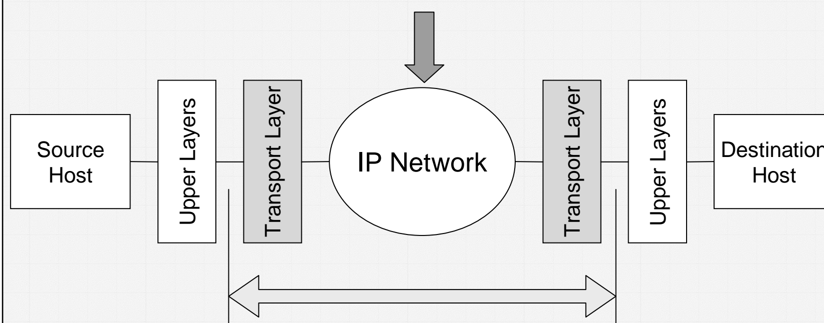
Course Overview

- Introduction and History
- Data in Wireless Cellular Systems
- Data in Wireless Local Area Networks
- Internet Protocols
- Routing and Ad-Hoc Networks
- TCP over Wireless Link
  - some slides in this section are from the Tutorial on TCP for Wireless and Mobile Hosts, prepared by Nitin Vaidya, see http://www.cs.tamu.edu/faculty/vaidya/presentations.html
- Services and Service Discovery
- System Support for Mobile Applications

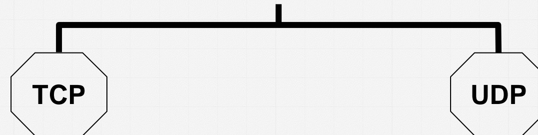
Transport Protocol

- What is the role of the "Transport Layer" ?



Two Transport Protocols

- The Internet uses 2 transport protocols



Connection-Oriented

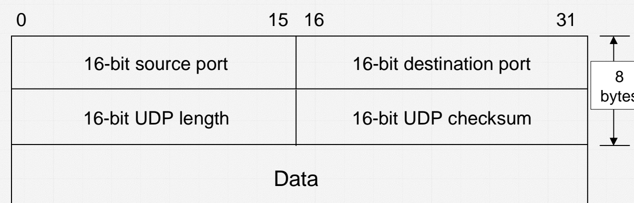
- Comprehensive
- Full-duplex
- Acknowledgment
- Sequencing
- Variable length segmentation
- Error control

Connectionless (Datagram)

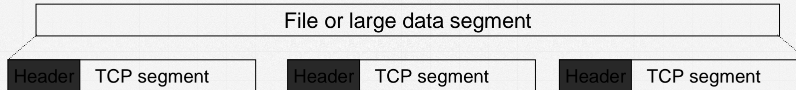
- Very simple
- No error control
- No sequencing

The User Data Protocol (UDP)

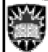
- UDP is much simpler protocol than TCP
- It is designed to transport individual datagrams (no sequence numbers)
- No acknowledgment
- It is used when high reliability is not needed
- The most common use is by protocols that handle name lookups
- Checksum is optional



Overview of TCP

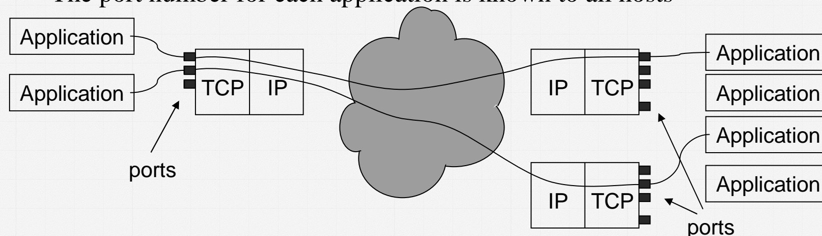


- TCP in the main Internet transport protocol. The UDP plays a supporting role (mostly house keeping functions)
- The TCP performs the following functions:
  - Divides the data into segments (21 to 64,000 bytes)
  - The sending TCP stamps the segments with sequence numbers
  - The receiving TCP acknowledges the segments
  - The receiving TCP controls the flow of segments
  - The TCP can flag data segments with different priorities (e.g. urgent, externally urgent/to be pushed ..)
  - TCP performs error correction
- The header of the TCP segment has several other fields and options



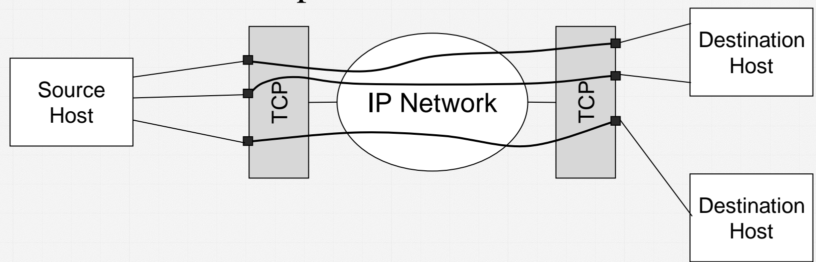
Ports and Sockets

- Communications through the Internet occurs between a "Client" application software and the same application software running on a "server"
- Each connection is uniquely identified by 4 addresses: (1) Client IP address, (2) client application port #, (3) server IP address and (4) server application port.
- Application ports on the server side are called: "Well-known Sockets". The port number for each application is known to all hosts



TCP Features

- Virtual circuit connection
- Full Duplex. Two-way simultaneous data flow.
- Reliable. Checks the integrity of the received data
- Allows for multiple connections

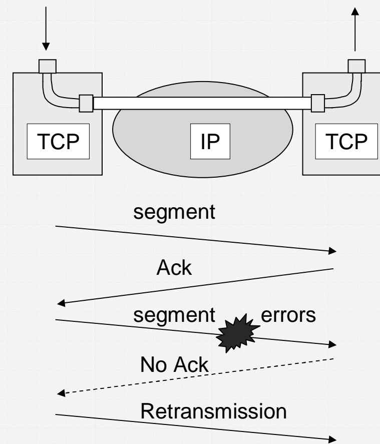


Well-known Port Numbers

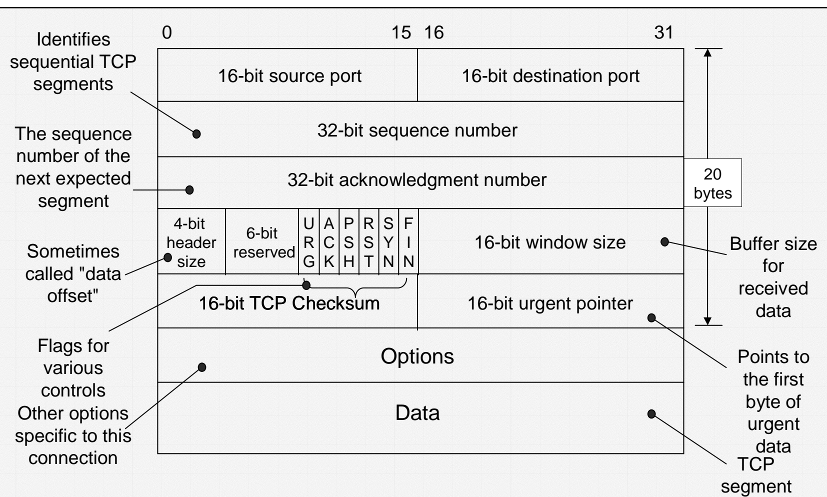
Port Number	Description
0	Reserved
20	FTP-data
23	Telnet
25	SMTP
70	Gopher
79	Finger Protocol
80	WWW

Error Control at the TCP Level

- Octet-stream-oriented error control
- Transmissions occur in segments. Each segment has a sequence number. The sequence number is the first octet of the segment
- The receiving TCP host sends an acknowledgment. The ACK number is the next expected data octet.
- Data that has not been acknowledged are re-transmitted.



The TCP Header

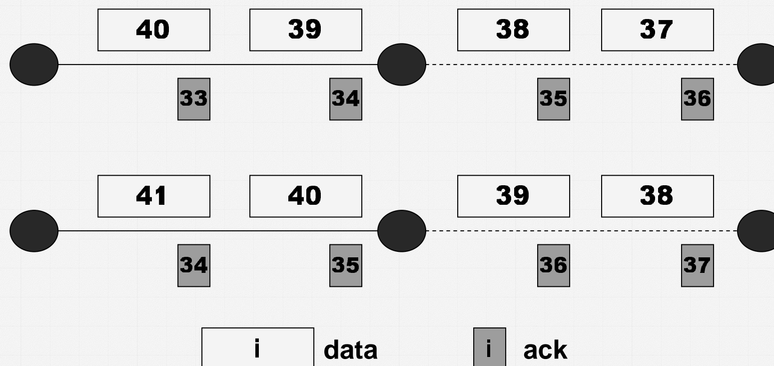


Transmission Control Protocol (TCP)

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

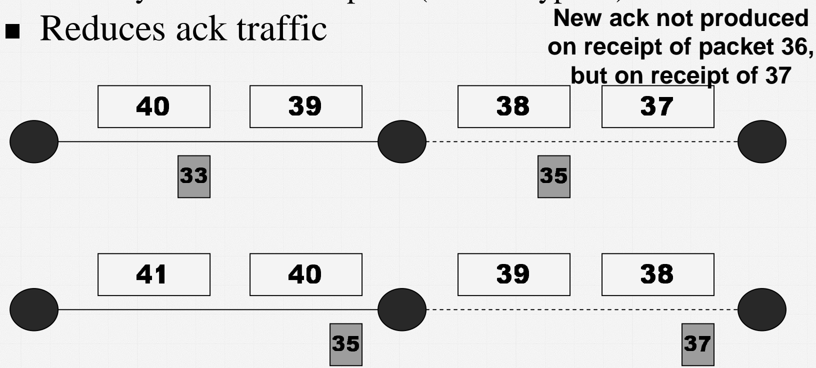
Cumulative Acknowledgements

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



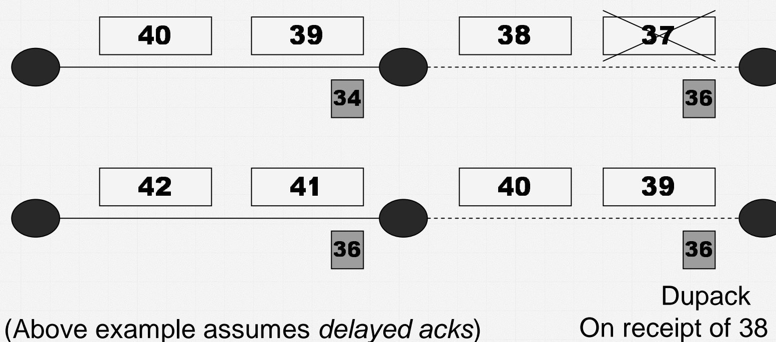
Delayed Acknowledgements

- An ack is delayed until
  - another packet is received, or
  - delayed ack timer expires (200 ms typical)
- Reduces ack traffic



Duplicate Acknowledgements

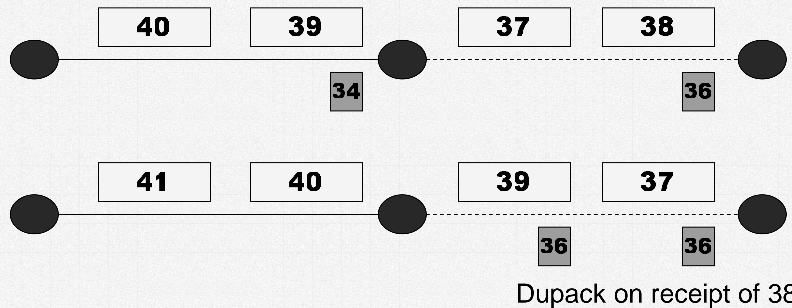
- A dupack is generated whenever an out-of-order segment arrives at the receiver



(Above example assumes *delayed acks*)

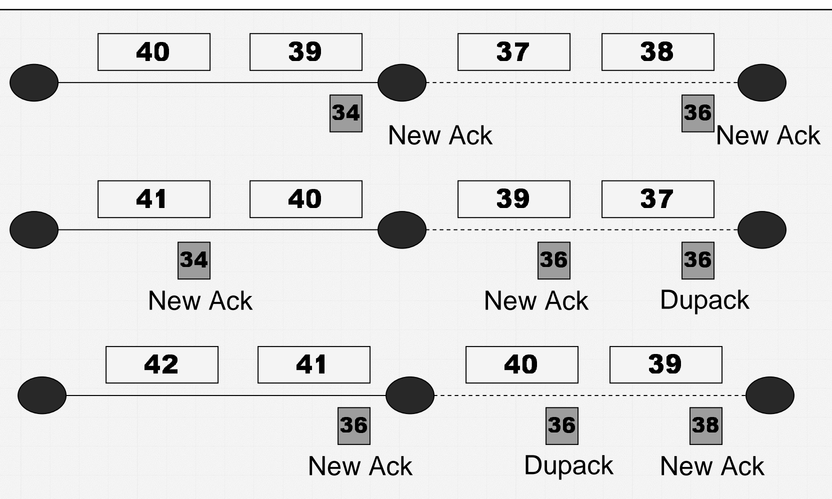
Duplicate Acknowledgements

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



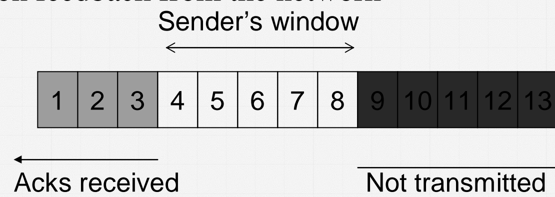
Dupack on receipt of 38

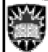
Number of dupacks depends on how much OOO a packet is



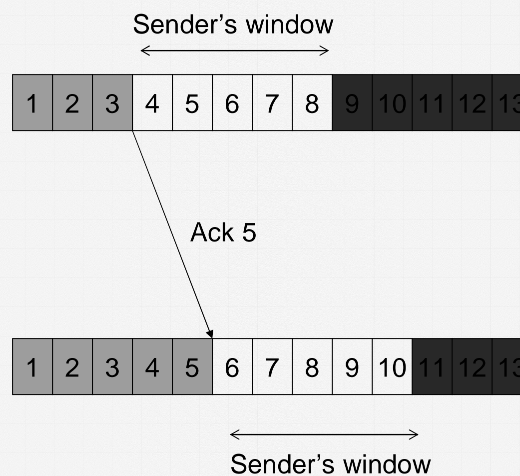
Window Based Flow Control

- Sliding window protocol
- Window size minimum of
  - receiver's advertised window - determined by available buffer space at the receiver
  - congestion window - determined by the sender, based on feedback from the network





Window Based Flow Control

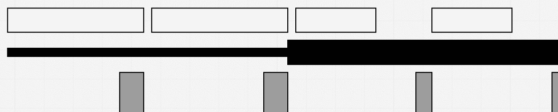


Ack Clock

- TCP window flow control is "self-clocking"
- New data sent when old data is ack'd
- Helps maintain "equilibrium"
- Congestion window size bounds the amount of data that can be sent per round-trip time
- Throughput <= W / RTT

Ideal Window Size

- Ideal size = delay * bandwidth
  - delay-bandwidth product



- What if window size < delay*bw ?
  - Inefficiency (wasted bandwidth)
- What if > delay*bw ?
  - Queuing at intermediate routers
    - increased RTT due to queuing delays
  - Potentially, packet loss

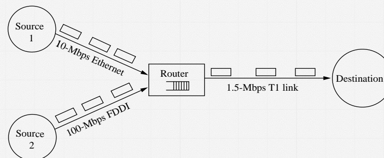
Keeping the Pipe Full

- Bandwidth & Delay x Bandwidth Product

Bandwidth	Delay x Bandwidth Product
T1 (1.5Mbps)	18KB
Ethernet (10Mbps)	122KB
T3 (45Mbps)	549KB
FDDI (100Mbps)	1.2MB
STS-3 (155Mbps)	1.8MB
STS-12 (622Mbps)	7.4MB
STS-24 (1.2Gbps)	14.8MB

Congestion Control

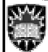
- Two sides of the same coin
  - pre-allocate resources to avoid congestion
  - send data and control congestion if (and when) is occurs



- Two points of implementation
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)
- Underlying service model
  - best-effort
  - no quality of service guarantees

TCP Congestion Control

- Idea
  - assumes best-effort network
  - each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (self-clocking)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity

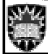


Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: CongestionWindow
  - limits how much data source has in transit
  - MaxWin =
    MIN(CongestionWindow,AdvertisedWindow)
  - EffWin = MaxWin - (LastByteSent - LastByteAcked)

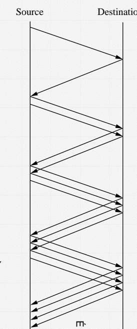
Additive Increase/Multiplicative Decrease

- Idea:
  - increase CongestionWindow when congestion goes down
  - decrease CongestionWindow when congestion goes up
- Question: how does the source determine whether or not the network is congested?
- Answer: implicitly through packet loss
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion



Additive Increase/Multiplicative Decrease

- Algorithm:
  - increment CongestionWindow by one packet per RTT (linear increase)
  - divide CongestionWindow by two whenever a timeout occurs (multiplicative decrease)
- In practice: increment a little for each ACK
  - Increment = (MSS * MSS)/CongestionWindow
  - CongestionWindow += Increment



How does TCP detect a packet loss?

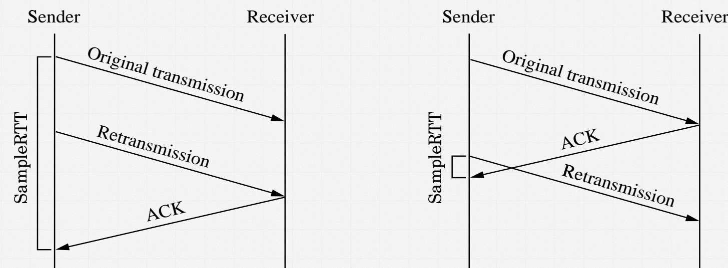
- Retransmission timeout (RTO)
- Duplicate acknowledgements

Detecting Packet Loss Using Retransmission Timeout (RTO)

- At any time, TCP sender sets retransmission timer for only one packet
- If acknowledgement for the timed packet is not received before timer goes off, the packet is assumed to be lost
- RTO dynamically calculated
  - Connection may be between two machines on same LAN (want low RTO value) or two machines on opposite sides of Atlantic (need higher RTO value)
  - Network connection between two machines introduces predictable and constant delay per packet (can use tighter bound) or highly variable packet delay (use less tight bound to avoid unnecessary retransmissions)
  - Use observed time difference between packet sent and acknowledgment received to estimate RTO

Adaptive Retransmission

- Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission

Adaptive Retransmission

- Jacobson/Karels Algorithm
- New calculation for average RTT
  - Diff = SampleRTT - EstimatedRTT
  - EstimatedRTT = EstimatedRTT + (δ x Diff)
  - Deviation = Deviation + δ(|Diff|- Deviation)
  - where δ is a fraction between 0 and 1
- Consider variance when setting timeout value
  - TimeOut = μ x EstimatedRTT + ϕ x Deviation
  - where μ = 1 and ϕ = 4
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control

Fast Retransmission

- Timeouts can take too long
  - how to initiate retransmission sooner?
- Fast retransmit

Detecting Packet Loss Using Dupacks Fast Retransmit Mechanism

- Dupacks may be generated due to
  - packet loss, or
  - out-of-order packet delivery
- TCP sender assumes that a packet loss has occurred if it receives three **dupacks** consecutively

Congestion Avoidance and Control

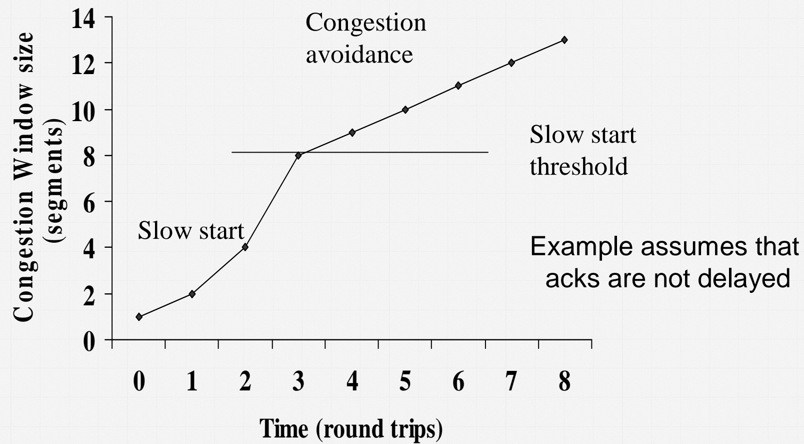
Slow Start

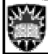
- initially, congestion window size cwnd = 1 MSS (maximum segment size)
- increment window size by 1 MSS on each new ack
- slow start phase ends when window size reaches the slow-start threshold
- cwnd grows **exponentially** with time during slow start
  - factor of 2 per RTT
  - Could be less if sender does not always have data to send

Congestion Avoidance

- On each new ack, increase cwnd by 1/cwnd packets
- cwnd increases **linearly** with time during congestion avoidance
  - 1 MSS per RTT

Congestion Avoidance





Congestion Control

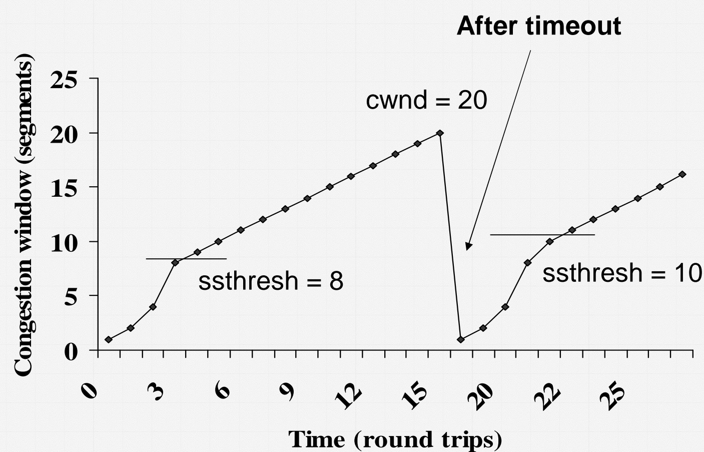
- On detecting a packet loss, TCP sender assumes that network congestion has occurred
- On detecting packet loss, TCP sender drastically reduces the congestion window
- Reducing congestion window reduces amount of data that can be sent per RTT
  - throughput may decrease

Congestion Control - Timeout

- On a timeout, the congestion window is reduced to the initial value of **1 MSS**
- The slow start threshold is set to half the window size before packet loss
  - more precisely,

    ssthresh = maximum of min(cwnd,receiver's advertised window)/2 and 2 MSS
- **Slow start** is initiated

Congestion Control - Timeout



Congestion Control - Fast retransmit

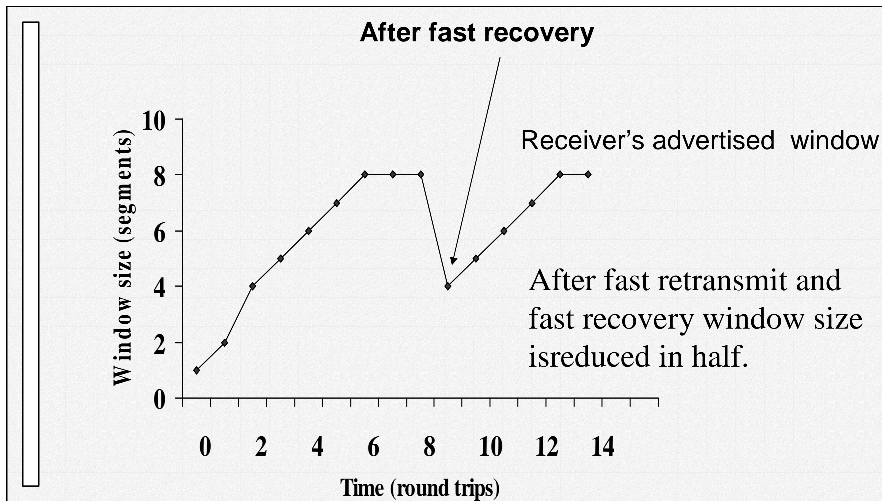
- Fast retransmit occurs when multiple (>= 3) dupacks come back
- Fast recovery follows fast retransmit
- Different from timeout : slow start follows timeout
  - timeout occurs when no more packets are getting across
  - fast retransmit occurs when a packet is lost, but latter packets get through
  - ack clock is still there when fast retransmit occurs
  - no need to slow start

Fast Recovery

- ssthresh =
  min(cwnd, receiver's advertised window)/2
  (at least 2 MSS)
- retransmit the missing segment (fast retransmit)
- cwnd = ssthresh + number of dupacks
- when a new ack comes: cwnd = ssthreh
  - enter congestion avoidance

Congestion window cut into half

Fast Retransmit/Fast Recovery

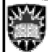


TCP and Mobile Computing

- TCP is (most?) popular transport layer protocol
- designed for wired networks
  - low error rate
  - requirement to share bottlenecks
- key assumptions in TCP are:
  - packet loss is indication of congestion, not transmission error
  - rather aggressive response to congestion is needed to ensure fairness and efficiency
- wireless links and mobile computing violate these assumptions:
  - packets lost due to unreliable physical media
  - packets can get lost due to handover

TCP and Mobile Computing

- packet losses over wireless link often in bursts
  - will trigger slow start rather than fast retransmit
- packet loss no indication of congestion
  - reduction of congestion window will reduce throughput
  - getting back to previous window size may take long
- problem caused by mismatch of wireless link properties with assumptions underlying TCP design
- multiple suggestions to improve TCP performance:
  - link-level retransmissions: improve reliability of wireless link
  - network layer solutions: SNOOP
  - transport layer solutions: I-TCP (indirect TCP), Mowgli
  - session layer solutions: establish end-to-end session layer connection, manages two separate TCP connections



Link Layer Mechanisms Forward Error Correction

- Forward Error Correction (FEC) can be use to correct small number of errors
- Correctable errors hidden from the TCP sender
- FEC incurs overhead even when errors do not occur
  - Adaptive FEC schemes can reduce the overhead by choosing appropriate FEC dynamically
- FEC does not guard/protect from packet loss due to handover

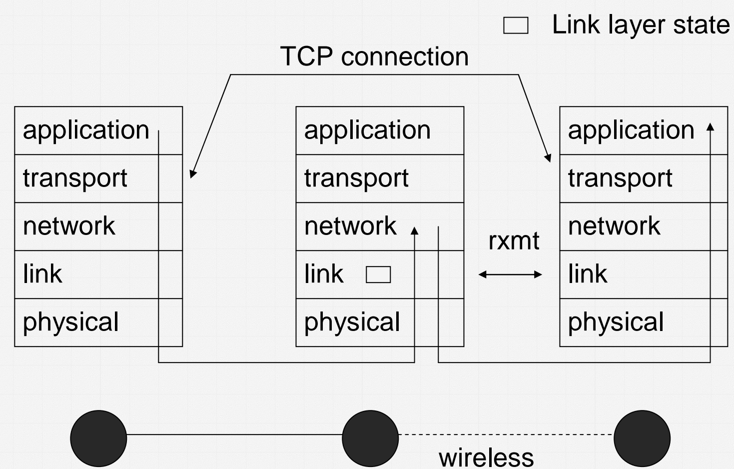
Link Layer Mechanisms
Link Level Retransmissions

- Link level retransmission schemes retransmit a packet at the link layer, if errors are detected
- Retransmission overhead incurred only if errors occur
  - unlike FEC overhead

In general

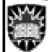
- Use FEC to correct a small number of errors
- Use link level retransmission when FEC capability is exceeded

Link Level Retransmissions



Link Level Retransmissions Issues

- How many times to retransmit at the link level before giving up?
  - Finite bound -- semi-reliable link layer
  - No bound -- reliable link layer
- What triggers link level retransmissions?
  - Link layer timeout mechanism
  - Link level acks (negative acks, dupacks, …)
  - Other mechanisms (e.g., Snoop, as discussed later)
- How much time is required for a link layer retransmission?
  - Small fraction of end-to-end TCP RTT
  - Large fraction/multiple of end-to-end TCP RTT

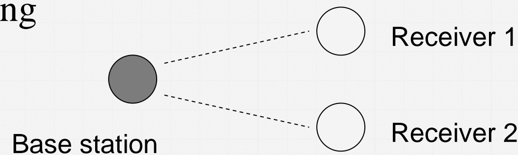


Link Level Retransmissions Issues

- Should the link layer deliver packets as they arrive, or deliver them in-order?
  - Link layer may need to buffer packets and reorder if necessary so as to deliver packets in-order

Link Level Retransmissions Issues

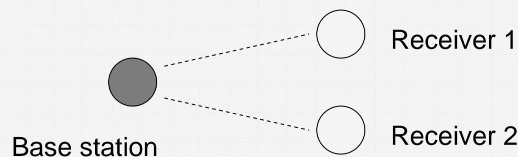
- Retransmissions can cause head-of-the-line blocking



- Although link to receiver 1 may be in a bad state, the link to receiver 2 may be in a good state
- Retransmissions to receiver 1 are lost, and also block a packet from being sent to receiver 2

Link Level Retransmissions Issues

- Retransmissions can cause congestion losses



- Attempting to retransmit a packet at the front of the queue, effectively reduces the available bandwidth, potentially making the queue at base station longer
- If the queue gets full, packets may be lost, indicating congestion to the sender
- Is this desirable or not ?

Link Level Retransmissions
An Early Study

- The sender's Retransmission Timeout (RTO) is a function of measured RTT (round-trip times)
  - **Link level retransmits increase RTT, therefore, RTO**
- **If errors not frequent**, RTO will **not** account for RTT variations due to link level retransmissions
  - When errors occur, the sender may timeout & retransmit before link level retransmission is successful
  - Sender and link layer both retransmit
  - Duplicate retransmissions (interference) waste wireless bandwidth
  - Timeouts also result in reduced congestion window

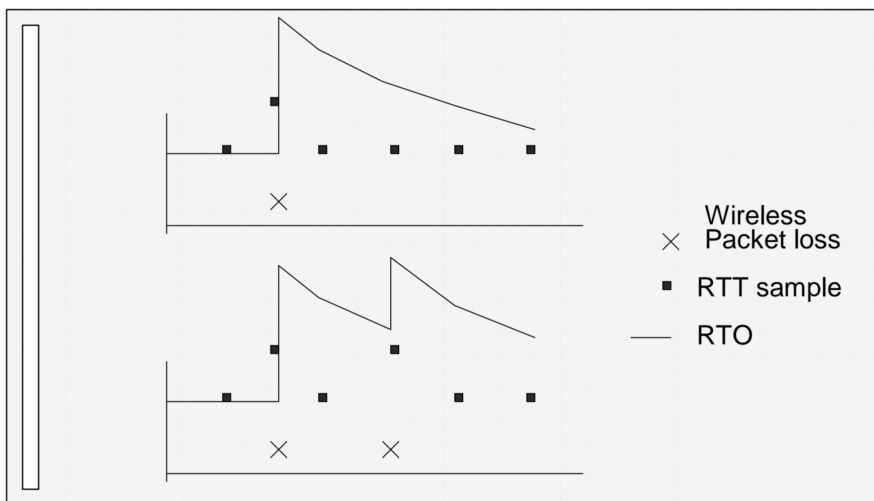
A More Accurate Picture

- Early analysis does not accurately model real TCP stacks
- With large **RTO granularity**, interference is unlikely, if time required for link-level retransmission is small compared to TCP RTO
  - Standard TCP RTO granularity is often large (500 ms)
  - Minimum RTO (2*granularity) is large enough to allow a small number of link level retransmissions, if link level RTT is relatively small
  - Interference due to timeout not a significant issue when wireless RTT small, and RTO granularity large

Link Level Retransmissions A More Accurate Picture

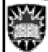
- **Frequent errors** increase RTO significantly on slow wireless links
  - RTT on slow links large, retransmissions result in large variance, pushing RTO up
  - Likelihood of interference between link layer and TCP retransmissions smaller
  - But congestion response will be delayed due to larger RTO
  - When wireless losses do cause timeout, much time wasted

RTO Variations



Large TCP Retransmission Timeout Intervals

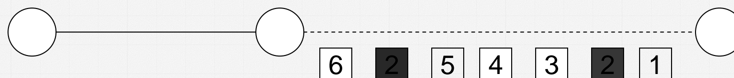
- Good for reducing interference with link level retransmits
- Bad for recovery from congestion losses
- Need a timeout mechanism that responds appropriately for both types of losses
  - Open problem



Link Level Retransmissions

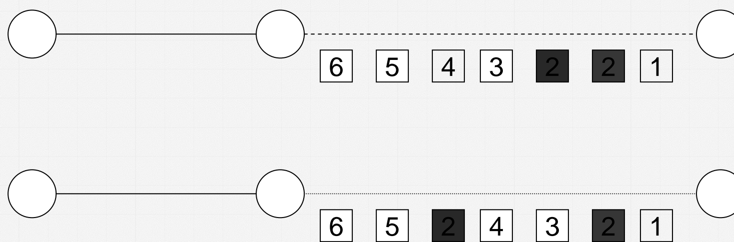
- Selective repeat protocols can deliver packets out of order
- Significantly out-of-order delivery can trigger TCP fast retransmit
  - Redundant retransmission from TCP sender
  - Reduction in congestion window
- Example: Receipt of packets 3,4,5 triggers dupacks

Lost packet
Retransmitted packet



Link Level Retransmissions
In-order delivery

- To avoid unnecessary fast retransmit, link layer using retransmission should attempt to deliver packets “almost in-order”



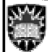
Link Level Retransmissions
In-order delivery

- Not all connections benefit from retransmissions or ordered delivery
  - audio
- Need to be able to specify requirements on a per-packet basis
  - Should the packet be retransmitted? How many times?
  - Enforce in-order delivery?
- Need a standard mechanism to specify the requirements
  - open issue (IETF PILC working group)

Link Layer Schemes: Summary

When is a reliable link layer beneficial to TCP performance?

- if it provides *almost in-order* delivery
- TCP retransmission timeout large enough to tolerate additional delays due to link level retransmits
- Basic ideas:
  - Hide wireless losses from TCP sender
  - Link layer modifications needed at both ends of wireless link
    - TCP need not be modified

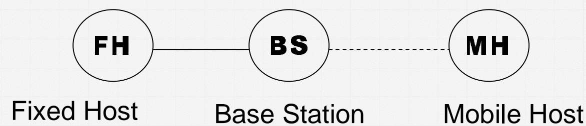


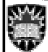
Split Connection Approach

- 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

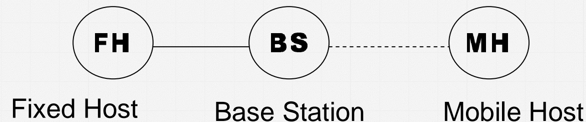
- Connection between wireless host MH and fixed host FH goes through base station BS
- FH-MH = FH-BS + BS-MH



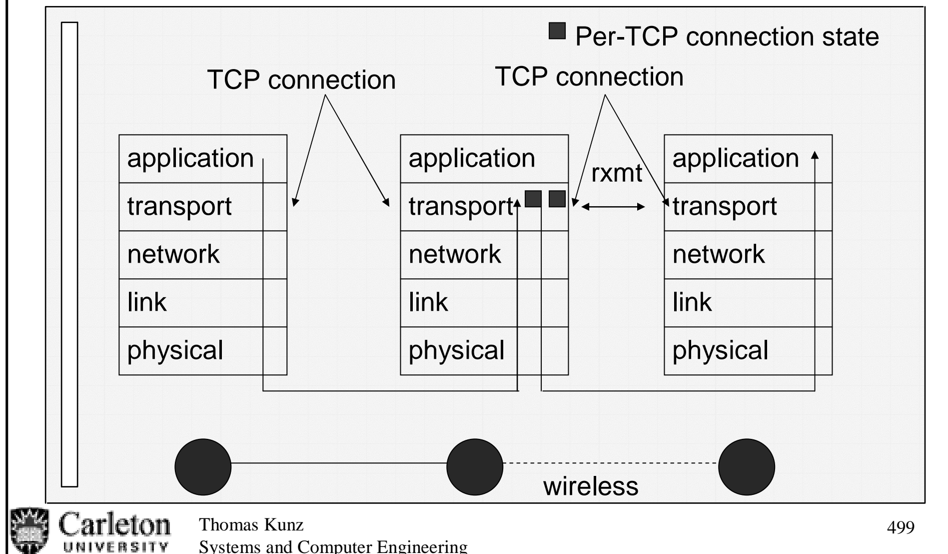


Split Connection Approach

- 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



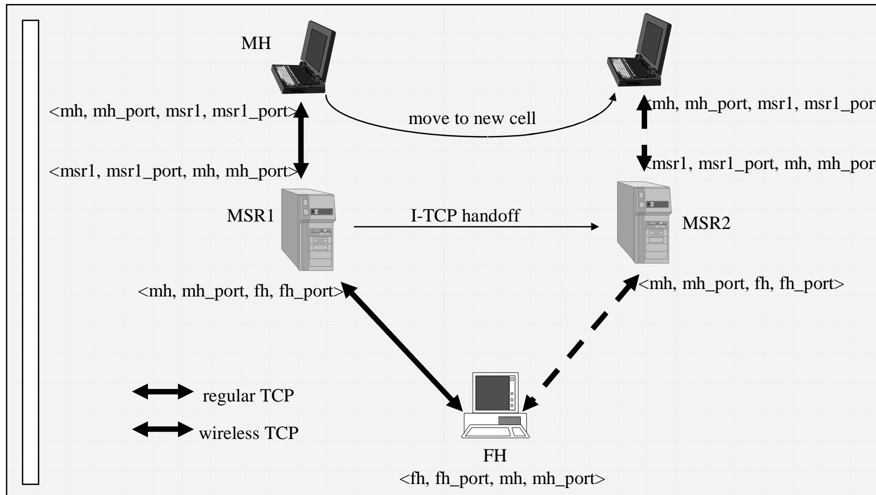
Split Connection Approach



I-TCP

- basic idea: split communication between mobile host (MH) and fixed host (FH) into two separate interactions
- each connection can be tuned to accommodate the special characteristics of the underlying physical media
  - use standard TCP between MSR and FH, both on wired backbone
  - special wireless TCP between MH and MSR, where packet loss does not trigger congestion avoidance

I-TCP: Connection Setup



I-TCP

- throughput improved, particularly for wide-area connections, compared to regular TCP

Connection Type	No moves	Overlapped cells	Disjoint cells, 0 sec between	Disjoint cells, 1 sec between
Regular TCP	65.49 kB/s	62.59 kB/s	38.66 kB/s	23.73 kB/s
I-TCP	70.06 kB/s	65.37 kB/s	44.83 kB/s	36.31 kB/s

I-TCP performance over local area

Connection Type	No moves	Overlapped cells	Disjoint cells, 0 sec between	Disjoint cells, 1 sec between
Regular TCP	13.35 kB/s	13.26 kB/s	8.89 kB/s	5.19 kB/s
I-TCP	26.78 kB/s	27.97 kB/s	19.12 kB/s	16.01 kB/s

I-TCP performance over wide area

(from: Bakre and Badrinath, “I-TCP: Indirect TCP for Mobile Hosts”, Proceedings of the International Conference on Distributed Computing Systems (ICDCS 15), May 1995, Vancouver, Canada, pages 136-143)

Split Connection Approach: Classification

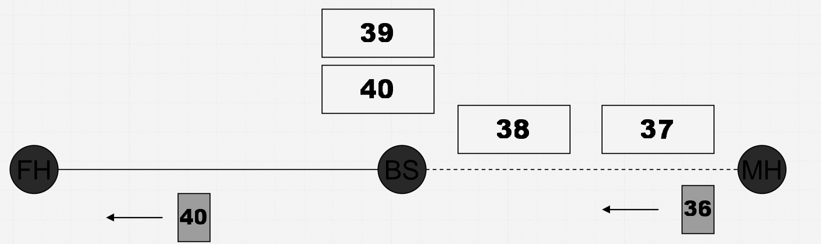
- Hides transmission errors from sender
- Primary responsibility at base station
- If specialized transport protocol used on wireless, then wireless host also needs modification

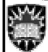
Split Connection Approach: Advantages

- 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
- 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

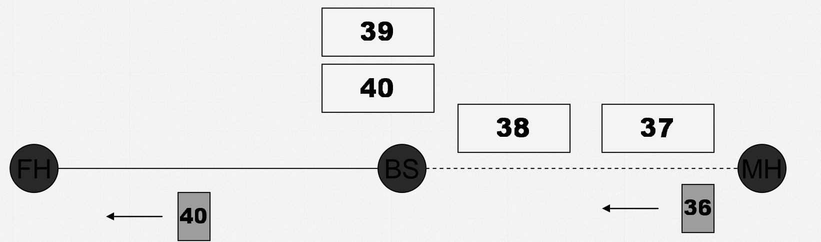




Split Connection Approach: Disadvantages

- BS (MSR in I-TCP) retains hard state

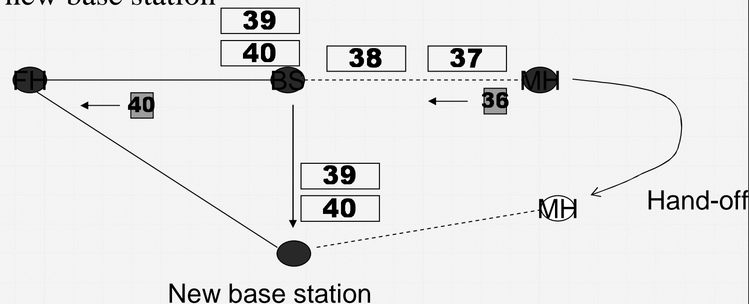
  BS failure can result in loss of data (unreliability)
  - If BS fails, packet 40 will be lost
  - Because it is ack'd to sender, the sender does not buffer 40



Split Connection Approach: Disadvantages

- BS retains hard state

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

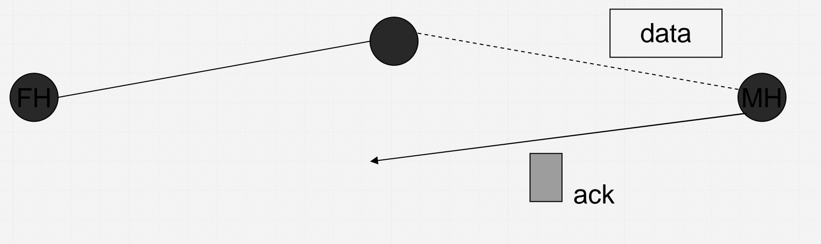


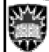
Split Connection Approach: Disadvantages

- 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
- 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

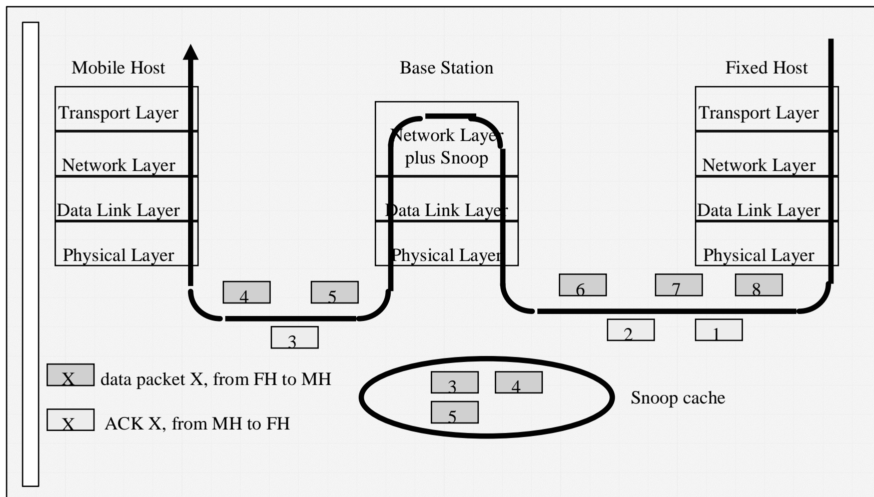




Snoop: Network Layer Solution

- idea: modify network layer software at base station
- changes are transparent to MH and FH
  - no changes in TCP semantics (unlike I-TCP)
  - less software overhead (packets pass TCP layer only twice)
  - no application relinking on mobile host
- modifications are mostly in caching packets and performing local retransmissions across the wireless link by monitoring (*snooping*) on TCP acks
- results are impressive:
  - speedups of up to 20 times over regular TCP
  - more robustness when dealing with multiple packet losses

Snoop: Architecture



Snoop: Description of Protocol

- processing packets from FH
  - new packet in the normal TCP sequence:
    - cache and forward to MH
  - packet out-of sequence and cached earlier:
    - sequence number > last ack from MH: packet probably lost, forward it again
    - otherwise, packet already received at MH, so drop
      - but: original ACK could have been lost, so fake ACK again
  - packet out-of sequence and not cached yet:
    - packet either lost earlier due to congestion or delivered out-of-order: cache packet and mark as retransmitted, forward to MH

Snoop: Description of Protocol

- processing ACKs from MH:
  - new ACK: common case, initiates cleaning up of snoop cache, update estimate of round-trip time for wireless link, forward ACK to FH
  - spurious ACK: less than last ACK seen, happens rarely. Just drop ACK and continue
  - duplicate ACK: indicates packet loss, one of several actions:
    - packet either not in cache or marked as retransmitted: pass duplicate ACK on to FH
    - first duplicated ACK for cached packet: retransmit cached packet immediately and at high priority, estimate number of expected duplicate ACKs, based on # of packets sent after missing one
    - expected successive duplicate ACKs: ignore, we already initiated retransmission. Since retransmission happens at higher priority, we might not see total number of expected duplicate ACKs

Snoop: Description of Protocol

- design does not cache packets from MH to FH
  - bulk of packet losses will be between MH and base
  - but snooping on packets generates requests for retransmissions at base much faster than from remote FH
  - enhance TCP implementation at MH with “selective ACK” option:
    - base keeps track of packets lost in a transmission window
    - sends bit vector back to MH to trigger retransmission of lost packets
- mobility handling:
  - when handoff is requested by MH or anticipated by base station, nearby base stations begin receiving packets destined for MH, priming their cache
  - caches synchronized during actual handoff (since nearby bases cannot snoop on ACKs)

Snoop: Performance

- no difference in very low error rate environment (bit error rate $< 5x10^{-7}$)
- for higher bit error rates, Snoop outperforms regular TCP by a factor of 1 to 20, depending on the bit error rate (the higher, the better Snoop's relative performance)
- even when every other packet was dropped over the wireless link, Snoop still allowed for progress in transmission, while regular TCP came to a grinding halt
- Snoop provides high and consistent throughput, regular TCP triggers congestion control often, which leads to periods of no transmission and very uneven rate of progress

Snoop: Evaluation

- most effort spent on direction FH->MH
  - authors argue that not much can be done for MH->FH
    - losses occur over first link, the unreliable wireless link
- Internet drops 2%-5% of IP packets, tendency rising
  - assume that IP packet is lost in wired part of network:
    - receiver (FH) will issue duplicate ACKs
    - this should trigger fast retransmit rather than slow start (?)
    - nothing is done to ensure that ACKs are not dropped over last link
    - retransmission of data packet over wireless link is subject to unreliable link and low bandwidth again
  - Snoop could potentially benefit from caching packets in both directions
    - how would this differ from link-layer retransmission policy?

TCP over Wireless: Summary

- Discussed only a few ideas, for a more complete discussion, see Tutorial on TCP for Wireless and Mobile Hosts by Nitin Vaidya, http://www.cs.tamu.edu/faculty/vaidya/presentations.html
- Topics ignored:
  - asymmetric bandwidth on uplink and downlink (for example in some cable or satellite networks)
  - wireless link extends over multiple hops, such as in an ad-hoc network
  - connections fail due to spurious disconnections or route failures in ad-hoc networks
- Many proposals focus on downlink only
- Many proposals, most try to avoid changing TCP interface or semantics, but more work necessary