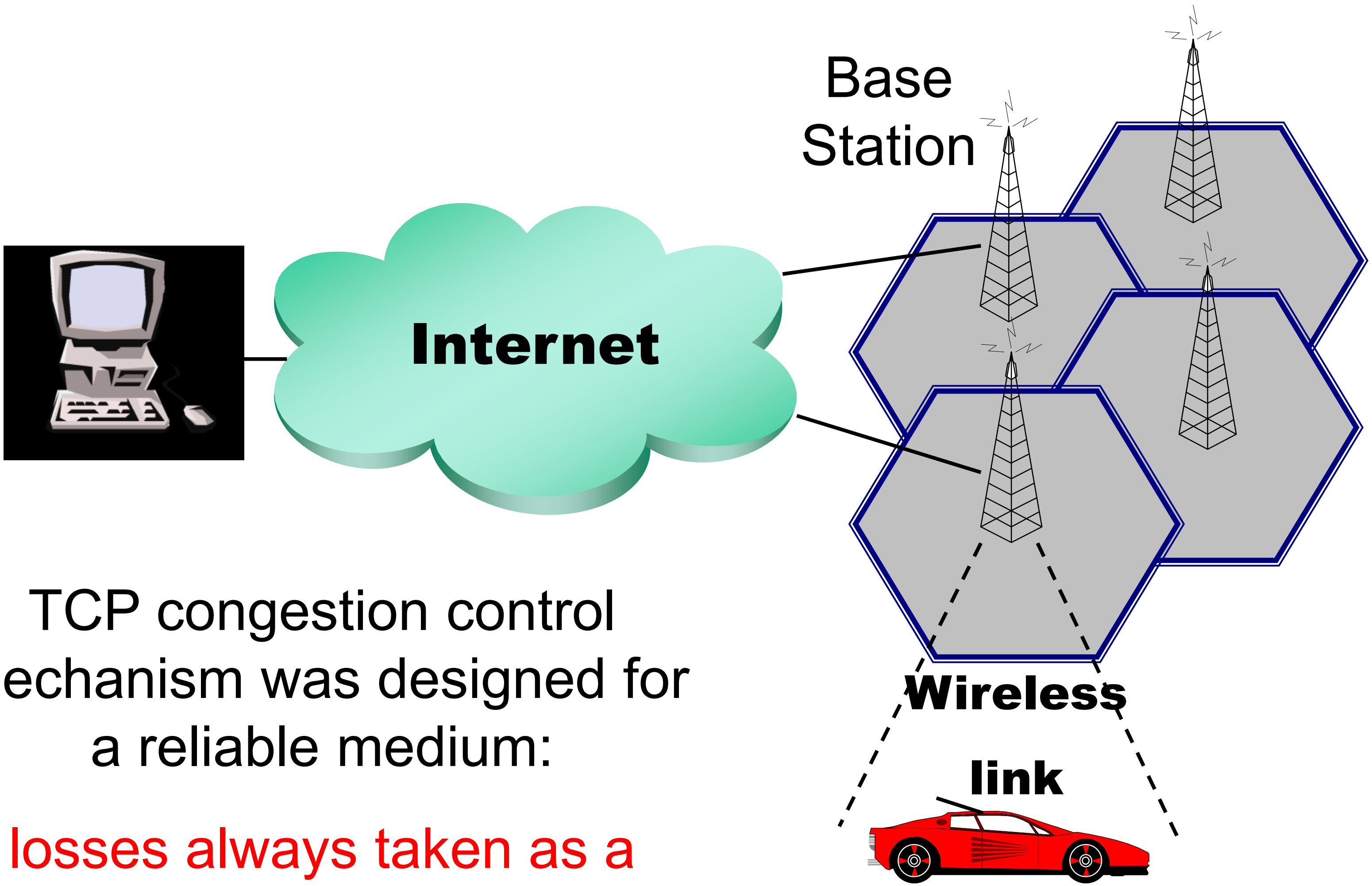
Wireless Networks for Mobile Applications

Prof. Claudio Palazzi
cpalazzi@math.unipd.it

Università degli Studi di Padova

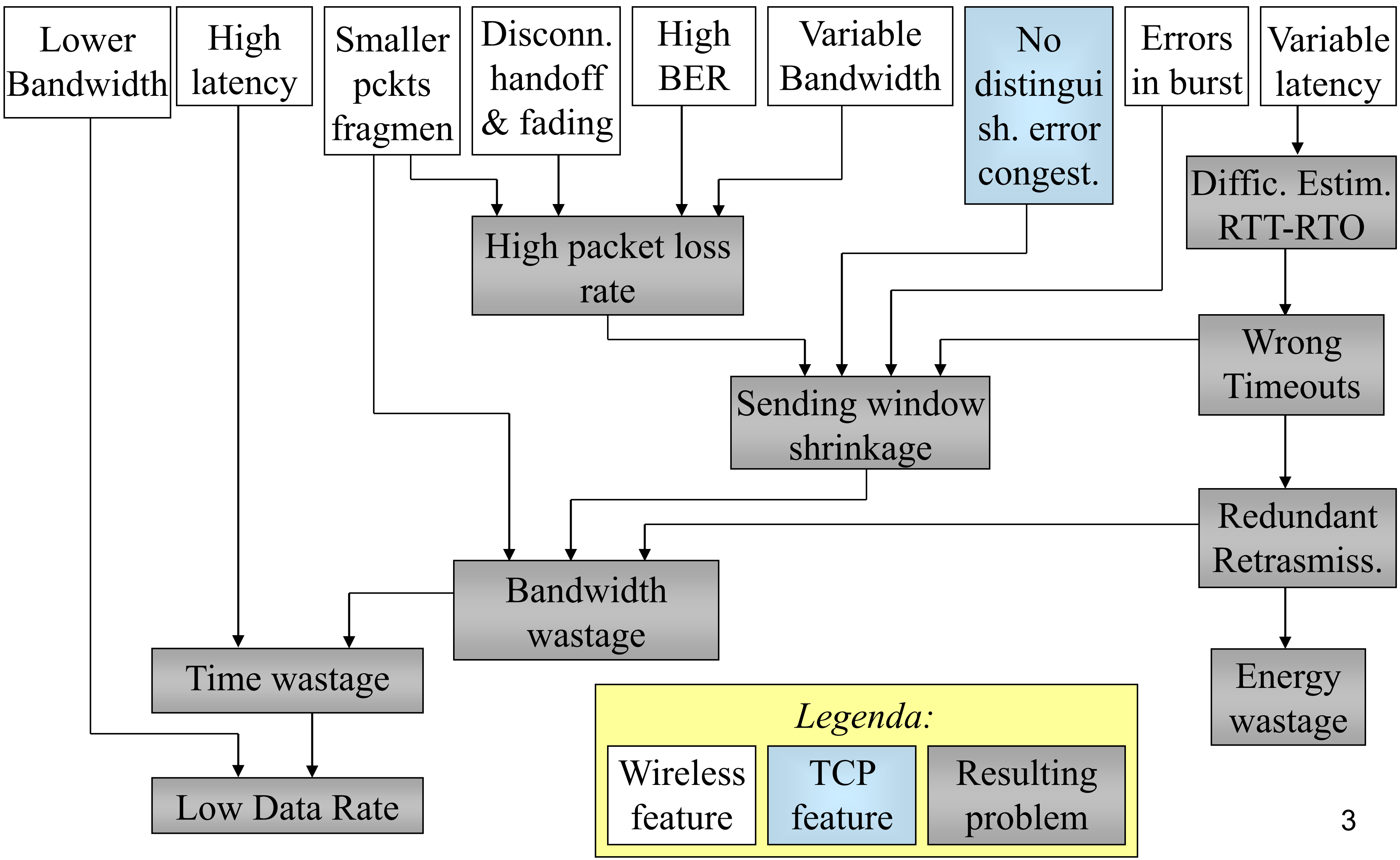
TCP over Wireless?



TCP congestion control mechanism was designed for a reliable medium:

losses always taken as a congestion sign

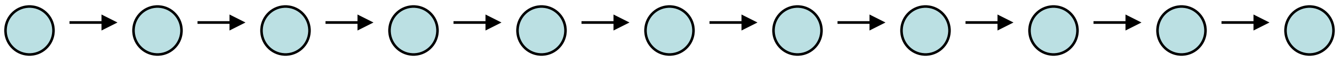
TCP-Wireless Interaction

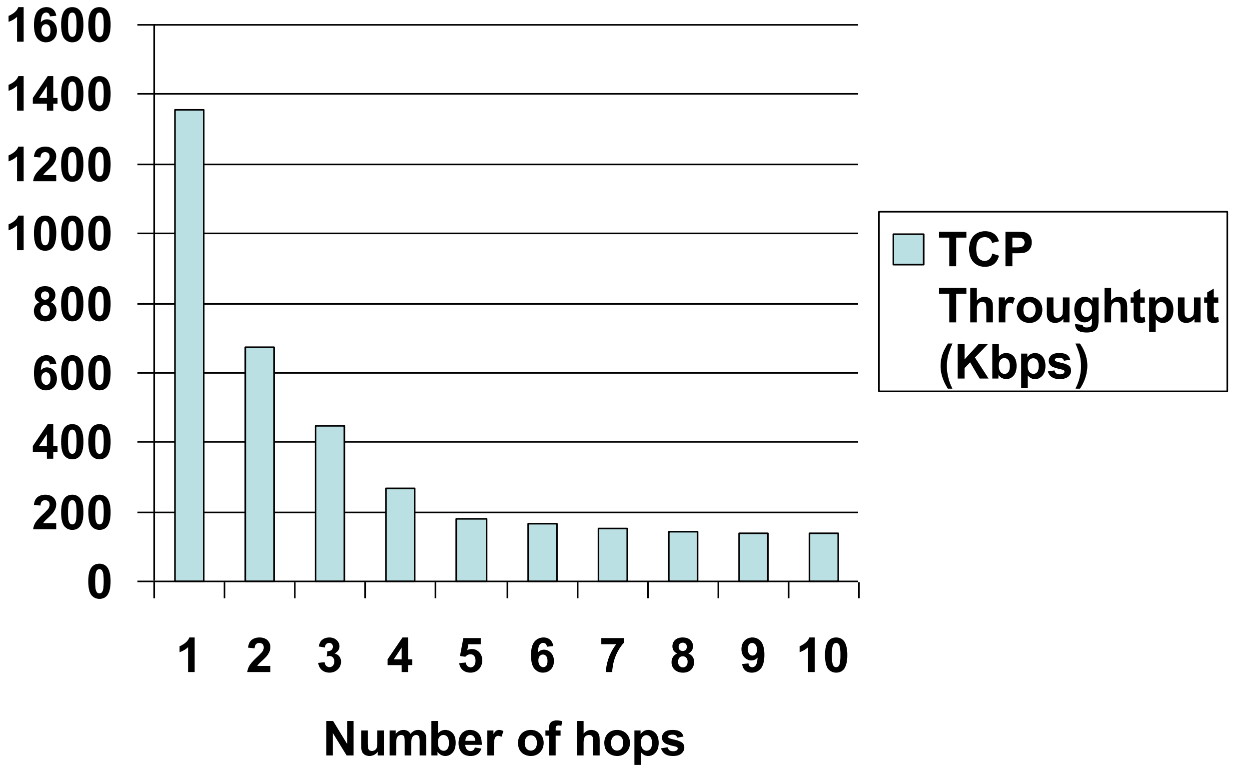


Wireless Problems: Impact on TCP

- Error losses
  - TCP assumes congestion and reduces cwnd
- Losses in bursts
  - Multiple cwnd reductions
- Long delays (satellites)
  - RTT-unfairness
- Variable delays
  - Wrong RTO computation
- Disconnections
  - Multiple timeouts
- Variable bandwidth
  - Sudden loss bursts or bandwidth wastage

Impact of Multi-Hop Wireless Paths (e.g., in MANETs)



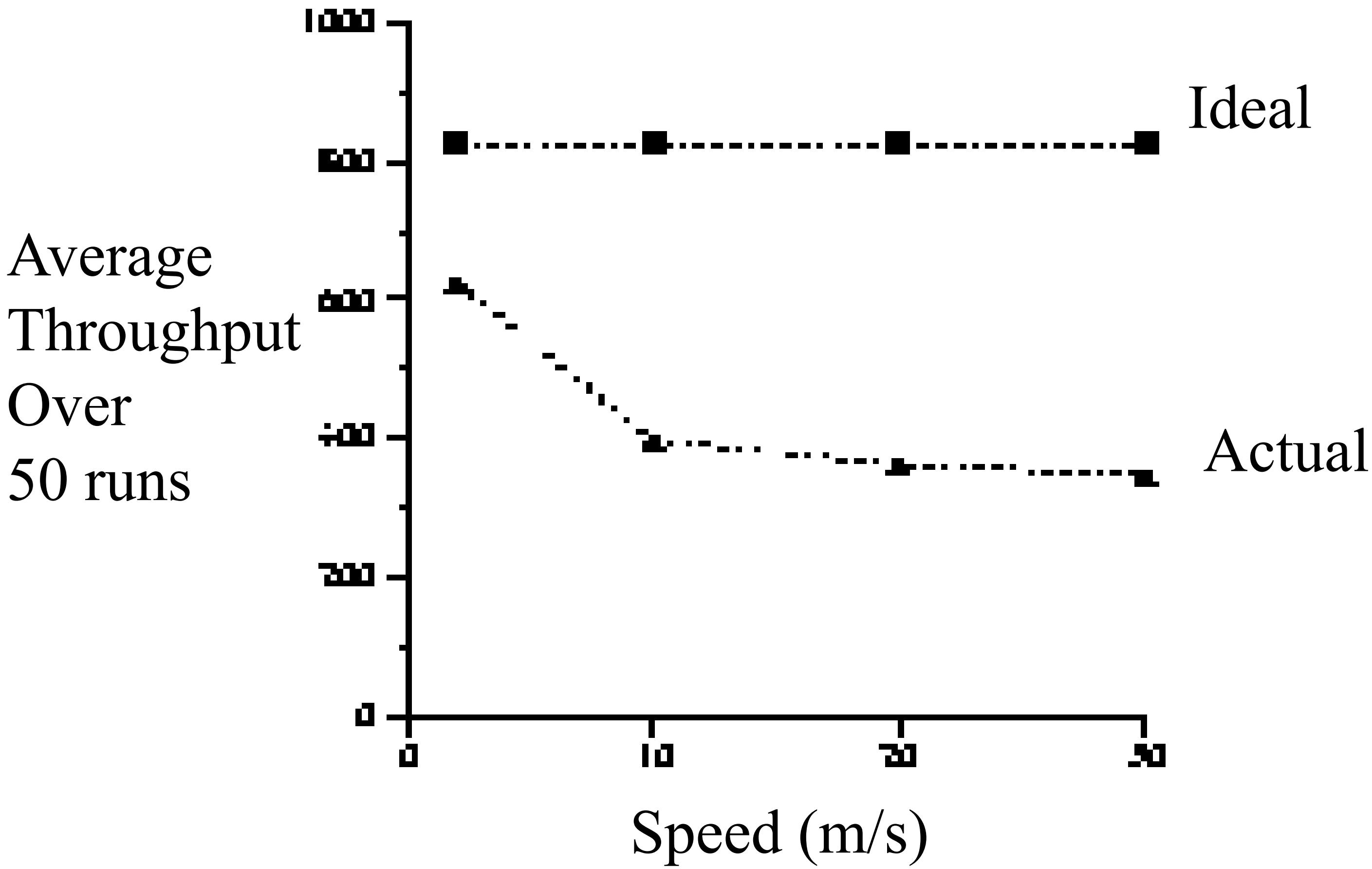


TCP Throughput using 2 Mbps 802.11 MAC

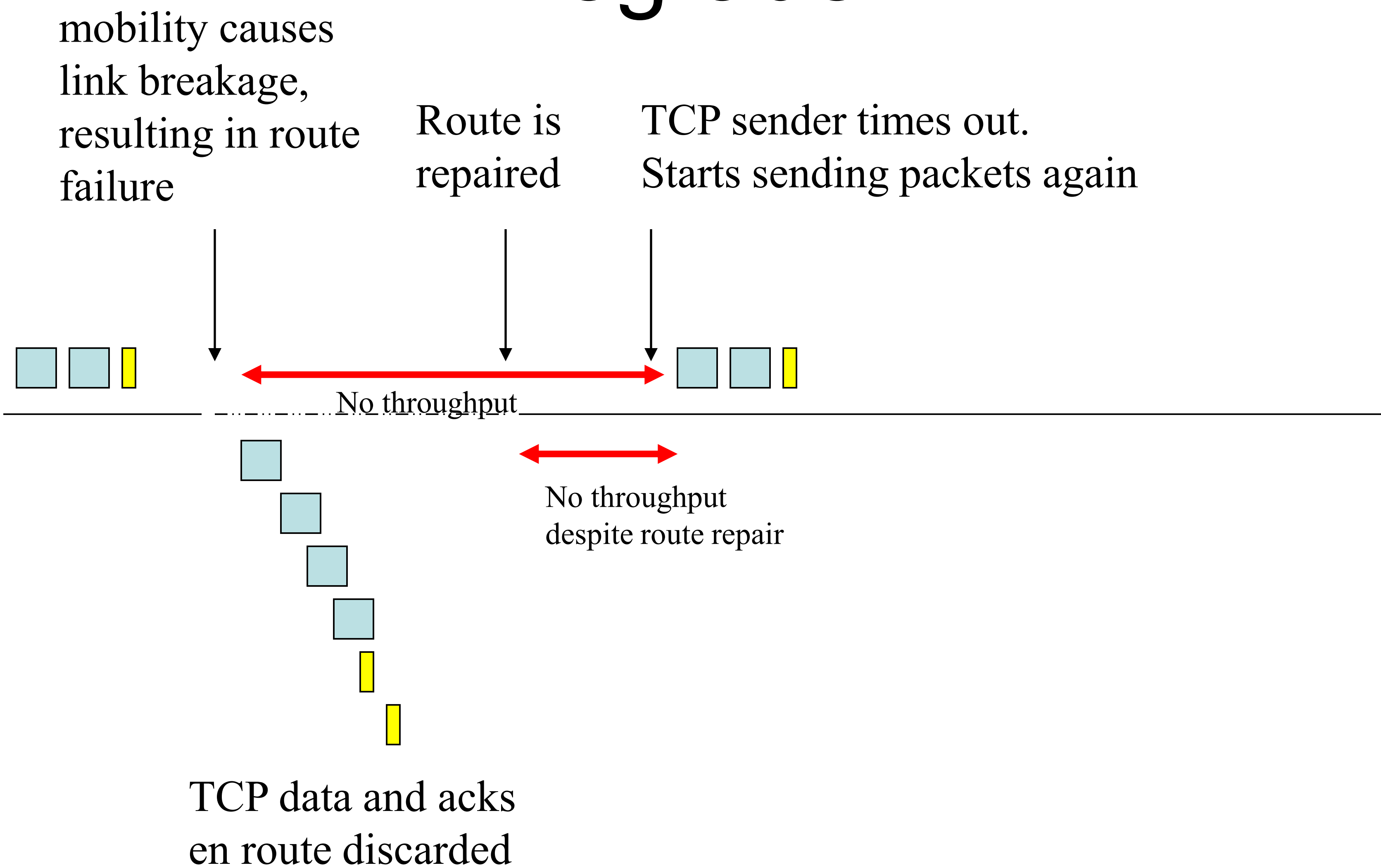
Throughput Degradations with Increasing Number of Hops

- Packet transmission can occur on at most one hop among three consecutive hops
- Increasing the number of hops from 1 to 2, 3 results in increased delay, and decreased throughput
- Increasing number of hops beyond 3 allows simultaneous transmissions on more than one link, however, degradation continues due to contention between TCP Data and Acks traveling in opposite directions
- When number of hops is large enough, the throughput stabilizes due to *effective pipelining*

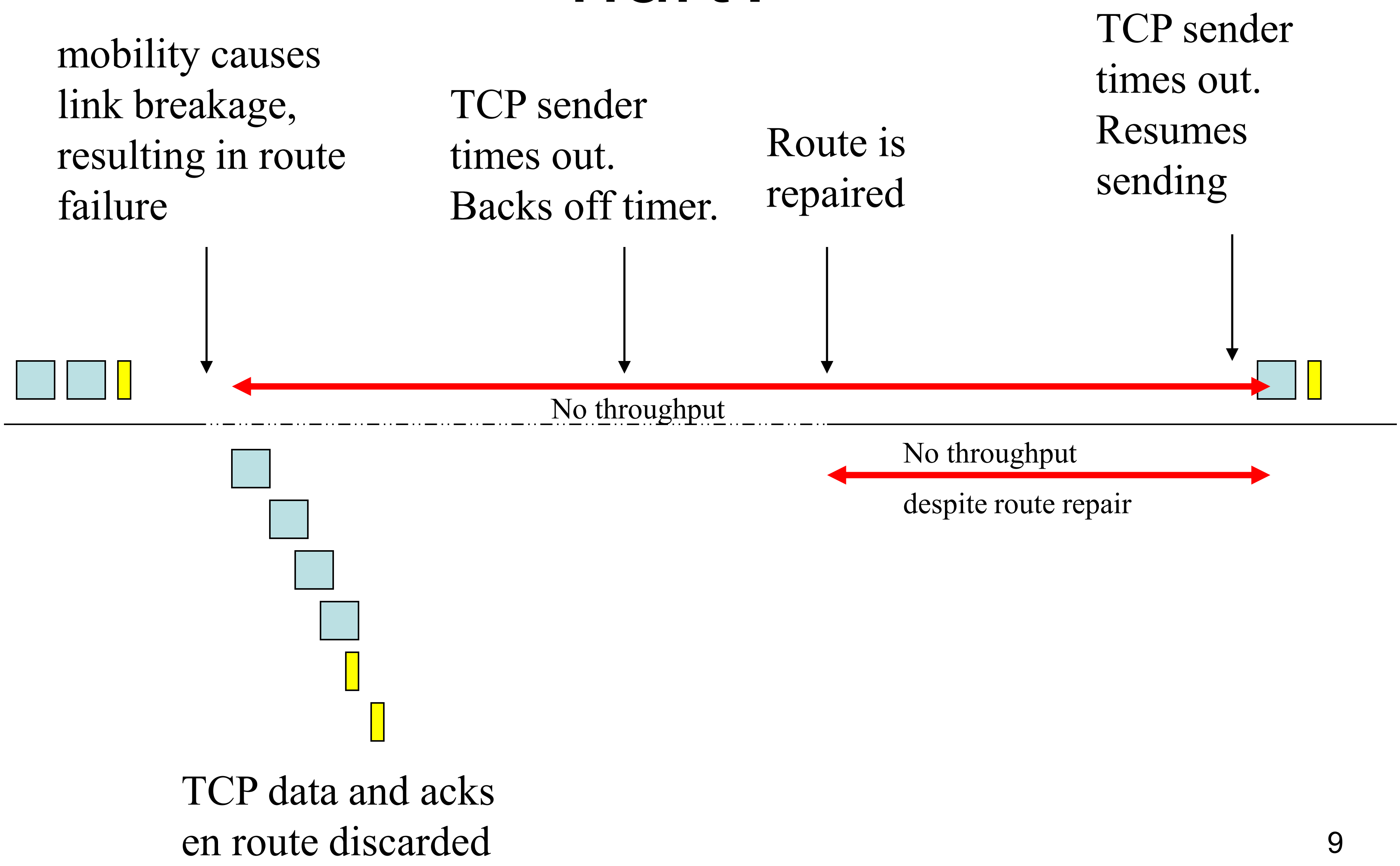
Mobility: Throughput generally degrades with increasing speed



Why Does Throughput Degrade?



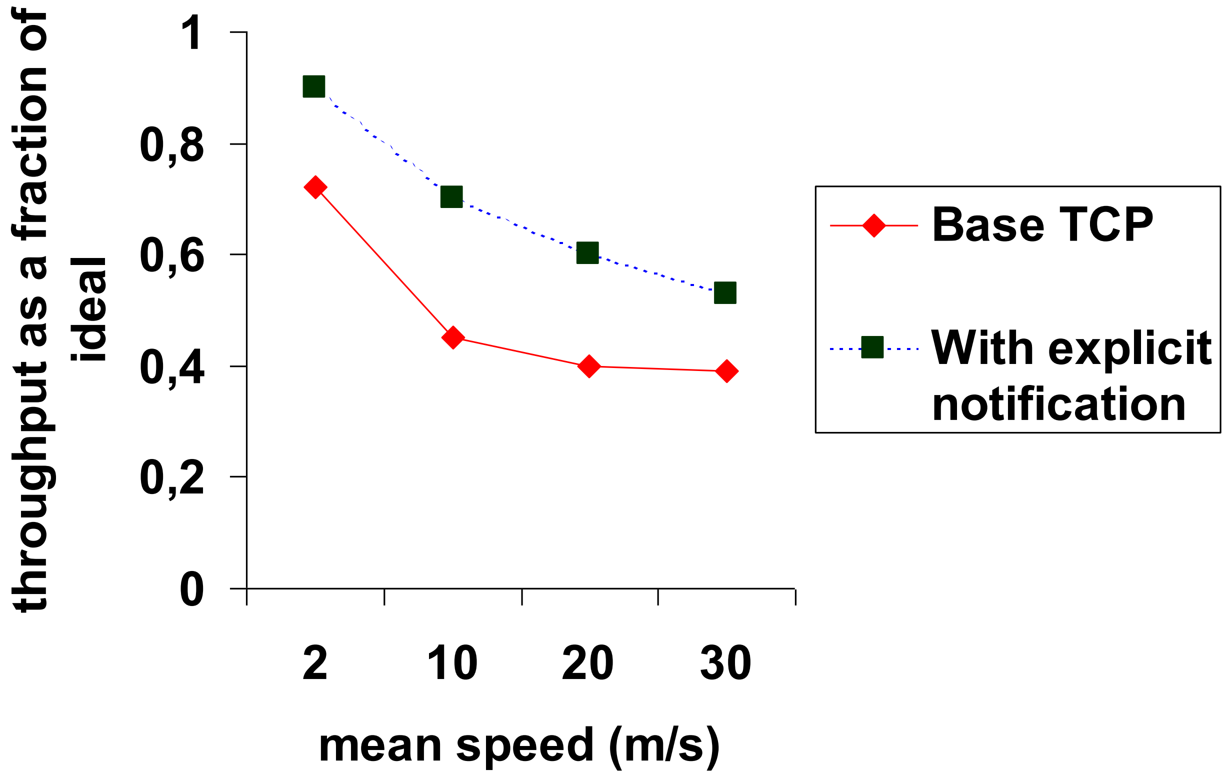
Why Does Repair Latency hurt?



How to Improve Throughput (Bring Closer to Ideal)

- Network feedback
- Inform TCP of route failure by explicit message
- Let TCP know when route is repaired
  - Probing (eg, persistent pkt retransmissions)
  - Explicit link repair notification
- Alleviates repeated TCP timeouts and backoff

Performance with Explicit Notification



Transport layer solutions: a taxonomy

- Connection split:
  - Local retransmissions
  - Quick actions on the wireless link
  - TCP specific for wireless link
- Pure End-to-End:
  - New transport protocol
  - Sender is aware of wireless link
  - End-to-End paradigm preservation

Transport Protocols

Traditional TCP:

- TCP Reno
- TCP New Reno
- TCP Vegas
- TCP Sack

Connection split:

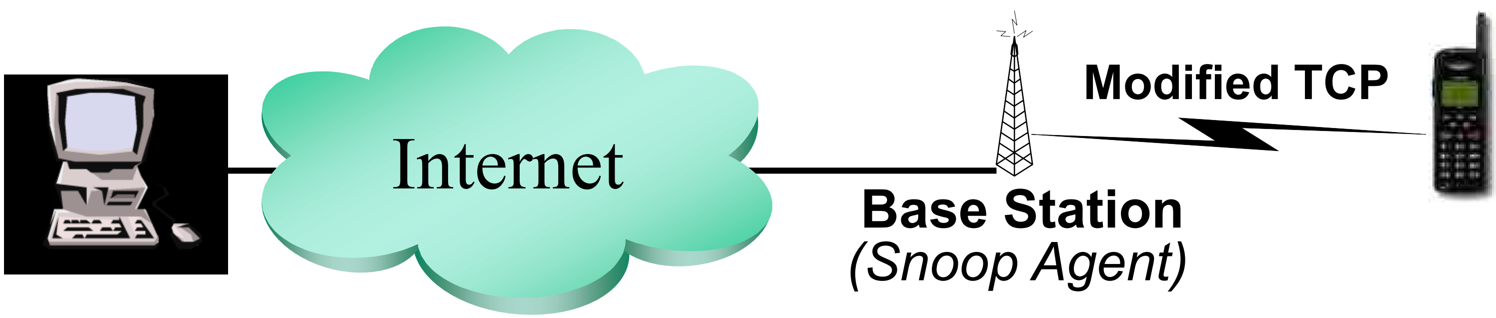
- I-TCP
- M-TCP
- Snoop Protocol
- Proxy

Pure End-to-End:

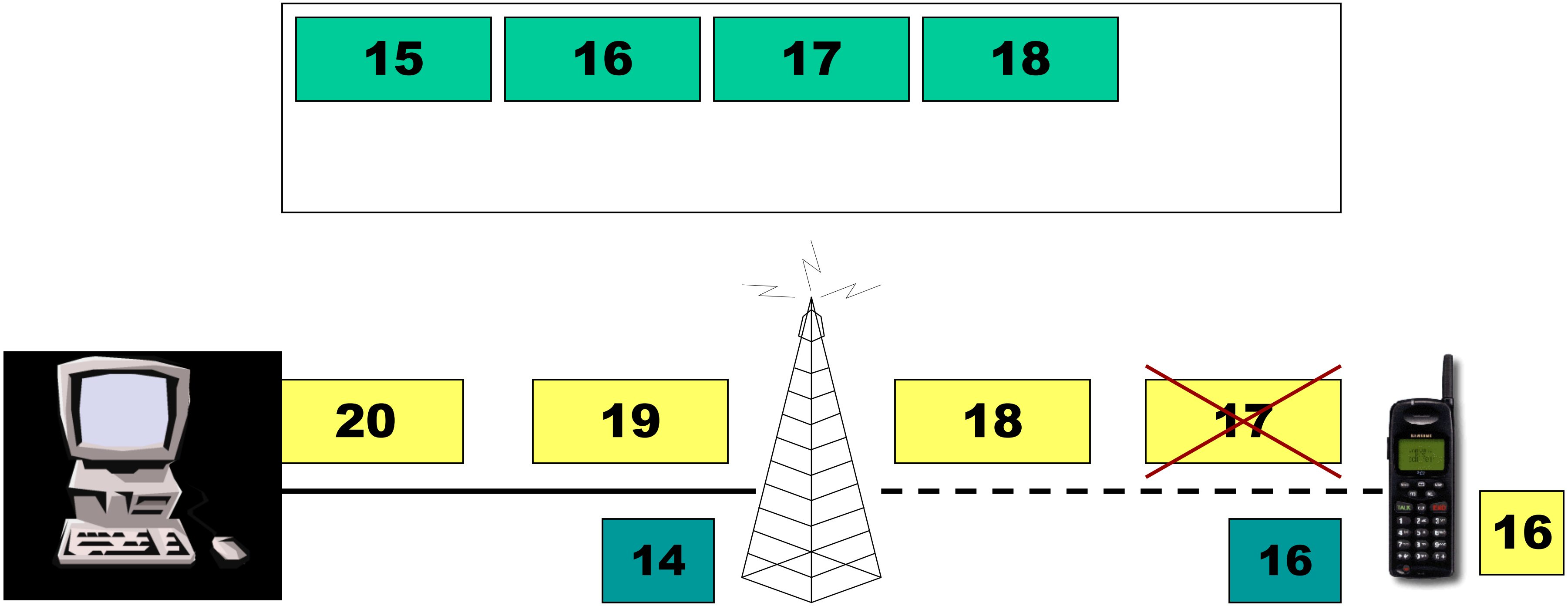
- Delayed Dupacks
- TCP-Aware
- Freeze-TCP
- TCP Probing
- WTCP
- TCP Westwood
- TCP Hybla
- TCP CUBIC
- TCP High Speed
- TCP Compound
- TCP Fast
- …

Snoop Protocol (Balakrishnan et al., 1995)

- Designed to address high BER
- The Base Station implements a *Snoop Agent*:
  - Monitoring of all packets in transit in both directions
  - All packets not yet acked are cached on the base station:
    - Local retransmission s of lost data
    - Dupack filtering to hide losses to the sender (otherwise it would perform redundant retransmissions and shrinkage of the congestion window)

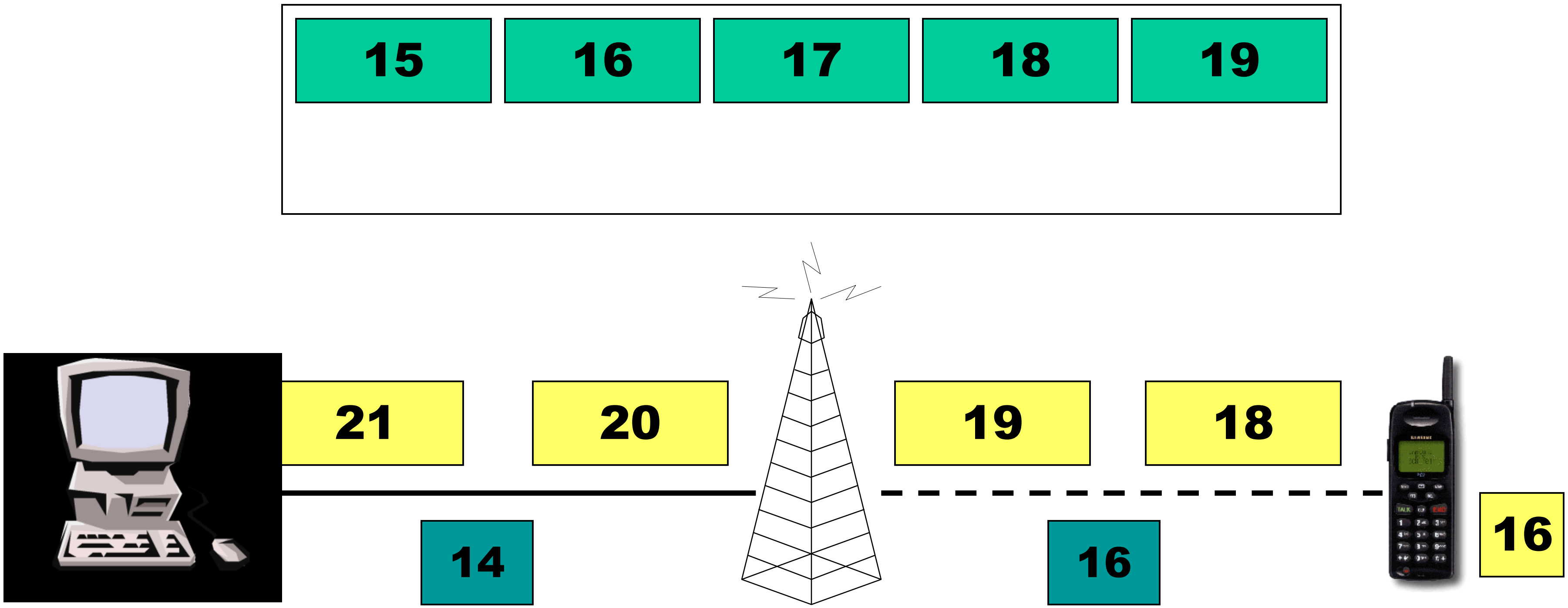


Snoop Protocol – Example (1/9)



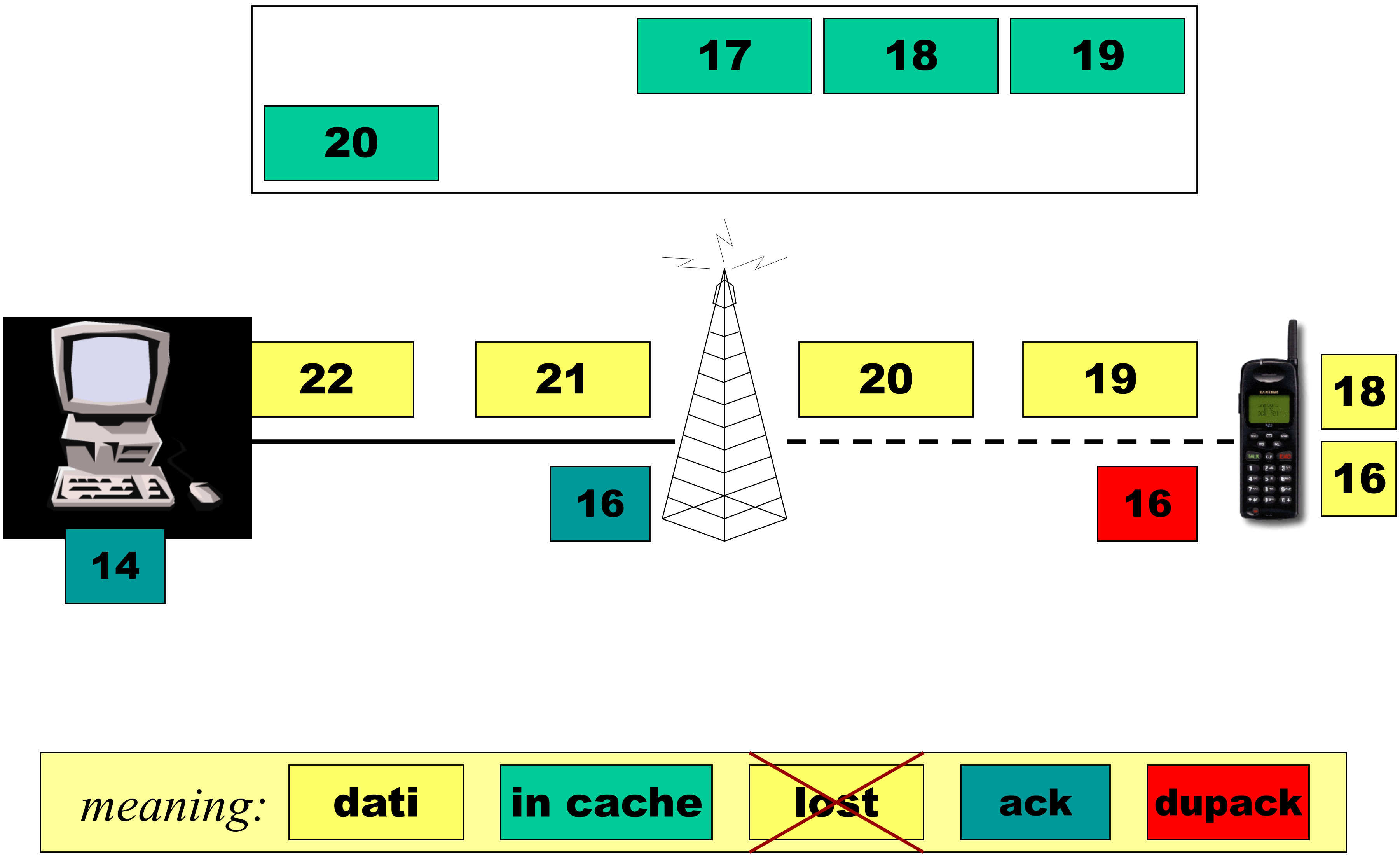
meaning: dati | in cache | lost | ack | dupack

Snoop Protocol – Example (2/9)

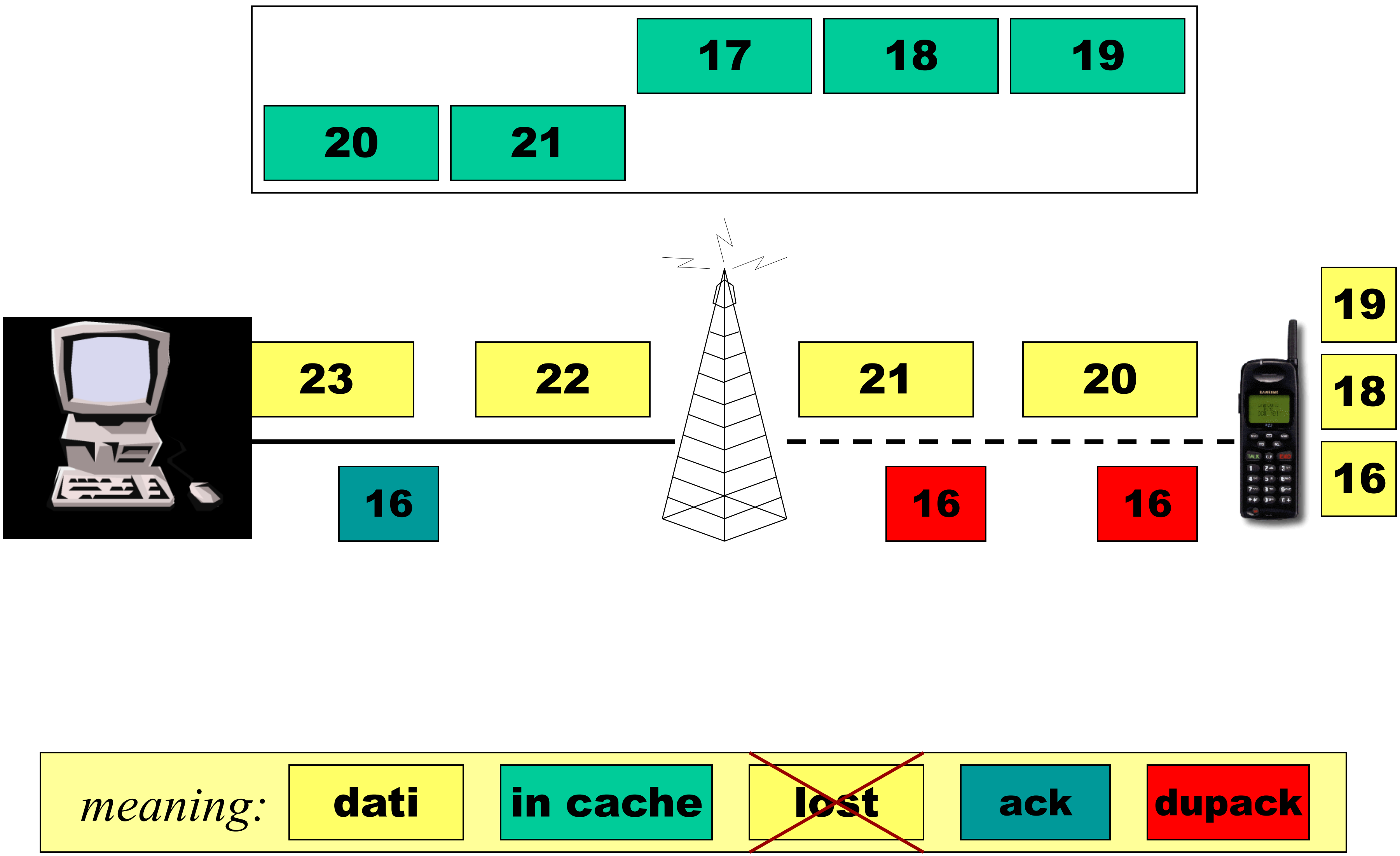


meaning: dati | in cache | ~~lost~~ | ack | dupack

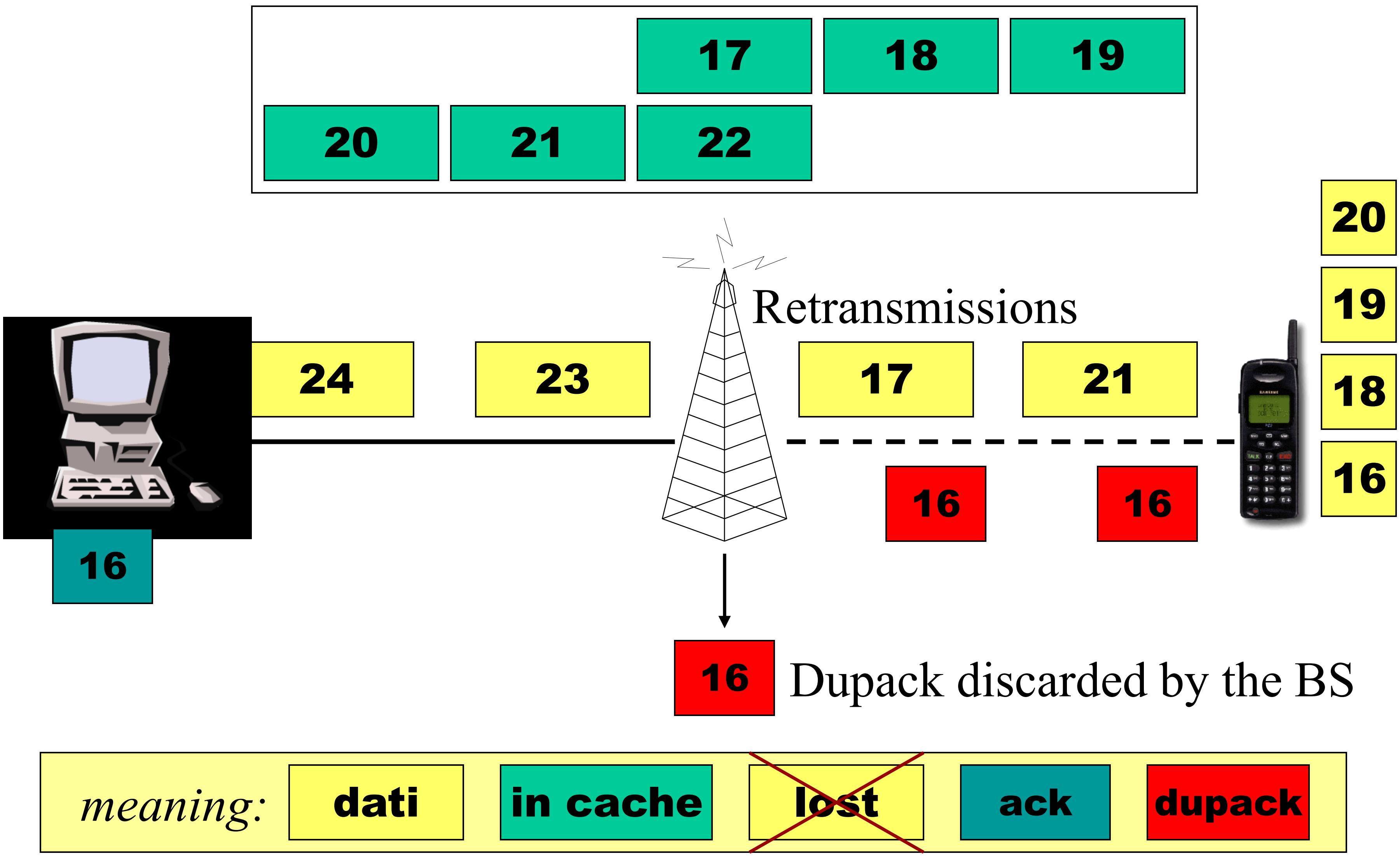
Snoop Protocol – Example (3/9)



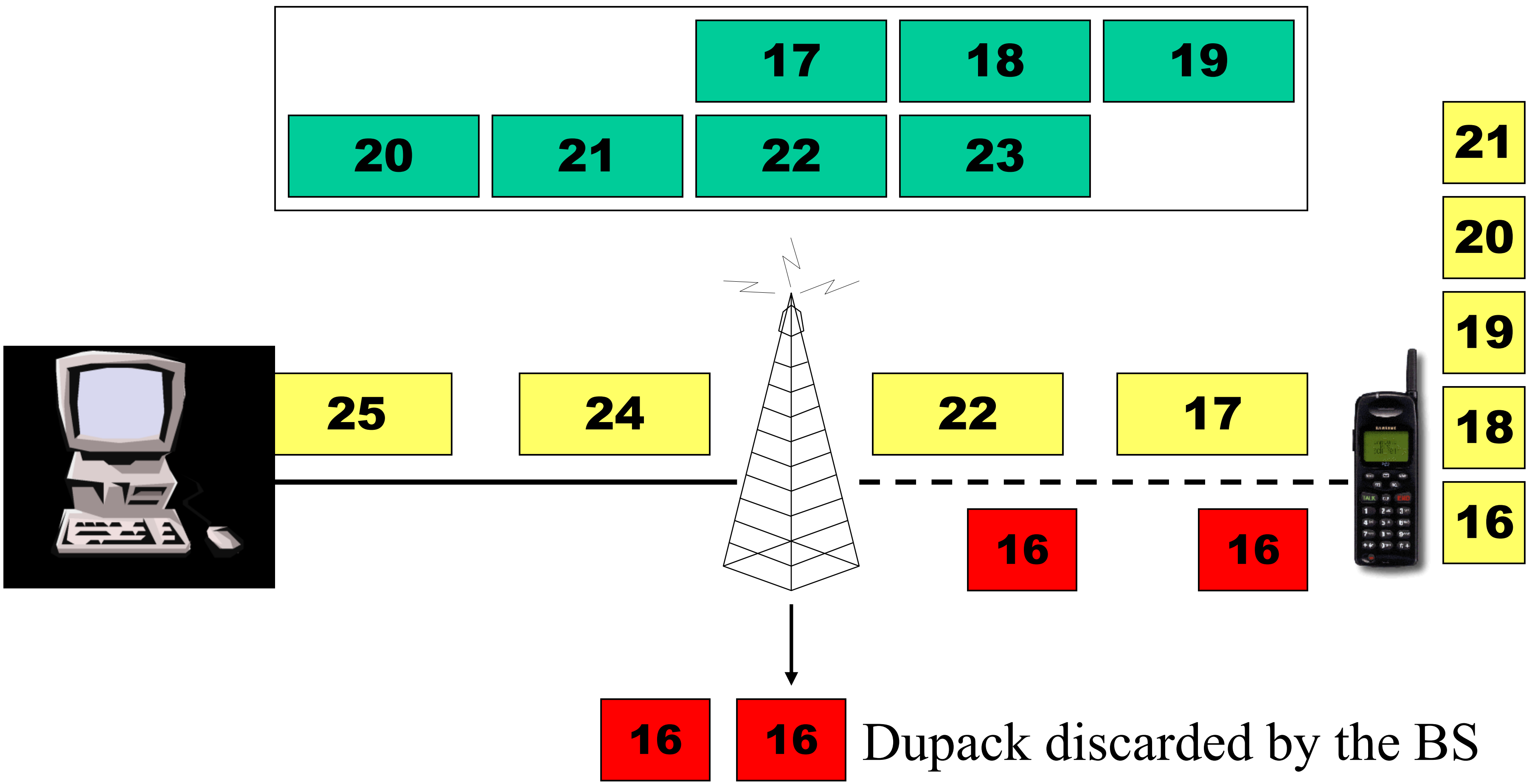
Snoop Protocol – Example (4/9)



Snoop Protocol – Example (5/9)

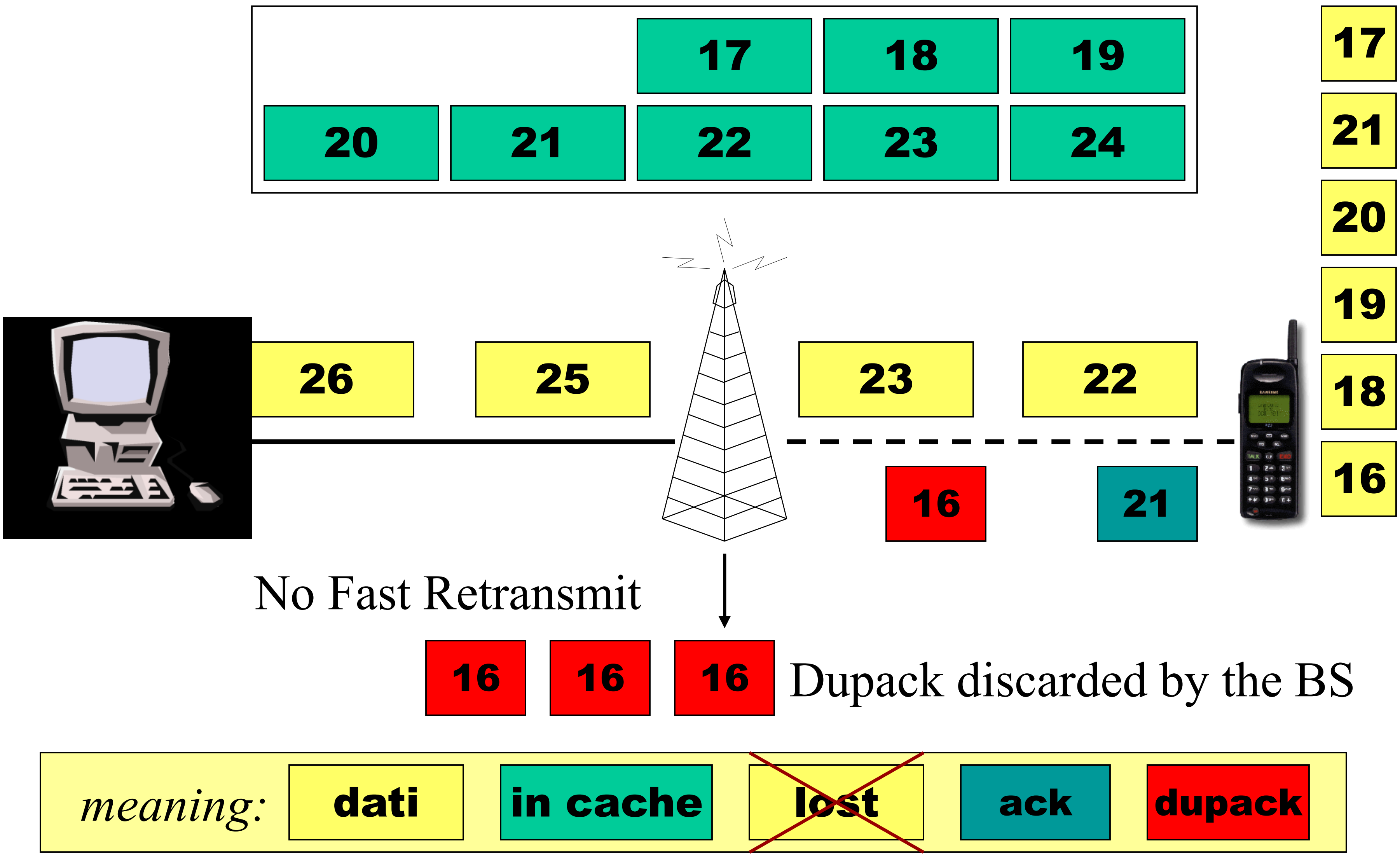


Snoop Protocol – Example (6/9)

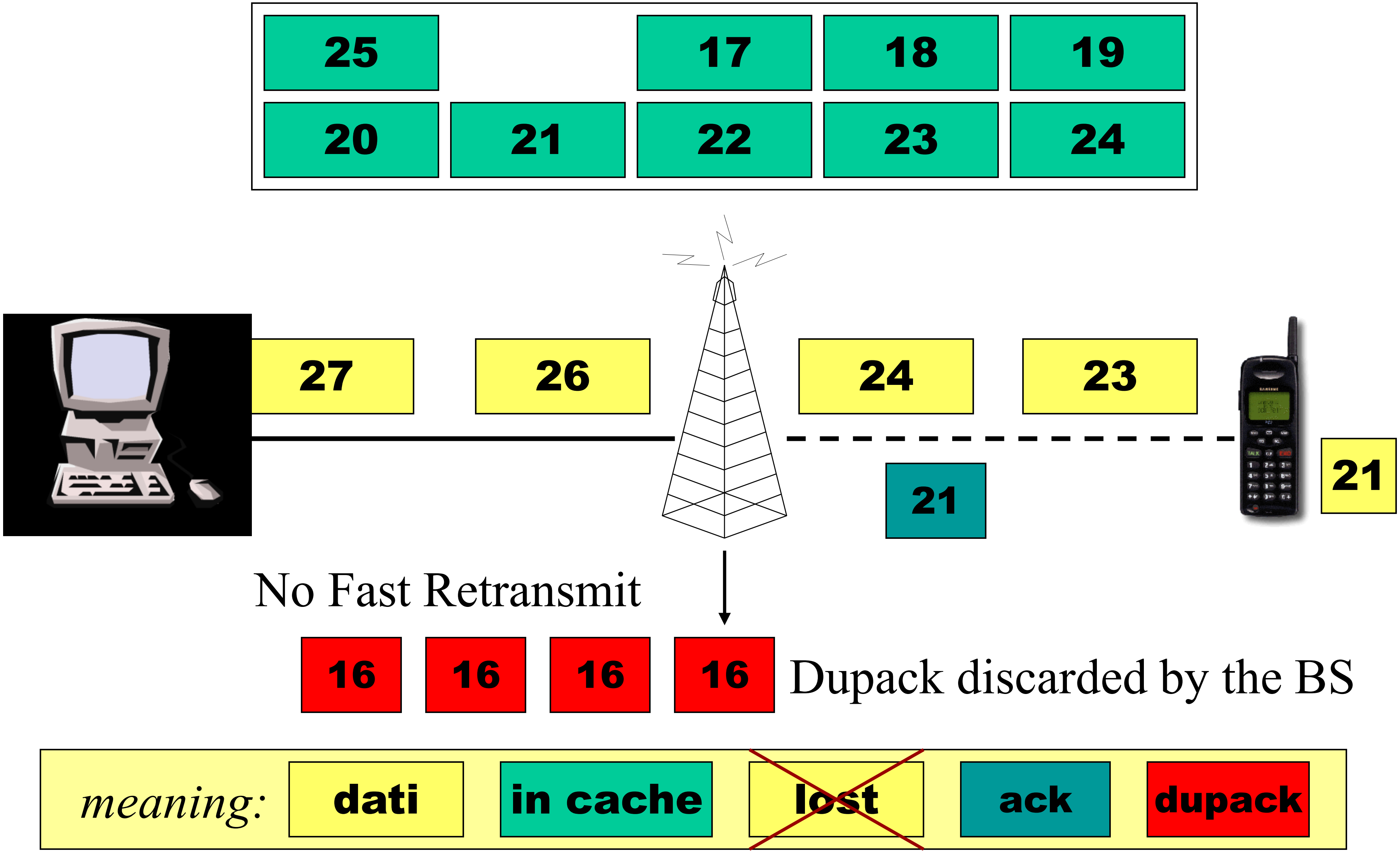


meaning: **dati** **in cache** **lost** **ack** **dupack**

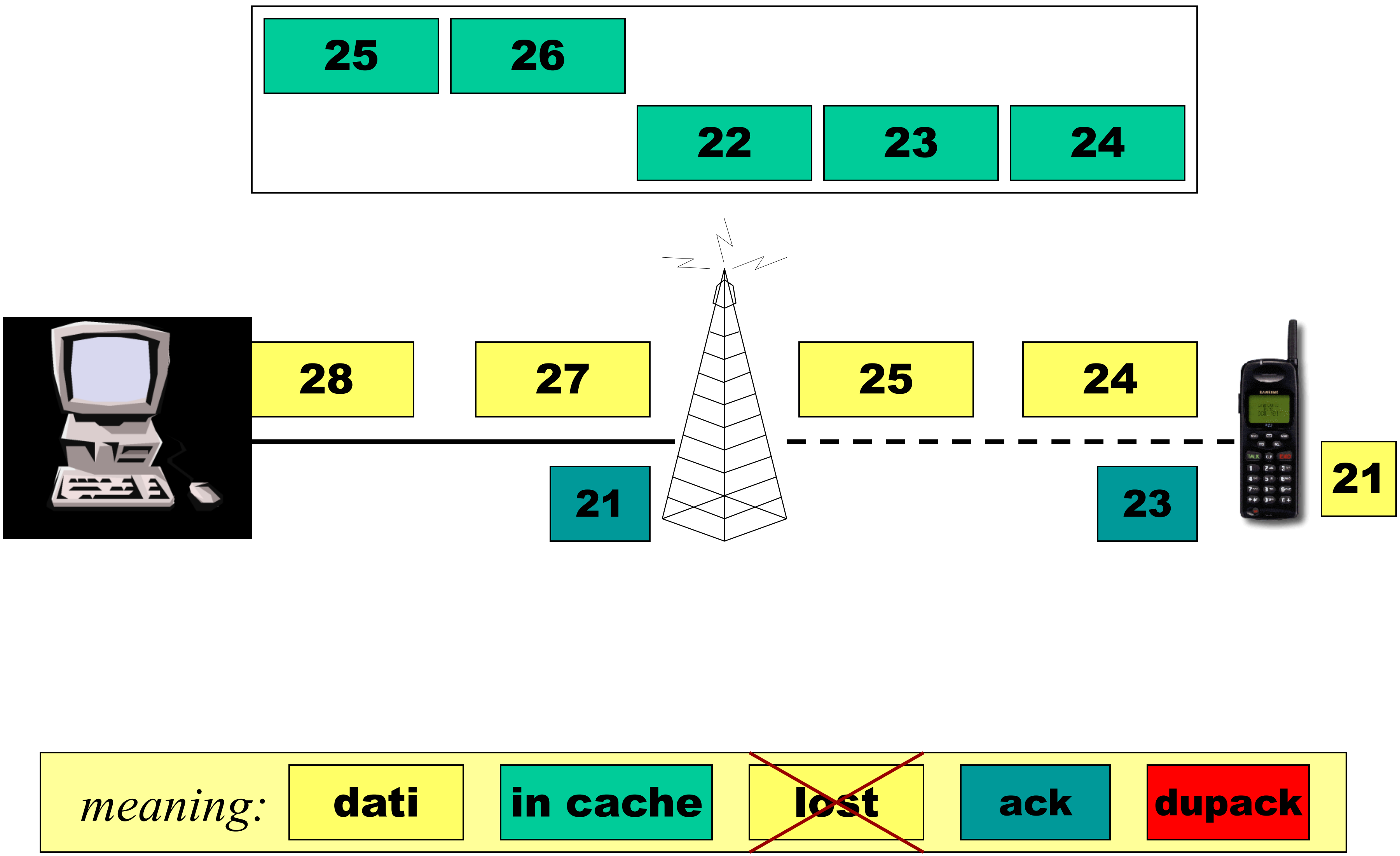
Snoop Protocol – Example (7/9)



Snoop Protocol – Example (8/9)



Snoop Protocol – Example (9/9)



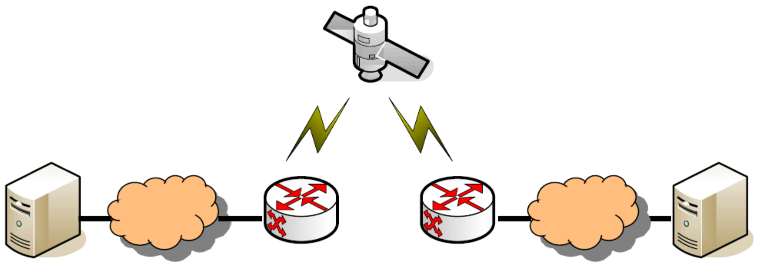
Snoop Protocol: Pro & Cons

- Pro:
  - End-to-End semantics preservation (almost)
  - Local (and timely) loss recovery
  - Addresses high BER

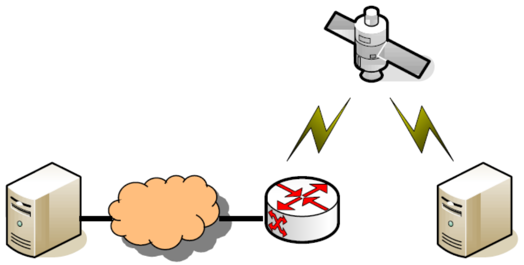
- Cons:
  - Requires little RTTs on the wireless link
  - Does not guarantee against long disconnections
  - Not utilizable immediately after a handoff (no packets in new cache)

Satellite Scenarios

- Geostationary Orbit (GEO) satellites → 36000 km
- Low Earth Orbit (LEO) satellites → 100-1500 km



Backbone configuration



Direct to home configuration

- Great increase in the Round Trip Time (RTT)
  - Up to 600 ms for GEO systems
- Non negligible Packet Error Rate (PER) due to the radio channel
  - Typical values in the range of [0-10%] depending on satellite constellation, weather conditions, antenna position, mobility, etc.

Slow Start & Congestion Avoidance Models

- Also referred as Van Jacobson algorithm
- In the Slow Start (SS) phase *W* is increased by 1 segment per every new ACK received
- In the Congestion Avoidance (CA) phase *W* is increased by 1/*W* segment per every new ACK received

previous congestion window

$$W_{i+1} = \begin{cases} W_i + 1 & SS \\ W_i + 1/W_i & CA \end{cases}$$

new congestion window after receiving one ACK

W is doubled every RTT

time in slow start

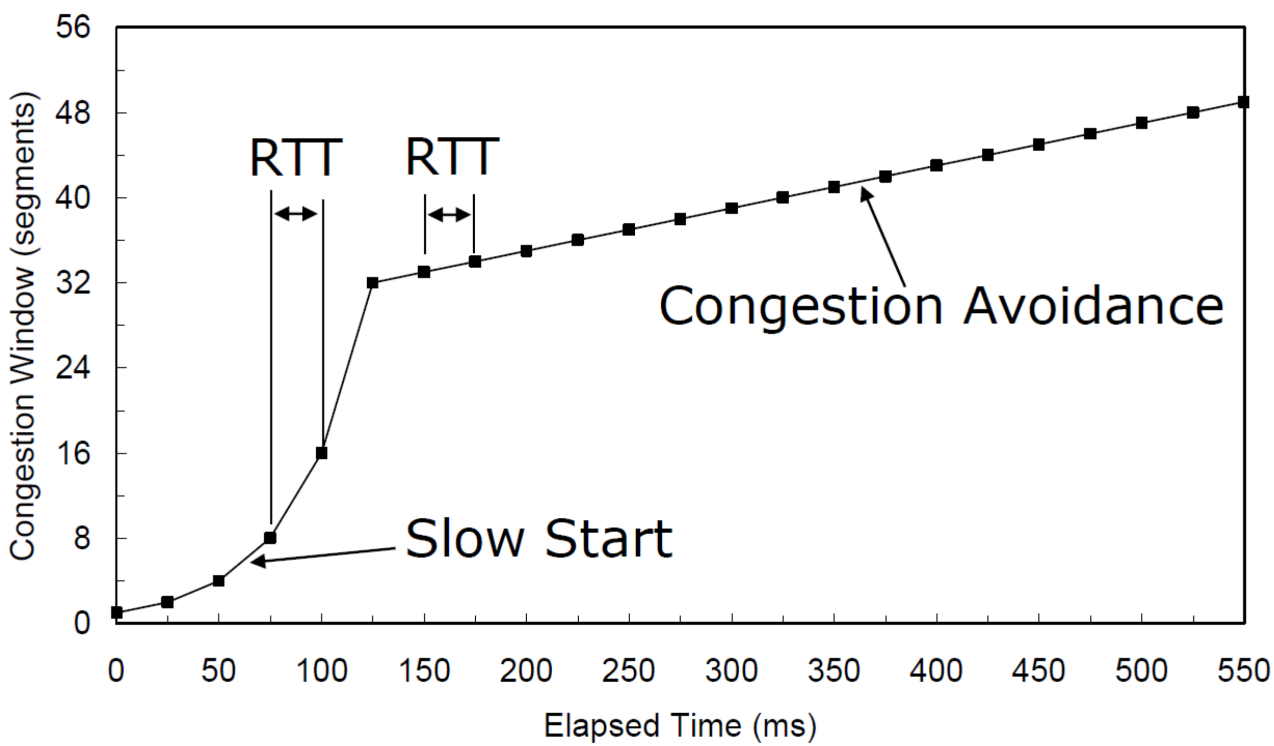
$$W(t) = \begin{cases} 2^{\frac{t}{RTT}} & 0 \le t < t_\gamma \quad SS \\ \dfrac{t - t_\gamma}{RTT} + \gamma & t \ge t_\gamma \quad CA \end{cases}$$

Number of RTTs elapsed since entered in congestion avoidance

ssthresh value

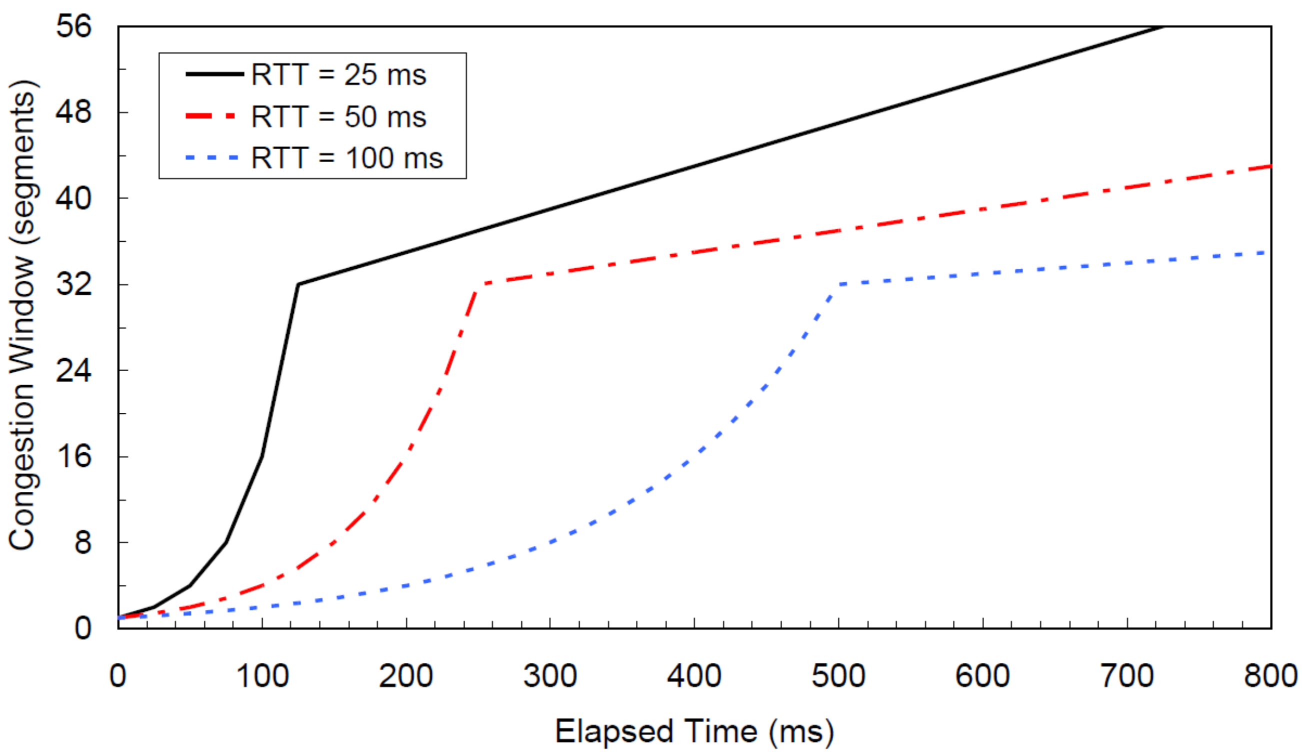
Slow Start & Congestion Avoidance Models

- In SS *W* is doubled at every RTT
- IN CA *W* is increased by 1 at every RTT
  - The discrete time behavior of *W* can be effectively approximate by a continuous time model



RTT Unfairness

- The longer the RTT the slower the *W* growth rate

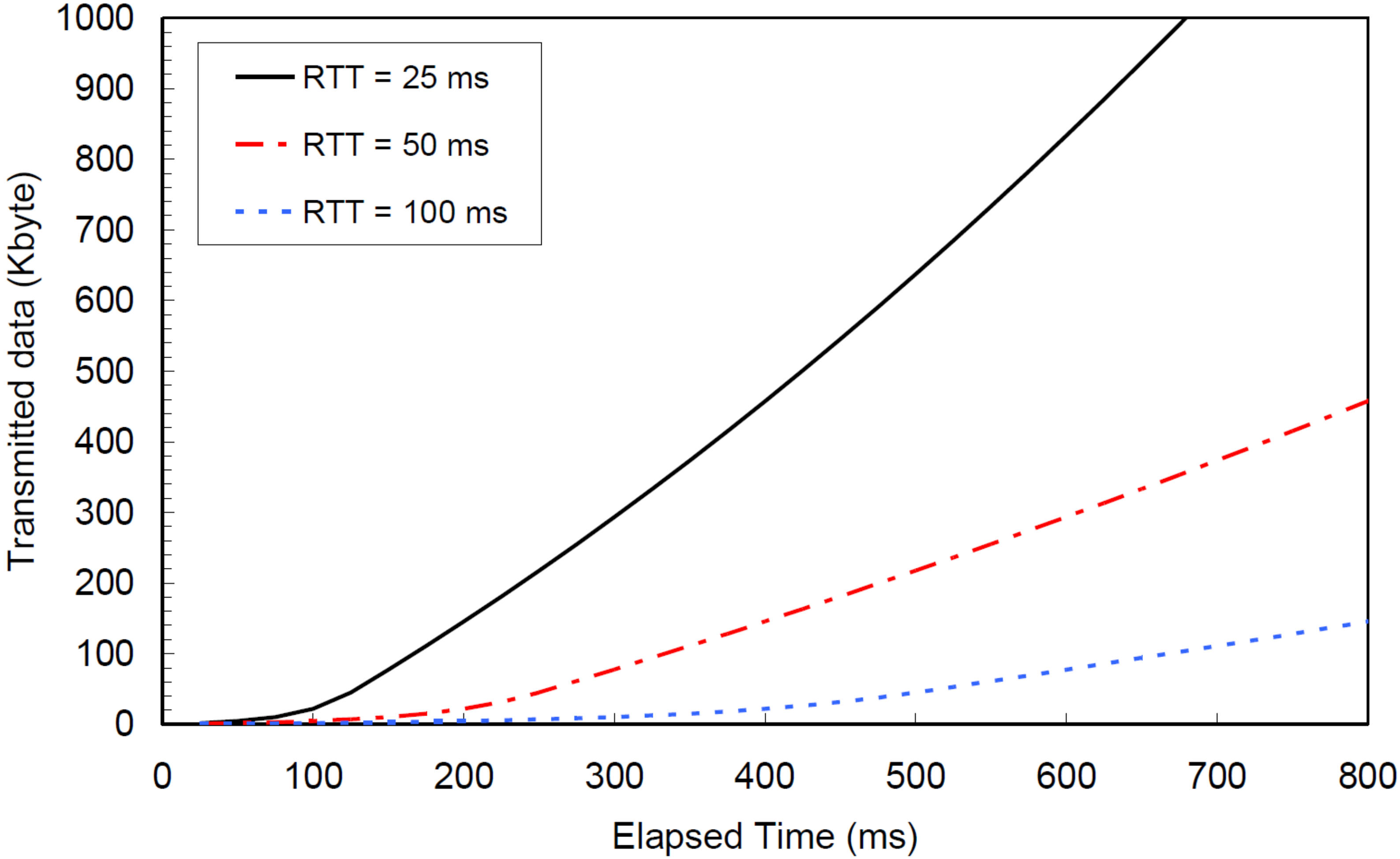


- Transmission rate *B(t)* (segments/sec) is given by:

$$B(t) = W(t) \,/\, RTT$$

RTT Unfairness

- Transmitted data vs RTT



TCP Hybla

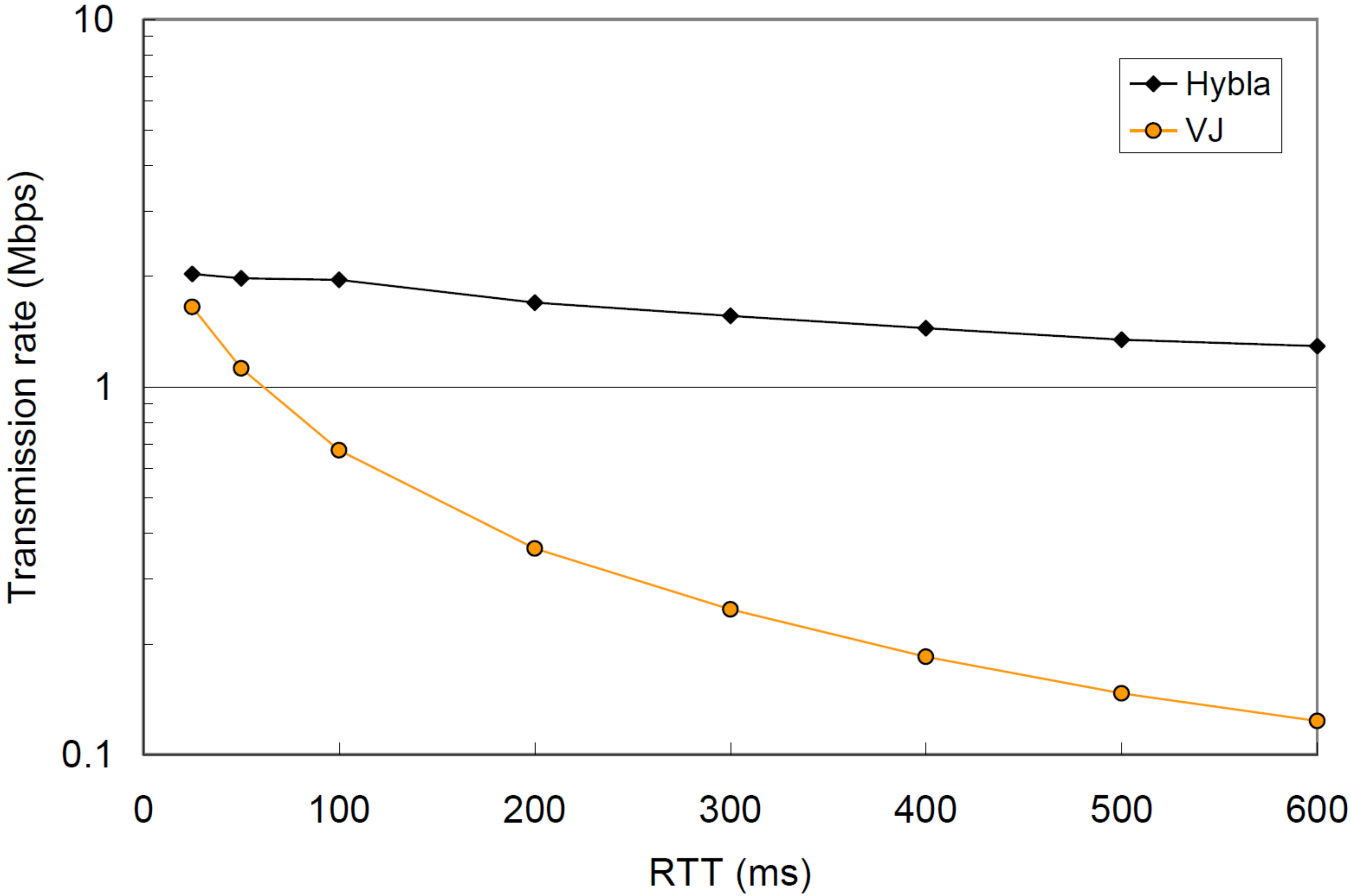
- TCP Hybla was first presented in 2004* with the aim of equalize the transmission rate against the RTT
  - Introduction of a parameter $\rho = RTT/RTT_0$
  - RTT is the actual Round Trip Time
  - RTT_0 is a reference Round Trip Time (e.g. RTT_0 =25ms)

$$W^H(t) = \begin{cases} \rho 2^{\rho \frac{t}{RTT}} & 0 \leq t < t_{\gamma,0} \quad \text{SS} \\ \rho\left[\rho \dfrac{t - t_{\gamma,0}}{RTT} + \gamma\right] & t \geq t_{\gamma,0} \quad \text{CA} \end{cases}$$

$$B^H(t) = \frac{W^H(t)}{RTT} = \begin{cases} \dfrac{2^{\frac{t}{RTT_0}}}{RTT_0} & 0 \leq t < t_{\gamma,0} \quad \text{SS} \\ \dfrac{1}{RTT_0}\left[\dfrac{t - t_{\gamma,0}}{RTT_0} + \gamma\right] & t \geq t_{\gamma,0} \quad \text{CA} \end{cases}$$

*C. Caini, and R. Firrincieli, "TCP Hybla: a TCP Enhancement for Heterogeneous Networks," Wiley Int. J. Satellite Commun. Netw., vol. 22, pp. 547-566, Sep.-Oct. 2004.

Satellite Link with Large RTT



TCP Hybla: Pros & Cons

- Pros:
  - End-to-End solution
  - Code modifications only at sender side
  - RTT used to speed up transmission speed for connections with long RTTs (e.g., satellites) in order to reach RTT fairness

- Cons:
  - Agressive behavior may result in multiple losses
  - Measured RTT is sensitive to buffer size
  - No handling of BER or disconnections
  - Fairness & Friendliness?

TCP Westwood & TCP Westwood+

- Pure End-to-End
- Flow Control based on an estimation of the available/eligible bandwidth (*BWE*):
  - Monitoring of acks' arrival rate at sender side
  - Use of this *BWE* to set cwnd e ssthresh after a loss:

3 dupack arrival:

- **ssthresh=BWE*RTTmin**

instead TCP New Reno:

`ssthresh = cwnd/2`

- if `(cwnd > ssthresh)` then `cwnd=ssthresh`

Timeout expiration:

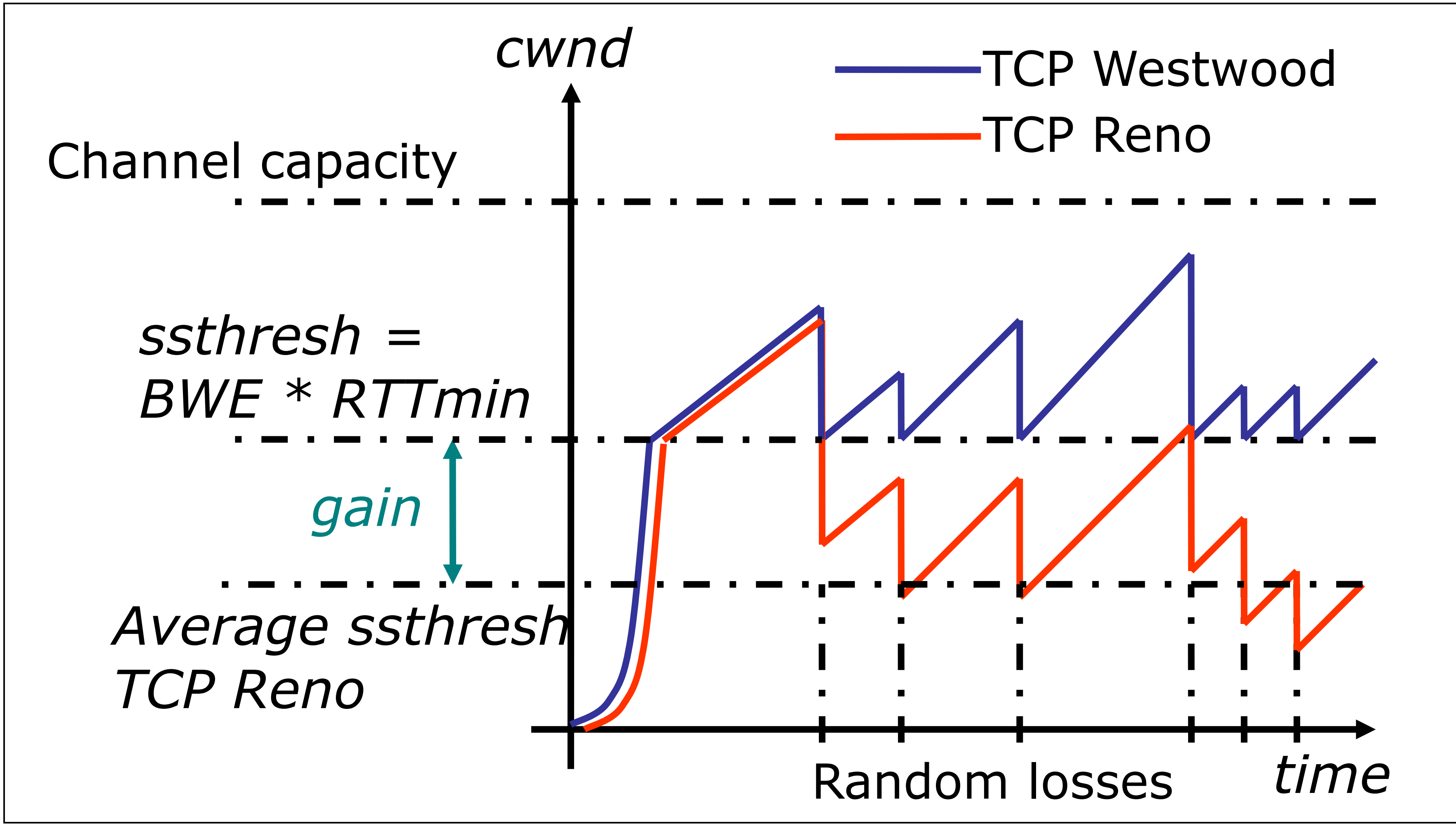
- **ssthresh=BWE*RTTmin**

instead TCP New Reno:

`ssthresh = cwnd/2`

- `cwnd = 1`

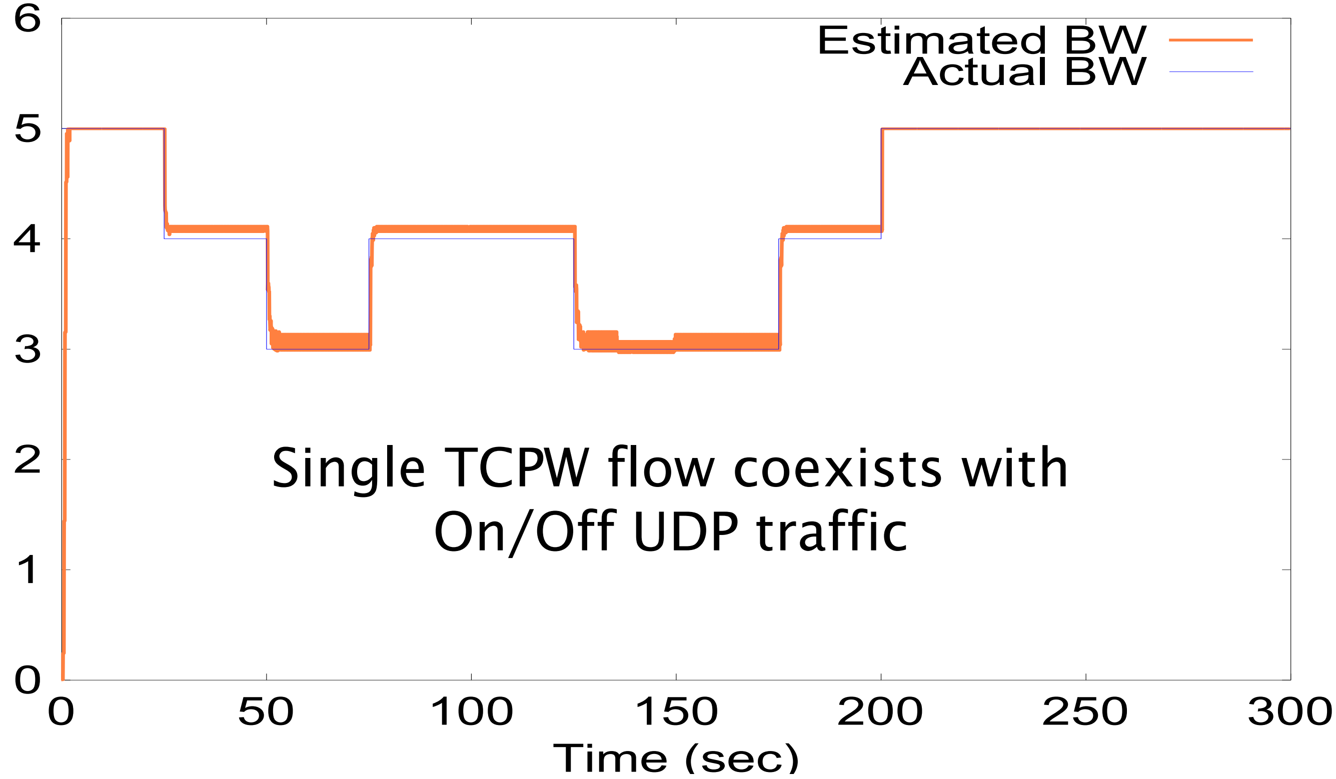
TCP Westwood vs. TCP Reno



TCP Westwood: Estimation

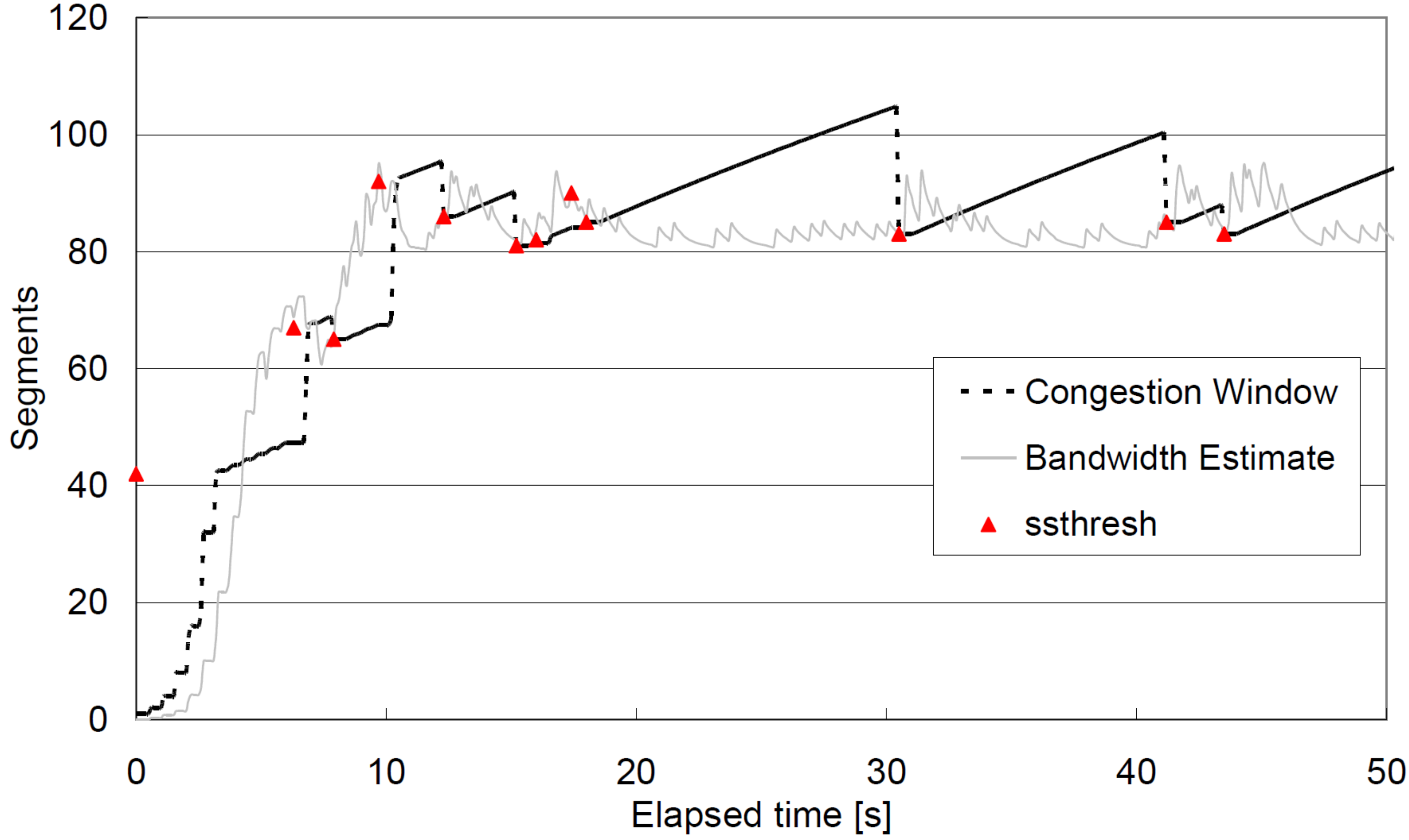
- Compute the Rate Estimate (*RE*) to enhance congestion control
  - RE is computed at the sender by *sampling* and *exponential filtering*
  - Samples are determined from ACK *inter-arrival times* and info in ACKs regarding amounts of *bytes delivered*
- RE is used by the sender to properly set *cwnd*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

Fair RE = "Residual Bandwidth" Estimate?

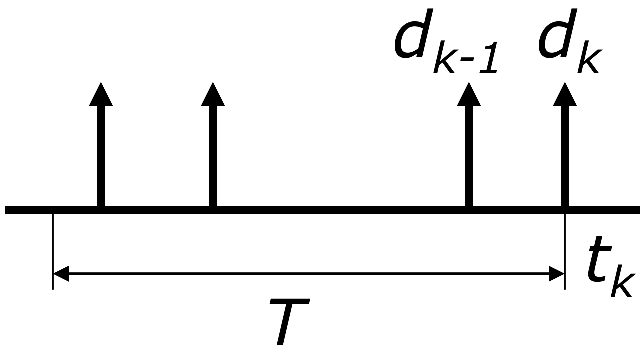


Single TCPW flow at equilibrium does estimate "residual bottleneck bandwidth"

Terrestrial Wireless Link with PER



TCPW Rate Estimation (TCP RE)



T is the sample interval

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

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

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

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

TCP Westwood: Pros & Cons

- Pros:
  - bandwidth estimation at sender side to set ssthresh & cwnd so as to reach higher throughput
  - Code modifications only at sender side

- Cons:
  - Wrong Bandwidth Estimation over asymmetric links
  - No specific mechanism to handle disconnections or very high BER
  - Fairness & Friendliness?

TCP Adaptive-Selection

- Going back to end-to-end enhancements, few question arise:
  - is it necessary to make a definitive choice among TCP enhancements?
  - Why not to select optimized TCP variant on different connections on the same server in an adaptive way?
  - Is there any room for performance improvement?
  - Is it feasible to simultaneously run different TCP enhancements on the same machine?
- The Adaptive-Selection* concept try to answer to all these questions

*C. Caini, R. Firrincieli, and D. Lacamera, "The TCP 'Adaptive- Selection' Concept", IEEE Systems Journal, vol. 2, no. 1, pp.83-89, Mar. 2008.

TCP Adaptive-Selection

- The TCP adaptive-selection concept is very simple:
  - On the same server not a single TCP variant, but concurrent use of different TCP enhancements to match the different characteristics of connections.
- It can be applied in different ways, depending on:
  - the agent that performs the TCP selection (i.e. receiver, intermediate router, sender)
  - the possible exploitation of a cross-layer approach
  - the possibility to change the TCP version on an on-going connection
    - "dynamic" adaptive-selection (like gears in a car)

TCP Adaptive-Selection

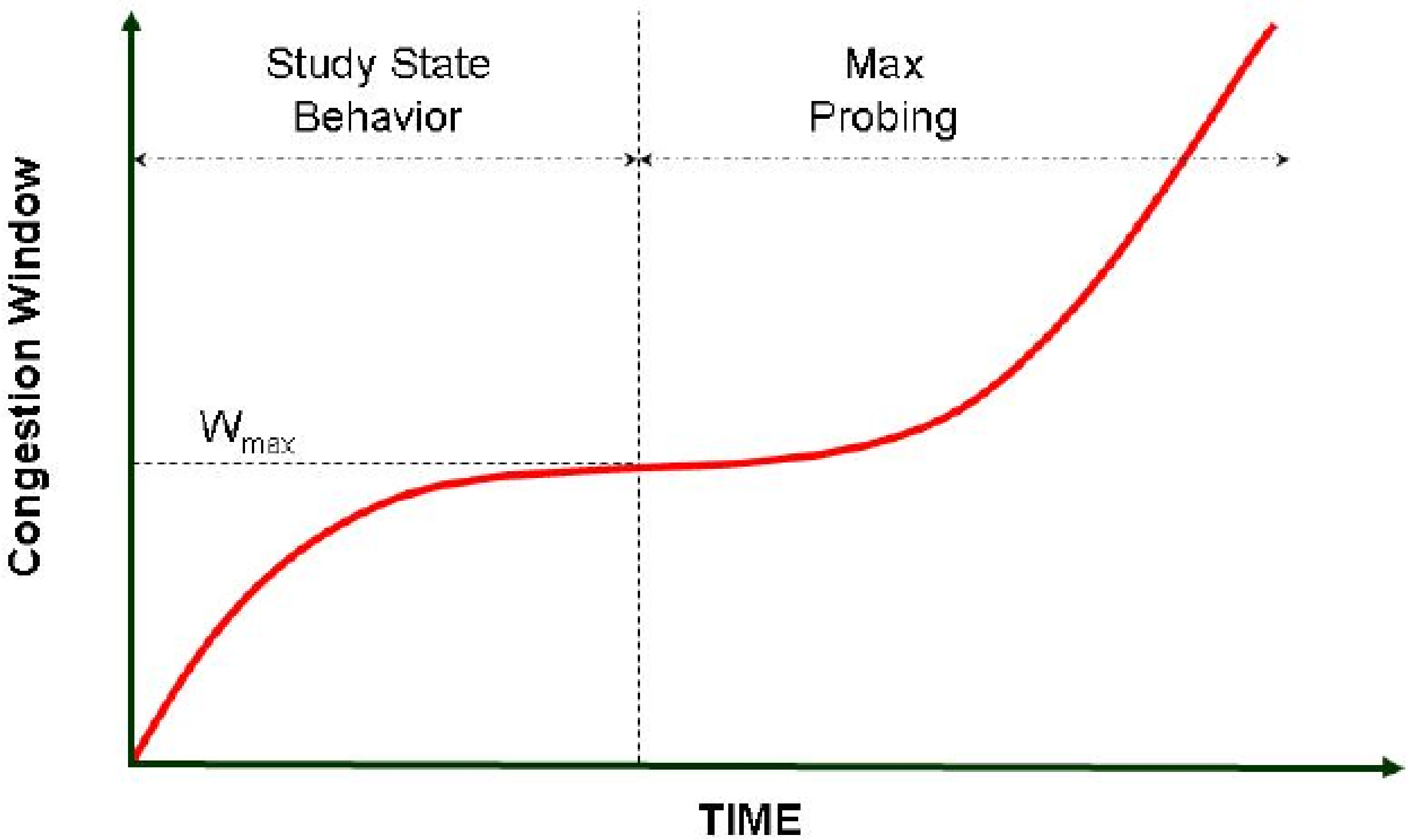
- Linux OS appears the most convenient choice to implement TCP adaptive-selection
  - Most TCP variants are already available as modules
  - A new "TCP adaptive-selection" module that calls other modules could be the solution
- Several possibilities for the decision criteria
  - TCP internal parameters (such as RTT and /or Bandwidth estimation)
  - Cross-layer information
  - Reliable channel estimation

Need for quick and efficient metrics to determine best choice at any time

TCP CUBIC

- Optimized congestion control algorithm for high speed networks with high latency (Long Fat Pipes/Networks)
- The cwnd is a cubic function of time since last congestion event
  - inflection point set to the window prior to the last congestion event
    - CUBIC grows very quickly initially
    - Slow down and maintains stable to a value around the cwnd when the congestion happened
    - If no loss happens (maybe some flow left the network leaving more bandwidth available) quickly grows again
- Major difference between TCP CUBIC and standard TCPs
  - TCP CUBIC does not rely on the receipt of ACKs to increase the cwnd
  - TCP CUBIC's cwnd depends only on the last congestion event
    - Less RTT-unfairness since the window growth is independent of RTT
- TCP CUBIC is implemented and used by default in Linux kernels 2.6.19 and above

TCP CUBIC: Cwnd Growth



Class Project Idea!

- Take various TCP protocols and test/compare them in a realistic new environment
  - Mobility
  - Starbuck's / Coffee Shop
  - UMTS
  - …
- NS-2/NS-3 simulations or Linux
- <u>Alternative</u>: read and present paper(s) on TCP (or general congestion control) for some wireless environment