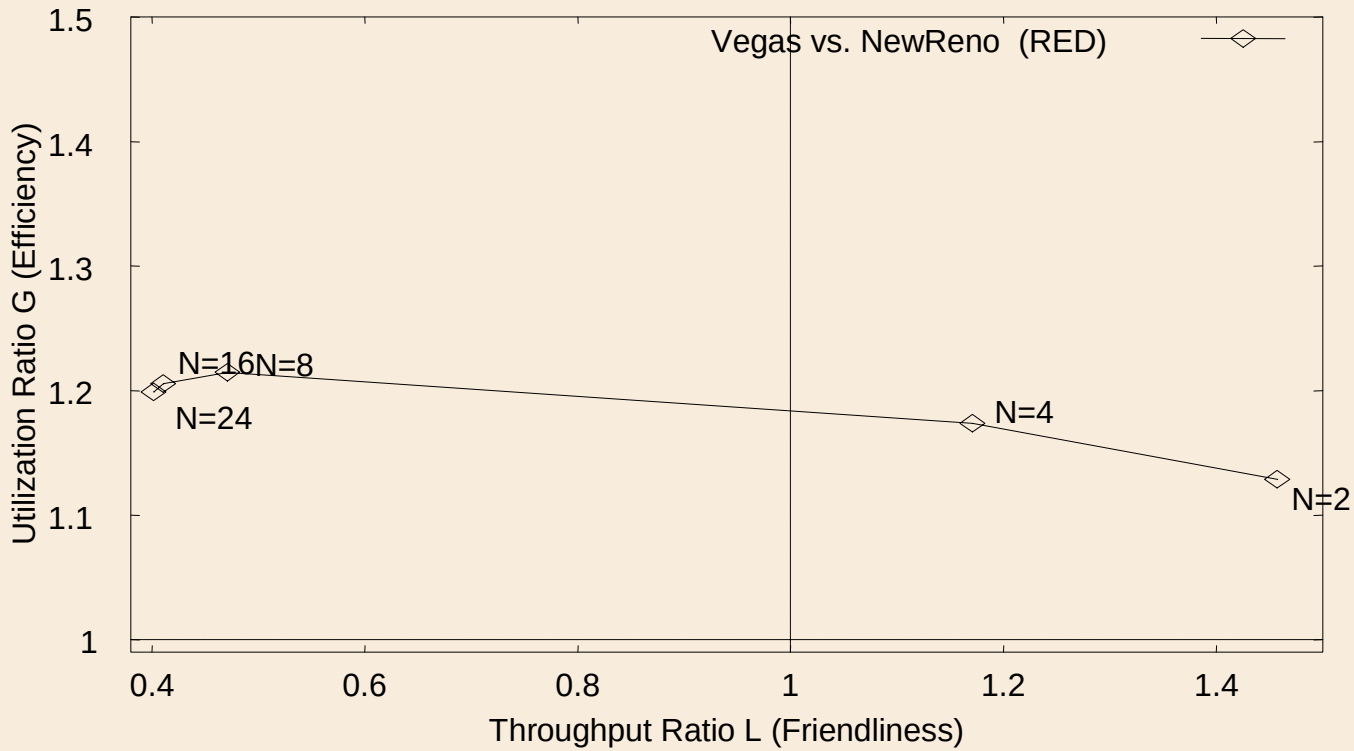


Controllable Friendliness via Adaptive Estimation in ABSE

Can control Friendliness by appropriate changes to the function below:



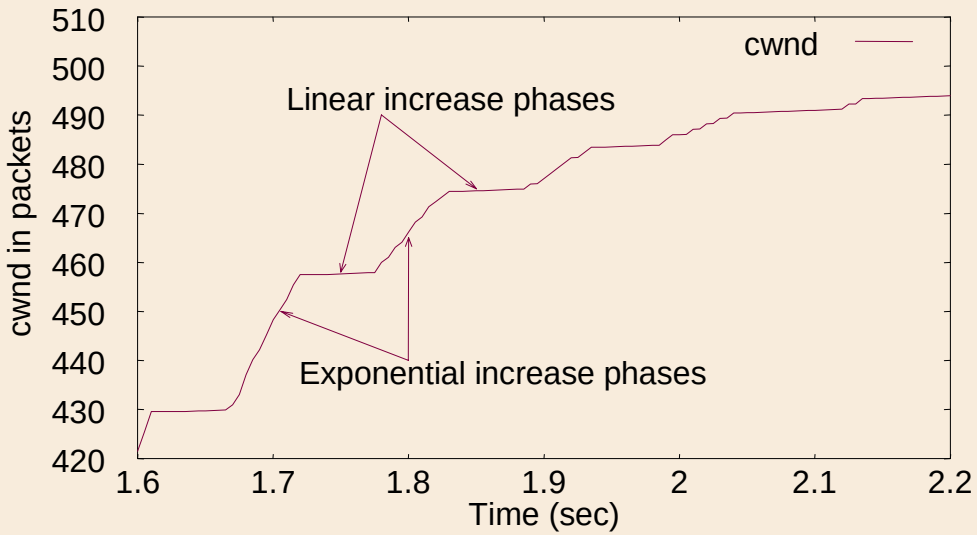
E/F Profile of Vegas/FAST



Vegas uses <u>fixed</u> targeted queue length => varying friendliness depending on number of connections!

Use of ERE in Agile Probing

- Take advantage of ERE to implement agile alternative to Slow Start: *Agile Probing*
  - Adaptively and repeatedly reset *ssthresh* to ERE until sender reaches pipe size or encounters packet loss



- Includes multiple mini 'exponential increase', and mini 'linear increase' phases
- *cwnd* grows slower as it approaches BDP
- Connection converges faster to its pipe size with less buffer overflow

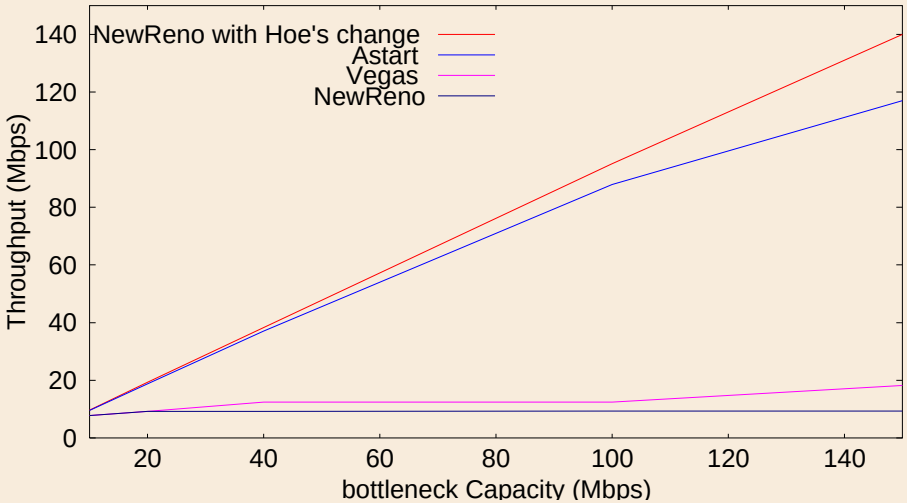
Agile Probing (Aprobe)

"Aprobe" is invoked at Startup or after a timeout

```
if ( 3 DUPACKS are received)
   switch to Congestion Avoidance phase;
else (ACK is received)
   if (ssthresh < (ERE*RTTmin)/seg_size)
         ssthresh =(ERE*RTTmin)/seg_size;
   endif
   if (cwnd >ssthresh)  /*mini linear increase phase*/
      increase cwnd by 1/cwnd;
  else if (cwnd <ssthresh)  /*mini exponential increase phase*/
      increase cwnd by 1;
  endif
endif
```

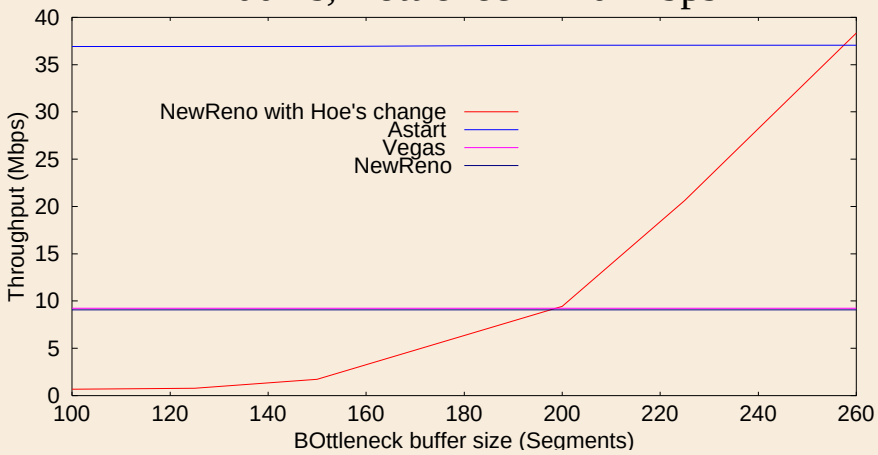
First 20 Seconds Throughput

RTT =100ms, Buffer =BDP

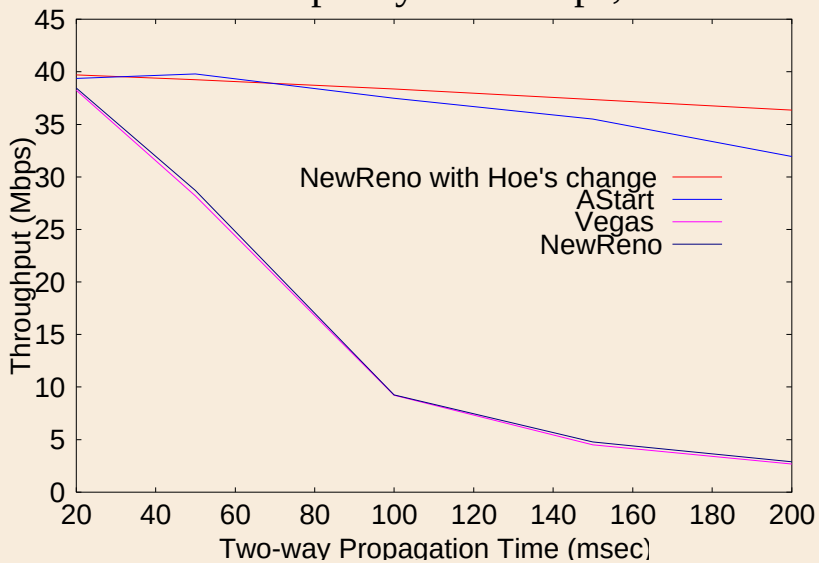


- Good scaling with capacity and propagation time
- Robust to buffer size variation

RTT =100ms, Bottleneck =40 Mbps



Bottleneck capacity = 40 Mbps, Buffer =BDP

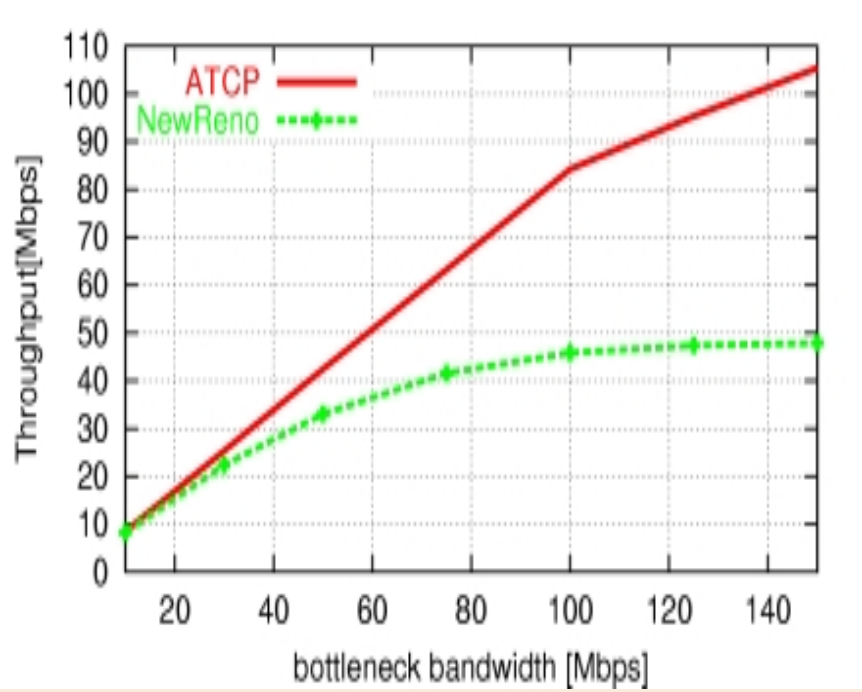


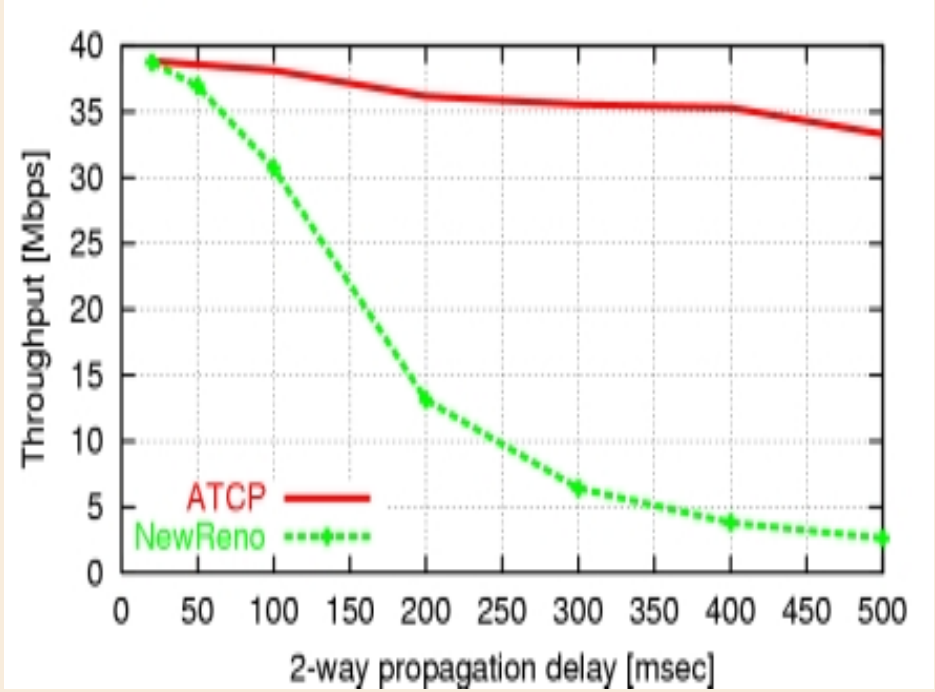
Persistent Non-Congestion Detection (PNCD)

Persistent Non-Congestion Detection is useful:

- During Congestion Avoidance, when load reduces drastically
- After prematurely entering Congestion Avoidance due to error loss in Startup
- After a physical capacity increase (e.g. switching to different link technology)

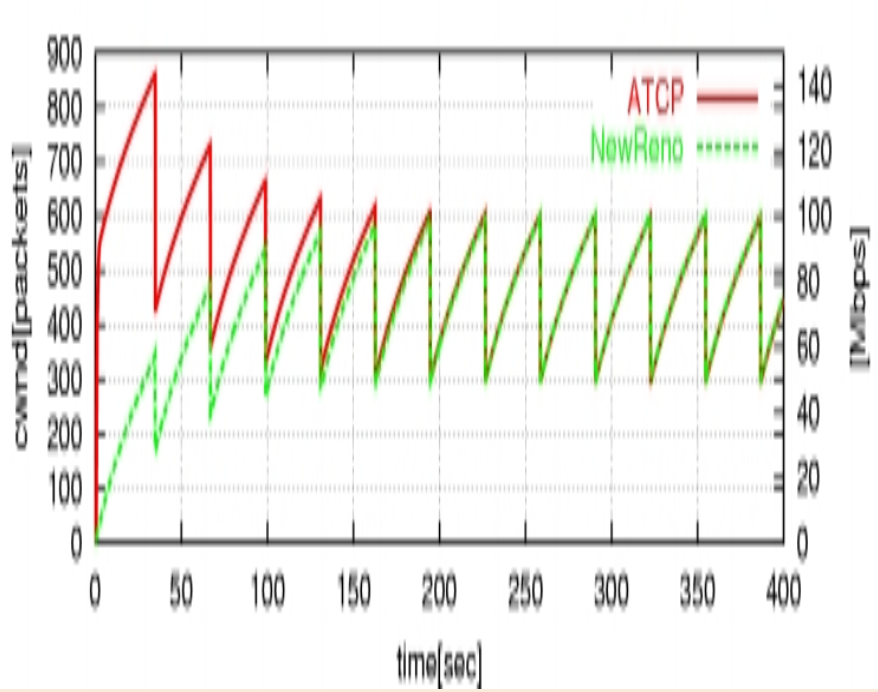
Throughput Under Dynamic Loading



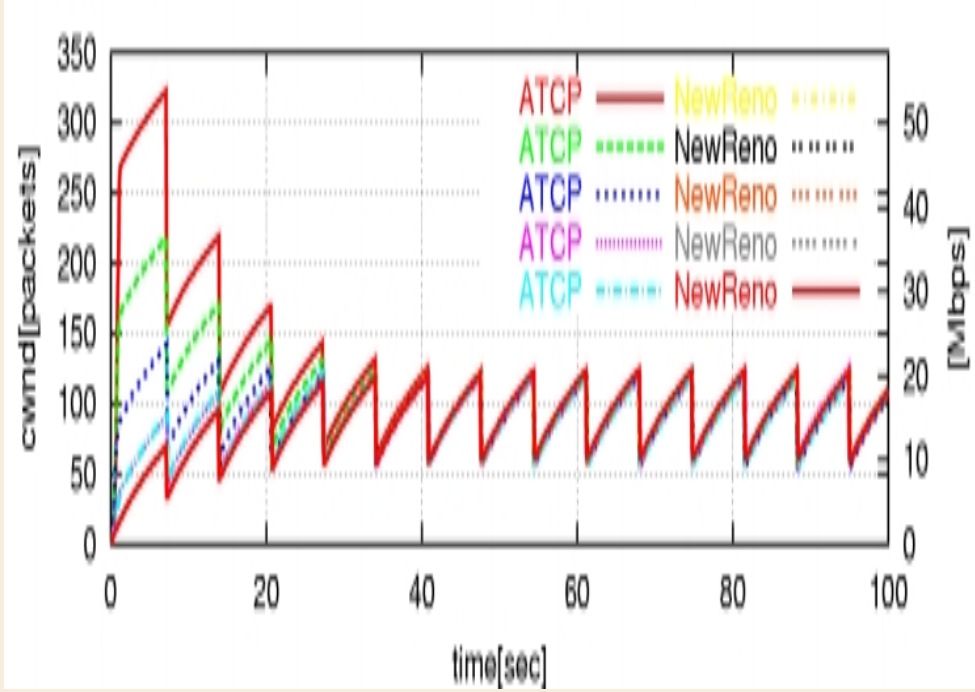


- Simulation runs for 600 sec.
- Randomly generated 100 flows with 30 sec lifetime each
- TCPW scales with capacity and RTT under dynamic load

Convergence/Friendliness



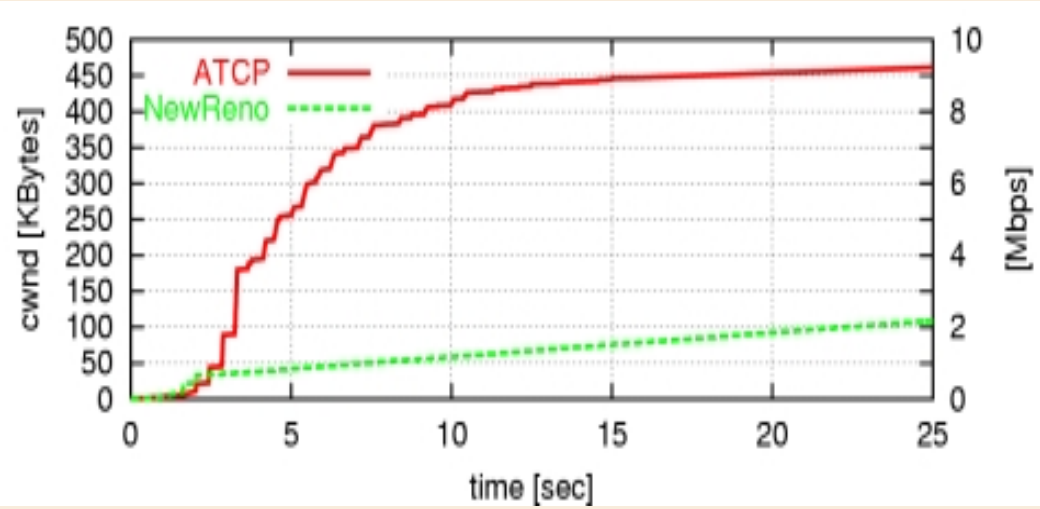
2 connections(one TCPW one NewReno)



10 connections (5 each)

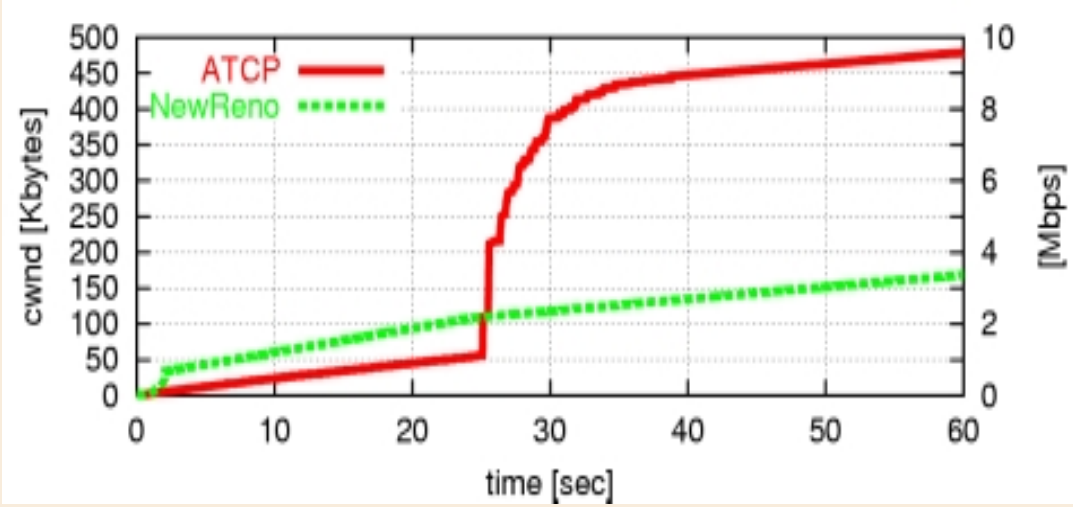
•TCPW with Agile Probing converges to the same rate as NewReno, showing friendliness

Measurements Results
(FreeBSD Implementation)

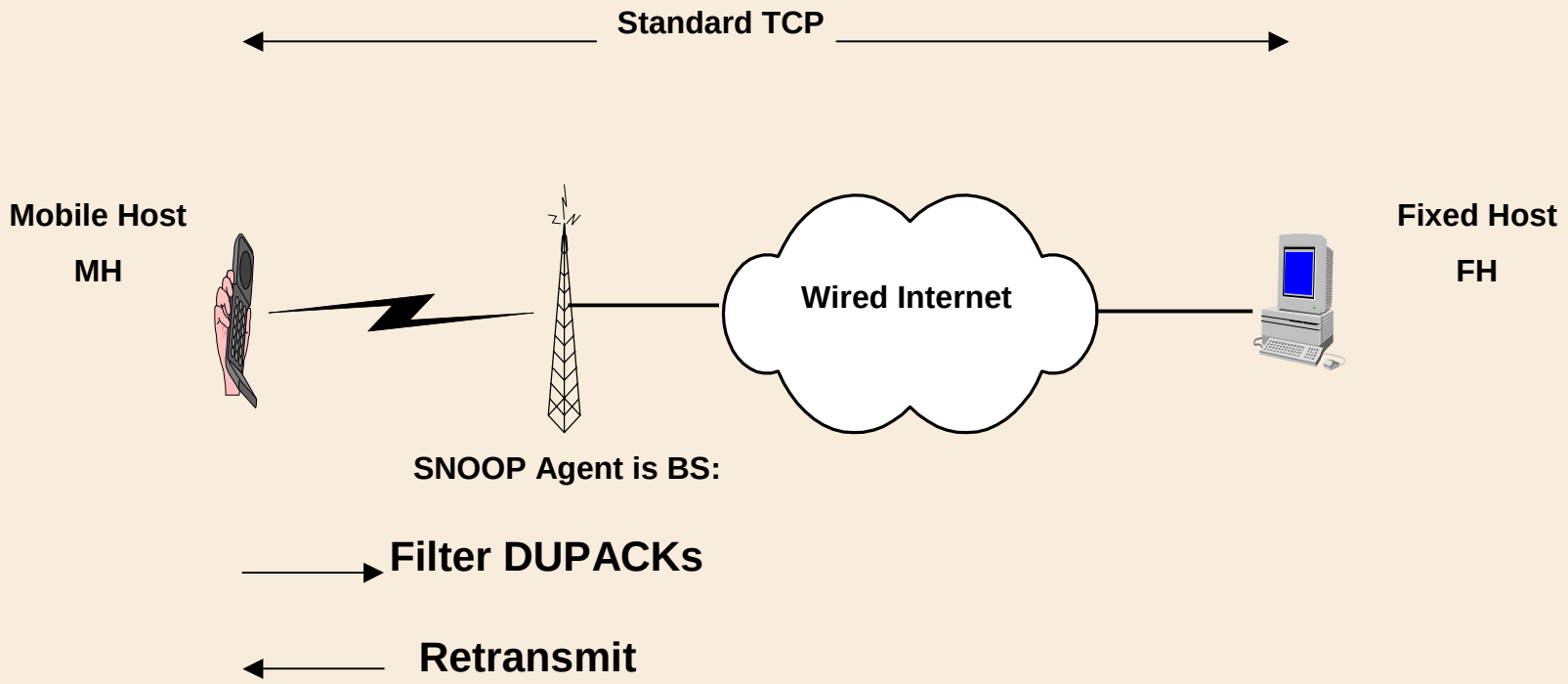


Startup invokes Agile Probing

PNCD invokes Agile Probing



Alternative Approaches: Snoop



Snoop (2)

- Snoop Agent monitors every packet in both direction
- Maintain a cache of unacknowledged TCP data to allow link layer retransmission
- When DUPACKs received by BS from MH, retransmit on wireless link, if packet present in buffer
- Prevents fast retransmit at TCP sender FH by filtering the DUPACKs at BS

Snoop (3)

Advantages

- No Change to TCP in Fixed Hosts
- Local recovery from wireless loss
- Reduce unnecessary Fast Retx at sender

Disadvantages

- Link layer at BS needs to be TCP-aware
- Fails if TCP headers are encrypted, in which case BS can't snoop

XCP: eXplicit Control Protocol

- Senders express their setting (cwnd, RTT) to routers, and routers express changes required to senders
- Recognizes two types of requirements for Congestion Control:
  - Efficiency: Achieve high link utilization
  - Allocation: Allocate bandwidth according to desired criteria; e.g. fairness, QoS, etc.
- Approach: Decouple controls for efficiency and allocation :
  - Control aggregate traffic to achieve efficient link utilization
  - Divide link bandwidth among connections to achieve desired allocation criteria

XCP: eXplicit Control Protocol (2)

- Efficiency Controller
  - Goal: Match aggregate input traffic to link capacity & drains the queue
  - Algorithm:
    - Aggregate "used pipe" is changed by Δ
    - Δ increases with an increase in "spare pipe" size; and
    - Δ decreases with an increase in the router queue size; i.e.

$$\Delta = \alpha \; x \; Spare\ Bandwidth \; x \; d_{avg} - \beta \; x \; Queue\ Size$$

XCP: eXplicit Control Protocol (3)

- Allocation Controller
  - Goal: Divide Δ among flows to converge to allocation criteria
  - Algorithm (AIMD):

    If $\Delta > 0 \Rightarrow$ Divide Δ equally between flows (regardless of current rate)

    If $\Delta < 0 \Rightarrow$ Divide Δ between flows in proportion to their current rates

Need to estimate number of flows *N:*

$$N = \sum_{pkts\ in\ d_{avg}.} \frac{RTT_i}{d_{avg} \times Cwnd_i}$$

XCP: eXplicit Control Protocol (4)

Advantages

- Scalable for bandwidth and delay
- Decouples efficiency and fairness problems
- No per-flow state

Disadvantages

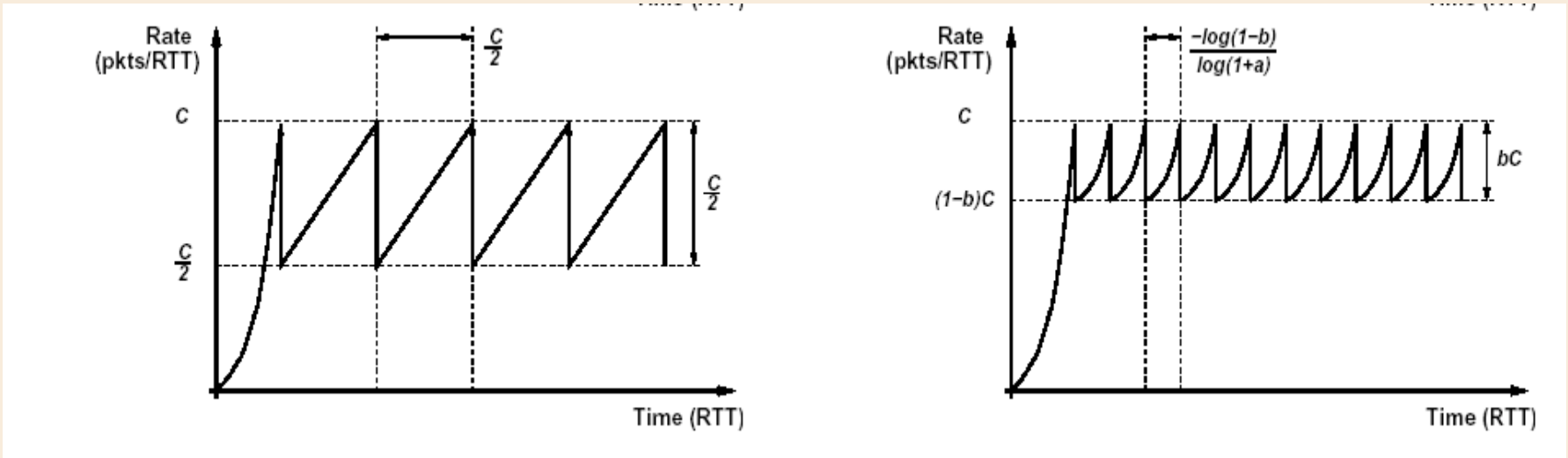
- Needs router participation
- Malicious Sender can falsify the header and mess up the feedback calculation
- Impact on accuracy of using *avgRTT* as fundamental cycle for all flows sharing a link

Scalable TCP (STCP)

- Generalized STCP Algorithm:
  - Let *a* and *b* be constants and *cwnd* be the congestion window:
    - For each ACK in a RTT without a loss:

      $cwnd = cwnd + a$
    - For each window experiencing a loss:

      $cwnd = cwnd - b*cwnd$
  - In STCP, $a = 0.01$, $b = 0.125$
- Parameters a and b are not adaptive? Still useful for intranets



High Speed (HS) TCP

- Standard TCP uses *Additive Increase* parameter a =1 per RTT upon ACK, and *Multiplicative Decrease* parameter b = 0.5 upon a loss
- In HS TCP, the value of a and b depends on the current cwnd size
  - Increase cwnd *faster* with larger cwnd
  - Decrease cwnd *slower* with larger cwnd
  - Act like standard TCP at small cwnd
- No provision for handling error losses? And
- Increase/decrease is not directly related to dynamic loading?

FAST TCP

- As in Vegas, FAST uses queuing delay as congestion measure, aims to stabilize cwnd
- Updates window every RTT, not every ACK
- Window update rule (latest we know):

  $cwnd = cwnd\ (baseRTT/RTT) + \alpha$

  Where *baseRTT* is the minimum *RTT* observed by the connection, and α is a preset positive parameter
- Upon a loss, act similar to TCP NewReno
- No provision for handling error losses ?
- Friendliness depends on setting of α, which does not adapt to current loading?

Summary

- TCP Westwood is able to deal with dynamic, large leaky pipes, with only sender side modification:
  - Better utilization
  - Better Fairness
  - Controllable and Self-tuned Friendliness
- Compared to STCP, FAST and High Speed TCP:
  - TCPW is equipped with the ability to better handle error losses, and self tune its friendliness to existing conditions, (except for buffer space: difficult problem for all protocols!)
- Compared to Snoop, and XCP
  - Sending rate determination in TCPW done at sender only; TCPW does not need help from lower layer or intermediate nodes
  - Performance slightly less than Snoop, but close enough!
  - **Need to compare to XCP, TCPW should provide lesser performance, the question is by how much?**

On-going Work

- Control modeling and analysis
- Wireless measurement experiments
- Service differentiation at Transport layer
- Mining transmitted stream, in addition to ACK stream
- Detailed comparison to STCP, High Speed TCP, FAST, and XCP; especially at Gbps speeds
- Improvements in fundamentals of packet dispersion measurements
- Analytic studies of data and ACK stream statistics
- Extension to video streaming control

References (Cont.)

- *Hari Balakrishnan, Srinivasan Seshan, Etan Amir, and Randy Katz*, "Improving TCP/IP Performance Over Wireless Networks", Mobicom, November 1995
- *H. Balakrishnan and R. H. Katz*, "Explicit Loss Notification and Wireless Web Performance", Proceedings of IEEE GLOBECOM'98 Internet Mini-Conference, Sydney, Australia, November 1998
- *K. Fall and S. Floyd*, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," Computer Communication Review, V. 26 N. 3, July 1996, pp. 5-21
- *S. Floyd and V. Jacobson*, "Random Early Detection gateways for congestion Avoidance," IEEE/ACM transactions on Networking, August 1993
- *S. Floyd*, "TCP and Explicit Congestion Notification," ACM Computer Communication Review, V. 24 N. 5, pp. 10-23, Oct. 1994

References (Cont.)

- *Dina Katabi, Mark Handley, and Charles Rohrs*, "Internet Congestion Control for Future High Bandwidth-Delay Product Environments." submitted to Sigcomm 2002
- *Cheng Jin, David X. Wei and Steven H. Low*, "IETF Draft: FAST TCP for high-speed long-distance networks," draft-jwl-tcp-fast-01.txt, June 30, 2003
- *Tom Kelly*, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks Submitted for publication", December 2002.
- *Sally Floyd*, "HighSpeed TCP for Large Congestion Windows", Internet draft draft-ietf-tsvwg-highspeed-01.txt, work in progress, August 2003

References

- *Lawrence Brakmo, Sean O'Malley, and Larry Peterson*, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", 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