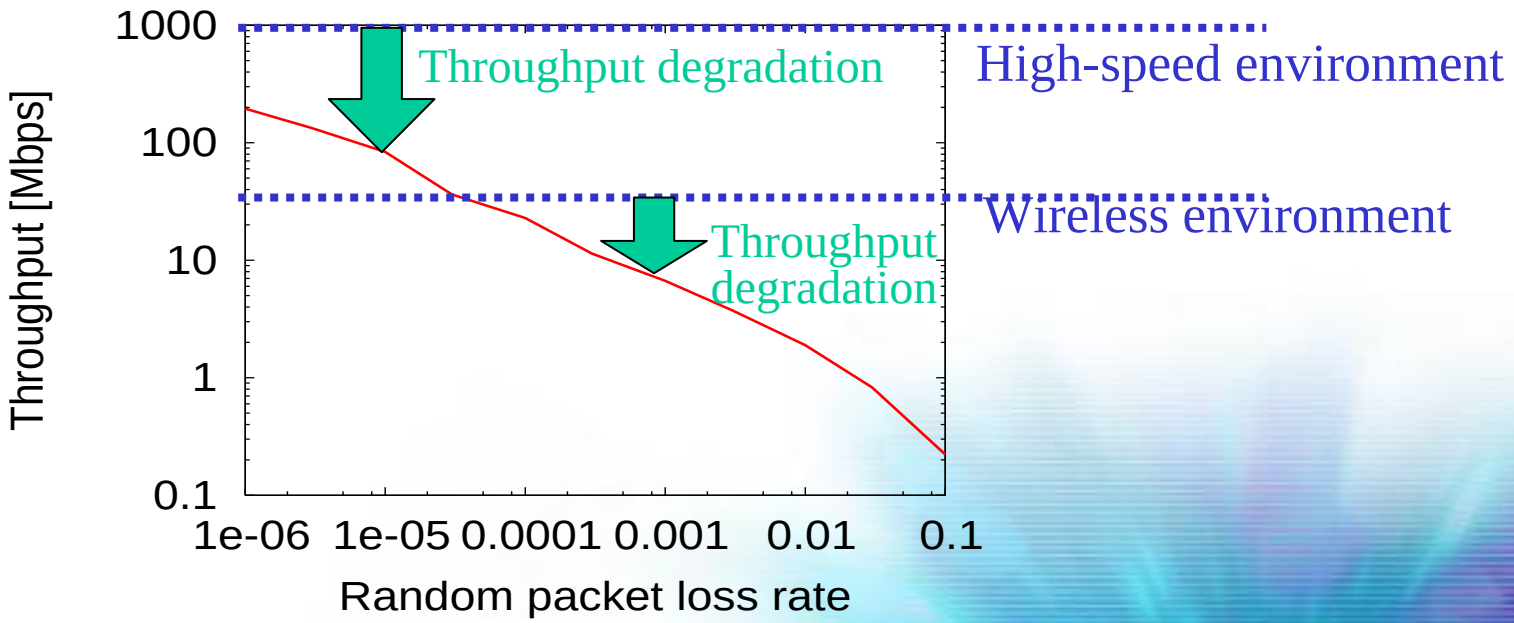
TCP-AdaptiveReno: Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm

HIDEyuki Shimonishi, Tutomu Murase
NEC Corp.
Sep. 2, 2006 at PFLDnet2006
h-shimonishi@cd.jp.nec.com

Empowered by Innovation NEC

Motivation

- Throughput degradation of TCP in high-speed and long distance environment
  - (Non-congestion) packet losses cause throughput degradation
  - Slow recovery from packet losses



Empowered by Innovation NEC

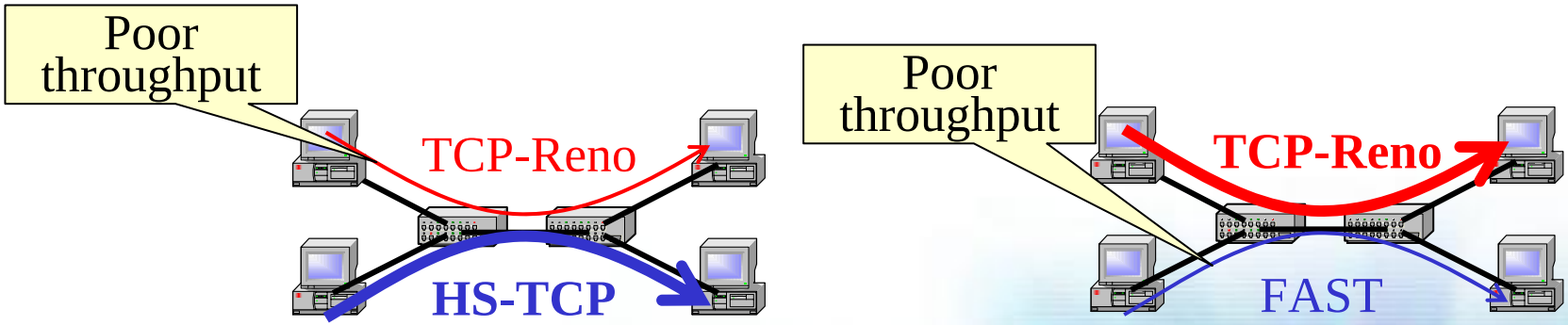
New TCP variants

(1) High Speed TCP (HS-TCP) [S. Floyd, RFC 3649, 2003]

- Tune congestion window increase/decrease based on congestion window size

(2) Scalable TCP [T. Kelly, PFDnet03, 2003]

(3) FAST [S. H. Low, draft-jwl-tcp-fast-01.txt, 2003]

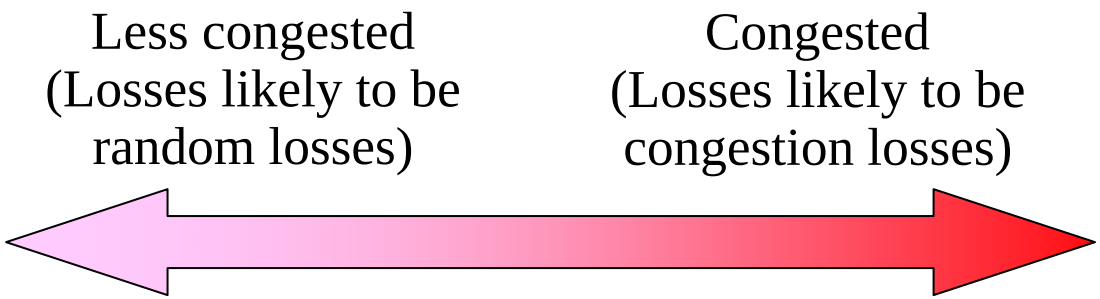


Challenge: friendliness to existing protocols (TCP-Reno)

Empowered by Innovation NEC

Improving efficiency-friendliness tradeoff

- Tune congestion window increase/decrease based on congestion estimation



TCP Westwood BBE		
Adaptive congestion window decrease	Based on the estimations	1/2
Adaptive congestion window increase	1MSS/RTT + α	1MSS/RTT
TCP Adaptive RENO (TCP-AReno)		

Empowered by Innovation NEC

Congestion level estimation

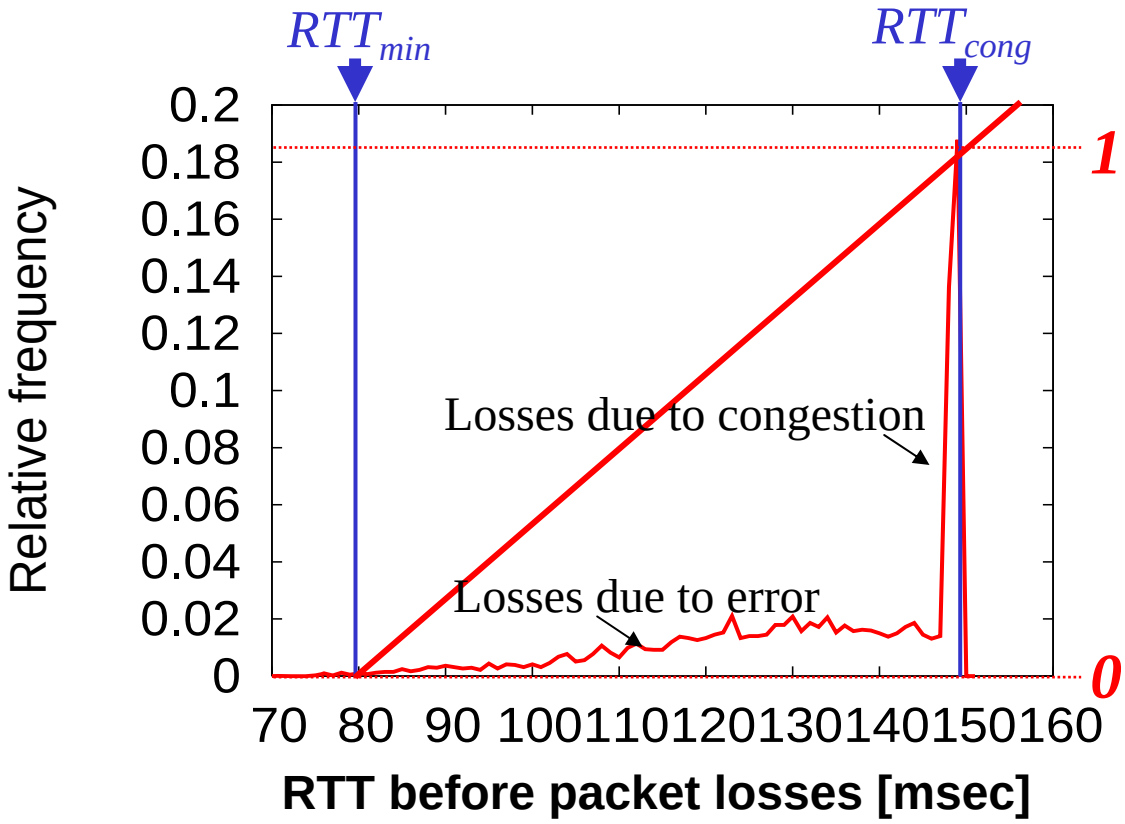
- “Congestion” ?
  - Congestion := a moment when coexisting TCP-Reno flows likely to reduce congestion window
    := a moment when packet losses likely to happen

- Possible condition indicator
  - Network feedbacks like (multi-bit) ECN
  - One way delay / **RTT**

Empowered by Innovation NEC

Congestion level estimation (cont'd)

- RTT dynamic range between RTT_{min} and RTT_{cong}
  - RTT_{min}: minimum RTT (= propagation delay)
  - RTT_{cong}: RTT value when packet losses likely to happen
- Congestion := position of current RTT in RTT dynamic range



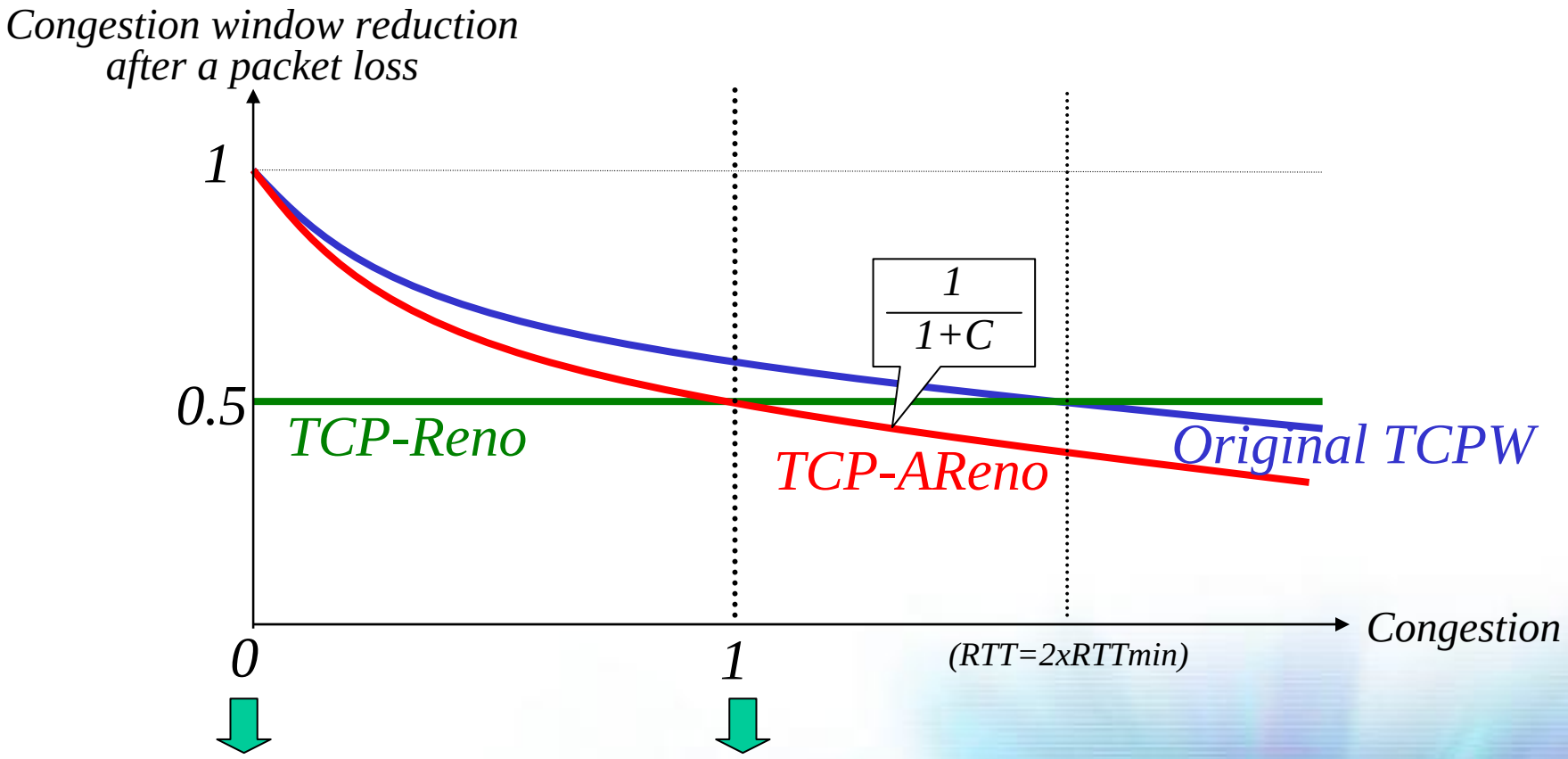
RTT_{cong} is an exponential average of RTT right before losses

Congestion level

$$c = \frac{RTT - RTT_{min}}{RTT_{cong} - RTT_{min}}$$

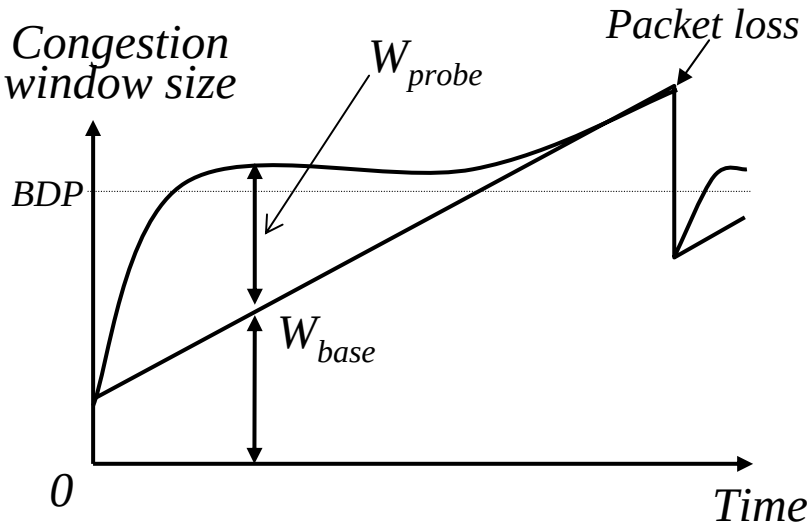
Empowered by Innovation NEC

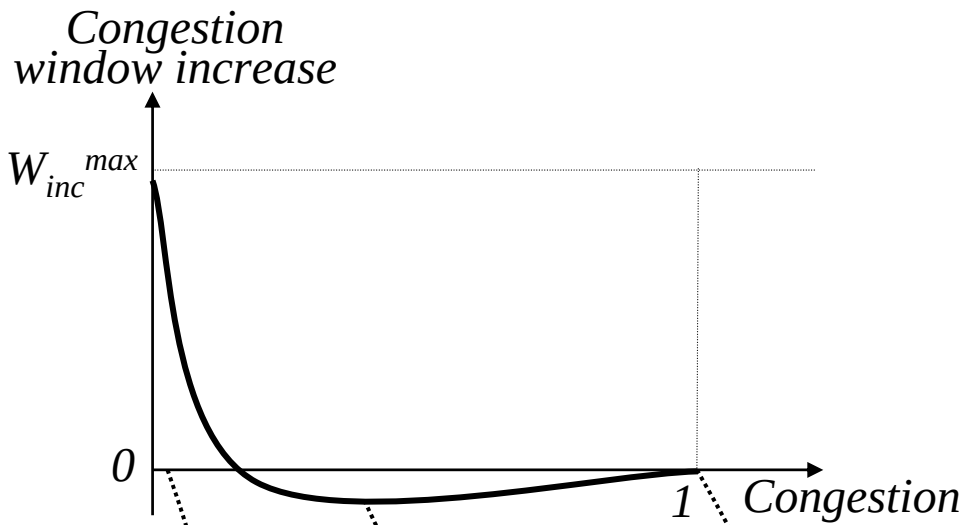
Congestion window reduction



Empowered by Innovation NEC

Congestion window increase





Congestion window = W_{base} + W_{probe}

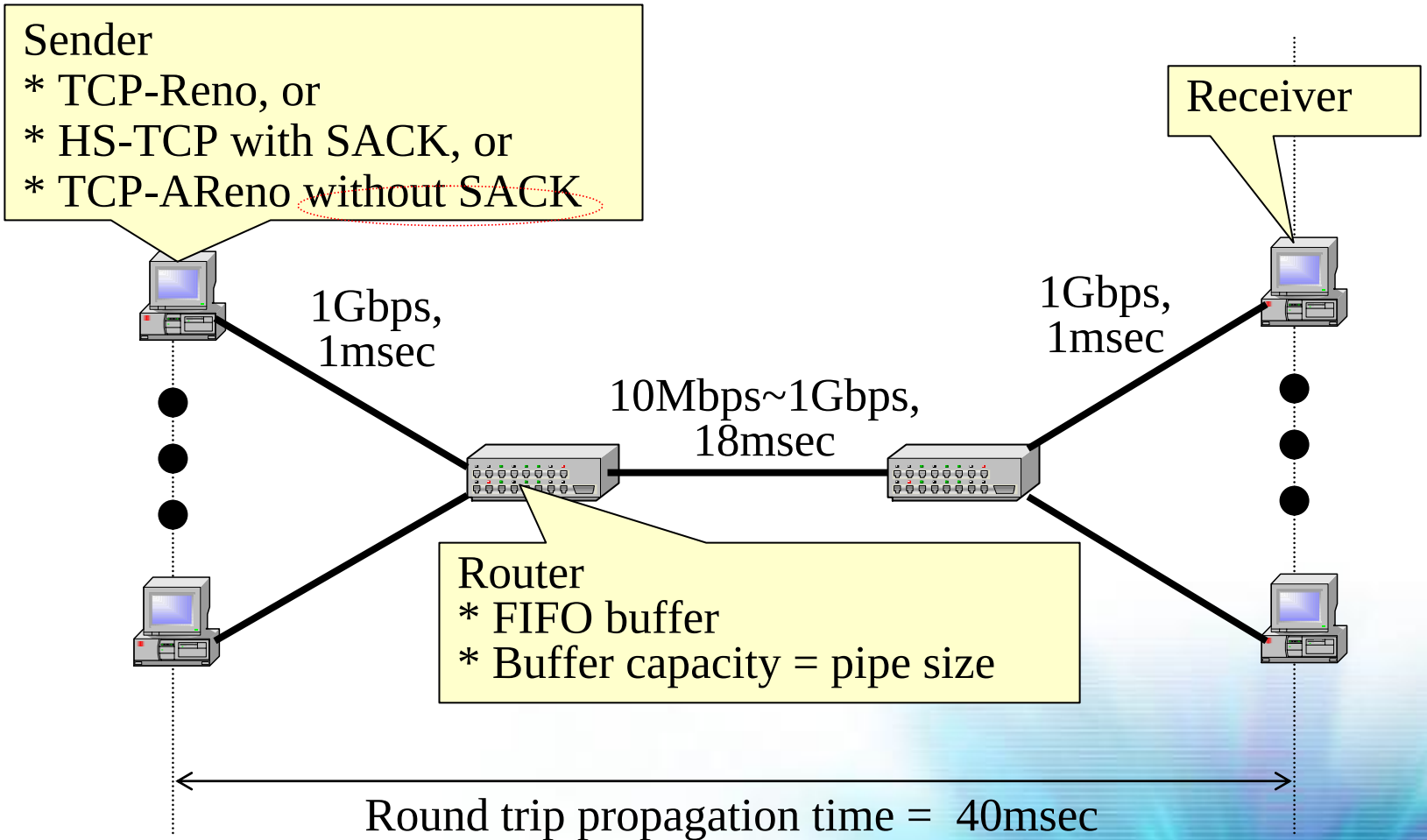
- W_{base}: increased like TCP-Reno
- W_{probe}: c≈0; increase faster – quickly fill the capacity

  0<c<1; W_{probe} converges to 0 – friendliness to TCP-Reno

  c≈1; No increase/decrease in W_{probe} part – packet loss behavior like Reno
  – avoid multiple losses
  – avoid synchronization issue

Empowered by Innovation NEC

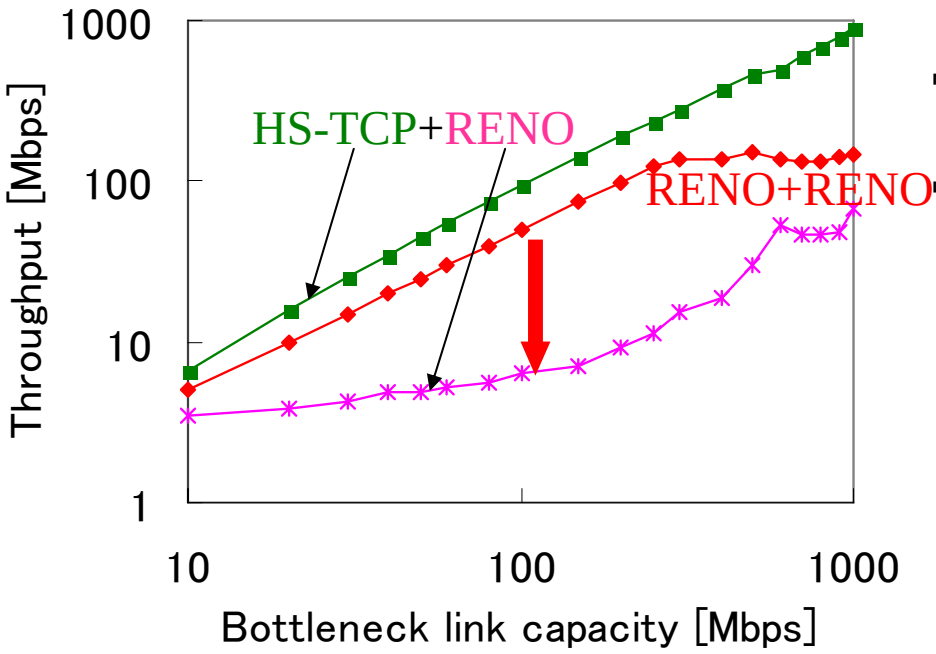
Simulation study

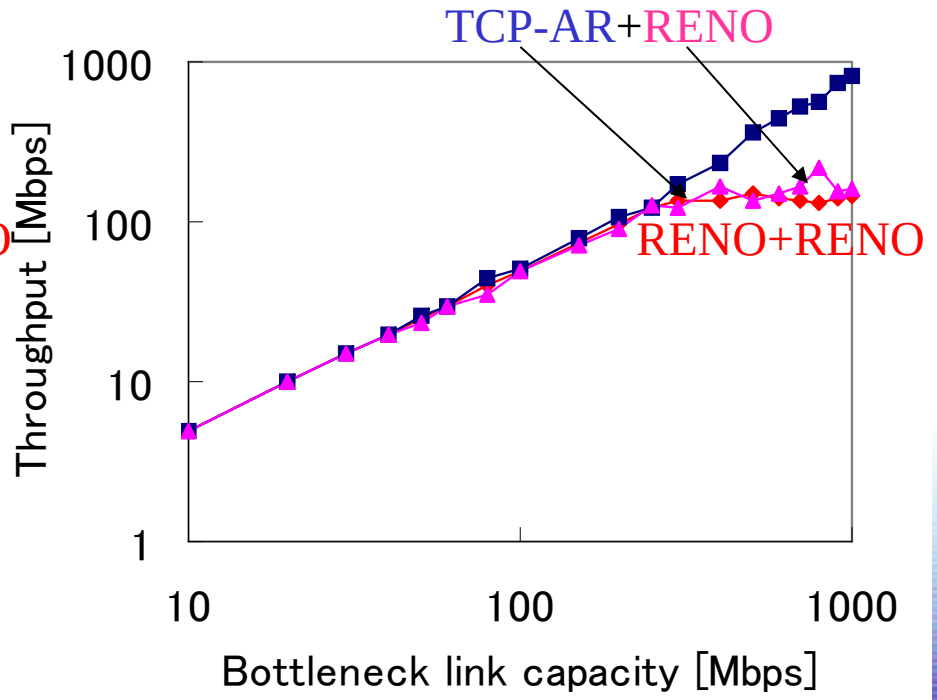


Empowered by Innovation NEC

Scalability and friendliness

- Two competing flows with various link capacities
  - TCP-Reno v.s. TCP-Reno/HS-TCP/TCP-AReno
  - 10^{-6} random packet loss

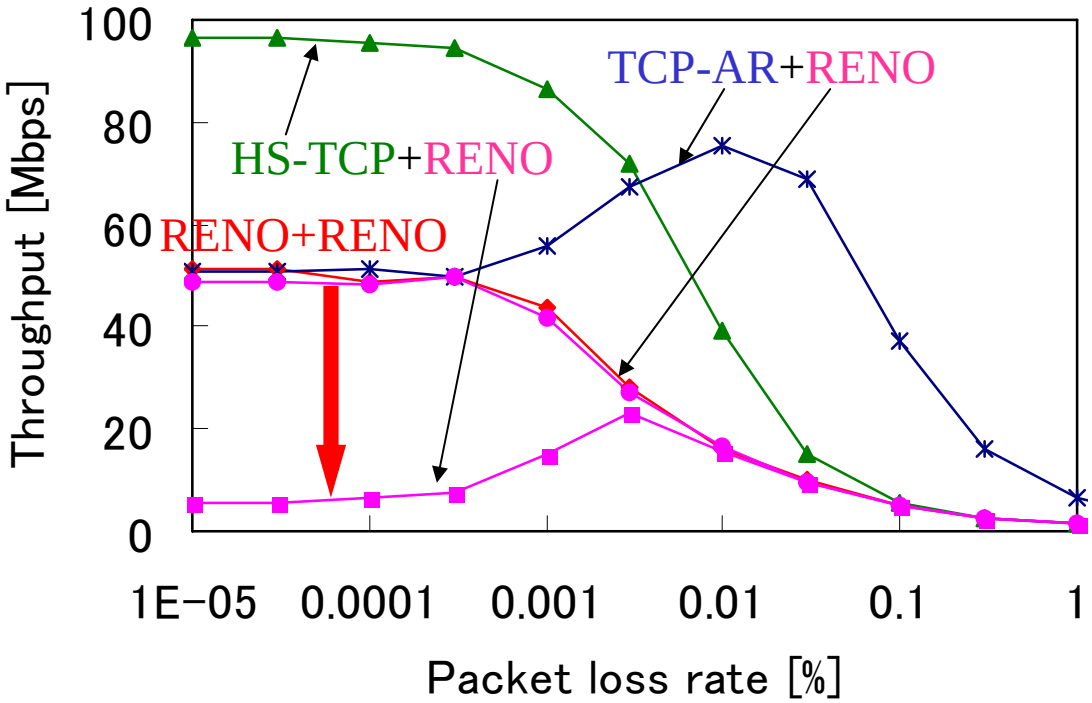




Empowered by Innovation NEC

Robustness to random packet losses

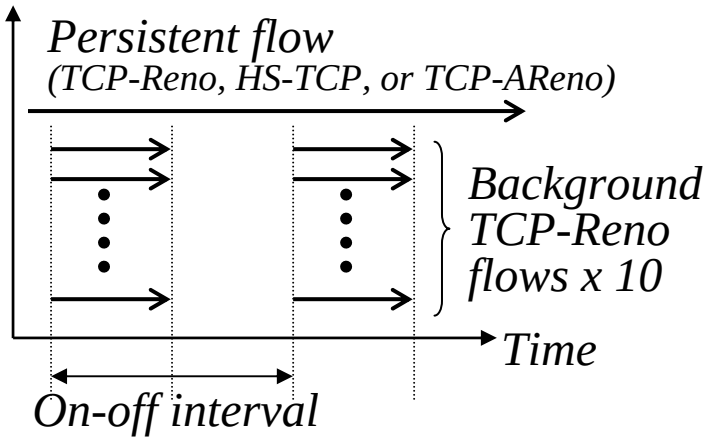
- Two competing flows with various random packet loss rate
  - TCP-Reno and TCP-Reno/HS-TCP/TCP-AReno
  - 100Mbps link capacity



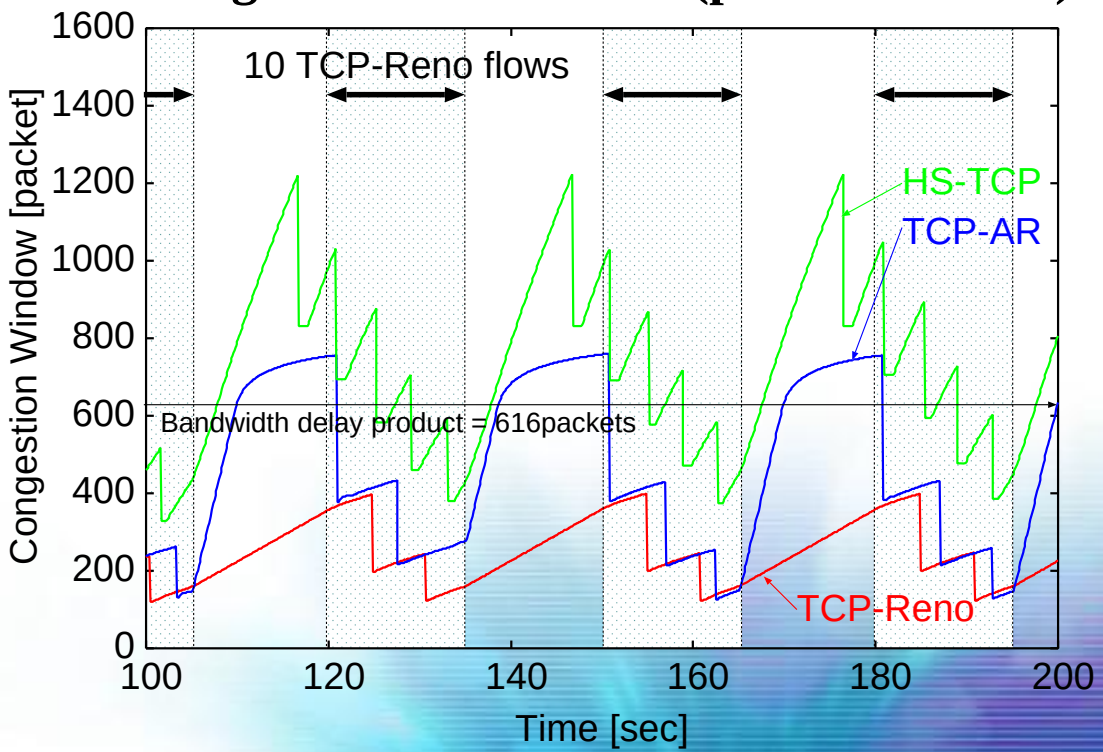
Empowered by Innovation NEC

Coexisting with on-off flows (1)

- One persistent flow: TCP-Reno, or HS-TCP, or TCP-AReno
- 10 background on-off flows: TCP-Reno



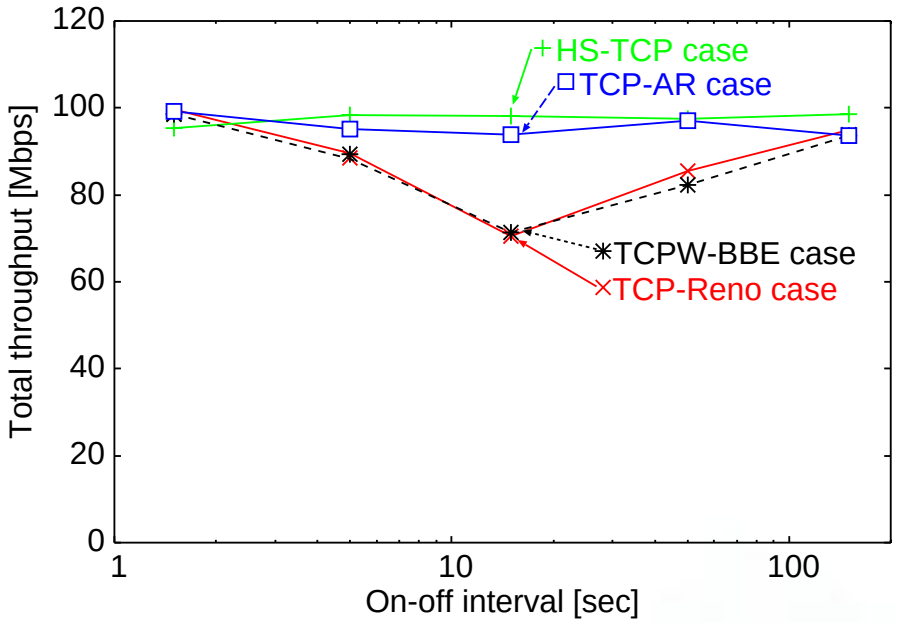
Congestion window size (persistent flow)



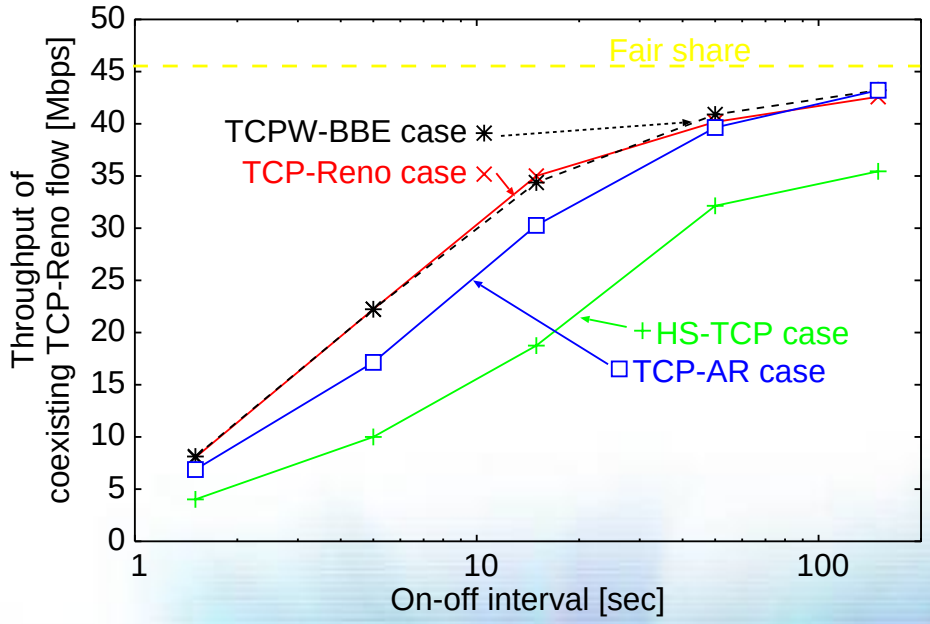
Empowered by Innovation NEC

Coexisting with on-off flows (2)

Total throughput
(1 persistent and 10 on/off flows)

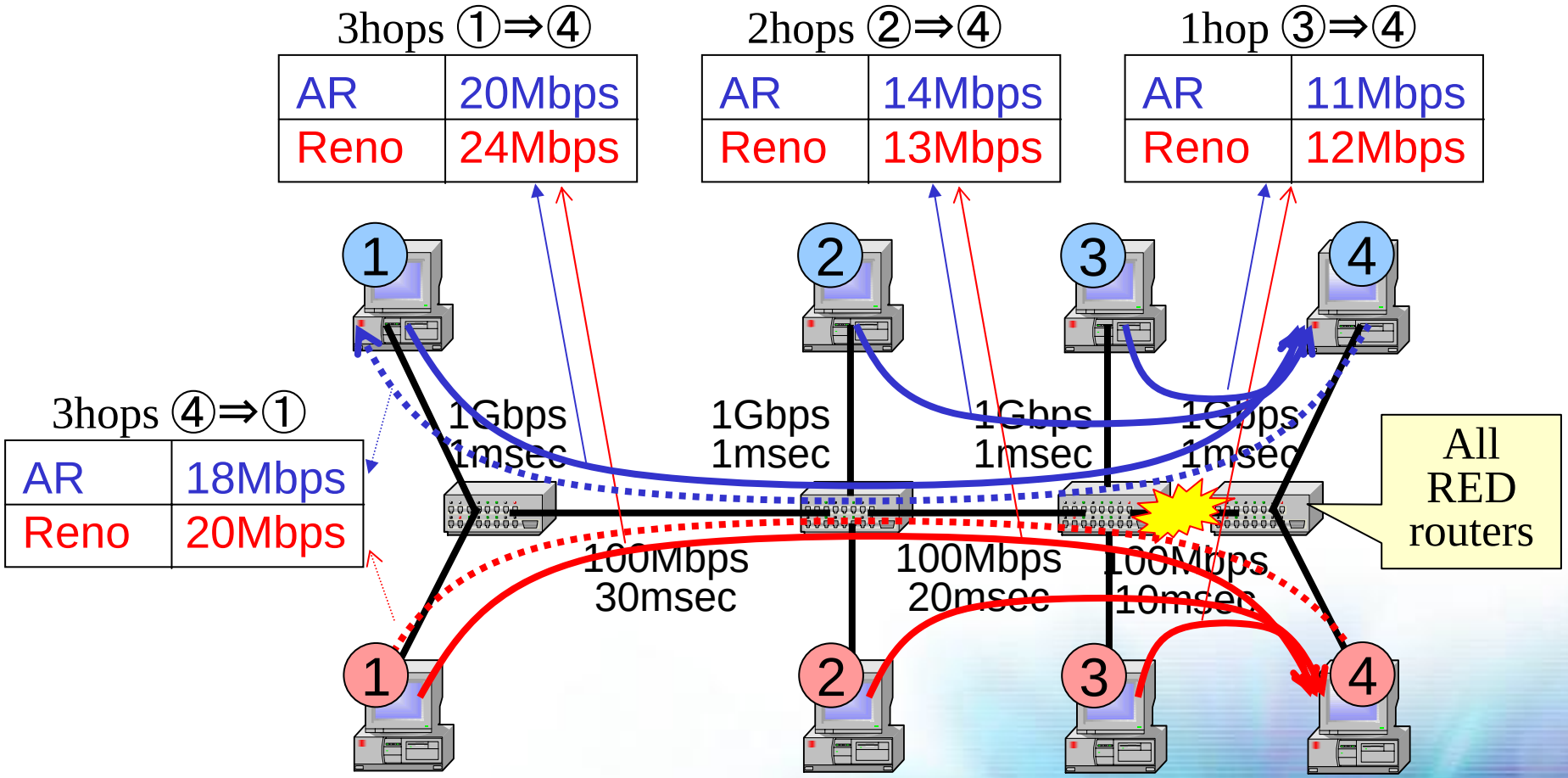


Aggregated throughput
of 10 on/off flows



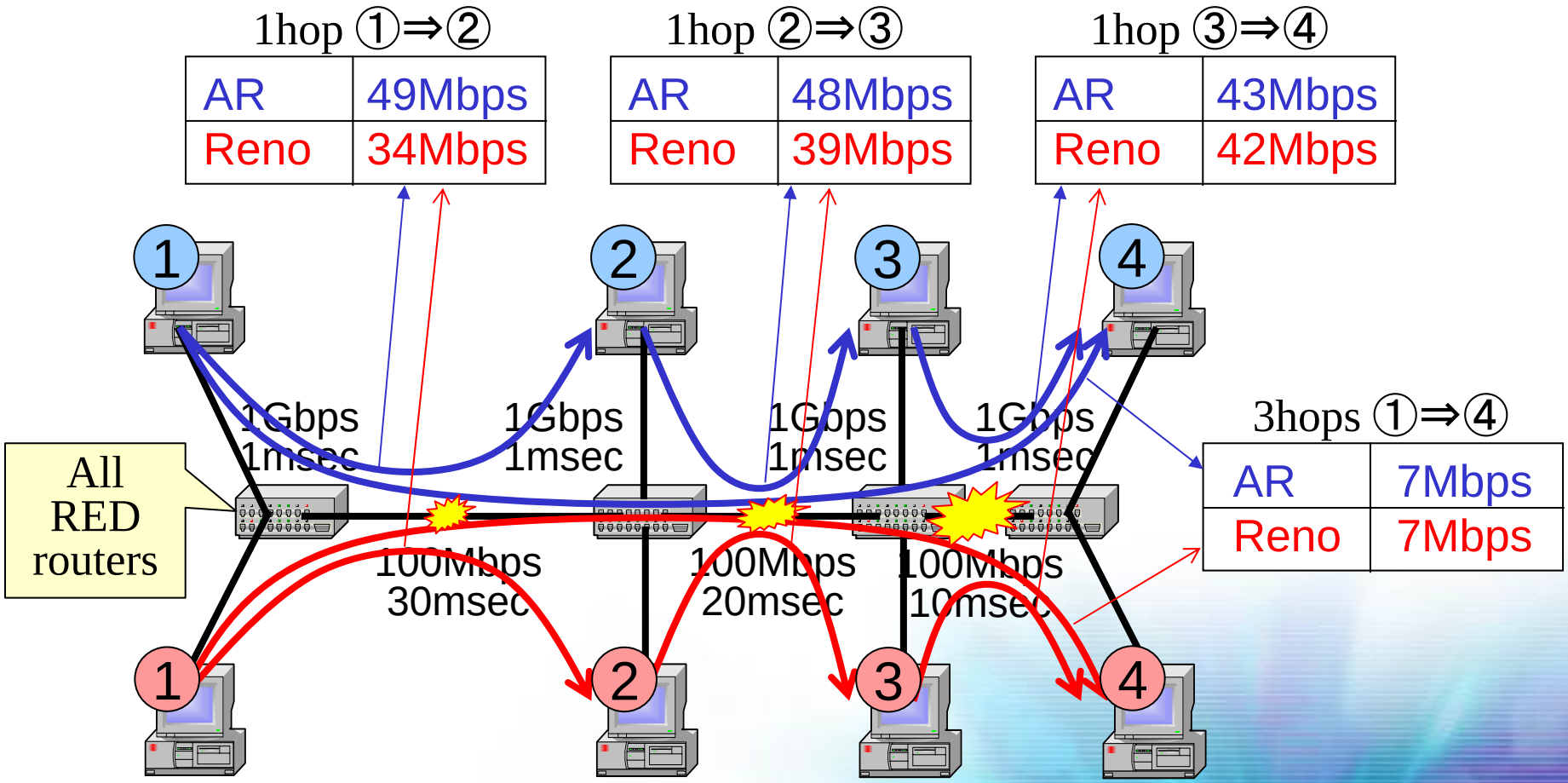
Empowered by Innovation NEC

Reverse traffic case



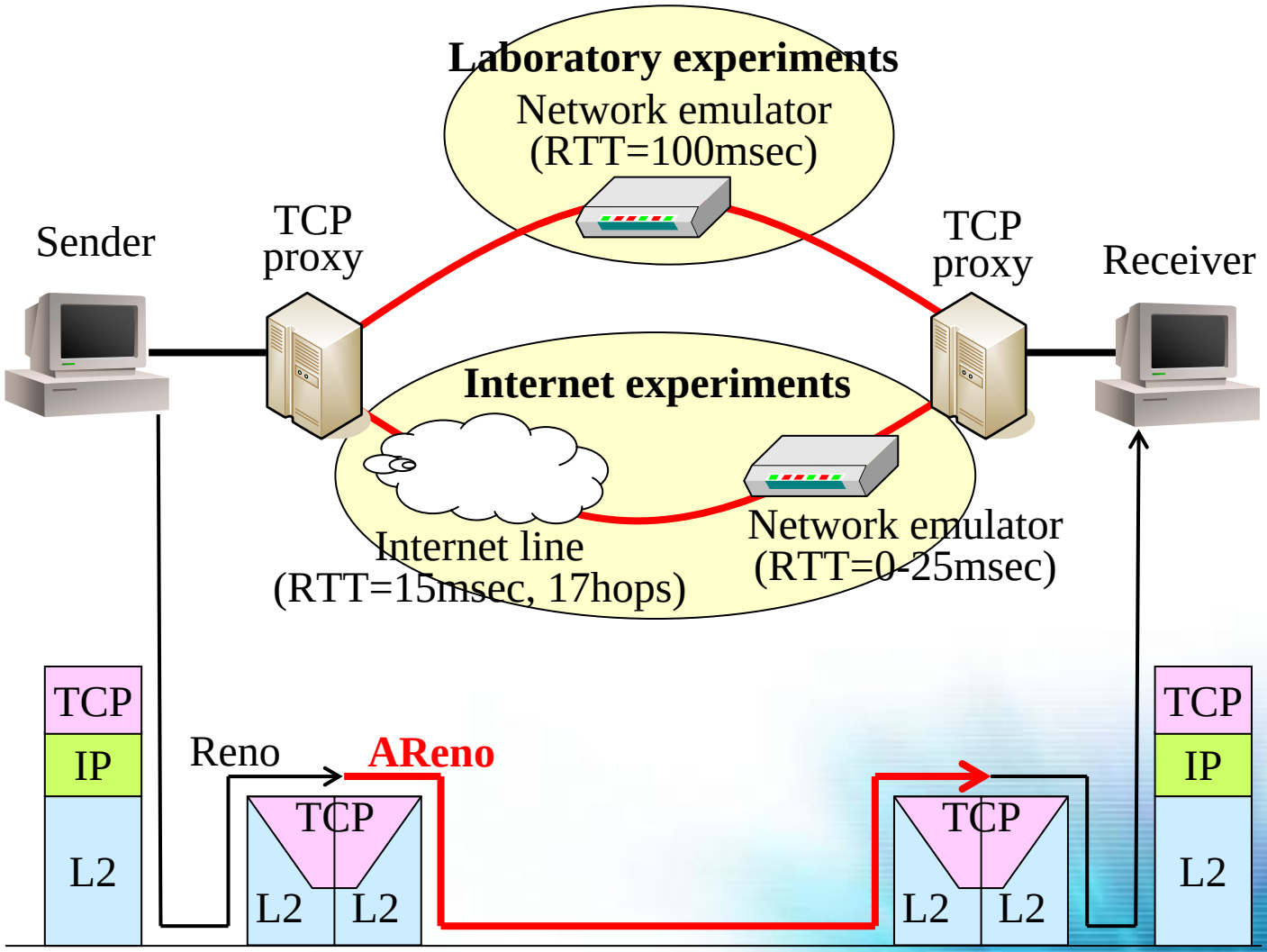
Empowered by Innovation NEC

Multiple bottleneck case



Empowered by Innovation NEC

Experimental study

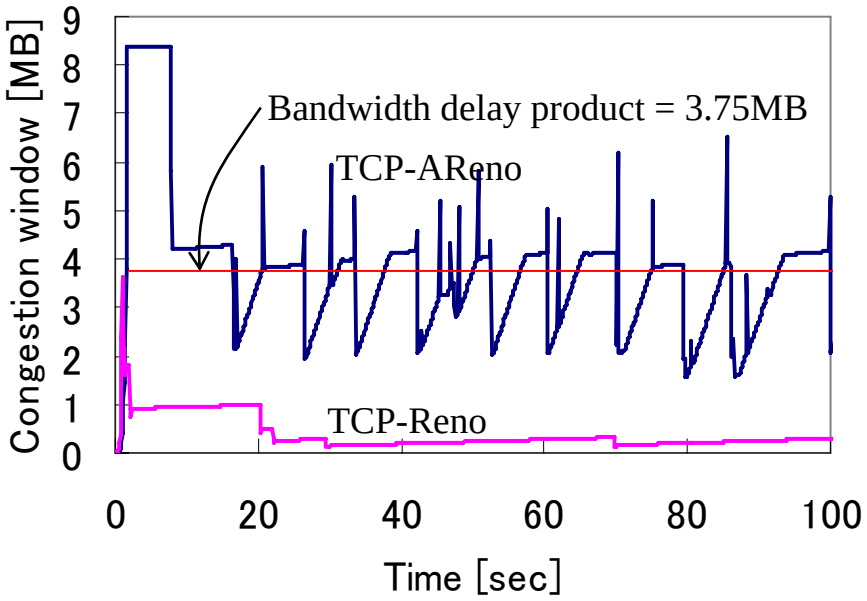


Empowered by Innovation NEC

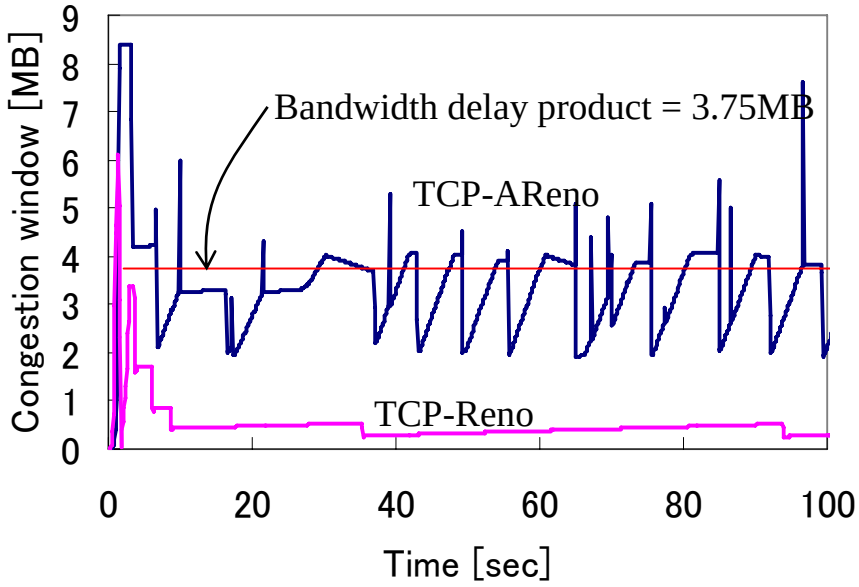
Laboratory measurements (1)

- 100Mbps bottleneck
- 10^{-5} random packet loss
- 100ms RTT

Separate
TCP-Reno or TCP-AReno



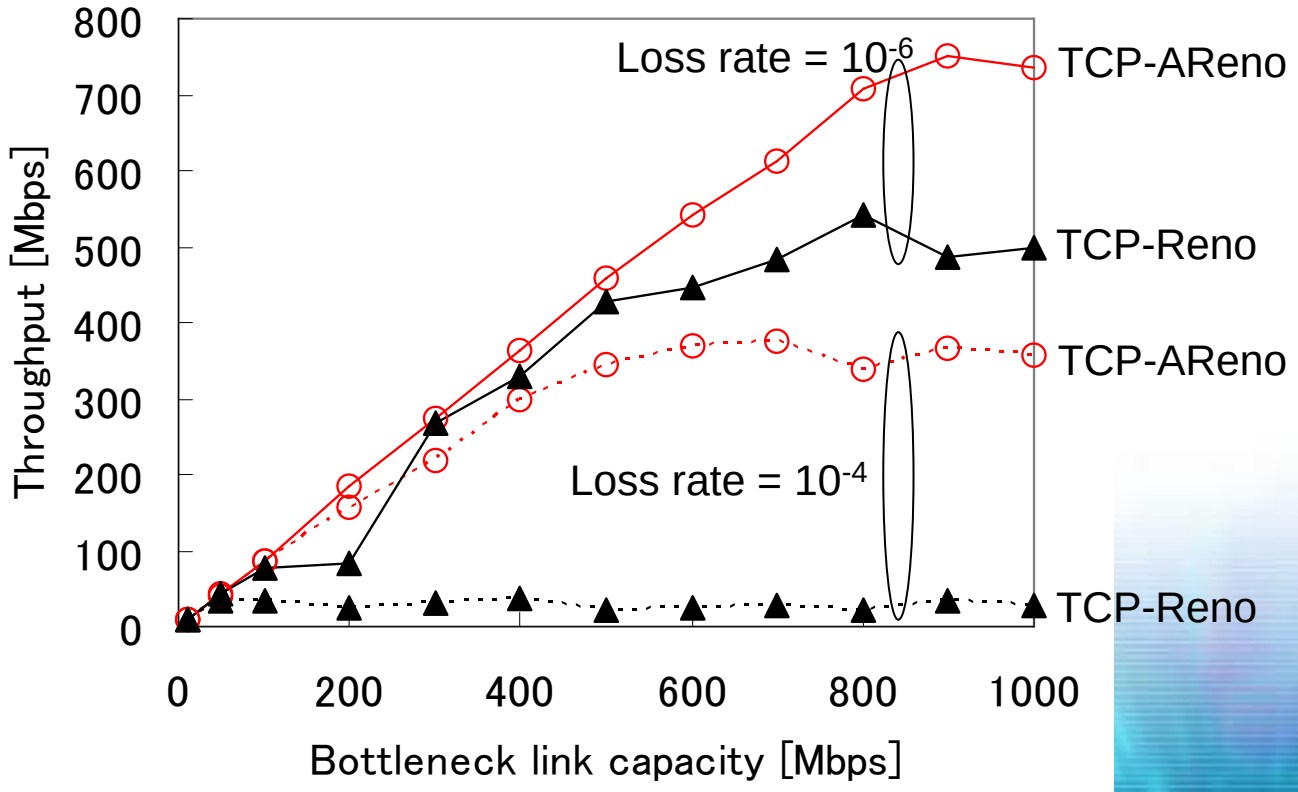
Competing
TCP-Reno v.s. TCP-AReno



Empowered by Innovation NEC

Laboratory measurements (2)

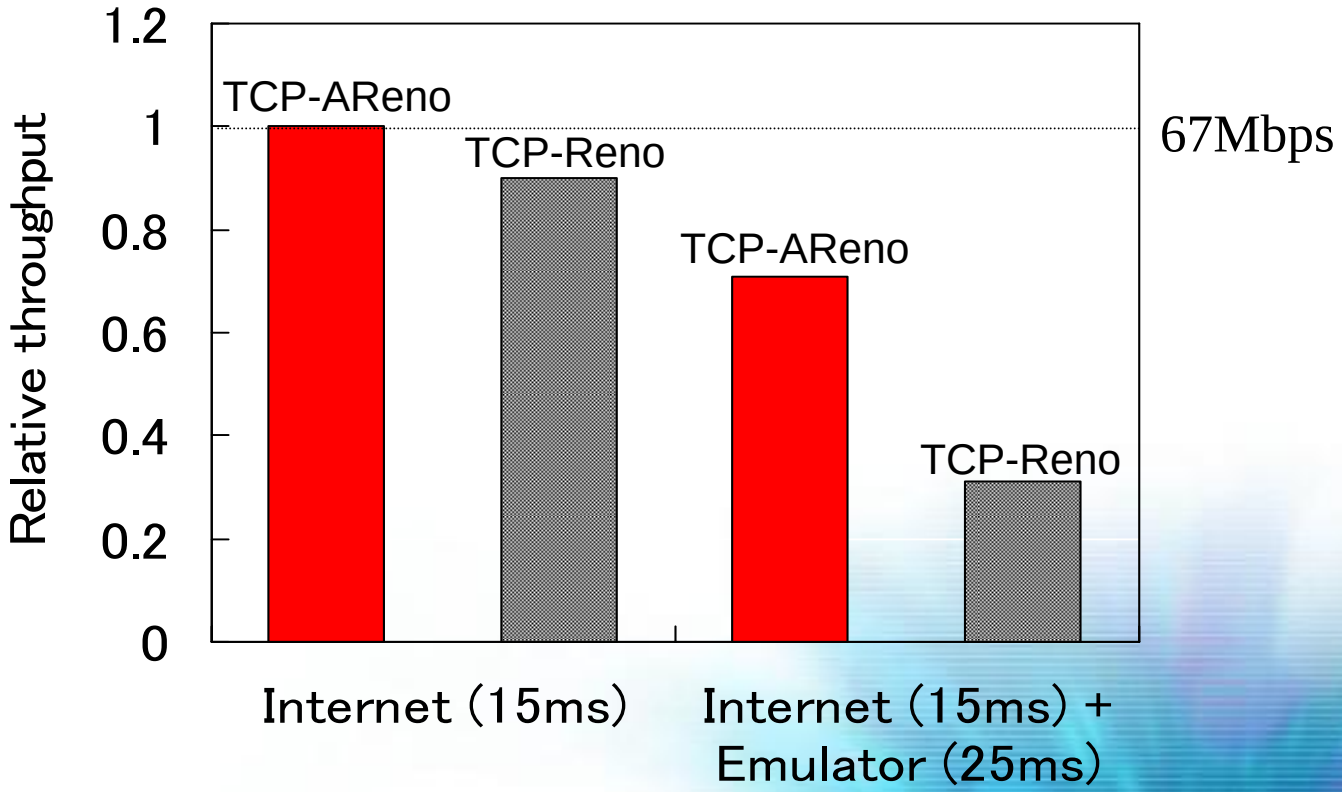
- A lone TCP-Reno or TCP-AReno
- 20ms RTT



Empowered by Innovation NEC

Internet measurements

- 15msec Internet line + 25msec network emulator
- No additional random packet losses at emulator



Empowered by Innovation NEC

Summary

- TCP Adaptive Reno
- Rely on congestion estimation, via buffer estimation
- Whenever it finds congestion, just behaves like Reno
  Otherwise, goes faster

Empowered by Innovation NEC