TCP on
Wireless Ad Hoc Networks

CS 218 Oct 22, 2003

- TCP overview
- Ad hoc TCP : mobility, route failures and timeout
- TCP and MAC interaction study
- TCP fairness achieved with Active Neighbor estimate
- The problem of fairness and the NRED solution
- TCP over wired/wireless links

TCP ad hoc: Relevant literature

Holland and Vaidya: Impact of Routing and Link Layers on TCP Perofrmance in mobile ad hoc nets, Mobicom 99

T. D. Dyer and R. V. Boppana, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks," In Proceedings of Mobihoc 2001, 2001.

K. Tang and M. Gerla, "Fair Sharing of MAC under TCP in Wireless Ad Hoc Networks," In Proceedings of IEEE MMT'99, Venice, Italy, Oct. 1999.

•Kaixin Xu, et al TCP Behavior across Multihop Wireless Networks and the Wired Internet *-ACM WoWMoM 2002 (co-located with MobiCom 2002), Atlanta, Ga, Sep. 2002*

TCP Congestion Control

- **end-end control (no network assistance)**
- **sender limits transmission:**

  `LastByteSent-LastByteAcked`
  `£ CongWin`

Roughly,

$$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \ \text{Bytes/sec}}$$

- **`CongWin` is dynamic, function of perceived network congestion**

How does sender perceive congestion?

- **loss event = timeout *or* 3 duplicate acks**
- **TCP sender reduces rate (`CongWin`) after loss event**

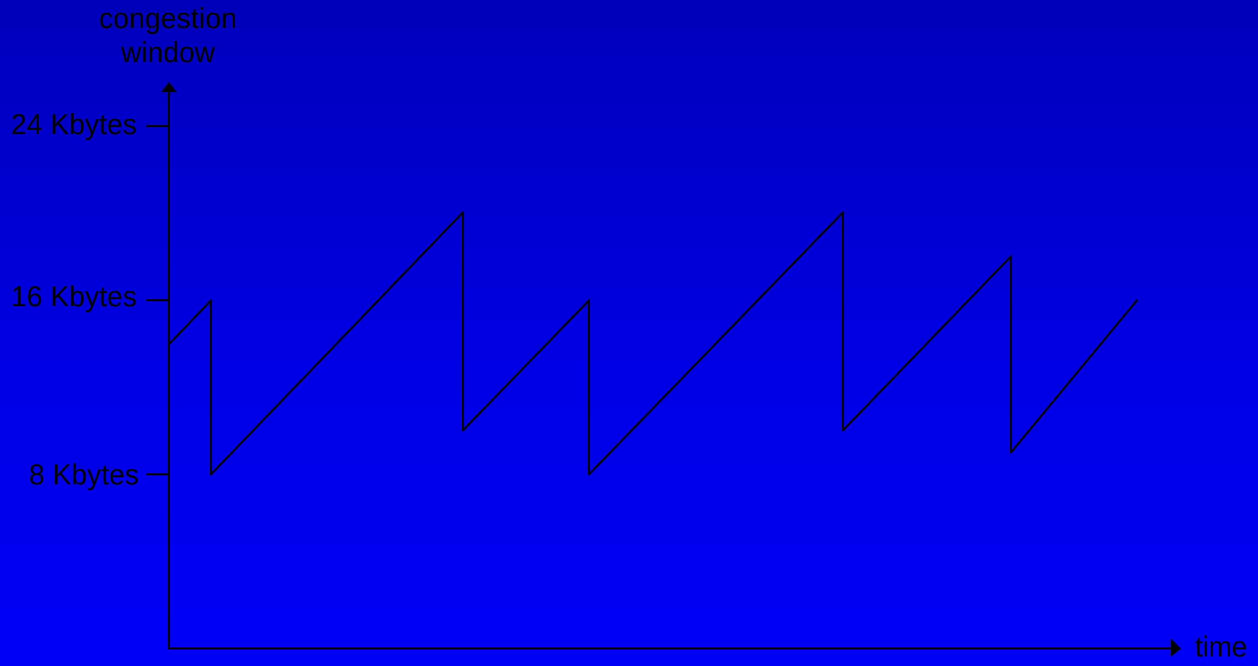
two mechanisms:

- AIMD
- slow start

TCP AIMD

multiplicative decrease: **cut** `CongWin` **in half after loss event**

additive increase: **increase** `CongWin` **by 1 MSS every RTT in the absence of loss events:** ***probing***



Long-lived TCP connection

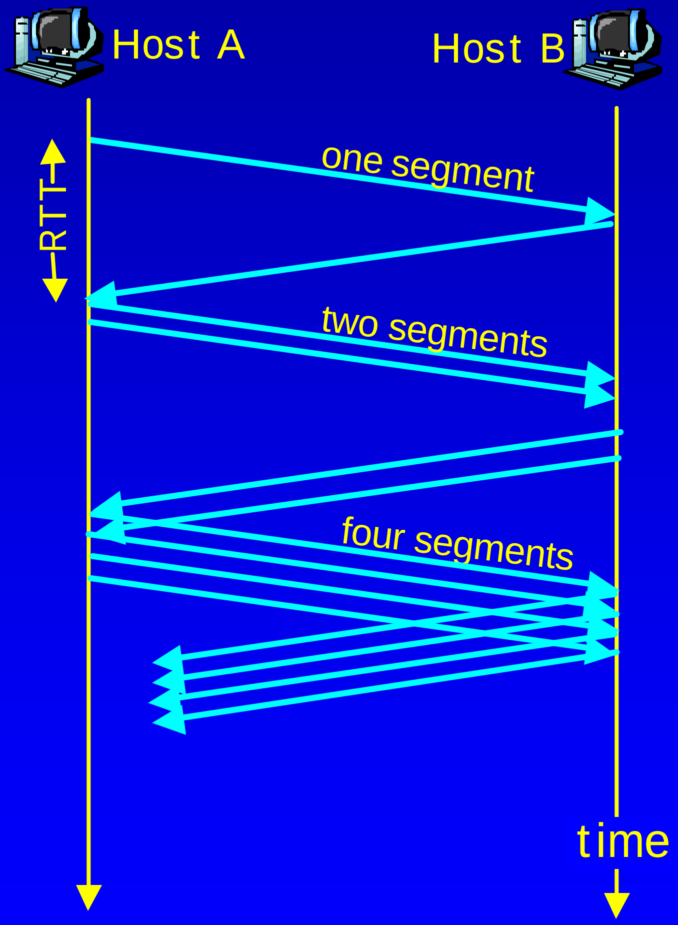
TCP Slow Start

- **When connection begins,** `CongWin` **= 1 MSS**
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- **available bandwidth may be >> MSS/RTT**
  - desirable to quickly ramp up to respectable rate

- **When connection begins, increase rate exponentially fast until first loss event**

TCP Slow Start (more)

- **When connection begins, increase rate exponentially until first loss event:**
  - double **`CongWin`** every RTT
  - done by incrementing **`CongWin`** for every ACK received
- **Summary: initial rate is slow but ramps up exponentially fast**



Refinement

- **After 3 dup ACKs:**
  - **`CongWin`** is cut in half
  - window then grows linearly
- **<u>But</u> after timeout event:**
  - **`CongWin`** instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

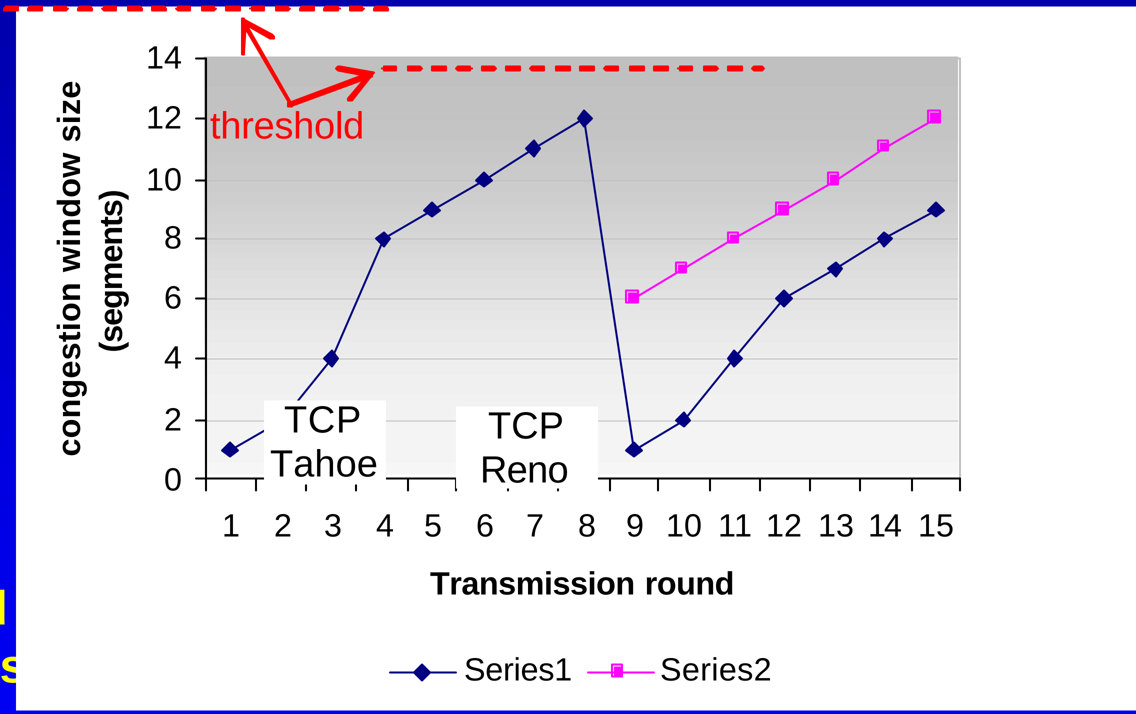
Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout before 3 dup ACKs is "more alarming"

Refinement (more)

Q: When should the exponential increase switch to linear?

A: When `CongWin` gets to 1/2 of its value before timeout.



Implementation:

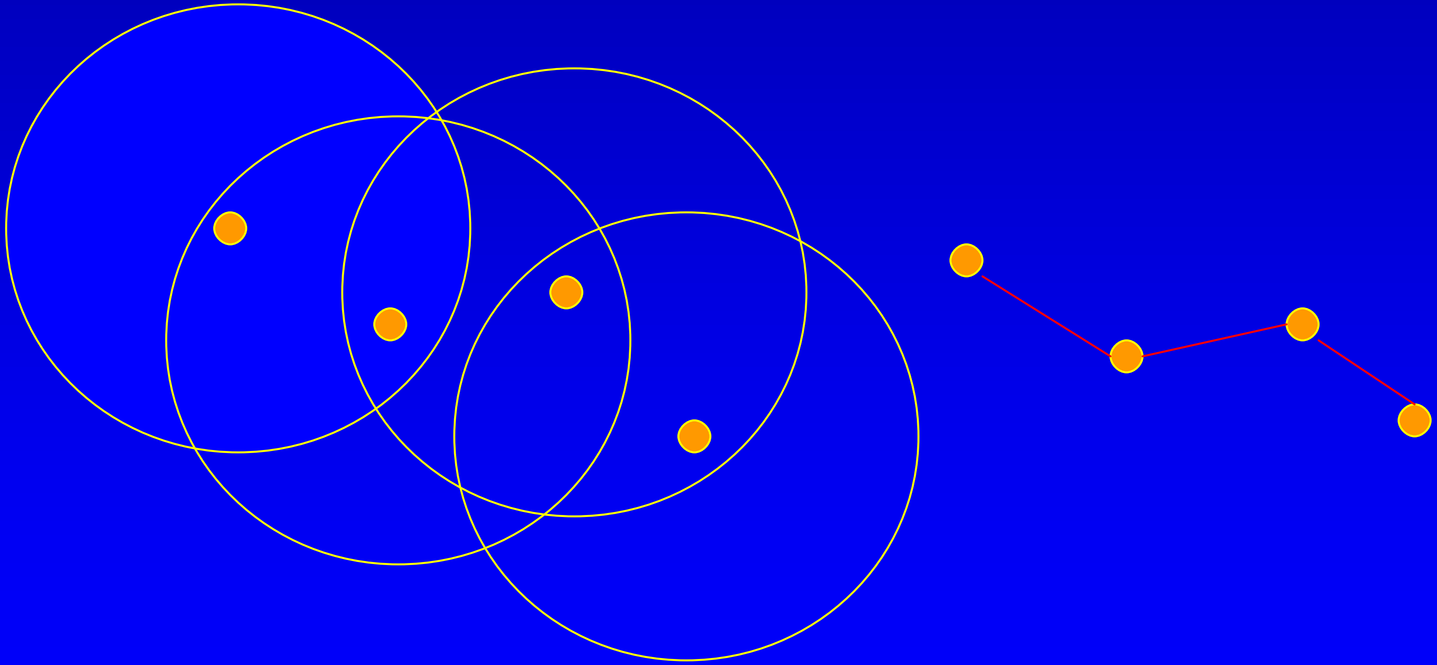
- **Variable Threshold**
- **At loss event, Threshold set to 1/2 of CongWin jus before loss event**

Summary: TCP Congestion Control

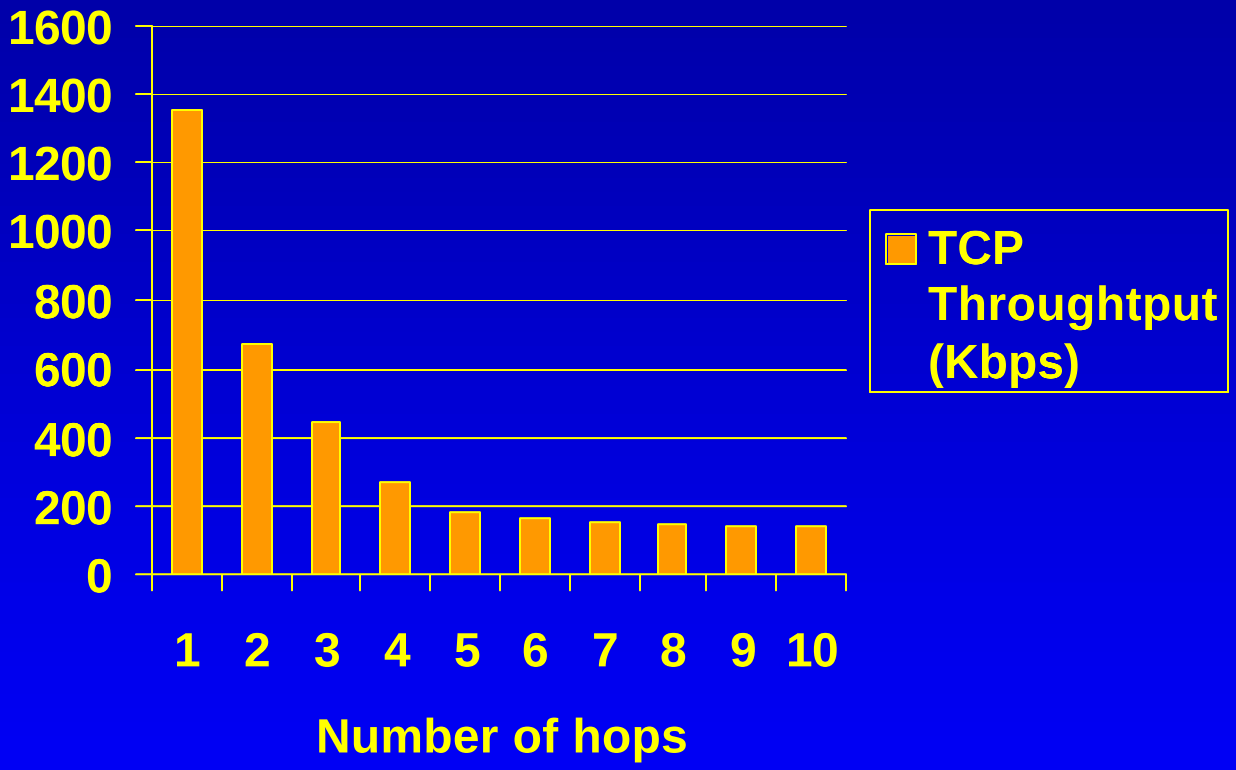
- When `CongWin` is below `Threshold`, sender in slow-start phase, window grows exponentially.
- When `CongWin` is above `Threshold`, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, `Threshold` set to `CongWin/2` and `CongWin` set to `Threshold`.
- When timeout occurs, `Threshold` set to `CongWin/2` and `CongWin` is set to 1 MSS.

Impact of Mobility on TCP

- Mobility causes route changes



Impact of Multi-Hop Wireless Paths



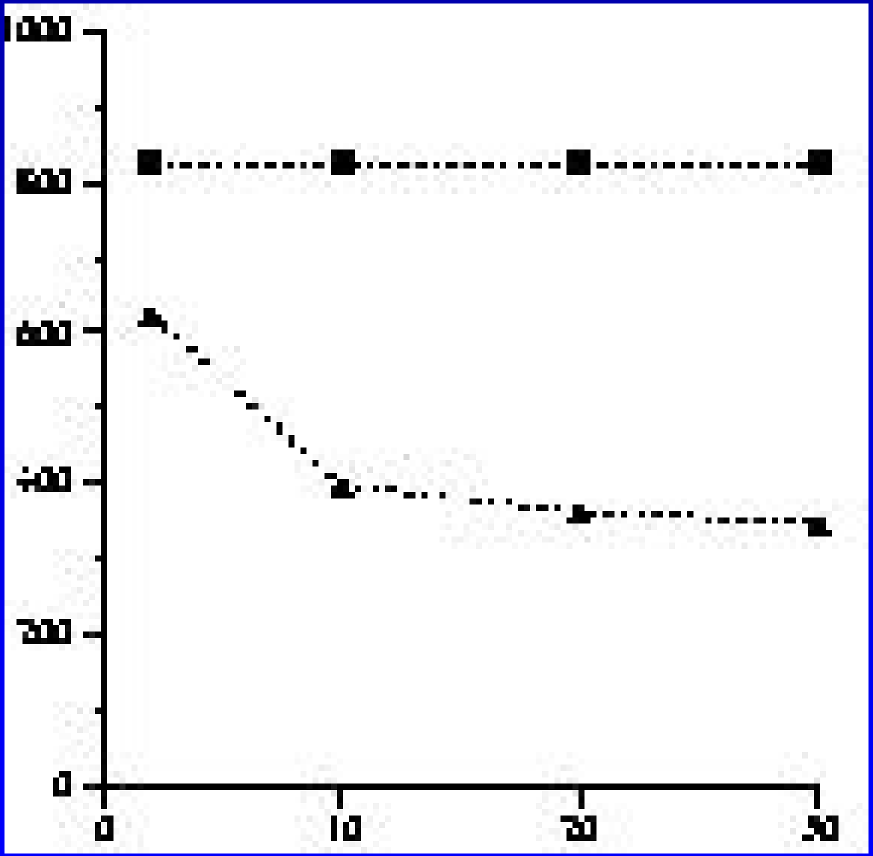
TCP Throughput using 2 Mbps 802.11 MAC

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 …

Average
Throughput
Over
50 runs



Ideal

Actual

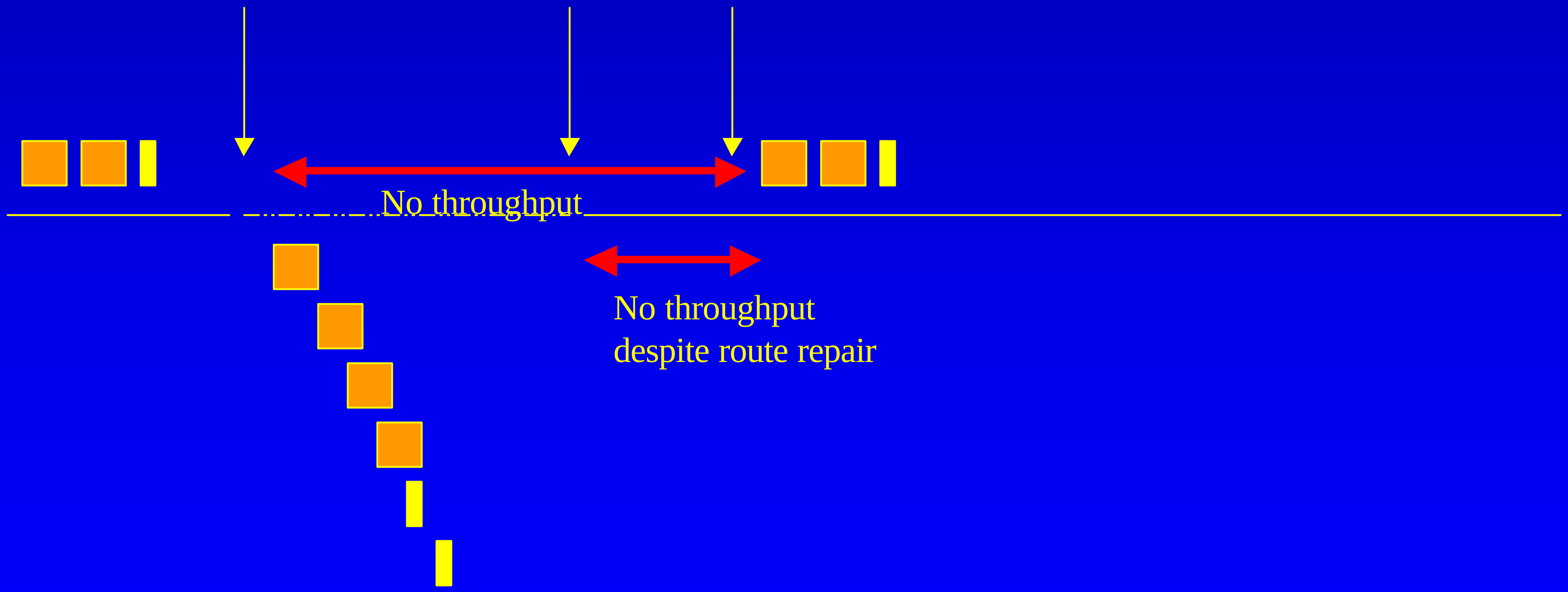
Speed (m/s)

Why Does Throughput Degrade?

mobility causes link breakage, resulting in route failure

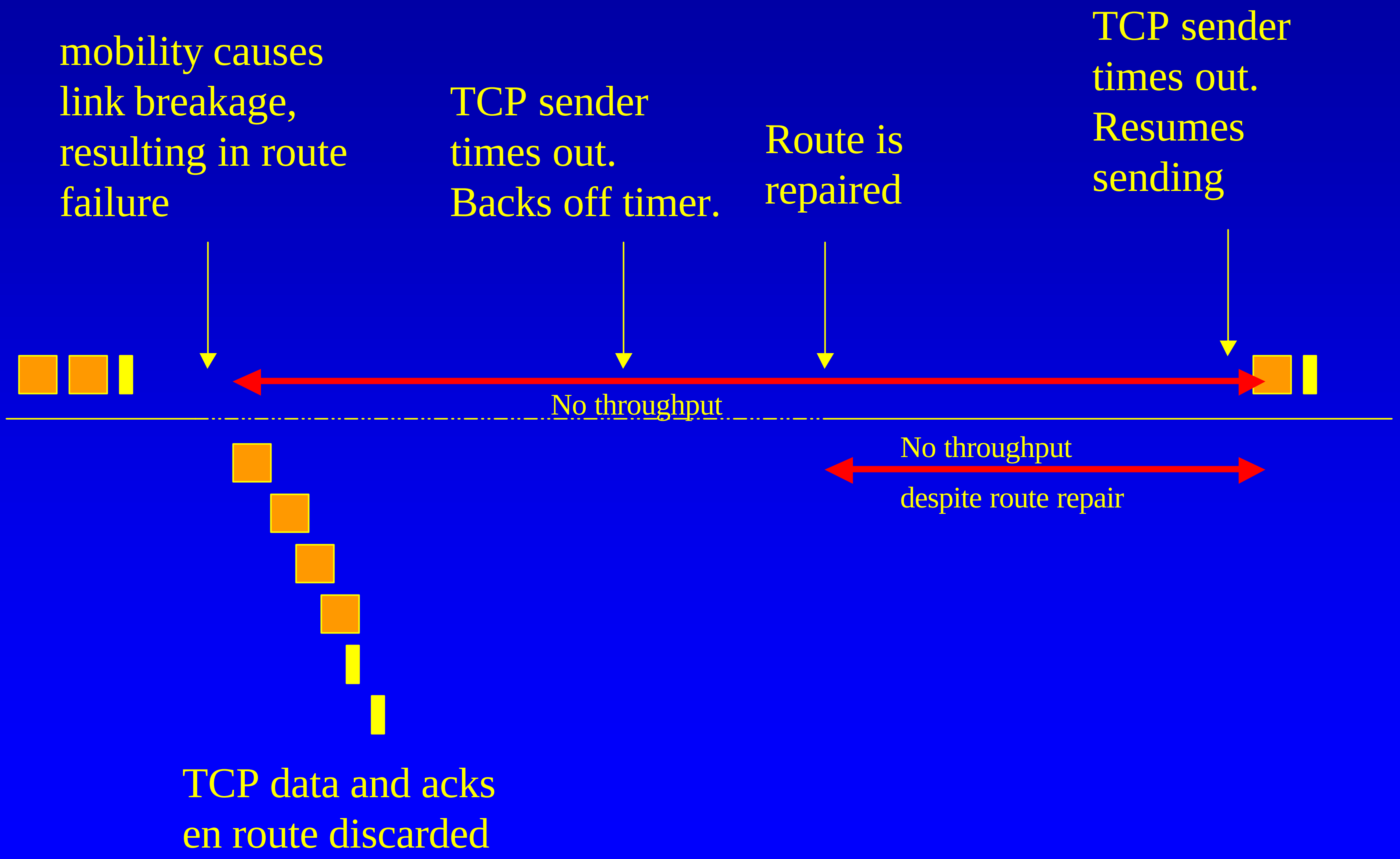
Route is repaired

TCP sender times out. Starts sending packets again



TCP data and acks en route discarded

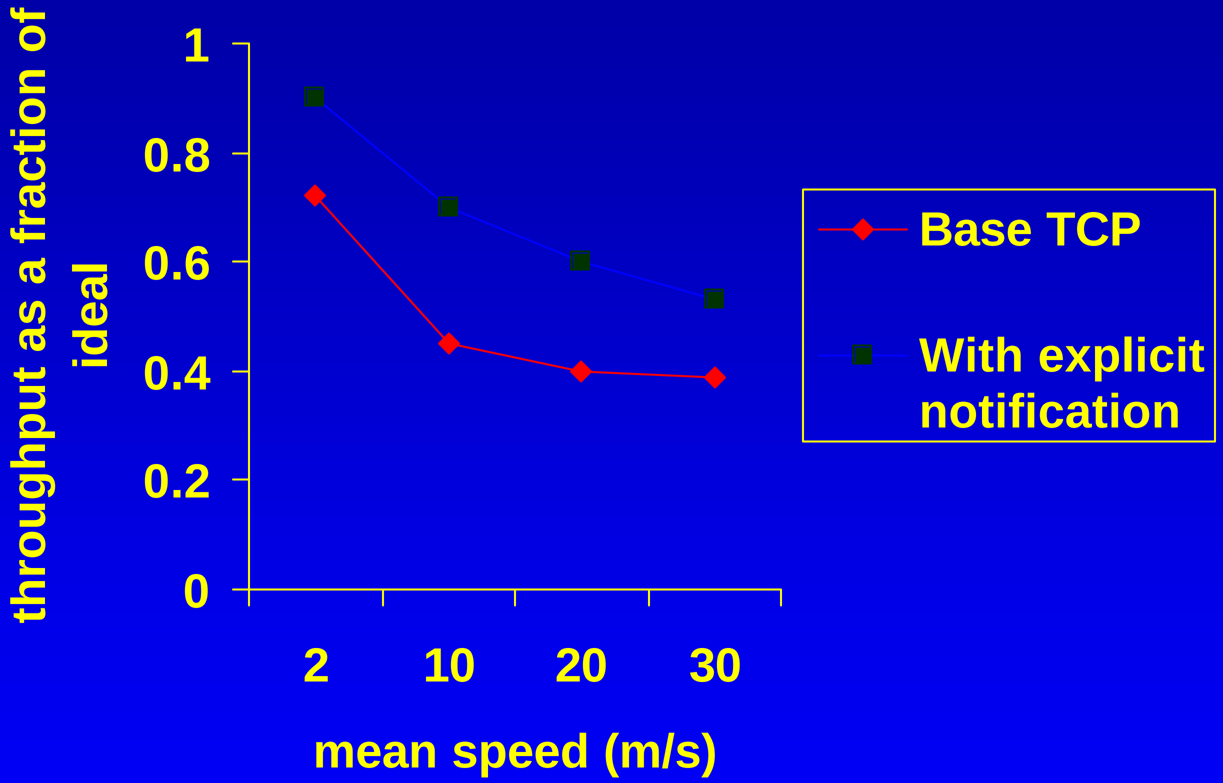
Why Does Repair Latency hurt?


mobility causes link breakage, resulting in route failure
TCP sender times out. Backs off timer.
Route is repaired
TCP sender times out. Resumes sending
No throughput
No throughput
despite route repair
TCP data and acks en route discarded

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**

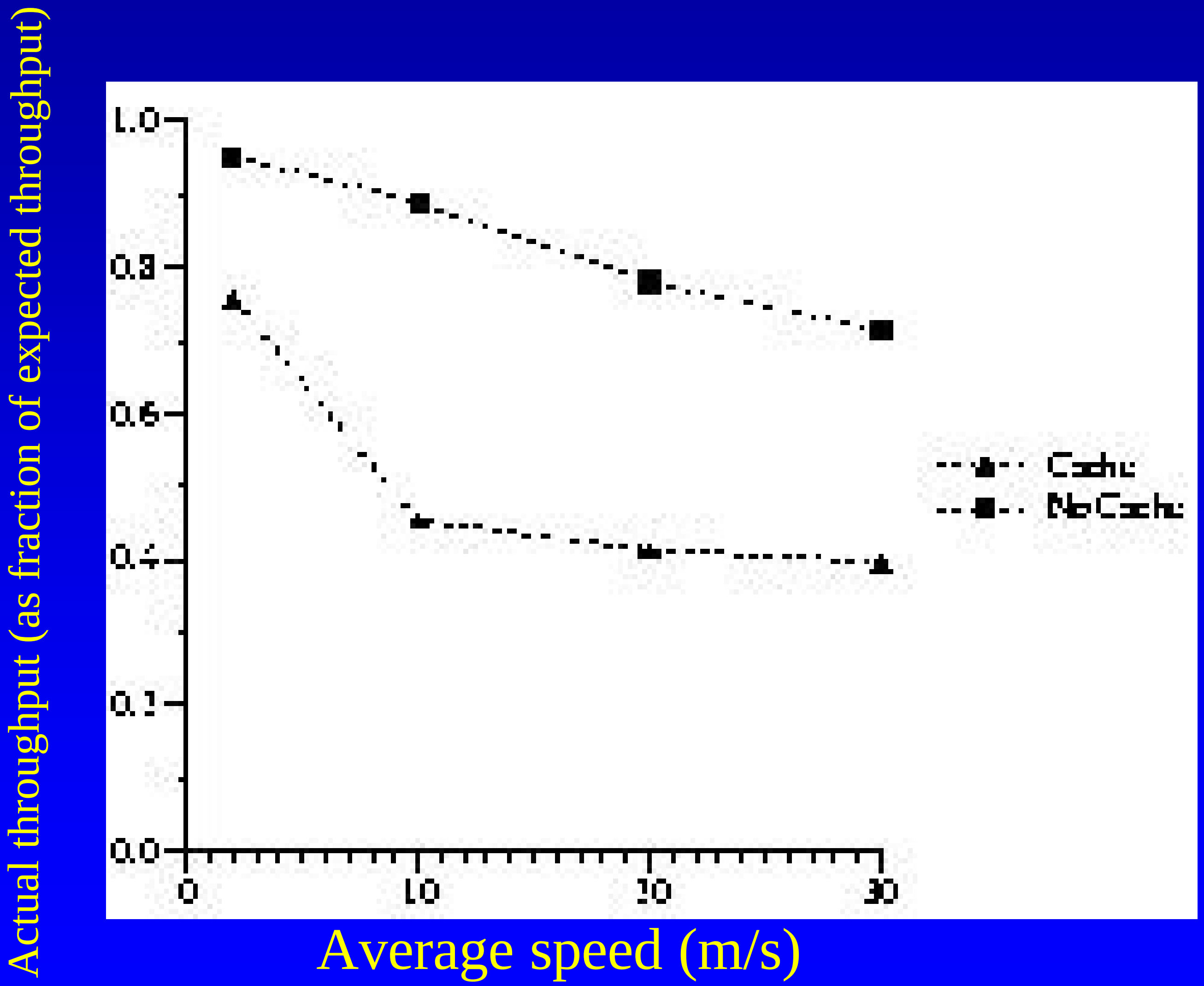
Performance with Explicit Notification



Impact of Caching

- **Route caching has been suggested as a mechanism to reduce route discovery overhead [Broch98]**
- **Each node may cache one or more routes to a given destination**
- **When a route from S to D is detected as broken, node S may:**
  - Use another cached route from local cache, or
  - Obtain a new route using cached route at another node

To Cache or Not to Cache

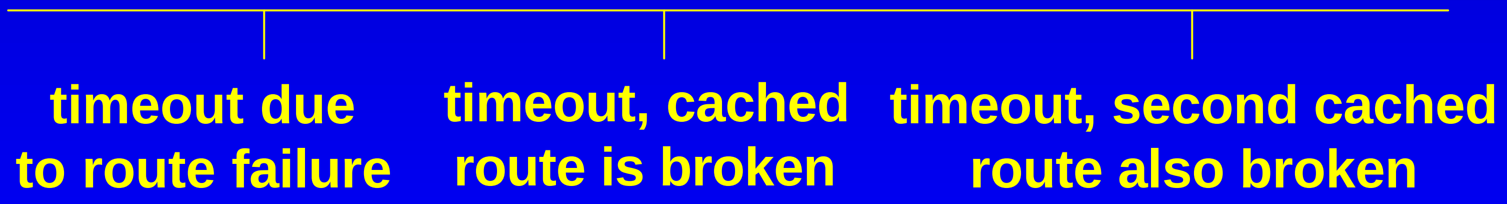


Why Performance Degrades With Caching

- When a route is broken, route discovery returns a cached route from local cache or from a nearby node

- After a time-out, TCP sender transmits a packet on the new route.

  However, what if the cached route has also broken after it was cached?


timeout due to route failure
timeout, cached route is broken
timeout, second cached route also broken

- Another route discovery, and TCP time-out interval
- Process repeats until a good route is found

Issues
To Cache or Not to Cache

- Caching can result in faster route "repair"

- Faster does not necessarily mean correct!

- If incorrect repairs occur often enough, caching performs poorly

- Need mechanisms for determining when cached routes are stale

Caching and TCP performance

- Caching can reduce overhead of route discovery even if cache accuracy is not very high

- But if cache accuracy is not high enough, gains in routing overhead may be offset by loss of TCP performance due to multiple time-outs

TCP Performance

Two factors result in degraded throughput in presence of mobility:

- **Loss of throughput that occurs while waiting for TCP sender to timeout (as seen earlier)**
  - This factor can be mitigated by using explicit notifications and better route caching mechanisms

- **Poor choice of congestion window and RTO values after a new route has been found**
  - How to choose *cwnd* and *RTO* after a route change?

Issues
Window Size After Route Repair

- Same as before route break: may be too optimistic
- Same as startup: may be too conservative
- Better be conservative than overly optimistic
  - Reset window to small value after route repair
  - Let TCP ramp up to suitable window size
  - Anyway, window impact low on paths with small delay-bdw product

Issues
RTO After Route Repair

- **Same as before route break**
  - If new route long, this RTO may be too small, leading to premature timeouts and unnecessary retransmissions
- **Same as TCP start-up (6 second)**
  - May be too large
  - May result in slow response to next packet loss
- **Another plausible approach: new RTO = function of old RTO, old route length, and new route length**
  - Example: new RTO = old RTO * new route length / old route length
  - Not evaluated yet
  - Pitfall: RTT is not just a function of route length