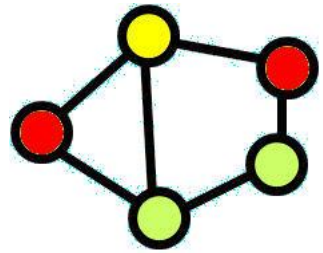
Slide supporting material



Lesson 16: Different TCP Versions, Analytical Details and Implementation

Giovanni Giambene

Queuing Theory and Telecommunications: Networks and Applications
2nd edition, Springer

Different TCP Versions

TCP Congestion Control Design Goals

- **Efficiency**
  - TCP should achieve a high goodput by efficiently using network resources
- **Fairness**
  - **Intra- and inter- protocols**
    - All TCP flows sharing the same bottleneck link should have the same percentage utilization of the bottleneck link
    - **Friendliness** is a concept similar to fairness, but applied to different protocols (e.g., different TCP versions)
  - **RTT**
    - Intra-TCP protocol fairness should also be achieved among competing TCP flows with different RTTs
- **Stability**
  - The TCP cwnd behavior should reach a steady state.

Historical Notes on RFCs and main TCP versions

- **1981**: The basic/initial RFC for TCP is RFC 793. In this version, **there is not cwnd, but only rwnd**. When a packet loss occurs we have to wait for an RTO expiration, to recover the packet loss according to a Go-Back-N scheme.
- **1986**: **Slow Start and Congestion Avoidance algorithms defined by Van Jacobson** and firstly supported by **TCP Berkeley** version.
  - V. Jacobson, "Congestion Avoidance and Control", *Computer Communication Review*, Vol. 18, No. 4, pp. 314-329, August 1988.
- **1988**: Slow Start, Congestion Avoidance, and **Fast Retransmit** (3 DUPACKs) supported by **TCP Tahoe**. Van Jacobson first implemented TCP Tahoe in the 1988 BSD release (BSD stands for Berkeley Software Distribution, a computing library used by UNIX systems).
- **1990**: Slow Start, Congestion Avoidance, **Fast Retransmit, and Fast Recovery** supported by **TCP Reno** (RFC 2001). In 1990, Van Jacobson first implemented TCP Reno in the 4.3BSD Reno release.

Historical Notes on RFCs and main TCP versions (cont'd)

- **1996**: Use of the **SACK option** for the selective recovery of packet losses according to RFC 2018, followed then by RFC 2883.

- **1999**: RFC 2582 is the first RFC describing **TCP NewReno**, then substituted by RFC 3782. RFC 2582 also includes the slow-but-steady and impatient variants of TCP NewReno with a differentiated management of RTO when **multiple packet losses occur in a window of data**.

- **2004**: RFC 3782 describes an improved TCP NewReno version (the careful variant) with **a better management of retransmissions after an RTO expiration**.

TCP Reno

- TCP Reno was defined by Van Jacobson in 1990 (RFC 2001). As soon as **three duplicated ACKs (DUPACKs) are received (i.e., four identical ACKs are received)**, a segment loss is assumed and a Fast Retransmit / Fast Recovery (FR/FR) phase starts:
  - ssthresh is set to cwnd/2 (i.e., flightsize/2);
  - The last unacknowledged segment is soon retransmitted (fast retransmit);
  - cwnd = ssthresh + ndup, where initially ndup = 3 due to three DUPACKs to start the FR/FR phase. This inflates cwnd by the number of segments that have left the network and that are cached at the receiver.
  - Each time another DUPACK arrives, increment cwnd by the segment size (cwnd = cwnd + 1). This inflates the cwnd for the additional segment, which has left the network. Then, transmit a packet, if allowed by the new cwnd value.
  - When the first non-DUPACK is received (an ACK acknowledging all packets sent or **even a 'partial ACK', acknowledging some progress in the sequence number in the case of multiple packet losses in a window of data**), cwnd is set to ssthresh (window deflation) and the fast recovery phase ends.
  - Then, a new congestion avoidance phase starts.

TCP Reno (cont'd)

- TCP Reno may avoid drastic reduction in goodput when a packet loss occurs (as it occurs with Tahoe).

- **TCP Reno performs well in the presence of sporadic packet losses, but when there are multiple packet losses in the same window of data FR/FR phase can be terminated before recovering all losses (multiple FR/FR phases are used) and an RTO may occur**; this problem has been addressed by the TCP NewReno version.

TCP NewReno

- TCP NewReno is one of the most commonly-used congestion control algorithms. TCP NewReno (initially defined in RFC 2582 and then refined by RFC 3782) is based on **an FR/FR algorithm started when there are 3 DUPACKs**.
- In the presence of **multiple packet losses in a window of data**, RFC 2582 (year 1999) specified a mechanism (called "careful variant"), which **avoids unnecessary multiple FR/FR phases and manages all these losses in a single FR/FR phase**. Then, RFC 3782 (year 2004) has considered the "careful variant" of the FR/FR algorithm as the reference one for TCP NewReno.
- NewReno uses a **'recover' variable**, representing the maximum order of the segment sent when 3 DUPAKCs are received.
  - A **partial ACK** acknowledges some, but not all the outstanding packets at the start of the Fast Recovery phase, as specified in the 'recover' variable.
  - A **full ACK** acknowledges all the outstanding packets at the start of the Fast Recovery phase.

S. Floyd, T. Henderson, A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, 2004.

TCP NewReno (cont'd)

- With TCP Reno, the first partial ACK causes TCP to leave the FR/FR (Fast Recovery) phase by deflating cwnd back to ssthresh. Instead, **with TCP NewReno, partial ACKs do not take TCP out of the FR/FR phase: partial ACKs received during Fast Recovery are treated as an indication that the packet immediately following the acknowledged packet has been lost, and needs to be retransmitted**.
- When multiple segments are lost from a single window of data, NewReno can recover them, retransmitting one segment per RTT until all lost segments from that window are delivered correctly.
- The FR/FR phase is concluded when a **full ACK** is received.
- Then, a new **congestion avoidance phase** is performed with ssthresh equal to half of the cwnd value just before the start of the FR/FR phase.

TCP NewReno Variants

- The **Slow-but-Steady** and **Impatient** variants of NewReno differ in their **Fast Recovery behavior**, specifically with respect to when they reset the RTO timer.
  - The Slow-but-Steady variant **resets timer RTO after receiving each partial ACK** and continues to make small adjustments to the cwnd value. **The TCP sender remains in the FR/FR mode until it receives a full ACK. Typically no RTO occurs.**
  - The Impatient variant **resets timer RTO only after receiving the first partial ACK**. Hence, in the presence of multiple packet losses, the Impatient variant can conclude too long FR/FR phases by allowing timer RTO to expire so that all lost segments are recovered according to a Go-Back-N approach and a slow start phase.
  - In RFC 3782, the **Impatient variant is recommended** over the Slow-but-Steady variant.

Micro-Analysis and Macro-Analysis of TCP Behavior

- Microanalysis is the study of the TCP behavior in terms of cwnd, RTT, RTO, sequence number, and ACK number with **the finest time granularity** (RTT basis) in order to verify the reaction of the TCP protocol to the different cases and conditions.

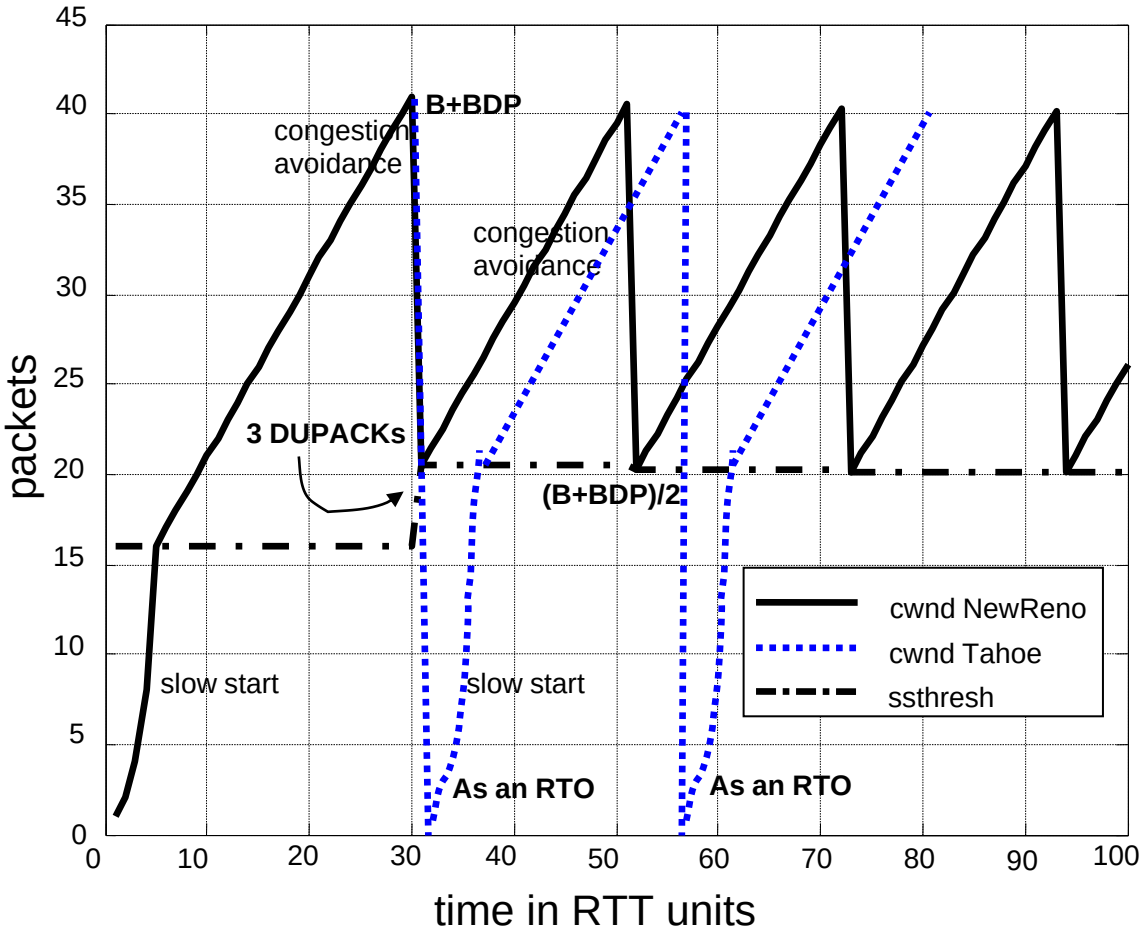
- This study is opposed to the **macroanalysis**, which deals with the evaluation of the macroscopic TCP behavior in terms of long-range time **averages**, such as: average throughput, average goodput, fairness, etc…

Cwnd Sawtooth Behaviors for Tahoe and Reno/NewReno

Hp) Single TCP flow; Sockets buffers (rwnd) > B+BDP; initial ssthresh < B+BDP; no cross-traffic

Th) At regime, cwnd oscillates between B+BDP and (B+BDP)/2 according to a periodic sawtooth behavior.

The pipe is fully-utilized when BDP ≤ cwnd ≤ B+BDP.



B = 20 pkts

BDP = 20 pkts

Initial ssthresh = 16 pkts

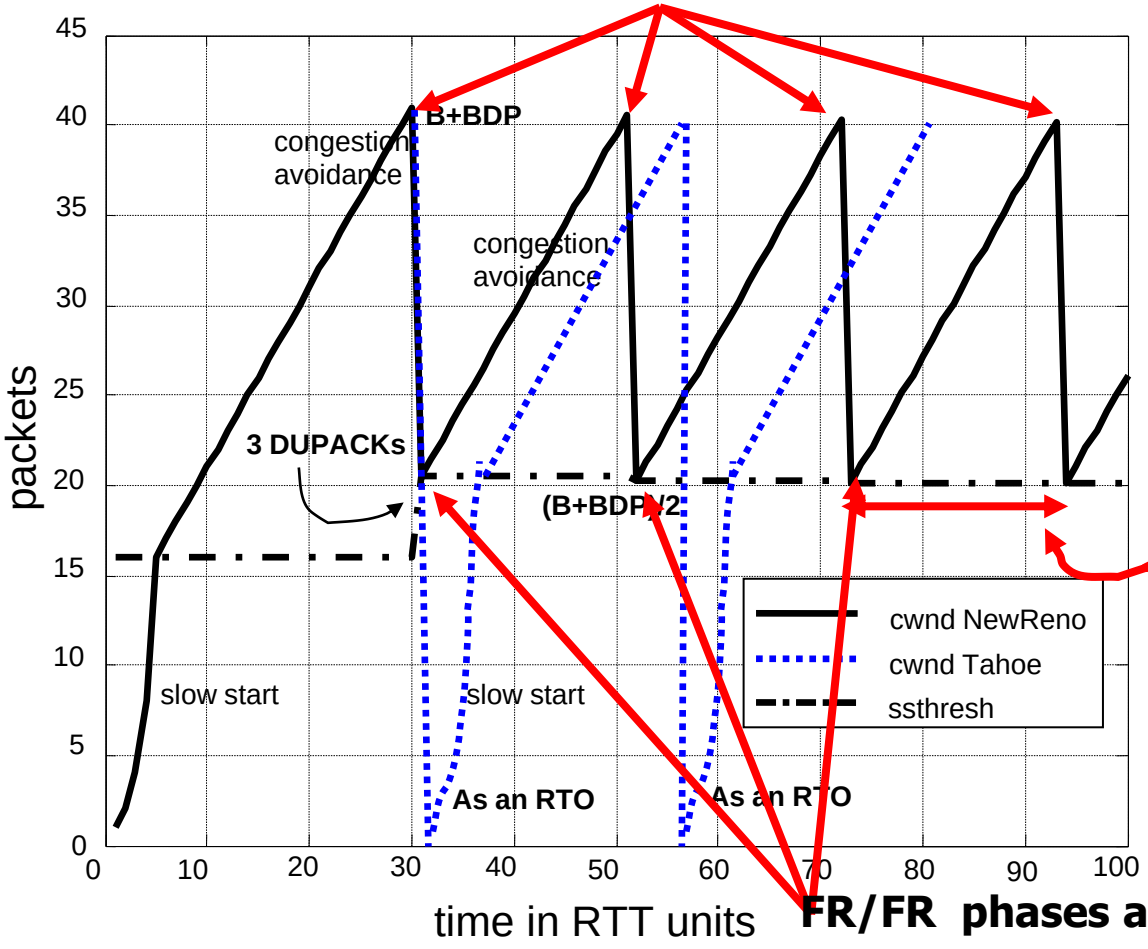
Cwnd Sawtooth Behaviors for Tahoe and Reno/NewReno

Periodical losses due to buffer overflow with mean rate (NewReno): $PLR = \frac{8}{3(B+BDP)^2}$

Hp) Single TCP flow; Sockets buffers (rwnd) > B+BDP; initial ssthresh < B+BDP; no cross-traffic

Th) At regime, cwnd oscillates between B+BDP and (B+BDP)/2 according to a periodic sawtooth behavior.

The pipe is fully-utilized when BDP ≤ cwnd ≤ B+BDP.



The cycle time of cwnd is equal to (B+BDP)/2 in RTT units. In LFN networks this cycle time can be quite long.

FR/FR phases are concentrated in these short intervals

Cwnd Sawtooth Behaviors …

- **If rwnd > B+BDP, the quantity of bits injected by the source** up to time t, $\alpha(t)$, due to the TCP protocol can be approximately determined as the integral of cwnd as a function of time:

$$\alpha(t) = \int_0^t cwnd(t)dt \quad [\text{bits}]$$

  - $\alpha(t)$ is the **arrival curve**.

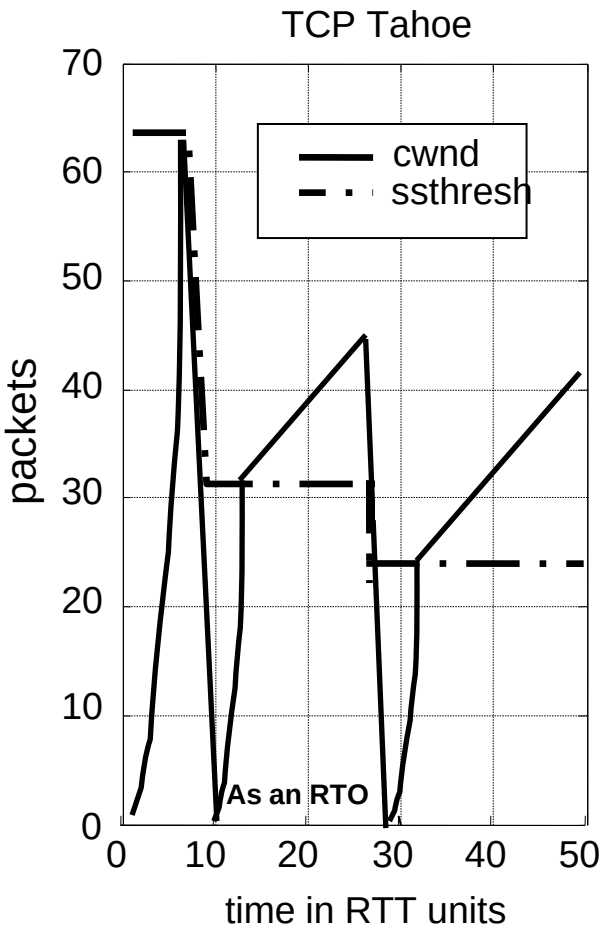
- If the **initial ssthresh value is bigger than BDP+B** (actually just BDP if it is the actual BDP based on RTT), the initial slow start phase causes a significant traffic injection well beyond the capacity of the network. This entails that **the slow start phase ends with multiple packet losses, a drop of cwnd, and possible RTO expiration**.

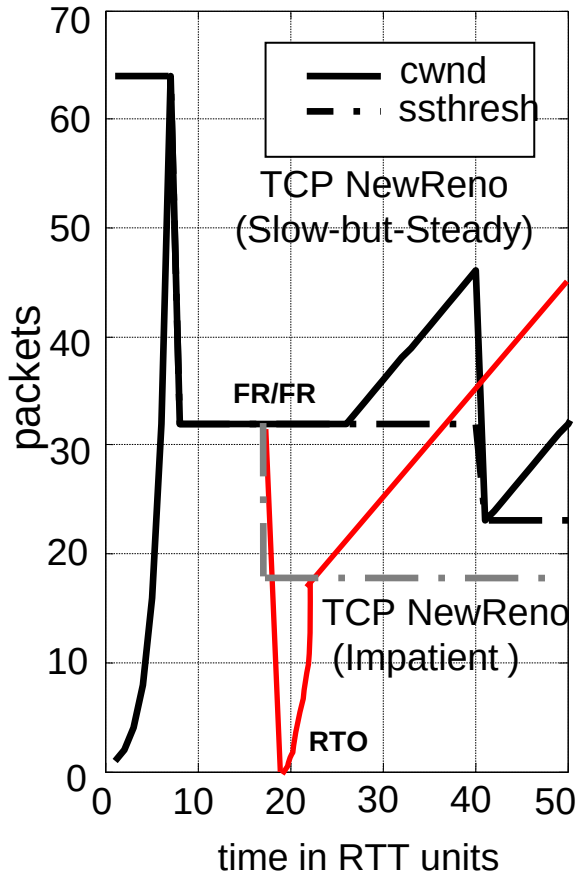
Cwnd Sawtooth Behaviors for High Initial ssthresh

Hp) Single TCP flow; sockets buffers (rwnd) > B+BDP; initial ssthresh >> B+BDP; no cross-traffic

Th) The initial slow-start phase experiences many packet losses.

Impatient version: if RTO = 2xRTTs = 1 s (GEO satellite scenario), there is an RTO expiration if there are more than 3 packet losses in a window of data.





The behaviors of these graphs are indicative.

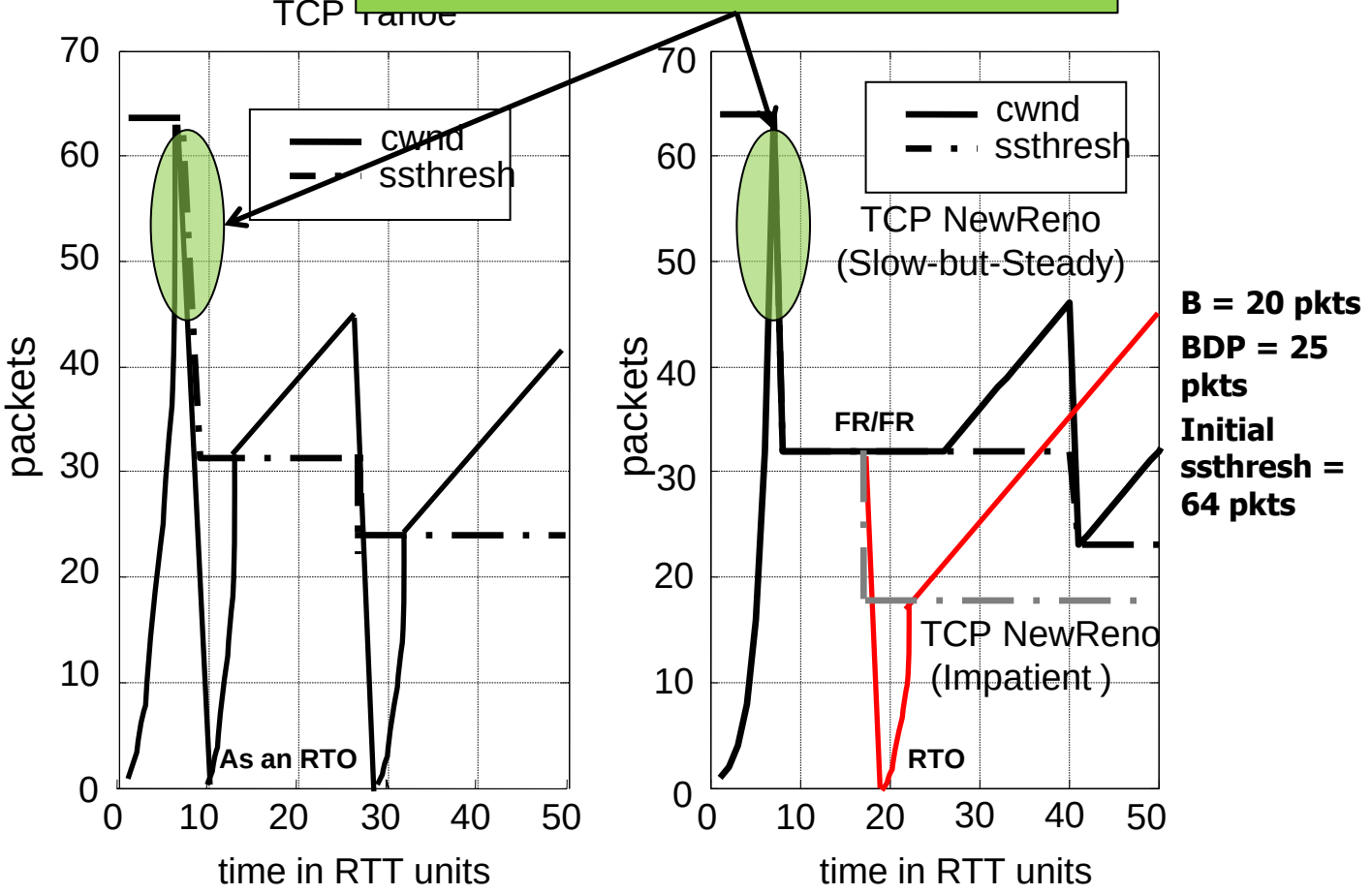
Cwnd Sawtoo
for High Initia

Traffic peaks with multiple packet losses at both the socket buffer and the bottleneck link buffer.

Hp) Single TCP flow; sockets buffers (rwnd) > B+BDP; initial ssthresh >> B+BDP; no cross-traffic

Th) The initial slow-start phase experiences many packet losses.

Impatient version: if RTO = 2xRTTs = 1 s (GEO satellite scenario), there is an RTO expiration if there are more than 3 packet losses in a window of data.



The behaviors of these graphs are indicative.

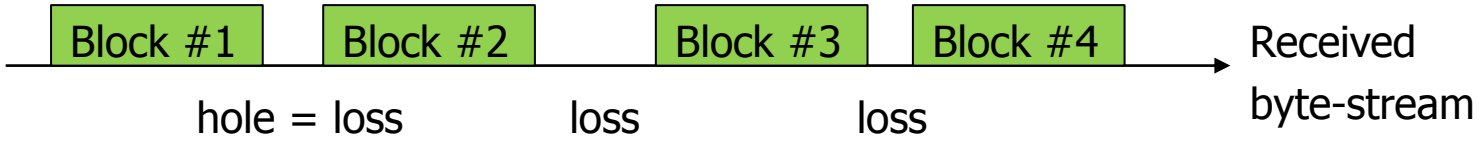
TCP with SACK Option

- **TCP Reno and NewReno retransmit at most 1 lost packet per RTT during the FR/FR phase**, so that the pipe can be inefficiently used during the recovery phase in the presence of multiple losses.
- With Selective ACK (SACK) enabled (RFCs 2018 and 2883), **the receiver informs the sender about all successfully-received segments: the sender only retransmits lost segments**.
- Support for SACK is negotiated at the beginning of a TCP connection between sender and receiver. Both sender and receiver need to agree on the use of SACK: **use of the SACK-permit option in the three-way handshake phase**. SACK does not change the meaning of the ACK field in TCP segments.
- **A contiguous group of correctly-received bytes represents a block**; bytes just below the block and just above the block have not been received.

M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgement Options", RFC 2018, Oct. 1996
K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP", *Computer Communication Review*, July 1996

TCP with SACK Option (cont'd)

- **The SACK option has to be sent by the receiver** to inform the sender of non-contiguous blocks of data received and queued.
- **If SACK is enabled, SACK options should be used in all ACKs not ACKing the highest sequence number in the receiver queue.** A SACK option in the TCP header can permit to specify a maximum of **4 blocks**.

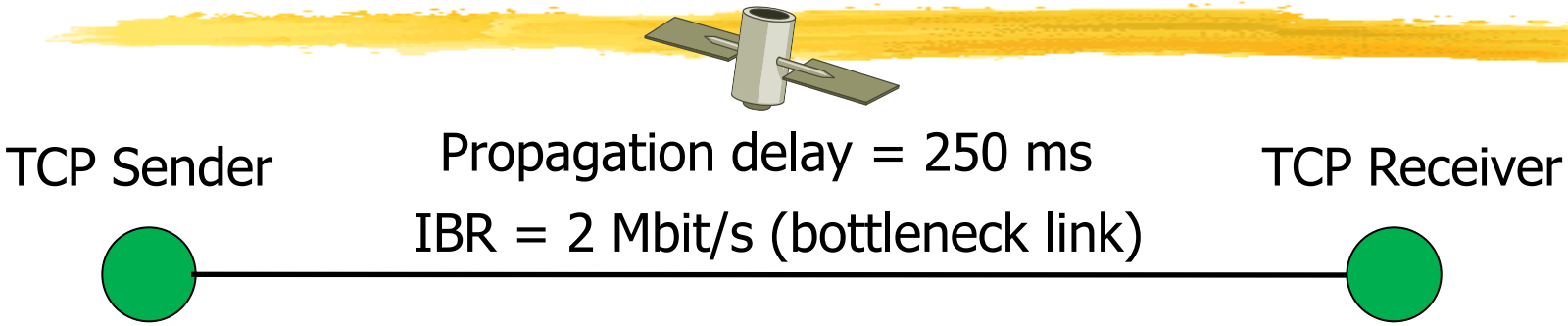


- The implementation of **SACK combined with TCP Reno** by S. Floyd requires a new state variable called 'pipe'.
- **Whenever the sender enters the fast recovery phase (after 3 DUPACKs received), it initializes 'pipe', as an estimate of how many packets are outstanding in the network, and sets cwnd to half of its current value.**

TCP with SACK Option (cont'd)

- If pipe > cwnd, no packet can be sent, since the number of in-flight data is larger than the cwnd value.
- Pipe is decremented by 1 when the sender receives a partial ACK with a SACK option reporting that new data have been received.
- Whenever pipe becomes lower than cwnd, it is possible to send packets, starting from the missing ones (holes as reported by SACK) and then new ones. Thus, **more than one lost packet can be sent in one RTT**.
- Pipe is incremented by 1 when the sender sends a new packet or retransmits an old one.
- **Exit fast recovery when a full ACK is received**.

Example of NewReno Micro-Analysis (GEO Satellite Case)



$$RTT \approx RTD = 2 \times propagation\ delay = 500\ ms$$

$$MTU = 1500\ \text{bytes}$$

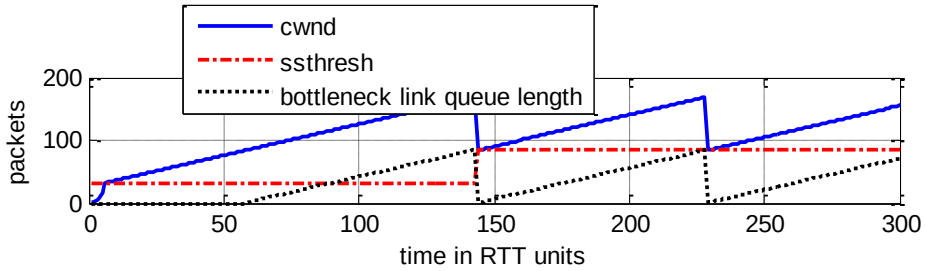
$$initial\ ssthresh = 32\ \text{pkts}$$

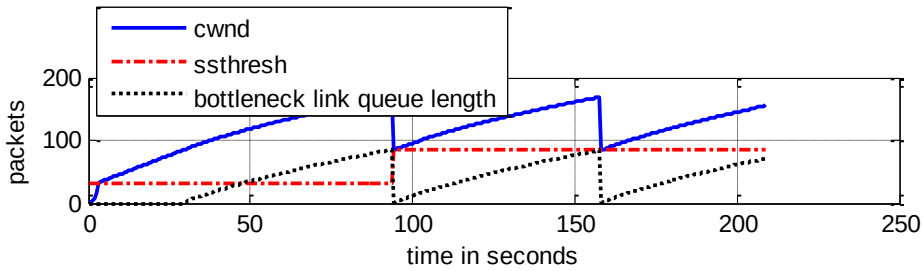
$$BDP = \frac{RTT \times IBR}{MTU \times 8} = 84\ pkts$$

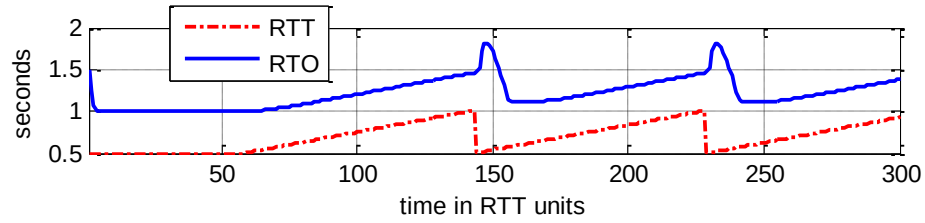
Example of NewReno Micro-Analysis

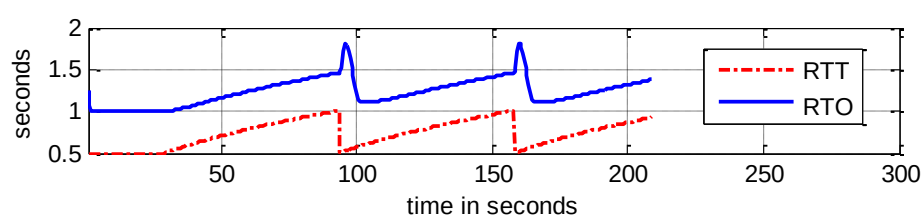
B = 84 pkts

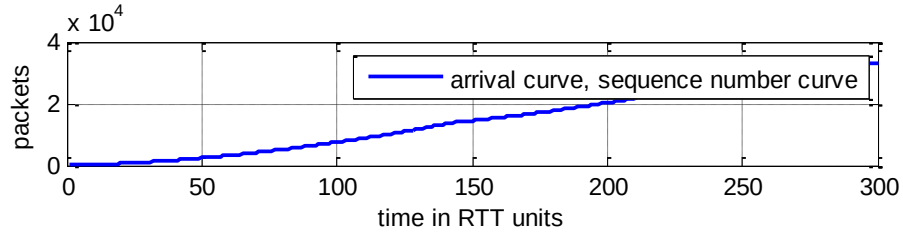
$cwnd_{max} = BDP + B = 168$ pkts

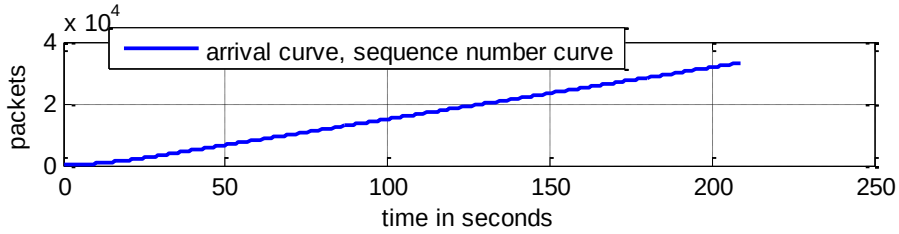










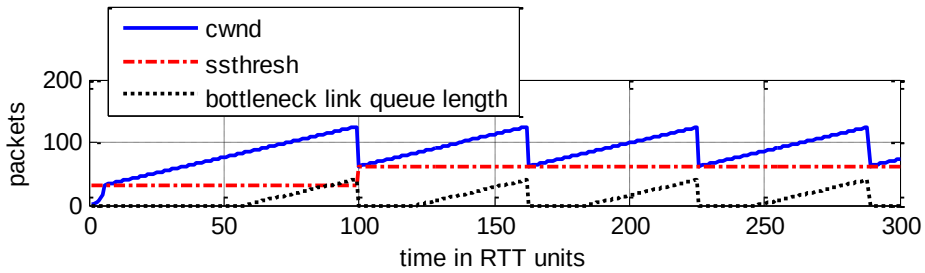


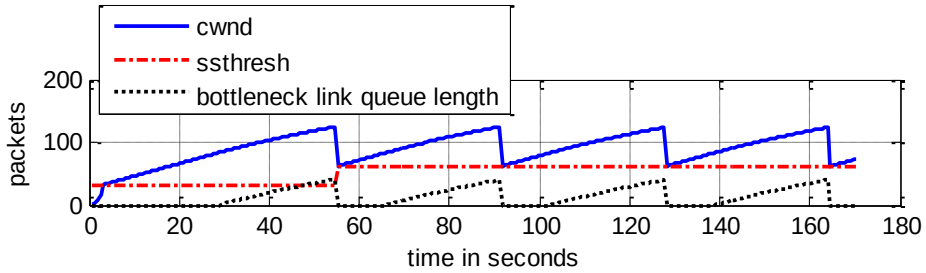
We have not considered here the RTT granularity due to ticks.

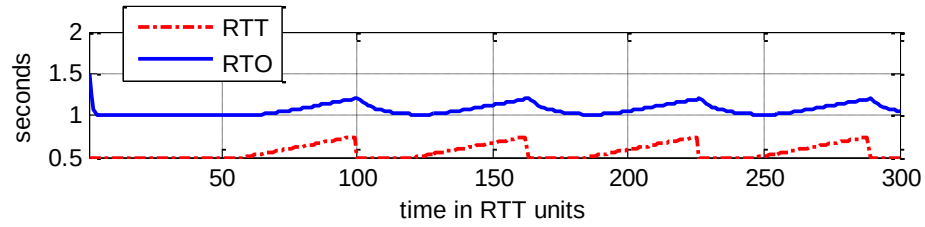
Example of NewReno Micro-Analysis

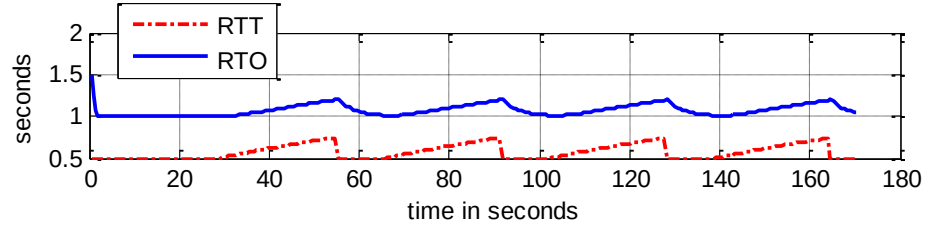
B = 40 pkts

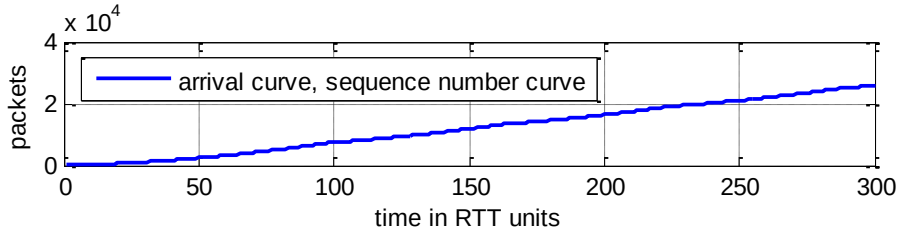
$cwnd_{max}$ = BDP + B = 124 pkts

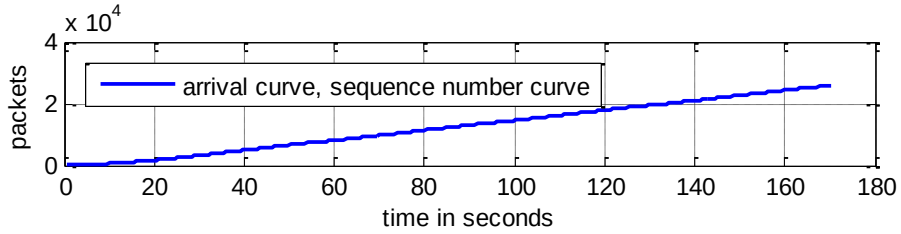










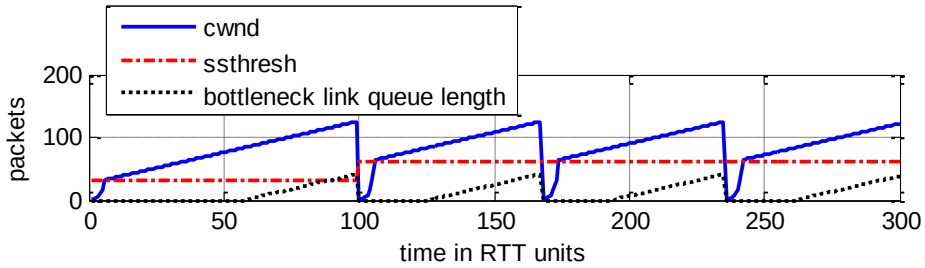


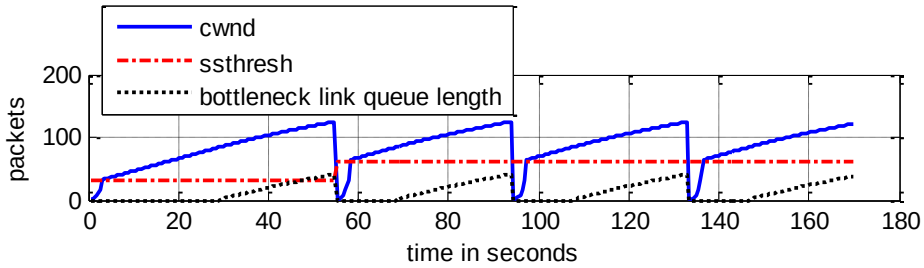
We have not considered here the RTT granularity due to ticks.

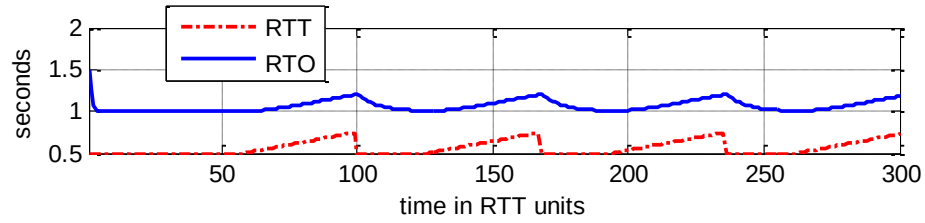
Example of Tahoe Micro-Analysis

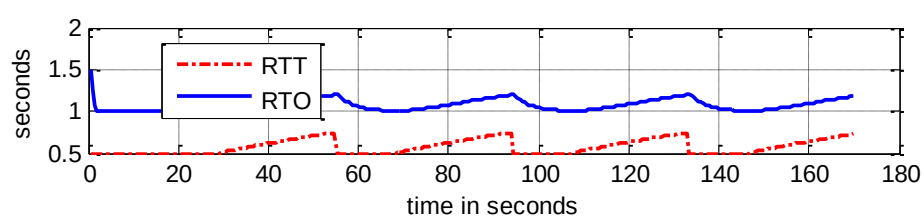
B = 40 pkts

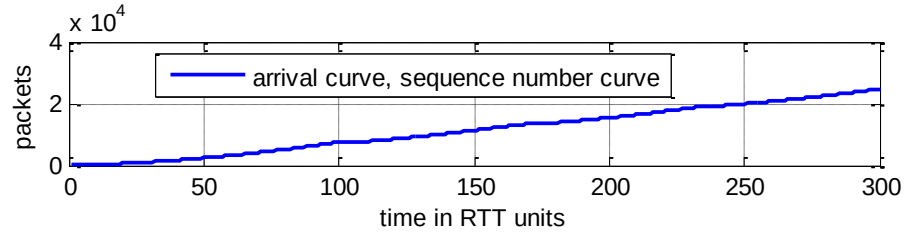
$cwnd_{max}$ = BDP + B = 124 pkts

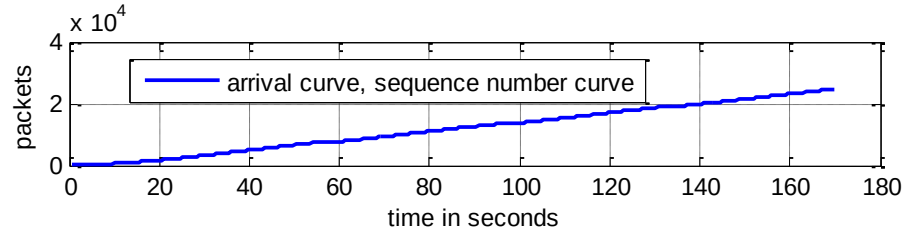












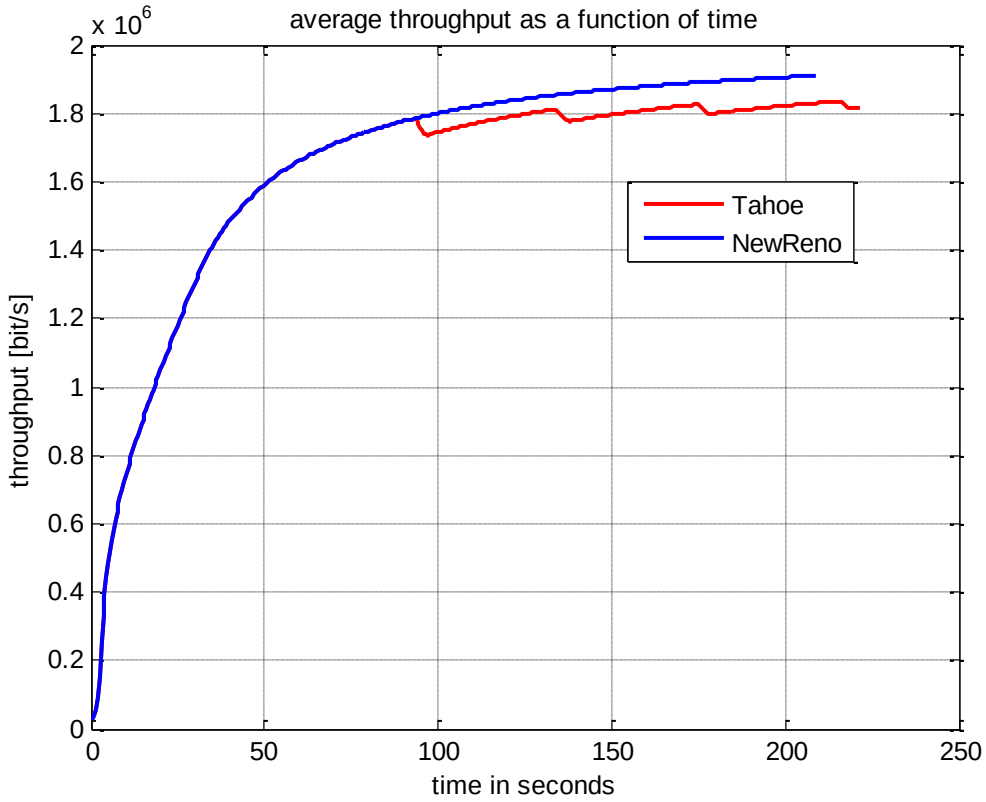
We have not considered here the RTT granularity due to ticks.

Example of Tahoe/NewReno Macro-Analysis: avg. Th.

The average throughput is derived as sum of cwnds on RTT basis divided by the total time elapsed:

$$\Gamma(n) = \frac{\sum_{i=1}^{n} cwnd_i}{\sum_{i=1}^{n} RTT_i}$$

B = 84 pkts

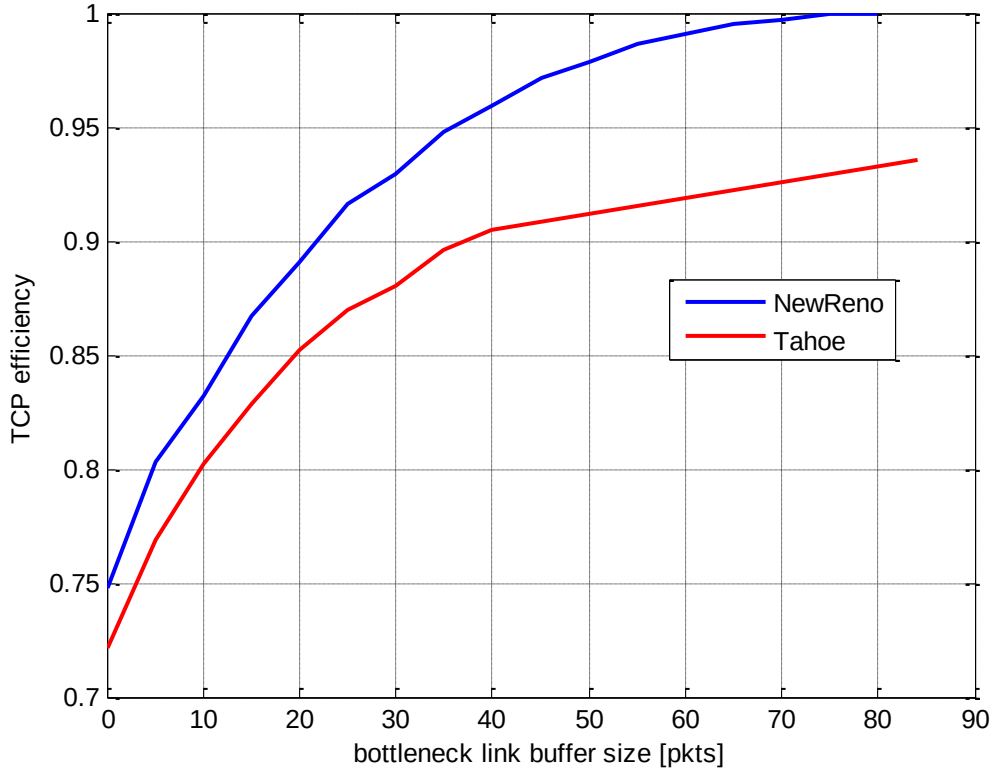


Example of Tahoe/NewReno Macro-Analysis: Efficiency

Study of the efficiency as a function of the bottleneck link buffer size from B = 0 to B = BDP = 84 pkts

$$\eta = \frac{\Gamma(B)}{IBR}$$

When B = 0, the efficiency of TCP NewReno is minimum, 75%. When B tends to BDP, the efficiency tends to 100% with TCP NewReno.



Design of the Buffer of the Bottleneck Link

- **The optimal buffer B value** is the minimum B value allowing to maintain the pipe constantly filled so that **cwnd never goes below BDP (i.e., the pipe never becomes empty**, and the link is exploited at the maximum rate of IBR); **a rule-of-thumb is to consider B = BDP packets**.
  - **At regime, cwnd of NewReno oscillates between 2BDP and BDP, the pipe is always loaded at about IBR, and the buffer occupancy oscillates between full and empty conditions.**
  - **ssthresh is the TCP memory of recent congestion events. At regime, ssthresh is equal to BDP. The regime ssthresh value can represent an estimate of the system bandwidth.**

Design of the Buffer of the Bottleneck Link

BDP ≤ cwnd ≤ B+BDP

- **The optimal buffer B value** is the minimum value allowing to maintain the pipe constantly filled so that **cwnd never goes below BDP (i.e., the pipe never becomes empty**, and the link is exploited at the maximum rate of IBR); **a rule-of-thumb is to consider B = BDP packets**.
  - **At regime, cwnd of NewReno oscillates between 2BDP and BDP, the pipe is always loaded at about IBR, and the buffer occupancy oscillates between full and empty conditions.**
  - **ssthresh is the TCP memory of recent congestion events. At regime, ssthresh is equal to BDP. The regime ssthresh value can represent an estimate of the system bandwidth.**

TCP Analysis

Square-Root Formula for TCP Throughput/Goodput

- **At regime,** the average TCP throughput Γ (the average goodput γ) **at network layer** can be approximated by the square-root formula below, which is valid under the following assumptions: **B = 0**, **RTT = constant (i.e., RTT ≡ RTD)**, and **neglecting RTO events.**

$$\Gamma = \min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \quad \left[\frac{\text{bit}}{\text{s}}\right] \qquad \gamma = (1-p)\times\min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \quad \left[\frac{\text{bit}}{\text{s}}\right]$$

where **p (p < 0.1, otherwise RTOs have impact) denotes the segment loss rate**, α is a coefficient, which depends on the TCP version and type of losses (e.g., $\sqrt{\alpha} = 1.31$ **for NewReno with random losses**).

- Throughput/goodput of standard TCP is quite sensitive to the increase in p.

M. Mathis, J.Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, Vol. 27, No. 3, July 1997.

Square-Root Formula for TCP Throughput/Goodput

MTU is here measured in bytes and RTT is here expressed in seconds.

- **A** P throughput Γ (the average goodput γ) **a** pproximated by the square-root formula b he following assumptions: **B = 0**, **RTT = con t (i.e., RTT ≡ RTD)**, and **neglecting RTO events.**

$$\Gamma = \min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \left[\frac{\text{bit}}{\text{s}}\right] \quad \gamma = (1-p)\times\min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \left[\frac{\text{bit}}{\text{s}}\right]$$

where **p (p < 0.1, otherwise RTOs have impact) denotes the segment loss rate**, α is a coefficient, which depends on the TCP version and type of losses (e.g., $\sqrt{\alpha} = 1.31$ **for NewReno with random losses**).

- Throughput/goodput of standard TCP is quite sensitive to the increase in p.

M. Mathis, J.Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, Vol. 27, No. 3, July 1997.

Square-Root Formula for TCP Throughput/Goodput

The minimum is needed to avoid that a too low p value causes this quantity to go beyond the physical limit of IBR.

- CP throughput Γ (the average goodput γ) approximated by the square-root formula r the following assumptions: **B = 0**, **RTT = c ant (i.e., RTT ≡ RTD)**, and **neglecting RTO events.**

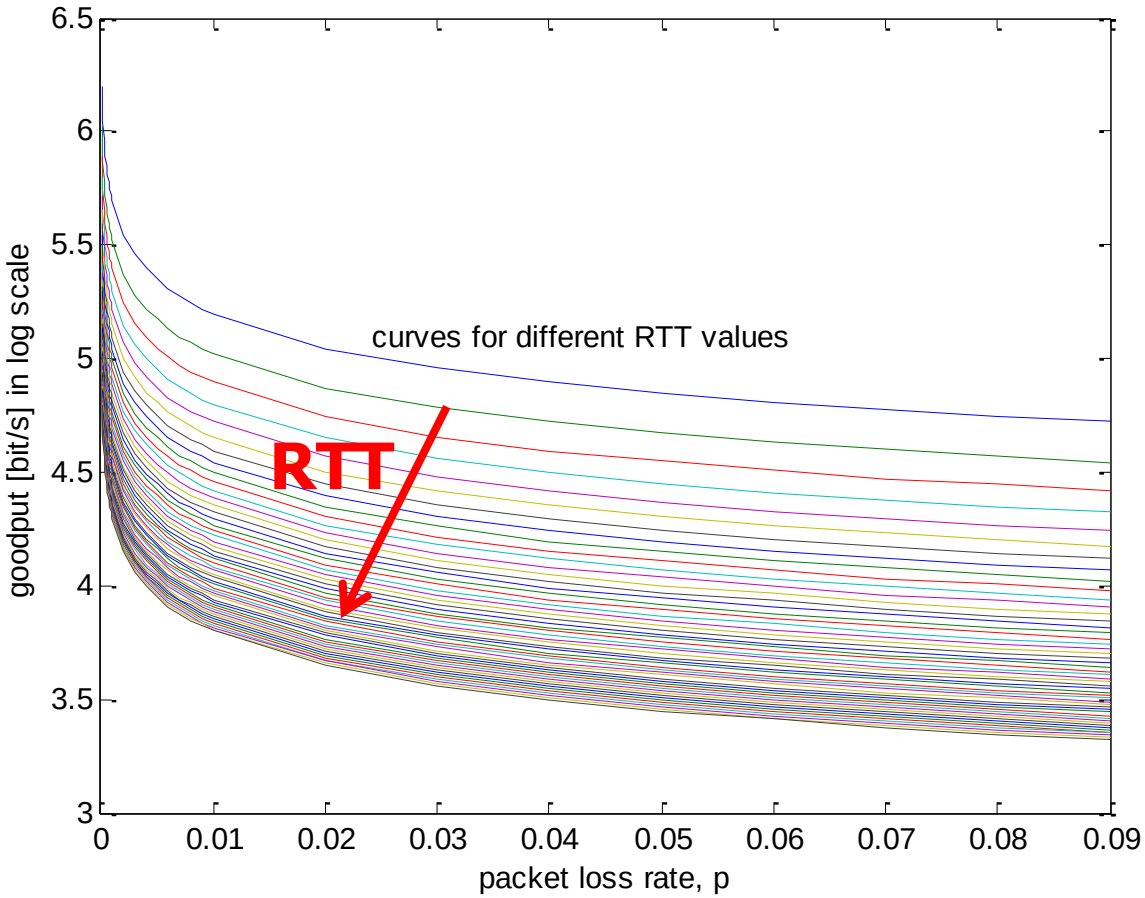
$$\Gamma = \min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \quad \left[\frac{\text{bit}}{\text{s}}\right] \qquad \gamma = (1-p)\times\min\left\{\frac{MTU \times 8}{RTT}\sqrt{\frac{\alpha}{p}}, IBR\right\} \quad \left[\frac{\text{bit}}{\text{s}}\right]$$

where **p (p < 0.1, otherwise RTOs have impact) denotes the segment loss rate**, α is a coefficient, which depends on the TCP version and type of losses (e.g., $\sqrt{\alpha} = 1.31$ **for NewReno with random losses**).

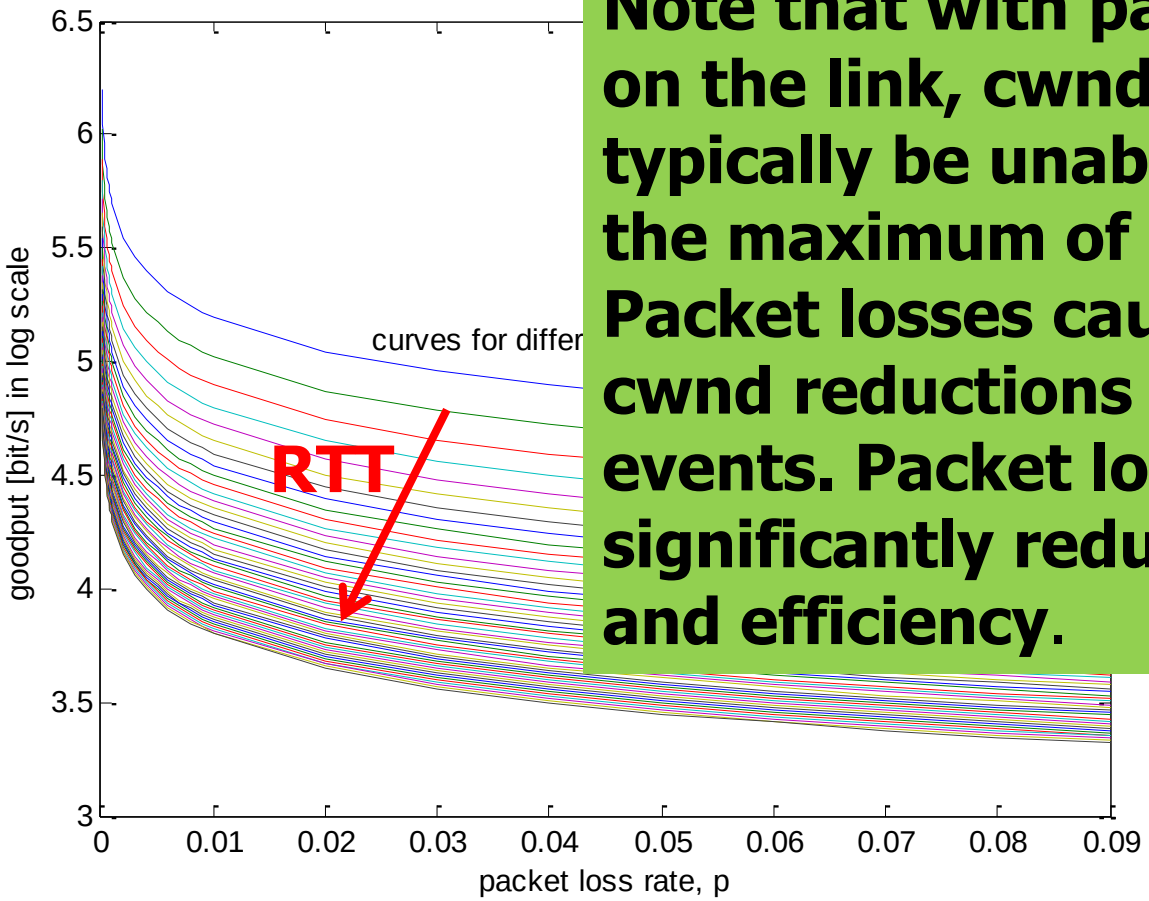
- Throughput/goodput of standard TCP is quite sensitive to the increase in p.

M. Mathis, J.Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, Vol. 27, No. 3, July 1997.

Square-Root Formula for TCP Throughput/Goodput (cont'd)



Square-Root Formula for TCP Throughput/Goodput (cont'd)

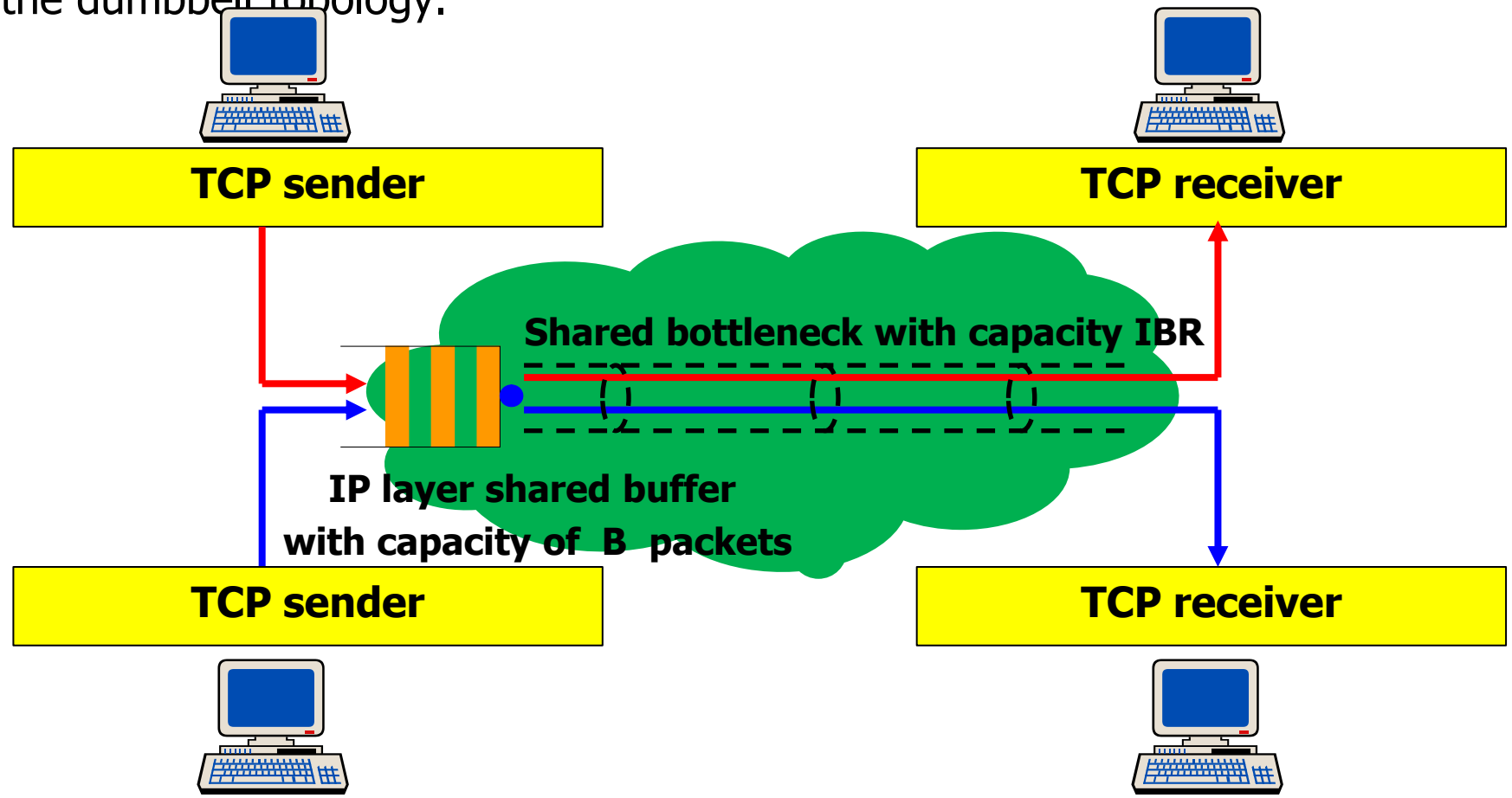


Note that with packet losses on the link, cwnd will typically be unable to reach the maximum of BDP + B. Packet losses cause sudden cwnd reductions or RTO events. Packet losses significantly reduce goodput and efficiency.

Fairness for TCP Traffic Flows

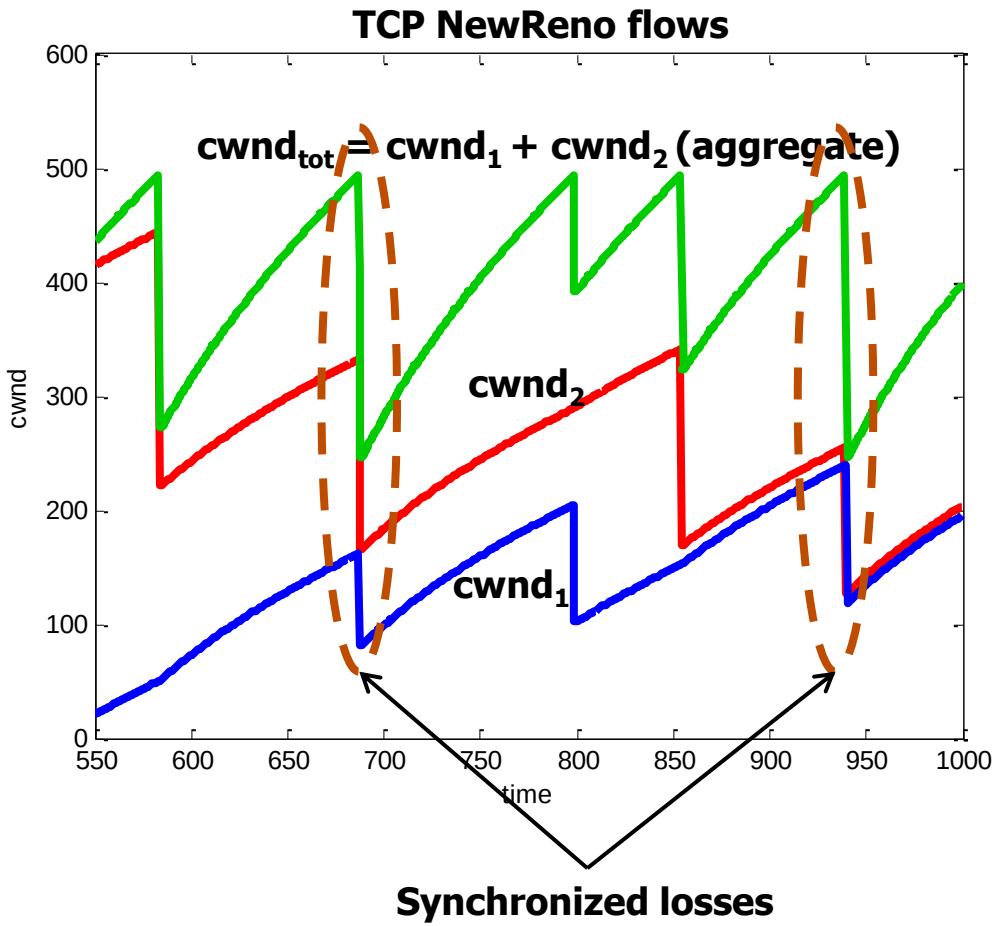
TCP Flows Sharing a Bottleneck

- We consider **multiple (two) TCP flows** sharing a bottleneck link according to the dumbbell topology.



Synchronized Losses for TCP Flows Sharing a Bottleneck

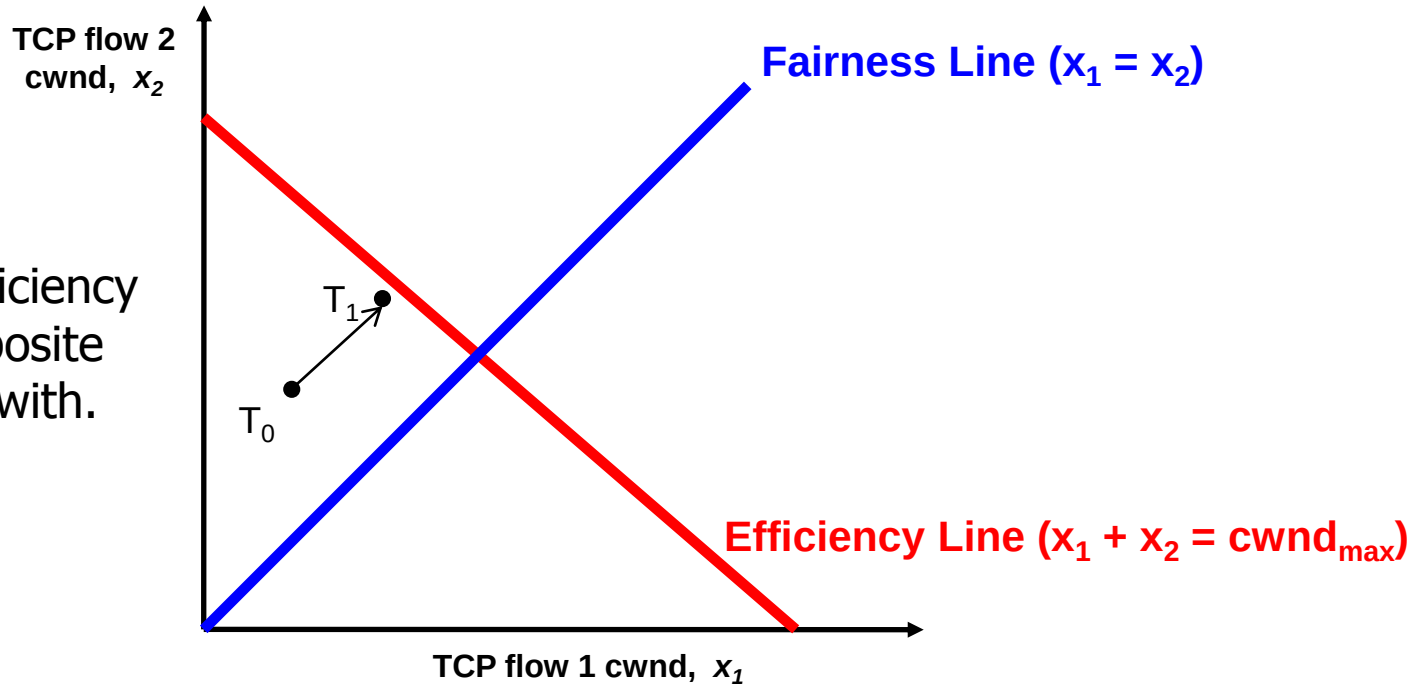
- If the **drop tail FIFO policy** is adopted for the buffer of the bottleneck link, all TCP flows experience buffer congestion at the same time, thus having **synchronized packet losses**.
  - All these TCP flows reduce their traffic injection at the same time due to synchronized losses.
  - There are intervals of time where the **bottleneck link is significantly underutilized**.



TCP Fairness

- Let us consider two TCP flows sharing a bottleneck link.
- The relative phases of the two cwnds have an impact on their behaviors.
- Let $x_1(x_2)$ denote the cwnd of flow #1 (#2). The fairness of **two TCP flows sharing a bottleneck link** can be studied by means of the graph of x_2 versus x_1 under the constraint $x_1 + x_2 \leq \text{cwnd}_{max} = B + BDP$:

Fairness and efficiency may be two opposite aspects to deal with.



TCP Fairness Measure

- **Jain fairness index Φ:**

$$\Phi = \frac{\left(\sum_{i=1}^{n} \Gamma_i\right)^2}{n\sum_{i=1}^{n} \Gamma_i^2}$$

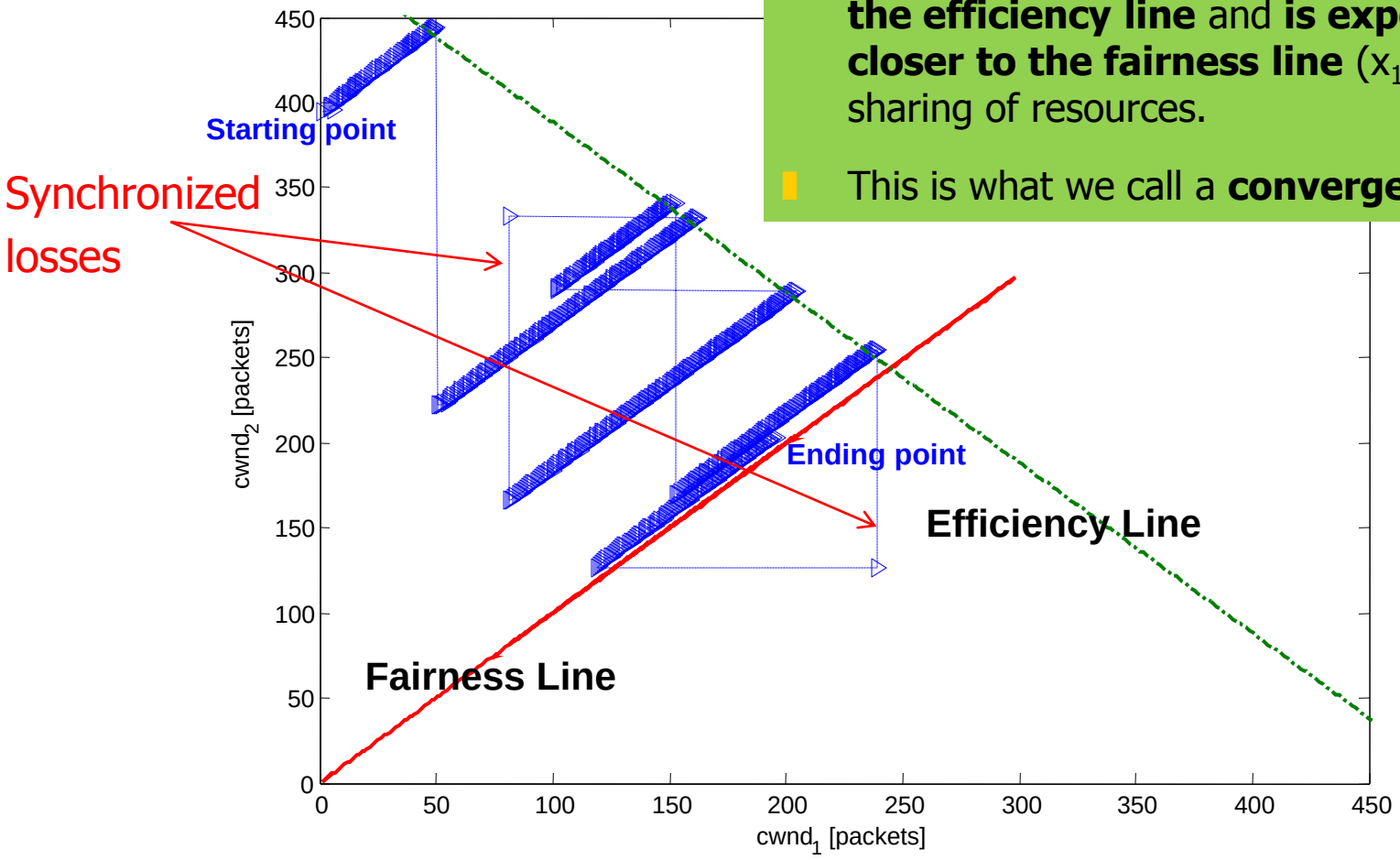
n: Number of flows

Γ_i : Average throughput of the i-th flow

- If all the n TCP flows sharing a bottleneck link (with IBR) achieve the same throughput (Γ_i = IBR/n), **the fairness index is maximum and equal to 1**.
- The **minimum fairness value is 1/n**, obtained when all TCP flows have Γ_i = 0, except one with Γ_i = IBR.

TCP NewReno Convergence

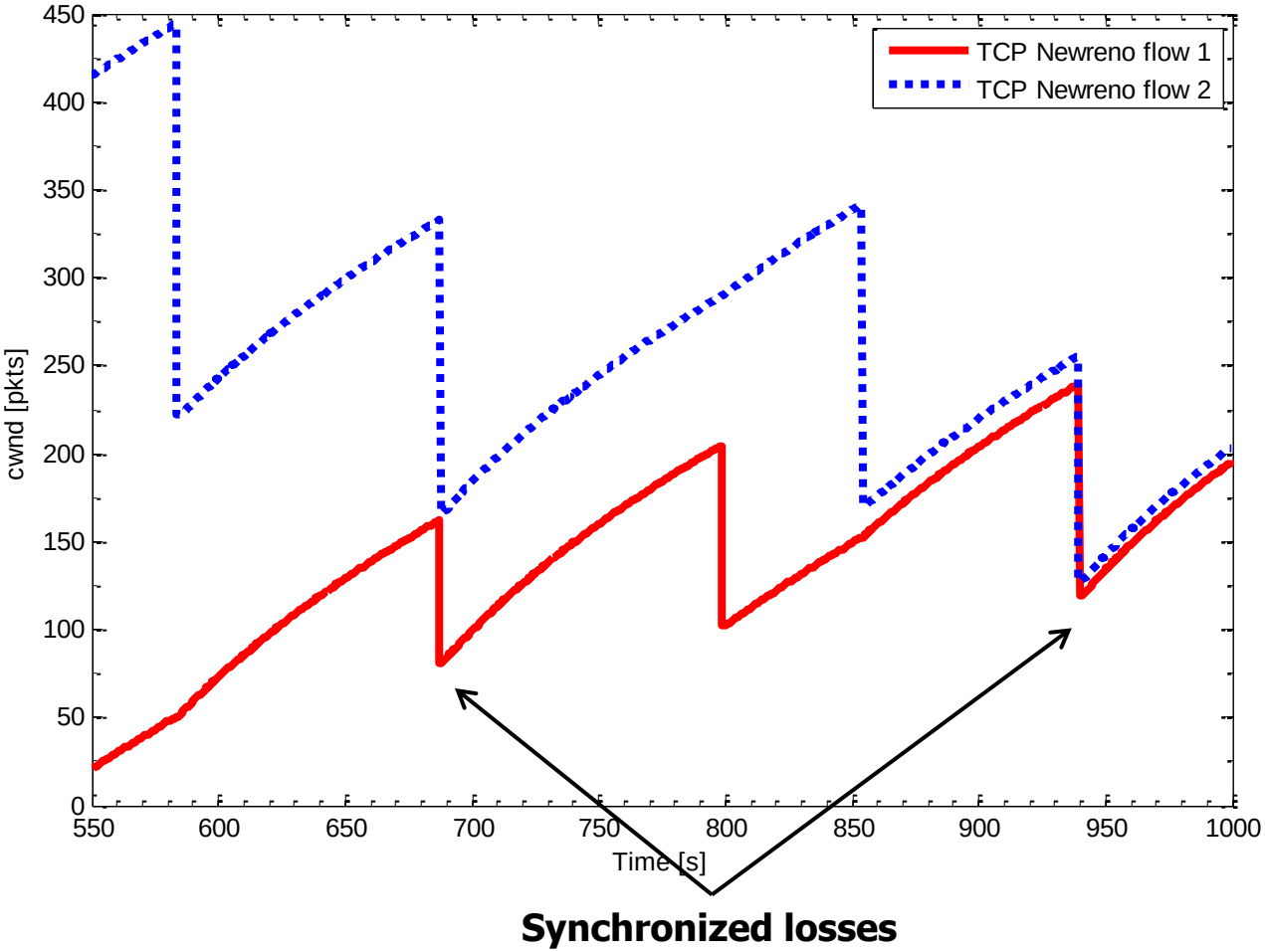
- The **behavior of the point (x_1, x_2) for two TCP flows of the same type** (i.e., both Reno or both NewReno) sharing the same bottleneck is depicted below. **This point oscillates below the efficiency line** and **is expected to move closer to the fairness line** ($x_1 = x_2$) for a fair sharing of resources.
- This is what we call a **convergent behavior**.



TCP Convergence Time

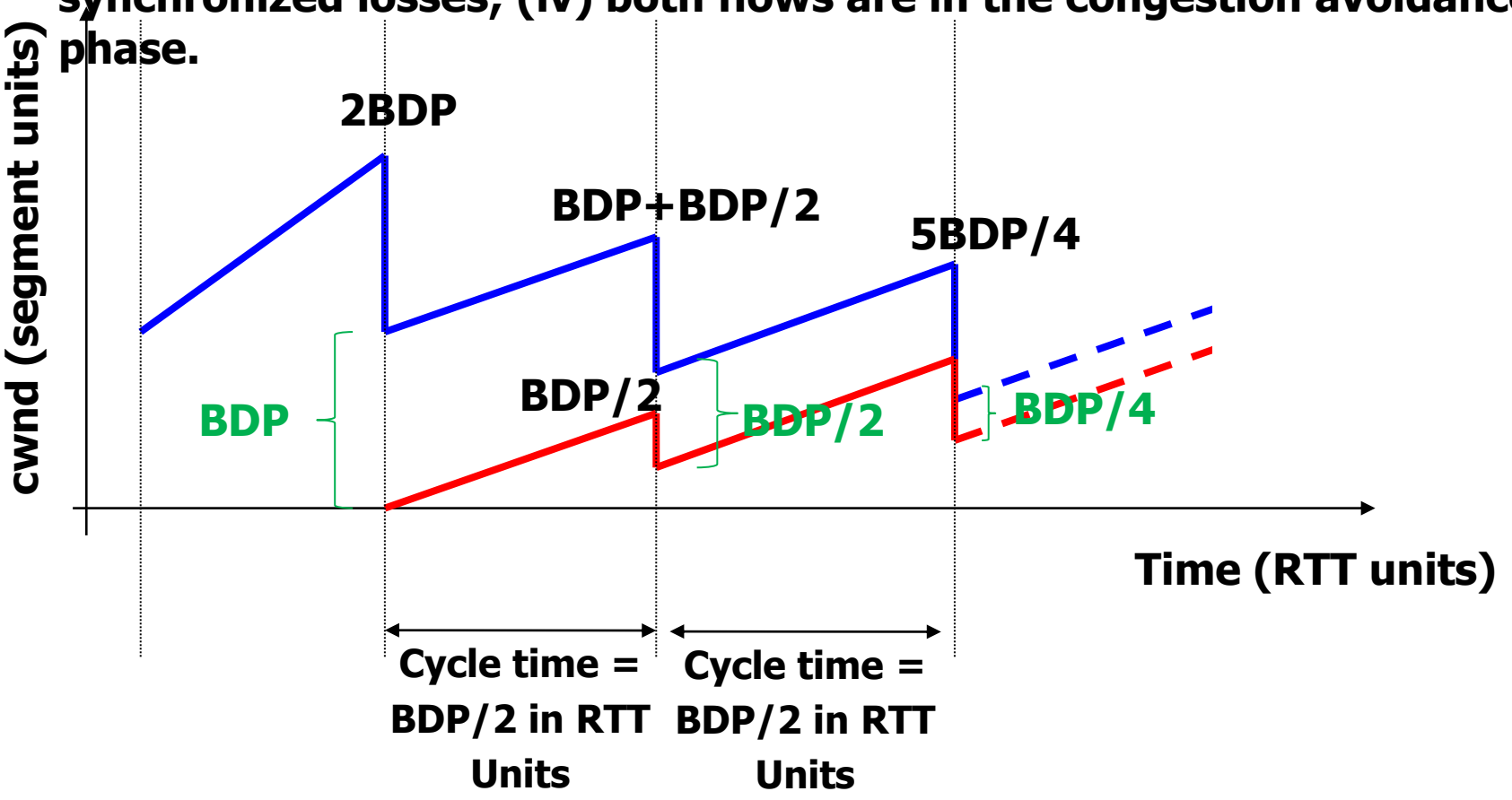
The same graph as before, but now the cwnd behaviors are shown as a function of time.

- The **Convergence time** is the time needed from a single (elephant) TCP flow saturating the bottleneck link, to the instant when a new started TCP flow reaches a fair sharing of the bottleneck link capacity ($x_1 \approx x_2$).
- **Convergence is not assured in general and depends on the TCP version.**



TCP NewReno Convergence Time Analysis

- Hypotheses: (i) B = BDP; (ii) the second flow starts when the first one has the maximum cwnd = 2BDP (worst-case); (iii) perfectly synchronized losses; (iv) both flows are in the congestion avoidance phase.



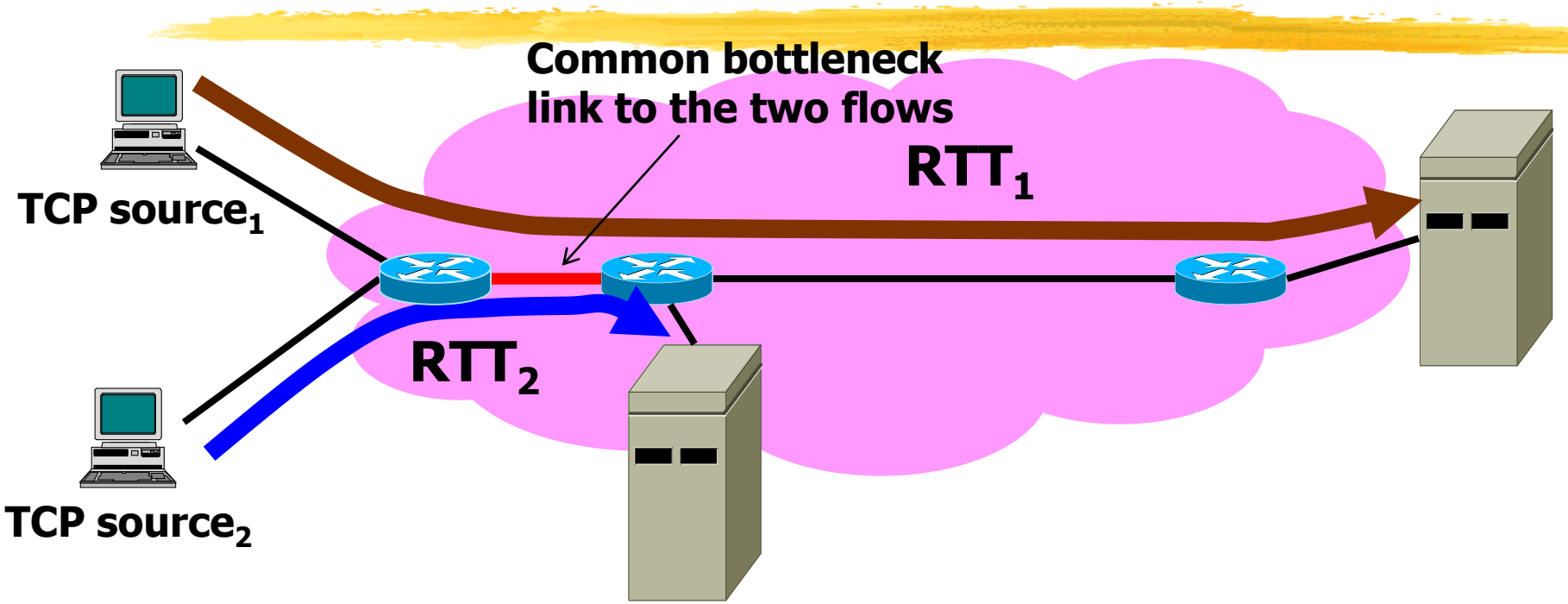
TCP NewReno Convergence Time Analysis (cont'd)

- **The duration of each cycle is BDP/2 in RTT units.**
- **At each cycle, the cwnd difference between the two flows halves.** Hence, $\log_2(\text{BDP})$ cycles are needed to achieve convergence.
- The product of the number of cycles and the cycle duration yields the **TCP NewReno convergence time** T_{NewReno} under our assumptions:

$$T_{\text{NewReno}} = \frac{BDP}{2}\log_2(BDP) \quad [\text{RTT units}]$$

If BDP = 100 pkts (LFN), $T_{\text{NewReno}} \approx 332$ [RTTs]

RTT Fairness



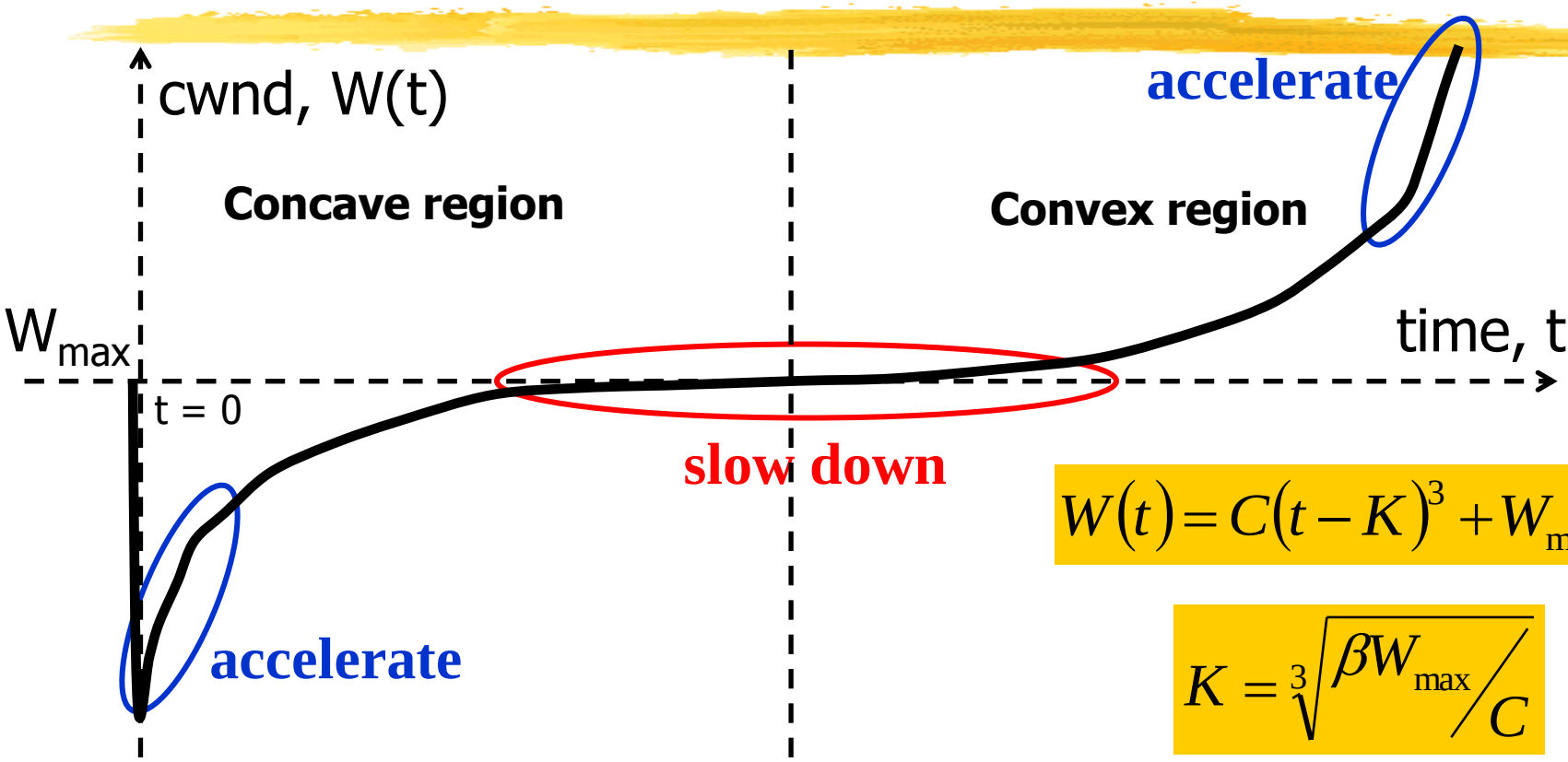
- Different (say 2) TCP connections may experience quite different RTT values, and a good TCP protocol should allow the different TCP flows to fairly share the bottleneck link bandwidth, regardless of their RTT values.
- **RTT fairness index = ratio of the average throughputs of the two flows Γ_1/Γ_2 with different RTTs, typically proportional to RTT_2/RTT_1.**

TCP Versions for LFN Networks (e.g., High-Speed Networks or Satellite Networks)

New TCP Versions for LFN and Simulation Tools

- In the last few years, many TCP variants have been proposed to address the **under-utilization of LFN networks due to the slow growth of cwnd**. Some examples of these versions are: HS-TCP, S-TCP, BIC, CUBIC, etc.. The cwnd behaviors of many of these variants and more can be found at the following URL:
  - http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html
- Even if the cwnd growths of these new protocols are scalable and good for LFNs, **fairness and convergence are major issues**.
  - **The main problem is to find a "suitable" growth function for cwnd.**
- Very important free **simulators for the networks** (suitable for simulating many TCP versions, routing, etc.) are ns-2 and the new ns-3. More details can be found at the following links:
  - http://nsnam.isi.edu/nsnam/index.php/User_Information
  - http://www.nsnam.org/

CUBIC TCP: cwnd Behavior



where **C** (= 0.4) is a scaling factor, **t** is the elapsed time from the last cwnd (W) reduction due to a packet loss at time t = 0, W_{max} is the maximum cwnd (W) value before the last reduction, and **β** is a constant used in a multiplicative decrease of cwnd after a packet loss operated as follows: $W(0) \leftarrow W_{max} - \beta W_{max} = (1-\beta) W_{max}$. where $\beta = 0.2$ so that $1 - \beta = 0.8$.

CUBIC TCP: cwnd Behavior (cont'd)

- **The cwnd growth function of CUBIC TCP depends on a cubic law of the time elapsed since the last packet loss; the cwnd grow time is independent of ACKs (and then on RTT).**
  - ACKs are still needed to understand the segments that have been correctly received.
- Cwnd growth slows down as it gets closer to **the value before last reduction (= W_{max}).**
- **K is the time needed to recover after a packet loss the same W_{max} value before the loss. The value of K has been determined by imposing $W(0) = C(-K)^3 + W_{max} = W_{max} - \beta W_{max}$.**
- CUBIC TCP is the default TCP version in Linux kernels (2.6.19 or above).

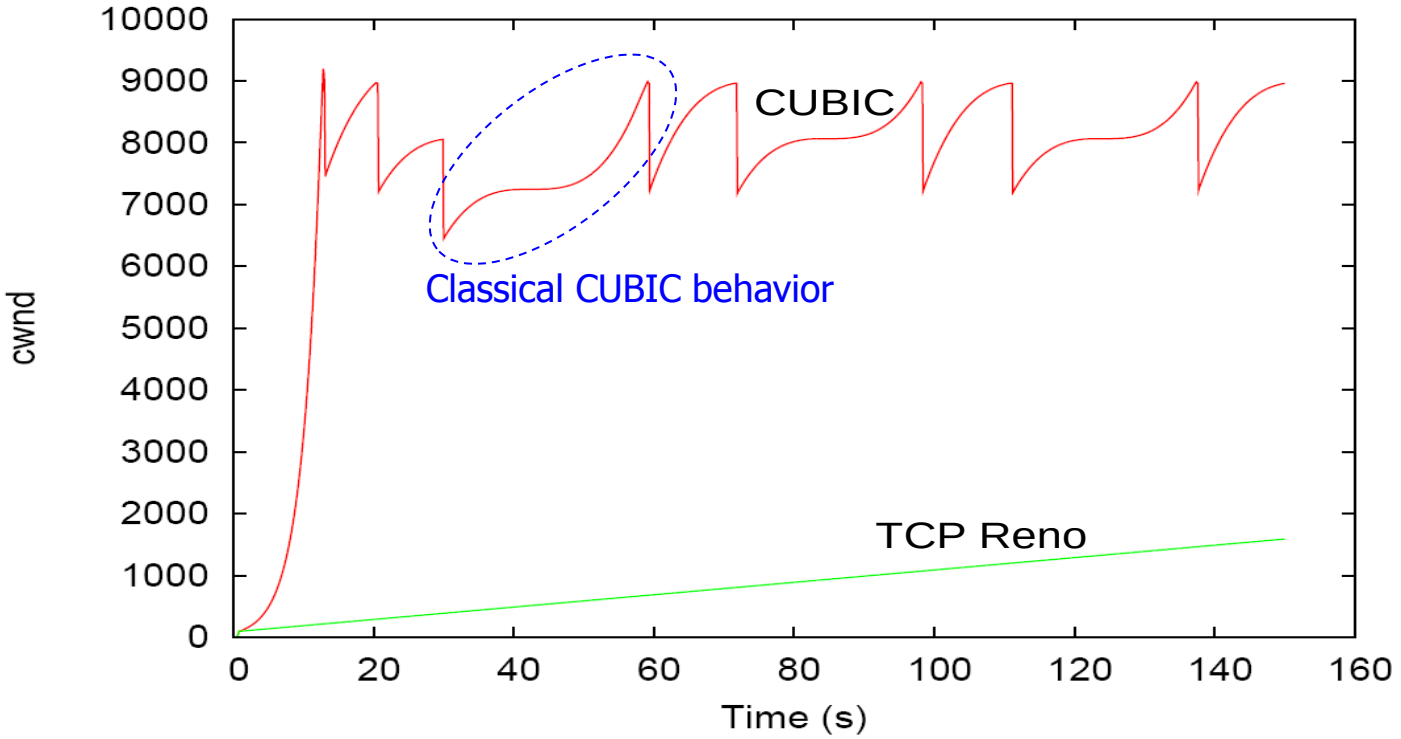
I. Rhee, L. Xu, S. Ha, "CUBIC for Fast Long-Distance Networks", IETF Internet-Draft, February 2007.

CUBIC TCP: Design Issues

- CUBIC exhibits the following properties:
  - **Stability**: CUBIC TCP has a very slow cwnd increase in the transition between the concave and convex growth regions, which allows the network to stabilize before CUBIC starts looking for more bandwidth.
  - **RTT fairness**: CUBIC TCP achieves RTT fairness among flows since the window growth is independent of RTT.
  - **Intra-protocol fairness**: there is the convergence for the cwnds of two competing CUBIC flows.
  - CUBIC TCP exhibits however **inter-protocol fairness issues** with other TCP versions, as shown in the following slide.

CUBIC versus Other TCP Versions

CUBIC TCP is sharing the bottleneck link with TCP NewReno.



- There is **no convergence** to a fair sharing of capacity: serious inter-protocol fairness problems.

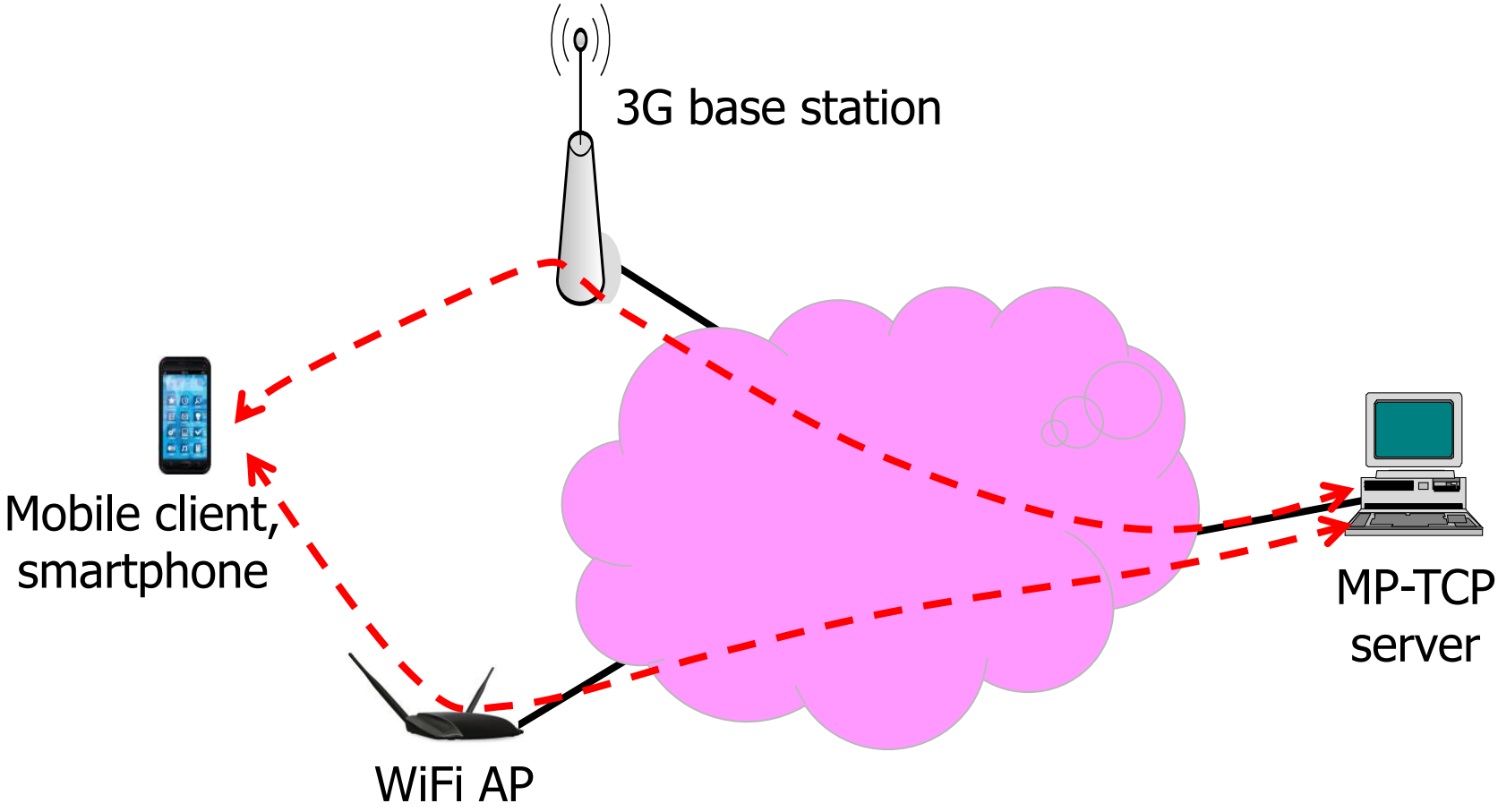
Compound TCP (CTCP)

- Compound TCP (CTCP) aggressively adjusts the congestion window (cwnd) to optimize TCP traffic injection in LFN networks.
- Compound TCP maintains two cwnd values: a **TCP NewReno-like (loss-based) window** and a **delay-based window**.
- **The size of the actual sliding window used is the sum of these two windows.**
- If the delay is low, the delay-based window rapidly increases to improve the utilization of the network. Once queuing is experienced, the delay window gradually decreases.

Multi-Homing and Multi-Path TCP

- Multi-Path TCP (MP-TCP) is a new approach to improve TCP performance exploiting **multiple source-destination paths** (e.g., **RFCs 6182 and 6824**, respectively of 2011 and 2013).
  - Requirement: we need a multipath transport layer solution that is transparent to middleboxes, that are network nodes with protocols up to transport (NATs, Firewalls, Gateways, PEPs, etc.)
- MP-TCP assumes that both sender and receiver are modified and that one or both of them can have multiple IP addresses (**multi-homing, multi-addressed, multi-network adapter**) to exploit different network paths.
  - According to RFC 6824, MP-TCP improves the throughput if **multiple paths can be used in parallel** for a destination or can make TCP robust in case of link disconnections (**additional path used as a backup**).
  - Each sub-flow is characterized by a suitable congestion control mechanism and a sub-flow sequence number. Sub-flows are bound together by means of a token identifier.
- MP-TCP signaling is based on **optional TCP header fields** (signaling: set up multiple sub-flows, reassembly of data, sub-flow termination, etc.).
- **LHCNet** (network for physics) is performing experiments with MP-TCP on end hosts of its multi-Gbit network for load balancing purposes.

MP-TCP and Mobility: HetNet Scenario WiFi/3G/4G



TCP Versions Implemented and Measurements

W. Richard Stevens, "TCP/IP Illustrated, Vol 1: The Protocols", Addison-Wesley Professional Computing Series, 2012.

TCP Versions and Operating Systems

- Many TCP algorithms are supported by the major operating systems:
  - TCP AIMD (*) and CTCP for the **Windows family** (e.g., Windows XP/Vista/7/Server/8).
  - TCP AIMD (*), BIC, CUBIC, HSTCP, Hybla, Illinois, STCP, Vegas, Veno, Westwood+, and YeAH for the **Linux family** (e.g., RedHat, Fedora, Debian, Ubuntu, SuSE).
  - TCP NewReno is a common TCP version for **UNIX** (Berkeley Software Distribution, BSD).
  - The TCP version used in **MAC OS X** operating system is based on the BSD version (FreeBSD 5 and therefore 4.4BSD) and is using TCP NewReno.

(*) AIMD can be considered quite close to NewReno.

TCP Versions and Operating Systems (cont'd)

- **Both Windows and Linux users can change their TCP algorithms and settings by means of a line of command.** Linux users can even design and then add their own TCP algorithms.

- Under Vista/Windows 7, the following prompt command is available to verify/to modify TCP settings:

netsh int tcp show global

- **CTCP is enabled by default in Server 2008** and disabled by default in computers running Windows Vista and Windows 7. CTCP can be enabled (disabled) by means of a suitable command (Vista/Windows 7):

netsh interface tcp set global congestionprovider=ctcp

(netsh interface tcp set global congestionprovider=default)

- The change of TCP version has impact only if this is done on the TCP sender.

TCP Versions and Operating Systems (cont'd)

- Example of use of the prompt command "netsh int tcp show global":

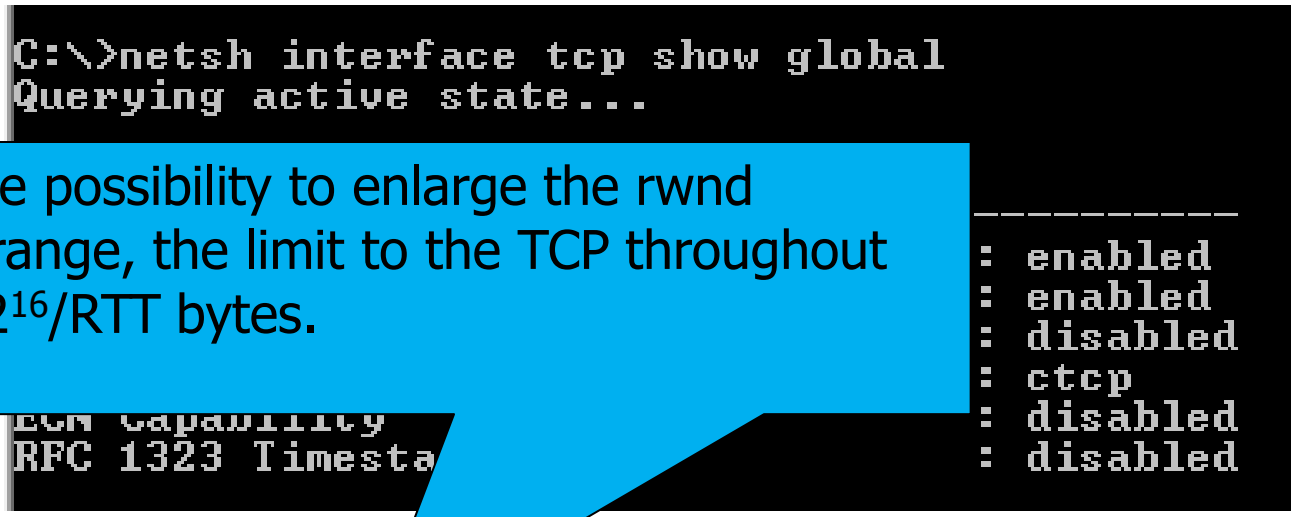
```
C:\>netsh interface tcp show global
Querying active state...

TCP Global Parameters
----------------------------------------------
Receive-Side Scaling State          : enabled
Chimney Offload State               : enabled
Receive Window Auto-Tuning Level    : disabled
Add-On Congestion Control Provider  : ctcp
ECN Capability                      : disabled
RFC 1323 Timestamps                 : disabled
```

- It is possible to set different options, such as window scaling to enlarge the rwnd range, timestamp options to improve the RTT estimate, ECN, etc.

TCP Versions and Operating Systems (cont'd)

- **Example of use of the prompt command "netsh int tcp show global":**



- **It is possible to set different options, such as window scaling to enlarge the rwnd range, timestamp options to improve the RTT estimate, ECN, etc.**

TCP Versions and Operating Systems (cont'd)

- **Example of use of the prompt command "netsh int tcp show glob**

A timestamp is an optional field in the TCP header that contains the current value of the clock of the sender. In particular, the sender places a timestamp value in each segment sent. The receiver reflects this value in the ACK, thus allowing an accurate RTT calculation at the sender for every ACK. This is useful because current implementations measure RTT only once per window of data and this could not be accurate for LFN networks.

- **It is possible to set different options, su as window scaling to enlarge the rwnd range, timestamp options to improve the RTT estimate, ECN, etc.**

TCP Versions and Operating Systems (cont'd)

- The different operating systems use distinct settings for some basic TCP parameters as follows:
  - Microsoft Windows XP: Initial cwnd of 1460 bytes and maximum possible (initial) rwnd of 65535 bytes.
  - Microsoft Windows 7: Initial cwnd of 2920 bytes (i.e., more than one segment) and maximum possible rwnd of 65535×2^2 bytes by means of the window scaling option according to RFC 1323.
  - Ubuntu 9.04: Initial cwnd of 1460 bytes and maximum possible rwnd of 65535×2^5 bytes.
  - MAC OS X Leopard 10.5.8: Initial cwnd of 1460 bytes and maximum possible rwnd of 65535×2^3 bytes.

R. Dunaytsev. *TCP Performance Evaluation over Wired and Wired-cum-Wireless Networks*. PhD thesis, TUT Tampere, 2010.

Testing TCP Performance: Iperf

- **Iperf is a free tool to measure TCP goodput (bandwidth),** allowing the tuning of various parameters. **Iperf reports bandwidth, delay variation, and datagram loss**.
- Developed by the National Laboratory for Applied Network Research (NLANR) project, iperf is now maintained and developed on Sourceforge at http://sourceforge.net/projects/iperf
- The –s option sets the server (**TCP receiver**)
- The –c option with the IP address of the server sets the client (**TCP sender**)
- The –w option can be used **to set a particular TCP window size (socket buffer size)**. This value should be 'aligned' with BDP for an optimal TCP goodput performance.
- The –h option is used for the help of the commands.

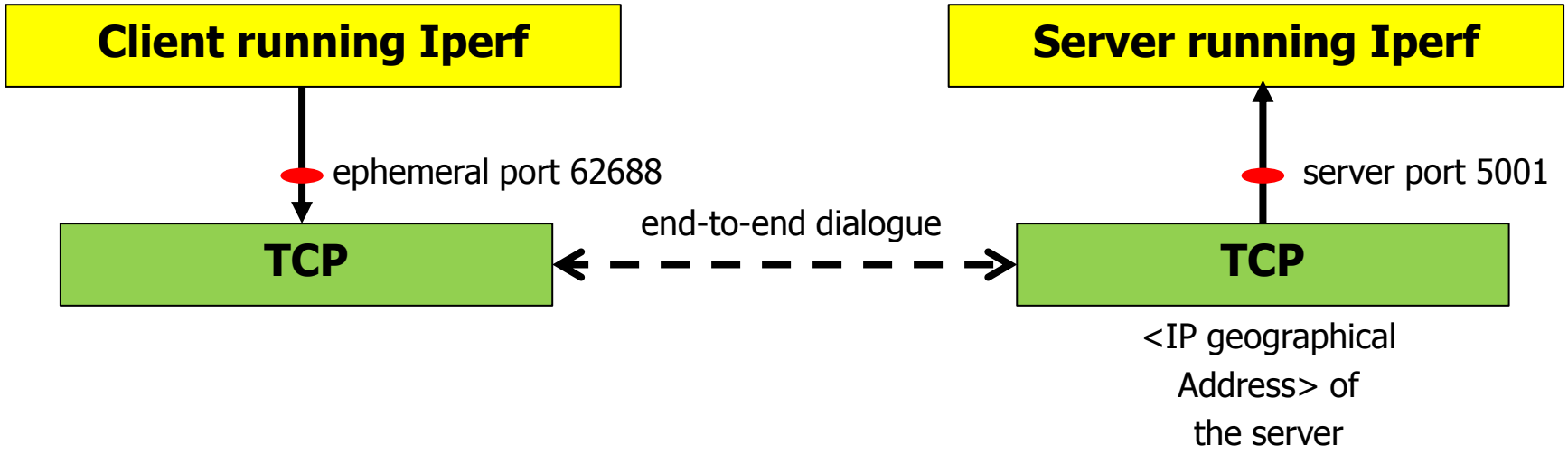
Testing TCP Performance: Iperf

- **Iperf is a free tool to measure TCP goodput (bandwidth),** allowing the tuning of various parameters. **Iperf reports bandwidth, delay variation, and datagram loss**.
- Devel
(NLAN
http:/
- The –
- The –
sende
- The –w option can be used **to set a particular TCP window size (socket buffer size)**. This value should be 'aligned' with BDP for an optimal TCP goodput performance.
- The –h option is used for the help of the commands.

For instance if one system is connected with Gigabit Ethernet (@ 1Gbit/s), but the other one with Fast Ethernet (@100Mbit/s) and the measured round trip time is 150 ms, then the window size (socket buffer size) should be set to 100 Mbit/s x 0.150 s / 8 = 1875000 bytes (≈ BDP), so setting the TCP window to a value of 2 MBytes would be a good choice.

Testing TCP Performance: Iperf (cont'd)

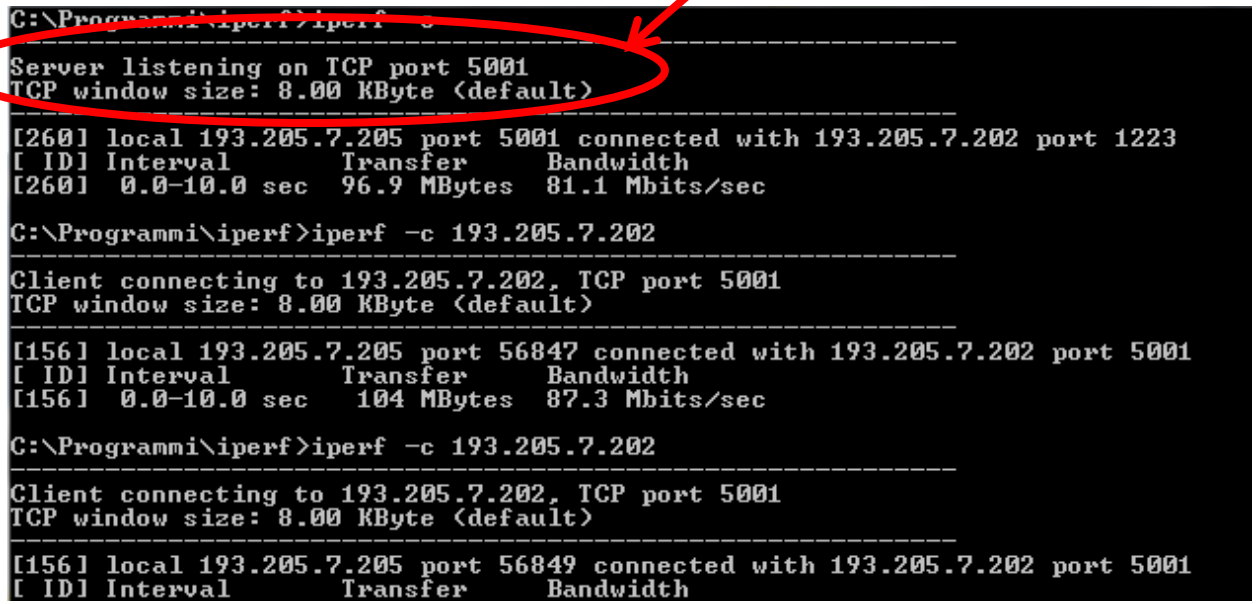
- **The configuration of this experiment is show below:**



- **We have to run Iperf on both server (TCP receiver) and client (TCP sender)** to exchange traffic and measure the TCP performance.
  - We run 'iperf -s' on the server to enable it to receive traffic sent from the client via TCP.
  - Then, we run 'iperf -c <IP address> on the client to send data to the server by means of TCP.

Testing TCP Performance: Iperf (cont'd)

- Iperf performs repeated file transfers for 10 s and measures the resulting average capacity (bandwidth).



rwnd = 8 kB for the operating system.

Thank you!

giovanni.giambene@gmail.com