

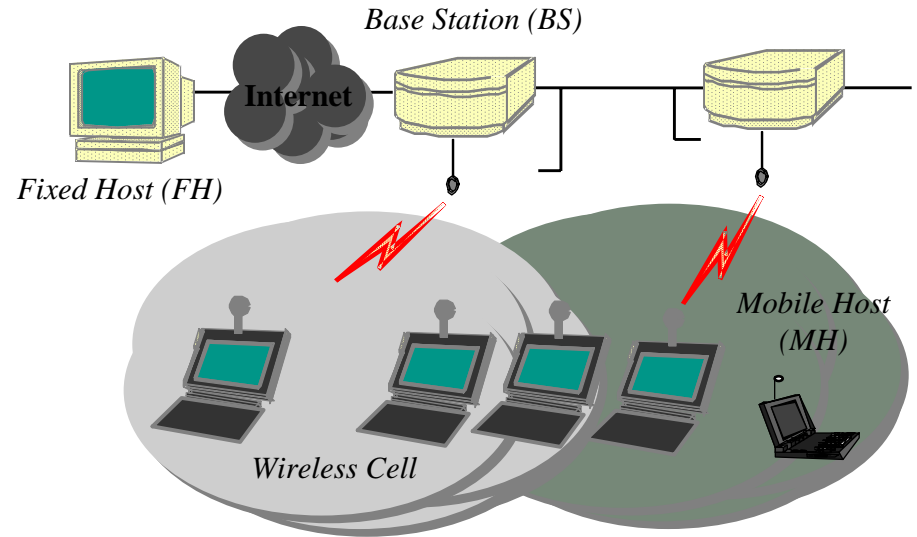
Advanced Topics in Computer Networks

Wireless Transport

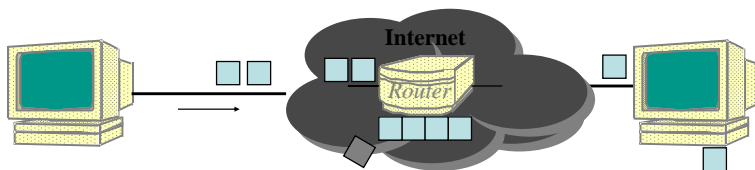
Chalermek Intanagonwiwat

Slides courtesy of Golden G. Richard III, Mary Baker, Nitin Vaidya, and Hari Balakrishnan

Cellular Network Topology



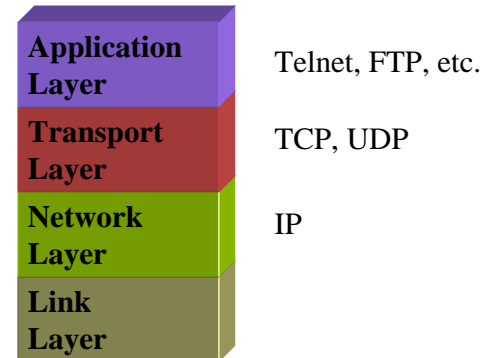
Internet Service Model



A best-effort network: losses & reordering can occur

- Need *reliable* data transport protocols
 - Web, file transfer, remote terminal
- Functions
 - Efficient *loss recovery*
 - Robust *congestion* and *flow control*

TCP/IP Issues



Transmission Control Protocol (TCP)

- Reliable ordered delivery
- Implements congestion avoidance and control
- Reliability achieved by means of retransmissions if necessary

Transmission Control Protocol (cont.)

- End-to-end semantics
 - Acknowledgements sent to TCP sender confirm delivery of data received by TCP receiver
 - Ack for data sent only **after** data has reached receiver

TCP Basics

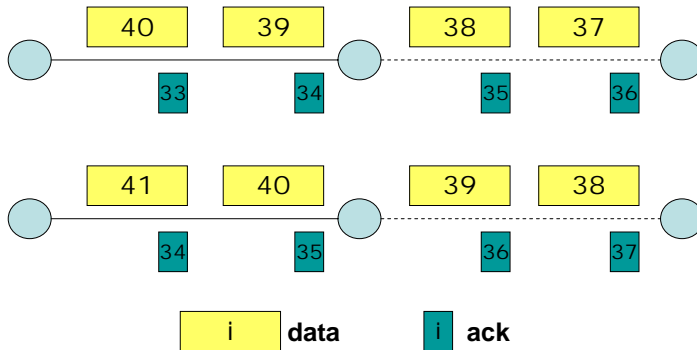
- Cumulative acknowledgements
- An acknowledgement ack's all contiguously received data
- TCP assigns byte sequence numbers
- For simplicity, we will assign packet sequence numbers

TCP Basics (cont.)

- Also, we use slightly different syntax for acks than normal TCP syntax
 - In our notation, *ack i* acknowledges receipt of packets through packet *i*

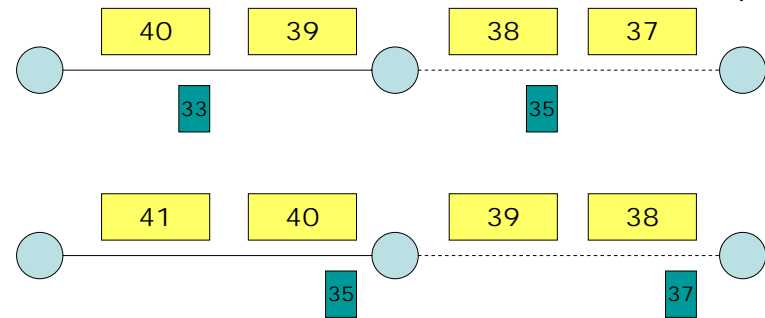
Cumulative Acknowledgements

- A new cumulative acknowledgement is generated only on receipt of a **new in-sequence** packet



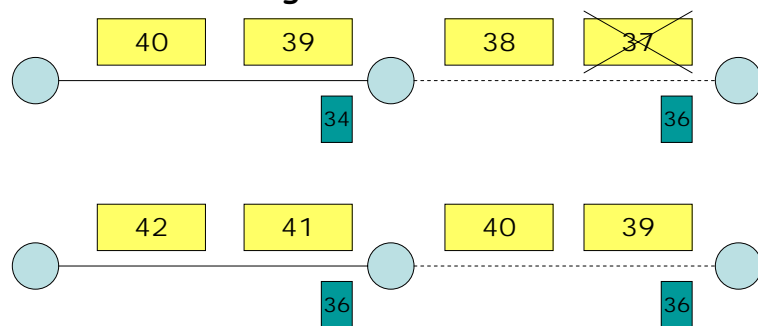
Delayed Acknowledgements

- An ack is delayed until
 - another packet is received, or
 - delayed ack timer expires (200 ms typical)
- Reduces ack traffic
 - New ack not produced on receipt of packet 36, but on receipt of 37



Duplicate Acknowledgements

- A **dupack** is generated whenever an **out-of-order** segment arrives at the receiver

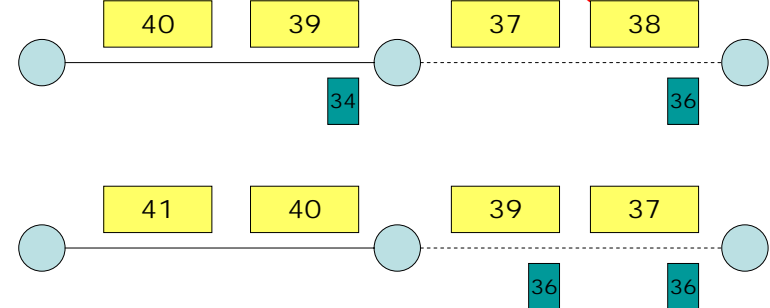


Dupack
On receipt of 38

(Above example assumes *delayed acks*)

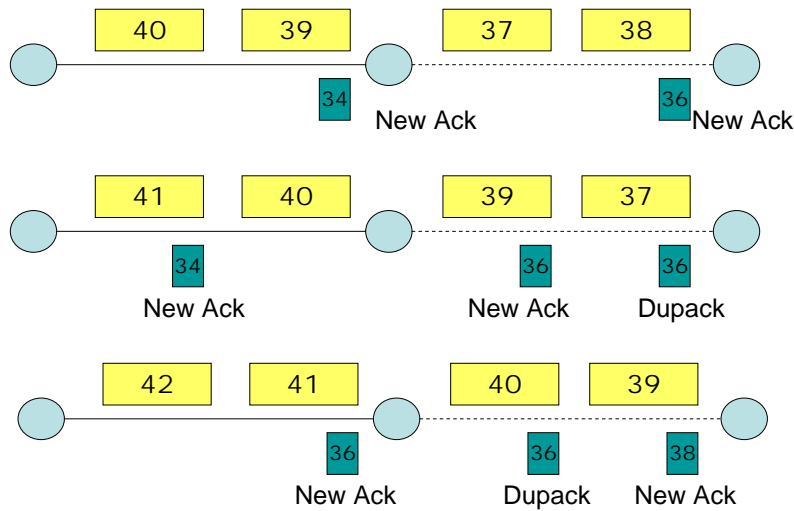
Duplicate Acknowledgements

- Duplicate acks are **not delayed**
- Duplicate acks may be generated when
 - a packet is **lost**, or
 - a packet is delivered **out-of-order (OOO)**

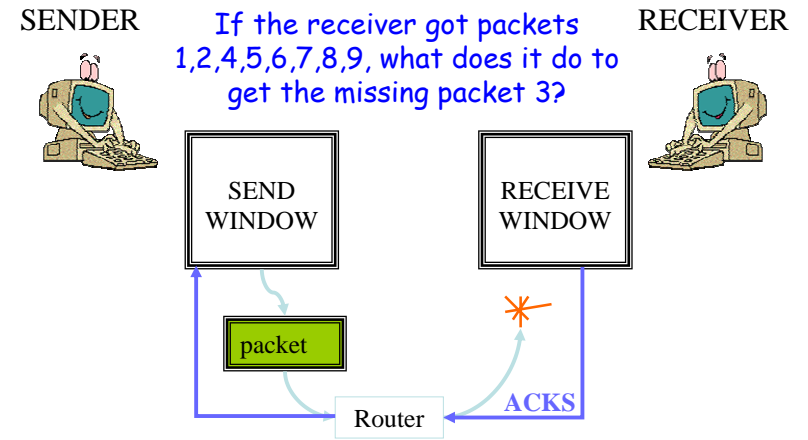


Dupack
On receipt of 38

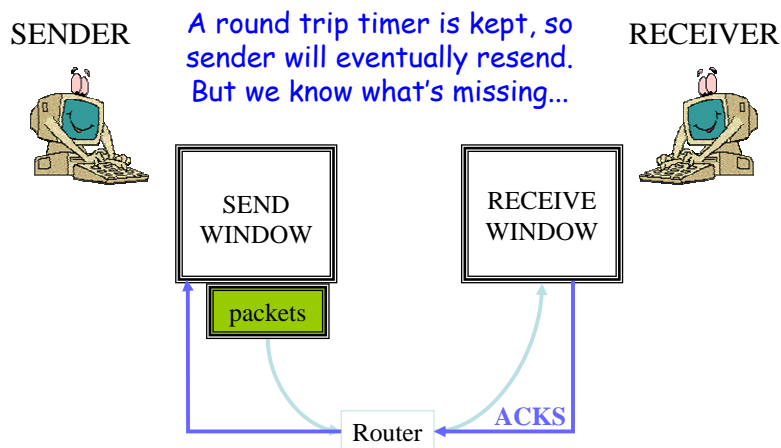
Number of dupacks depends on how much OOO a packet is



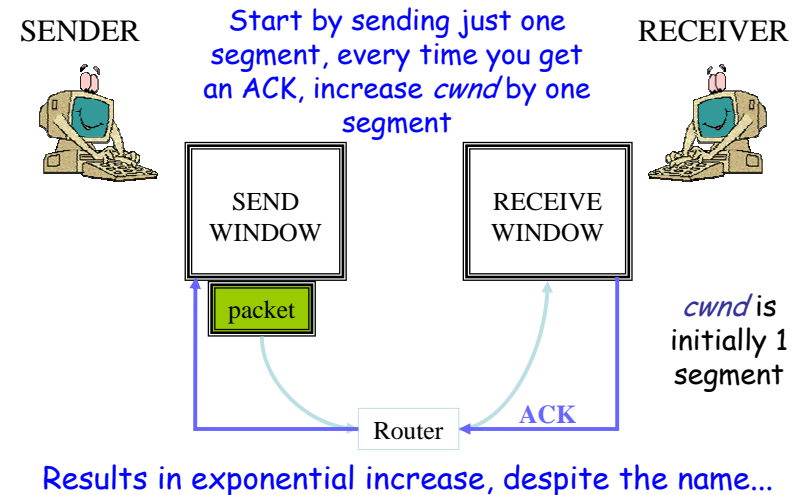
What About Lost Packets?



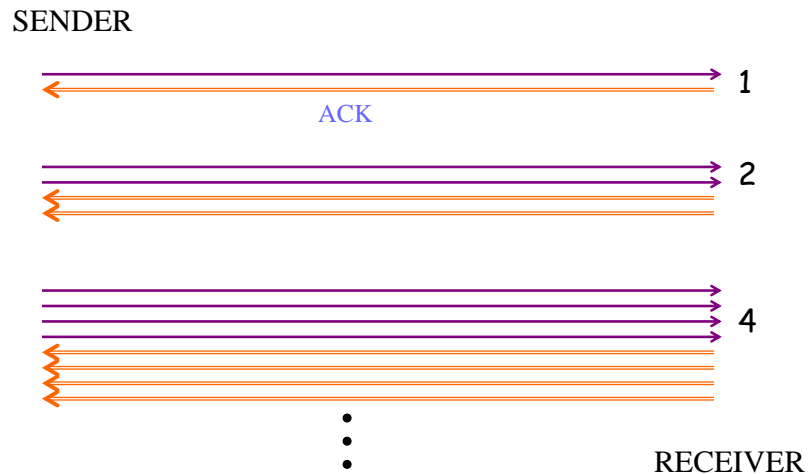
Lost/Reordered Packets...



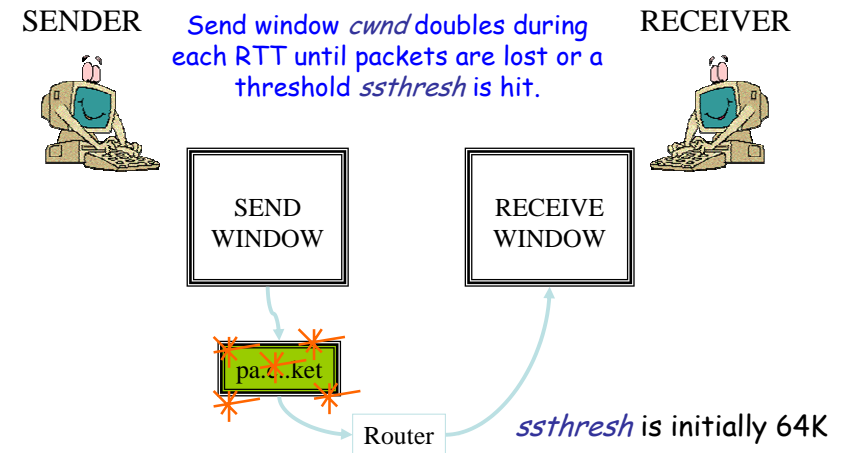
Slow Start



Slow Start == Exponential Increase

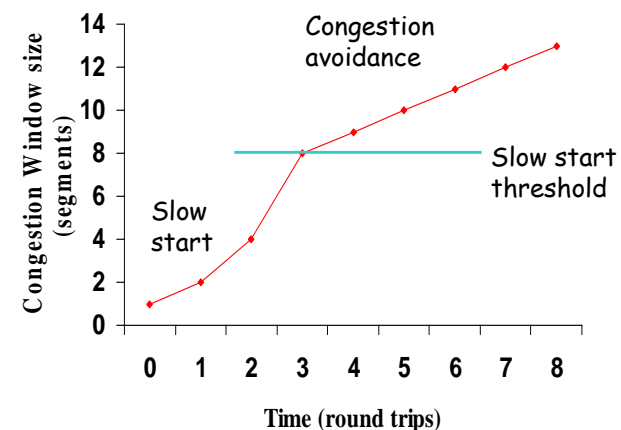


The End of Slow Start...



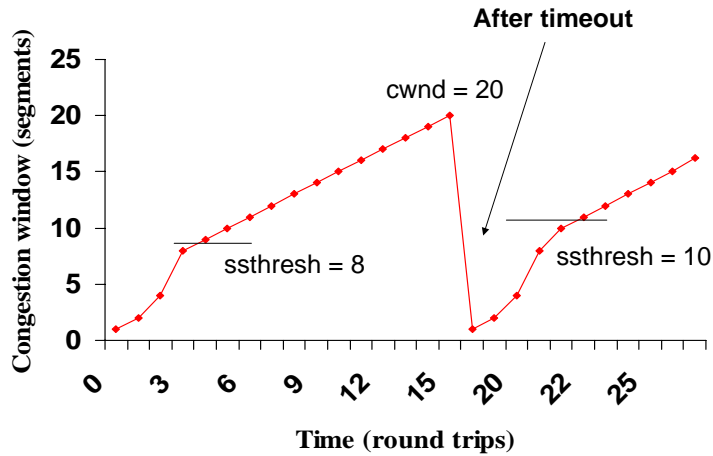
Congestion Avoidance

- When slow start is terminated because *cwnd* exceeds *ssthresh*, increase *cwnd* by a smaller amount on each ACK
- Congestion avoidance phase increases send window more slowly than slow start
- When a packet is lost, TCP assumes it is caused by network congestion (!)
- Set *ssthresh* to $cwnd / 2$, set *cwnd* to 1, restart slow start



Example assumes that acks are not delayed

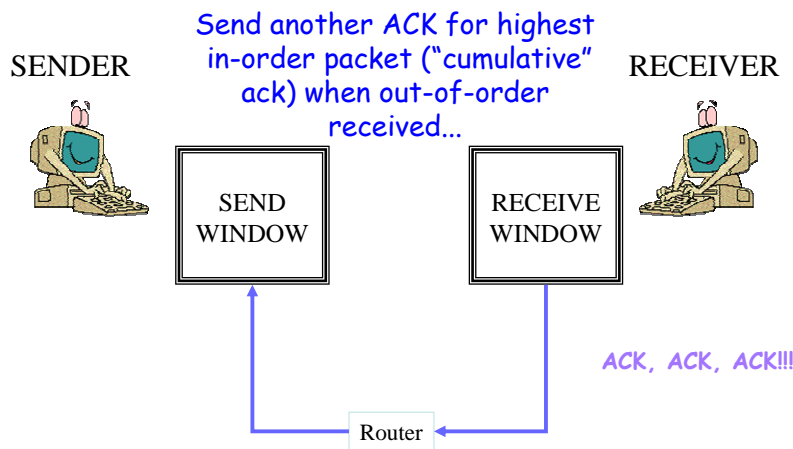
Slowstart after Timeout



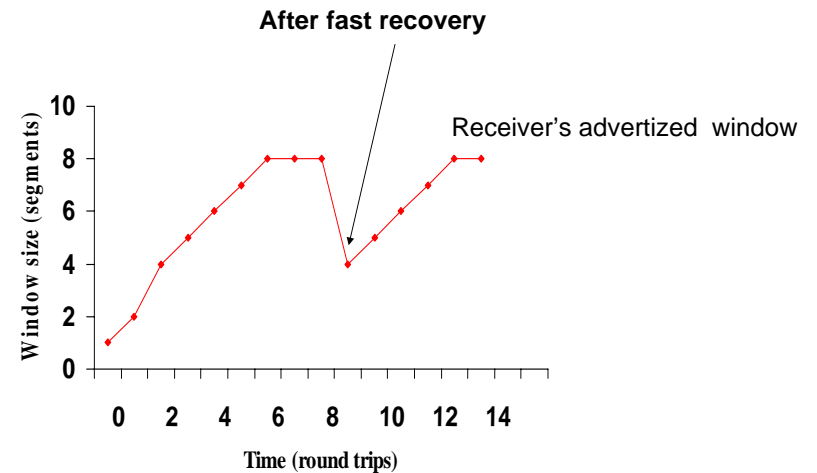
Fast Recovery

- Two ways to detect (possible) packet loss:
 - (1) **Timeout**
 - Result: restart slow start
 - (2) **Duplicate ACKS**
 - Data must still be flowing, because out-of-order packets are being received...
 - Receiver sends duplicate of last in-order ACK
- Fast recovery enters congestion avoidance *without* re-starting slow start for case (2)
- Attempt to keep the pipe full under moderate congestion

Fast Retransmit



THREE duplicate ACKs received will result in retransmission—three to avoid possibility of just out-of-order packets



After fast retransmit and fast recovery window size is reduced in half.

TCP Reno

- Slow-start
- Congestion avoidance
- Fast retransmit
- Fast recovery

Fast Recovery

Fast recovery can result in a timeout with multiple losses per RTT.

- **TCP New-Reno** [Hoe96]
 - stay in fast recovery until all packet losses in window are recovered
 - **can recover 1 packet loss per RTT without causing a timeout**
- **Selective Acknowledgements (SACK)** [mathis96rfc2018]
 - provides information about out-of-order packets received by receiver
 - **can recover multiple packet losses per RTT**

Problem overview

- Packet loss in wireless networks may be due to
 - Bit errors
 - Handoffs
 - Congestion (rarely)
 - Reordering (rarely, except for certain types of wireless nets)

Problem overview (cont.)

- TCP assumes packet loss is due to
 - Congestion
 - Reordering (rarely)
- TCP's congestion responses are triggered by wireless packet loss (which normally has nothing to do with congestion)

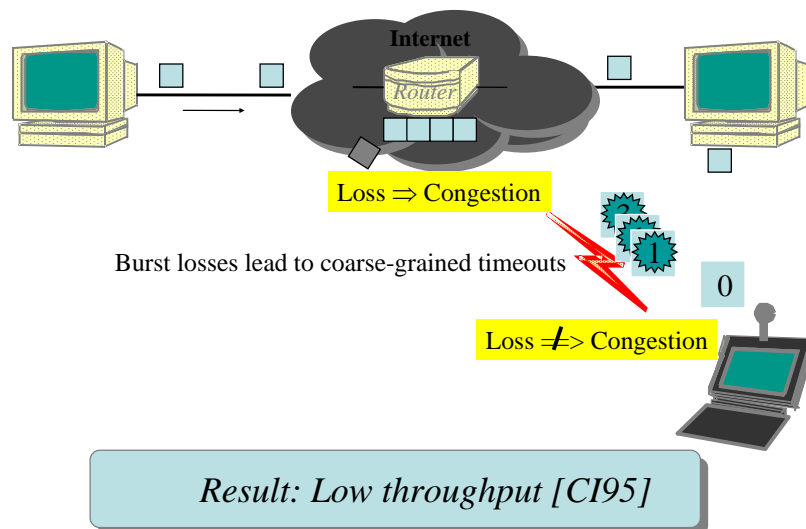
Other problems in a wireless environment

- There are often bursts of errors due to poor signal strength in an area or duration of noise
 - More than one packet lost in TCP window
- Delay is often very high, although you usually only hear about low bandwidth
 - RTT quite long
 - Want to avoid request/response behavior

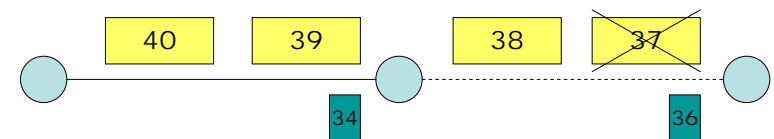
Wireless Bit Errors

- If number of errors is small, they may be corrected by an error correcting code
- Excessive bit errors result in a packet being discarded, possibly before it reaches the transport layer

Wireless Bit Errors (cont.)

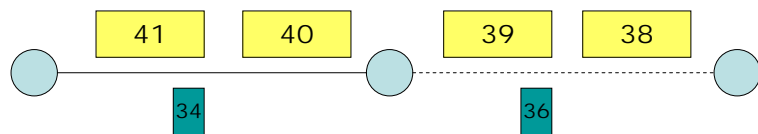


Bit Errors May Cause Fast Retransmit



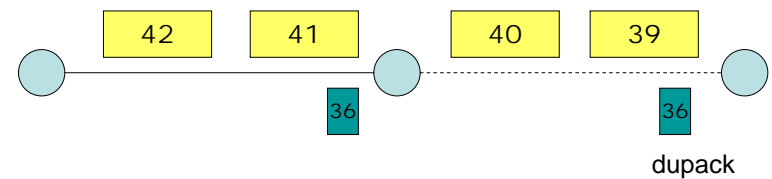
Example assumes delayed ack - every other packet ack'd

Bit Errors May Cause Fast Retransmit (cont.)



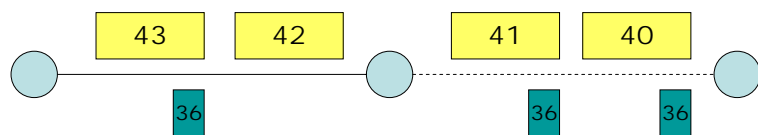
Example assumes delayed ack - every other packet ack'd

Bit Errors May Cause Fast Retransmit (cont.)



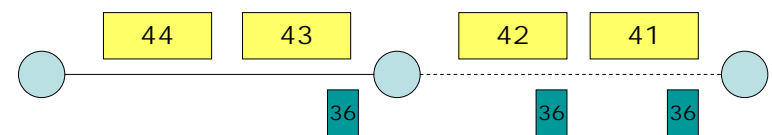
Duplicate acks are not delayed

Bit Errors May Cause Fast Retransmit (cont.)



Duplicate acks

Bit Errors May Cause Fast Retransmit (cont.)



3 duplicate acks trigger fast retransmit at sender

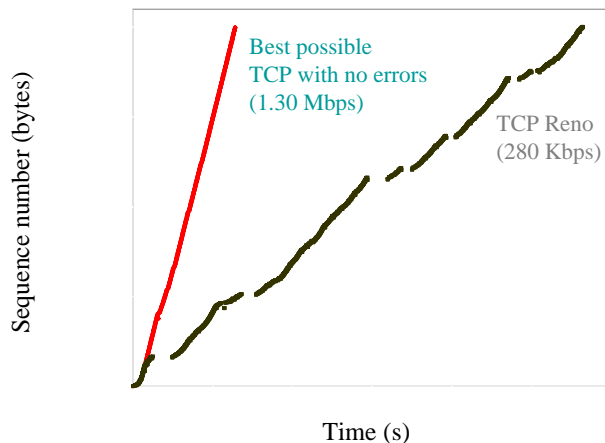
Bit Errors May Cause Fast Retransmit (cont.)

- Fast retransmit results in
 - retransmission of lost packet
 - reduction in congestion window
- Reducing congestion window in response to errors is **unnecessary**
- Reduction in congestion window **reduces the throughput**

Burst Errors May Cause Timeouts

- If wireless link remains unavailable for extended duration, a window worth of data may be lost
 - driving through a tunnel
 - passing a truck
- Timeout results in slow start
- Slow start reduces congestion window to 1 MSS, reducing throughput
- Reduction in window in response to errors **unnecessary**

Performance Degradation



2 MB wide-area TCP transfer over 2 Mbps Lucent WaveLAN

Sometimes Congestion Response May be Appropriate in Response to Errors

- On a CDMA channel, errors occur due to **interference from other user**, and due to **noise [Karn99pilc]**
 - Interference due to other users is an indication of congestion. If such interference causes transmission errors, it is appropriate to reduce congestion window
 - If noise causes errors, it is not appropriate to reduce window

Sometimes Congestion Response May be Appropriate in Response to Errors (cont.)

- When a channel is in a bad state for a **long duration**, it might be better to let TCP backoff, so that it does not unnecessarily attempt retransmissions while the channel remains in the bad state
[Padmanabhan99pic]

For this Class

- We consider errors for which reducing congestion window is an **inappropriate** response

Solution categories

- Entirely new transport protocol
 - Hard to deploy widely
 - End-to-end protocol needs to be efficient on wired networks too
 - Must implement much of TCP's flow control
- Modifications to TCP
 - Maintain end-to-end semantics
 - May or may not be backwards compatible

Solution categories (cont.)

- Split-connection TCP
 - Breaks end-to-end nature of protocol
 - May be backwards compatible with end-hosts
 - State on basestation may make handoffs slow
 - Extra TCP processing at basestation

Solution categories (cont.)

- Link-layer protocols
 - Invisible to higher-level protocols
 - Does not break higher-level end-to-end semantics
 - May not shield sender completely from packet loss
 - May adversely interact with higher-level mechanisms
 - May adversely affect delay-sensitive applications

Solution categories (cont.)

- Snoop protocol
 - Does not break end-to-end semantics
 - Like a LL protocol, does not completely shield sender
 - Only soft state at base station - not essential for correctness

Overall points

- Key performance improvements:
 - Knowledge of multiple losses in window
 - Keeping congestion window from shrinking
 - Maybe even avoiding unnecessary retransmissions
- Two basic approaches
 - Shield sender from wireless nature of link so it doesn't react poorly
 - Make sender aware of wireless problems so it can do something about it

Fast Retransmission for Loss Recovery during Handoff [Caceres95]

- During the long delay for a handoff to complete, a whole window worth of data may be lost
- After handoff is complete, acks are not received by the TCP sender
- Sender eventually times out, and retransmits

Fast Retransmission for Loss Recovery during Handoff [Caceres95] (cont.)

- If handoff still not complete, another timeout will occur
- Performance penalty
 - Time wasted until timeout occurs
 - Window shrunk after timeout

Caceres: Testbed

Initiate TCP connections between MH (mobile host) and SH (stationary host)

MH moves between cells

In experiments, mobility is actually handled in software—the MH can really see both base stations, but software forces the MH to “forget” that it can see one base and handoff to another

Much easier to setup, as you can imagine...

Doesn't require experimenters to physically move equipment

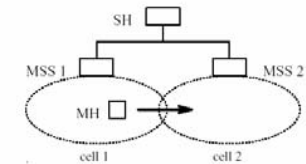
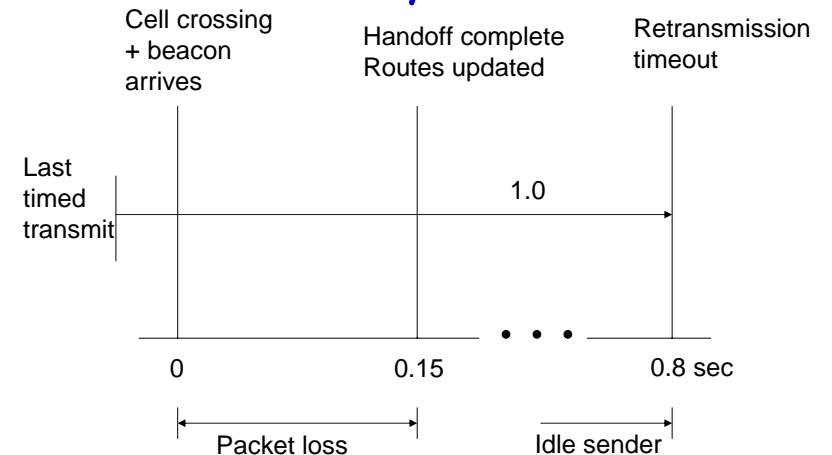


Fig. 1. Wireless networking testbed

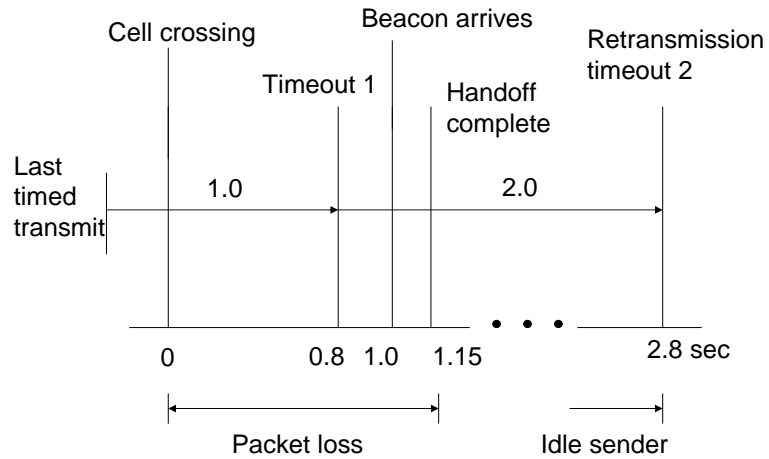
Scenarios

- NO HANDOFF
 - Mobile unit stays in one place
 - Base case
- OVERLAPPING CELLS
 - Mobile unit moves between access points, but remains in contact with “old” base station until relationship with “new” is established
- NON-OVERLAPPING, 0-second RENDEZVOUS DELAY
 - Mobile unit loses contact with “old” base station during movement, but does not need to wait for beacon from “new” base station
 - Base case for non-overlapping coverage
- NON-OVERLAPPING, 1-second RENDEZVOUS DELAY
 - Mobile unit loses contact with old base station during movement and must wait 1 second for a beacon before establishing connection with new base station

0-second Rendezvous Delay : Beacon arrives as soon as cell boundary crossed



1-second Rendezvous Delay : Beacon arrives 1 second after cell boundary crossed



Initial Observations

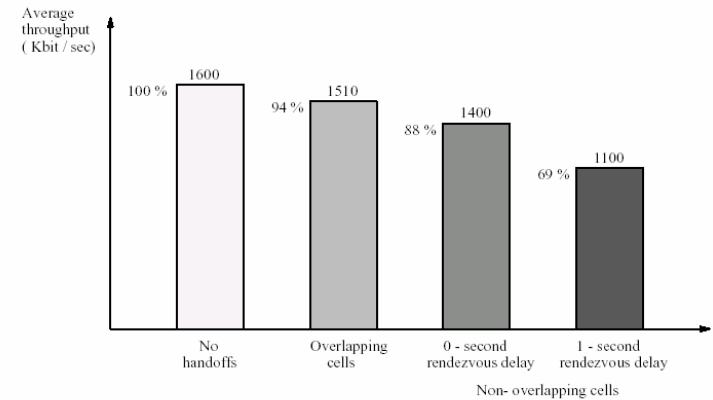


Fig. 2. Loss of throughput due to host motion

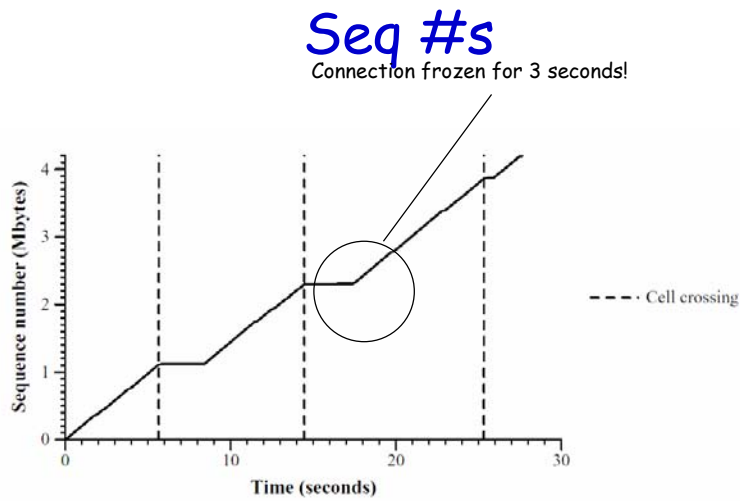


Fig. 3. Behavior of TCP sequence number in response to cell boundary crossings

(1 second rendezvous delay case)

Congestion Window

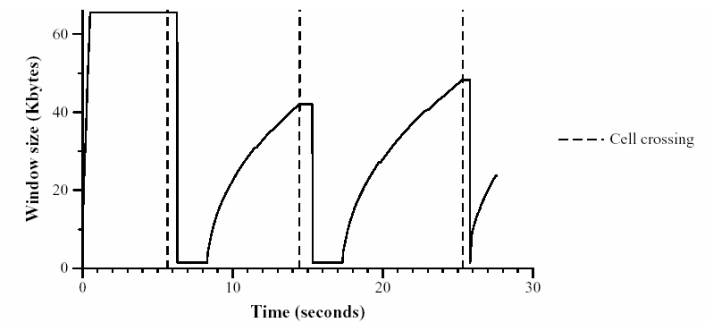


Fig. 4. Behavior of TCP congestion window in response to cell boundary crossings

Non-overlapping cells with 1sec rendezvous delay, movement every 8sec

Dips are a result of the slow-start algorithm being triggered
Delays == BAD for interactive users!

TCP Performance

- Degradation in case 2 (overlapping cells) is due to *encapsulation and forwarding delay* during handoff
- Additional degradation in cases 3 and 4 due to packet loss and idle time at sender

Mitigation Using Fast Retransmit

- When MH is the TCP receiver: after handoff is complete, it sends 3 dupacks to the sender
 - this triggers fast retransmit at the sender
 - instead of dupacks, a special notification could also be sent
- When MH is the TCP sender: invoke fast retransmit after completion of handoff

Forcing Fast Retransmit

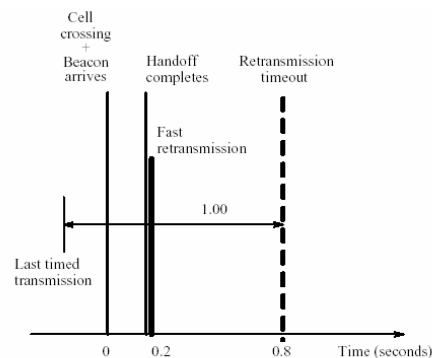


Fig. 7. Fast retransmission after a handoff with a 0-second rendezvous delay

Hack network stack to force fast retransmission immediately after handoff is complete. Makes connection resume operation almost 0.6sec faster!!

More Dramatic w/ 1sec Delay

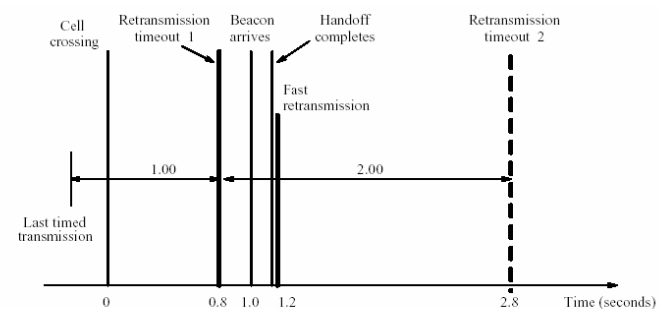


Fig. 8. Fast retransmission after a handoff with a 1-second rendezvous delay

Exponential backoff of retransmission time because of **multiple expirations** of retransmission timer. Fast retransmit really helps in this case.

Delay Drops, Too

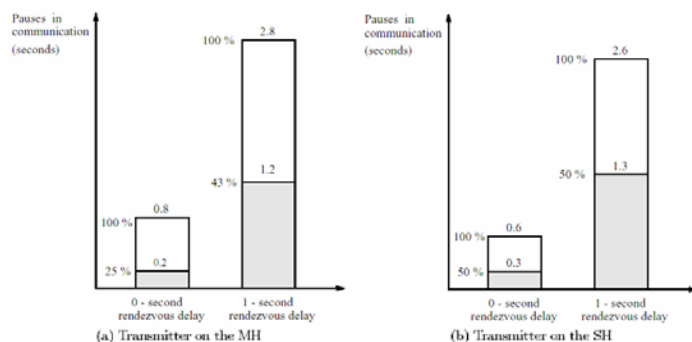


Fig. 9. Improvements in latency due to fast retransmissions

Improving Performance by Smooth Handoffs [Caceres95]

- Provide sufficient overlap between cells to avoid packet loss
- or
- Buffer packets at BS
 - Discard the packets after a short interval
 - If handoff occurs before the interval expires, forward the packets to the new base station
 - Prevents packet loss on handoff

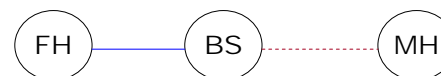
Snoop Protocol

- Shield TCP sender from wireless vagaries
 - Eliminate adverse interactions between protocol layers
 - Congestion control only when congestion occurs
- The End-to-End Argument [SRC84]
 - Preserve TCP/IP service model: end-to-end semantics
 - *Is connection splitting fundamentally important?*
- Eliminate non-TCP protocol messages

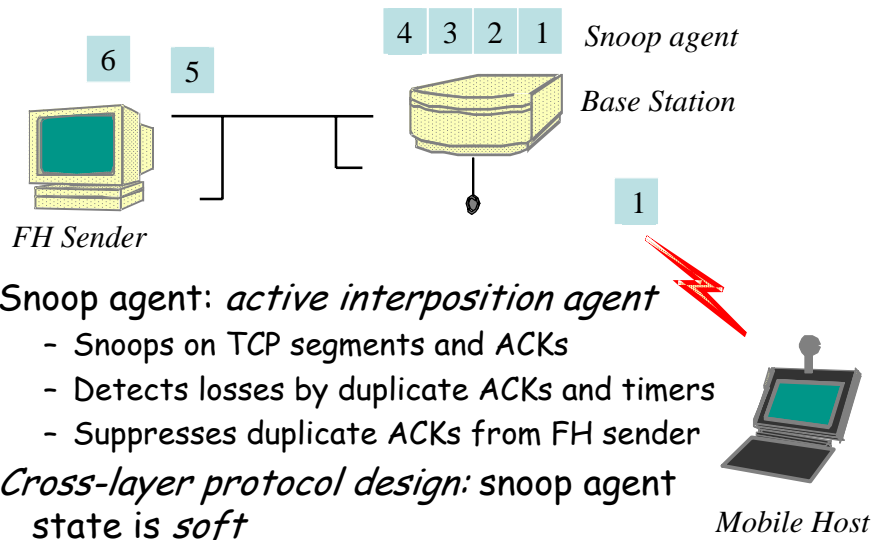
Fixed to mobile: transport-aware link protocol
Mobile to fixed: link-aware transport protocol

Snoop Protocol

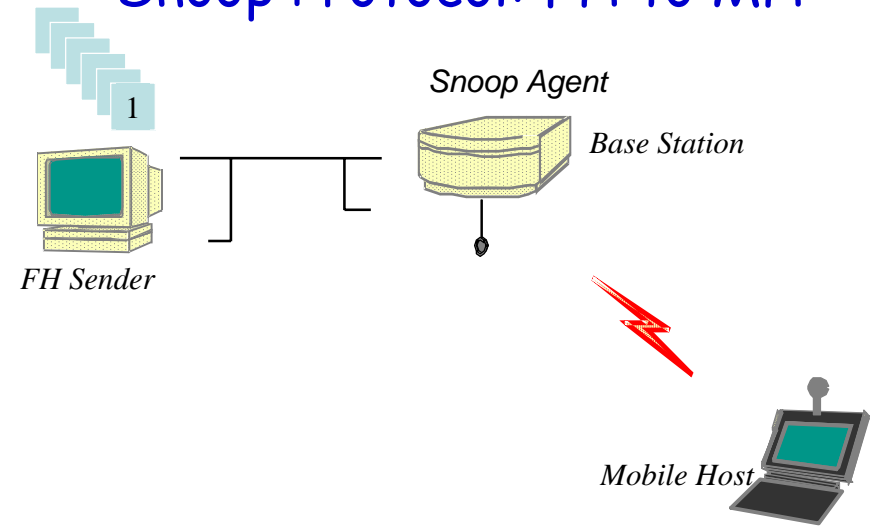
- Buffers data packets at the base station BS
 - to allow link layer retransmission
- When dupacks received by BS from MH, retransmit on wireless link, if packet present in buffer
- Prevents fast retransmit at TCP sender FH by dropping the dupacks at BS



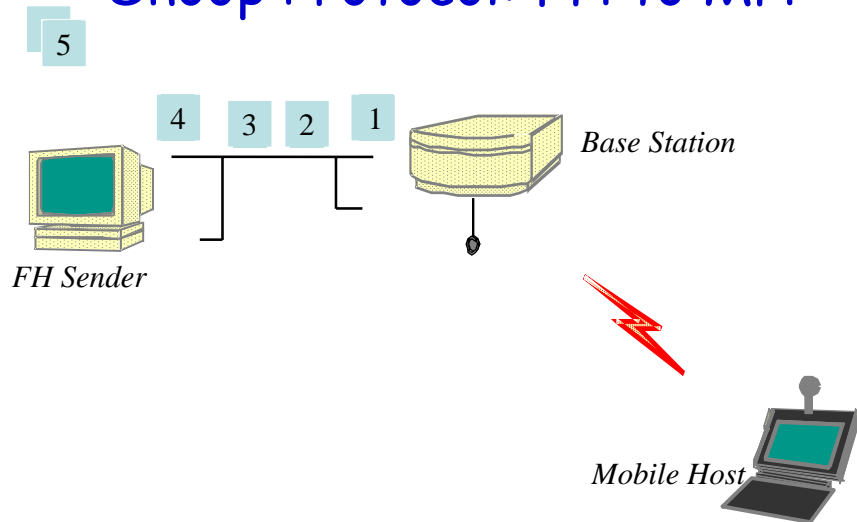
Snoop Protocol: FH to MH



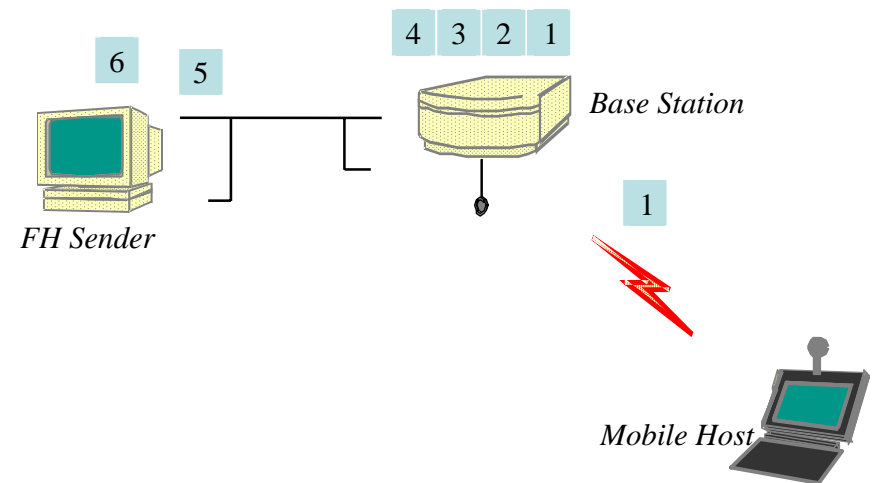
Snoop Protocol: FH to MH



Snoop Protocol: FH to MH

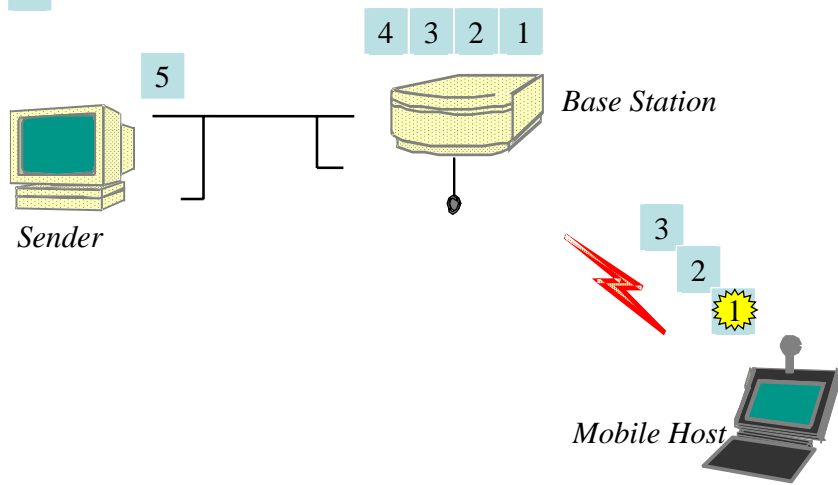


Snoop Protocol: FH to MH

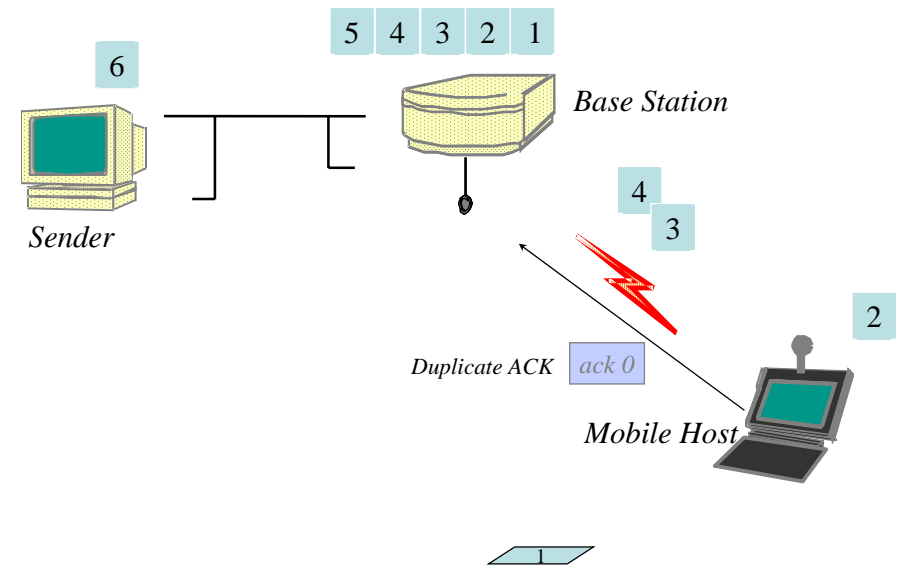


Snoop Protocol: FH to MH

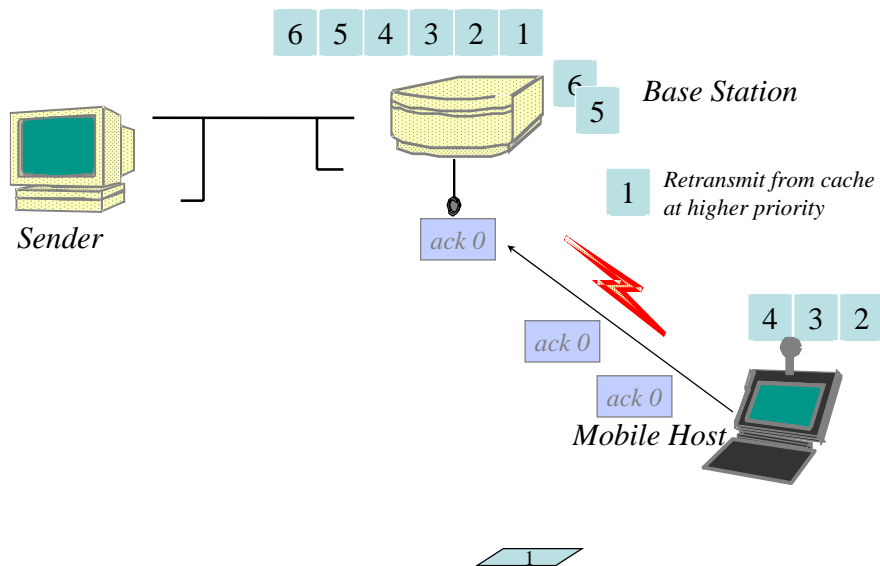
6



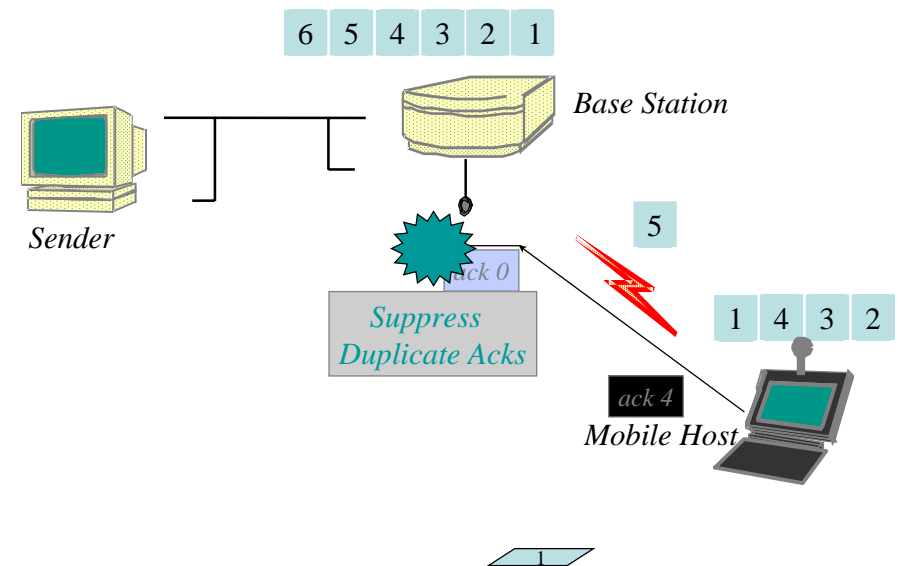
Snoop Protocol: FH to MH



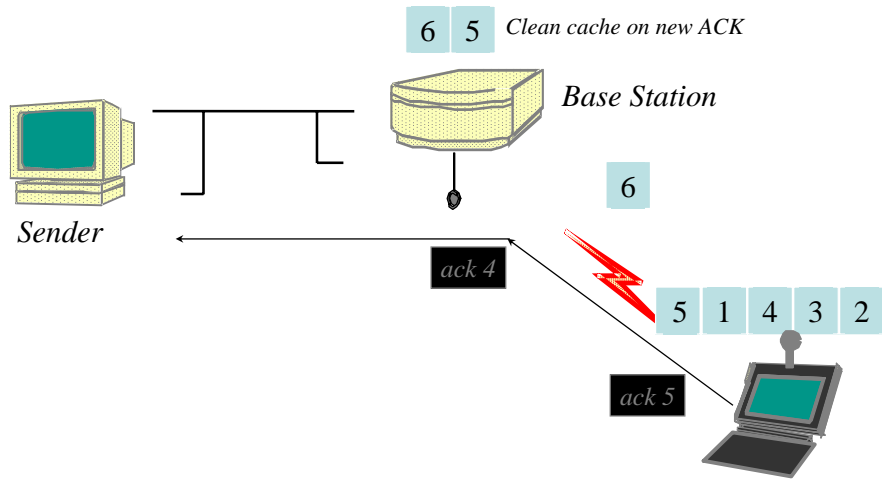
Snoop Protocol: FH to MH



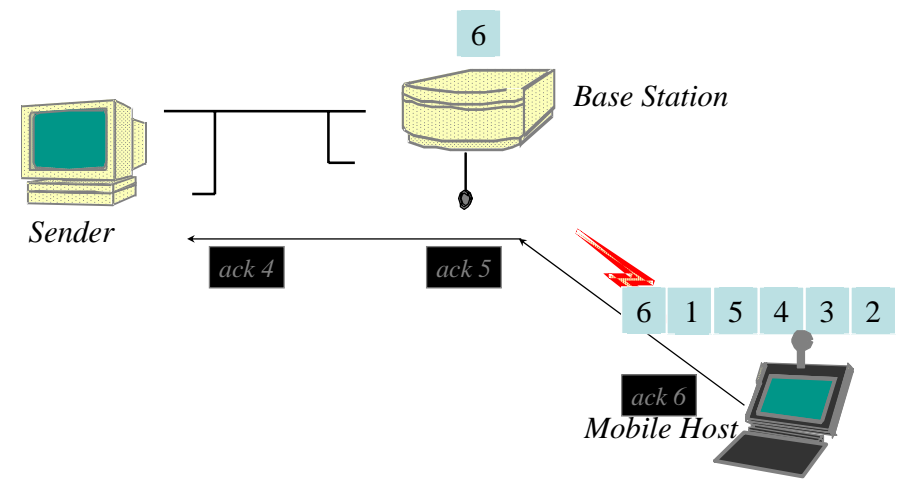
Snoop Protocol: FH to MH



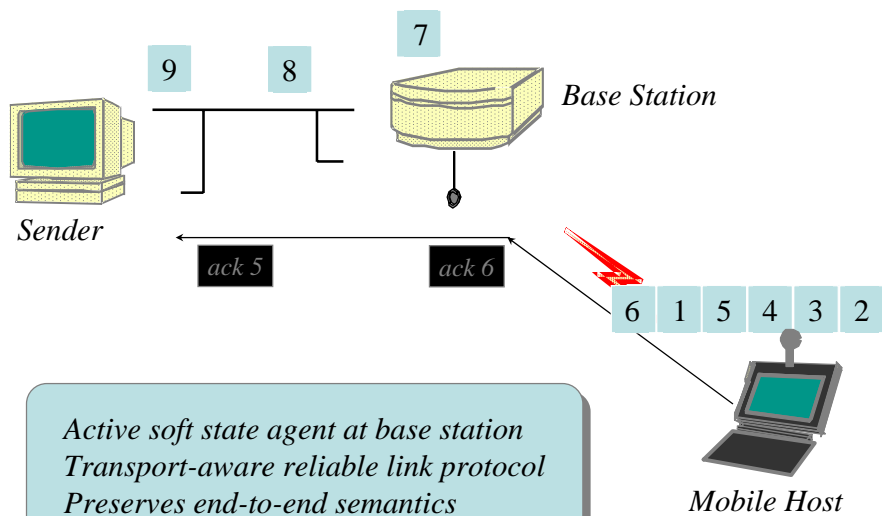
Snoop Protocol: FH to MH



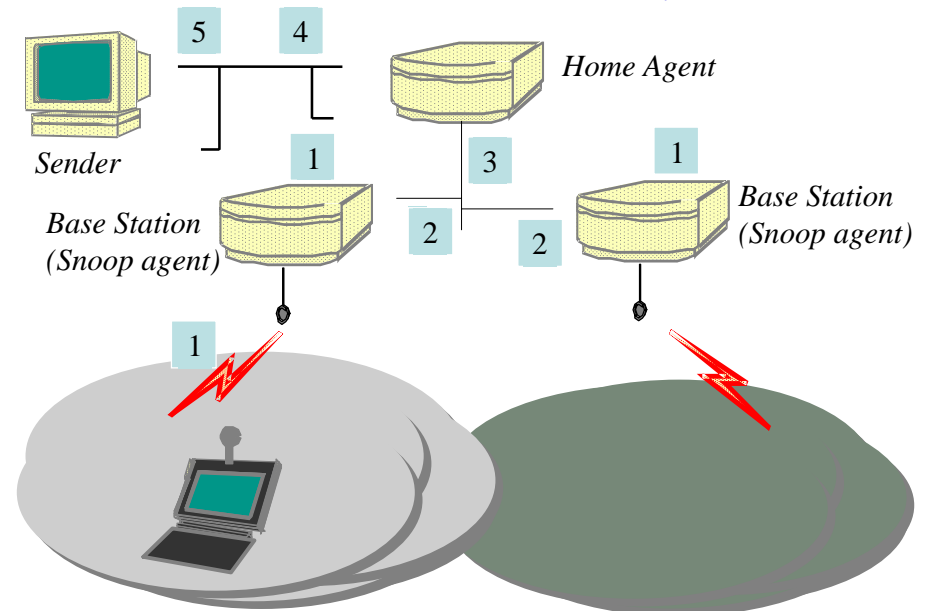
Snoop Protocol: FH to MH



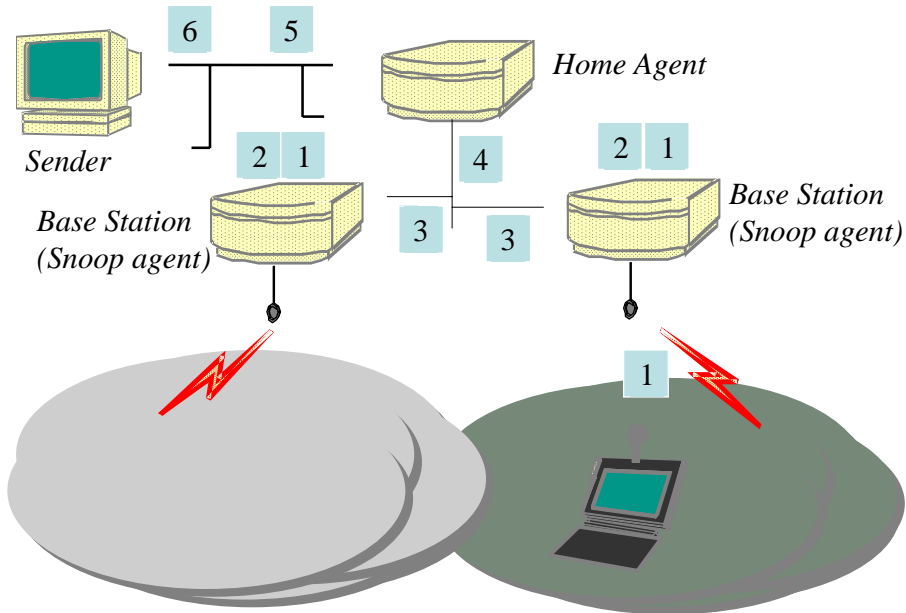
Snoop Protocol: FH to MH



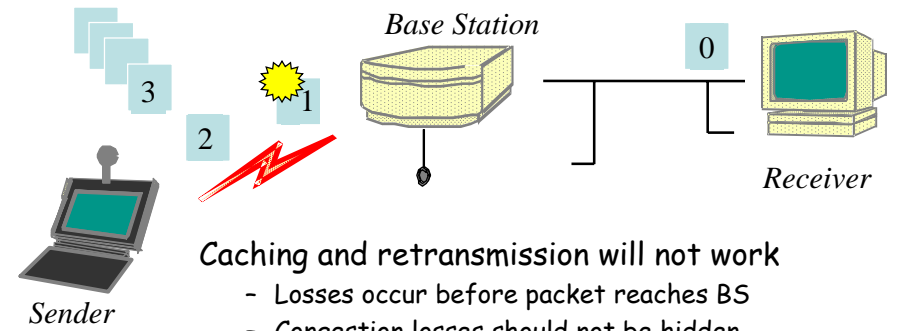
Handling Mobility



Handling Mobility



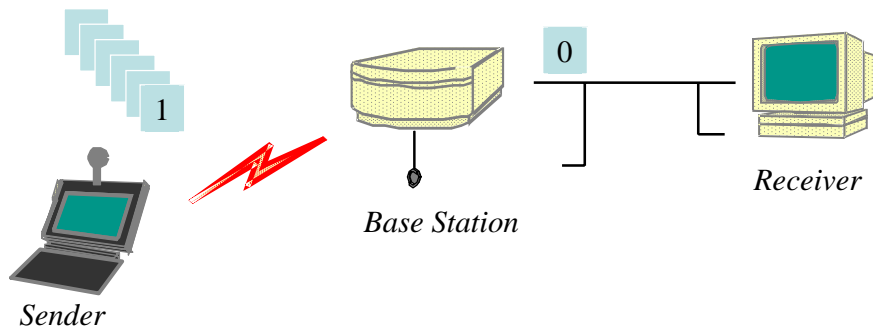
Snoop Protocol: MH to FH



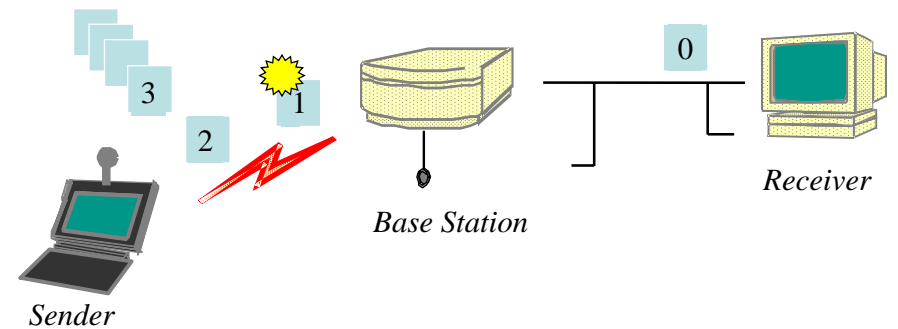
Solution: *Explicit Loss Notifications (ELN)*

- In-band message to TCP sender
- General solution framework

Snoop Protocol: MH to FH

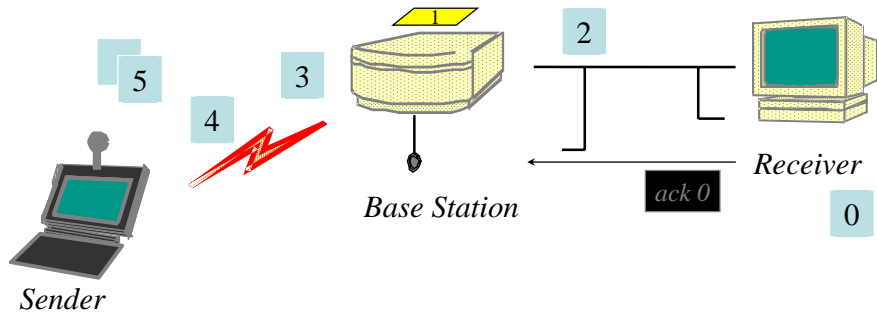


Snoop Protocol: MH to FH

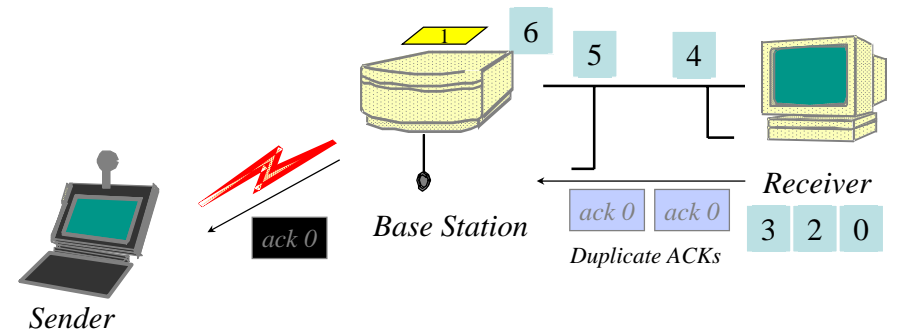


Snoop Protocol: MH to FH

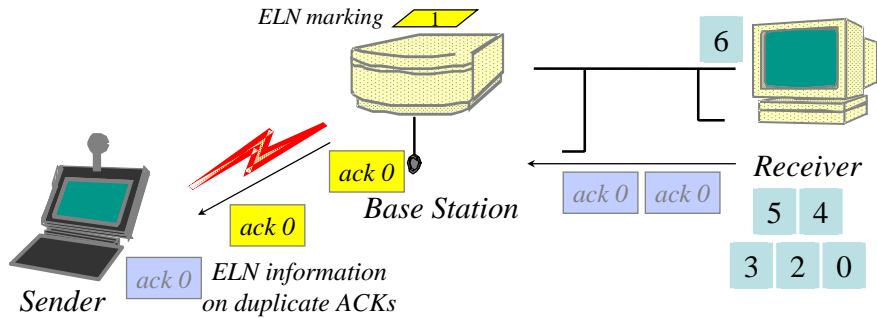
Add 1 to list of holes after checking for congestion



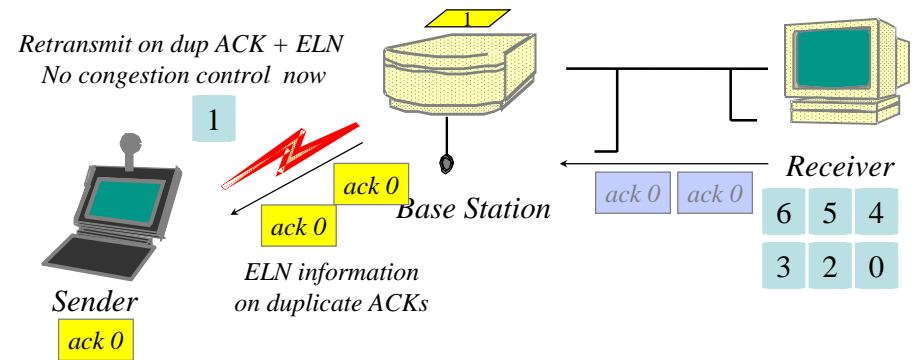
Snoop Protocol: MH to FH



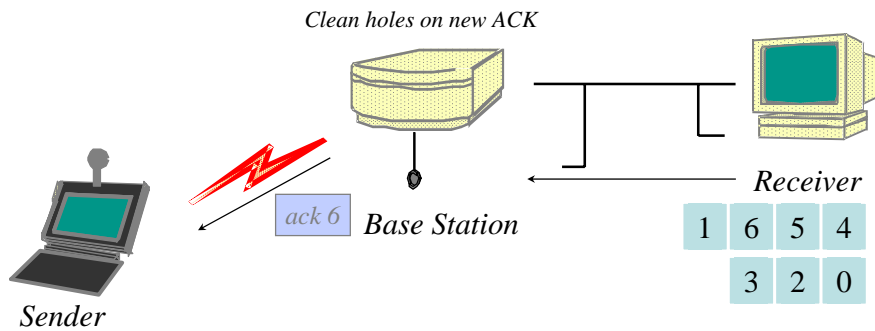
Snoop Protocol: MH to FH



Snoop Protocol: MH to FH



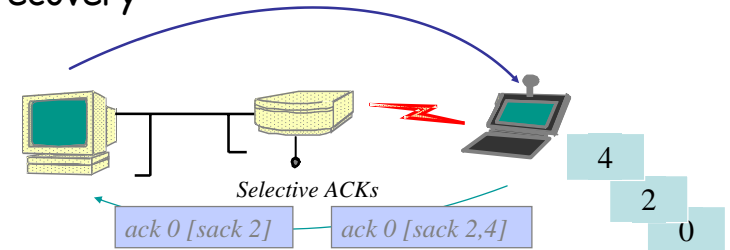
Snoop Protocol: MH to FH



Link-aware transport decouples congestion control from loss recovery
Technique generalizes nicely to wireless transit links

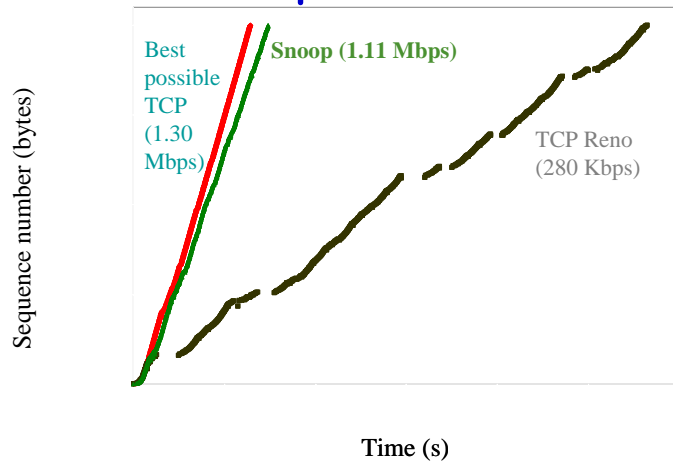
End-to-End Enhancements

- ELN to decouple congestion from loss recovery



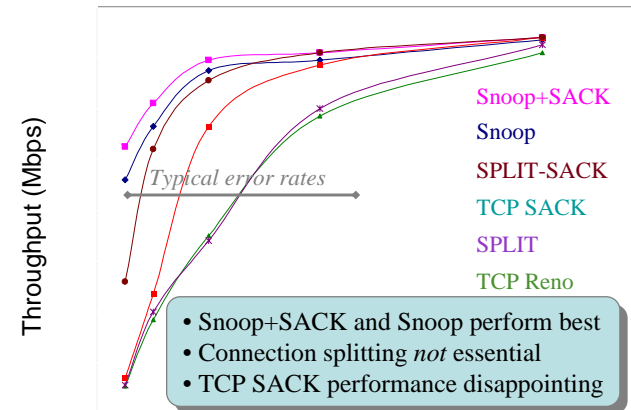
- Selective ACKS (SACK) for burst losses [FF96, KM96, MMFR96, B96]
- Snoop protocol: no changes to fixed hosts on the Internet

Snoop Performance Improvement



2 MB wide-area TCP transfer over 2 Mbps Lucent WaveLAN

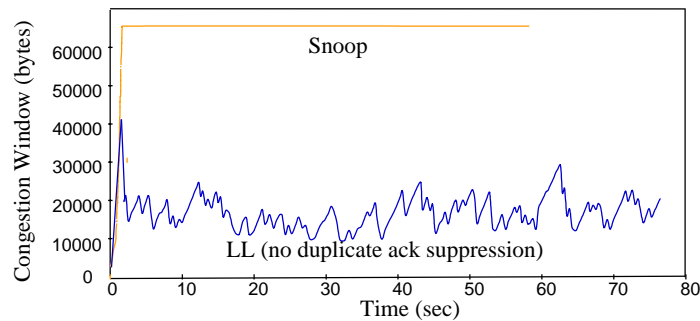
Performance: FH to MH



1/Bit-error Rate (1 error every x Kbits)

2 MB local-area TCP transfer over 2 Mbps Lucent WaveLAN

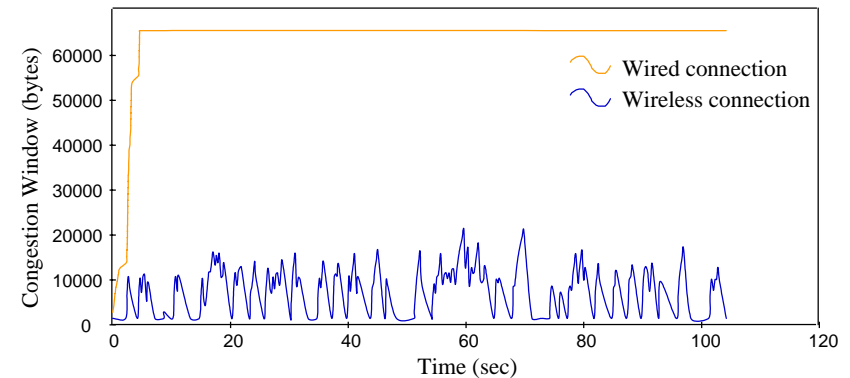
Benefits of TCP-Awareness



- 30-35% improvement for Snoop: LL congestion window is small (but no coarse timeouts occur)
- Connection bandwidth-delay product = 25 KB

Suppressing duplicate acknowledgments and TCP-awareness leads to better utilization of link bandwidth and performance

Split-Connection Congestion Window



Wired connection does not shrink congestion window but wireless connection times out often, causing sender to stall

Snoop Protocol When Beneficial?

- Snoop prevents fast retransmit from sender despite transmission errors, and out-of-order delivery on the wireless link
- OOO delivery causes fast retransmit **only if** it results in at least 3 dupacks

Snoop Protocol When Beneficial?

- If wireless link level delay-bandwidth product is less than 4 packets, a simple (TCP-unaware) link level retransmission scheme can **suffice**
 - Since delay-bandwidth product is small, the retransmission scheme can deliver the lost packet without resulting in 3 dupacks from the TCP receiver

Snoop Protocol : Advantages

- High throughput can be achieved
 - performance further improved using selective acks
- Local recovery from wireless losses
- Fast retransmit not triggered at sender despite out-of-order link layer delivery

Snoop Protocol : Advantages (cont.)

- End-to-end semantics retained
- Soft state at base station
 - loss of the soft state affects performance, but not correctness

Snoop Protocol : Disadvantages

- Link layer at base station needs to be **TCP-aware**
- Not useful if TCP headers are encrypted (IPsec)
- Cannot be used if TCP data and TCP acks traverse different paths (both do not go through the base station)

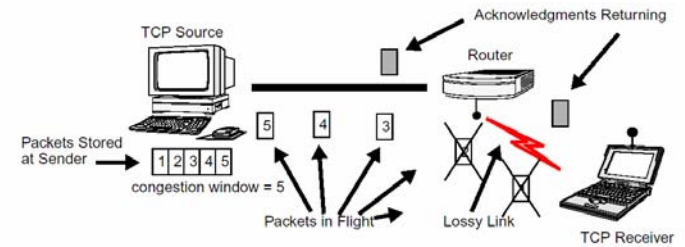
Link layer protocols investigated

- LL: TCP-ish one with cumulative acks and retransmit granularity faster than TCP's
- LL-SACK: Add selective acks
 - Cumulative ack with sequence # of packet causing ack

Link layer protocols investigated (cont.)

- LL-TCP-AWARE: snoop protocol
 - At base station cache segments
 - Detect and suppress duplicate acks
 - Retransmit lost segments locally
- LL-OPT or LL-TCP-AWARE-SACK: Combination of selective acks and duplicate ack suppression

How to Read the Other Diagrams...



Link Layer Approaches

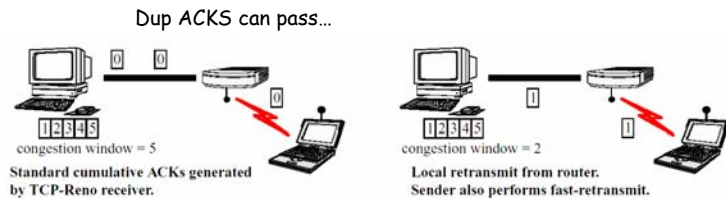


Figure 5. Link-Layer

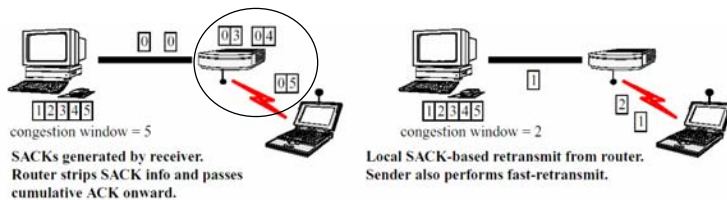


Figure 6. Link-Layer with SACK

More Link Layer

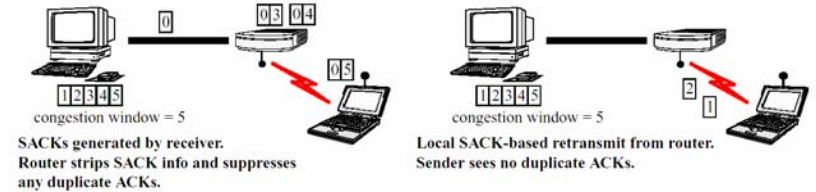


Figure 7. Link-Layer with SACK and TCP awareness

Experimental Setup

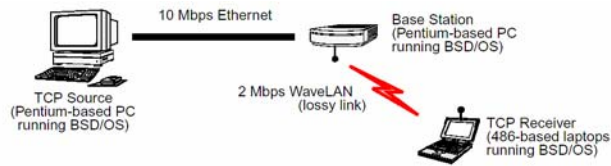
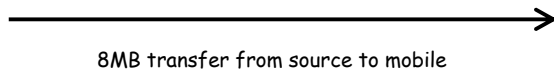


Figure 10. Experimental topology. There were an additional 16 Internet hops between the source and base station during the wide area experiments.



Performance of Link Layer Hacks

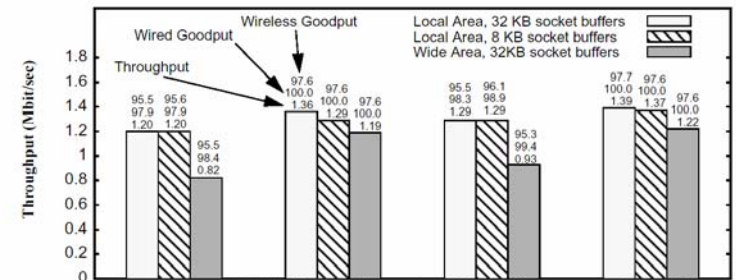


Figure 11. Performance of link-layer protocols: bit-error rate = 1.9×10^{-6} (1 error/65536 bytes).

More Link Layer Performance

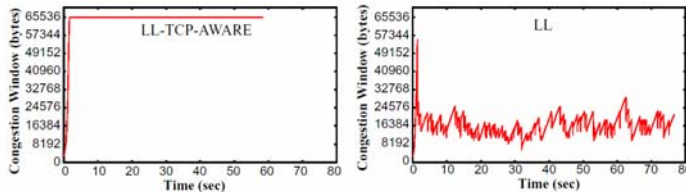


Figure 12. Congestion window size for link-layer protocols in wide area tests.

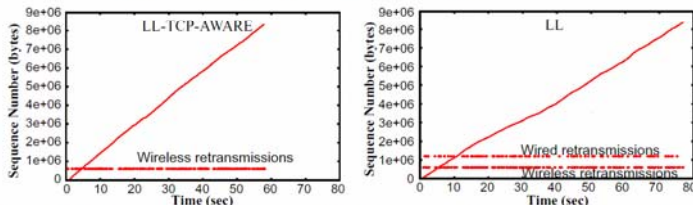


Figure 13. Packet sequence traces for LL-TCP-AWARE and LL. No coarse timeouts occur in either case. For LL-TCP-AWARE, the horizontal row of dots shows the times of wireless link retransmissions. For LL, the top row shows sender fast retransmission times and the bottom row shows both local wireless and sender retransmissions.

Link layer results

- Simple retransmission at link layer helps, but not totally
- Combination of selective acks and duplicate suppression is best
- Duplicate suppression by itself is good

Link layer results (cont.)

- Real problem is link layers that allow out-of-order packet delivery, triggering duplicate acks, fast retransmission and congestion avoidance in TCP
- Overall, want to avoid triggering TCP congestion handling techniques

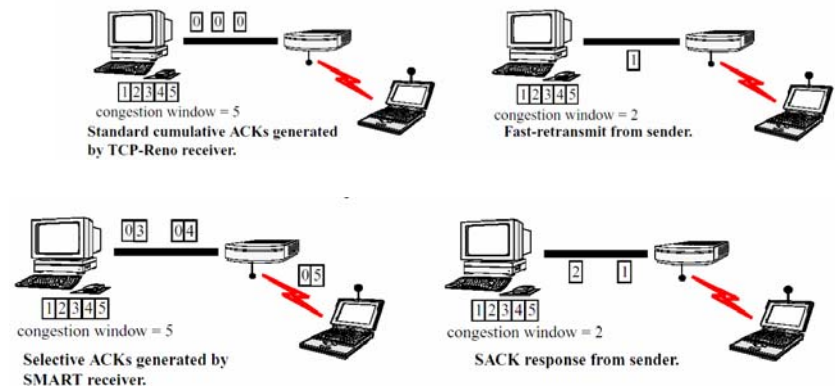
End-to-end protocols investigated

- E2E (Reno): no support for partial acks
- E2E-NewReno: partial acks allow further packet retransmissions
- E2E-SACK: cumulative ack with sequence # of packet causing ack

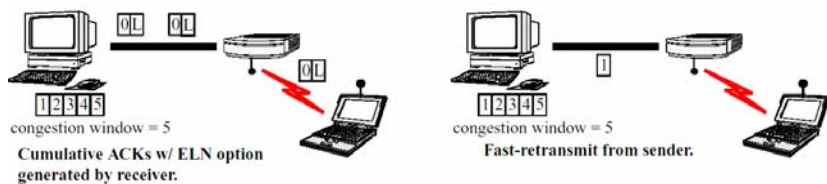
E2E protocols (cont.)

- E2E-ELN: explicit loss notification
 - Future cumulative acks for packet marked to show non-congestion loss
 - Sender gets duplicate acks and retransmits, but does not invoke congestion-related procedures
- E2E-ELN-RXMT: retransmit on first duplicate ack

TCP, TCP w/ SACK



TCP w/ ELN



(Experiments assume perfect knowledge of loss of packets due to bit errors on wireless link!)

Performance of End-to-End Hacks

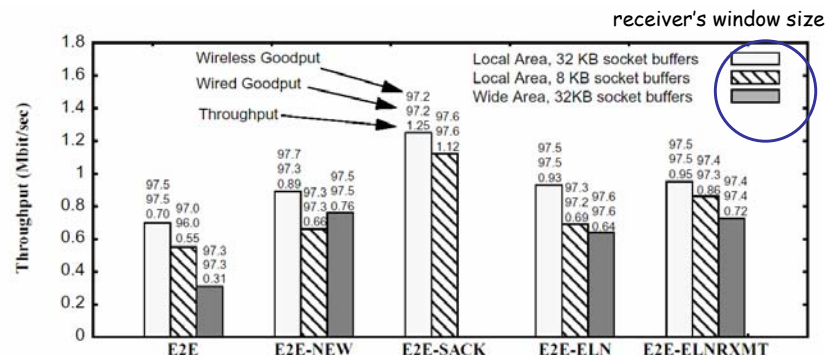


Figure 14. Performance of end-to-end protocols: bit error rate = 1.9×10^{-6} (1 error/65536 bytes).

More E to E Performance

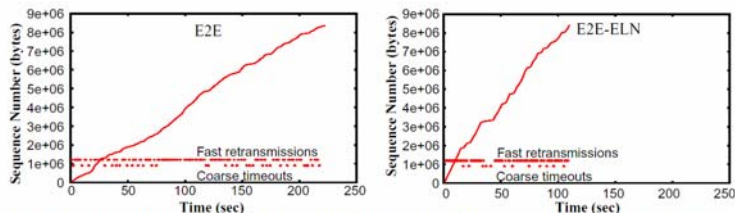


Figure 15. Packet sequence traces for E2E (TCP Reno) and E2E-ELN. The top row of horizontal dots shows the times when fast retransmissions occur; the bottom row shows the coarse timeouts.

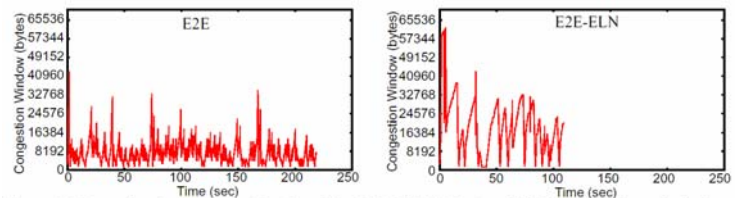


Figure 16. Congestion window size as a function of time for E2E (TCP Reno) and E2E-ELN. This figure clearly shows the utility of ELN preventing rapid fluctuations in the congestion window.

End-to-end results

- E2E (Reno): coarse-grained timeouts really hurt
 - Throughput less than 50% of maximum in local area
 - Throughput of less than 25% in wide area
- E2E-New Reno: avoiding timeouts helps
 - Throughput 10-25% better in LAN
 - Throughput twice as good in WAN

End-to-end results (cont.)

- ELN techniques avoid shrinking congestion window
 - Over two times better than E2E
- E2E-ELN-RXMT only a little better than E2E-ELN
 - Enough data in pipe usually to get fast retransmit from ELN
 - Bigger difference with smaller buffer size
 - Not as much data in pipe (harder to get 3 duplicate acks)

E2E results (cont.)

- E2E selective acks:
 - Over twice as good as E2E
 - Not as good as best LL schemes (10% worse on LAN, 35% worse in WAN)
 - Problem is still shrinkage of congestion window

E2E results (cont.)

- Haven't tried combo of ELN techniques with selective acks
 - ELN implementation in paper still allows timeouts
 - No information about multiple losses in window

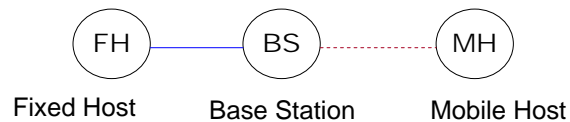
Split Connection Approach

- End-to-end TCP connection is broken into one connection on the wired part of route and one over wireless part of the route
- A single TCP connection split into two TCP connections
 - if wireless link is not last on route, then more than two TCP connections may be needed

Split Connection Approach

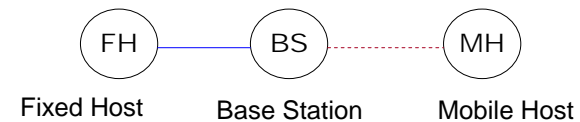
- Connection between wireless host MH and fixed host FH goes through base station BS

• $FH-MH = FH-BS + BS-MH$



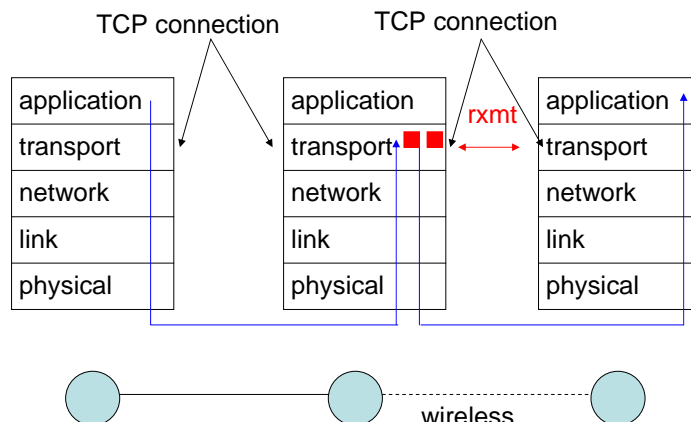
Split Connection Approach

- Split connection results in independent flow control for the two parts
- Flow/error control protocols, packet size, time-outs, may be different for each part



Split Connection Approach

■ Per-TCP connection state



Split connection protocols (cont.)

- SPLIT: uses TCP Reno over wireless link
- SPLIT-SACK: uses SMART-based selective acks

More Split TCP

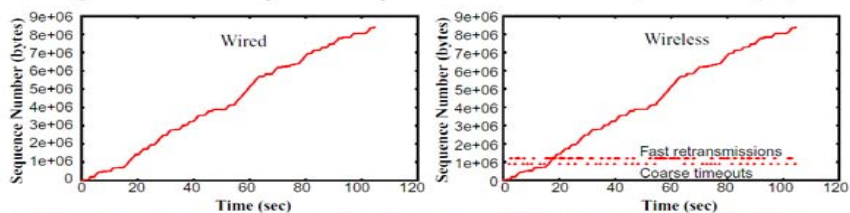


Figure 18. Packet sequence trace for the wired and wireless parts of the SPLIT protocol. The wireless part has two rows of horizontal dots: the top one shows the times of fast retransmissions and the bottom one the times of the timeout-based ones.

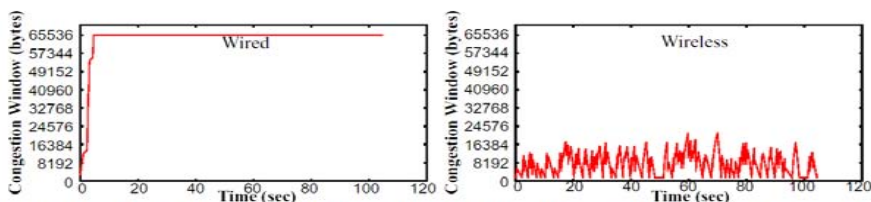
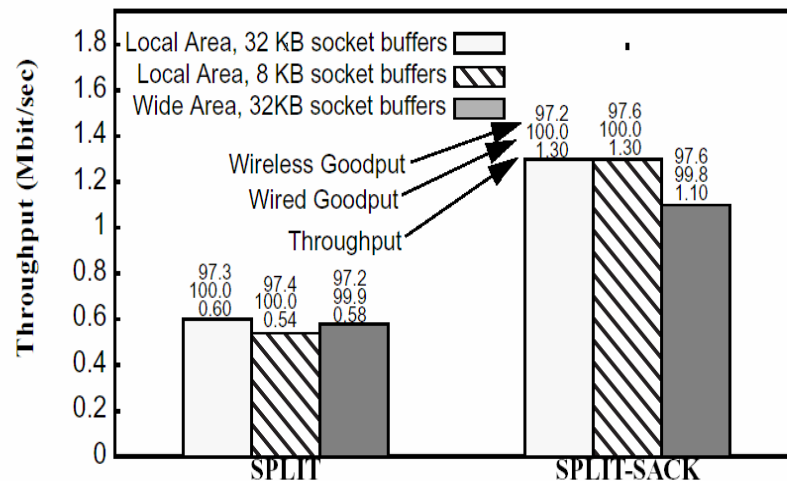


Figure 19. Congestion window sizes as a function of time for the wired and wireless parts of the split TCP connection. The wired sender never sees any losses and maintains a 64 KB congestion window. However, the wireless TCP connection's congestion window fluctuates rapidly.

Split TCP Approach



Split connection results

- SPLIT:
 - Wired goodput 100% since no retransmissions there
 - Eventually stalls when wireless link times out
 - Buffer space limited at base station

Split connection results (cont.)

- SPLIT-SACK:
 - Throughput better than SPLIT (at least twice as good)
 - Better performance of wireless link avoids holding up wired links as much
- Split connections not as effective as TCP-aware LL protocol, which also avoids splitting the connection

Split Connection Approach : Advantages

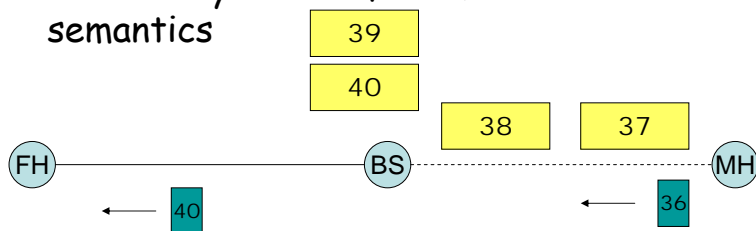
- BS-MH connection can be **optimized** independent of FH-BS connection
 - Different flow / error control on the two connections
- **Local recovery** of errors
 - **Faster** recovery due to relatively shorter RTT on wireless link

Split Connection Approach : Advantages

- **Good performance** achievable using **appropriate** BS-MH protocol
 - Standard TCP on BS-MH performs poorly when multiple packet losses occur per window (timeouts can occur on the BS-MH connection, stalling during the timeout interval)
 - **Selective acks improve performance** for such cases

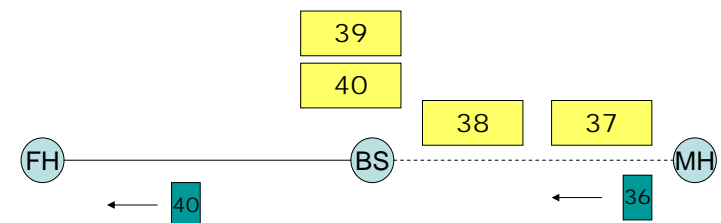
Split Connection Approach : Disadvantages

- End-to-end **semantics** violated
 - ack may be delivered to sender, before data delivered to the receiver
 - May not be a problem for applications that do not rely on TCP for the end-to-end semantics



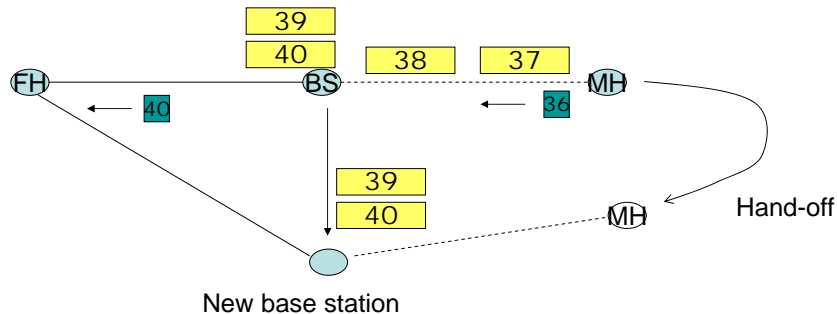
Split Connection Approach : Disadvantages

- BS failure can result in loss of data (unreliability)
 - If BS fails, packet 40 will be lost
 - Because it is ack'd to sender, the sender does not buffer 40



Split Connection Approach : Disadvantages

- Hand-off latency increases due to state transfer
 - Data that has been ack'd to sender, must be moved to new base station



Split Connection Approach : Disadvantages

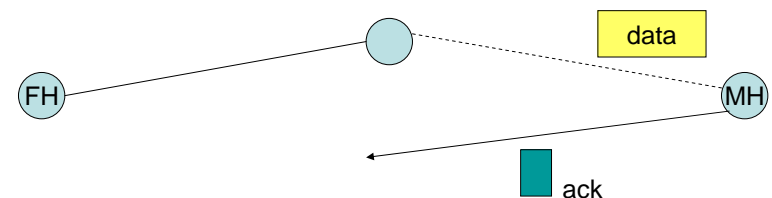
- Buffer **space** needed at BS for each TCP connection
 - BS buffers tend to get full, when wireless link slower (one window worth of data on wired connection could be stored at the base station, for each split connection)
- Window on BS-MH connection reduced in response to errors
 - may not be an issue for wireless links with small delay-bw product

Split Connection Approach : Disadvantages

- Extra **copying of data** at BS
 - copying from FH-BS socket buffer to BS-MH socket buffer
 - increases end-to-end **latency**
- Base station performs all retransmissions
 - What if wireless device is the sender?

Split Connection Approach : Disadvantages

- May not be useful if data and acks traverse different paths (both do not go through the base station)
 - Example: data on a satellite wireless hop, acks on a dial-up channel



Overall results

- Good TCP-aware LL shields sender from duplicate acks
 - Avoids redundant retransmissions by sender and base station
 - Adding selective acks helps a lot with bursty errors
- Split connection with standard TCP shields sender from losses, but poor wireless link still causes sender to stall
 - Adding selective acks over wireless link helps a lot
 - Still not as good as local LL improvement

Overall results (cont.)

- E2E schemes with selective acks help a lot
 - Still not as good as best LL schemes
- Explicit loss E2E schemes help (avoid shrinking congestion window) but should be combined with SACK for multiple packet losses

Explicit loss notification issues

- Receiver gets corrupted packet
- Instead of dropping it, TCP gets it, generates ELN message with duplicate ack
- What if header corrupted? Which TCP gets it?
 - Use FEC?

Explicit loss notification issues (cont.)

- Entire packet dropped?
 - Base station generates ELN messages to sender with ack stream
 - What if wireless node is the sender?

Conclusions / questions

- Not everyone believes in TCP fast retransmission
 - Error bursts may be due to your location
 - Maybe it doesn't change fast enough to warrant quick retransmission
 - A waste of power and channel
- Can information from link level be used by TCP?
 - Time scale may be such that by the time TCP or app adjust to information, it's already changed

Conclusions / questions (cont.)

- Really need to consider trade offs of packet size, power, retransmit adjustments
 - Worth increasing the power for retransmission?
 - Worth shrinking the packet size?