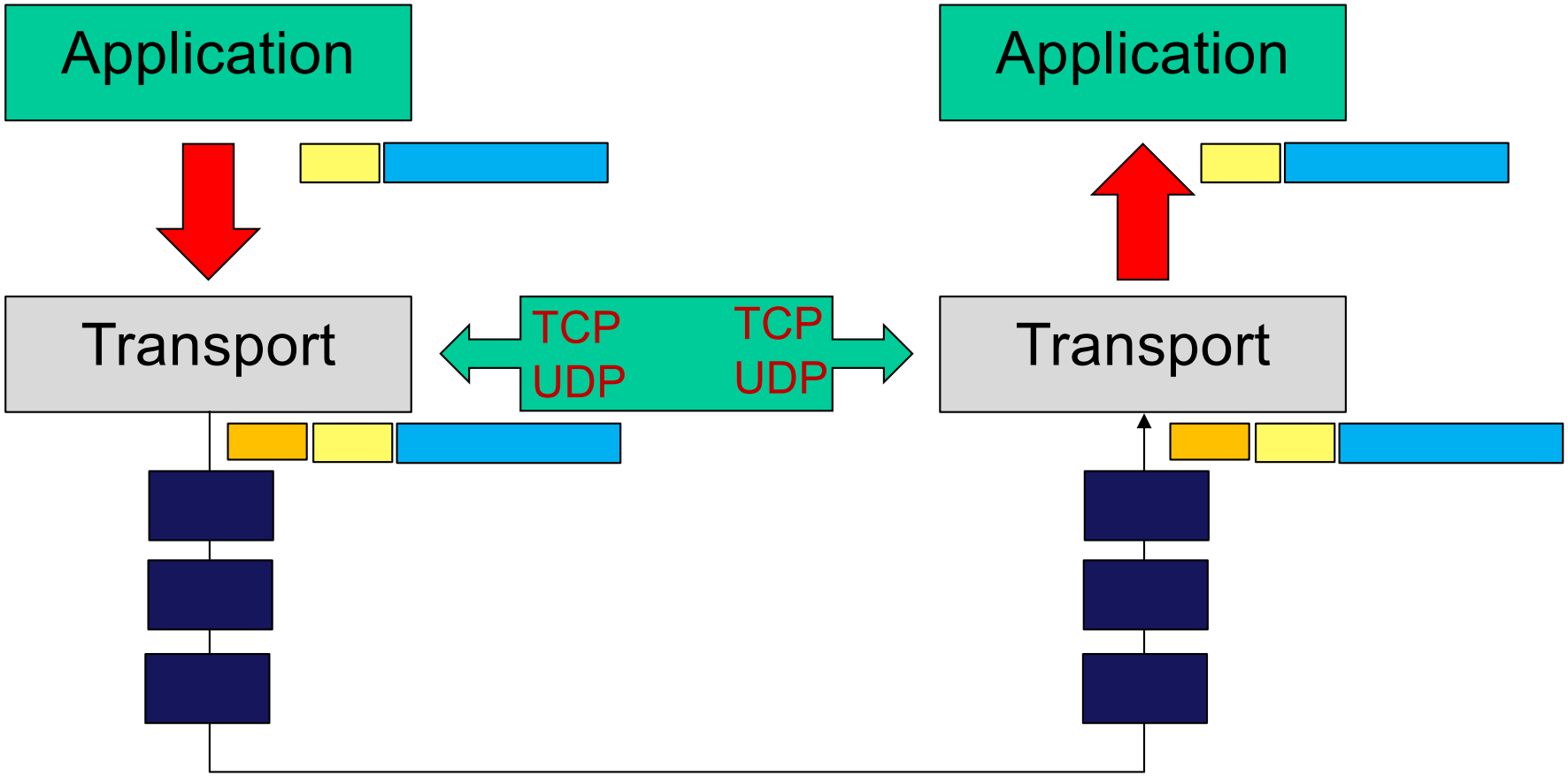
TRANSPORT LAYER

Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data Link	Network Interface
Physical	Physical

End to end application-transport exchange



TCP = Transmission Control Protocol
UDP = User Datagram Protocol

Transport layer protocols: TCP

- TCP (Transmission Control Protocol)
  - Internet applications most utilized protocol
  - Reliable packet delivery
  - **Specific satellite issues**

Worth to be deeply analyzed if used over satellite networks

TCP is responsible of the difference between the assigned and the experienced capacity

Rationale for reliable protocols

- Lower layers may be unreliable
  - Sometimes reliability provided through overload (repeated transmission) or FEC at different layers (decrease frame efficiency)
- Packets may be lost due to
  - Transmission Errors (channel impairments)
  - Networks overloading
  - Resource (network capacity, buffer) limitations
- Out of order packets delivery due to
  - Routing (especially with satellite constellations)
  - Different length path (for example in case of ISHO)

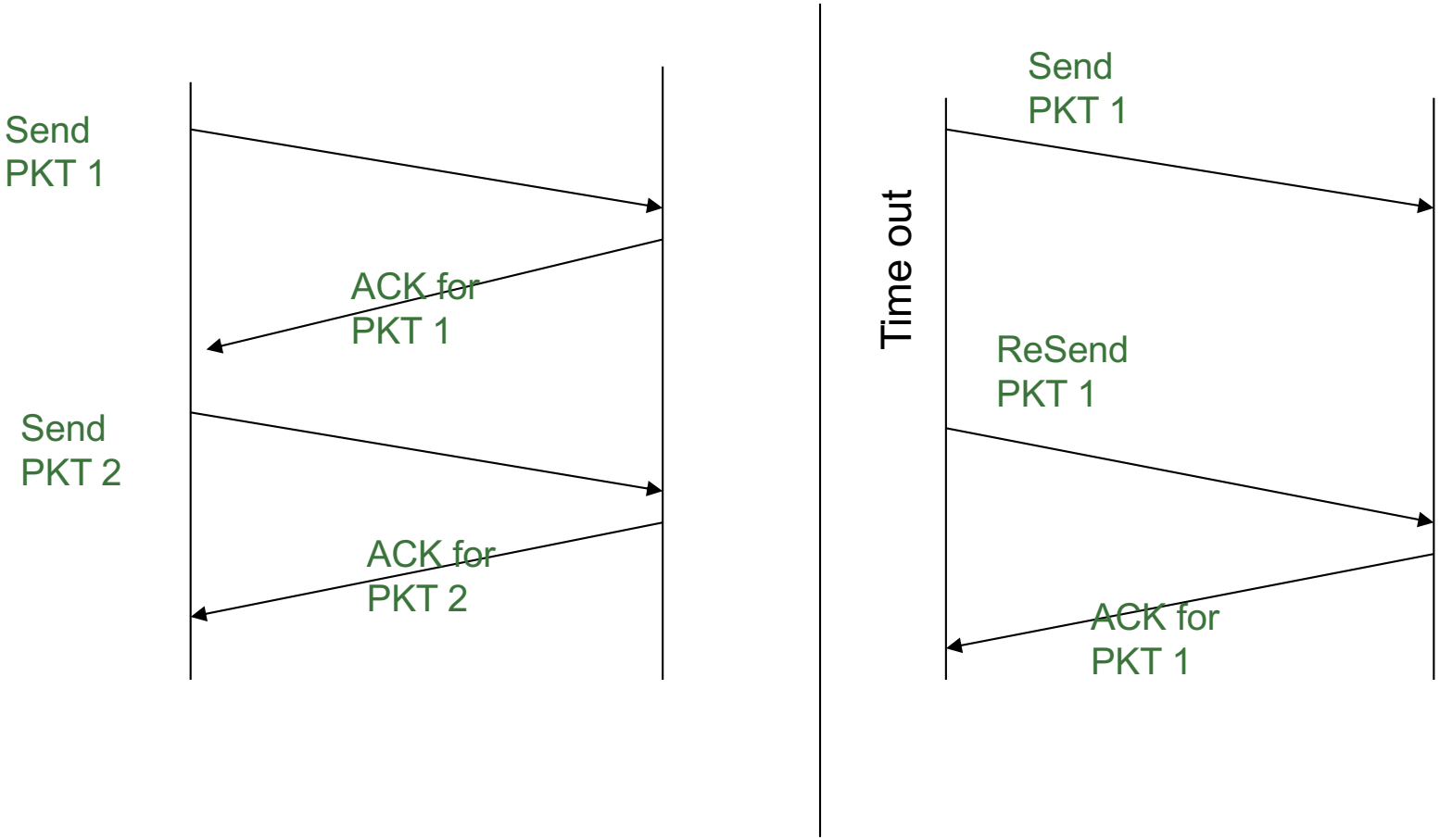
➔ Need for higher layer to provide guarantee in packets delivery.

TCP concept

- Connection oriented protocol that implies a connection between two hosts (specified by IP addresses) and one process on each host (specified by port numbers).
- It provides:
  - Reliable byte stream transport
  - In sequence delivery of data
    - Reorders out of order data
    - Discards duplicate data
  - Calculation and verification of a mandatory checksum
  - Interface to the applications (HTTP, FTP, Email, Telnet, …)

Guaranteed Delivery

- Positive Ack with Retransmission.

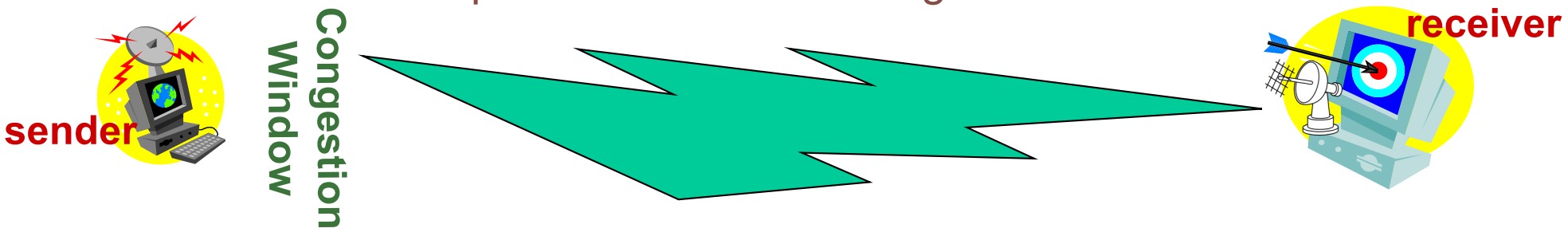


TCP Functions

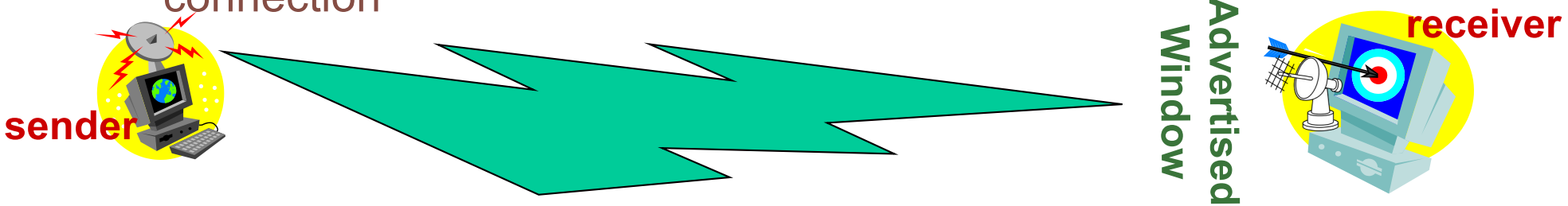
- Three main functions
  - Flow Control (Receiver may get swamped by Sender)
  - Congestion Control (Fill up Bandwidth)
  - Error recovery (Guaranteed Delivery)
- Two different ways to detect loss
  - Retransmission Time Out (RTO)
  - Duplicate ACK
- One more concept
  - CUMULATIVE Ack: ACK's report the maximum contiguous data received

TCP Windows

- The **Congestion Window** (cwnd) is flow control imposed by the sender
  - Assessment of perceived network congestion



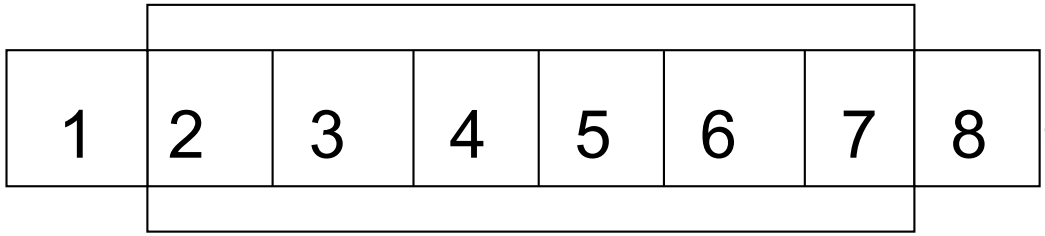
- The **Advertised Window** is flow control imposed by the receiver
  - Amount of available buffer space at the receiver for the current connection

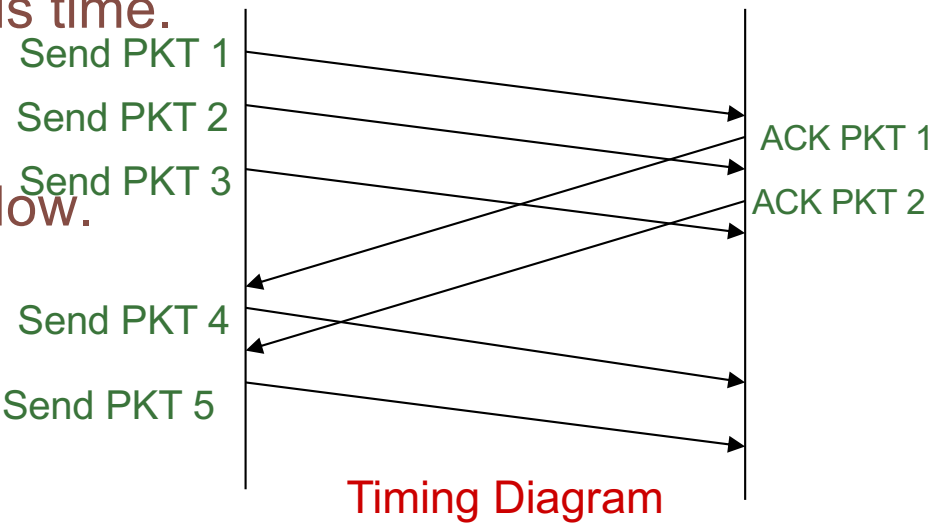


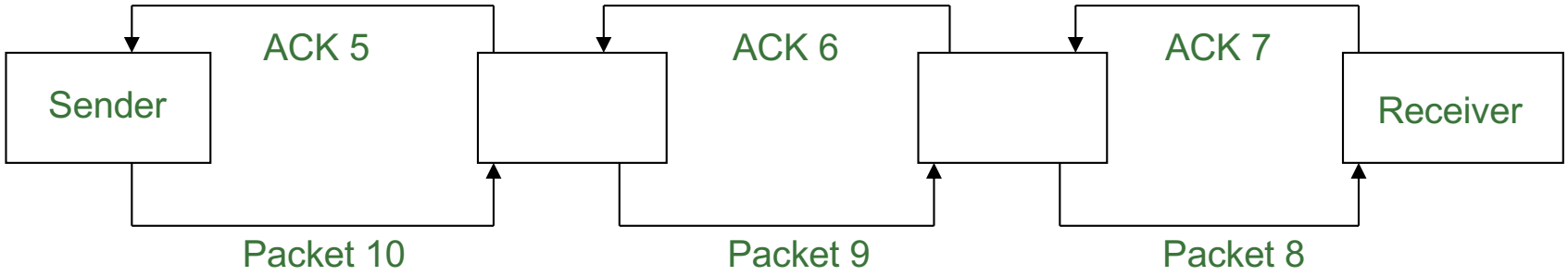
- **Current Window** = min (**cwnd**, **advertised window**)

TCP flow control: Sliding Windows

- In simple positive ack scheme
  - Sender cannot send next packet until it receives ack for earlier packet.
  - Network completely idle during this time.
- Sliding Window technique
  - Transmit all packets inside a window.

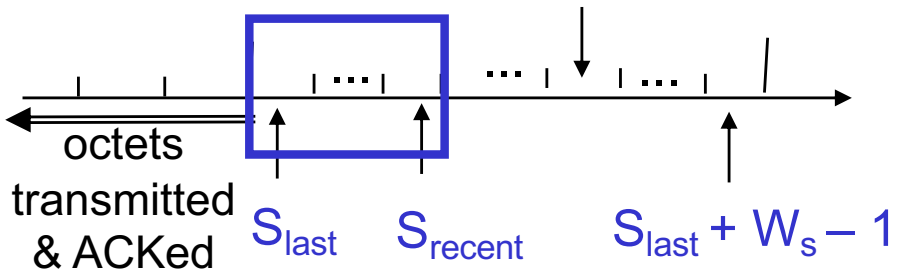






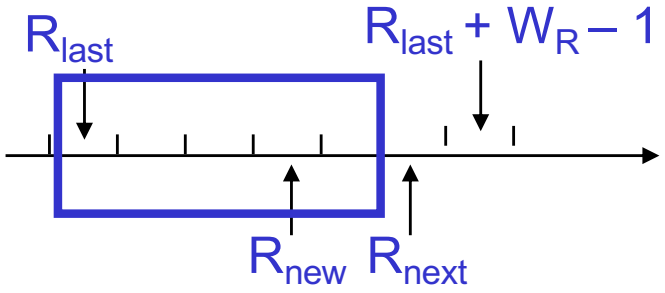
TCP flow control: Sliding Windows dynamics

Sender side Window $S_{last} + W_a - 1$



S_{last}	oldest unacknowledged byte
S_{recent}	highest-numbered transmitted byte
W_a	advertised window
$S_{last}+W_a-1$	highest-numbered byte that can be transmitted
W_s	sending window
$S_{last}+W_s-1$	highest-numbered byte that can be accepted from the application

Receive side Window



R_{last}	lowest-numbered byte not yet read by the application
R_{new}	highest numbered of correctly received byte
R_{next}	next expected byte
W_R	receive window (receiver buffer size)
$R_{last}+W_R-1$	highest-numbered byte that can be accommodated in receive buffer

The current window may not correspond to any of the two windows because it increases and decreases depending on Congestion Control

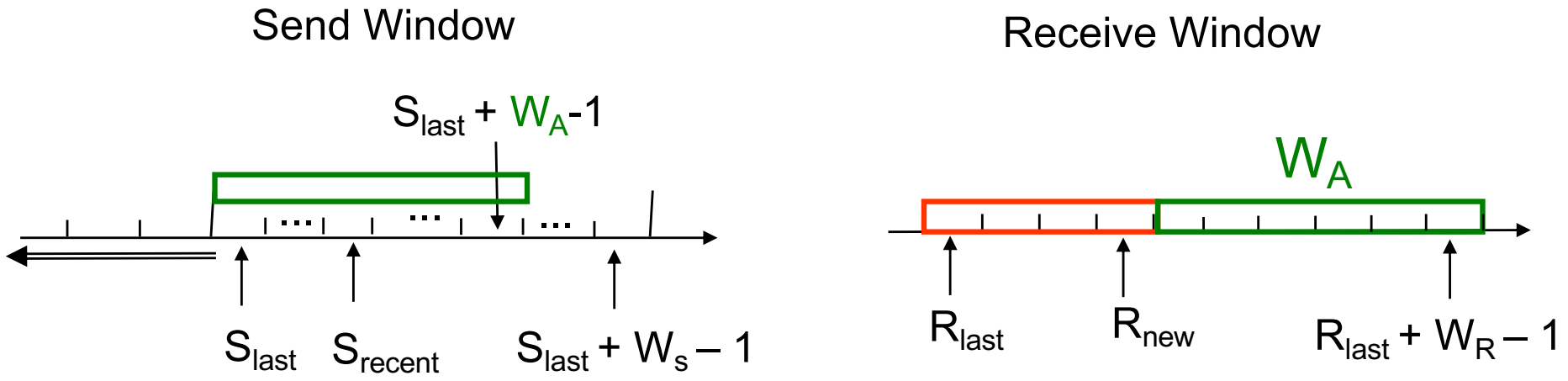
TCP flow control: Advertised window

TCP receiver controls the rate at which the sender transmits to prevent buffer overflow; advertises a window size specifying number of bytes that can be accommodated by the receiver.

$$W_A = W_R - (R_{new} - R_{last})$$

TCP sender obliged to keep number of outstanding bytes below W_A.

$$(S_{recent} - S_{last}) \leq W_A$$



Baseline Congestion Control

- Three algorithms implemented to avoid network overload
  - Slow Start
  - Congestion Avoidance
  - RTO (Retransmission Timeout)

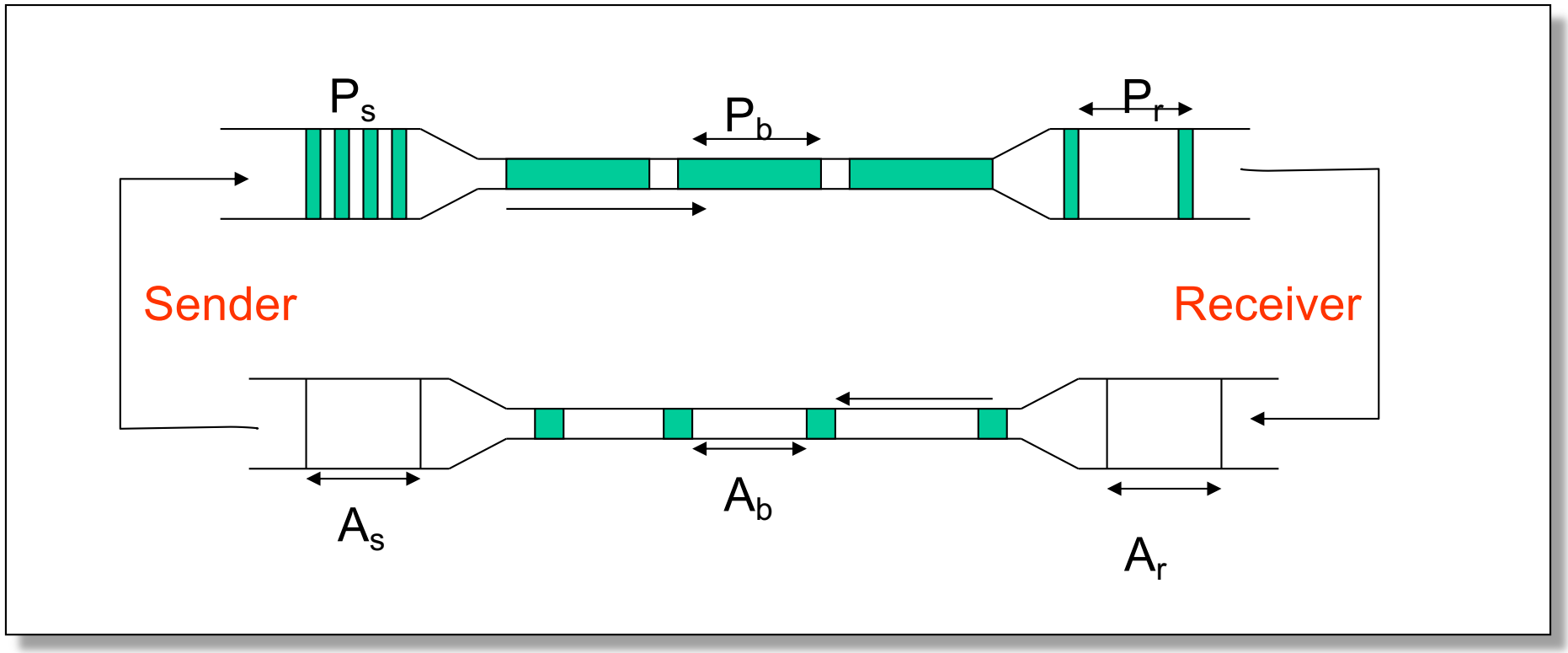
Growing mechanisms

Reaction to packet loss implying window reduction

Optimum Value for Window

- Value of window $\propto$
  - Bandwidth of bottleneck
  - Round Trip Time
- To maximize transmission volume packets are sent till ACK are received
  - **Opt_Window** * PKT Size = Bandwidth * RTT
- TCP tries to reach the optimum value of Window size through Slow Start or Congestion Avoidance

TCP self-clocking or ACK-clocking



- **Self-clocking** systems tend to be very stable under a wide range of bandwidths and delays.
- The main issue with self-clocking systems is getting them started.

Congestion Control: Main Parameters

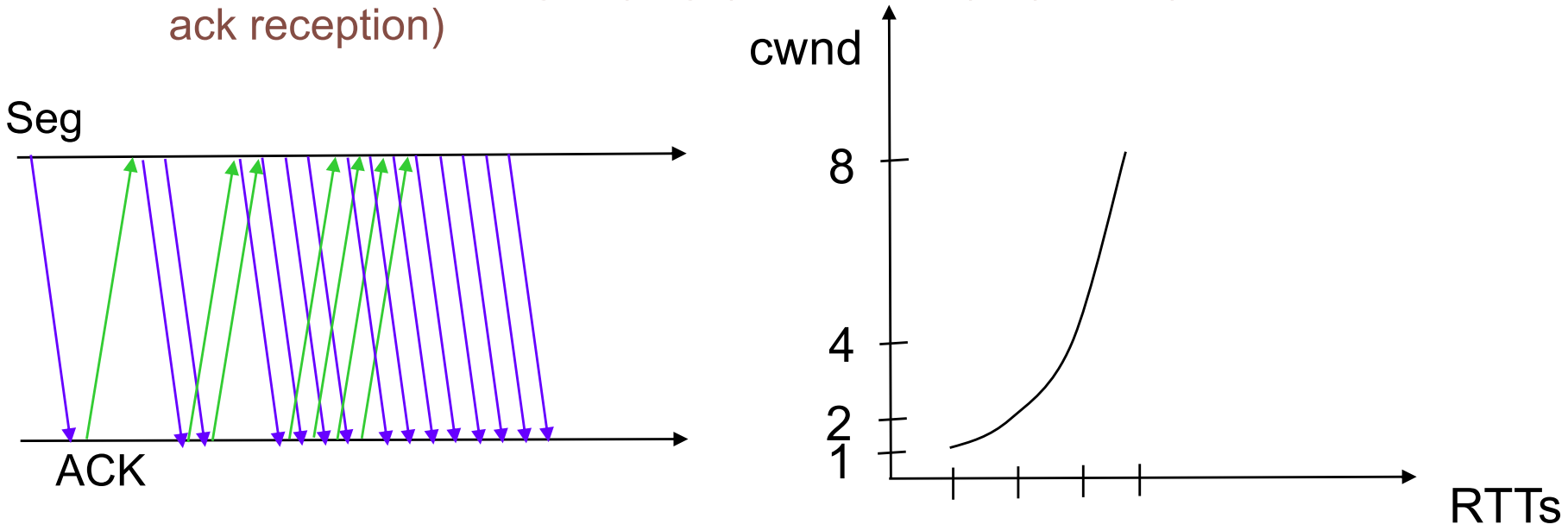
- *Congestion Window* ***cwnd***
  - Upper bound for non acknowledged transmitted data
  - It varies as a function of the estimated traffic in the network
- *Slow Start Threshold* ***ssthresh***
  - Its value determines the congestion control algorithm

  If ***ssthresh > cwnd*** ***Slow Start***

  If ***ssthresh < cwnd*** ***Congestion Avoidance***
  - The initial value determined by the receiving buffer size
- *Maximum Segment Size* ***MSS***
  - The largest chunk of data that TCP will send to the other end
  - At connection set up each end can announce its MSS

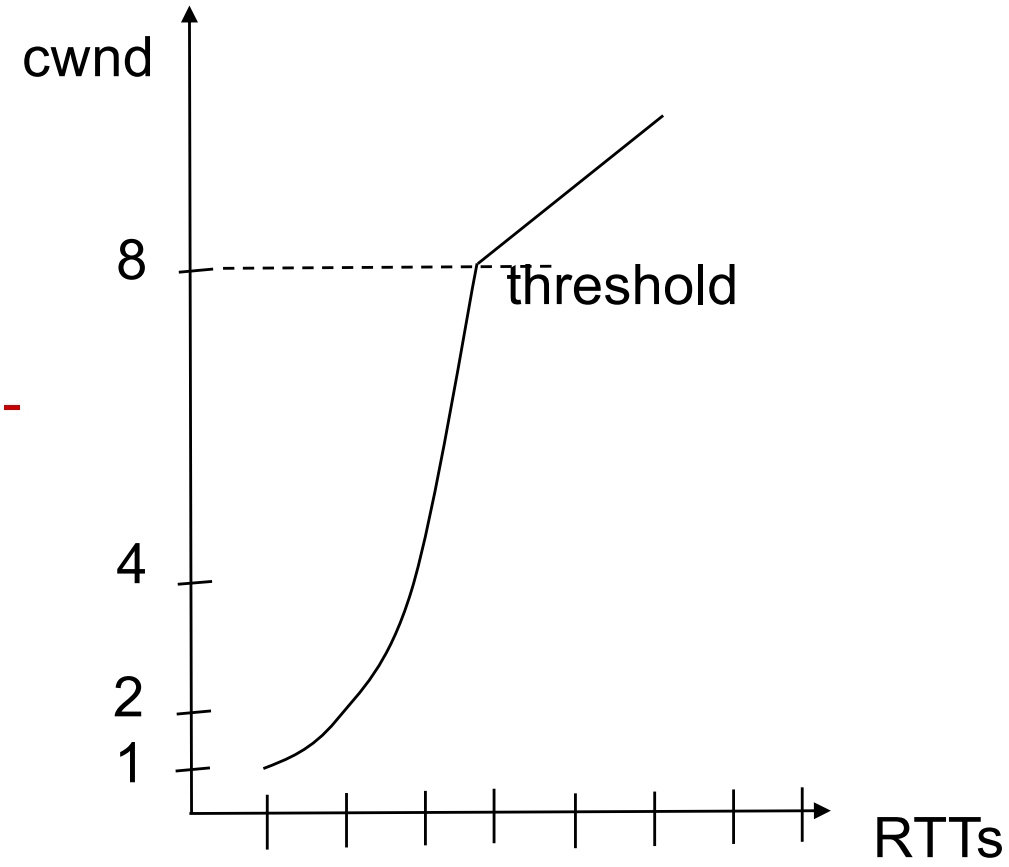
Slow Start mechanism

- **Slow start**: congestion window size increases by one segment upon receiving an ACK from receiver
  - initialized at ≤ 2 segments
  - used at (re)start of data transfer
  - congestion window increases exponentially
- Ends when SSThresh is reached
  - Slow Start Ramping in (Log_2(OptWinSize)+1)*RTT (until the last ack reception)



Congestion Avoidance mechanism

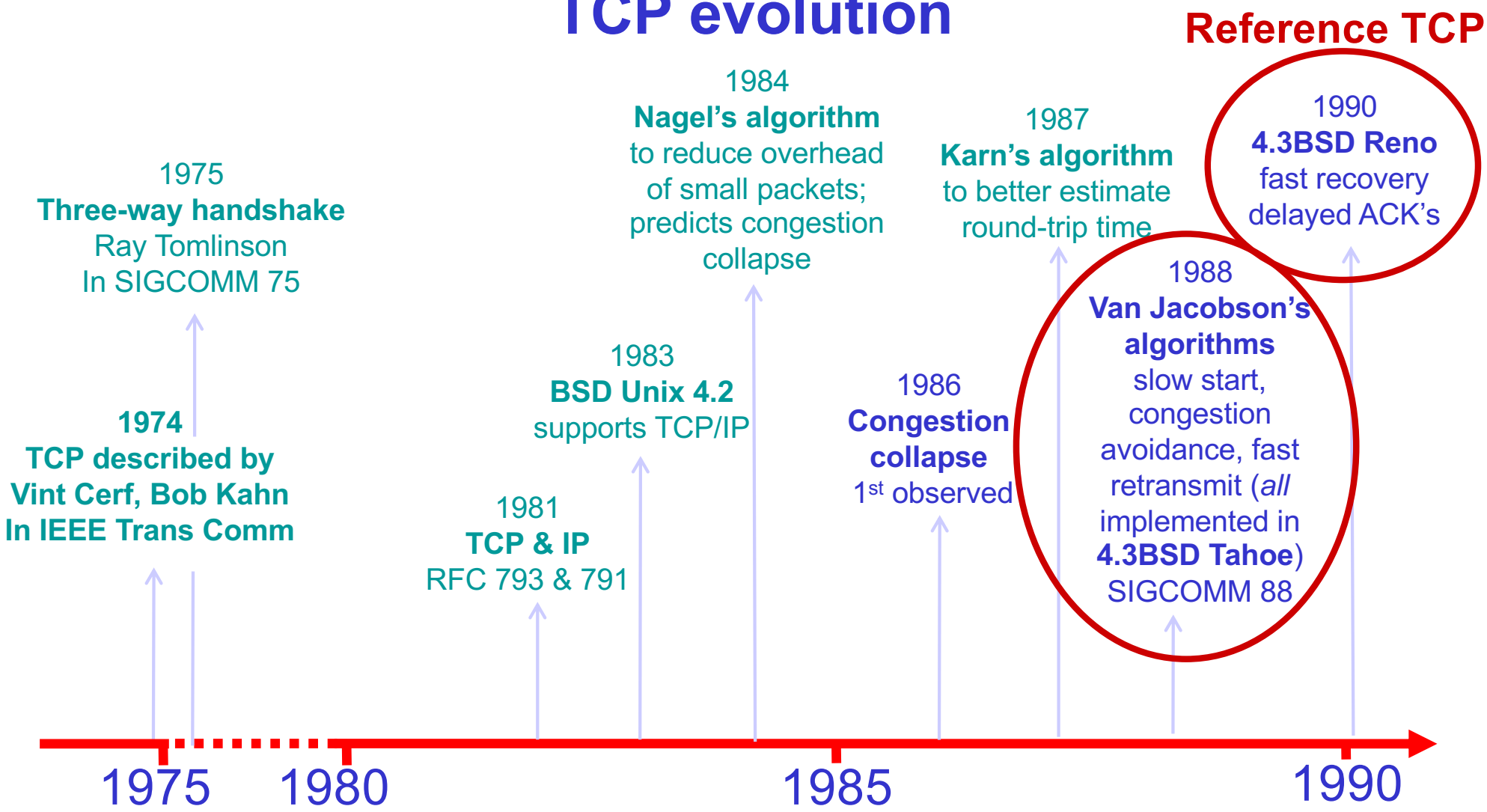
- Algorithm progressively sets a *congestion threshold*
  - When cwnd > threshold, slow down rate at which cwnd is increased
- Increase congestion window size by one segment per round-trip-time (RTT)
  - Each time an ACK arrives, cwnd is increased by 1/cwnd
  - In one RTT, cwnd segments are sent, so that total increase in cwnd is cwnd x 1/cwnd = 1
  - cwnd grows linearly with time



Loss detection: RTT and Timeout

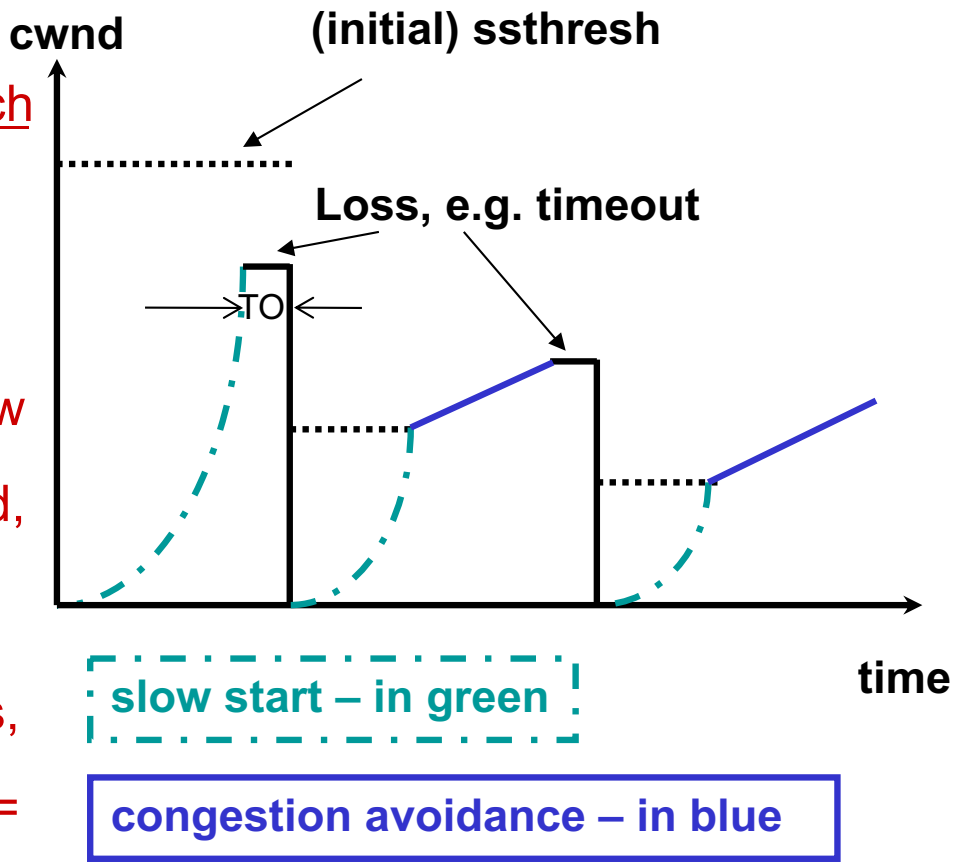
- Packets may experience different delays
- RTT (Round Trip Time) estimate
  - RTT = (a * oldRTT) + (1 – a) *newRTTsample
- Retransmission Timeout (RTO)
  - TCP keeps a timer for each packet it sends.
  - RTO =
    - b * RTT (b normally equal to 2) or
    - meanRTT + 4·σ (mean deviation of RTT)
  - If ACK not received within this time, TCP resends packet.
- Karn's Algorithm
  - Upon timeout, NewRTO = c * oldRTO for the retransmitted packet
  - Once received the ack for the retransmitted packet RTO resets oldRTO

TCP evolution



TCP Tahoe

- **Slow Start**: Starts with window size of one segment. Increases window by one for each packet acknowledged till "threshold" is reached.
- **Congestion Avoidance**: Increases window by one for every window of packets ACKed, which is equal to the RTT.
- **Multiplicative Decrease**: If timeout occurs, threshold = cwnd/2 (min value 2), window = 1, RTO = RTO * 2.



DUP ACKs in Tahoe (Fast Retransmit algorithm*)

- DUPACK generated when out of sequence packet are received [n^{th} lost but $(n+1, n+2, \ldots)^{th}$ received]
- If DUP ACKs are received
  - Indication that network is working and only some packet has been lost.
  - TCP Tahoe retransmits what appears to be the missing segment on receiving 3 DUP ACKs (Fast Retransmit).
  - Window is reduced to 1 and threshold set to (current) cwnd/2
  - This only takes into account that some packet has been lost
  - This mechanism introduces Reno.

* Fast with respect to wait for RTO expiration (if RTO>3DUPACK reception)

TCP Reno – Fast Recovery

- TCP Reno improves upon this behavior
  - **Introduces Fast Recovery**
  - Sets Threshold = Window/2 and Window = Threshold + 3 (1 for each DPACK supposing there is room in the pipe for 3 packets).
  - Retransmit (Fast Retransmission).
  - Each time another DUP ACK arrives, increments Window by 1.
  - When an ACK acks new data, sets window = Threshold and enter Congestion Avoidance.

Multiple losses in a window

- With two losses in a window, Reno will occasionally timeout.
- With three losses in a window, Reno will usually timeout.
- With four losses in a window, Reno is guaranteed to timeout!
- With three or more losses in a window, Tahoe typically out performs Reno!
  - It retransmits all the packets from the lost one (thus including eventual other lost packets)
  - Reno restarts from 1 packet after a further RTO with a ssthesh smaller than Tahoe

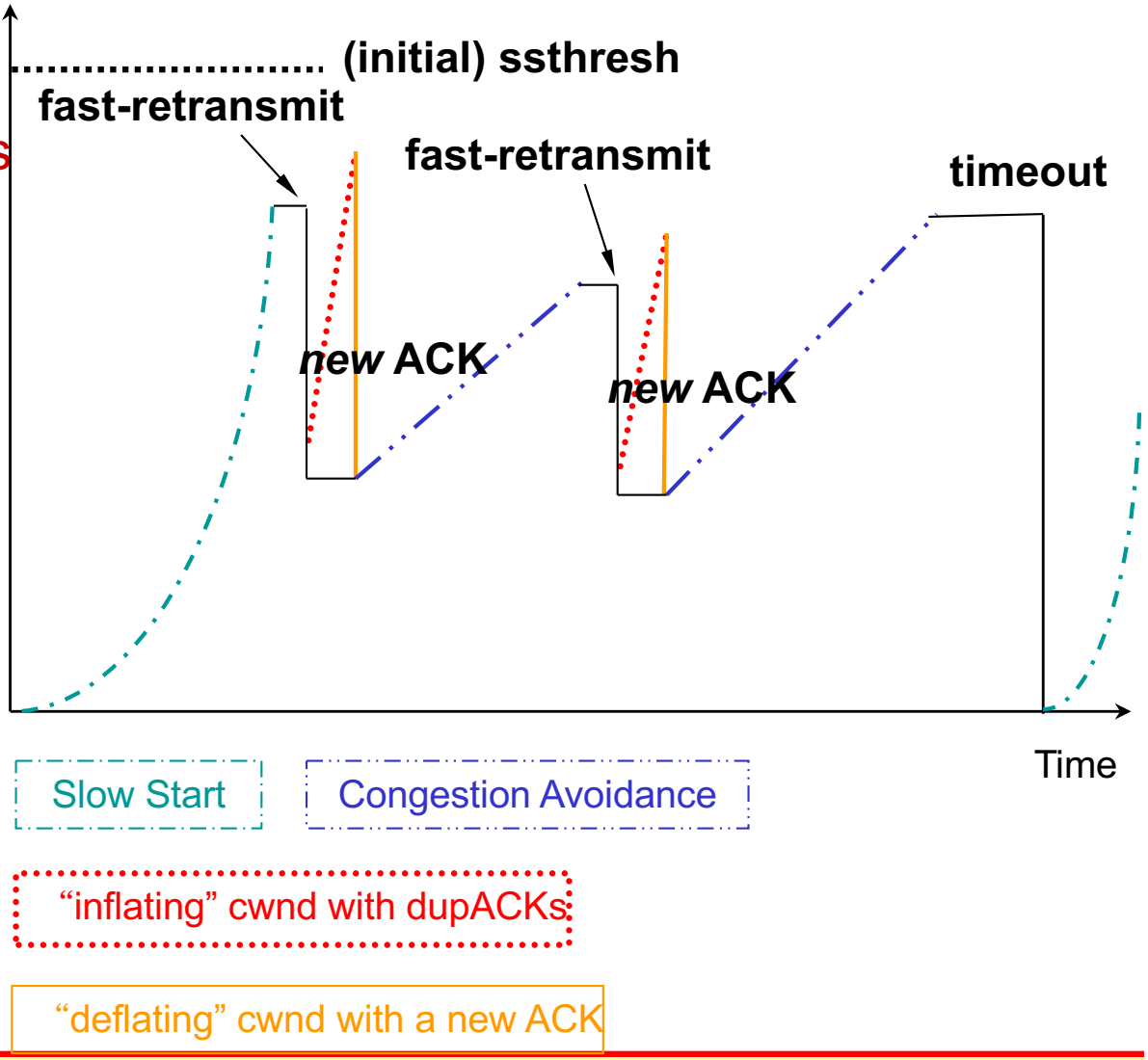
Fast Retransmit & Fast Recovery

- After receiving *3 dupACKS:*
  1. Retransmit the lost segment.
  2. Set ssthresh = flight size/2.
  3. Set cwnd = ssthresh, and ndupacks = 3.
     In Reno: send_win = min (rwnd, cwnd + ndupacks).
- If dupACK arrives:
  - ndupacks = ndupacks +1
  - Transmit new segment, if allowed.
- If new ACK arrives:
  - ndupacks = 0
  - Exit fast recovery.
- If RTO expires:
  - ndupacks = 0
  - Perform slow-start - (ssthresh = flight size/2, cwnd = 1 * MSS)

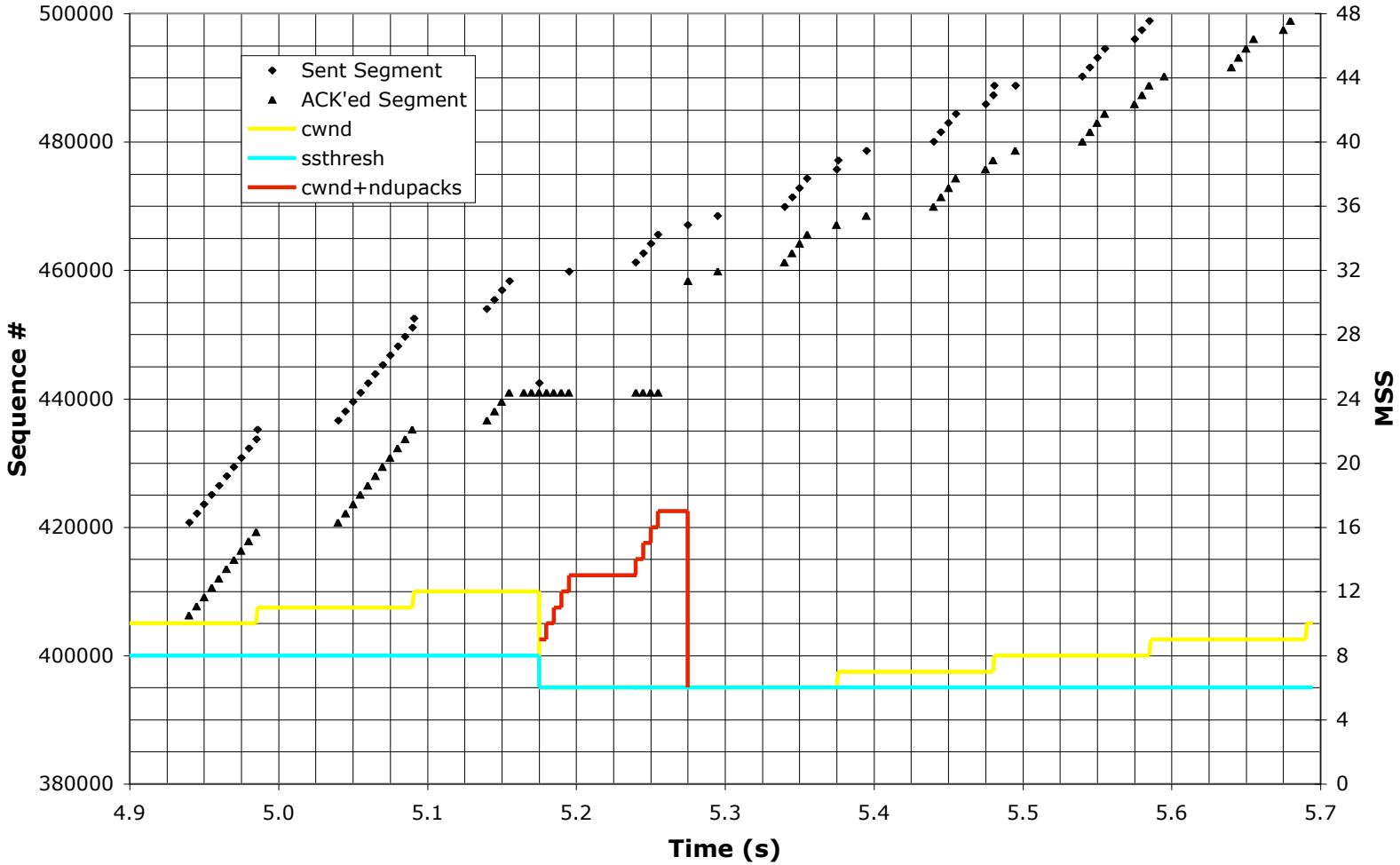
Fast Recovery

Concept

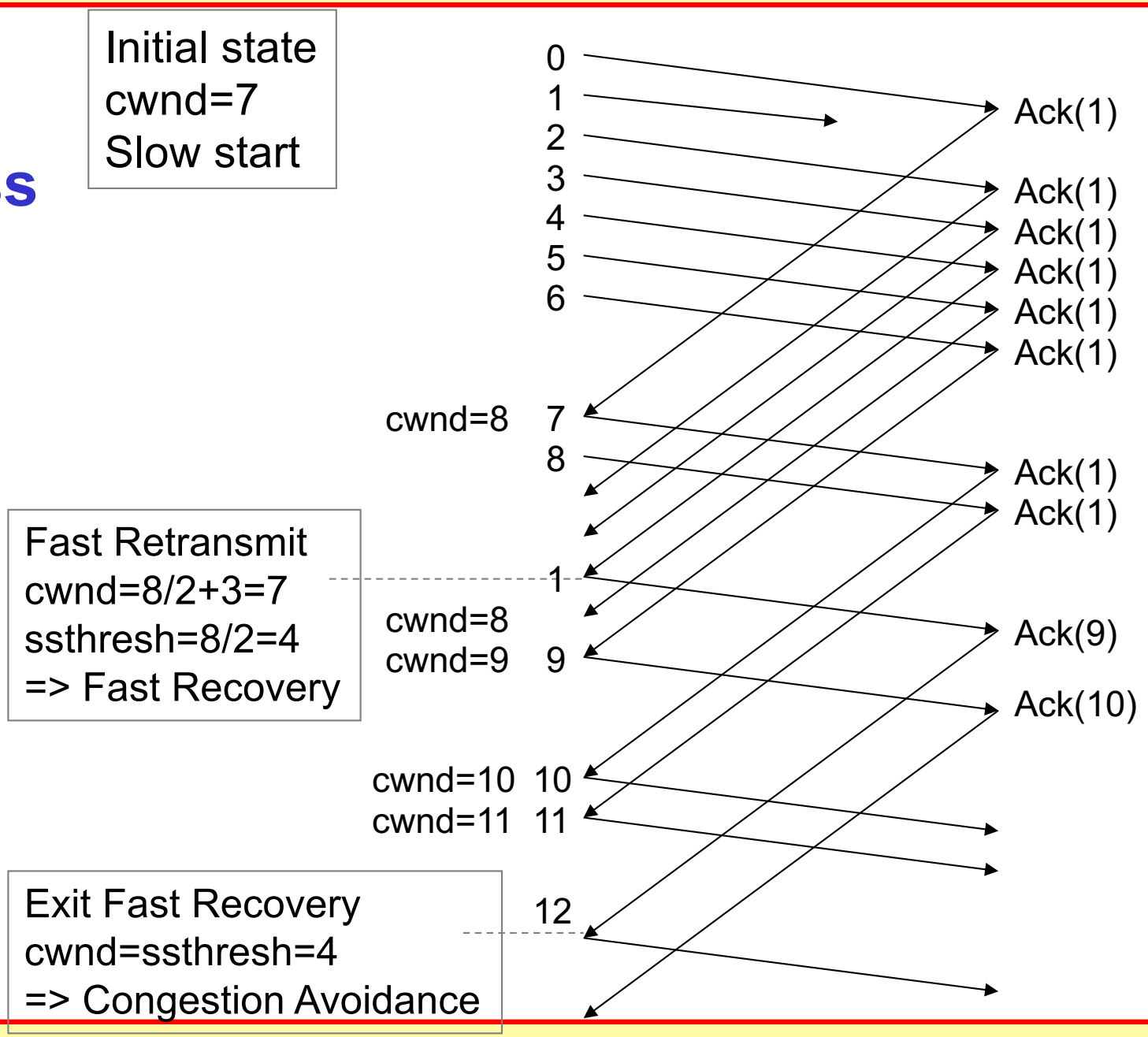
- After fast retransmit, reduces cwnd by half, and continue sending segments at this reduced level.
- The sender inflates cwnd by one on a dupACK



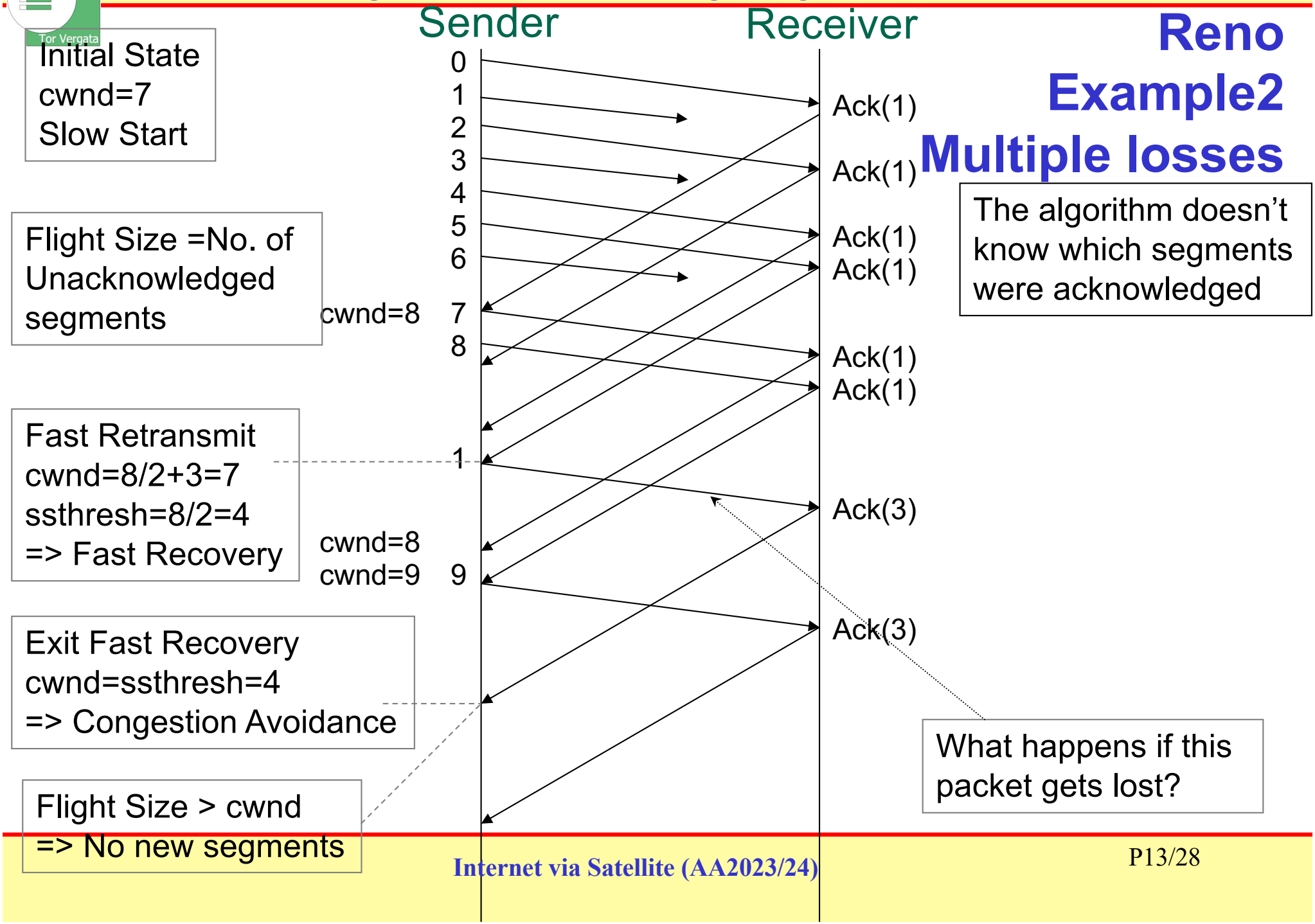
TCP Reno Trace – (with 1 dropped segment)



Reno Example1 Single loss



Reno Example2 Multiple losses



TCP NewReno

- **Idea:** If the sender will remember the number of the last segment that was sent before entering the Fast Retransmit phase, then it can find out if a "new" ACK (which is not **duplicate ACK**) does not include the last remembered segment (**"partial ACK"**), i.e. this is an evidence that more packets were lost before entering the Fast Retransmit.
- After becoming aware of such situation the sender will retransmit the next lost packet too and will stay at the Fast Recovery stage (the window is not halved)
- The sender will finish the Fast Recovery stage when it will get ACK that includes the last segment sent before the Fast Retransmit

TCP New Reno – Faster Recovery

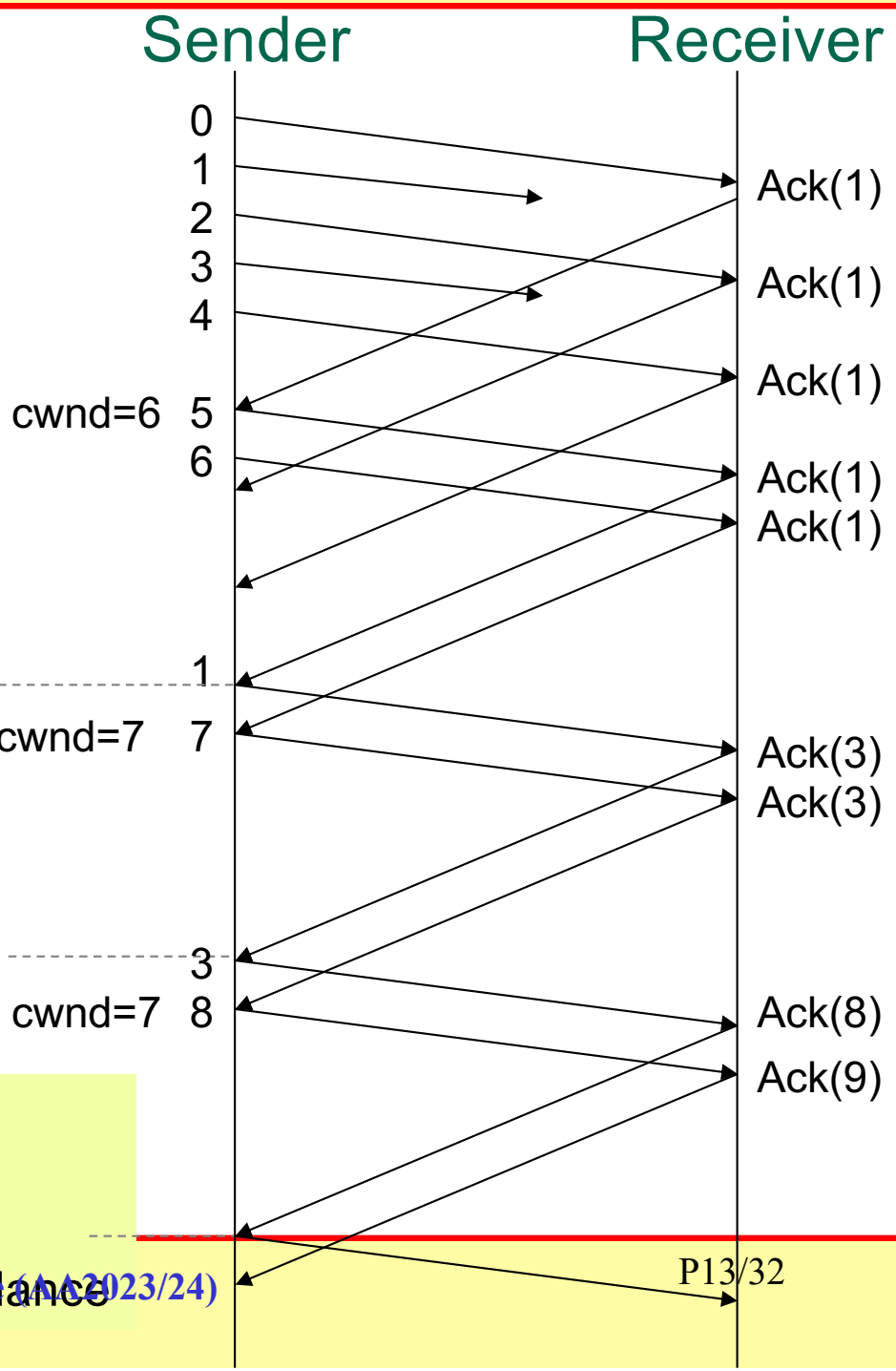
- New with respect to Reno:
  - Fast retransmit & modified fast recovery (faster recovery)
- Introduces partial ACK
  - Special ACK allowing to not exit Fast Retransmit and Fast Recovery
- Always allows to recover multiple losses in the same window
  - Continuing Fast Retransmit and Fast Recovery until all the ACKs arrive
- Disadvantage
  - 1 RTT to recover each packet lost

TCP NewReno – Retransmission due to Partial ACK

- When the sender gets a partial ACK, it retransmits the first unacknowledged segment
- Deflates the congestion window by the amount of new acknowledged data, then adds back one MSS (Maximum Segment Size) and sends a new segment if permitted by the new value of cwnd
  - This "partial window deflation" attempts to ensure that, when Fast Recovery eventually ends, approximately ssthresh amount of data will be outstanding in the network
  - Fast Recovery procedure (i.e., if any duplicate ACKs subsequently arrive, increment cwnd by MSS) doesn't stop.

New Reno Example

Initial State
cwnd=5
Slow Start



Fast Retransmit
cwnd=6/2+3=6
ssthresh=6/2=3
Recover=6
=> Fast Recovery

Recover >= Ack
Partial Ack
cwnd=7-(3-1)+1=6

Recover < Ack
Exit Fast Recovery
cwnd=ssthresh=3
=> Congestion Avoidance

SACK Option

- TCP SACK (Selective Acknowledgement)
  - ACK packet by packet
  - Allows to recover multiple losses in the same window within the same RTT
  - Alternative to CUM ACK
  - Requires a large TCP Header
  - Requires modification on both the TCP sender and destination side

Non satellite specific solutions to improve performance

- Physical layer
- Link layer
- Enhancement to congestion control
- Non standard algorithms acting on congestion control

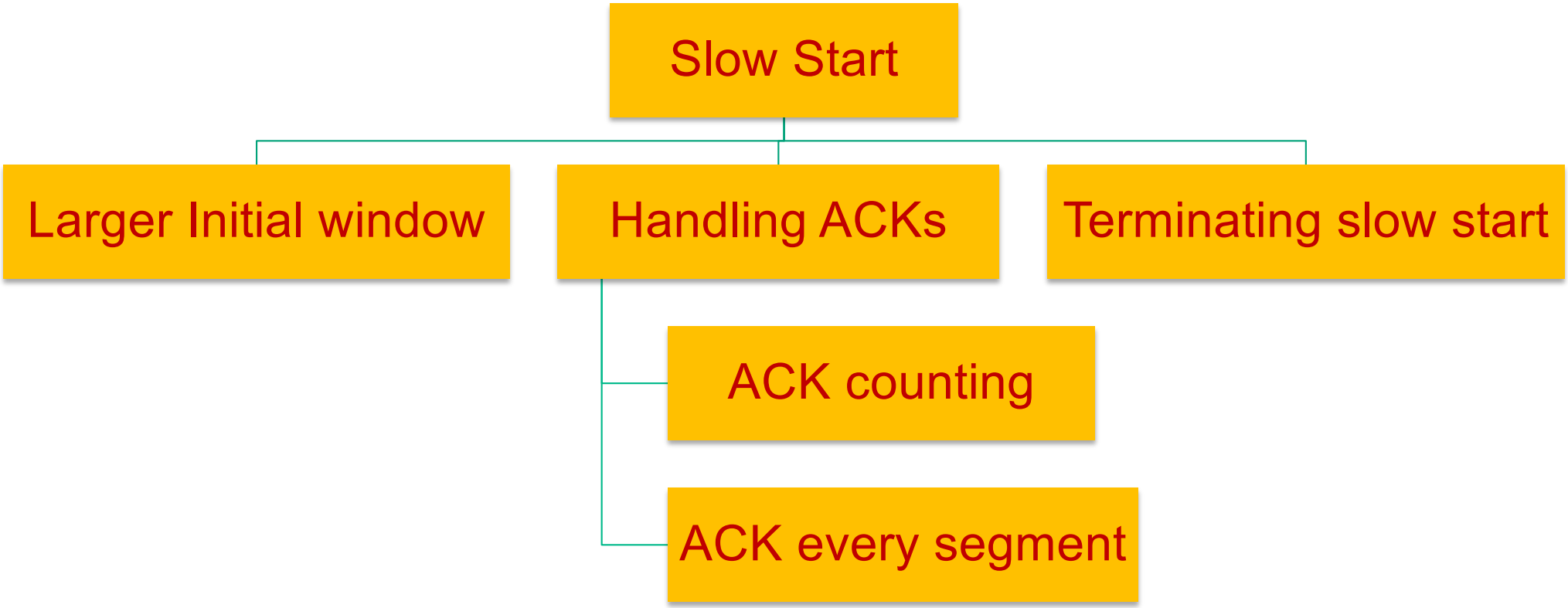
Physical and Link Layer Solutions to Satellite TCP

- Aim: to reduce losses not due to congestion
- Attempt to correct channel losses
  - FEC (Forward Error Correction)
    - Requires extra hardware, may be expensive.
    - Reduces bandwidth efficiency
    - May not work effectively in presence of large fading (due to rain etc) or when channel is down due to shadowing.
  - ARQ (Automatic Repeat Request)
    - Retransmission at the link layer,
    - This may cause an increase of the RTT

Transport layer solutions

- Modifications of TCP working process in terms of
  - Enhancement/options to standard TCP
  - Congestion control/Error recovery
  - Architecture

Enhancements to Slow Start



Slow Start Proposals (A)

- Larger initial window
  - Reduces Slow Start duration
  - Good for short-lived TCP connections or links with long RTT
  - Requires modification only on the TCP sender side
- ACK Counting (Byte Counting)
  - Window grows with the number of BYTES in the ACK packets (not with ACKs which can be relative to more packets of different dimension)
  - The growth depends on the actual quantity of data
  - UBC, Unlimited Byte Counting (number of non ACK bytes)
    - May cause congestion
  - LBC, Limited Byte Counting (limits growth to 2 segments)
    - Better performance than UBC
  - Allowed during Congestion Avoidance and not during Slow Start
  - Requires modification only on the TCP sender side

Slow Start Proposals (B)

- ACK every segment
  - Turning off delayed ACK during SS (DAASS, Delayed ACKs After Slow Start) the window grows more aggressively
  - The destination should be aware of the congestion control phase
- Terminating Slow Start
  - cwnd = ssthresh
  - Optimum value for ssthresh would allow bandwidth utilization to ramp up aggressively
  - No reliable way to measure ssthresh!
  - Requires modification on the TCP sender side but may require the destination too to predict available bandwidth

Enhancements to other mechanisms

- Delayed DUP ACK
  - Less traffic on the return link
  - Requires modification only on the TCP destination side
- Reduce Slow Start duration
  - ⇒ Better utilization of link capacity
- Identification of loss due to corruption
  - ⇒ Difficult to achieve (the sender is not informed when a corrupted packet is discarded)
- Avoid Congestion
  - ⇒ TCP reduces *cwnd*
- Recovery after loss
  - Sporadic Loss ⇒ Not necessary reduce the cwnd
- High BW * RTT Product.
  - Congestion Window may take long time to increase.
  - Use larger TCP packets
    - Path MTU discovery.
    - Send the largest packet the network can take and set the "don't Fragment" bit.
  - T/TCP (TCP for transactions)
    - Bypass Handshake and Slow Start (only at the beginning)
    - Use parameters from previous cached transactions.
    - After packet loss (time out or dup ack) restarts from slow start
    - Useful for short connections.
    - Sends data within SYNC packet
    - Requires modifications on both the sender and the destination side

Enhancements to other mechanisms (2)

- Explicit Congestion Notification (ECN)
  - A router going to experience congestion notifies the state to the sender
  - Additional functions needed at both router and sender sides
  - Forward notification ⇒ the destination is involved too
- Congestion Avoidance in shared links
  - Different connections utilize the same link with different RTT ⇒ Different growth rates for cwnd
  - The increments are equalized
- ACK congestion control
  - Congestion Notification on return link
- Multiple Connection
  - Larger initial cwnd, larger growth
  - Less sensitivity to sporadic loss:

N connections with cwnd=W⇒total cwnd = N*W

Single loss⇒total cwnd reduces to (N*W)-(W/2)

- TCP Header Compression
  - The first overhead full while in the following only the variations
  - Implemented on both ends utilizing compression
- Sharing state among similar connections
  - The same parameters of similar connections are set up (hosts belonging to the same network) thus avoiding Slow Start

New Congestion Control Algorithms: Vegas

- Introduced by Brakmo and Peterson (1994)
- Three changes to TCP Reno
  - Modified congestion avoidance
    - Don't wait for a timeout, if *actual throughput* < *expected throughput* the congestion window is decreased (Additive Increase Additive Decrease - AIAD)
  - New retransmission mechanism
    - *motivation: what happens if* sender never receives 3-dupACKs (due to lost segments or window size is too small.)
    - mechanism: sender does retransmission after a dupACK received, if RTT estimate > timeout.
  - Modified slow start
    - *motivation:* sender tries finding correct window size without causing a loss.

New Congestion Control Algorithms: Westwood

- Proposed by UCLA
- Keeps a measure of the RTT_{min}.
- Tries to estimate BW available.
  - BW = Size / $(t_k - t_{k-1})$
- When timeout occurs
  - Set window to 1 and threshold to BW * RTT_{min}.
  - Speedy recovery from Slow Start.
- When 3 DUP ACKs are received
  - Set threshold to BW * RTT_{min}.
  - Set Window to threshold (if Window > Thresh)

TCP in the Real World

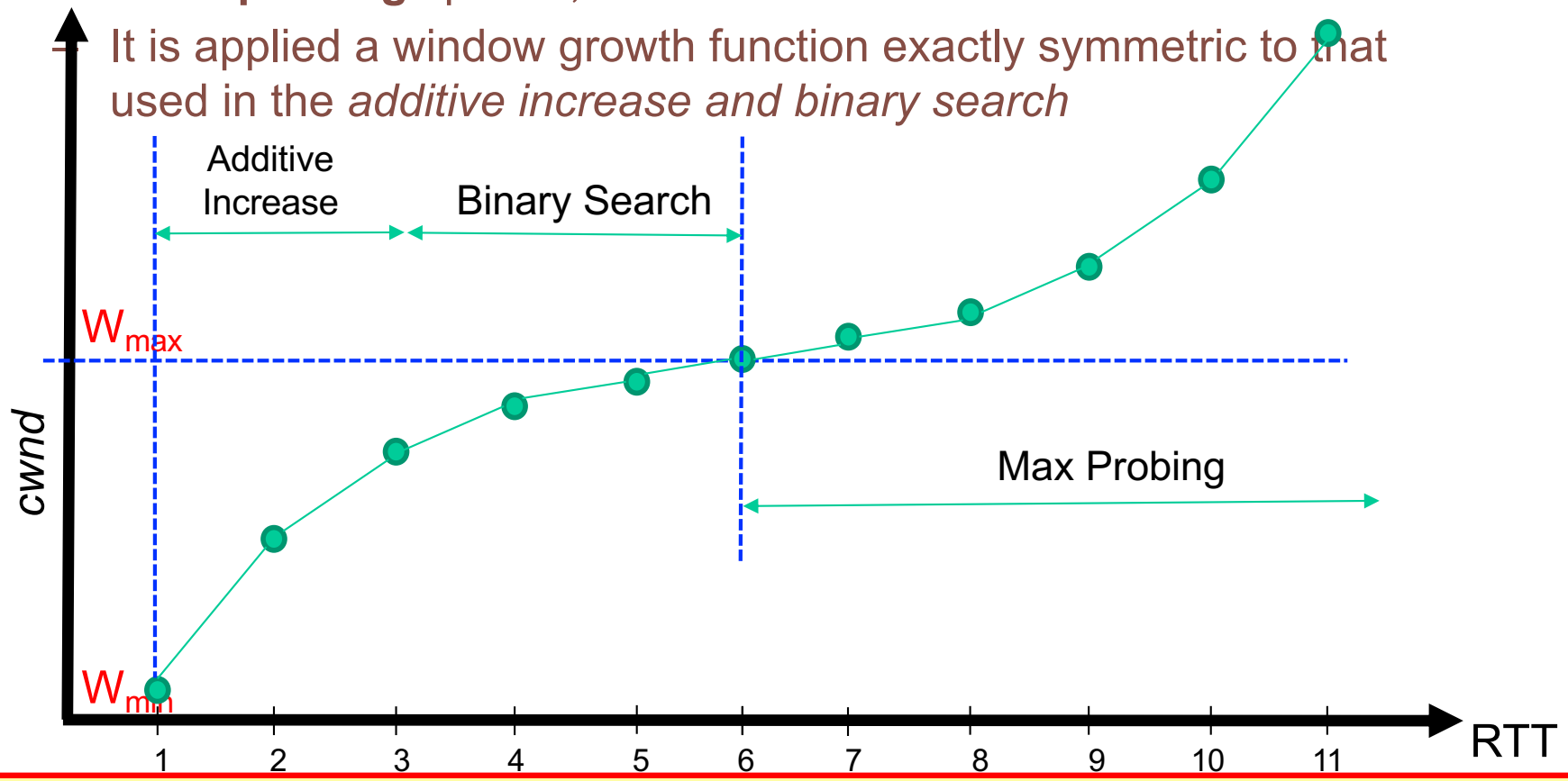
- Most popular variants today
  - Main target: TCP performs poorly on high bandwidth-delay product networks (like the modern Internet)
  - Compound TCP (Windows)
    - Based on Reno
    - Uses two congestion windows: delay based and loss based
    - Thus, it uses a *compound* congestion controller
  - TCP CUBIC (Linux)
    - Enhancement of BIC (Binary Increase Congestion Control)
    - Window size controlled by cubic function
    - Parameterized by the time *T* since the last dropped packet
  - BBR TCP (Linux-pushed by Google-same designer of TCP Tahoe ☺)
    - Outcome of the "*Make TCP Fast*" at Google
    - A recent revolutionary approach for TCP congestion control
    - Estimation of the BDP through two simultaneous measurements: RTT_{min} and *bottleneck BW*
    - Adaptation of supplementary mechanisms (out of TCP box): pacing, TCP segmentation offload (TSO), TCP small queue (TSQ).

TCP BIC (Binary Increase Congestion control)

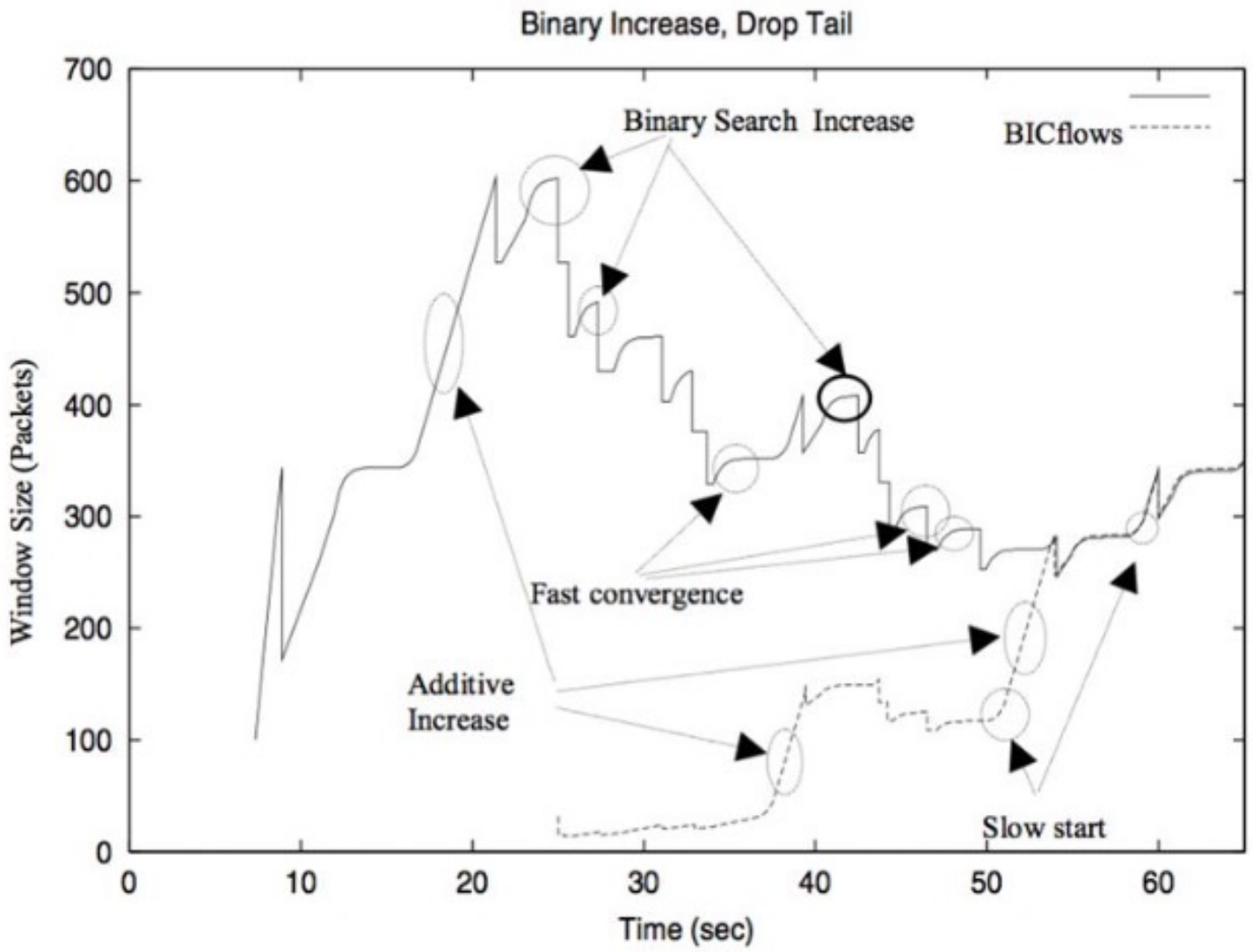
- The window size increase follows a so-called **additive increase** and **binary search**
- Slow Start
- After a loss:
  - W_{max}= window at the loss (similar to ssthresh)
  - W_{min}=**β***(window at the loss) → multiplicative decrease
- The window increase is given by $S_{min} < \frac{W_{max}-W_{min}}{2} < S_{max}$
  - S_{min} minimum allowed increment
  - S_{max} maximum allowed increment
  - The growth is initially linear and then becomes logarithmic
- In case of no losses at the updated window
  - New window becomes W_{min}
  - The window increase is repeated until $\frac{W_{max}-W_{min}}{2} < S_{max}$; in this case new window is set to W_{max}

TCP BIC (Binary Increase Congestion control)

- If W_{max} is achieved without losses, the equilibrium window size must be larger than the current W_{max}, then a new "maximum" must be found
  - **"max probing"** phase;
  - It is applied a window growth function exactly symmetric to that used in the *additive increase and binary search*



TCP BIC fairness



TCP CUBIC implementation

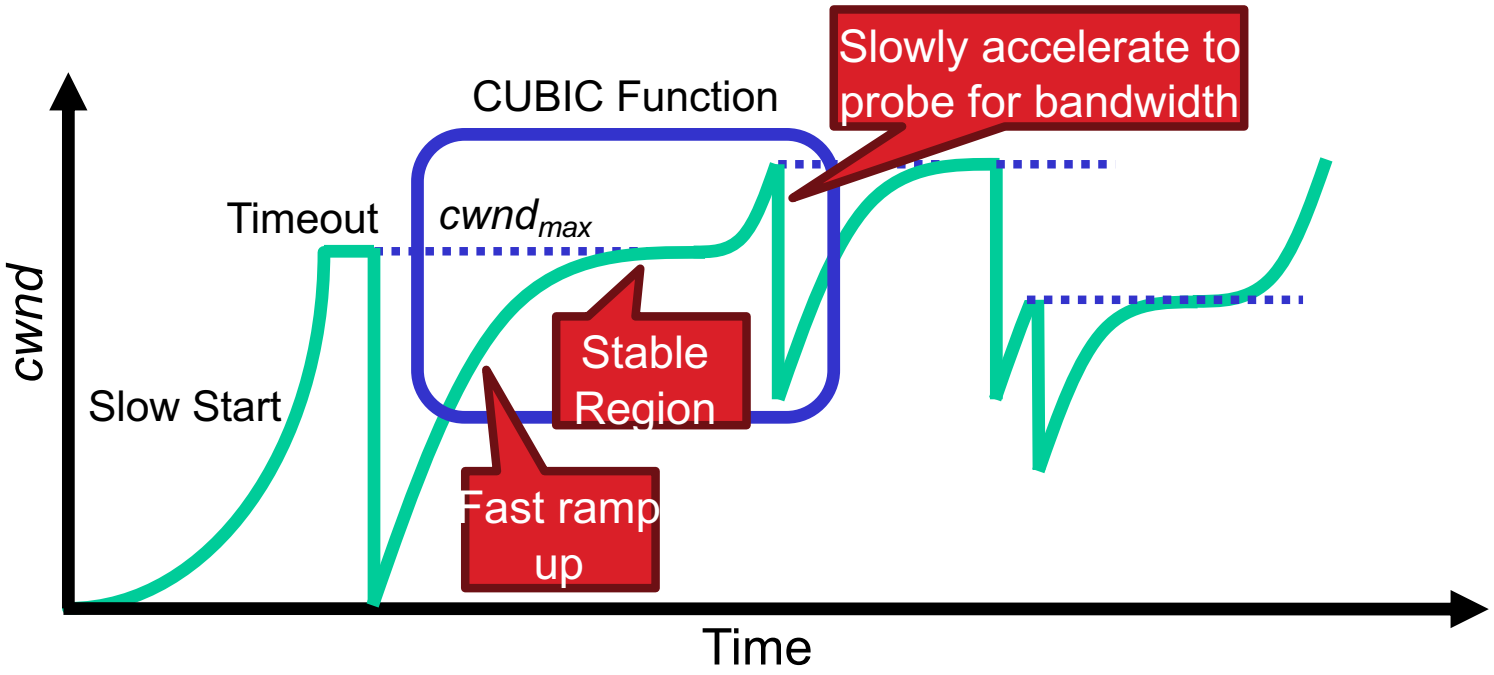
- Default TCP implementation in Linux
- Replaces BIC increase with a cubic function

$$W_{cubic} = C \cdot (T - K)^3 + W_{max}$$

C is a scaling constant $K = \sqrt[3]{\frac{W_{max} \cdot \beta}{C}}$

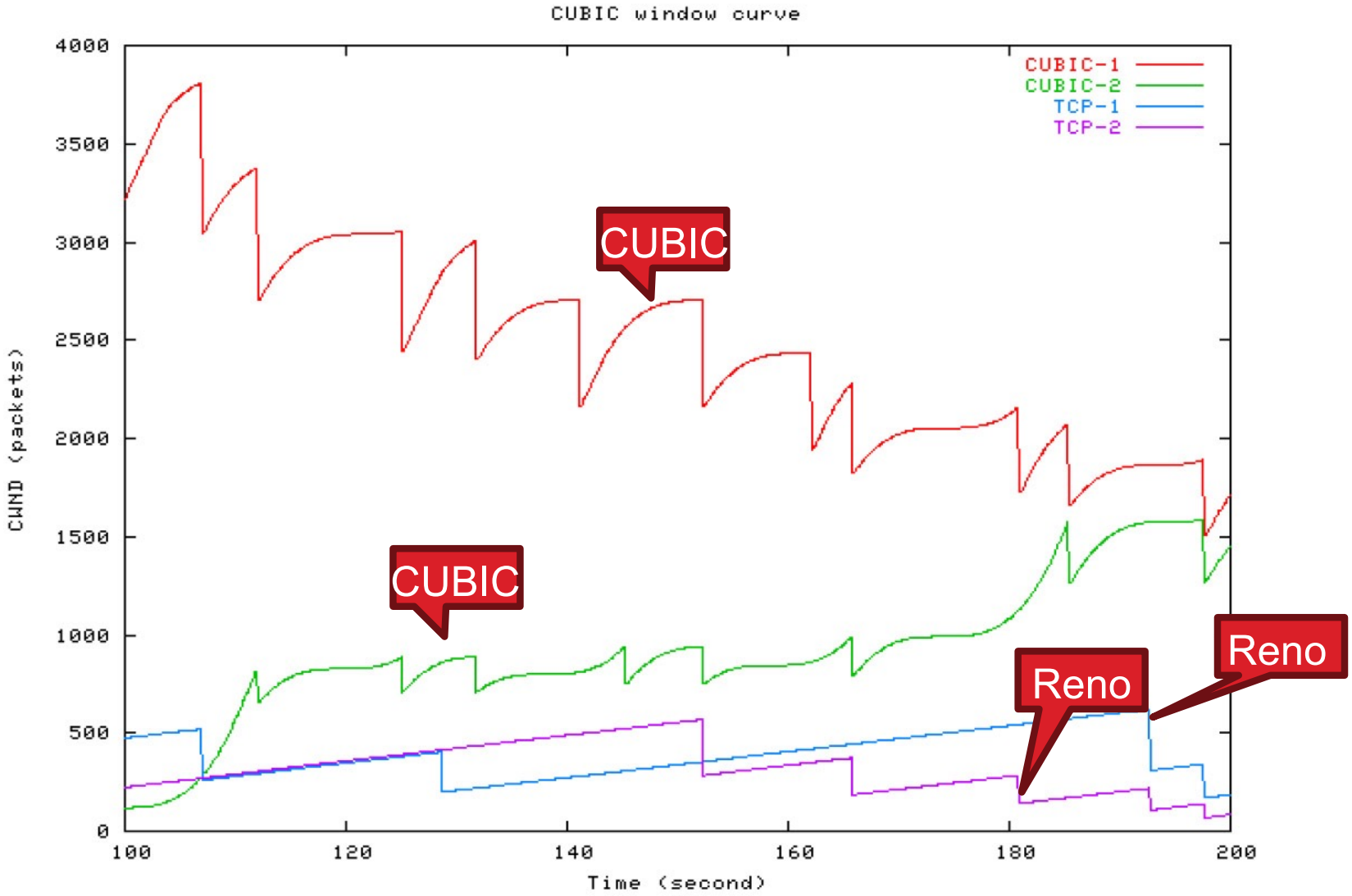
  - β → a constant fraction for multiplicative increase
  - T → time since last packet drop
  - W_{max} ➔ cwnd when last packet dropped

TCP CUBIC evolution in a satellite link

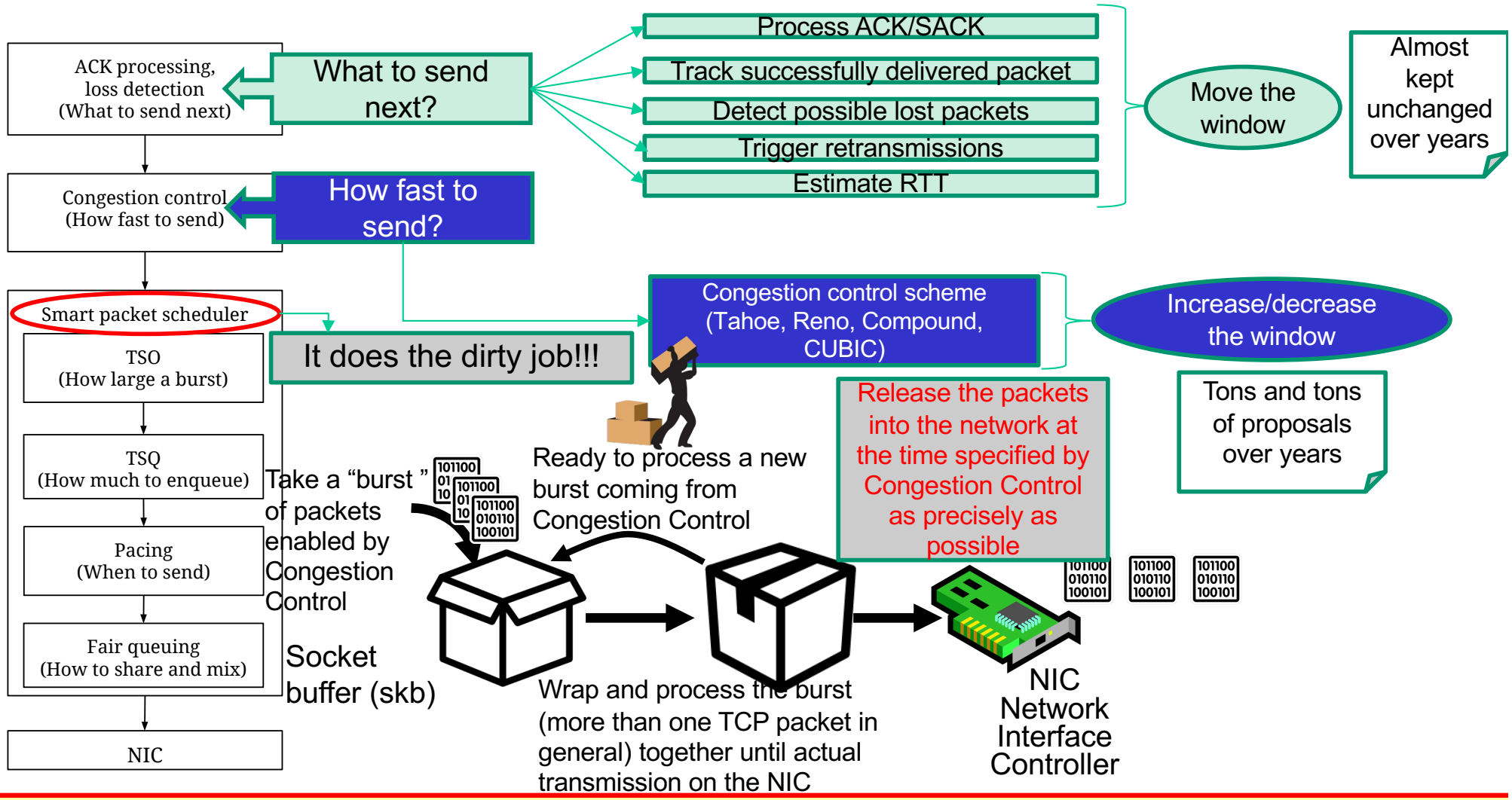


- Less wasted bandwidth due to fast ramp up
- Stable region and slow acceleration help maintain fairness
  - Fast ramp up is more aggressive than additive increase
  - To be fair to Tahoe/Reno, CUBIC needs to be less aggressive

TCP Cubic fairness and friendliness



Linux TCP sender real architecture



TSO (TCP Segmentation Offload): How large a burst?

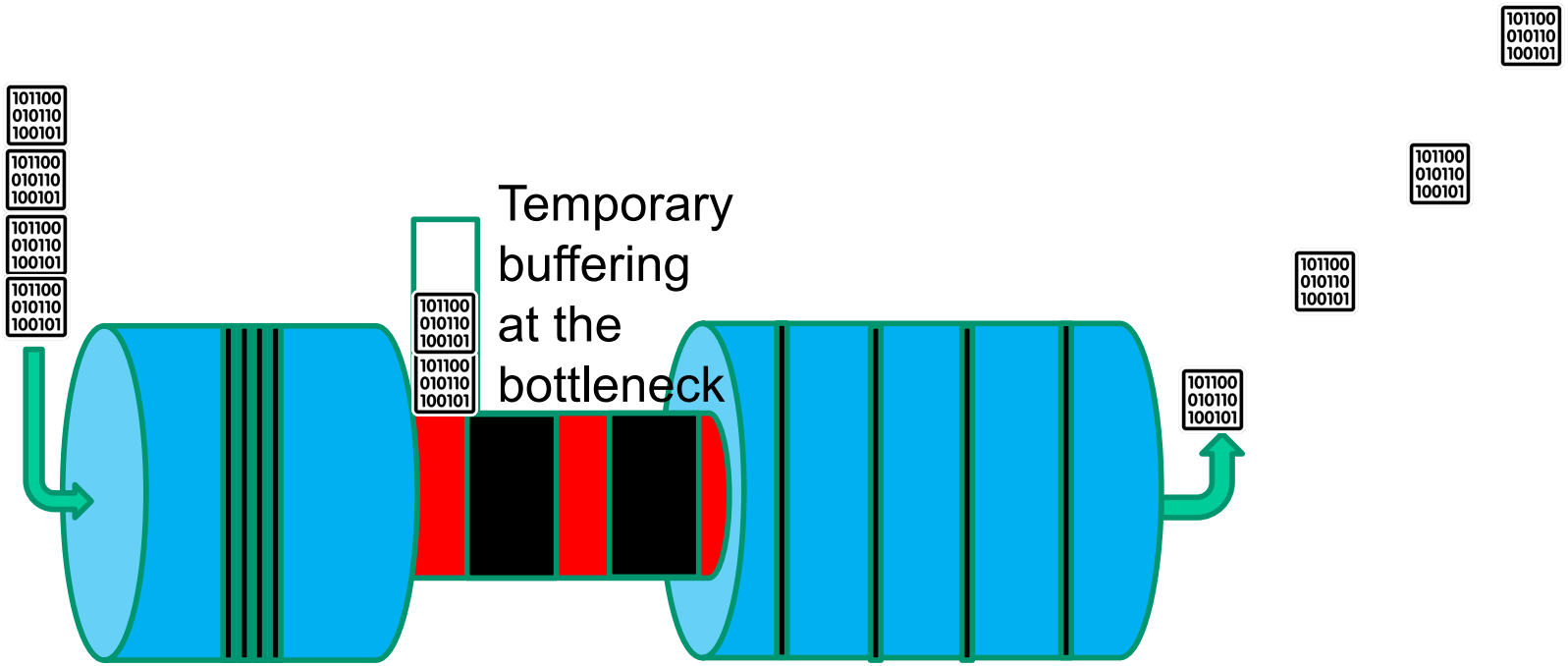
- To allow low CPU utilization even at high speed (*we expect the processing of a large number of packets at once*), Linux TCP sender employs segmentation offload mechanisms
  - TCP segmentation offload;
    - Group a number of packets and process them all together in a CPU clock, instead to process packets one by one (resource consuming!)
    - Bounded to a max of 64 kBytes
  - Balancing problem
    - Large bursts increase CPU efficiency
    - Small bursts make generated traffic most smoothed, avoiding temporary queuing in the network

TSQ: How much to enqueue?

- TCP Small Queue (TSQ) performs a local flow control, limiting the amount of data in the queues on the sending host!
  - Basically, irrespective of the cwnd value (that potentially could enable transmission of a very large amount of data) TSQ defines a limit on the data queued waiting for transmission
  - TSQ allows no more than 262 kByte (4 full-size TCP skbs) at the time
  - If TSQ is full (e.g. NIC is slow!), skb stops grabbing new data enabled by cwnd
- Even if cwnd grows at infinite, TSQ limits the actual transmission

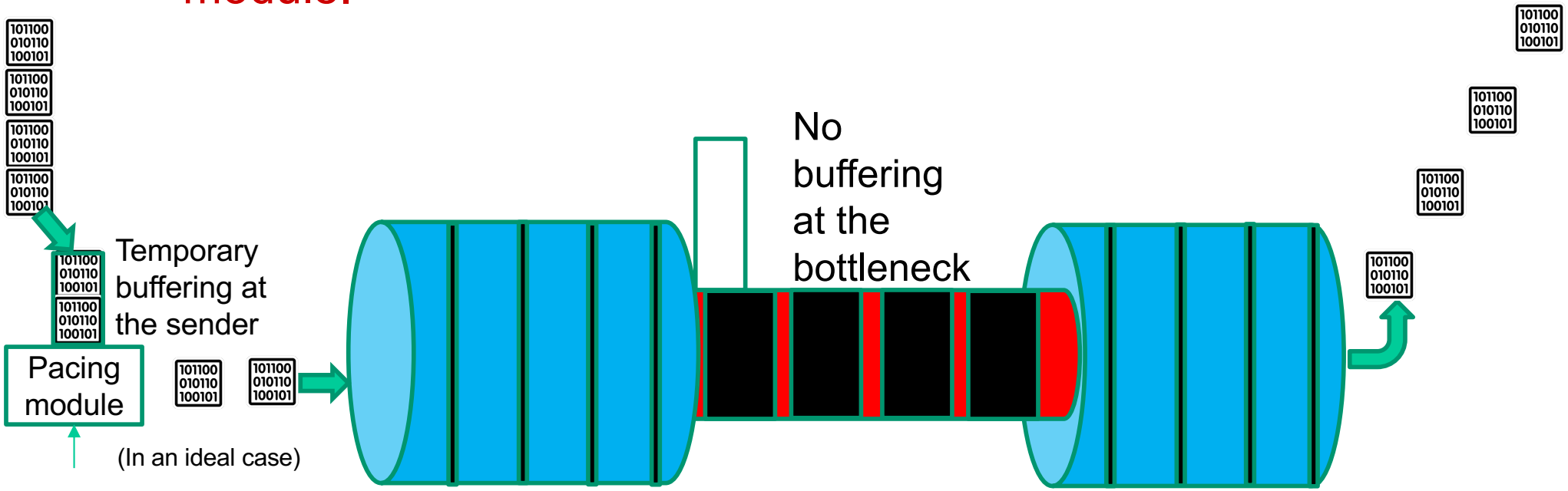
No Pacing: When to send?

- Without pacing a burst is sent as soon as it is available at the TCP source (according to *cwnd* and TSO operations)



Pacing: When to send?

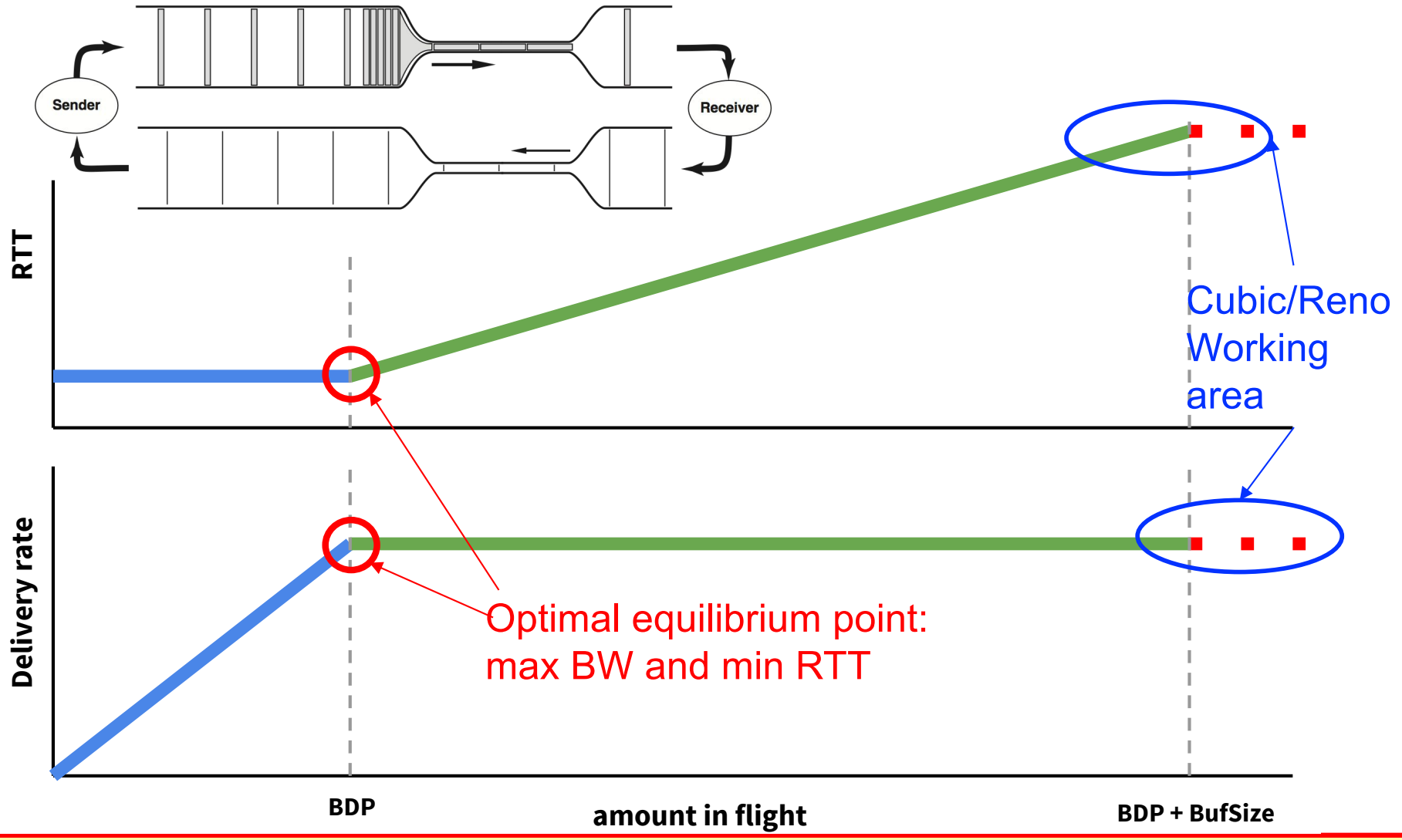
- The pacing component takes the chunks created by TSO and spreads them out in time, introducing empty time intervals between the chunks, according to the sending rate specified on the socket by the congestion control module.



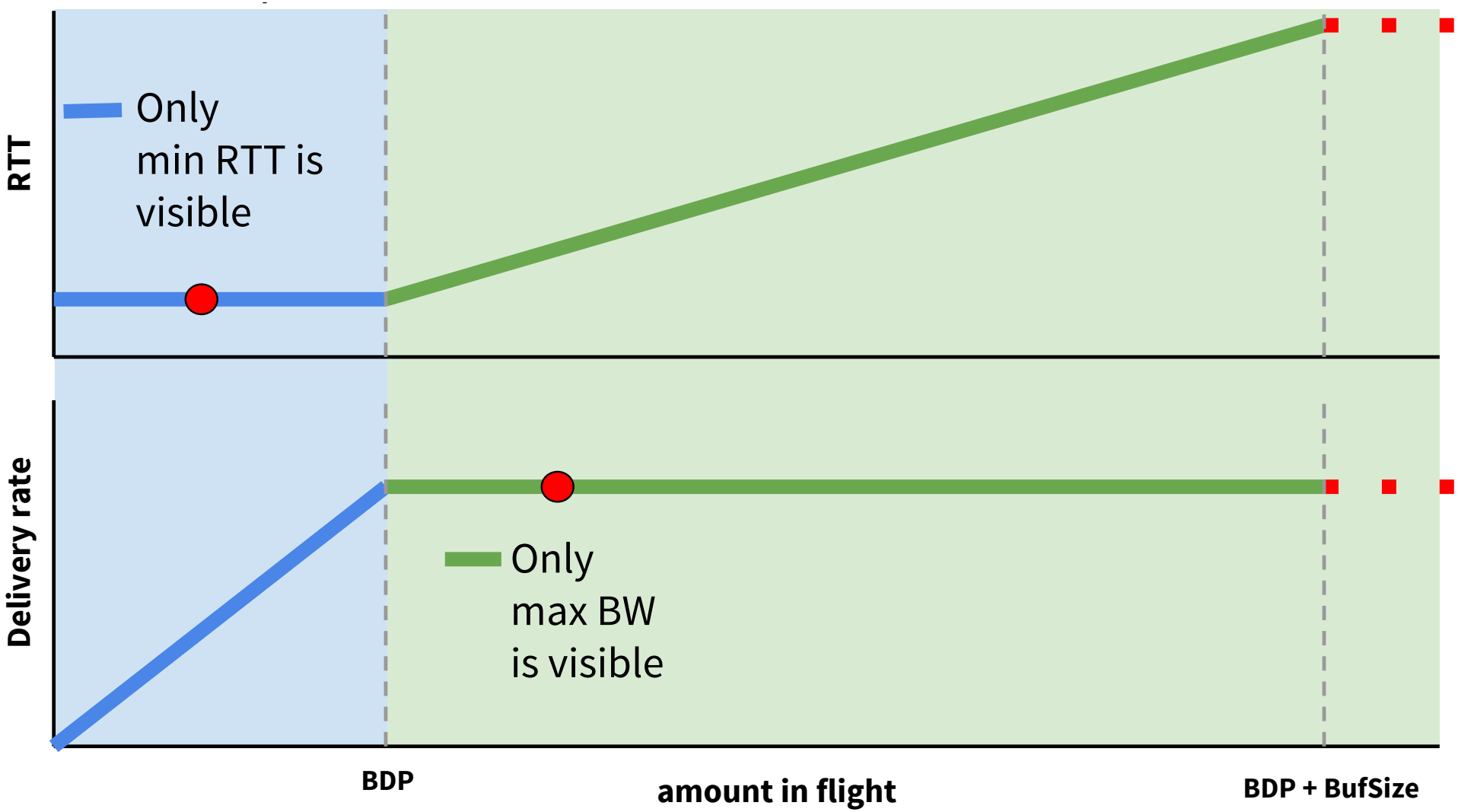
BBR congestion control

- A new Linux Congestion Control tailored for new TCP smart scheduler
  - BBR = Bottleneck Bandwidth and RTT
- Main target: MOVE THE BOTTLENECK AT THE SENDER
  - Once (If) assessed the bottleneck bandwidth, let's inject packets directly at such a rate → avoid buffering at the bottleneck
- Key objectives:
  - Measure the RTT_{min} (Rtprop = delay due to only propagation)
  - Measure the BtlBW = bandwidth at the bottleneck
- Bandwidth-Delay Product (BDP) = BtlBW * Rtprop
- Optimum delivery rate is achieve when *cwnd* = BtlBW * Rtprop

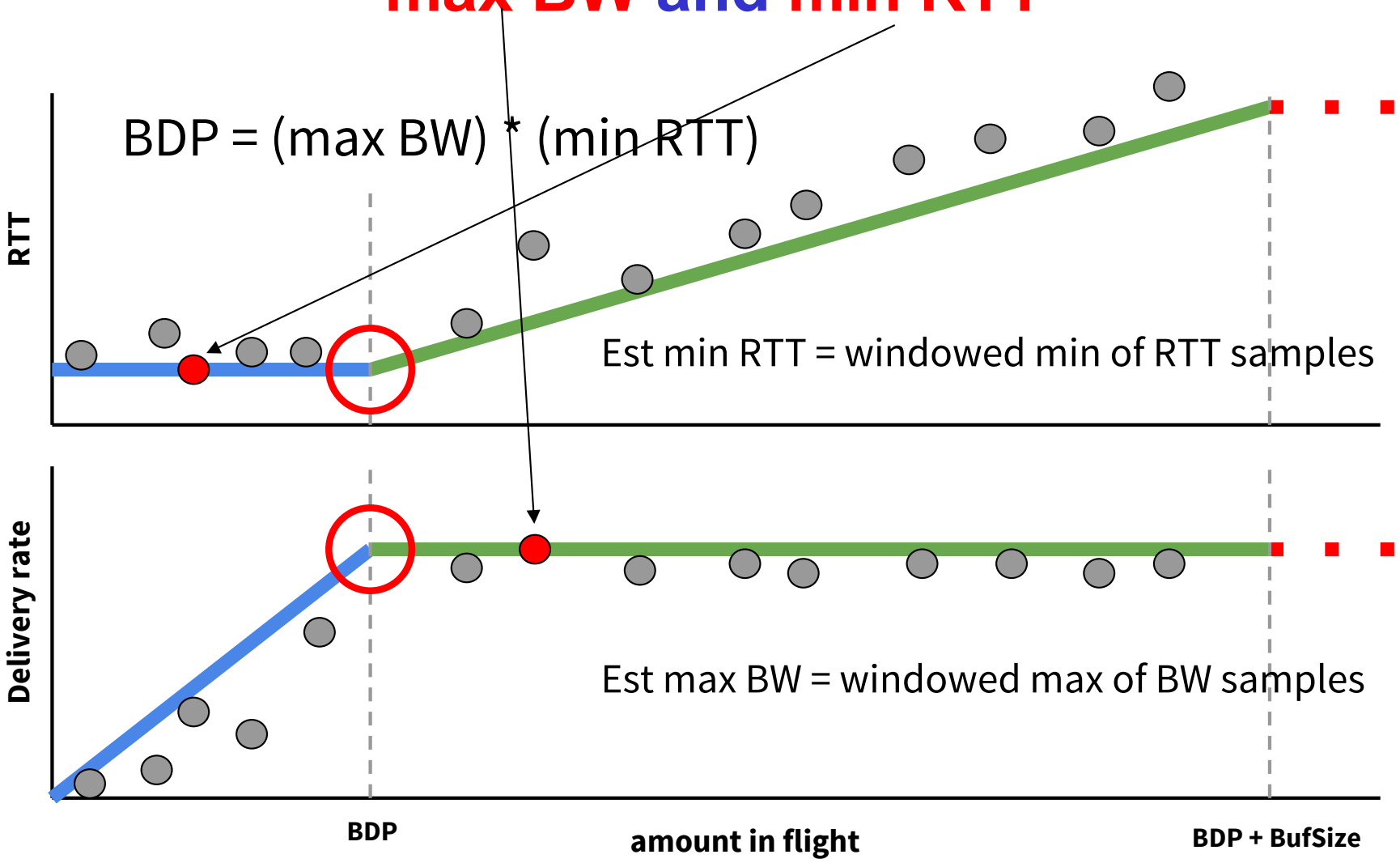
Congestion and bottleneck



How to "probe" optimal equilibrium point?



Estimating optimal equilibrium point: max BW and min RTT

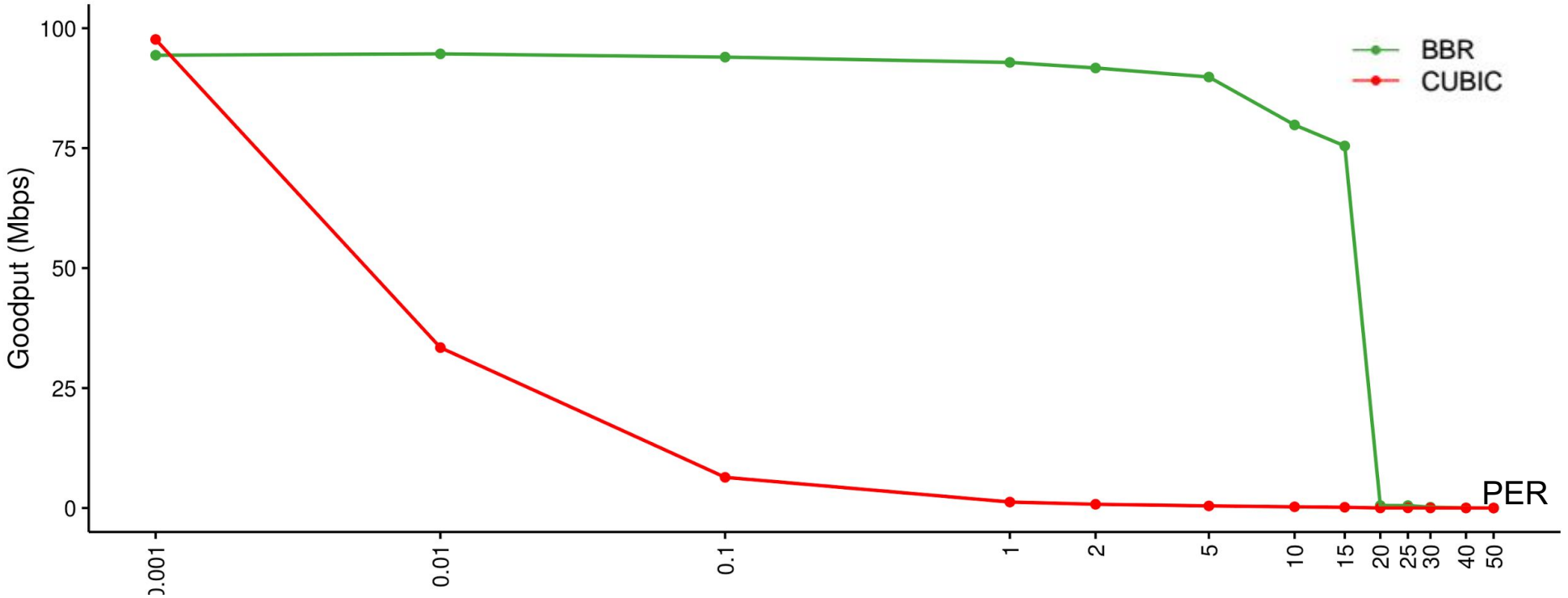


BBR algorithm

- Every ACK measures RTT and delivery time $\left(\frac{\Delta ACKed\_Data}{\Delta T}\right)$
  - Update *min_RTT* (10 second sliding window)
  - Update *max_rate* (10 round-trip sliding window)
- Paced sending rate defaults to the observed receiving rate
  - Basically transmission rate is controlled by pacing!
  - Also cwnd is set to *min_RTT*max_rate*
- Mostly sends (cwnd and pacing setting) to the default rate
  - Periodically increases rate to check if *max_rate* is changed
  - Periodically decreases rate to check if *min_RTT* is changed

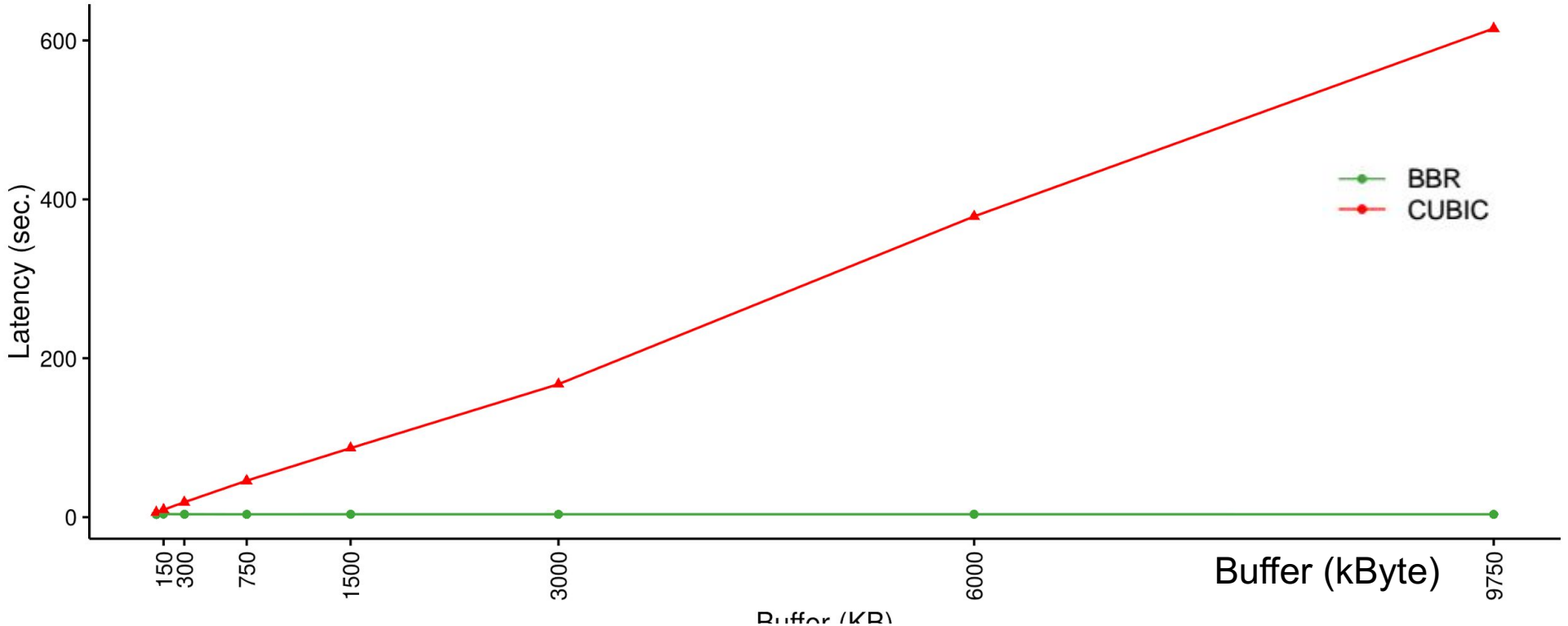
BBR performance

- Fully use bandwidth despite high loss



BBR performance

- Low queue delay despite bottleneck buffer size



Considerations on BBR

- Measurement intervals derived from "*experience*" (real measurement patterns)
  - Mirror of the past and partially of the current networks. What about new challenging scenarios?
  - How to react at fast changes? Which are the implications?
- Fairness (competition of BBR flows) and friendliness (competition with other flows, i.e. TCP or UDP) need investigation.
  - Convergence time
  - Impact of greedy flows on measurements
- Strong dependency on "smart sender scheduler"
  - Pacing is mandatory to support measurement assumptions
- Temporary RTT increases are considered as disasters!!!
  - Let's remember that in the last decade TCP developers assumed larger buffer, as a consequence RTT increase can not be completely avoided.
- Suitable for long transfers, limited and slow physical changes.

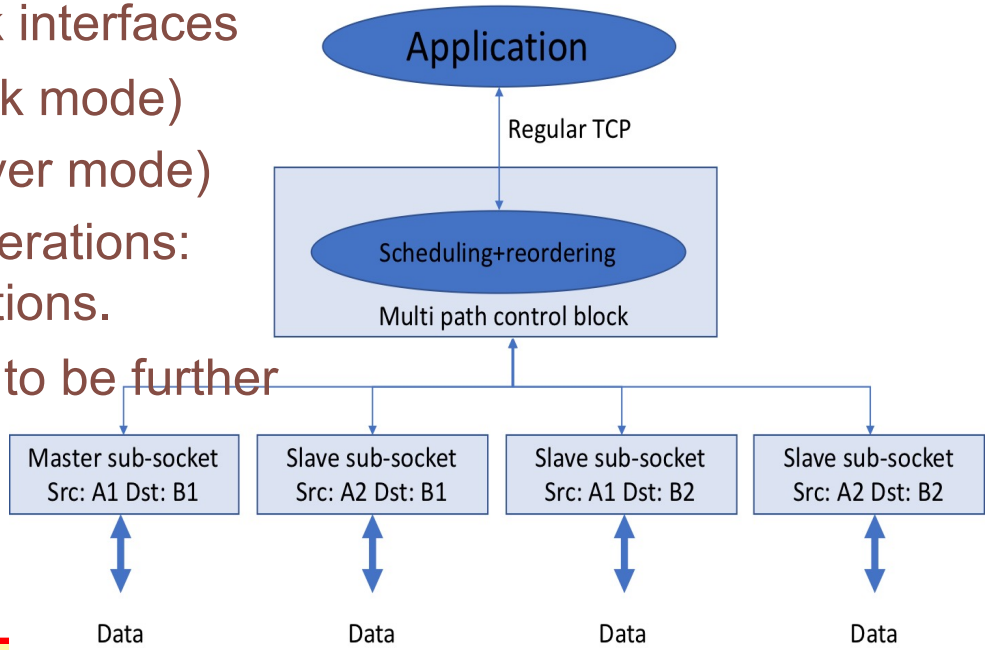
Alternative Internet protocol architectures

Starting issue

- TCP single-path protocol
- TCP connection is bound to the IP addresses of the two communicating hosts.
- If one of these addresses changes the connection fails.
- Loading the connection across more than one path results in packet reordering, and TCP misinterprets this reordering as congestion and slows down.

Advanced transport protocols

- MP (Multi Path)-TCP
  - Smart exploitation of multiple network interfaces
  - Tailored for long transfers (in multi-link mode)
  - Optimal to manage link failover (failover mode)
  - Need of a fine-debug of scheduler operations: detected problems in old implementations.
  - New release with better performance to be further validated.
- Multicast Reliable Transport protocol
  - FLUTE



MPTCP main characteristics

- Networks are intrinsically multipath (WiFi, 3G, 4G, etc.)
- Multipath TCP (MPTCP) is a major modification to TCP that allows multiple paths to be used simultaneously by a single transport connection.
- Multipath TCP allows multiple *subflows* to be set up for a single MPTCP session. An MPTCP session starts with an initial subflow, which is similar to a regular TCP connection as described above. After the first MPTCP subflow is set up, additional subflows can be established. Each additional subflow also looks similar to a regular TCP connection, complete with SYN handshake and FIN tear-down, but rather than being a separate connection, the subflow is bound into an existing MPTCP session. Data for the connection can then be sent over any of the active subflows that has the capacity to take it.
- The subflows are activated one by one and for each one a full three way handshaking is performed.

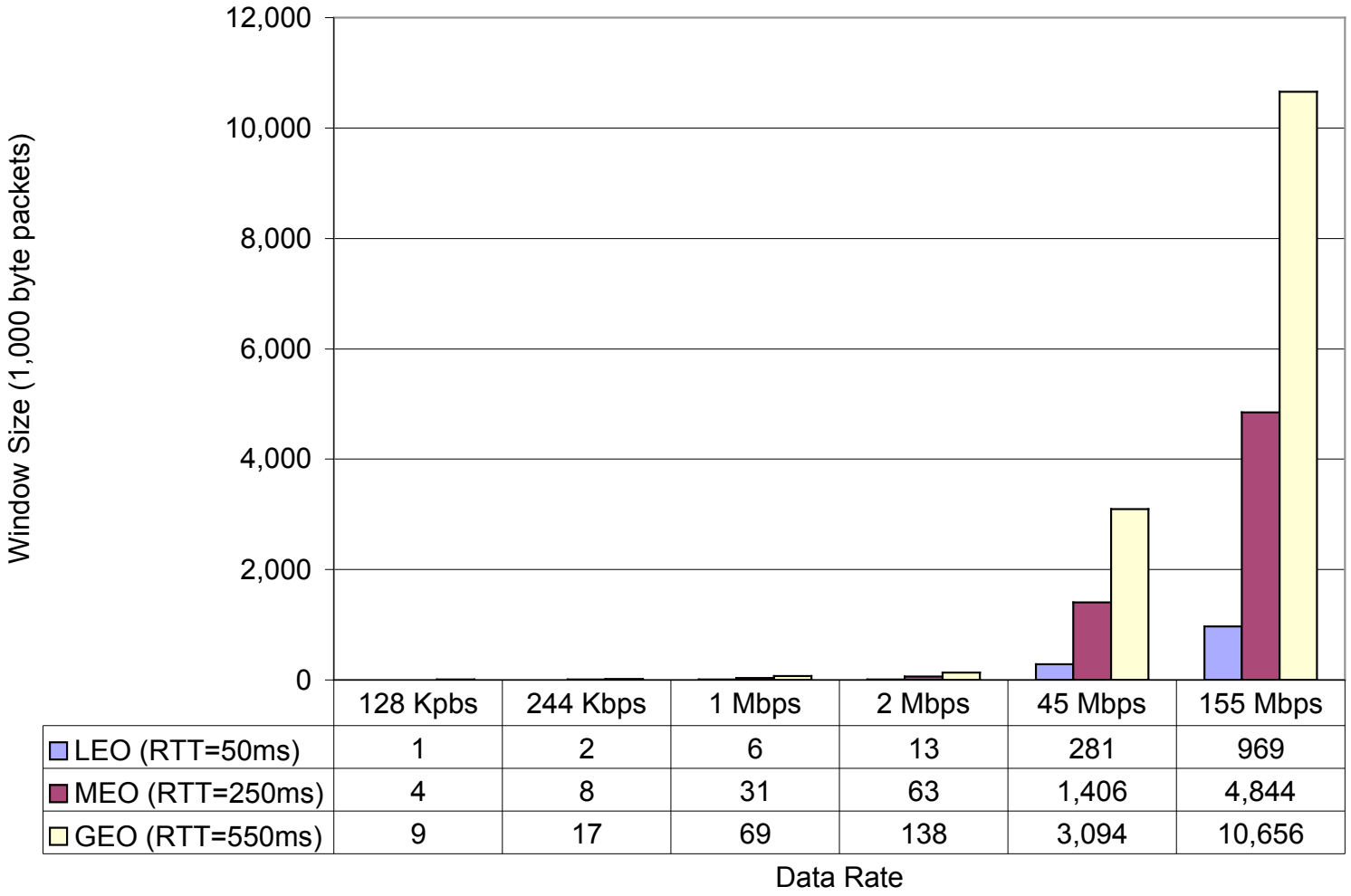
MPTCP main characteristics (2)

- The set of subflows that are associated with a Multipath TCP connection is not fixed. Subflows can be dynamically added and removed from a Multipath TCP connection throughout its lifetime. For example when (if) the terminal changes the network (WiFi access point or 3G/4G …) changing the relative IP address.
- Because the two paths will often have different delay characteristics, the data segments sent over the two subflows will not be received in order. Regular TCP uses the sequence number in each TCP packet header to put data back into the original order.
- Using TCP sequence number may not work because some middlebox (e.g. firewall) noting irregularity in the sequence numbers can discard the flow.
- Multipath TCP uses its own sequence numbering space. Each segment sent by Multipath TCP contains two sequence numbers: the subflow sequence number inside the regular TCP header, and an additional data sequence number (DSN) carried inside a TCP option.
- To perform congestion control, each TCP sender maintains a congestion window.

Impact of satellite link characteristics on TCP performance

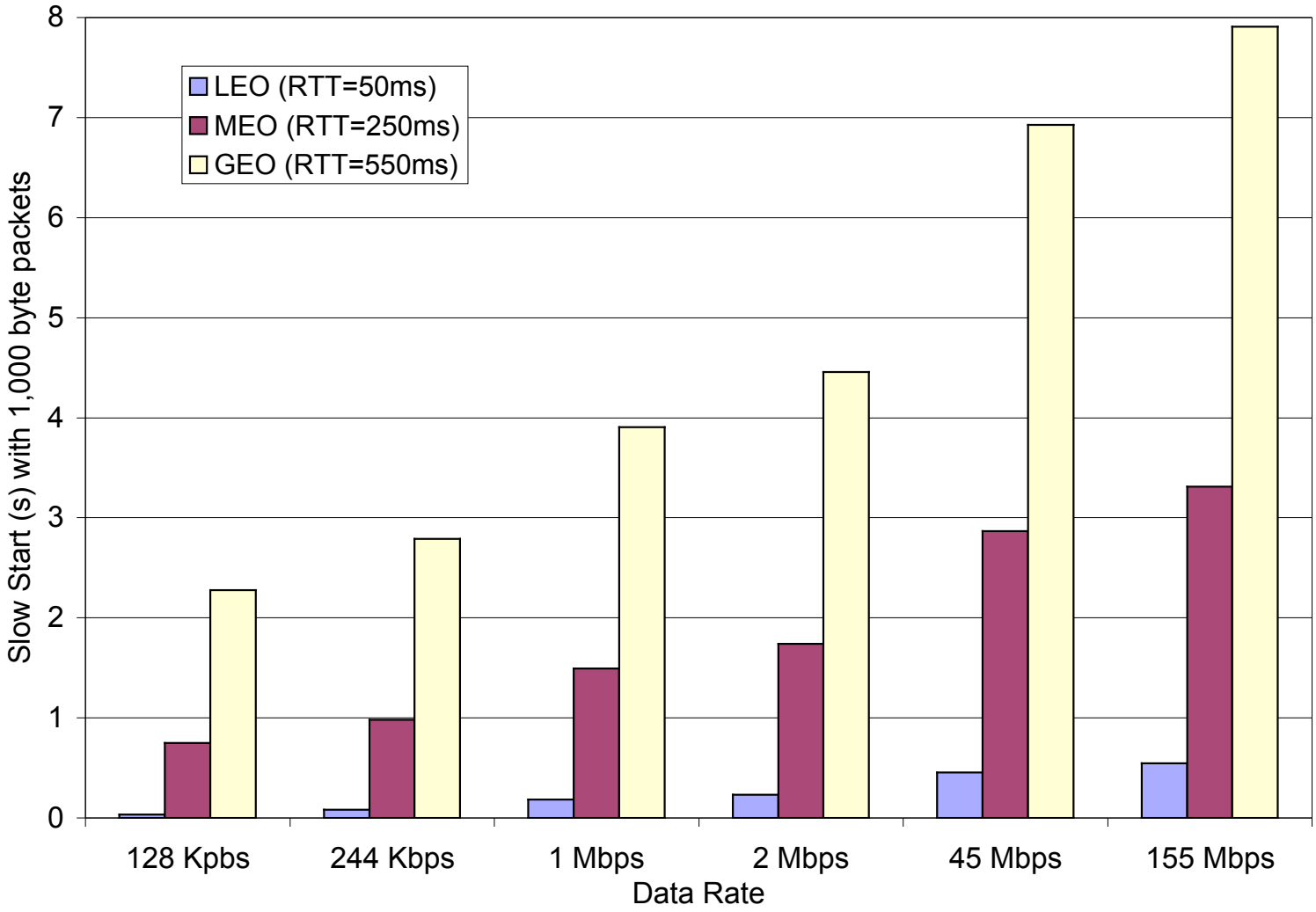
- Channel losses
  - Wired networks: Unavailability is negligible.
  - Wireless networks: Unavailability is higher (even 1% in satellite systems).
  - TCP assumes only forward link congested and when a packet is corrupted it was lost due to congestion (queue overflow) ➔ drops its window (Congestion Control).
  - Not all the losses are due to congestion ➔ not necessary to reduce windows dynamics
- Large delay
  - Round Trip Delay: GEO about 500ms, LEOs about 50-100ms.
  - Slows down windows growth process
  - Impacts BW*Delay
  - BW * RTT Product may be very large (especially if BW is large).
  - Impacts Optimum window size (Window * Pkt Size = BW * RTT)
  - Impacts RTO (usually multiple of RTT)
- Implications
  - Large optimum TCP window required (remember that TCP starts off with window size of 1).

Optimal Window Size (1,000-byte segments)



	128 Kpbs	244 Kbps	1 Mbps	2 Mbps	45 Mbps	155 Mbps
LEO (RTT=50ms)	1	2	6	13	281	969
MEO (RTT=250ms)	4	8	31	63	1,406	4,844
GEO (RTT=550ms)	9	17	69	138	3,094	10,656

Slow Start Ramping (1,000-byte segments)



Slow Start Ramping in $(Log_2(OptWinSize)+1)*RTT$

Further Satellite issues

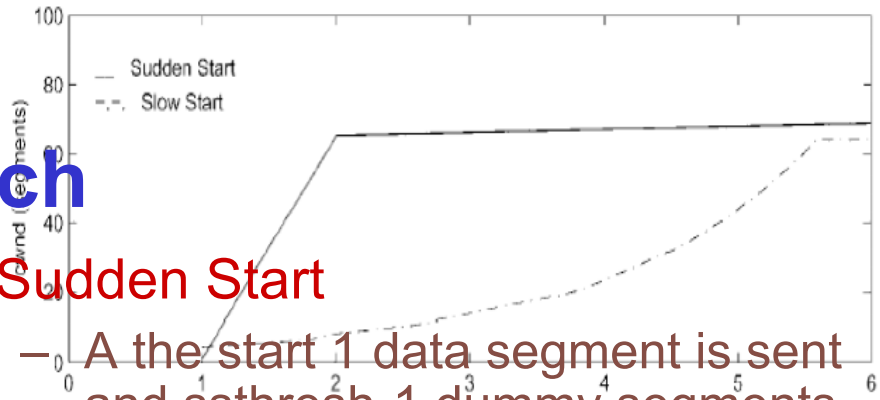
- Other characteristics affecting TCP
  - Handoff
    - Mobile user may have to transfer TCP connection from one satellite to another.
  - Asymmetric Use
    - One link may have less capacity than the other.
  - Delay may be variable
    - LEO satellites: Geometry is time-variant ➔ RTT may be variable.
    - GEO satellites: possibility of MAC layer control loops
    - TCP may not be able to update RTT quickly enough

Further TCP Problems over satellite

- Fairness when multiple connections share a bottleneck
  - TCP congestion control favors lower RTT, i.e., realistically not the satellite link
  - This feature is not specific to satellites
- Buffer management
  - Large windows required for efficient TCP
  - How to "optimally" allocate constrained memory at the endpoints for multiple TCP connections

TCP Peach

- Specifically designed for satellites
- Replaces Slow Start and Fast Recovery with *Sudden Start* and *Rapid Recovery*
- Both Sudden Start and Rapid Recovery use expandable "Dummy" packets to probe the characteristics of the network
  - Sent with a low priority, not a source of congestion
  - Contain no data, aren't recovered when lost
  - But when acknowledged, the TCP sender increases the window as a function of acknowledged data



- Sudden Start
  - A the start 1 data segment is sent and ssthresh-1 dummy segments.
  - For each dummy segment not discarded, 1 ACK is received and cwnd is increased of 1.
  - If N dummy segments are accepted and ACK, after 1 RTT the transmission rate will growth from 1/RTT to N/RTT, reaching the ssthresh
- Rapid Recovery
  - After loss and fast retransmit cwnd is halved but sending an adequate number of dummy segments the cwnd, in 1 RTT, may assume again the same value.

TCP Hybla

- In heterogeneous network, Hybla accelerates the increase of congestion window for the long RTT connections
- Window grows inversely proportional to RTT

$$\rho = \frac{RTT}{RTT_0}$$

$$W_{i+1} = \begin{cases} W_i + 2^{\rho} - 1 & \text{SS} \\ W_i + \dfrac{\rho^2}{W_i} & \text{CA} \end{cases}$$

- ρ=1 ➔ standard TCP equations

RTT_0= reference link RTT

Performance Enhancing Proxies

- Set of techniques employed to improve degraded TCP performance caused by characteristics of specific link environment (for example satellite) or subnetwork
- It implies the use of hardware devices and/or software tools
- Different types of PEP apply to different scenarios
- Recommendations are going to be developed by the Performance Implications of Link Characteristics WG (PILC)

Scope and generalities

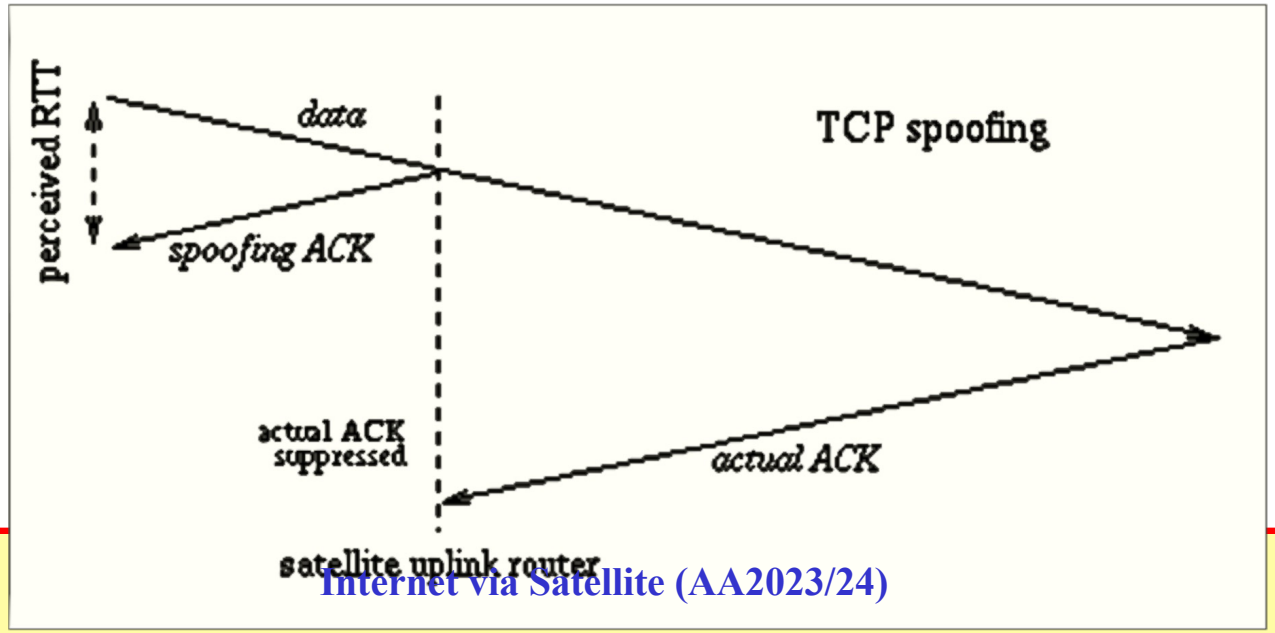
- Enhancing performance in terms of
  - Throughput
  - Usability of the link
  - Keeping TCP connection alive during periods of disconnection in wireless networks
- The implementation of such techniques may imply modification to the standard or to some TCP principle (for example end-to-end semantics)
- Utilization of such techniques only when strictly necessary
- The end user (or the network administrator) must be aware of the use of PEP especially if it interferes with IP layer security
- PEP applicable at any layer (in the following mainly transport and application layer PEP will be dealt with)
  - PEPs at lower level do not require network protocol modifications
  - Some PEP techniques operates across several layers requiring modification at each layer they act

Transport layer PEP (TCP PEP)

PROBLEM	SOLUTION
If ACKs may bunch together causing undesirable data segment burst	ACK spacing modification
Large bandwidth*delay product	Generation of local ACKs
SPOOFING	

TCP Spoofing

- Sometimes synonymous of TCP PEP
  - Intermediate gateway (agent) prematurely acks packets.
  - Intercepting and terminating a TCP connection in the middle as if the interceptor is the intended destination.
  - When it receives an ack, it suppresses it.
  - Transparent to both the sender and the receiver.
  - The spoofer takes the personality of both parties
  - Slow start can progress more rapidly.

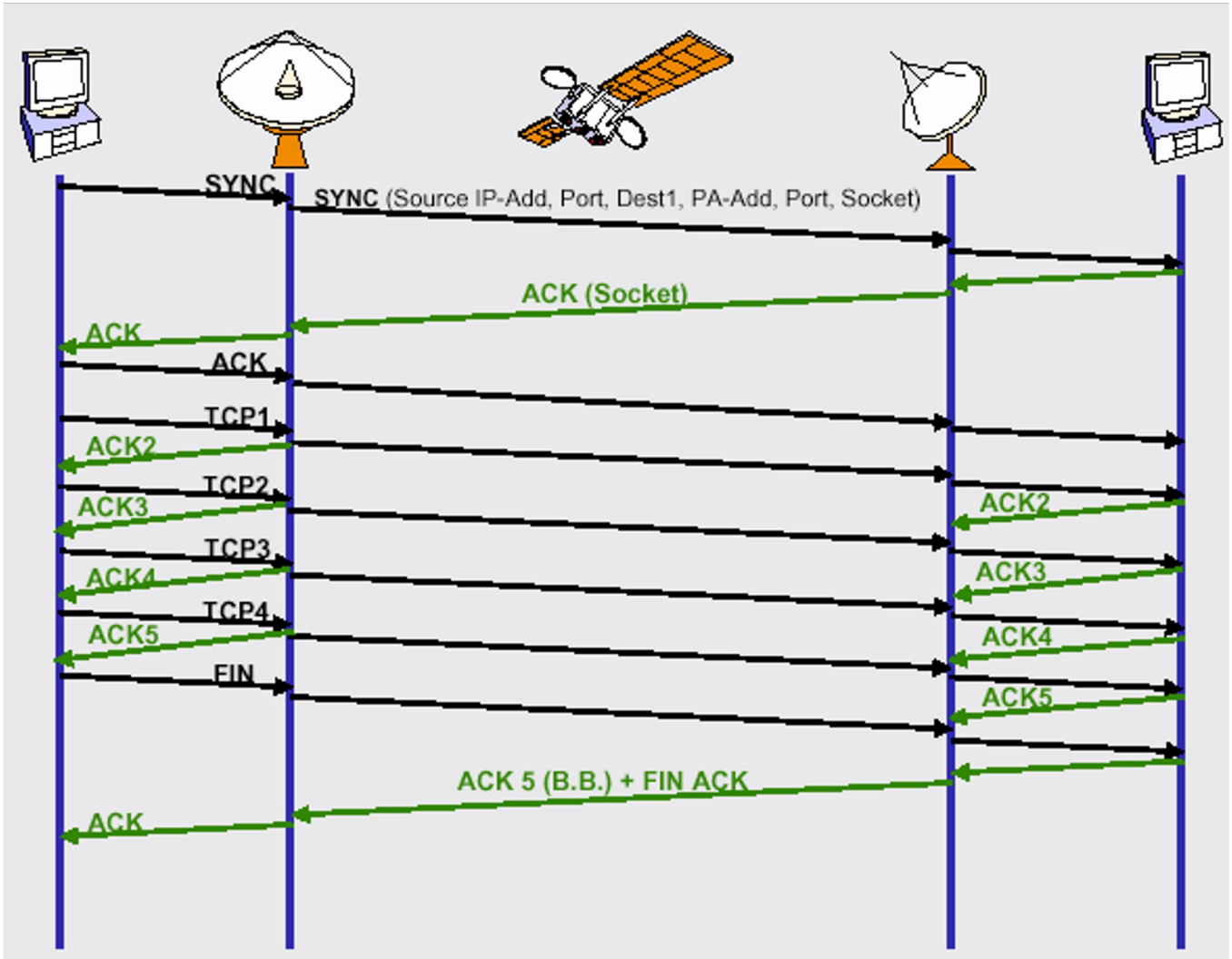


TCP spoofing (2)

Breaks end to end semantics

The end to end connection is kept.

Connection set up and final ack remain end-to-end



Application layer PEP

- Aims at improving performance in any environment, no specific to particular link characteristics
- Such proxies include:
  - Web caches
  - Relay Mail Transfer Agent (MTA)
- Improves application protocol as well as transport layer performance
- Unnecessary overheads (due to encoding) can be reduced

Distribution

INTEGRATED	DISTRIBUTED
Single PEP component within a single node	Two or more PEP components in multiple nodes

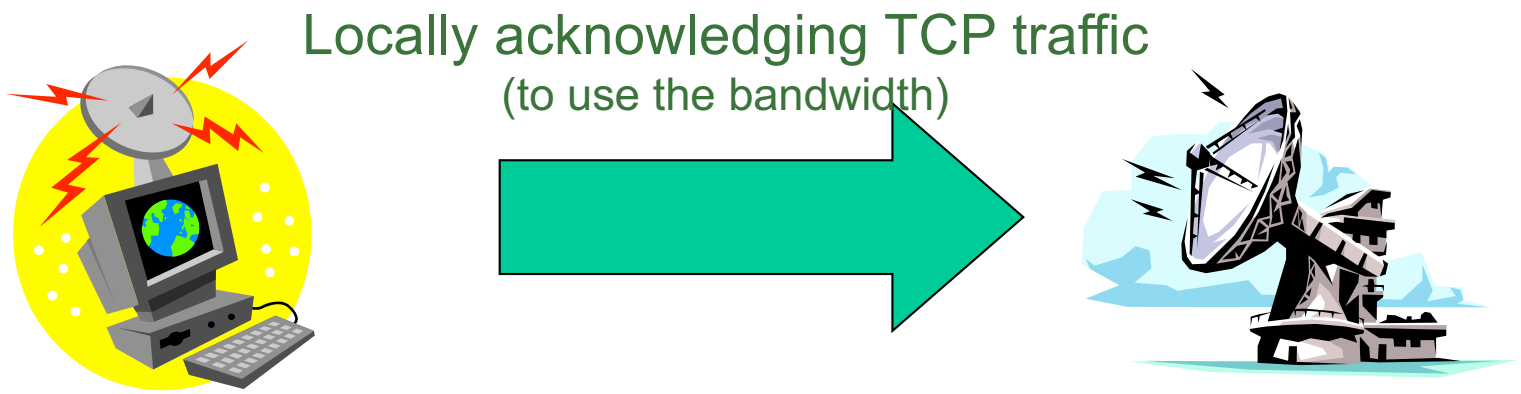
Symmetry

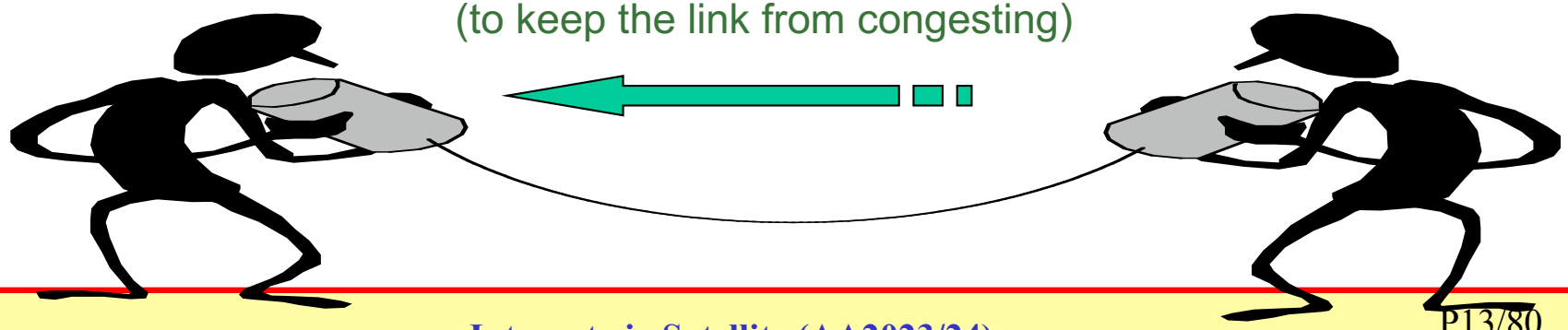
SYMMETRIC	ASYMMETRIC
Identical behavior in both directions (actions taken independent on which interface a packet is received)	Different behavior in each direction (the characteristics of the links on each side of the PEP differ or with asymmetric protocol traffic) Intersection between wired and wireless networks or request replay for HTTP traffic
BOTH with regard to different mechanisms employed	

Direction defined: in terms of the link (central to remote) or in terms of protocol traffic (TCP data flow or TCP ACK flow)

Distribution and Symmetry

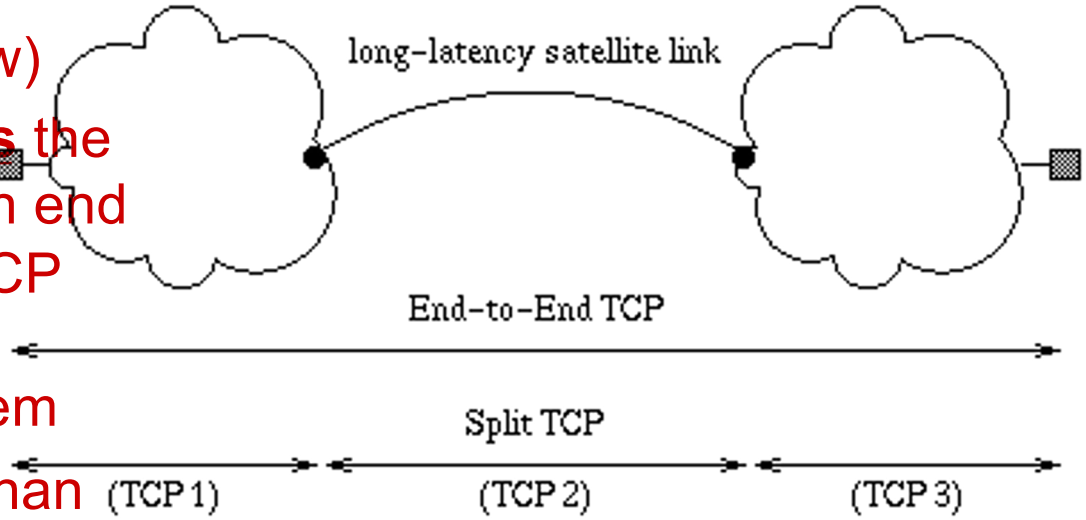
- CONCEPTUALLY INDEPENDENT (distributed PEP may operate either symmetrically or asymmetrically)
- Example: link with asymmetric amount of bandwidth





Split TCP

- Typical Connection (shown below)
- The proxy TCP agent **terminates** the TCP connection received from an end system and establishes a new TCP (or other transport protocol) connection to the other end system
- Split TCP connection into more than one



Split in distributed implementation

- In distributed architecture usually a third connection used between two PEPs
  - TCP connection optimized for the link or
  - Another protocol on top of UDP
  - Separate connection for each TCP connection or
  - Multiplex data from multiple connections across a single connection between the PEPs.

Split in integrated implementation

Addresses mismatch in TCP capabilities between two end systems

Window scaling (to extend the max amount of TCP data "in flight") useful in case of high BWD product

Window Scaling Capability

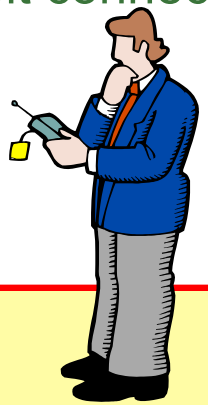
High BWD product

PEP

NO Window Scaling Capability

IP end-to-end connectivity not excluded

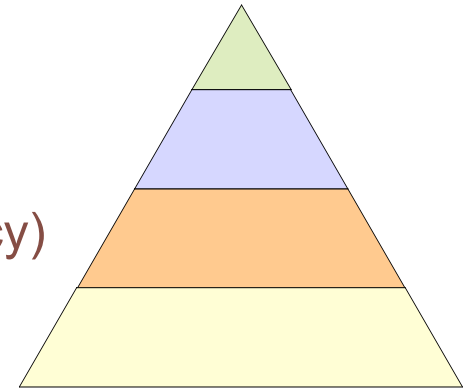
IF spit connection managed → Per application / Per connection → **AND** under control of the end user



The end user can decide a TCP connection → Split / End-to-end

Transparency

- Transparent
  - No modification to the end systems, transport end points or applications
- Not transparent
  - Modifications to both ends
  - Modifications to only one end
- Four levels
  - End user
  - Applications (application layer transparency)
  - Transport end points (transport layer transparency)
  - End system (network layer transparency)



USER AWARE, TCP/IP AND APPLICATIONS NOT

Transparency and Semantics

Transparency ≠ Semantics

TCP ACK spacing maintains end-to-end semantics

Split connection may not

Both can be implemented transparently to the transport endpoints at both ends

PEP MECHANISMS

- TCP ACK handling
  - ACK spacing
  - Local ACK
  - Local retransmission
  - ACK filtering and reconstruction
- Tunneling
- Compression
- Handling periods of Link disconnection with TCP
- Priority-based Multiplexing
- Protocol Booster Mechanisms

A PEP IMPLEMENTATION CAN IMPLEMENT MORE THAN ONE

TCP ACK handling

- TCP ACK Spacing
  - If ACKs tend to bunch together, it is used to eliminate bursts of TCP data segments to be sent.
- Local TCP ACK (employed with split)
  - PEP may locally acknowledge packets (when high BWD). It speeds up slow start and congestion window to open up.
  - Also negative ACK are used to trigger faster error recovery
- Local TCP retransmissions
  - PEP may locally retransmit packets lost on the path between the PEP and the receiving side using ACKs along with appropriate timeouts. Local ACK requires retransmission (not the reverse, ex SNOOP).
- TCP ACK Filtering and Reconstruction
  - Asymmetric links, ACKs may get congested in the low speed link if the asymmetry ratio is high. ACKs are filtered on one side and reconstructed on the other side. Details in [RCF2760] and [BPK97]

Tunneling

- Messages may be encapsulated to be carried across a particular link or to be forced to traverse a particular path.
- At the other end tunnel wrappers are removed.

- Used by a distributed split connection to carry the connection between the distributed PEPs.
- Used to support forcing TCP connection with asymmetric routing to go through the end points of a distributed PEP implementation.

Compression

- Reduces the number of bytes to be sent
- Good for bandwidth limited links
- Benefits:
  - Link efficiency
  - Effective link utilization
  - Reduced latency
  - Interactive response time
  - Decreased overhead
  - Reduced packet loss rate
- TCP and IP header compression
- Payload compression improves also security
- Application specific compression (particularly efficient)

Handling Periods of Link Disconnection with TCP

- The TCP PEP
  - Monitors the traffic from the sender to the receiver retaining the last ACK
  - Shuts down the TCP sender's window by sending the last ACK with a window set to zero
  - Put TCP sender in persist mode
  - The TCP receiver aware of the current state capable to freeze all timers
- With splitting, the disconnection can be hidden even for long time
  - If TCP PEP cannot forward data it stops receiving
  - TCP flow control prevents the TCP sender from sending more than the advertised window allowed by PEP.
  - A modified TCP version can allow to retain the state and all unacknowledged data and to perform local retransmission at the reconnection. The disconnection may or may not be hidden from the application and user.

Priority-based Multiplexing

- Scenario: slow link
  - Urgent data transfer (interactive connection) are assigned higher priority than less urgent transfer
  - Shorter response time is achieved
- The higher priority traffic may utilize the full (small) bandwidth
  - The low priority transfer may be suspended
  - It cannot be delayed for a long time (TCP sending timer expires)
  - It can be controlled in conjunction with Split TCP PEP (different priorities to different connections/applications)
  - Split PEP in intermediate node can delay the data delivery of lower priority flow for an unlimited time rescheduling the order

Protocol Booster Mechanisms

- Some specific to UDP
  - Asymmetrical: Extra UDP error detection
    - 16 UDP checksum optional ➔ not computed
    - Link with errors ➔ useful ➔checksum added by a PEP
  - Symmetrical: Forward Erasure Correction (FZC)
    - The encoding PEP adds a parity packet over a block
    - Reception, parity removed, missing data regenerated
  - Jitter control mechanism (extra latency)
    - Sending PEP adds a timestamp to outgoing packets
    - Receiving PEP delays packets

Implications

- End-to-end argument (already approached, not all violate)
  - Security (transport layer PEPs compatible with higher layer security, but a few applications include support for the use of transport, or higher, layer security)
    - Security Implications (end to end IPsec not compatible with split)
    - Security Implication Mitigation (differentiating traffic, IPsec between PEPs)
  - Fate Sharing (related to survivability to failure)
  - End-to-end reliability (to implement at application level)
  - End-to-end Failure Diagnostic (PEPs may delay detection)
- Asymmetric Routing (the location of the PEP causes non optimal routing)
- Mobile Hosts (problems with handover)
- Scalability (greater processing requirements than a router)
- Other Implications (QoS, network infrastructure)

Security

- Network (IP) layer security (IPsec) [RFC2401], can be used by any application, transparently.
- IPsec supports two modes:
  - Transport (IP header clear, IP data encrypted)
  - Tunnel (entire datagram encrypted)
- If IPsec is employed end-to-end, PEPs on intermediate nodes cannot examine the transport or application headers of IP packets because encryption of IP packets (in either transport or tunnel mode) renders the TCP header and payload unintelligible to the PEPs.
- Without being able to examine the transport or application headers, a PEP may not function optimally or at all.
- PEP disables end-to-end use of IPsec.

PEP implementation

- **Non-transparent** to the users and the users trust the PEP in the middle, IPsec can be used separately between each end system and PEP. In most cases undesirable or unacceptable alternative as the end systems cannot trust PEPs in general.
- *Not as secure as end-to-end security.* For example, the traffic is exposed in the PEP when it is decrypted to be processed and it can lead to potentially misleading security level assumptions by the end systems. If the two end systems negotiate different levels of security with the PEP, the end system which negotiated the stronger level of security may not be aware that a lower level of security is being provided for part of the connection. The PEP could be implemented to prevent this from happening by being smart enough to force the same level of security to each end system but this increases the complexity of the PEP implementation (and still is not as secure as end-to-end security).
- **Transparent PEP**, it is difficult for the end systems to trust the PEP because they may not be aware of its existence. Even if the user is aware of the PEP, setting up acceptable security associations with the PEP while maintaining the PEP's transparent nature is problematic (if not impossible).

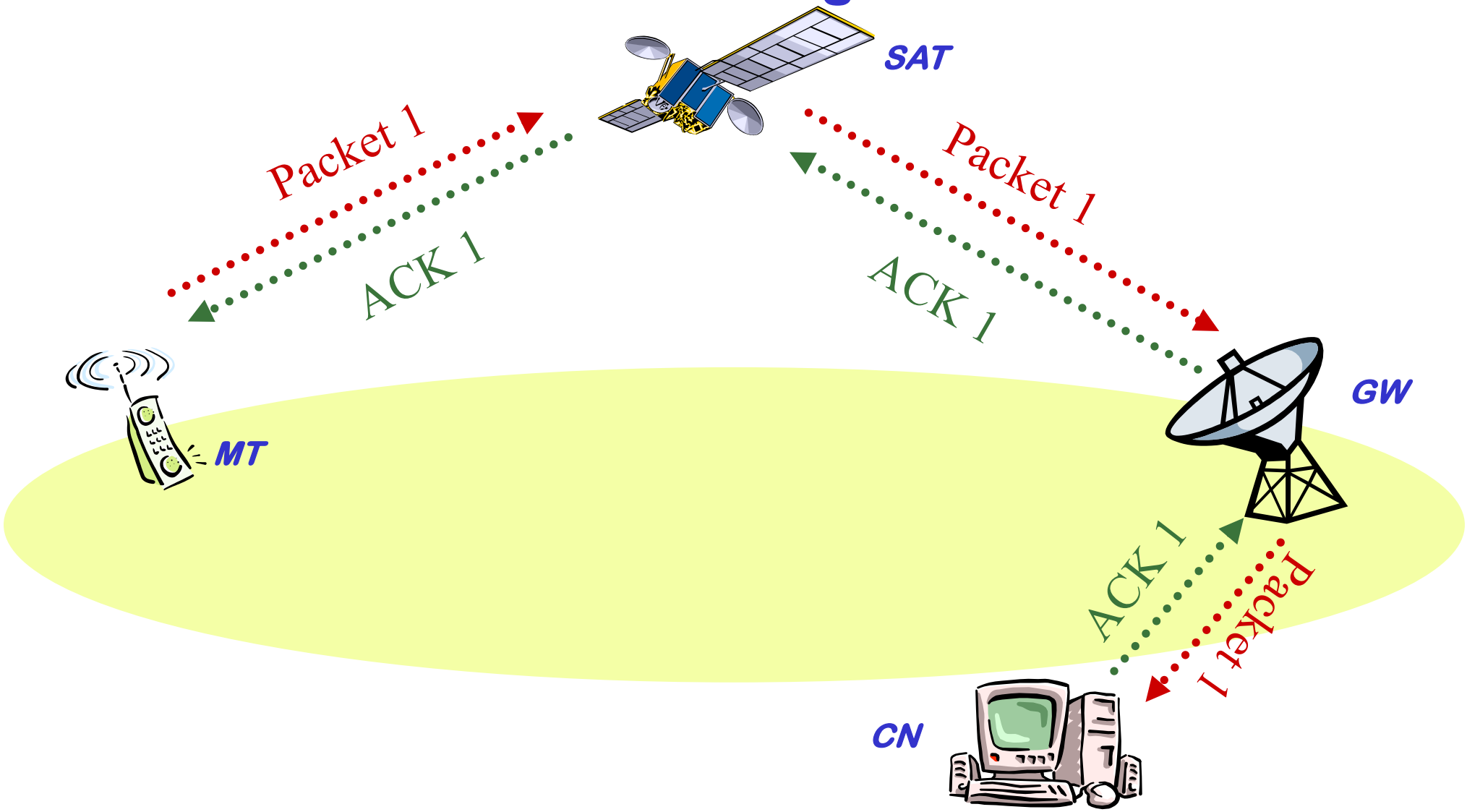
PEP support to IPsec

- Even when a PEP implementation does not break the end-to-end semantics of a connection, it may not be able to function in the presence of IPsec.
  - difficult to do ACK spacing if the PEP cannot reliably determine which IP packets contain ACKs of interest.
- No info of any PEP implementations, transparent or non-transparent, which provide support for end-to-end IPsec, except in a case where the PEPs are implemented on the end hosts.

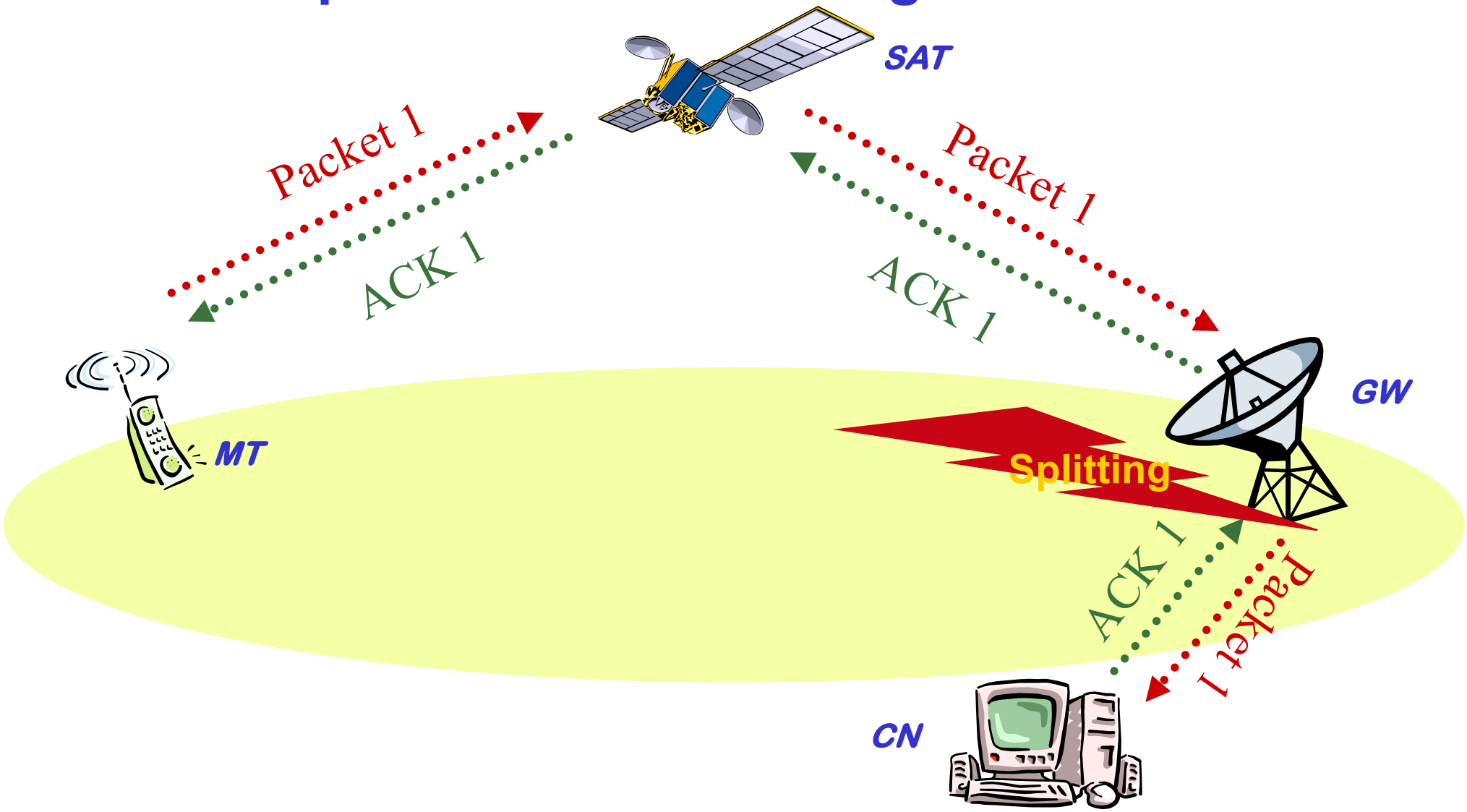
Security Implication Mitigations

- Use of IPsec for some traffic and not for other traffic,
  - PEP processing can be applied to the traffic sent without IPsec.
- IPsec implemented between the two PEPs of a distributed PEP implementation.
  - At least the traffic between the two PEPs is protected.
- Multilayer Ipsec
  - Add complexity to the already complex IPsec

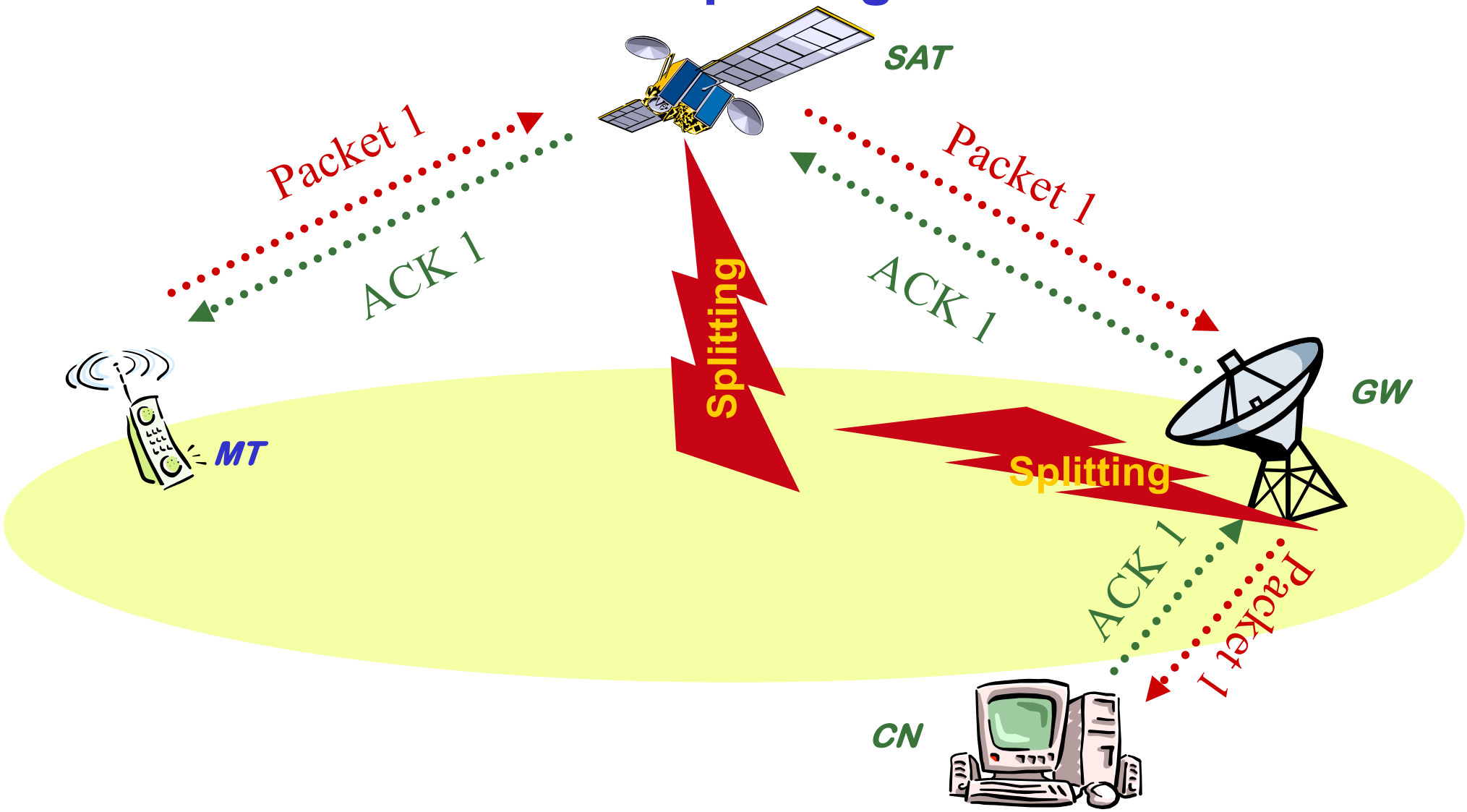
TCP connection through satellite link



TCP split connection through satellite link



TCP connection splitting satellite link



Advantages

- Standard TCP
  - The on-board transport improves performance using any TCP implementations.
  - Non-standard transports use an optimistic approach to flow-control, but the satellite network may also benefit from the TCP approach to congestion.
  - A satellite system incorporating multiple-hops (and orbits) with packet switching and routing can benefit from standard TCP mechanisms.
- Flexible Terminal Design
  - Using a standard TCP no impact on terminal design. By splitting the connection in space, disparate terminal types can still communicate although they use different transports. A disadvantaged terminal may use the same TCP stack for all connections, roaming between wired, wireless, and satellite segments. This terminal might benefit from an advantaged gateway optimized earth-to-space transport.
- Shadowing
  - If just one of the two links is shadowed (of course in case of mobility) the other one can go on working
  - Independent error-control mechanisms between earth-to-space connections enhances the robustness of the transport.
  - Independent on-board transports recovers from shadowed losses more quickly.
- Multicast
  - Using TCP on the uplink or the downlink allows for multicasting on the opposing ground link or in the space segment. TCP may operate on the uplink while a multicast downlink takes advantage of the satellite's broadcast capabilities to serve multiple terminals and additional satellites.

Disadvantages

- The end-to-end semantics of TCP is violated
  - Packets corrupted in the uplink gateway cache are unrecoverable
  - Solution:
    - satellite acks packets received on the uplink only after the first successful data transmission on the downlink without waiting for ack from downlink
    - Satellite uses some method to ensure that ongoing packets are not corrupted
  - Only packets both lost on the downlink and corrupted in memory are actually unrecoverable
- Regenerative payload needed
- Added complexity on board (but not much)
- Improves significantly with Reno but much less with new paradigms

TCP accelerators (XTP)

- Xpress Transport Protocol (XTP) designed to work well over a typical satellite link characterized by long latency, high loss and asymmetric bandwidth conditions.
- Orthogonal protocol functions allow XTP to separate the communication paradigm from the error control policy employed and also to provide a rate control independent from the flow control.
  - The data flow is regulated by an end-to-end windowing flow control mechanism based on 64-bit sequence numbers and a 64-bit sliding window.
  - XTP rate control mechanism allows both end-systems and intermediate nodes to specify the maximum bandwidth and the burst size they will accept on a particular connection.
- Finally, to detect and recover missing or corrupted data, XTP uses positive and, when appropriate, negative acknowledgements. Then, the error control mechanisms allow both go-back-N and selective retransmissions.

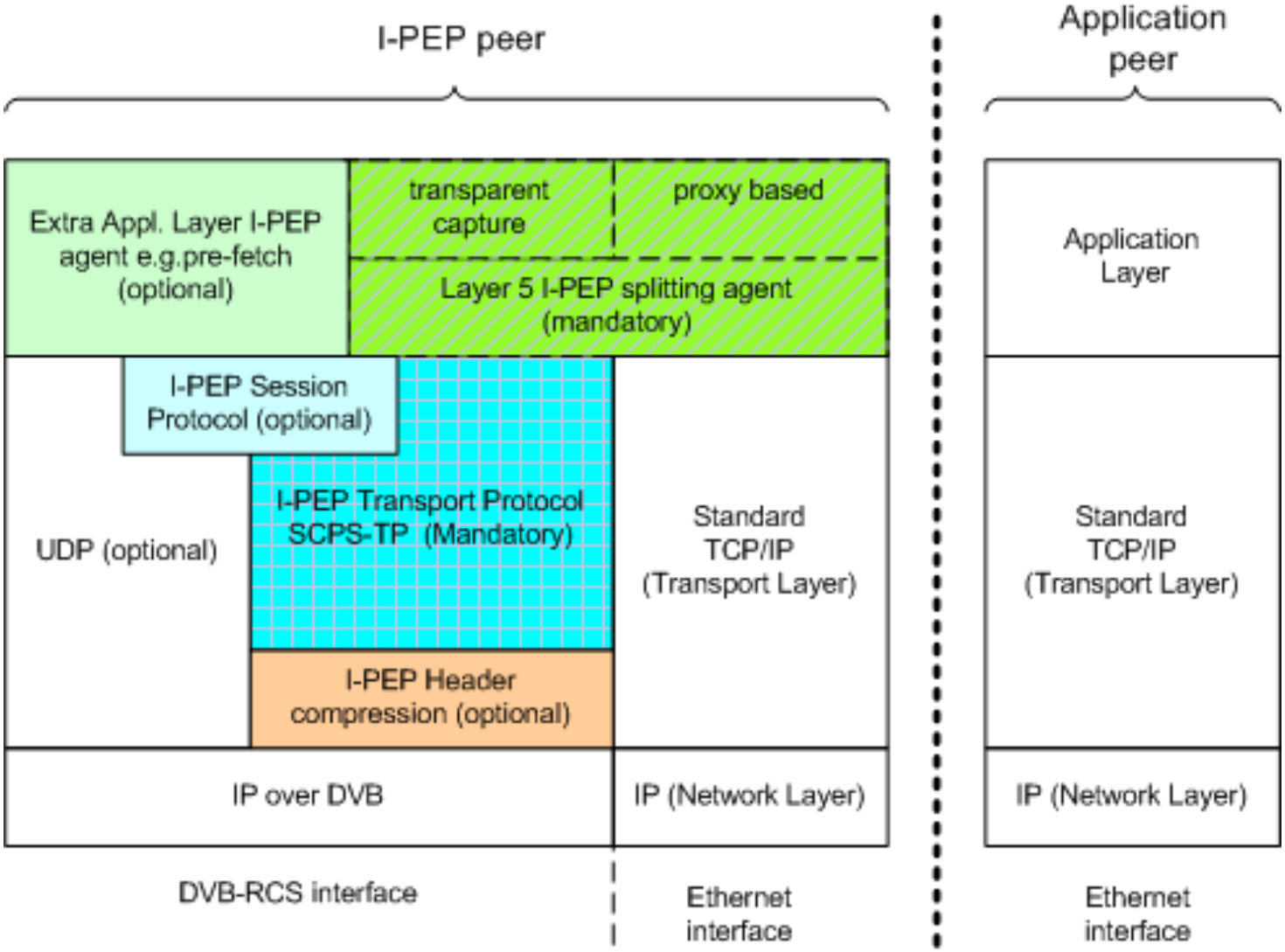
Other Transports

- SCPS (Space Communications Protocol Standards)
  - Highly optimized for space including header compression
  - Largely "overcome by events"
- STP (Satellite Transport Protocol)
  - Connectionless SSCOP (Service Specific Connection Oriented Protocol) for satellites
- Reliable multicast
  - PGM (Pragmatic General Multicast), SkyX and many more
- Unreliable, e.g. UDP/RTP/VoIP
  - Requires some form of congestion control or QoS

Interoperable-PEP (I-PEP)

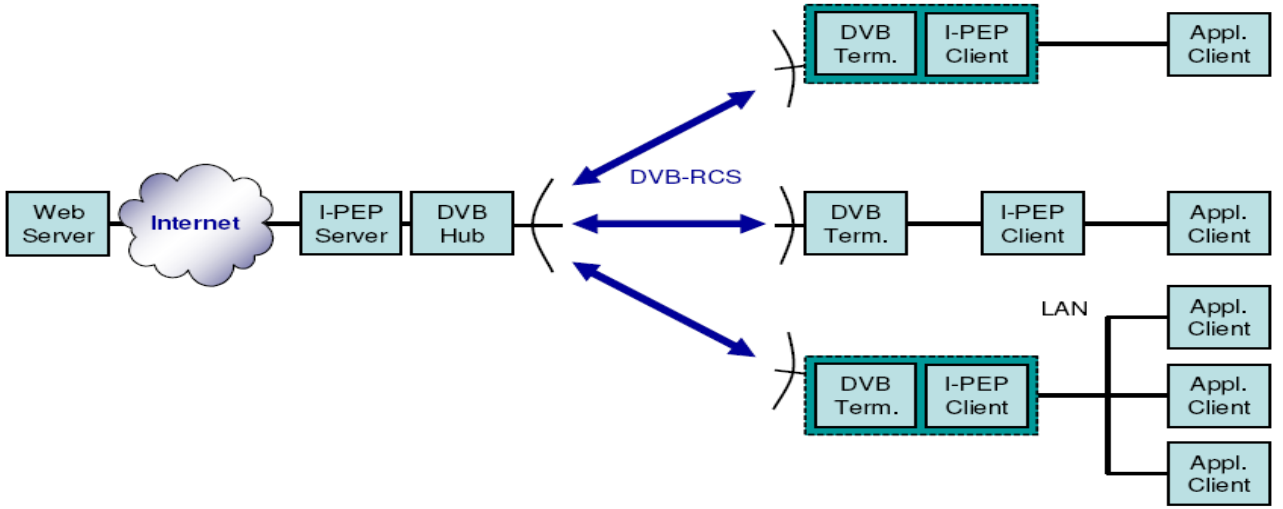
- I-PEP design and development has been driven by SatLab (working group led by ESA)
  - SatLab: Industry association for the promotion of the adoption of the DVB-RCS standard
- I-PEP scope
  - To provide specification to define a common PEP architectural elements of a DVB-RCS system
  - To guarantee interoperability among different possible vendor-specific implementations
- Specification for I-PEP
  - Adoption of a splitting architecture
  - Specification of operational and usage scenarios
  - Specification of a transport protocol for the satellite segment
    - SCPS-TP with specific option

I-PEP protocol stack

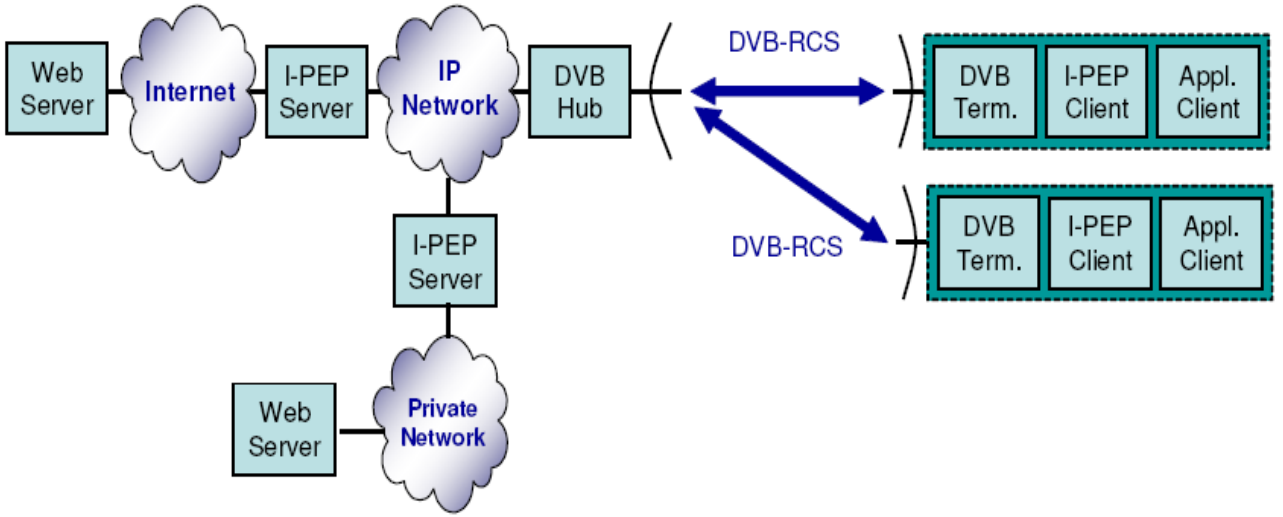


I-PEP Example of usage scenarios

- WAN-Side I-PEP co-located with Hub



- WAN-Side I-PEP at multiple ISP



TCP Accelerators (SCPS)

- "Space Communication Protocol Standards" (SCPS): set of layered protocols optimized for the space segments. SCPS can be used as an integrated protocol stack, or the individual protocols can be used in combination with Internet protocols. More specifically, the SCPS stack is composed of:
  - A file handling protocol (SCPS-FP);
  - A transport protocol (SCPS-TP);
  - A security protocol (SCPS-SP);
  - A scalable networking protocol (SCPS-NP).
- SCPS-TP is based on the Internet Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), the Internet Host Requirements Document and the "TCP Extensions for High Performance". It provides reliable end-to-end services and performs well in a space environment. SCPS also introduces extensions to the standard mechanisms in order to improve the performance of space communications. For example, SCPS-TP has developed three capabilities to address the possibility of data loss due to transmission errors:
  1) "Explicit Corruption Response". When an isolated data loss occurs, SCPS-TP keeps the transmission rate (controlled by the congestion window) and the retransmission timeout value unchanged.
  2) "Selective Negative Acknowledgement (SNACK)". The SNACK capability has been developed to identify specific data that requires retransmission and to request immediate retransmission of these data.
  3) "Header Compression". SCPS-TP uses a header compression capability to reduce the size of transmitted packets. This scheme is loss-tolerant, meaning that the loss of one packet does not make subsequent packets unintelligible.

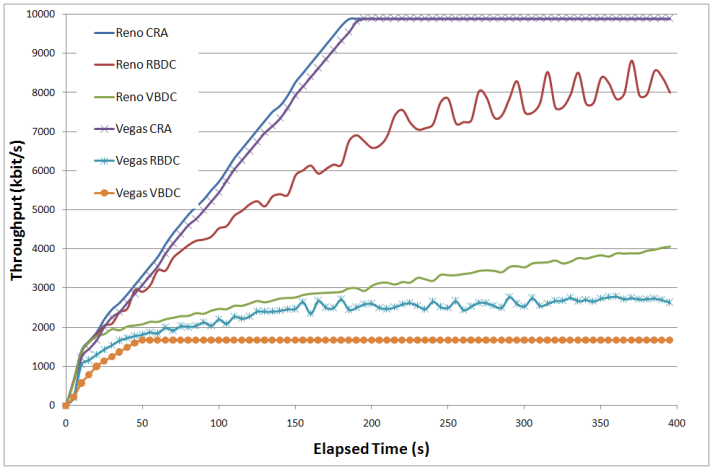
I-PEP and SCPS-TP

- Essentially, TCP with
  - Further options
    - e.g., to facilitate forwarding at I-PEP
  - Extensions (IETF-derived)
    - SNACK, SACK, T/TCP, RTT meas, PAWS, HC, ECN, etc.
  - Modifications of main interest
    - Possibility of alternative congestion control methods
      - TCP Vegas
      - Rate Control
      - Other, user-defined

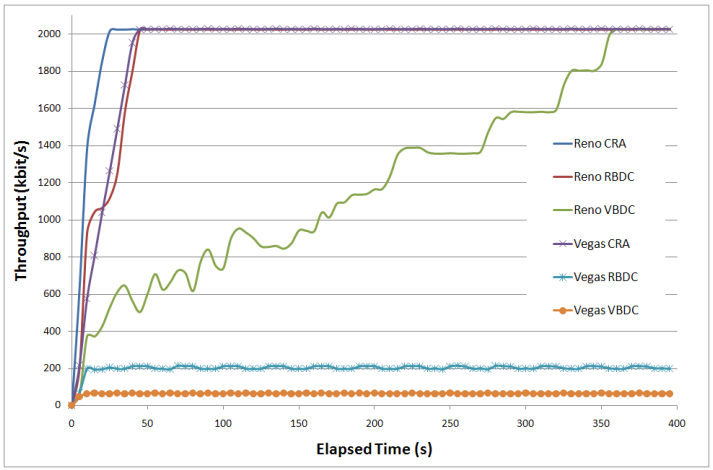
I-PEP performance over DVB-RCS

- TCP Throughput (long transfers over time)
  - (a) Forward Link (b) Return Link

(a)



(b)



Server: http://www.repubblica.it (several external links/banners)
TCP version Client side: BIC
Browser: Firefox 1.5 (HTTP 1.1) with Fasterfox extension

	Direct connection		*Hop via simulated satellite link* (504 8 ms delay)	
	First rendering	Total	First rendering	Total
Min	n.a.	3.39 s	21 s	28 s
Max	n.a.	7.4 s	30 s	64 s
Avg	n.a.	4.19 s	28 s	45 s

Server: http://www.repubblica.it
TCP version Client side: BIC
Browser: Firefox 1.5 (HTTP 1.1) with Fasterfox extension

	Direct connection		*Hop via simulated satellite link* (504 8 ms delay)	
	First rendering	Total	First rendering	Total
Min	1 s	1.3 s	n.a.	6.3 s
Max	5 s	7.1 s	6 s	13 s
Avg	n.a.	2.74 s	n.a.	9.21 s

Development of a protocol compliant with I-PEP

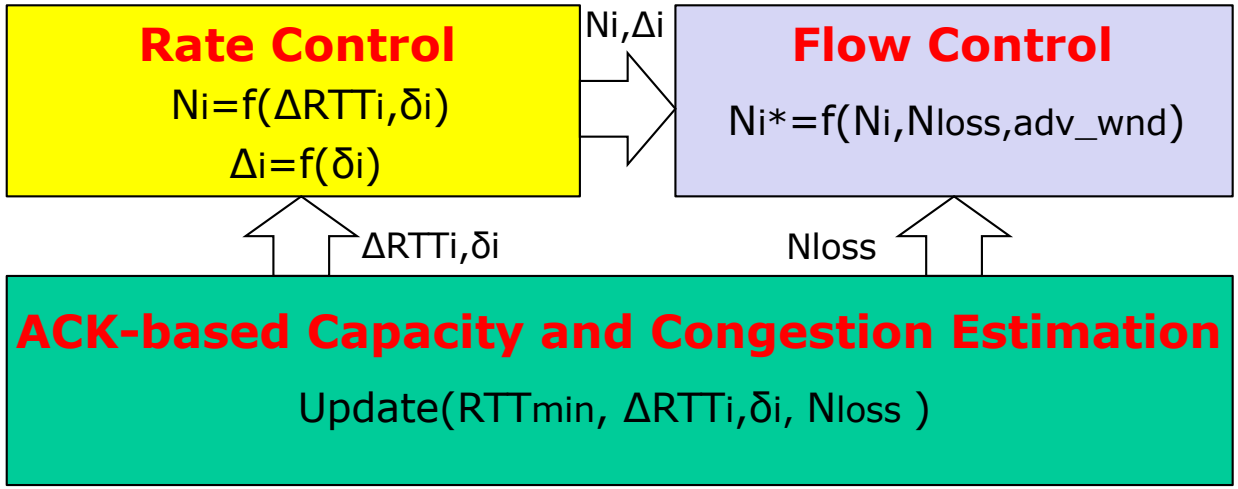
- Starting concept: TCP developed for quite large file transfers over congested links with relatively low BW-delay product.
- Design assumptions:
  - Run over a controlled environment: point-to-point link, whose characteristics are well-known (i.e. buffer sizes, MAC algorithms, overall bandwidth)
  - Bandwidth management/assignment is in charge of DAMA control loop
  - Target traffic profile is web browsing: relatively **short transfers**, interspaced with silence.

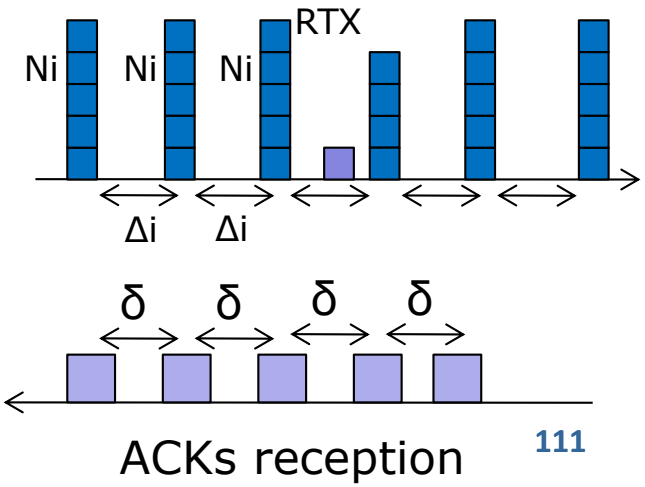
TCP Noordwijk: the basic idea

- **Burst-based transmission**: data sent in suitably sized, relatively large bursts rather than as a smooth flow of spaced packets.
  - Assumption of a controlled environment allows bursts management (i.e. tailored buffers)
  - Burst size (N) and time between two burst sending (Δ) regulate the transmission
- Two transmission phases
  - Initial **"blind" phase** (replacing Slow Start): N_0 and Δ_0 values are used. Initial values:
    - $N_0 \geq$ common web object size
    - Δ_0 such to match a target initial rate (N_0/Δ_0)
  - **A "rate control" phase** (replacing standard congestion control) where N_i and Δ_i are changed depending on both link characteristics and loading
    - ACK timing is used to estimate the optimal "rate".

TCP Noordwijk: general protocol architecture

- 3 functional components:
  - **ACK-based estimation**: statistics from ACK flow monitoring
  - **Flow control**: determines which and when packets must be transmitted
  - **Rate control**: adjusts N and Δ burst parameters





111

Rate Control (1/2)

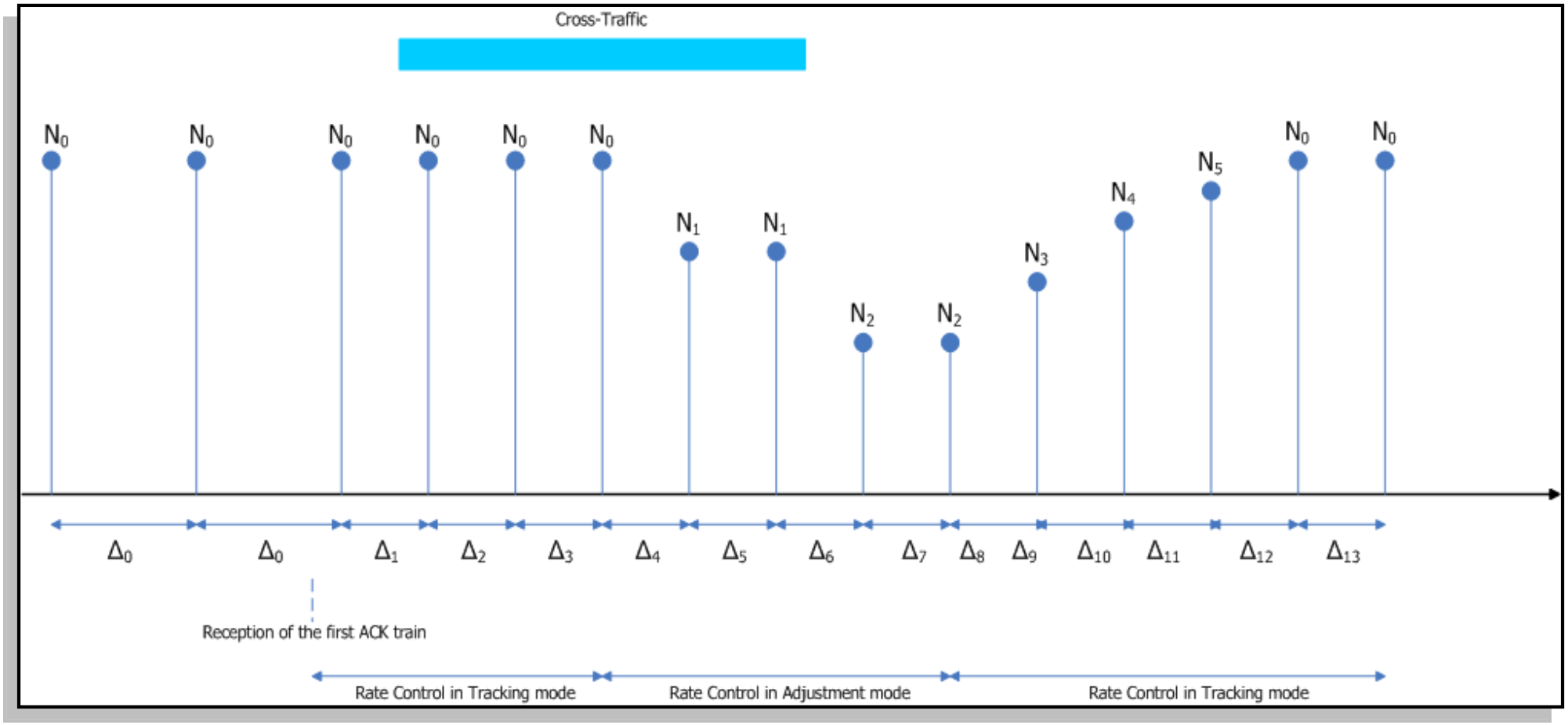
- **Rate Control** adapts the burst-based transmission to both the link capacity and its dynamic variations. This is achieved in two ways:
  - Increasing/decreasing the burst size (N), keeping TX interval among bursts (Δ) constant
  - Increasing/Decreasing TX interval (Δ), keeping burst size (N) constant.
- Specifically, N and Δ are updated at each ACK train reception based on inputs from the ACK-based estimation component: δ_i and ΔRTT_i
- Two algorithms are supported:
  - **Rate Tracking** (in absence of congestion $\rightarrow \Delta RTT_i < \beta$)
  - **Rate Adjustment** (in presence of congestion $\rightarrow \Delta RTT_i > \beta$)

β threshold which represents the maximum RTT component due to the "access delay."

Rate Control (2/2)

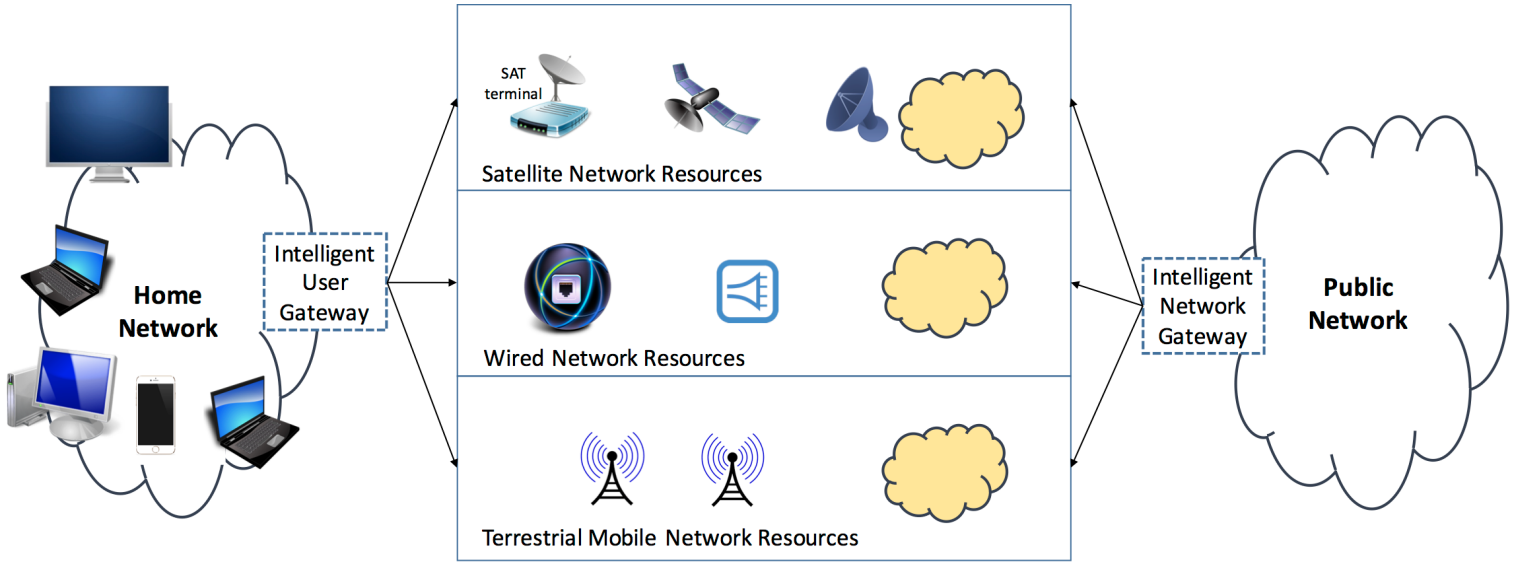
- **Rate Tracking** aims at adapting transmission rate to the maximum allowed rate through the following steps:
  - Burst size is gradually increased (logarithmic growth) up to the initial burst value (N_0). N_0 is considered as a reference value
  - Δ_i is set to the optimal value for N_0-bursts: $\Delta_i=\delta_i \bullet N_0$ (maximum rate is achieved when $N=N_0$)
  - Updated δ_i values are continuously provided by the ACK-based estimation component.
- **Rate Adjustment** reduces burst size proportionally to the experienced congestion level, while Δ_i is kept unchanged
- Since TCP Noordwijk is based on ACK train receptions, burst size can not be decreased under a given value N_{min} (e.g. 3 packets)
- In case such a threshold is reached, Δ_i is increased ($\Delta_{i+1}=\lambda \bullet \Delta_i$)

Conceptual iteration of TCP Noordwijk sender behaviour



A new target framework

- Vision of next communication networks:
  - Seamless exploitation of multiple access technologies through an **Intelligent User Gateway**;
  - Network-driven switching aimed to both maximize user experience and minimize access costs



An evolving scenario

New applications and technologies

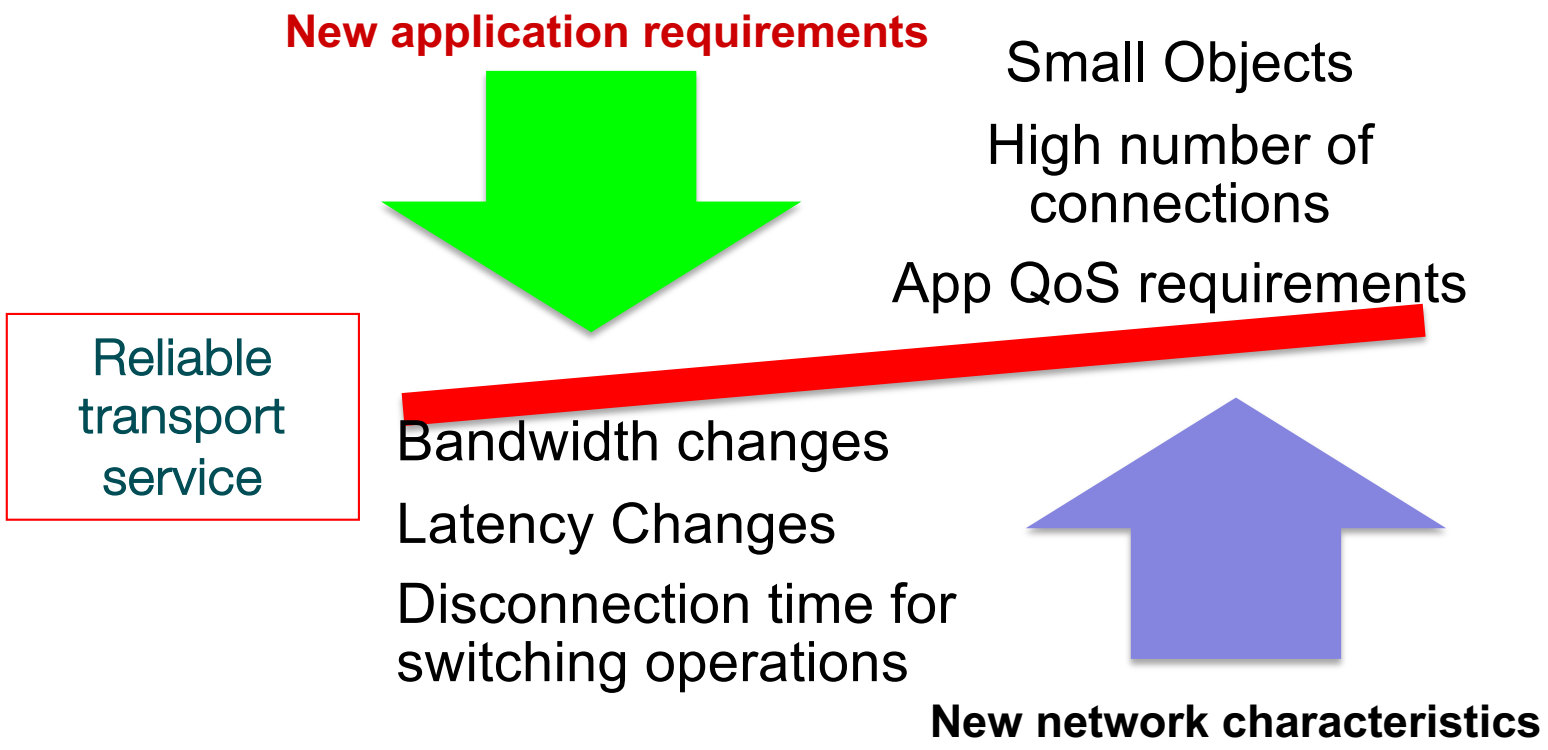
Several
long flows!

Small, frequent objects to deliver!!!

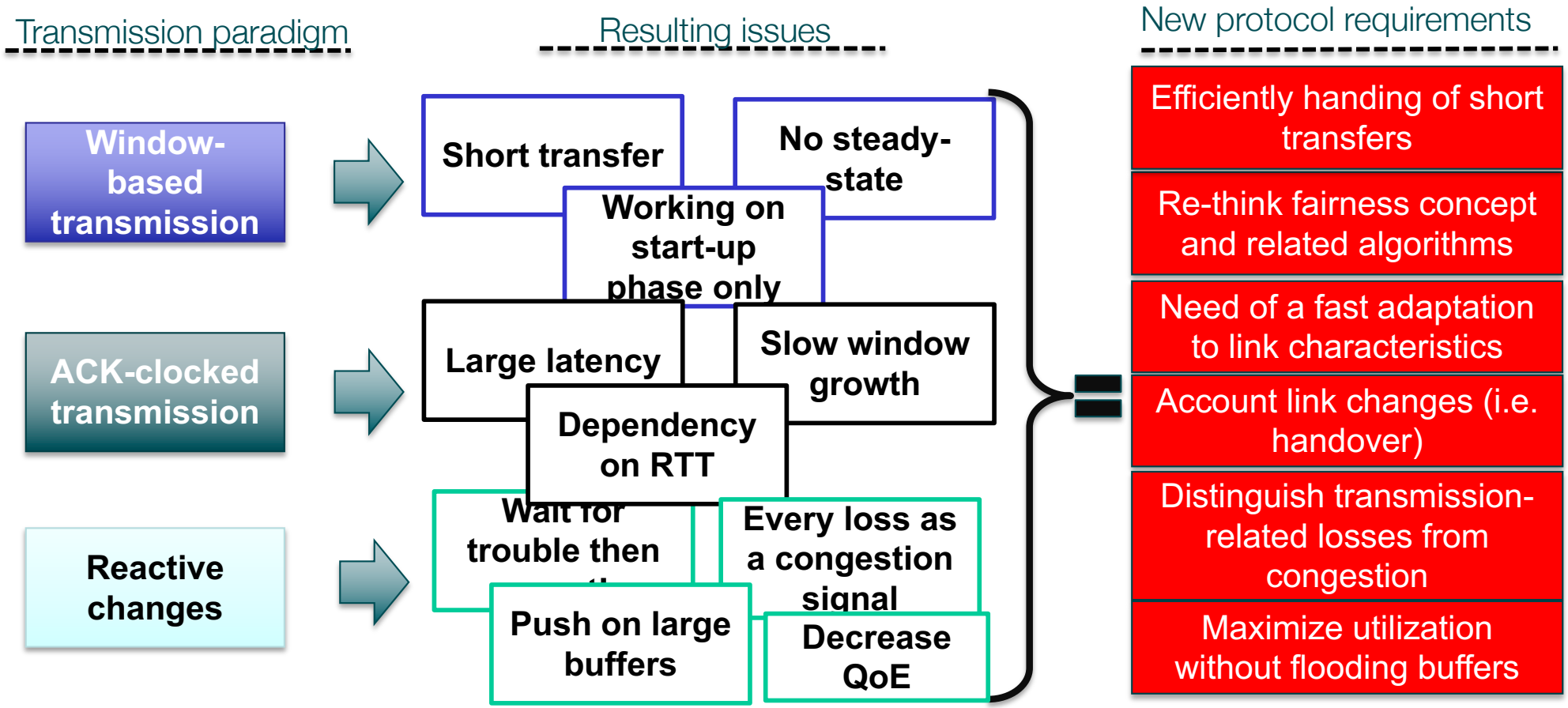
High-capacity alternative or complementary pipes

Possible dynamic switching among available pipes!!!

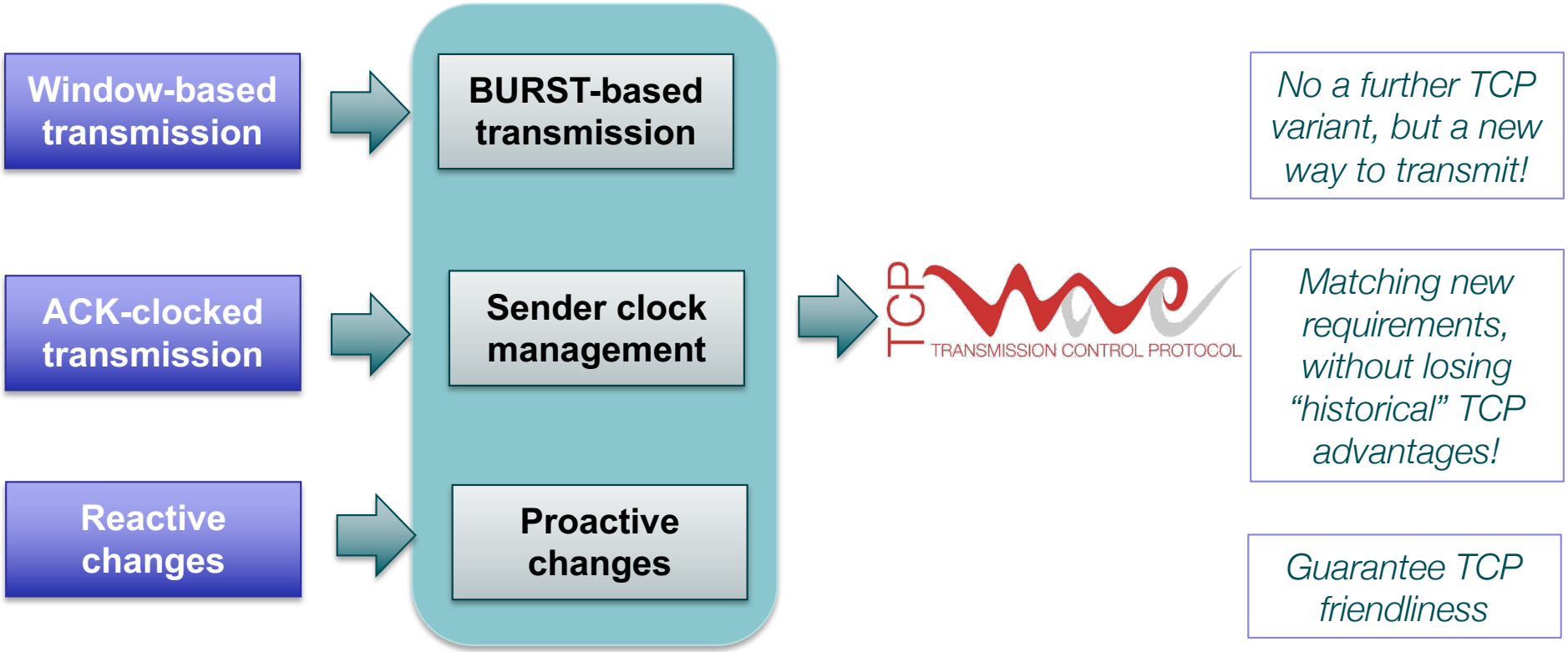
A new framework to design reliable transport protocols



Review of TCP transmission paradigm



Paradigm change

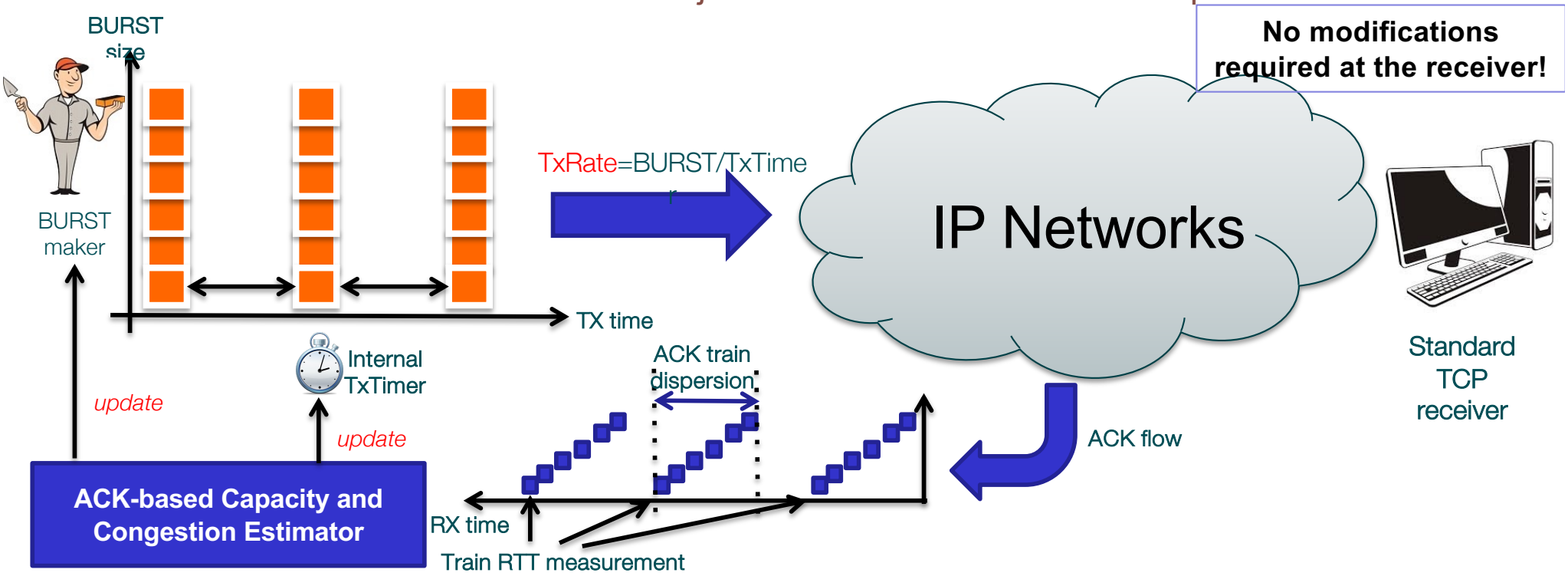


TCP Wave at a glance

- This new TCP-based protocol mainly leaves the window-based transmission paradigm to leverage on a **burst-based transmission**.
- ***BURST*** size replaces the Congestion Window (cwnd) and ***TxTimer*** schedules burst transmission, overcoming the ACK-clocked timing used by traditional TCP.
- **TCP Wave** target is to become a valid alternative to all standard TCP versions in every broadband communication environment (including terrestrial-only links), where traditional TCP control loop operates on a RTT basis.
- Burst transmission is completely uncorrelated by the ACK reception process. At the same time, transmission in bursts allows generation of **ACK trains**, which carry useful information to efficiently estimate available bandwidth:
  - RTT is still a good meter of the current congestion
  - ACK train dispersion tells about the overall system capacity.
- TCP-burst exploits such information to update both **BURST** size and ***TxTimer*** accordingly.

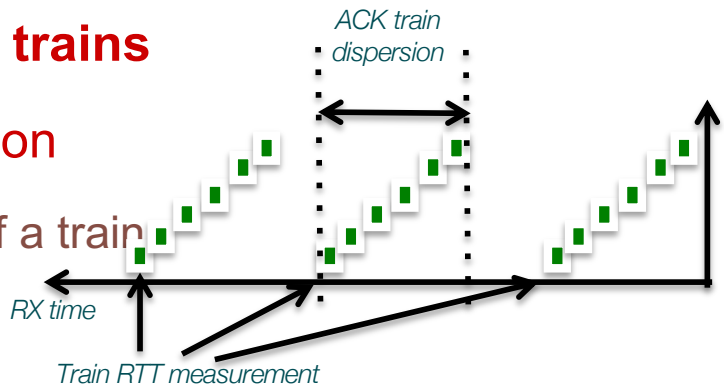
Burst transmission principles

- **BURST** replaces **TCP WINDOW** concept!
  - ***BURST*** = Number of packets to be transmitted at once
  - ***TxTimer*** = Timer to schedule transmission of next BURST
- Wave set up the **rate** taking into account cross-traffic (ΔRTT)
  - Main difference with Noordwijk which sends at the maximum possible rate



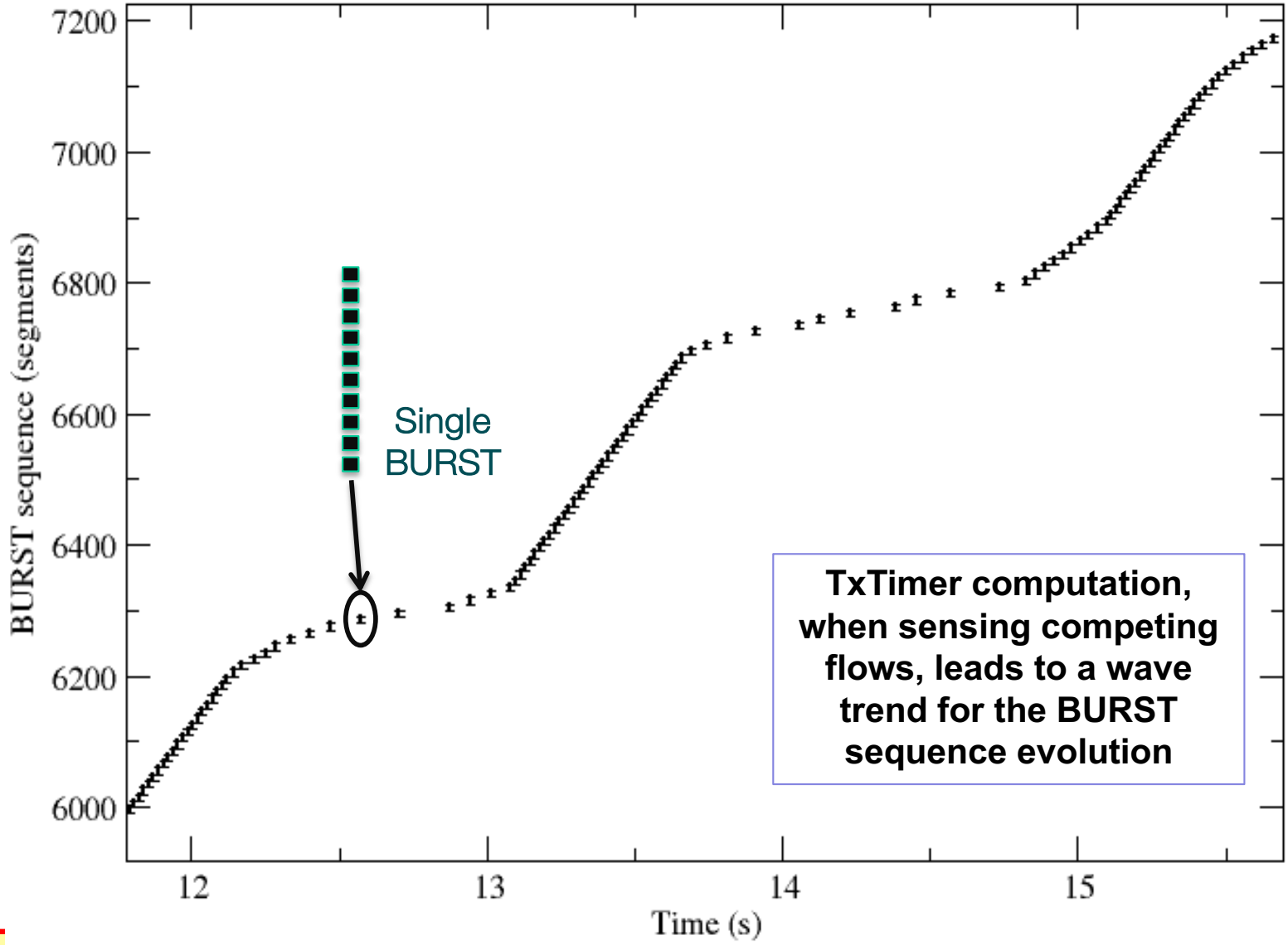
ACK train information

- Transmitting BURST allow the reception of **ACK trains**
- An ACK train carries an enriched set of information
  - *Pilot RTT* = RTT experienced by the first packet of a train
  - *ACK train dispersion* = ACK spreading over time



- **Pilot RTT** = RTT referred to the first packet of a train. Indication of the overall network congestion (when compared with minimum measured RTT)
- **ACK Train Dispersion** = Time between the first and the last ACK. Indication of the *overall system capacity* to process a BURST. It includes possible buffering and receiver processing as well as possible bw limitation on the return link.
  - *The best transmission scenario is to receive a continuous ACK flow (without spacing between consecutive ACK trains)-> this means to match maximum allowed rate!*

The Wave effect

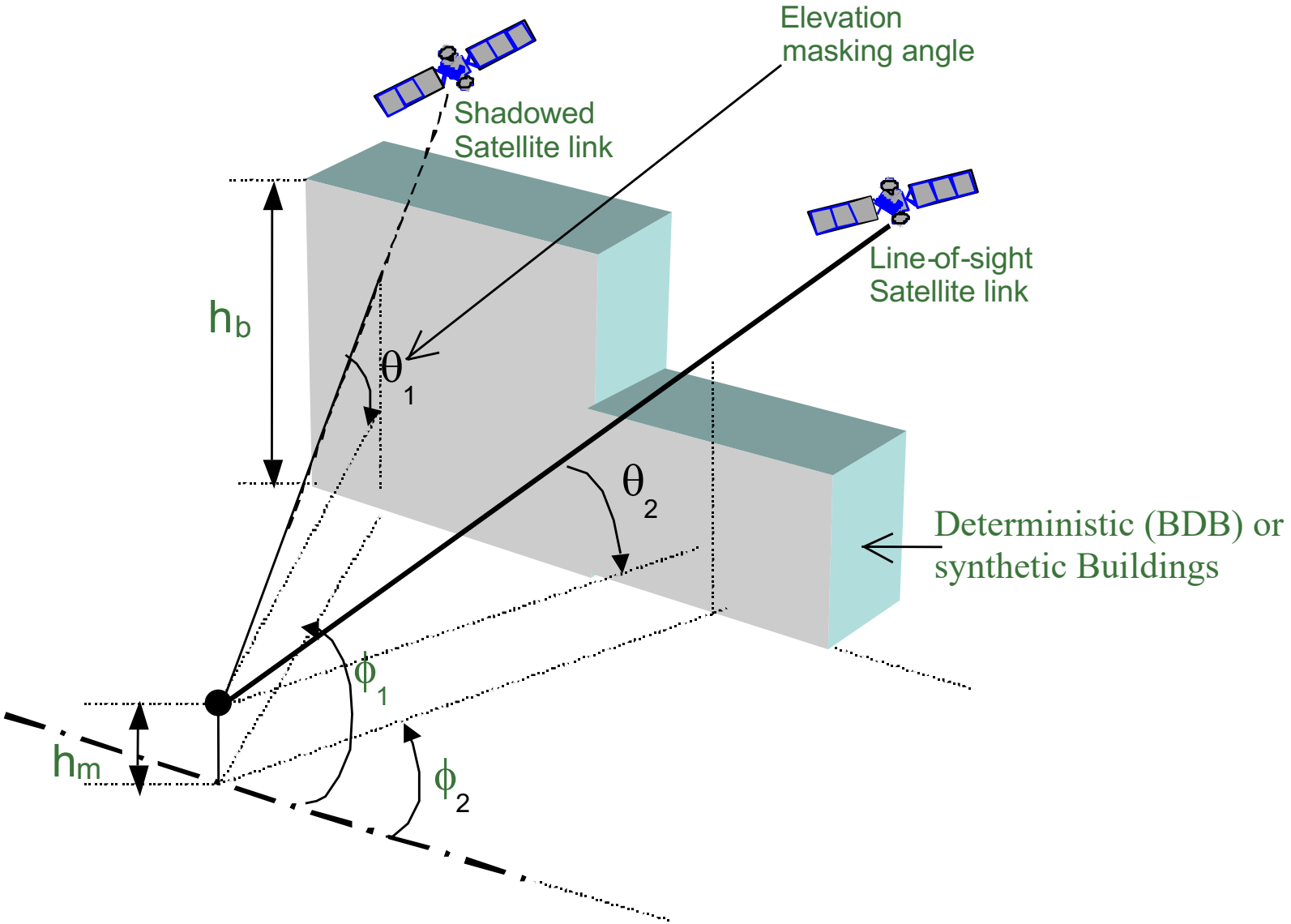


Performance Evaluation

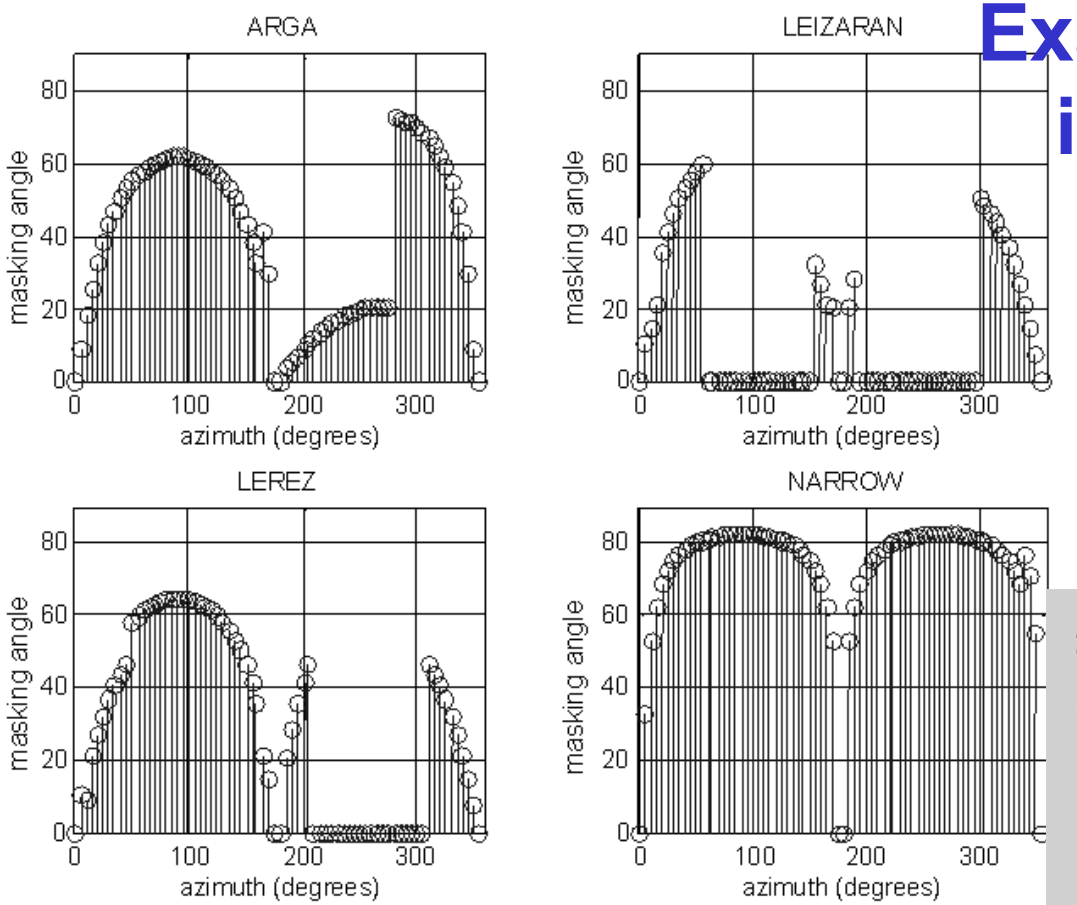
Land Mobile Satellite Channel Model

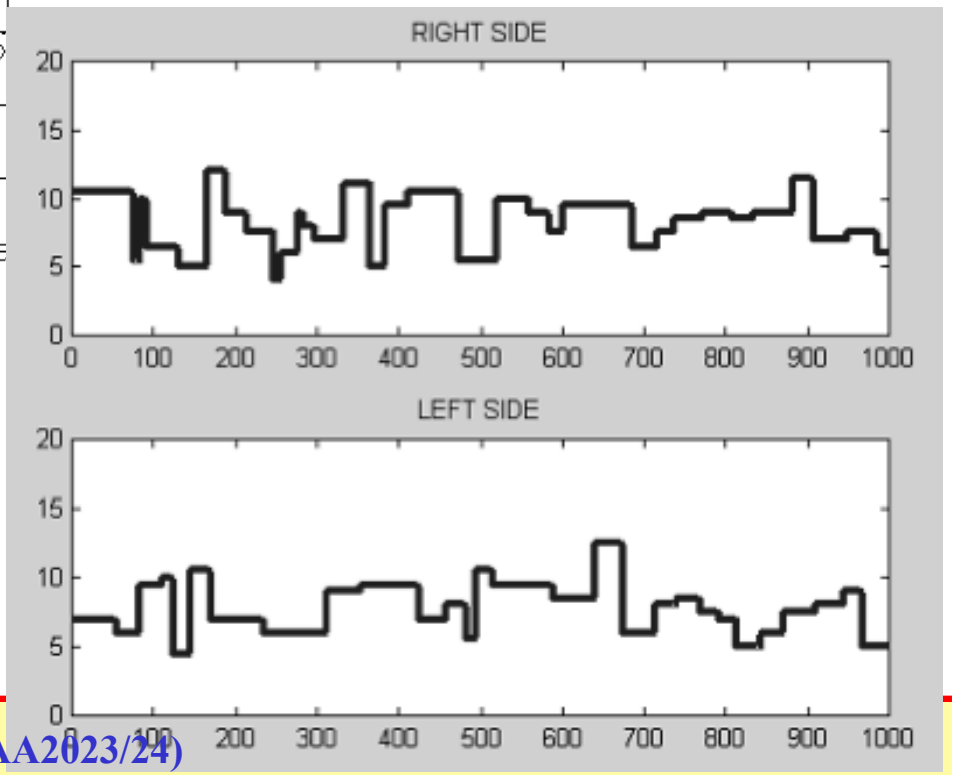
- Deterministic or statistical parameterization of urban environment
  - Canonical geometry traversed by a mobile receiver
  - Canyon street composed of building on both side
  - Measurement campaign
  - Characterization of buildings
  - Simulation environment generation
- Calculation of masking angles
  - Satellite-Terminal geometry computation
  - Channel State determination according to the surrounding environment

Geometrical Scenario



Example: Masking Angles in 4 streets of Madrid





Skyline Generated with Parameters of London

Mobile Terminal Experiment and Platform

- Mobile moves up and down the street (at constant speed)
- Shadowing (we assume complete blocking) occurs when satellite is hidden by skyline
- Handoff is invoked when satellite is shadowed or goes below the horizon
- Look for "unshadowed" satellite with highest elevation angle
- Satellite "diversity": two or more satellites always in view (e.g., Globalstar)

- Network Simulator NS-2
- Terminal Mobility and Shadowing Channel
  - Land Mobile Channel Model implementation
- Gateway
  - A SATNODE that can connect more than one satellite
  - Capability of routing between different orbits in non polar constellation
- Handoff due to channel impairments
  - The HO procedure has been related to the mobile channel

Experiments

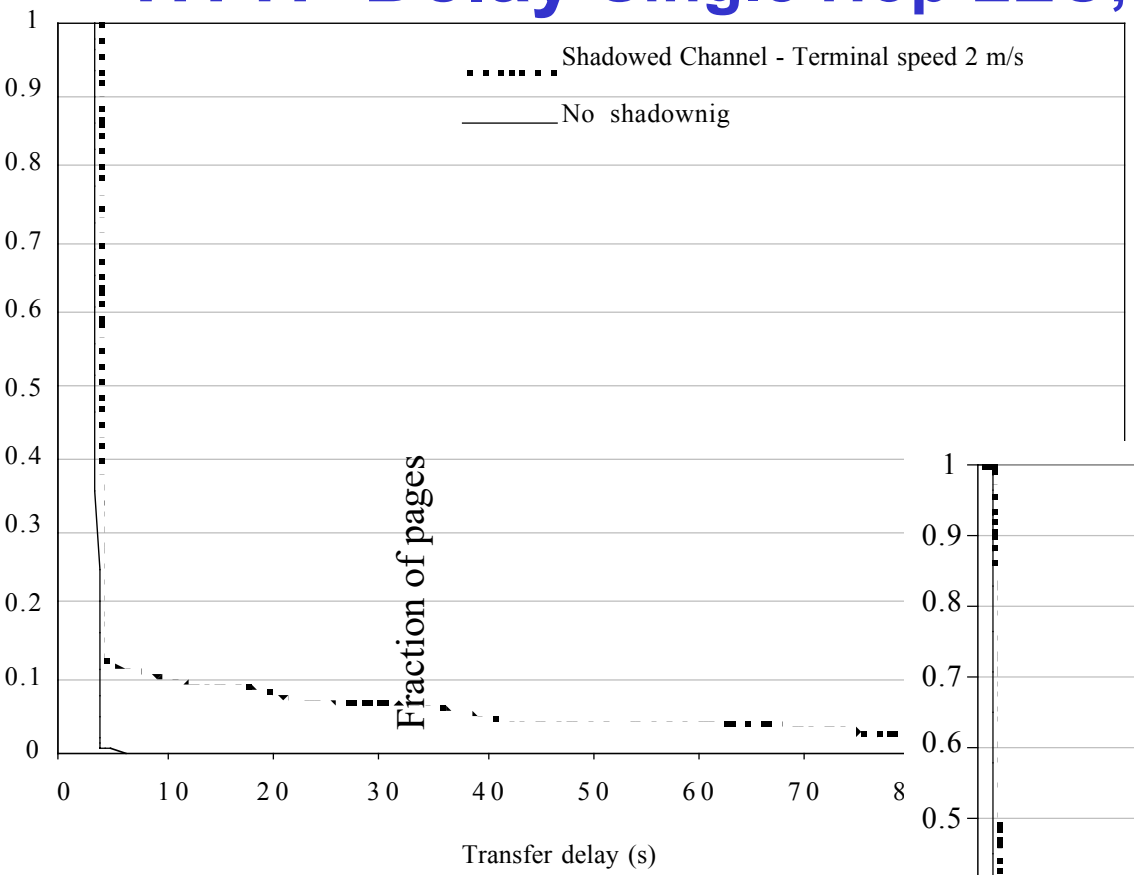
Scenarios

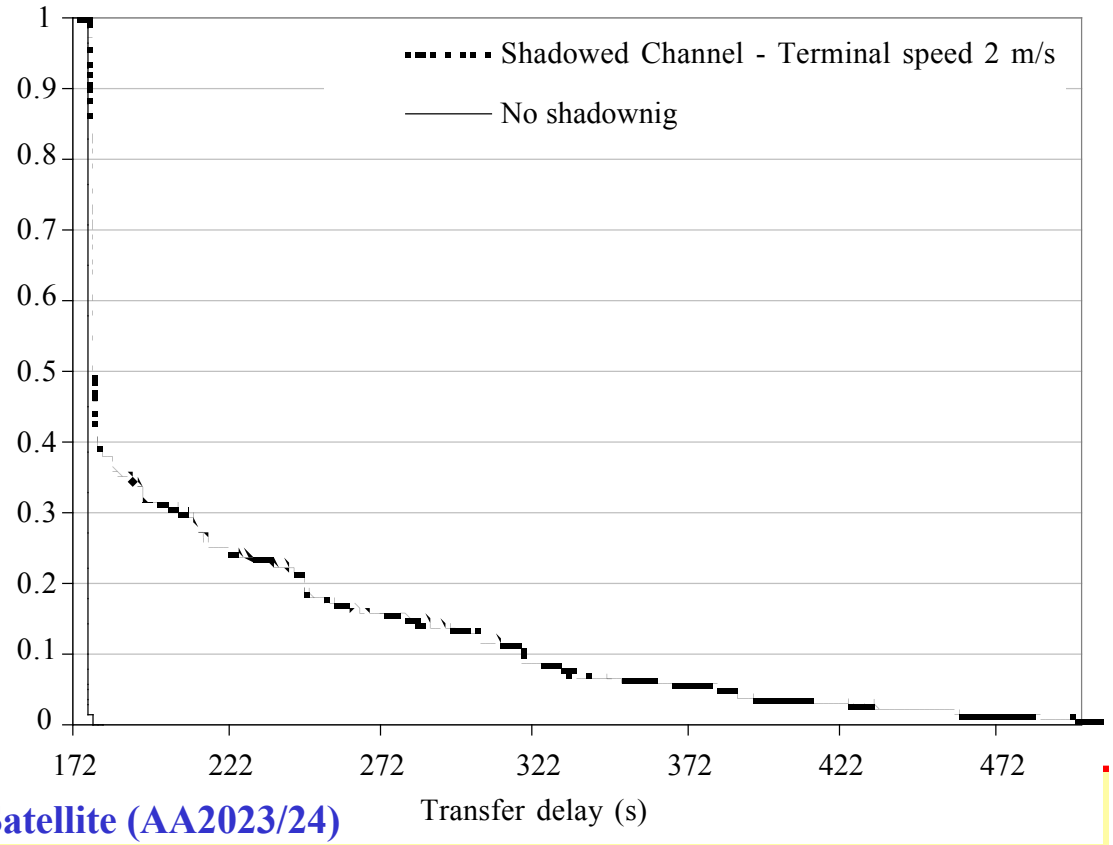
- LEO satellite scenarios
  - Single-hop
    - A terminal is connected to the gateway by one satellite
    - No ISL needed
  - Full satellite network
    - LEO satellite network provides full connectivity between two points on the Earth surface
    - Globalstar-like constellation assumed to have Intra orbit ISL; the Gateway provides connectivity among different orbits
- GEO satellite scenarios
  - single-hop (Rome -Sat- WDC)
  - two-hops with intermediate ISL (Rome-Sat1-Sat2-WDC)

Traffic Models and Channel Options

- HTTP
  - 5 pages of 5 kbytes each
  - Viewing time: 40s
- FTP
  - Fixed size transfer to get delay statistics
  - Fixed duration session to get throughput results
- Constant PER
  - Uniformly distributed
- Land Mobile Channel

HTTP Delay Single Hop LEO, 2 m/s, Shadowed, No PER

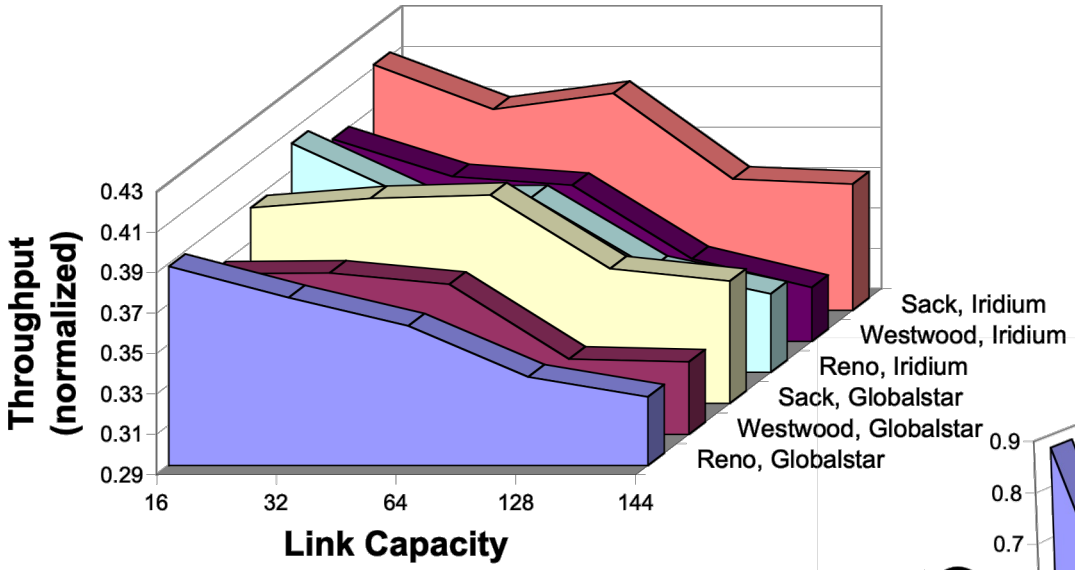




HTTP Delay
Single Hop LEO, 2 m/s, Shadowed, No PER

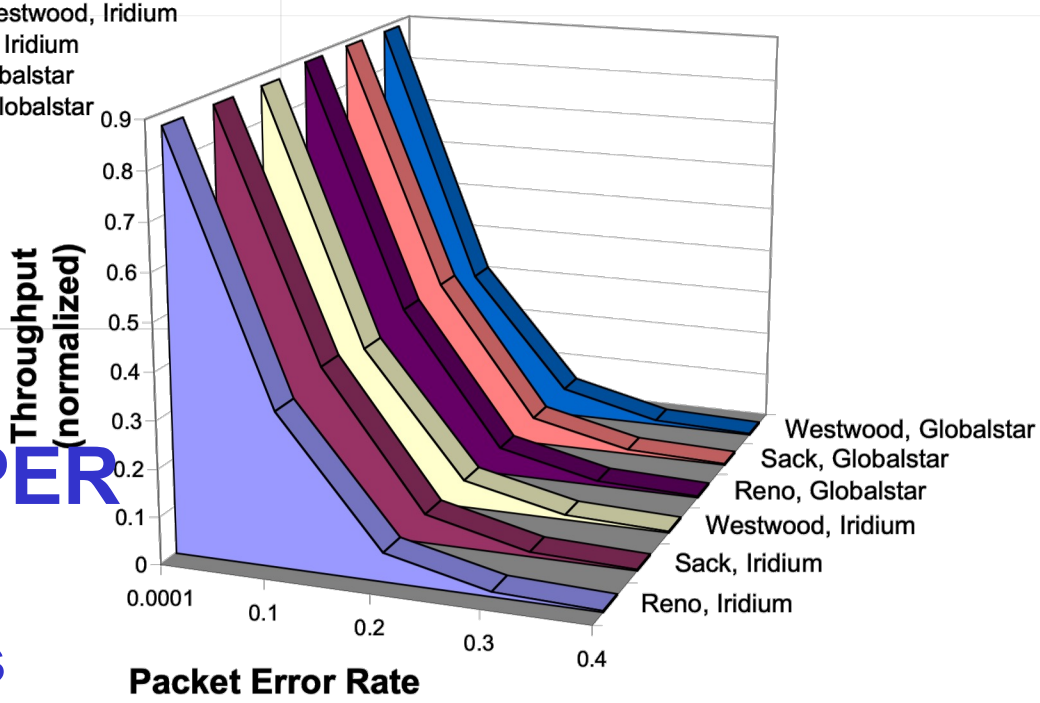
FTP Performance vs. Capacity

Single Hop LEO, Unshadowed, 10% PER



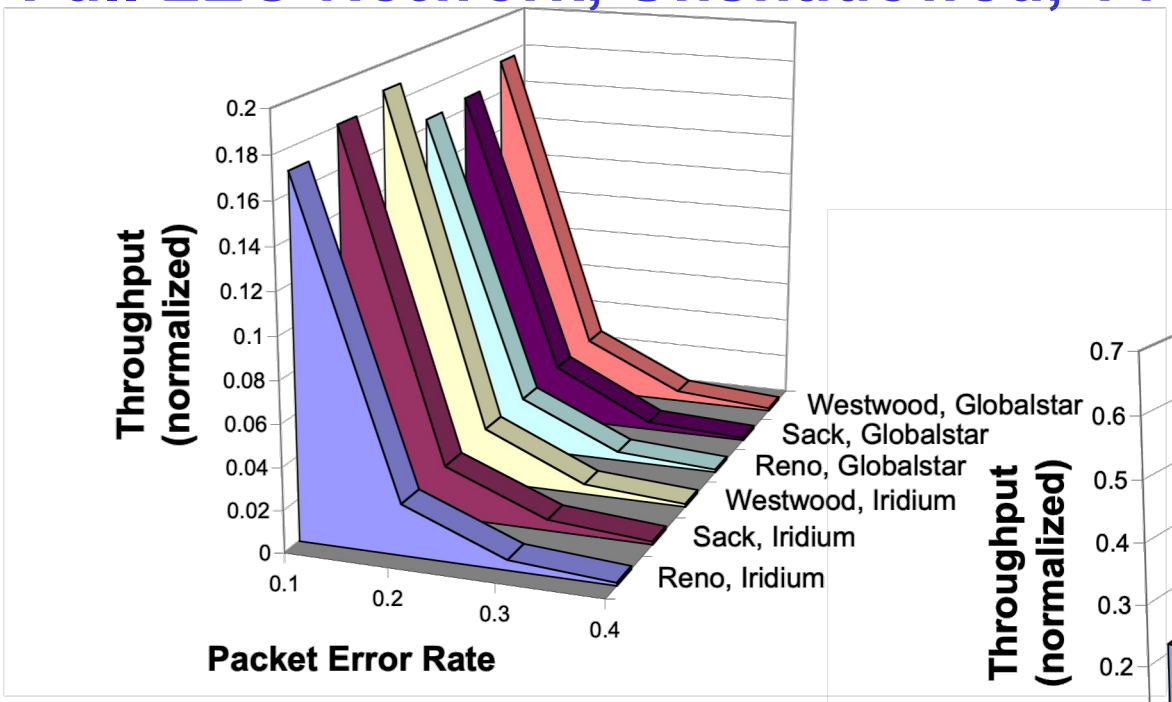
FTP Performance vs. PER

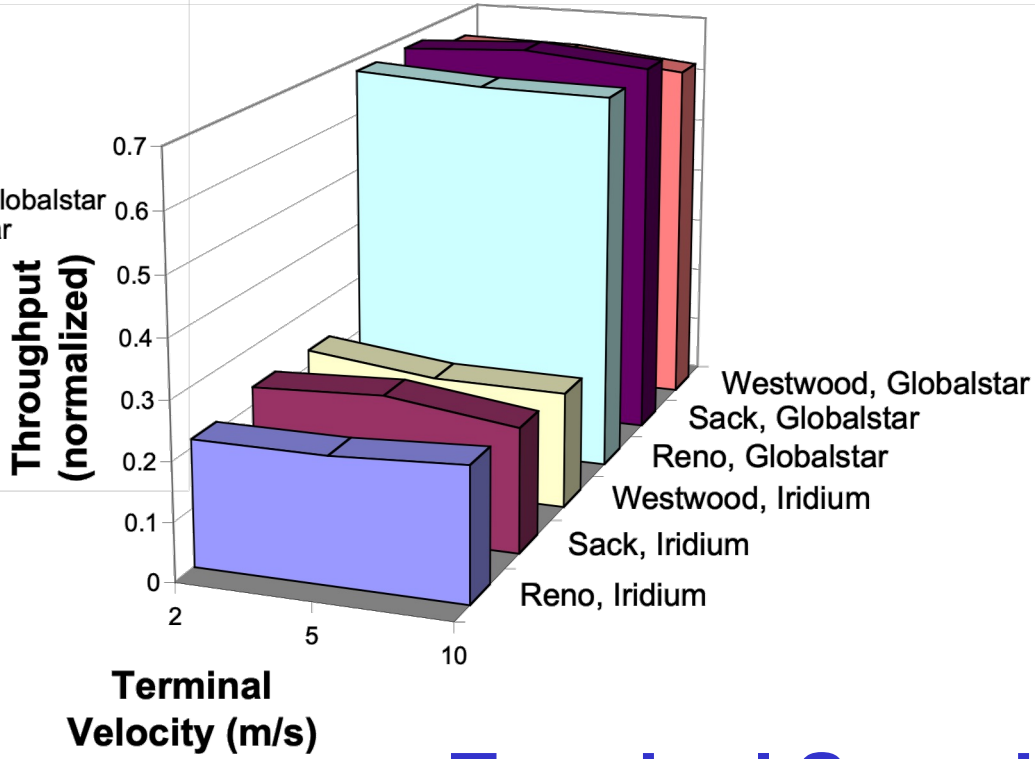
Single Hop LEO, Unshadowed, 144 kbit/s



FTP Performance vs. PER

Full LEO Network, Unshadowed, 144 kbit/s



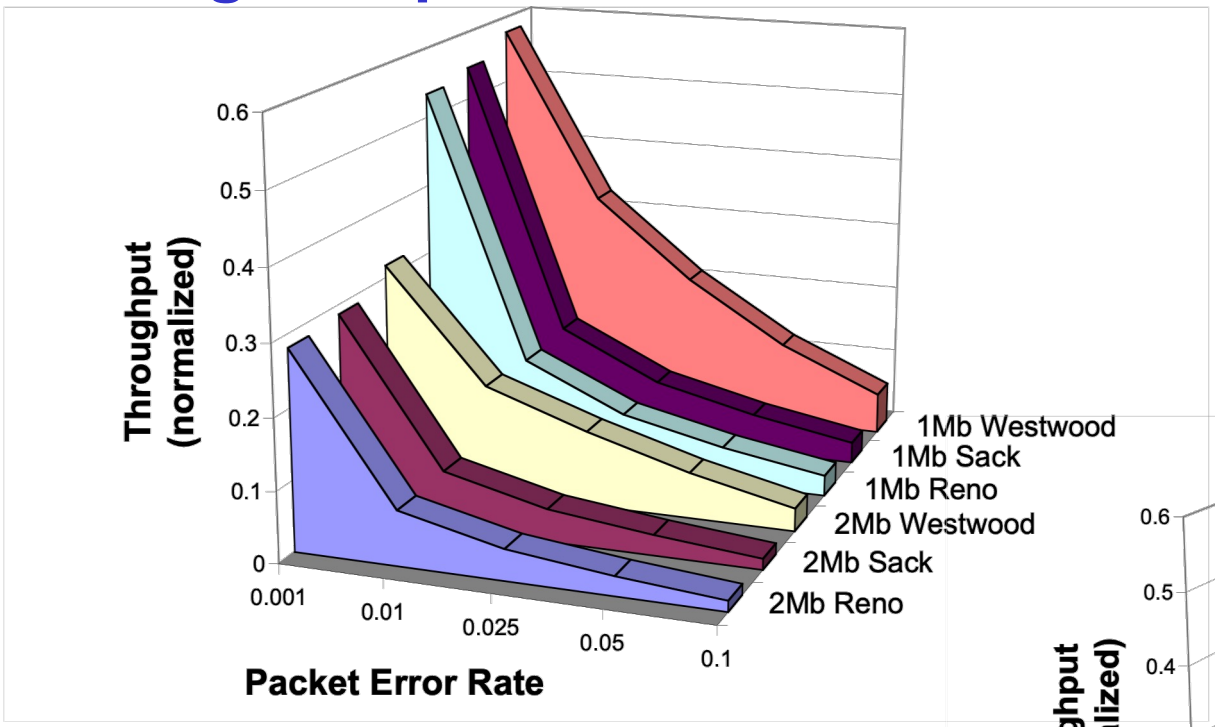


FTP Performance vs. Terminal Speed

Single Hop LEO, Shadowed, 144 kbit/s

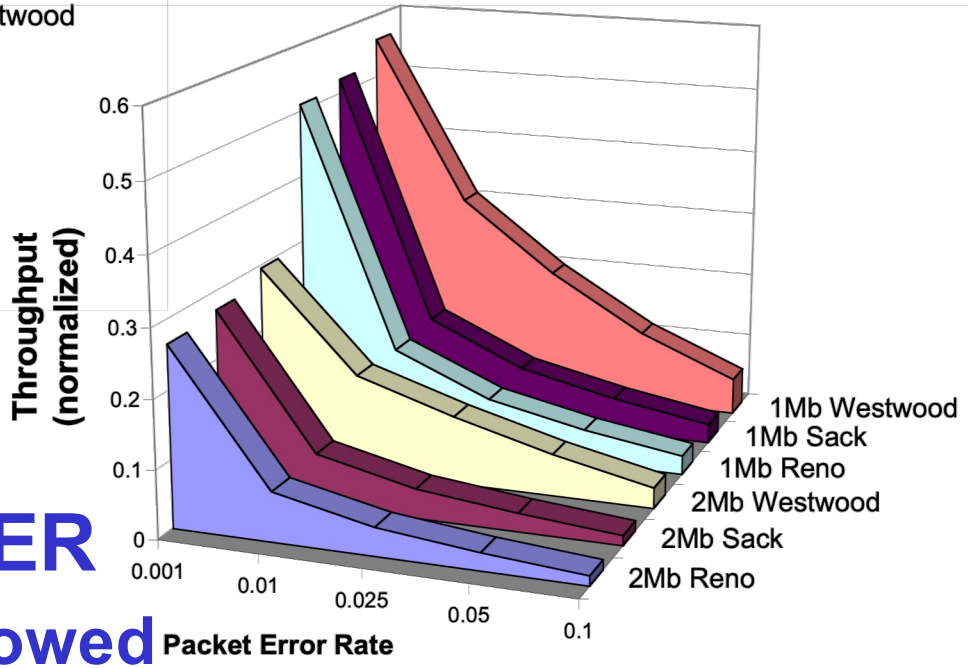
FTP Performance vs. PER

Single Hop GEO, Unshadowed



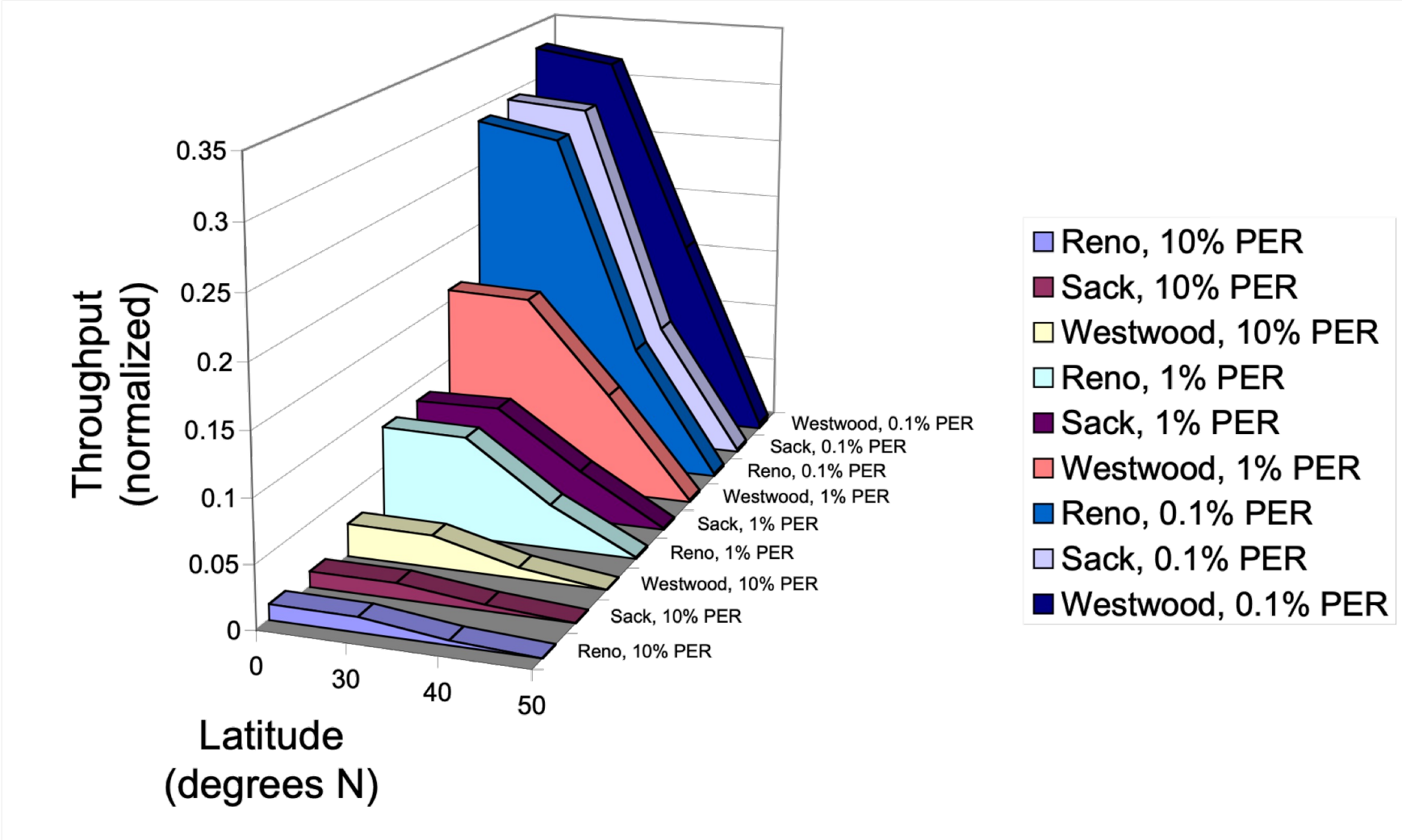
FTP throughput vs. PER

Double Hop GEO, Unshadowed



FTP Performance vs. Latitude and PER

Single Hop GEO, Shadowed, 2 Mbps



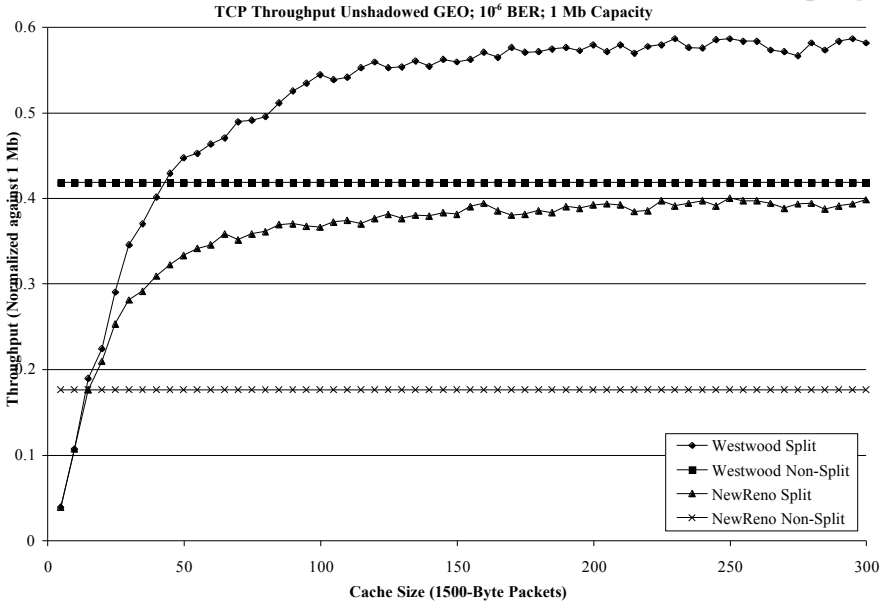
Split on satellite: assumptions and simulation environment

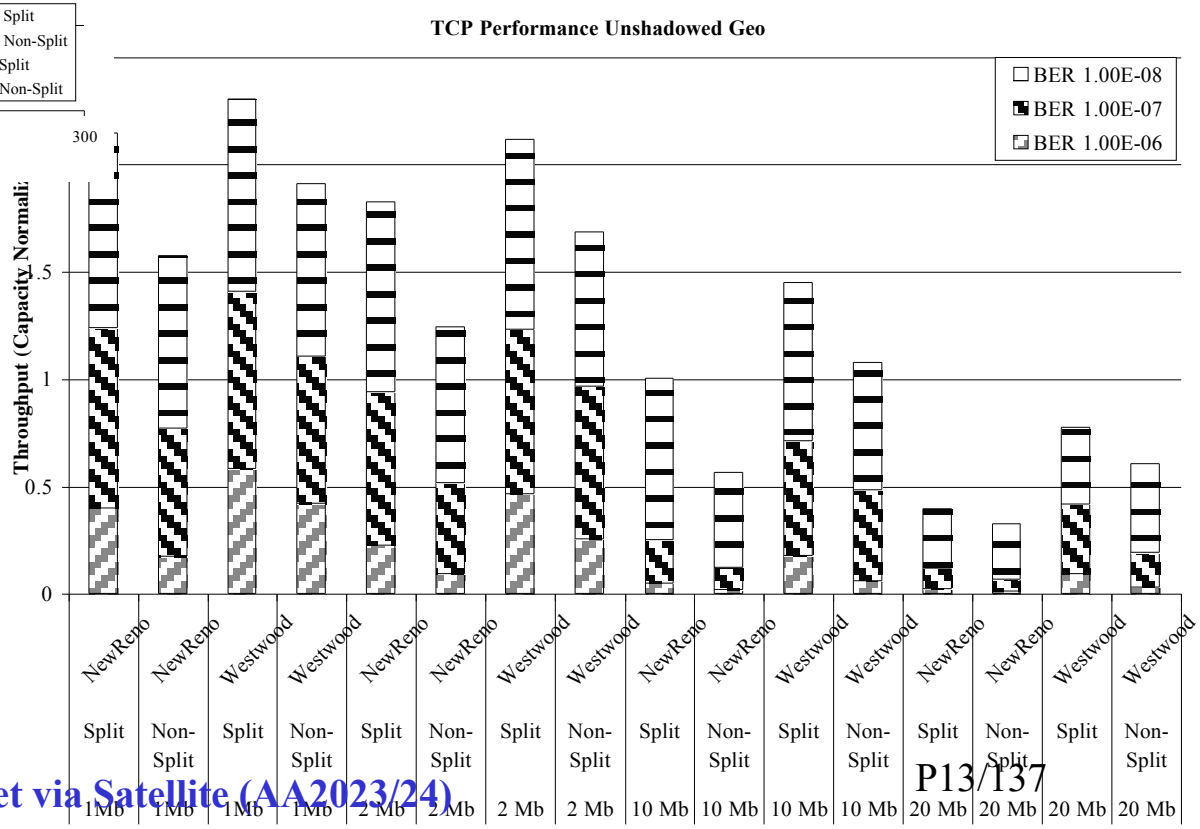
- The cache is composed of three segments
- Both shadowed and unshadowed case investigated
- Unshadowed case: 100 FTP sessions lasting 31 s each
- Shadowed case: 1000 FTP sessions lasting 31 s each
- 1500-byte packets
- TCP window sizes not limited by the buffer space of the source and sink nodes
- ns-2 simulator
- Only satellite link considered (the terrestrial tails are not relevant)
- Land Mobile Satellite Channel Model
  - Physical statistical model
  - Geometrical projections of buildings surrounding the terminal
- On board Forwarding Agent Model
  - ns-2 was enhanced with a new TCP forwarding agent
  - It receives data and ack from the uplink while sending and retransmitting packets on the downlink
- Advertised window uplink = 1/3 of the total cache
- Window on the down link = 1/3 of the total
- The advertised window is dynamic (ns-2 was suitably modified)

Simulation Scenarios

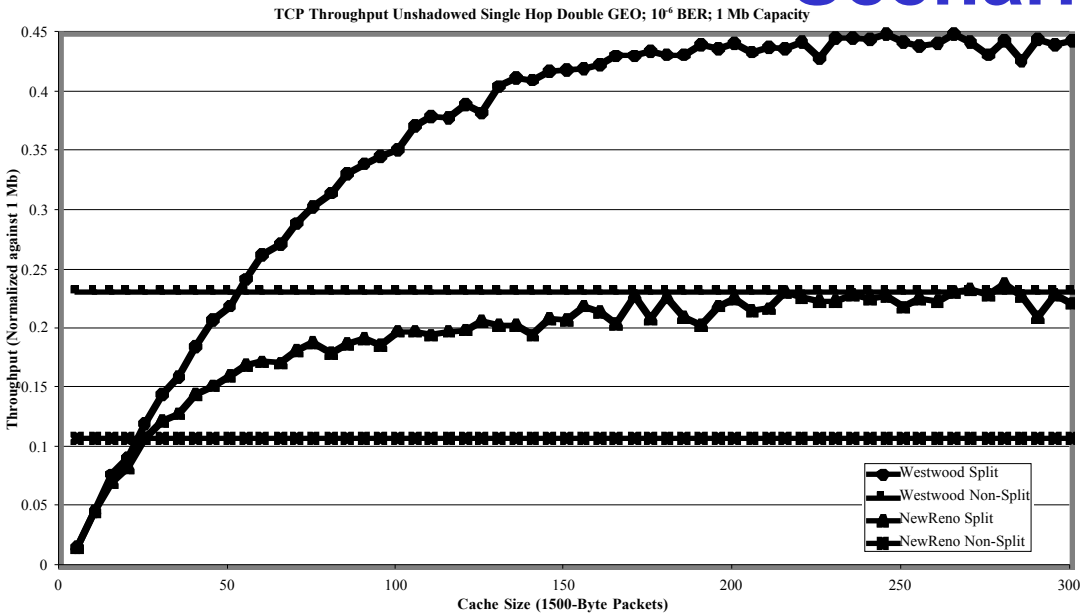
1. Single-Hop Unshadowed GEO: Two terminals connect via single GEO satellite.
2. Single-Hop Unshadowed double GEO: Two terminals connect via two GEO satellites. The GEOs communicate using an ISL.
3. Single-Hop Source-Shadowed GEO: as in the first case, but the TCP sender suffer from shadowing as described above.
4. Single-Hop Sink-Shadowed GEO: as in the first case, but the TCP receiver suffers from shadowing.

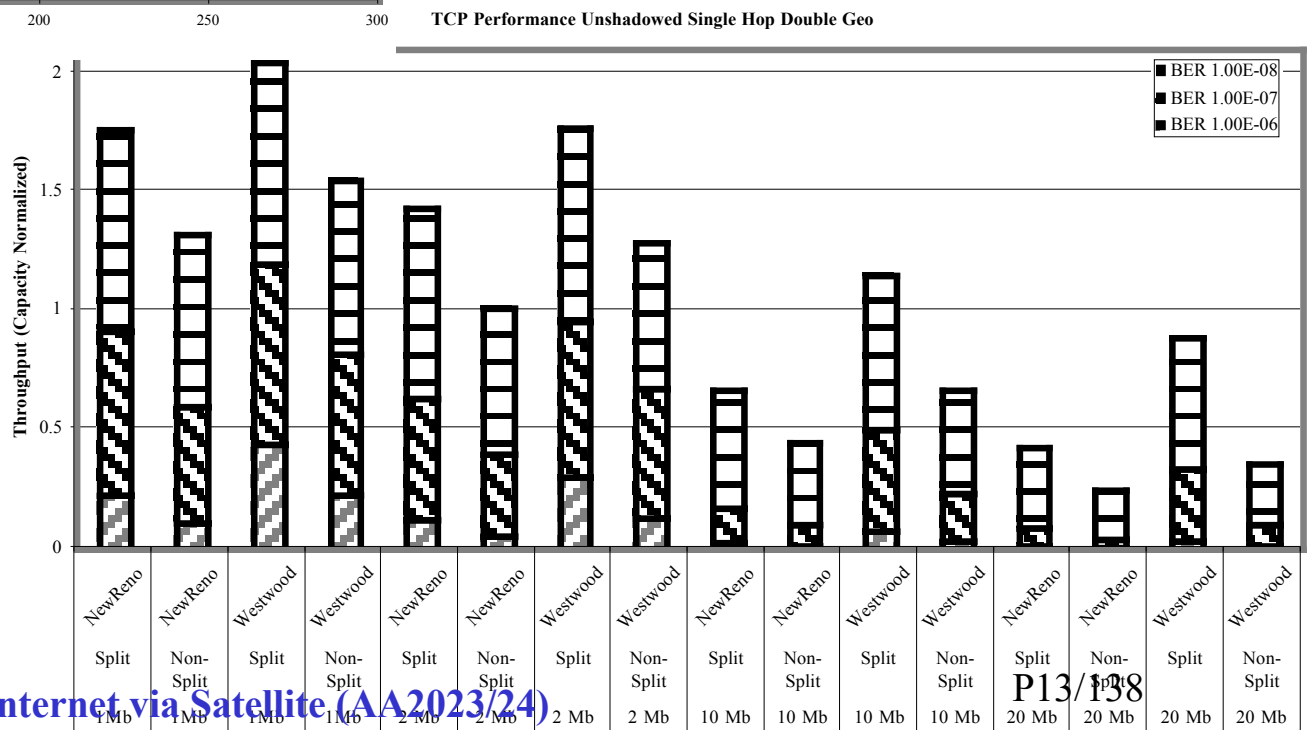
Scenario 1


TCP Throughput Unshadowed GEO; 10^-6 BER; 1 Mb Capacity
Throughput (Normalized against 1 Mb)
Cache Size (1500-Byte Packets)
Westwood Split
Westwood Non-Split
NewReno Split
NewReno Non-Split

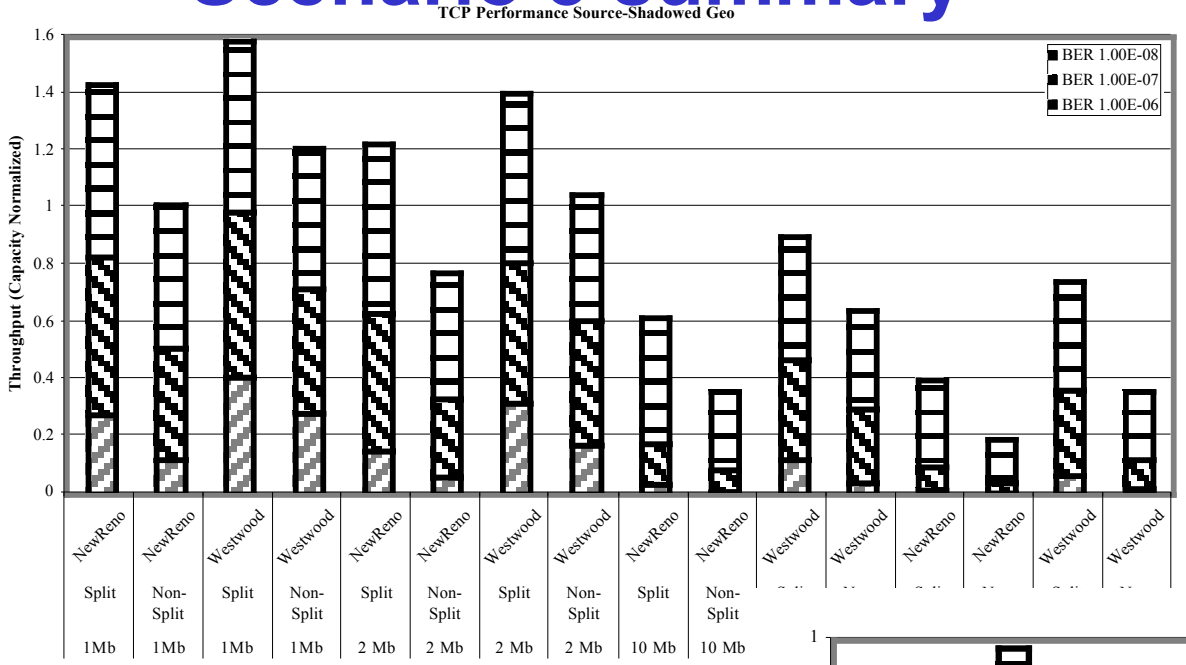

TCP Performance Unshadowed Geo
BER 1.00E-08
BER 1.00E-07
BER 1.00E-06
Throughput (Capacity Normali
NewReno Split
NewReno Non-Split
Westwood Split
Westwood Non-Split
1Mb
2 Mb
10 Mb
20 Mb

Scenario 2

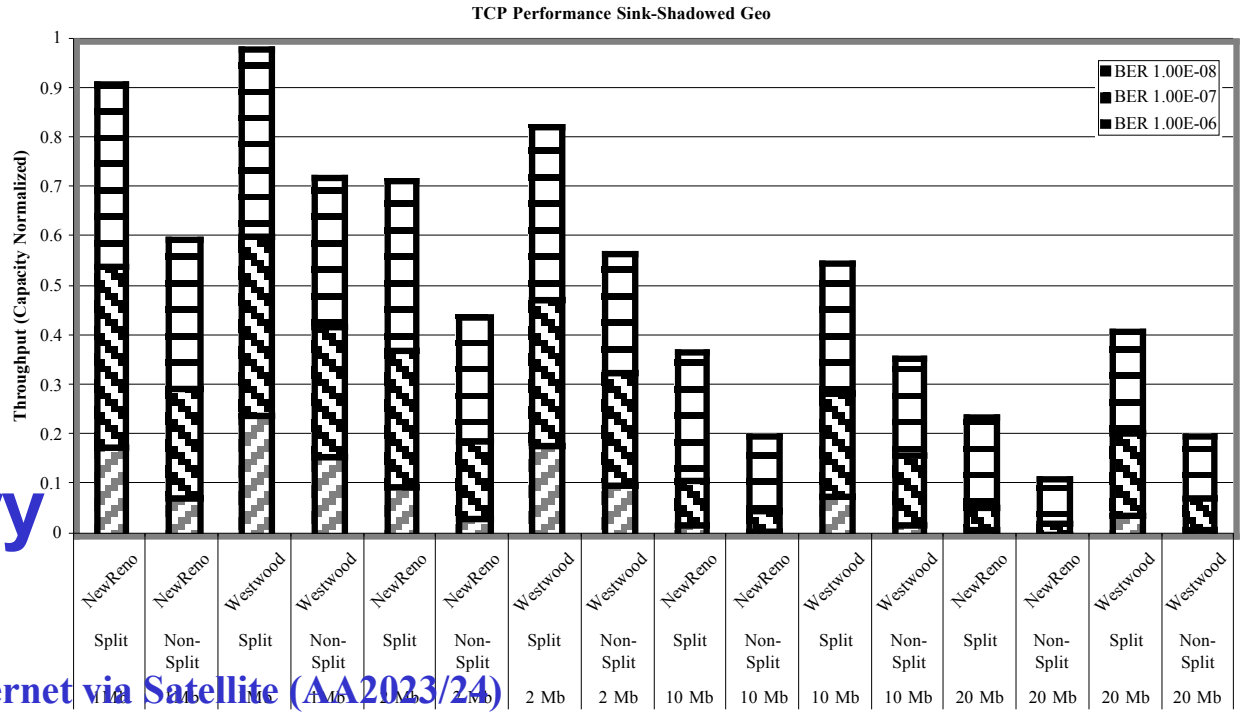

TCP Throughput Unshadowed Single Hop Double GEO; 10-6 BER; 1 Mb Capacity
Throughput (Normalized against 1 Mb)
Cache Size (1500-Byte Packets)
Westwood Split
Westwood Non-Split
NewReno Split
NewReno Non-Split


TCP Performance Unshadowed Single Hop Double Geo
Throughput (Capacity Normalized)
BER 1.00E-08
BER 1.00E-07
BER 1.00E-06
NewReno Split 1Mb
NewReno Non-Split 1Mb
Westwood Split 1Mb
Westwood Non-Split 1Mb
NewReno Split 2 Mb
NewReno Non-Split 2 Mb
Westwood Split 2 Mb
Westwood Non-Split 2 Mb
NewReno Split 10 Mb
NewReno Non-Split 10 Mb
Westwood Split 10 Mb
Westwood Non-Split 10 Mb
NewReno Split 20 Mb
NewReno Non-Split 20 Mb
Westwood Split 20 Mb
Westwood Non-Split 20 Mb

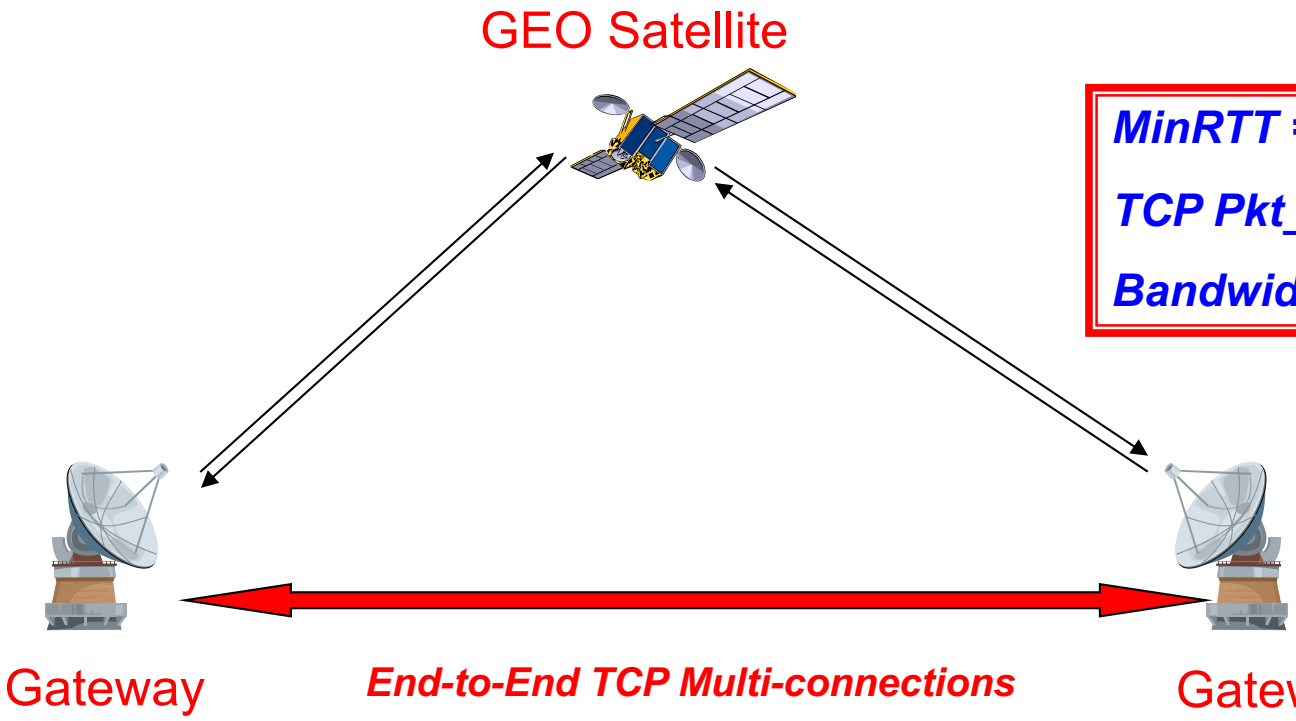
Scenario 3 summary



Scenario 4 summary



Impact of receiver buffer size: simulation scenario



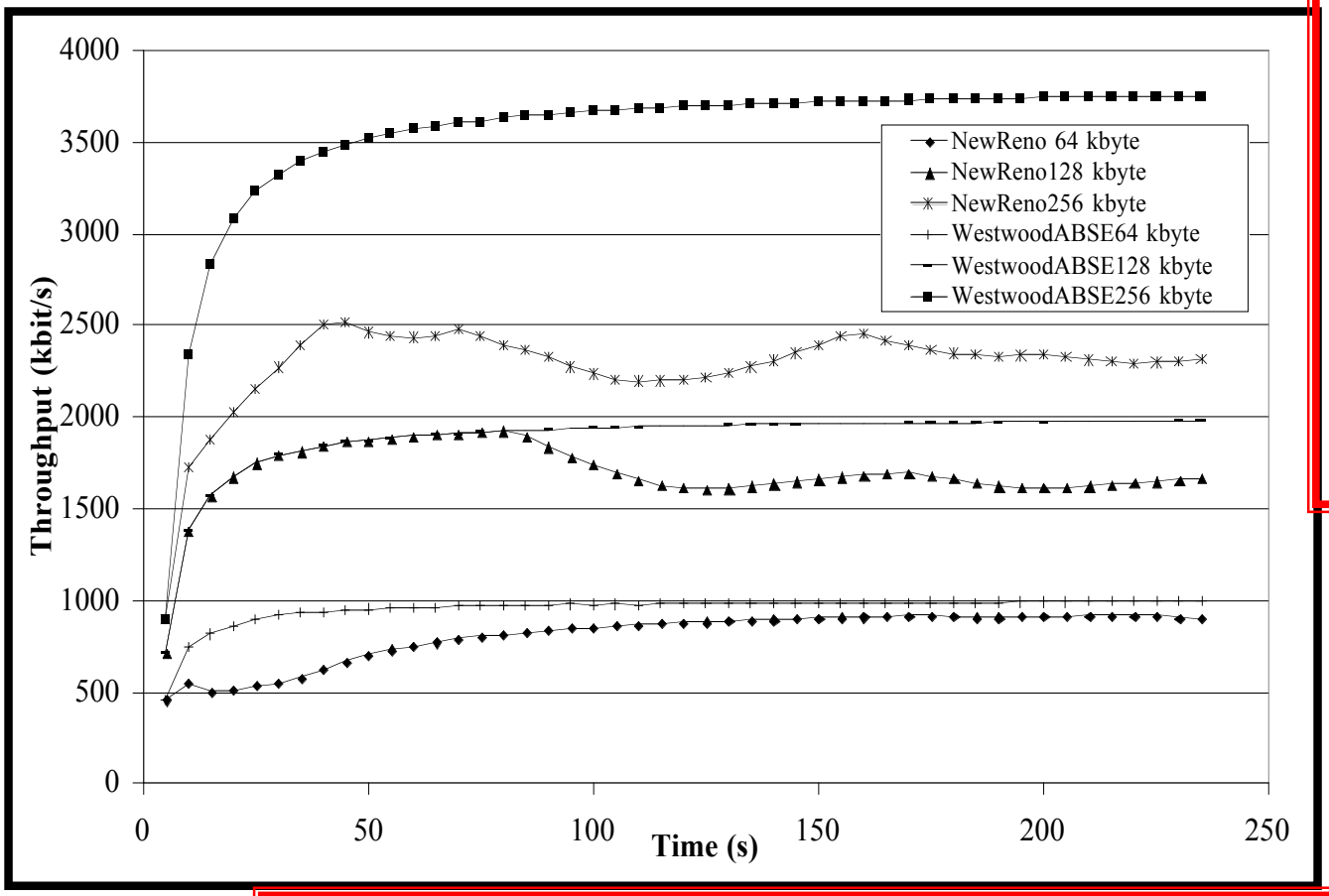
MinRTT = 508 ms

TCP Pkt_size = 1500 bytes

Bandwidth = 4 Mb/s

Transport Protocol: NewReno vs. Westwood ABSE

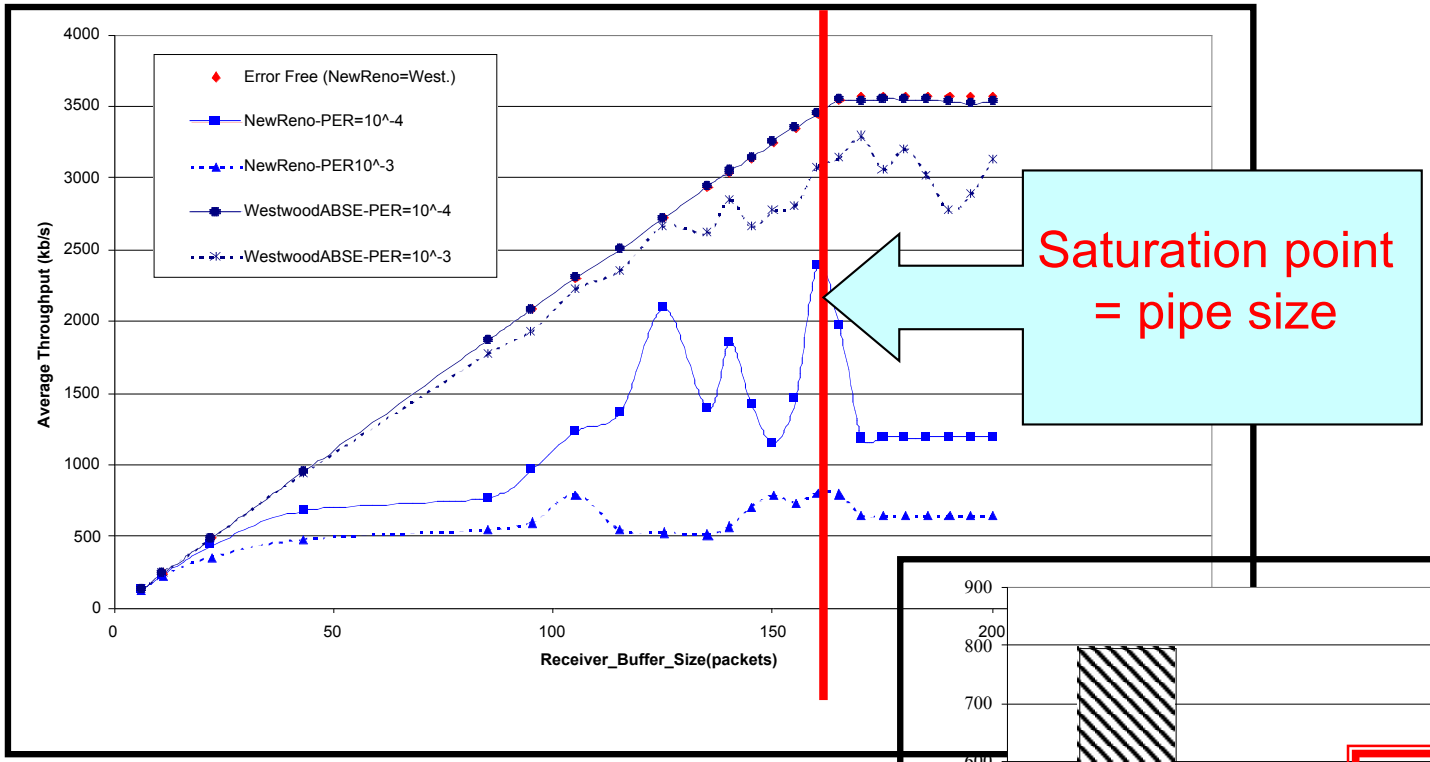
Impact of receiver buffer size: results

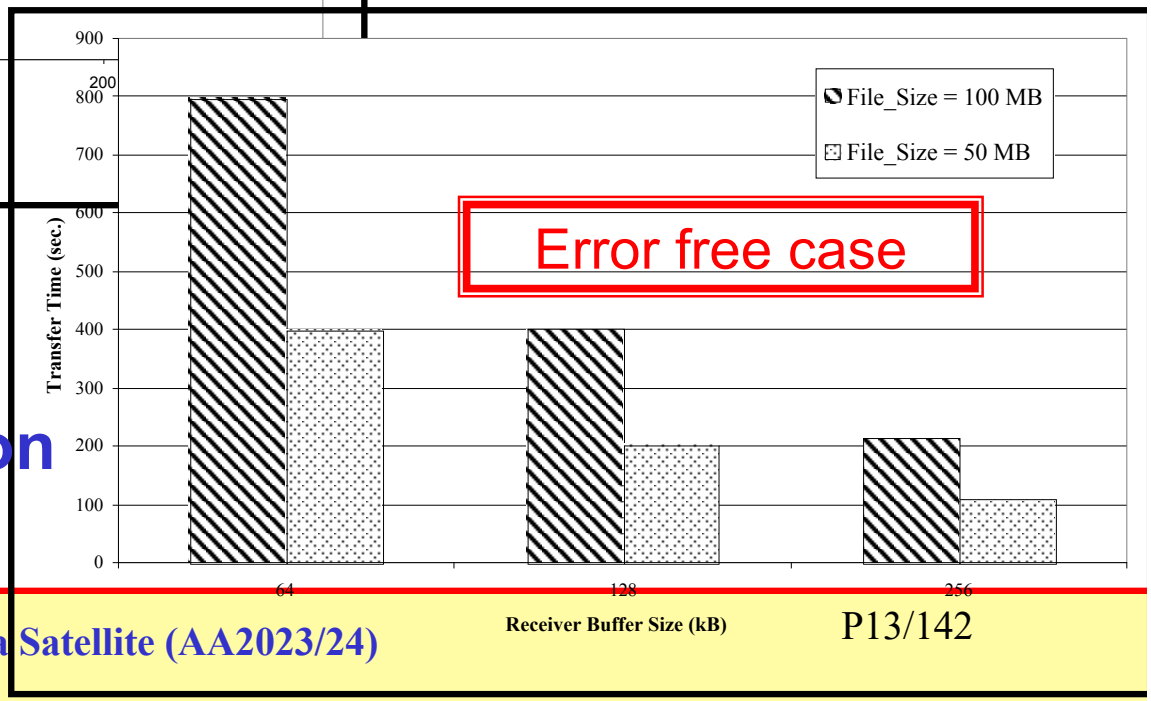


- Single TCP connection
- Low Error Rate (PER=10^{-4})
- Typical default values of "*maximum socket buffer size*":
  - 64 KB (i.e.Window NT, Linux 2.4)
  - 128 KB (i.e. Digital Unix 4.0)
  - 256 KB (i.e. BSD/OS)

"In every case Westwood reaches the maximum throughput allowed from the receiver buffer size, whereas New Reno is strongly limited by the transmission errors"

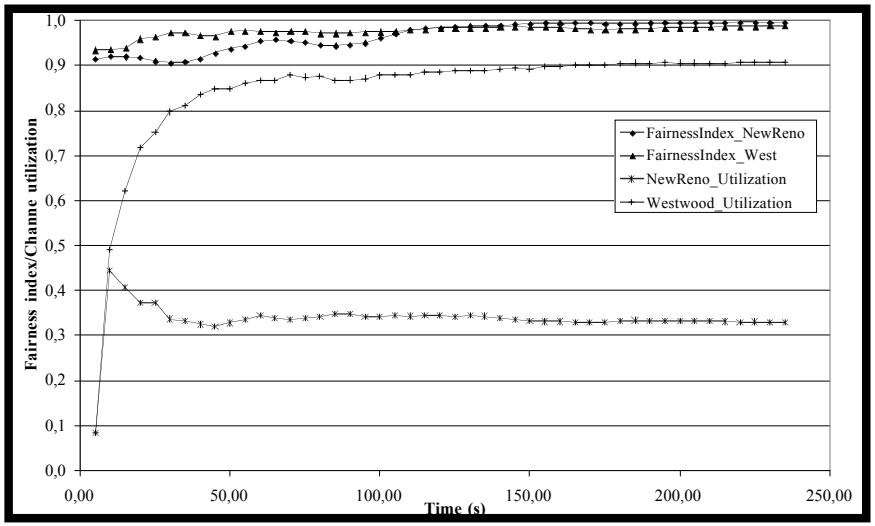
Receiver buffer optimization





Transfer time as a function of the buffer size

Fairness and channel utilitation



$$Fairness\_Index = \frac{\left(\sum x_i\right)^2}{n\sum\left(x_i\right)^2}$$ with $$x_i = \frac{T_i}{O_i}$$

Where:

T_i= measured average throughput;

O_i=fair average throughput

$$Total\_Utilitation = \frac{\sum T_i}{B}$$

Where *B*= channel bandwidth

5 flows considered

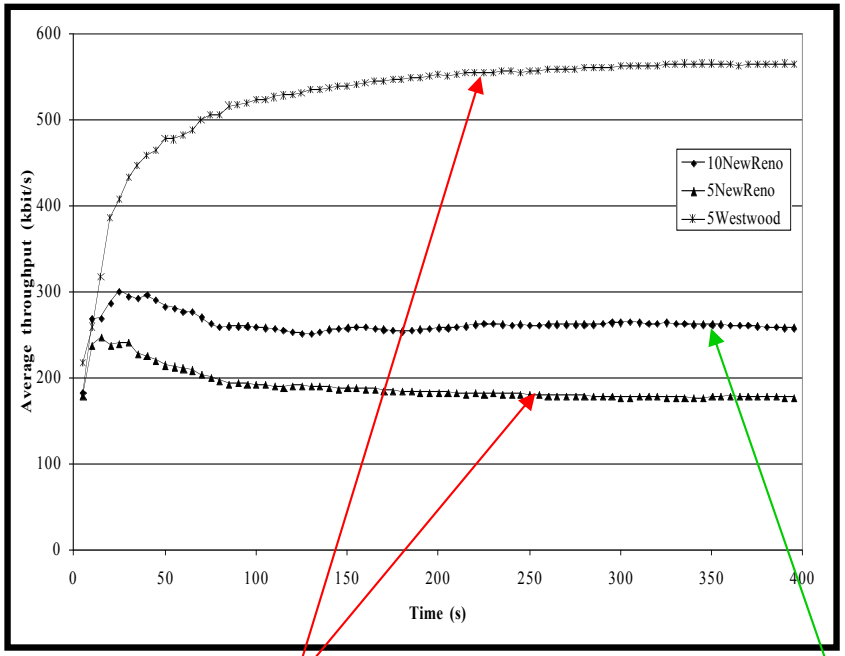
- ***Both NewReno and Westwood share fairly the bandwidth;***
- ***Westwood flows reach the optimal utilitation, whereas NewReno flows waste more than 60% of available bandwidth.***

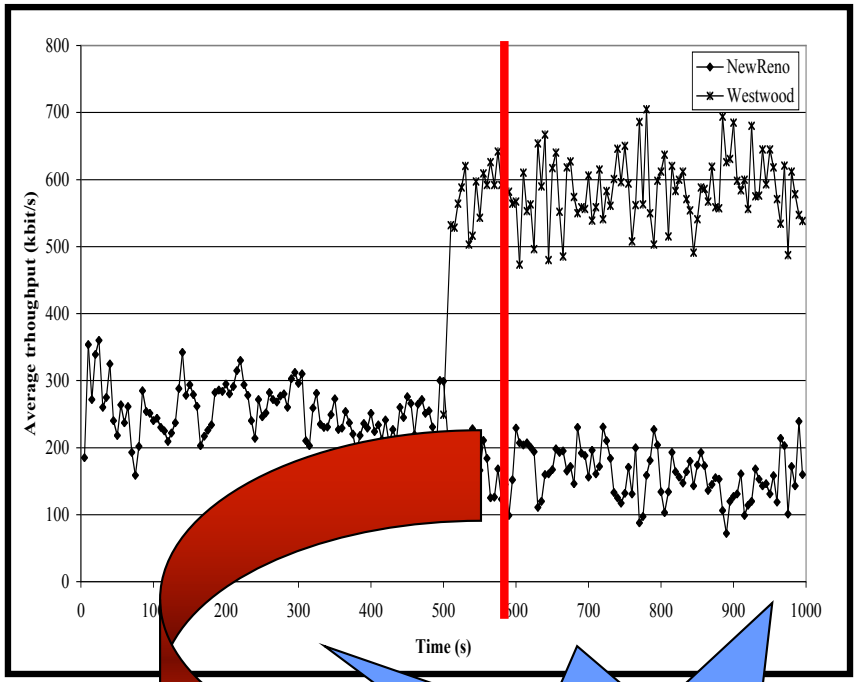
Friendliness

5 NewReno and 5 Westwood flows start simultaneously. We compared the bandwidth sharing with the case in which all the flows support TCP NewReno

Initially, 10 NewReno flows start simultaneously.

After 500 s, 5 Westwood flows replace 5 NewReno flows





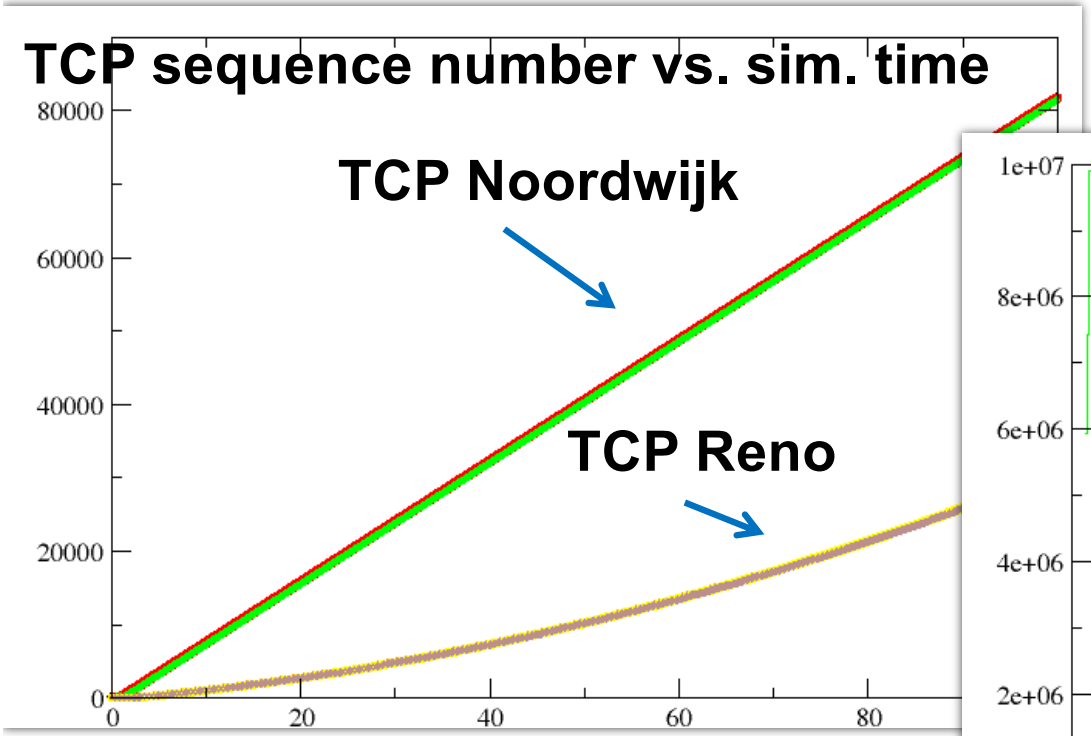
5 NewReno flows with 5 Westwood flows

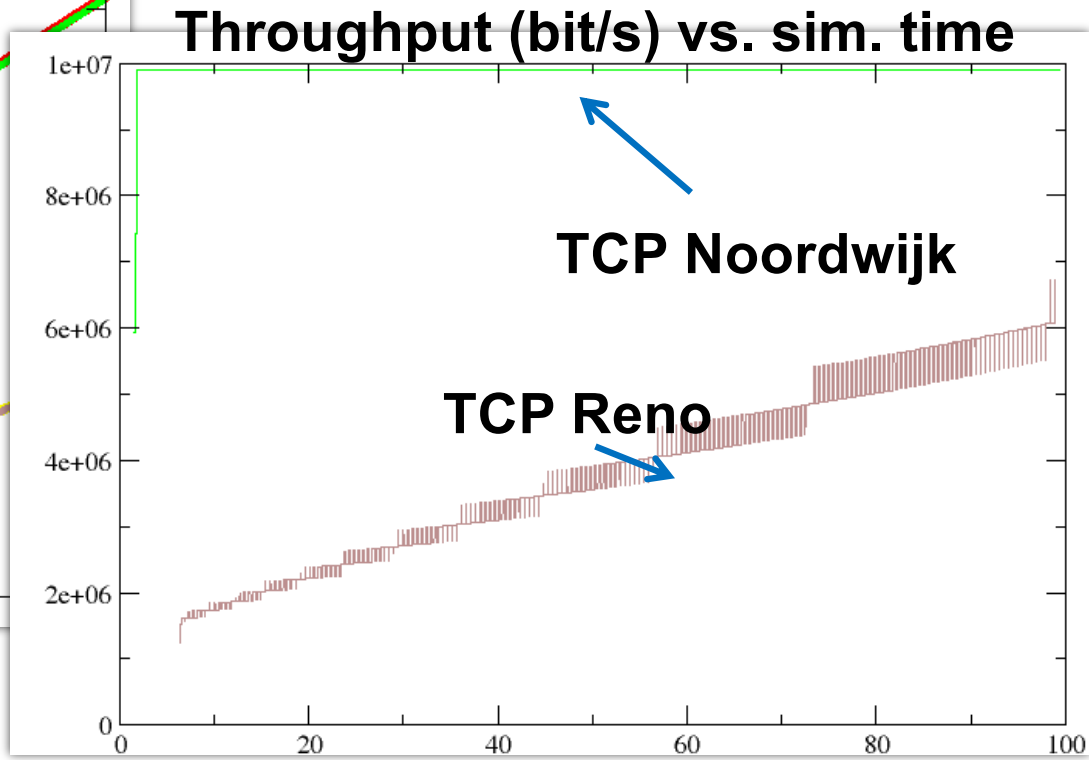
10 NewReno flows

5 Westwood replace 5 NewReno

TCP Noordwijk performance

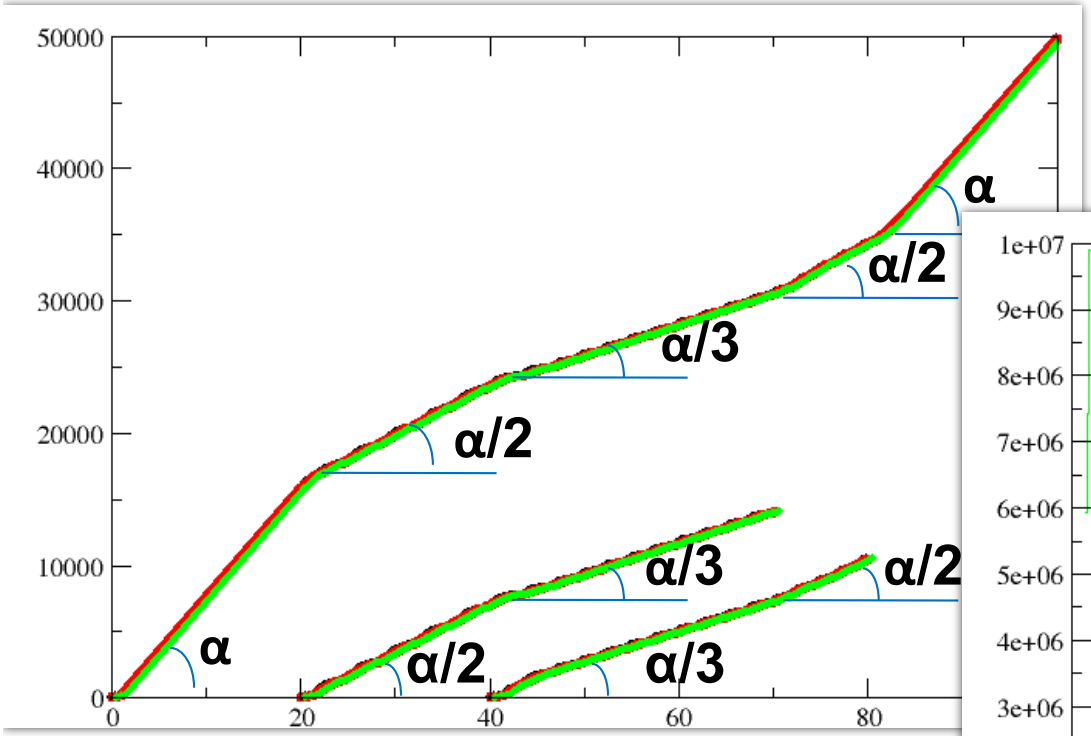
- Performance of a single (long) TCP connection
- FTP download from sat GW to ST





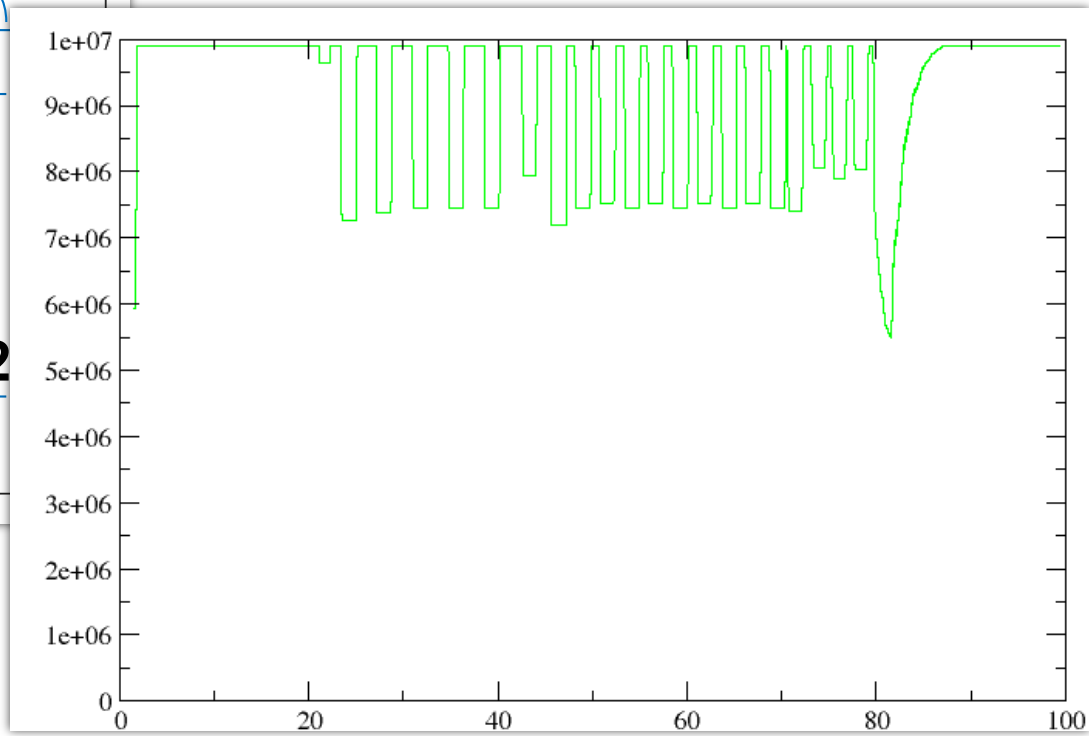
TCP Noordwijk performance (2)

- Multiple TCP Noordwijk connections: fairness & utilization



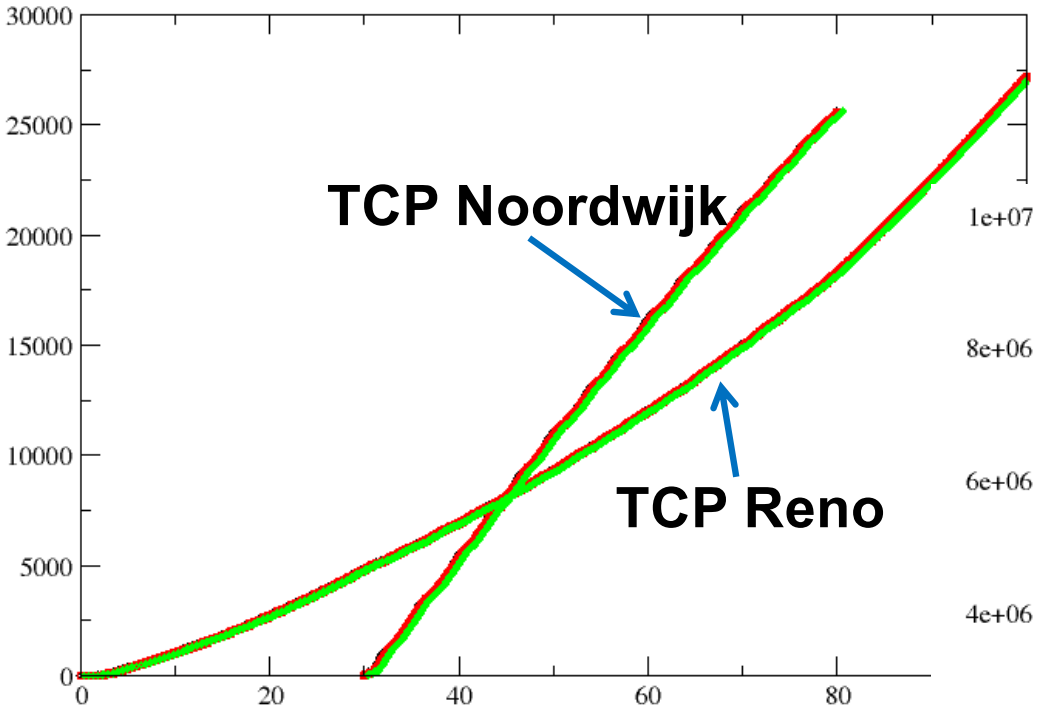
TCP sequence number vs. sim. time

Throughput (bit/s) vs. sim. time



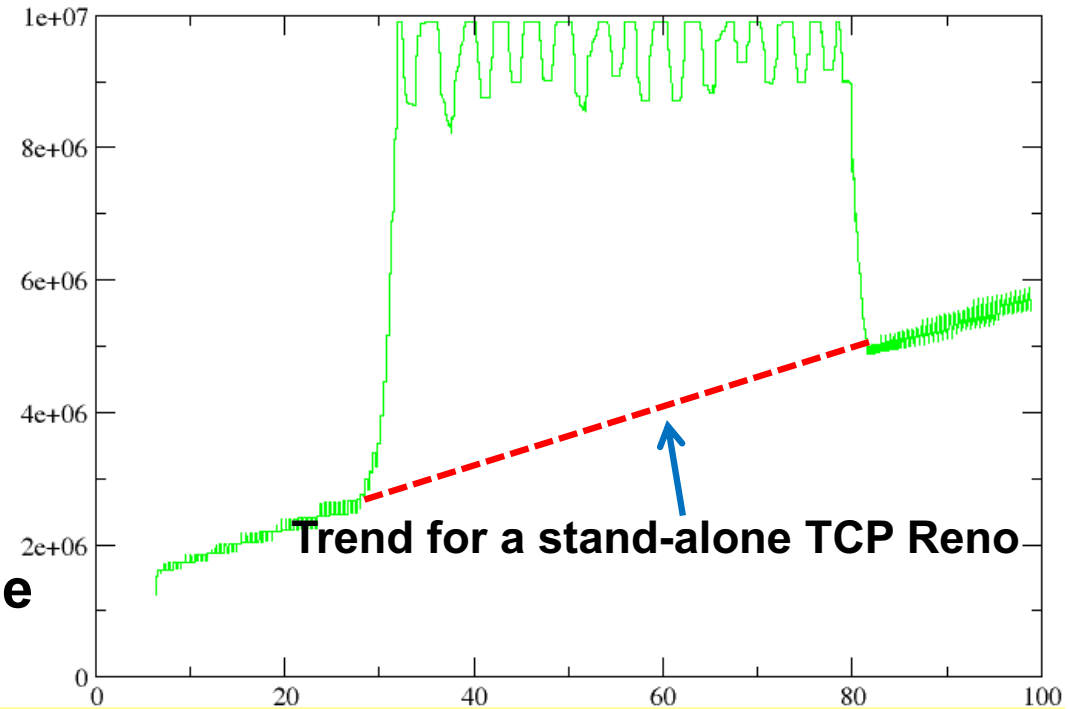
TCP Noordwijk performance (3)

- TCP Noordwijk friendliness : 1 TCP Noordwijk +1 TCP Reno

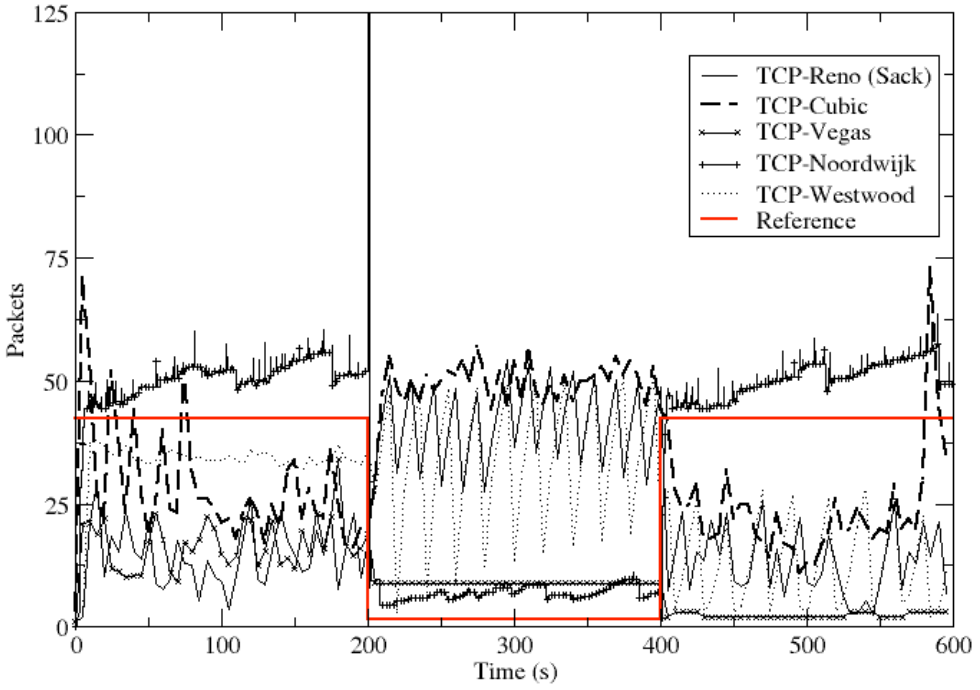


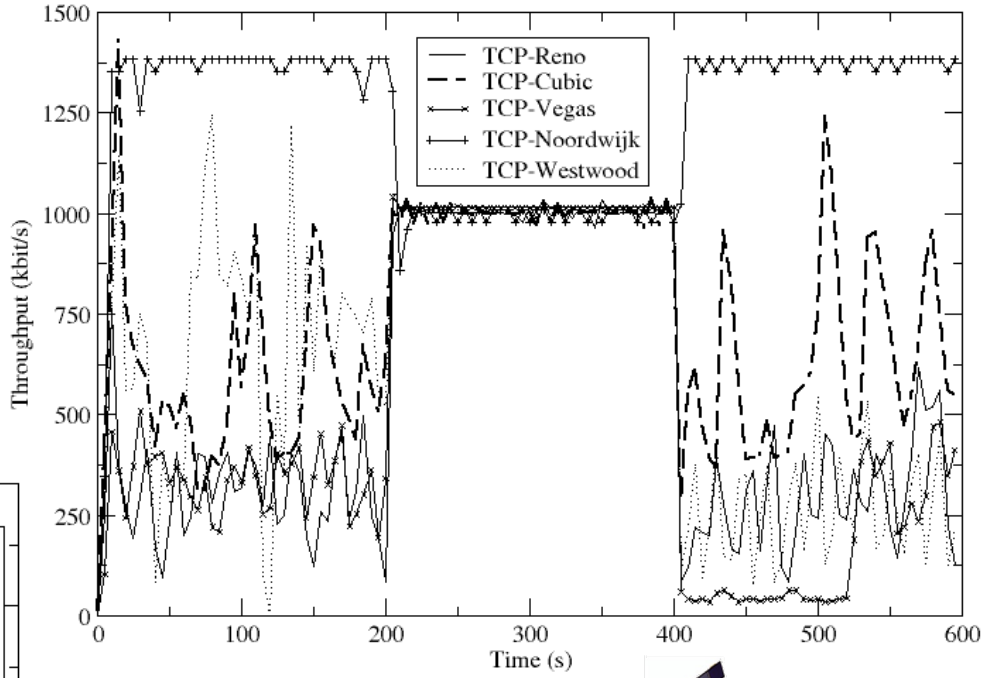
TCP sequence number vs. sim. time

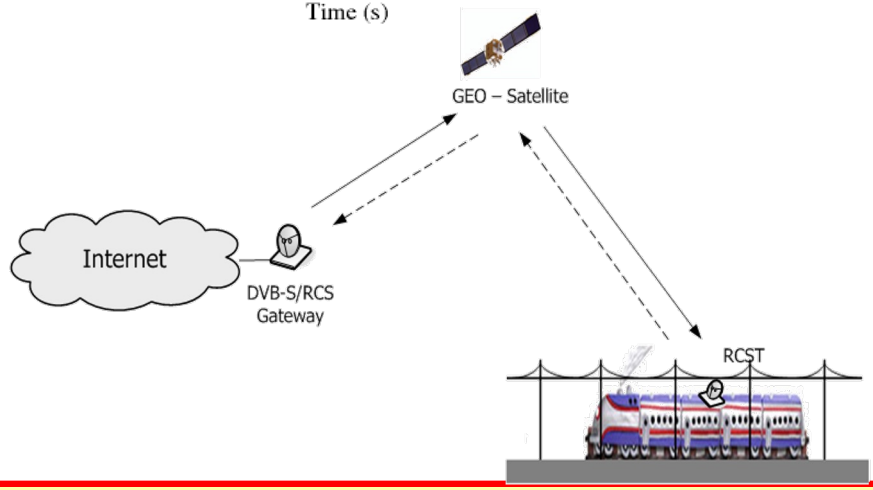
Throughput (bit/s) vs. sim. time



TCP Noordwijk over high speed railway channel



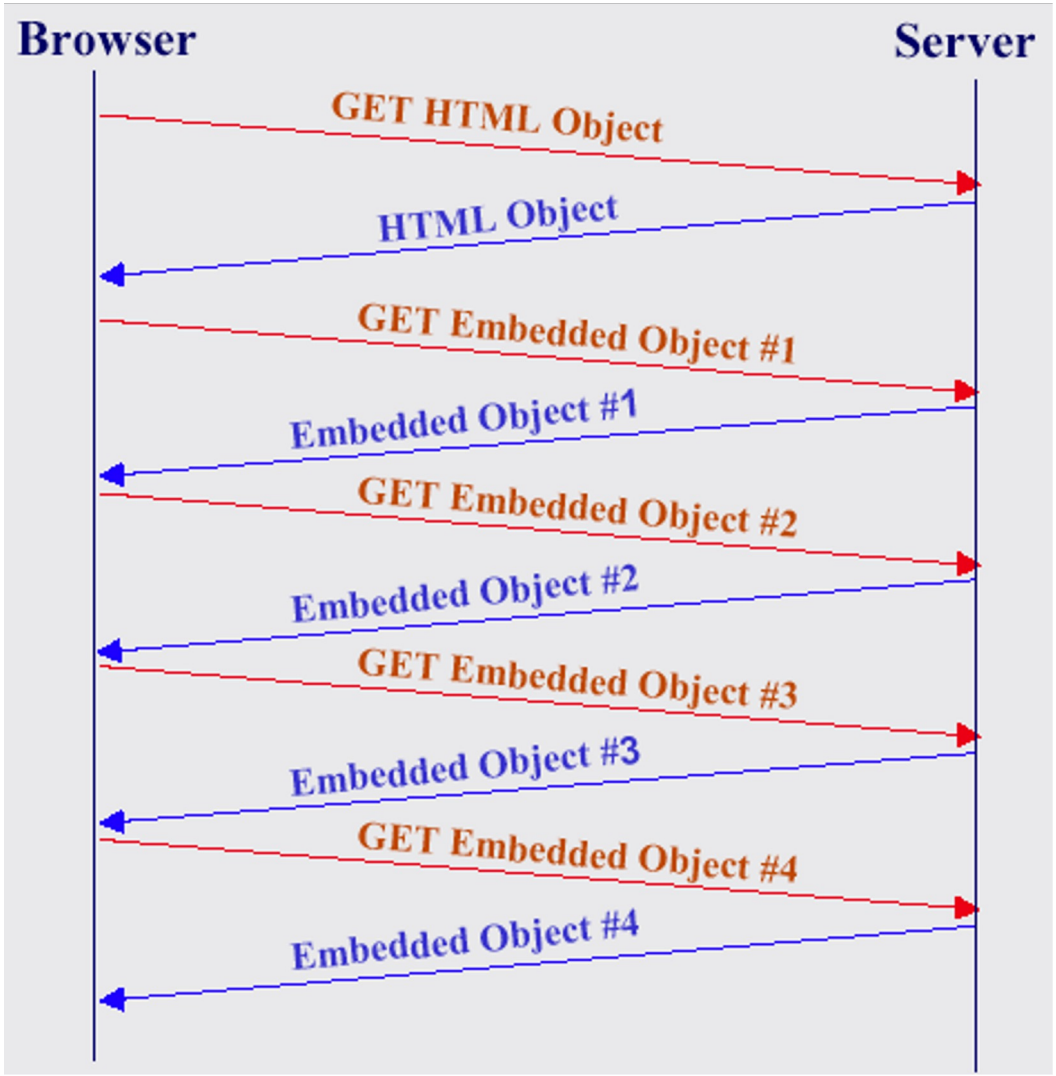




Application-Level solutions

- Using an Applications-Level gateway to "bookend" the pesky satellite link.
  - Quite similar to a split-connection
  - Canonical example: the web proxy
- HTTP 1.1 or persistent connections can improve performance for small transfers by keeping a connection alive which does not have to perform Slow Start for every web object.
- Browsing Over Satellite (problems)
  - Utilizes HTTP
  - For each object (HTML, Images, files etc.) a separate HTTP Get message is generated, this results in inefficient use of the satellite link
  - For each web page (URL – Uniform Resource Locator) at least one TCP connection is established, generating additional traffic
  - Each Get message adds a RTT (Round Trip Time)
  - A Get message will not be sent until the previous response has arrived (for most existing servers)

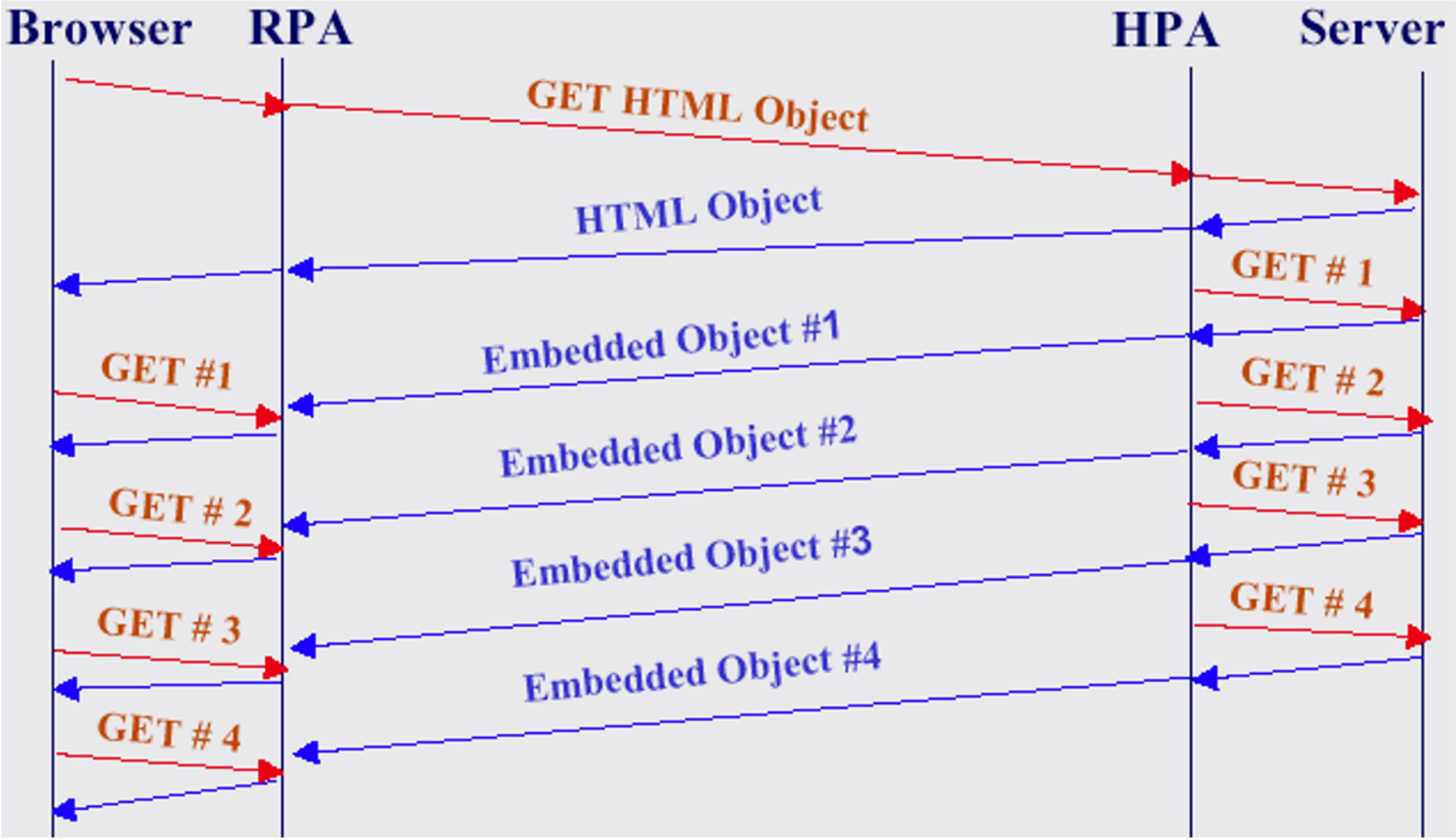
HTML Object plus 4 Embedded objects Get process



Internet Page Accelerator (IPA) - Solution

- Client – Server solution
- RPA (Remote Page Accelerator) at the VSAT and HPA (Hub Page Accelerator) at the Hub
- A single connection is established from each RPA to the HPA creating tunneling
- The Client S/W acts as the browser Proxy
- The HPA pre-fetching of embedded objects thus minimizes inbound traffic and improves user experience
- HTTP Get messages for the embedded objects are not transmitted through the satellite link
- RPA acts as the browser's proxy
- HPA pre-fetches the embedded objects
- Cookies handling
- Redirect URL handling
- Implementation:
  - RPA Client S/W runs over NT, Win 98
  - HPA Server runs over SUN Ultra 10
  - The HPA does not perform Proxy cache function

Browsing with IPA



Application level solutions: XFTP

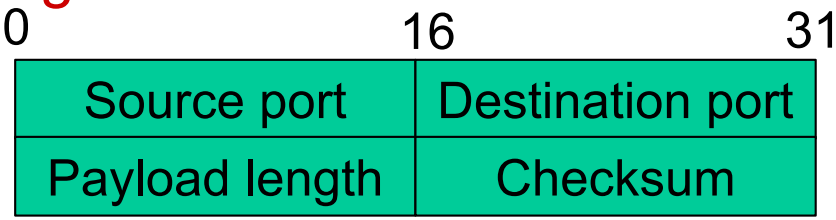
- XFTP uses multiple connections over the satellite
- Each connection shares the link
- Improves TCP performance
  - Reduces the optimal window size for each connection
  - Linearly speeds up Slow Start
- Too many connections creates instability
- Uses an adaptive algorithm to control the number of connections, and thereby the congestion on the link
  - When RTT increases, reduces the number of connections
  - When RTT decreases, increases the number of connections

Transport layer protocols: UDP

- UDP (User Datagram Protocol)
  - Real time application
  - Unreliable delivery
  - Connectionless

No major problems or specific issues over satellite networks but innovative protocols based on UDP will be addressed

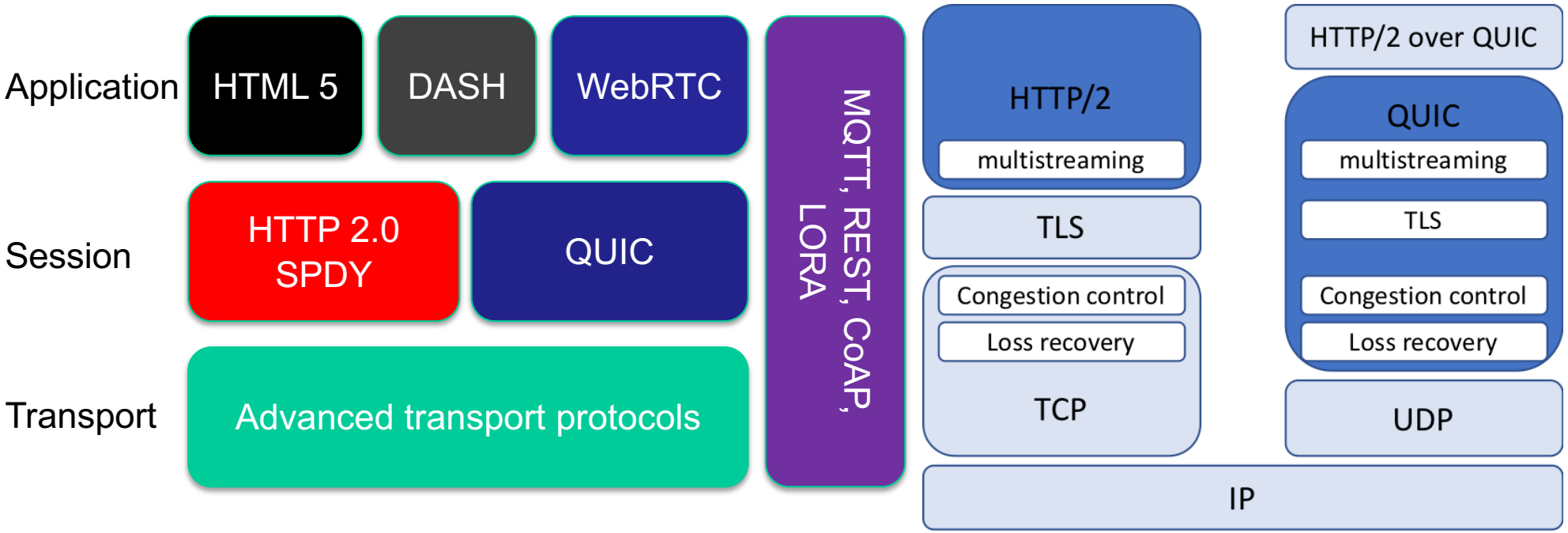
- Simple, connectionless, datagram
  - A peer is listening on a known port and the other peer sends packets to <other peer IP, port> without announcement
- Port numbers enable demultiplexing
  - 16 bits = 65535 possible ports
  - Port 0 is invalid
- Checksum for error detection
  - Detects (some) corrupted packets
  - Doesn't detect dropped, duplicated or reordered packets



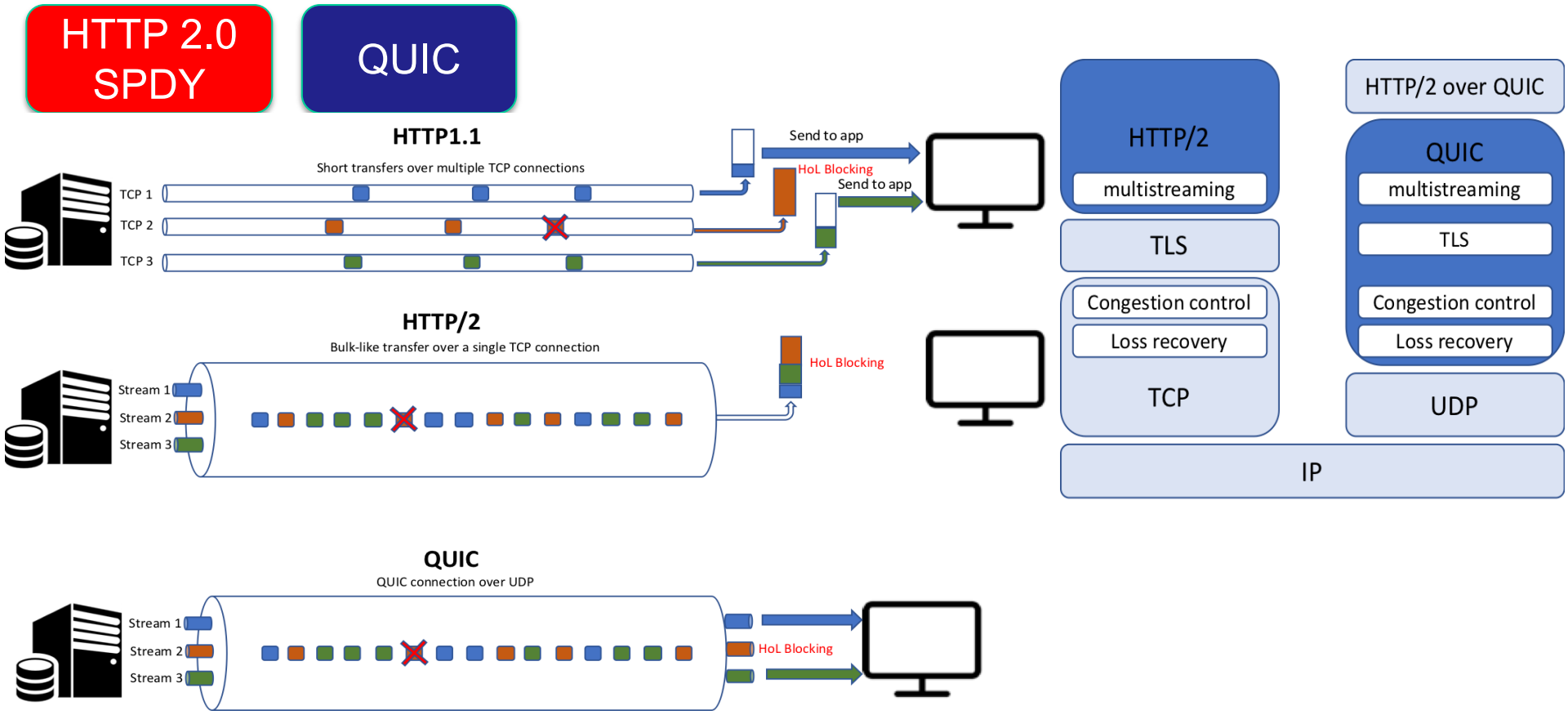
Rationale for UDP

- Invented after TCP
  - Not all applications can tolerate TCP
- Custom protocols can be built on top of UDP
  - To ensure reliability, strict ordering, flow control and congestion control
- Examples
  - Real-time media streaming (e.g. voice, video)
  - Facebook datacenter protocol

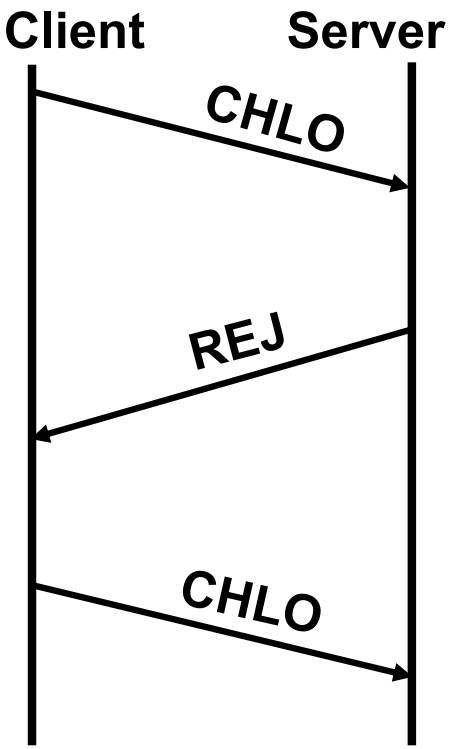
Innovative protocol architectures based on UDP



Future Internet protocols SotA and beyond



QUIC Connection set up

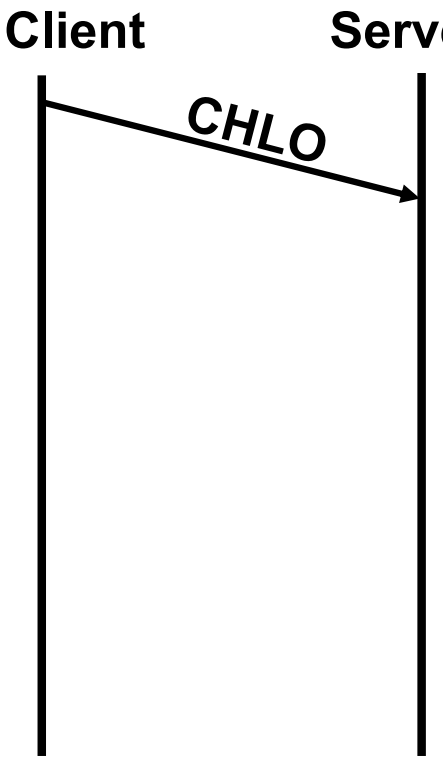


The client sends a first client hello (CHLO).

The server answers with a Rejection. This message contains all the useful information on the server to set up the connection.

The client sends a new CHLO. The connection is set up and the packets can start to be transferred.

The advantage of QUIC is that it is not necessary to set up again the connection through the mechanism CHLO-REJ-CHLO if the client connects to a server already connected in the past.



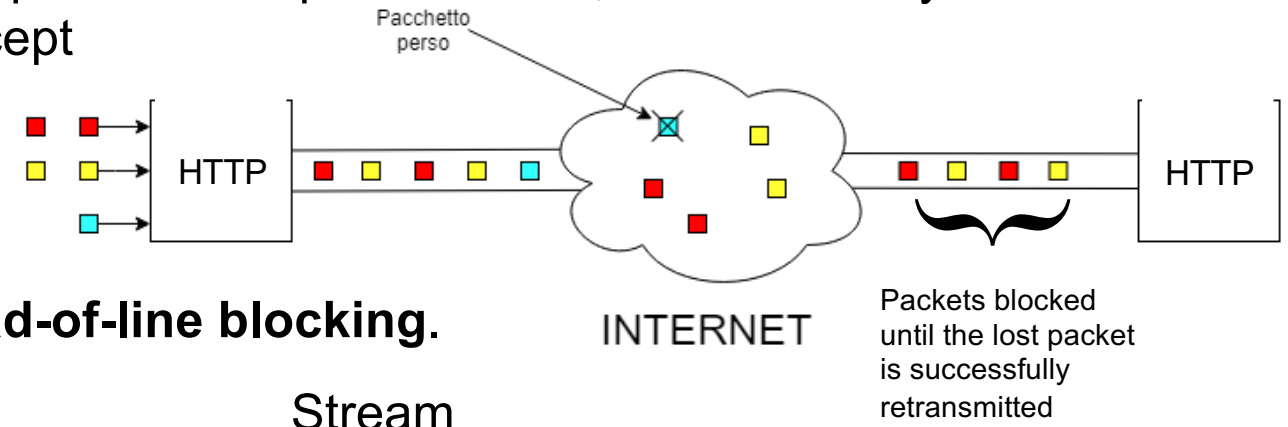
In this case the client has already the information necessary to set up the connection.

0-RTT obtained

Multiplexing

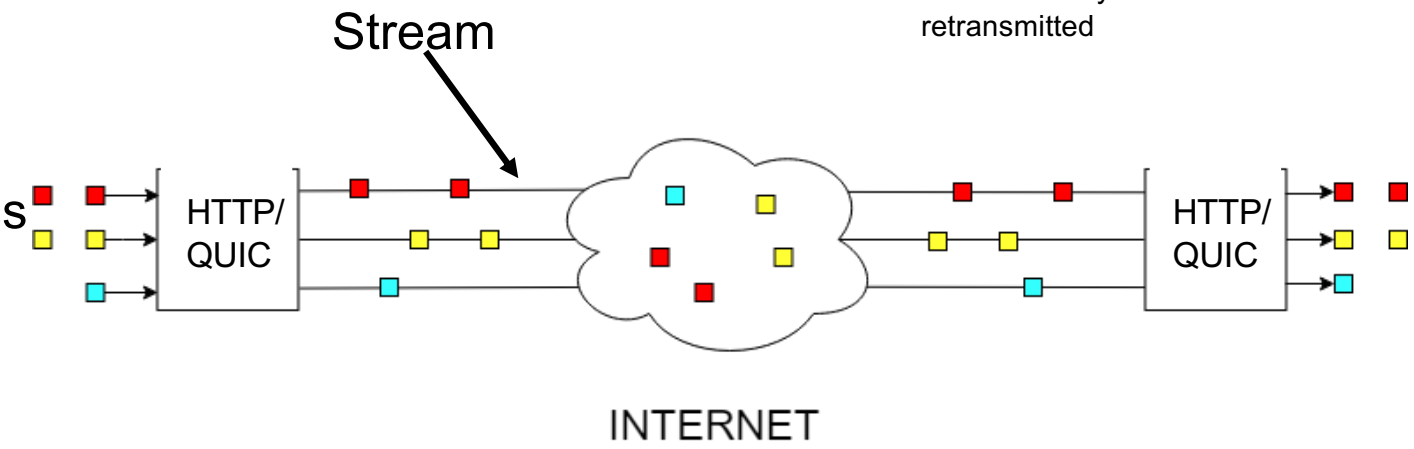
QUIC is a multiplexing oriented protocol. Data transfers occur through links named **Streams**.

TCP accepts only ordered packets. If a packet is lost, it is necessary to wait for their retransmission to accept the following packets.



This problem is called **head-of-line blocking**.

Every Stream is **independent** from the other one. Thus, the loss of one packet on a certain stream affects just that stream.



Survival to IP address change

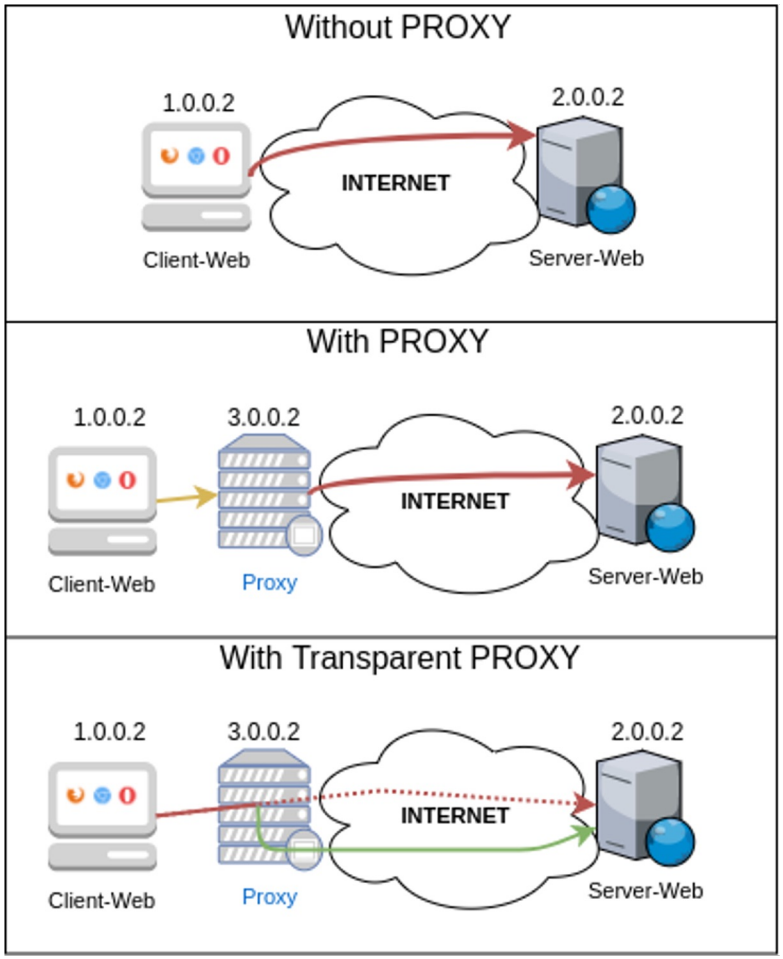
TCP connections don't survive to IP address change or port change.

Every QUIC connection is characterized by a Connection-ID.

The Connection-ID doesn't change for all the connection duration, allowing to survive to the IP address change.

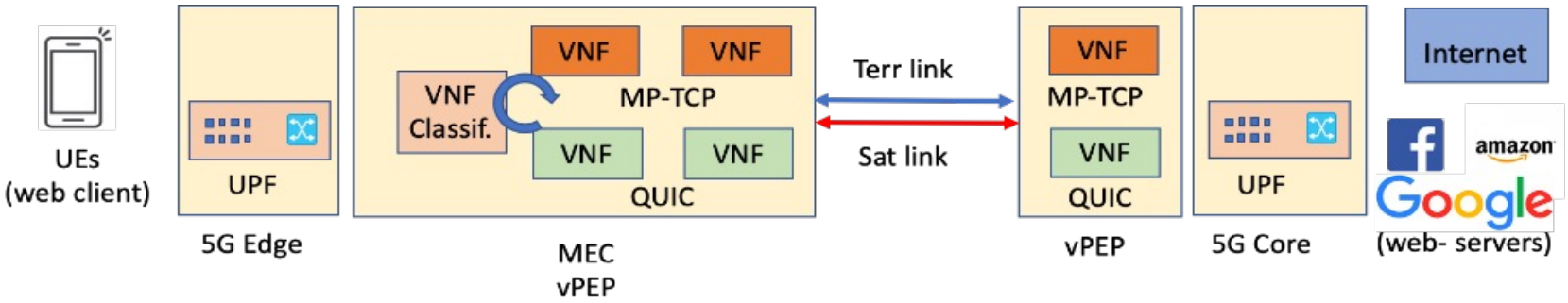
Proxy: basically how it works

- Normally Client-Web establishes a connection directly to the end-server
- The end-server will answer directly to the client
- If the client wants to use a proxy (**aware**) it sends an explicit request to the proxy to reach the end-server
- The end-server will answer to the proxy, then the proxy will report to the client
- With transparent proxy the client is **unaware** of the actions of the proxy. **With this approach QUIC is allowed to be inter-changeable with the MP-TCP proxy.**



Innovative architectures based on QUIC and MPTCP

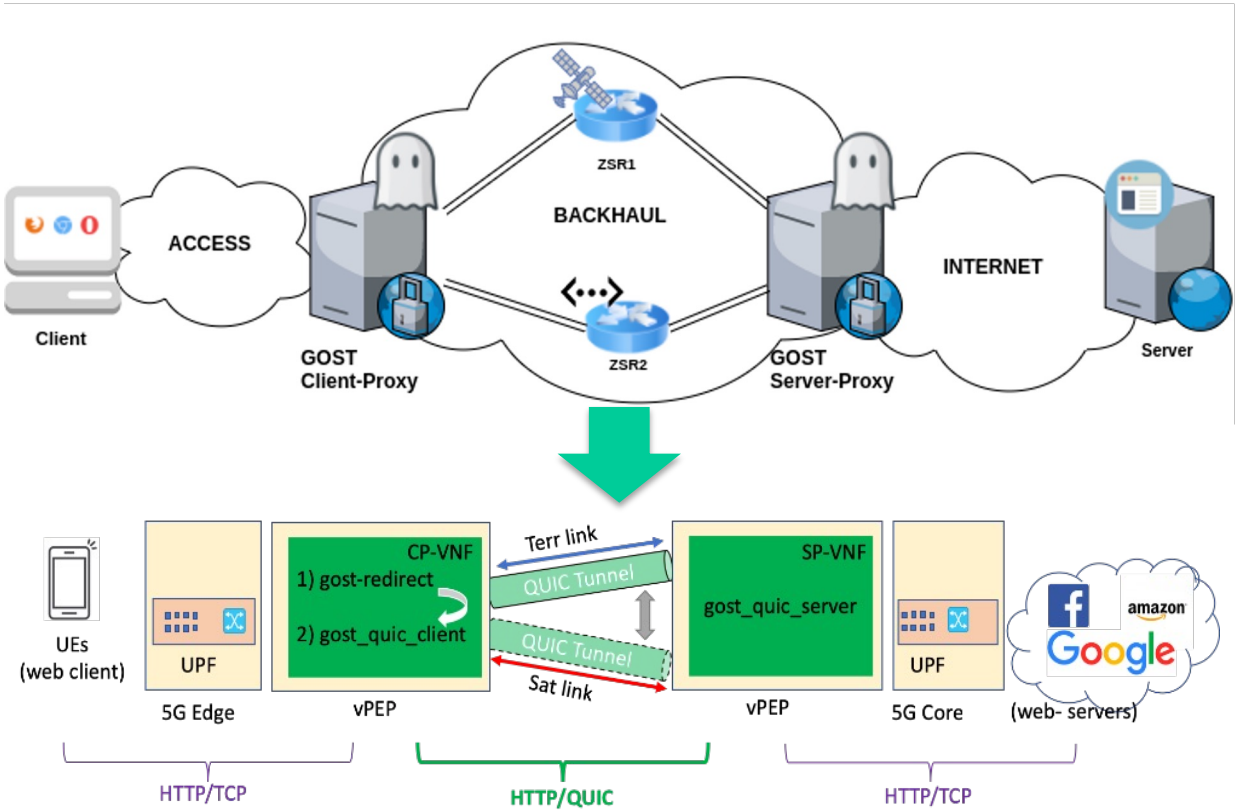
- It is possible to set up two different possible chains:
  - QUIC
  - MP-TCP



– Flow classification VNF is introduced
– Two sets of proxy-enabled chains must be defined

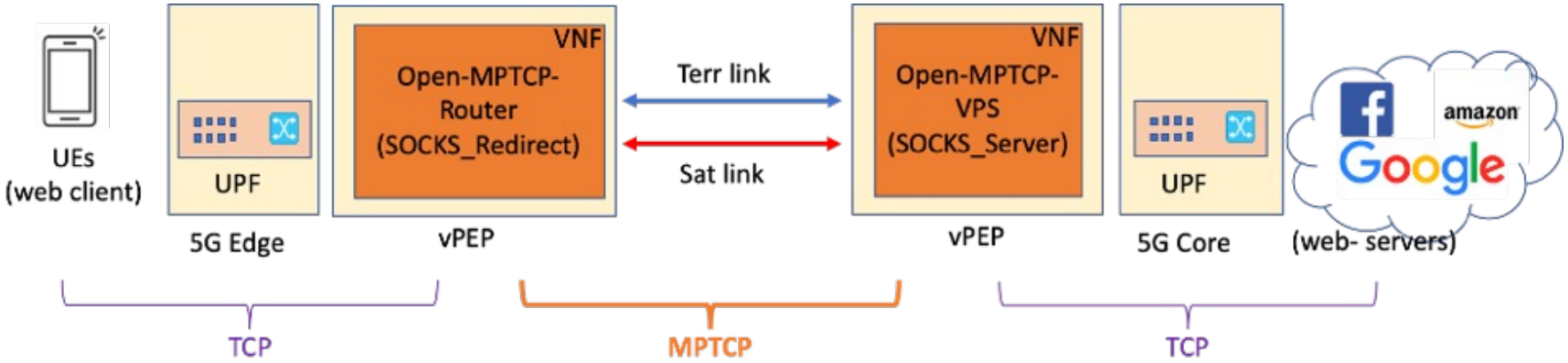
QUIC proxy architecture

- GOST (**GO S**imple **T**unnel) application runs over two machines
- A QUIC tunnel is established between these two machines
- Two paths are available in the backhaul network
- Client access to internet through a transparent proxy



MPTCP transparent proxy VNF

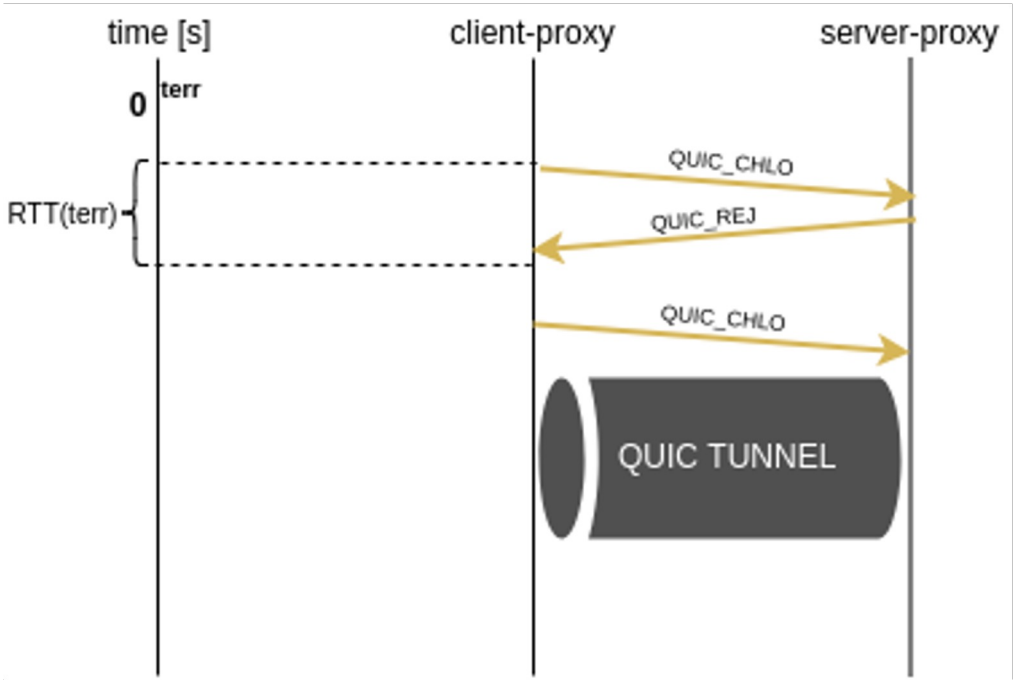
➢ MPTCP VNF implementation



- It has two main components;
  - MPTCP-Router (socks_redir) to transparently redirect TCP traffic to MPTCP
  - MPTCP-VPS (SOCKS5) as a reverse proxy to re-translate MPTCP sub-flows into TCP

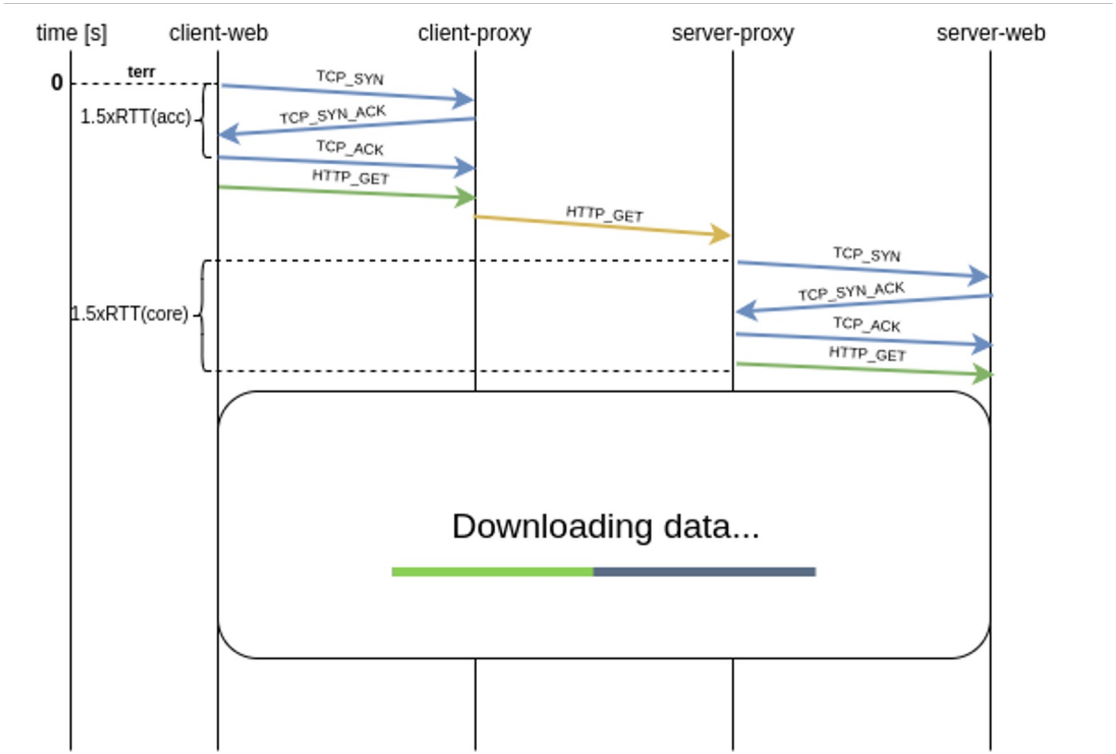
QUIC proxy initialization

- At least 1 WAN RTT
- One extra RTT if the client-proxy is not able to trust server-proxy (certificates issue)
- After the initialization, 0-RTT features can be exploited
- Keep-alive messages don't destroy the QUIC tunnel



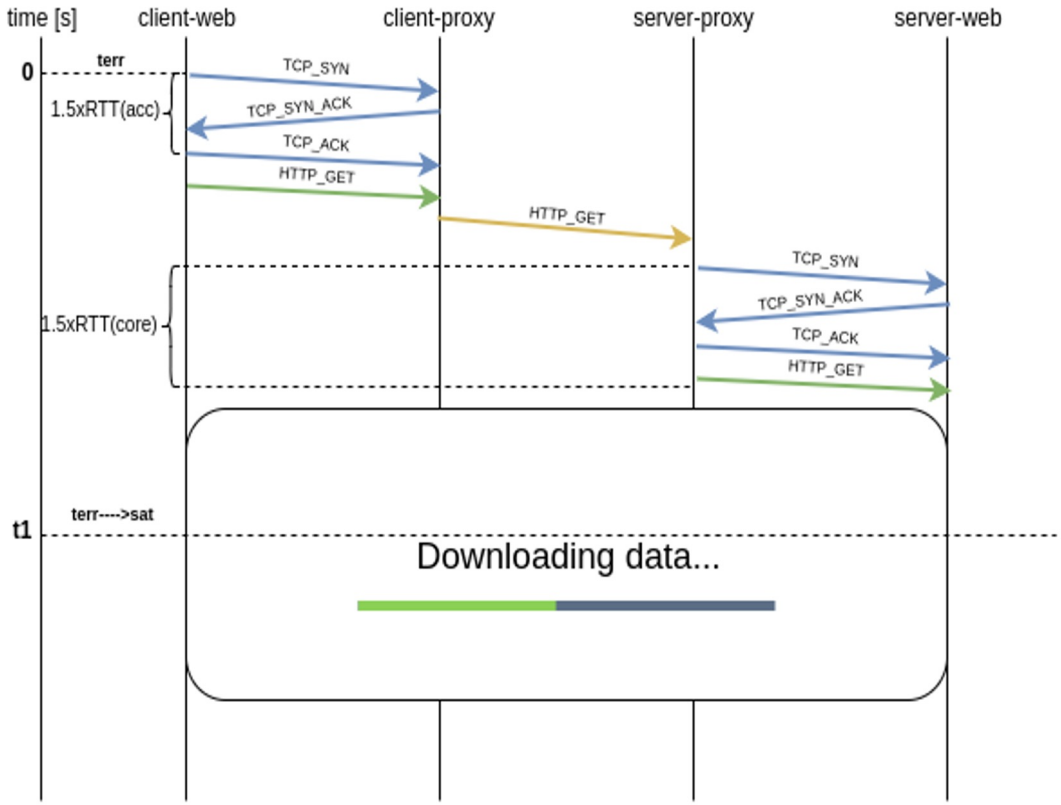
QUIC proxy 0-RTT

- After initialization on the QUIC tunnel 0-RTT is ready to use
- Signalling time is due only to TCP handshake at the edges of the communication



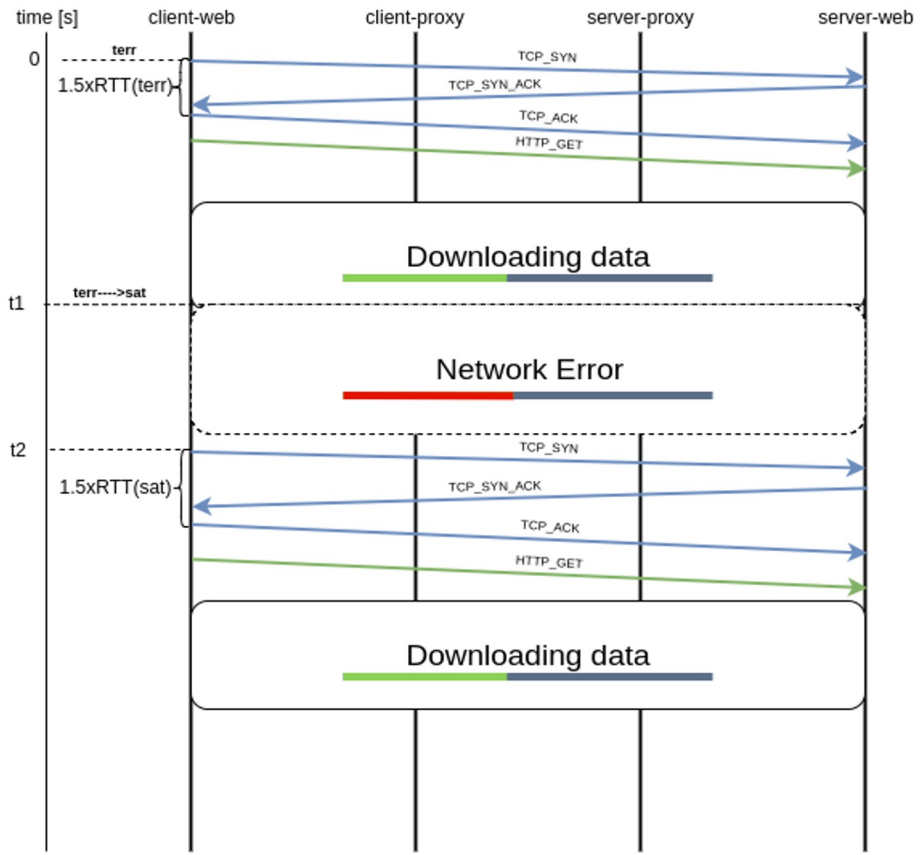
QUIC proxy Connection Migration

- Connection switching (orchestrator-driven) feature allows the client to change IP address without closing the connection
- Switching from terrestrial to satellite path involves only a changing of the network parameters



TCP vs QUIC proxy

- Using TCP end-to-end a changing of the IP address experiences a Network Error
- After a Network Error a new TCP handshake is needed



TCP vs QUIC improvements

- Downloading a 20 MBytes file
- Switching the path (terrestrial to satellite) at random time during the download