TCP for wireless networks

CS 444N, Spring 2002

Instructor: Mary Baker

Computer Science Department

Stanford University

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)
- TCP assumes packet loss is due to
  - Congestion
  - Reordering (rarely)
- TCP's congestion responses are triggered by wireless packet loss but interact poorly with wireless nets

TCP congestion detection

- TCP assumes timeouts and duplicate acks indicate congestion or (rarely) packet reordering
- Timeout indicates packet or ack was lost
- Duplicate acks may indicate packet reordering
  - Acks up through last successful in-order packet received
  - Called a “cumulative” ack
  - After three duplicate acks, assume packet loss, not reordering
  - Receipt of duplicate acks means some data is still flowing

Responses to congestion

- Basic timeout and retransmission
  - If sender receives no ack for data sent, timeout and retransmit
  - Exponential back-off
  - Timeout value is sum of smoothed RT delay and 4 X mean deviation
  - (Timeout value based on mean and variance of RTT)
- Congestion “avoidance” (really congestion control)
  - Uses congestion window (cwnd) for more flow control
  - Cwnd set to 1/2 of its value when congestion loss occurred
  - Sender can send up to minimum of advertised window and cwnd
  - Use additive increase of cwnd (at most 1 segment each RT)
  - Careful way to approach limit of network

Responses to congestion, continued

- Slow start – used to initiate a connection
  - In slow start, set cwnd to 1 segment
  - With each ack, increase cwnd by a segment (exponential increase)
  - Aggressive way of building up bandwidth for flow
  - Also do this after a timeout – aggressive drop in offered load
  - Switch to regular congestion control once cwnd is one half of what it was when congestion occurred
- Fast retransmit and fast recovery
  - After three duplicate acks, assume packet loss, data still flowing
  - Sender resends missing segment
  - Set cwnd to ½ of current cwnd plus 3 segments
  - For each duplicate ack, increment cwnd by 1 (keep flow going)
  - When new data acked, do regular congestion avoidance

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

Poor interaction with TCP

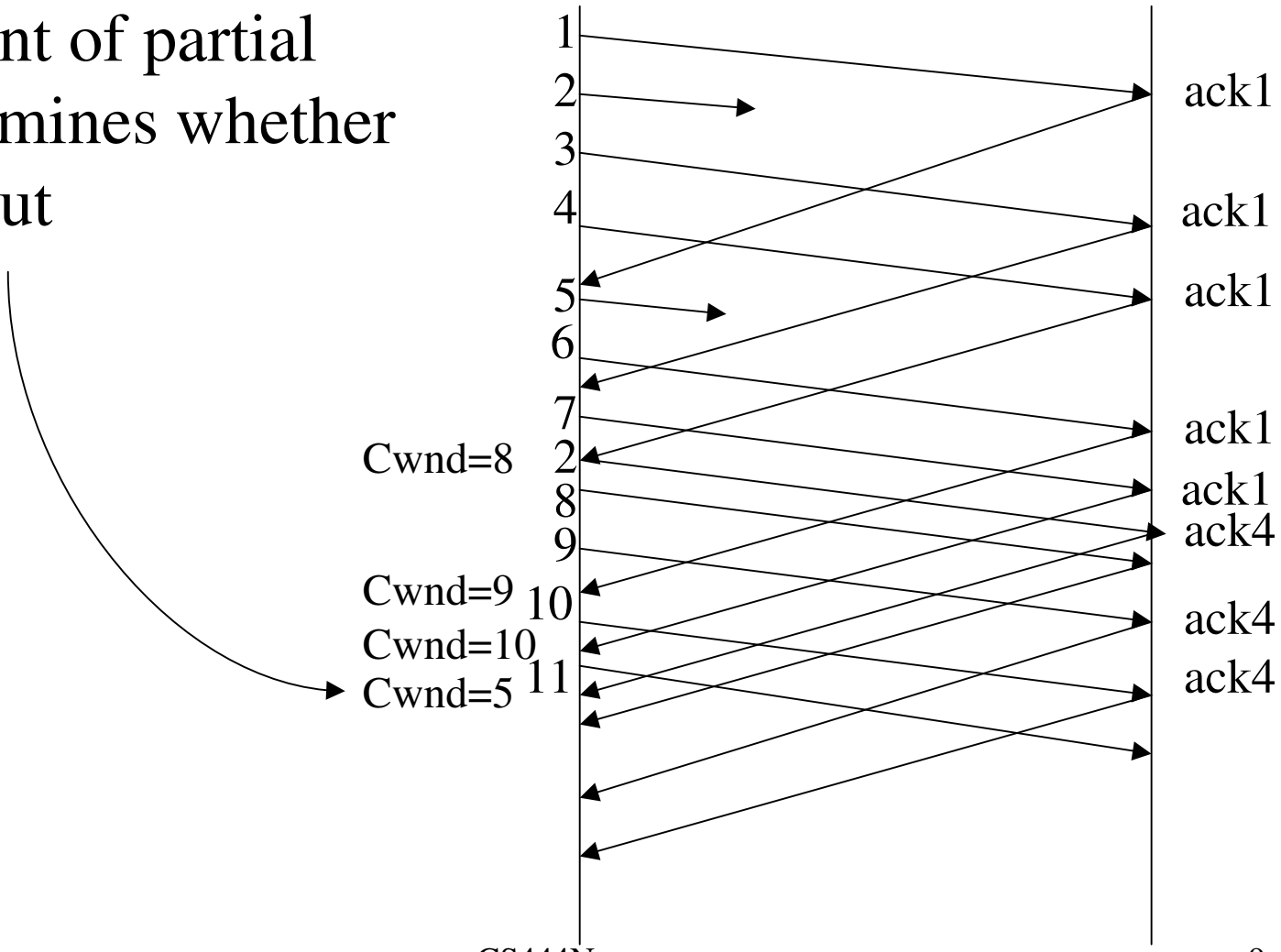
- Packet loss due to noise or hand-offs
  - Enter congestion control
  - Slow increase of cwnd
- Bursts of packet loss and hand-offs
  - Timeout
  - Enter slow start (very painful!)
- Cumulative ack scheme not good with bursty losses
  - Missing data detected one segment at a time
  - Duplicate acks take a while to cause retransmission
  - TCP Reno may suffer coarse time-out and enter slow start!
    - Partial ack still causes it to leave fast recovery
  - TCP New Reno still only retransmits one packet per RTT
    - Stay in fast recovery until all losses acked

Multiple losses in window

- Assume cwnd of 10
- 2nd and 5th packets lost
- 3rd duplicate ack causes retransmit of 2nd packet
- Also sets cwnd to 5 + 3 = 8
- Further duplicate acks increment cwnd by 1
- Ack of retransmit is partial ack since packet 5 lost
- In TCP Reno this causes us to leave fast retransmit
- Deflate congestion window to 5, but we've sent 11!

Coarse-grain timeout example

- Cwnd = 10
- Treatment of partial ack determines whether we timeout



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
- 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, continued

- 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
- 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

Link layer protocols investigated

- LL: TCP-ish one with cumulative acks and retransmit granularity faster than TCP's
- LL-SMART: addition of selective retransmissions
  - Cumulative ack with sequence # of of packet causing ack
- LL-TCP-AWARE: snoop protocol
  - At base station cache segments
  - Detect and suppress duplicate acks
  - Retransmit lost segments locally
- LL-SMART-TCP-AWARE: Combination of selective acks and duplicate ack suppression

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
- 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: ack describes 3 received non-contiguous ranges
- E2E-SMART: cumulative ack with sequence # of packet causing ack
  - Sender uses info for bitmask of okay packets
  - Ignores possibility that holes are due to reordering
  - Also problems with lost acks
  - Easier to generate and transmit acks

E2E protocols, continued

- 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

End-to-end results

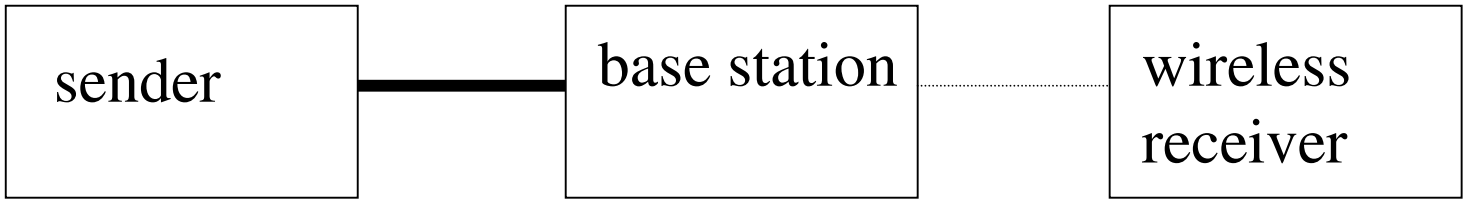
- E2E (Reno): coarse-grained timeouts really hurt
  - Throughput less than 50% of maximum in local area
  - Throughput of less than 25% in wide area
- E2E-New Reno: avoiding timeouts helps
  - Throughput 10-25% better in LAN
  - Throughput twice as good in WAN
- ELN techniques avoid shrinking congestion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

- 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
- 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 protocols

- Attempt to isolate TCP source from wireless losses
  - Lossy link looks like robust but slower BW link
- TCP sender over wireless link performs all retransmissions in response to losses



- Base station performs all retransmissions
- What if wireless device is the sender?
- SPLIT: uses TCP Reno over wireless link
- SPLIT-SMART: uses SMART-based selective acks

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-SMART:
  - 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

Error bursts

- 2-6 packets lost in a burst
- LL-SMART-TCP-AWARE up to 30% better than LL-TCP-AWARE
- Selective acks help in face of error bursts

Error rate effect

- At low error rates (1 error every 256 Kbytes) all protocols do about the same
- At 16 KB error rate, TCP-aware LL schemes about 2 times better than E2E-SMART and about 9 times better than TCP Reno
- E2E-SACK and SMART at high error rates:
  - Small cwnd
  - SACK won't retransmit until 3 duplicate acks
  - So no retransmits if window < 4 or 5
  - Sender's window often less than this, so timeouts
  - SMART assumes no reordering of packets and retransmits with first duplicate ack

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
- 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

Fast handoff proposals

- Multicast to old and new stations
  - Assumes extra support in network
  - Some concern about load on base stations
- Hierarchical foreign agents
  - Mobile host moves within an organization
  - Notifies only top-level foreign agent, rather than home agent
  - Home agent talks to top-level foreign agent, which doesn't change often
  - Requires foreign agents, extra support in network
- 10-30ms handoffs possible with buffering / retransmission at base stations

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

Network asymmetry

- Network is asymmetric with respect to TCP performance if the throughput achieved is not just a function of the link and traffic characteristics of the forward direction, but depends significantly on those of the reverse direction as well.
- TCP affected by asymmetry since its forward progress depends on timely receipt of acks
- Types of asymmetry
  - Bandwidth
  - Latency
  - Media-access
  - Packet error rate
  - Others? (cost, etc.)

BW asymmetry: one-way transfers

- Normalized bandwidth ratio between forward and reverse paths:
  - Ratio of raw bandwidths divided by ratio of packet sizes used
- Example:
  - 10 Mbps forward channel and 100 Kbps back link: ratio of bandwidths is 100
  - 1000-byte data packets and 40-byte acks: packet size ratio is 25
  - Normalized bandwidth ratio is 100/25 = 4
  - Implies there cannot be more than 1 ack for every 4 packets before back link is saturated
  - Breaks ack clocking: acks get spaced farther apart due to queuing at bottleneck link

BW asymmetry: two-way transfers

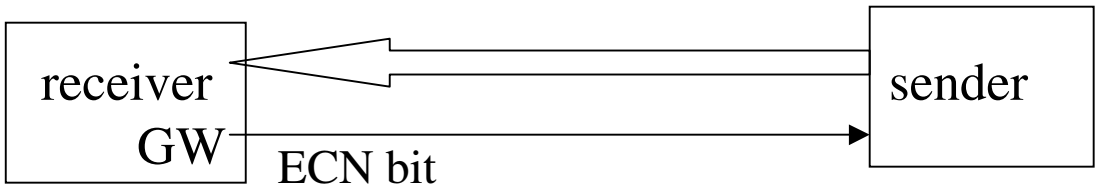
- Acks in one direction encounter saturated channel
- Acks in one direction get queued up behind large slow packets of other direction
- With slow reverse channel already saturated, forward channel only makes progress when TCP on reverse channel loses packets and slows down

Latency asymmetry in packet radio networks

- Multiple hops
  - Not necessarily same path through network
- Half-duplex radios
  - Cannot send and receive at same time
  - Must do “turn-around”
- Overhead per packet is slow due to MAC protocol
  - If you want to send to another radio, must first ask permission
  - Other radio may be busy (ack interference, for example)
  - Causes great variability in delays
  - Great variability causes retransmission timer to be set high

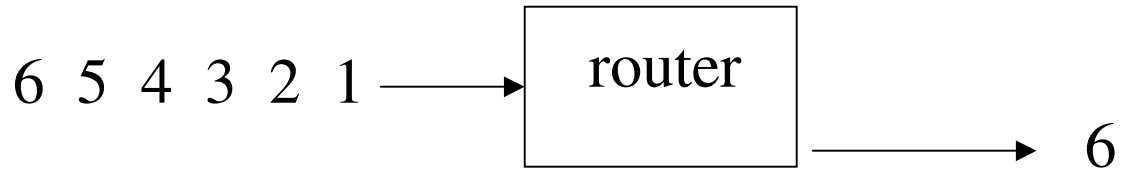
Solution: Ack congestion control

- Treat acks for congestion too
  - Gateway to weak link looks at queue size
  - If average size > threshold, set explicit congestion notification bit on a random packet
  - Sender reduces rate upon seeing this packet (Do we want this?!)
  - Receiver delays acks in response to these packets
  - New TCP option to get sender's window size – need >= 1 ack per sender window
  - Requires gateway support and end-point modification
  - How can you tell ECN's coming back aren't for congestion along that link?



Solution: ack filtering

- Gateway removes some (possibly all) acks sitting in queue if appropriate cumulative ack is enqueued
- Requires no per-connection state at router



Problems with ack-reducing techniques

- Sender burstiness
  - One ack acknowledges many packets
  - Many more packets get sent out at once
  - More likely to lose packets
- Slower congestion window growth
  - Many TCP increase window based on # of acks and not what they ack
- Disruption of fast retransmit algorithm since not enough acks
- Loss of a now rare ack means long idle periods on sender

Solution: sender adaptation

- Used in conjunction with ACC and AF techniques
- Sender looks at amount of data acked rather than # of acks
  - Ties window growth only to available BW in forward direction. Acks irrelevant.
- Counter burstiness with upper bound on # of packets transmitted back-to-back, regardless of window
- Solve fast retransmit problem by explicit marking of duplicate acks as requiring fast retransmit
  - By receiver in ACC
  - By reverse channel router in AF

Solution: ack reconstruction

- Local technique
- Improves use of previous techniques where sender has not been adapted
- Reconstructor inserts acks and spaces them so they will cause sender to perform well (good window, not bursty)
- Hold back some acks long enough to insert appropriate number of acks
- Preserves end-to-end nature of connection
- Trade-off is longer RTT estimate at sender

Solution: scheduling data and acks

- 2way transfers: data and acks compete for resources
- Two data packets together block an ack for a long time (sent in pairs during slow start)
- Router usually has both in one FIFO queue
- Try ack-first scheduling on router
  - With header compression, delay of ack is small for data
  - Unless on packet radio network!
  - Gateway does not need to differentiate between different TCP connections
  - Prevents starvation on forward transfer from data of reverse transfer

Overall results: 1-way, lossless

- C-SLIP can help a lot
  - Improves from 2Mbps to 7Mbps out of 10Mbps for Reno on 9.6Kbps channel
  - On 28.8Kbps channel, Reno and C-slip solves problem
- Ack filtering and congestion control help when normalized ratio is large and reverse buffer is small
- Ack congestion control never as good as ack filtering
- Ack congestion control doesn't work well with large reverse buffer
  - Does not kick in until the number of reverse acks is a large fraction of the queue
  - Time in queue is still big, so larger RTT

Overall results: 1-way, lossy

- AF without SA or AR is worse than normal Reno in terms of throughput, due to sender burstiness, etc.
- ACC is still not a good choice
- AF/AR has longer RTT
  - 97ms compared to 65 for AF/SA
- But much better throughput
  - 8.57Mbps compared to 7.82
  - Due to much larger cwnd

Results: 2-way transfers, 2nd started after

- Reno gets best aggregate throughput, but at total loss of fairness
  - It never lets reverse transfer into the game
  - 1st connection's acks fill up reverse channel
- ACC still in between
- AF gets fairness of almost equal throughput per connection (0.99 fairness index)

Results: 2-way transfers, simultaneous

- Reno 20% of runs:
  - Same problem with acks filling channel
- Reno 80% of runs:
  - If any reverse data packets make it into queue, acks of forward connect are delayed and cause timeouts
  - Gives other direction some room
  - Still not very fair
- AF: poor throughput on forward transfer, near optimal on reverse transfer
  - With FIFO scheduling, acks of forward transfer stuck behind data
  - Reverse connection continues to build window, so even more data packets to queue behind

Results, continued

- ACC with RED does much better!
  - RED prevents reverse transfer from filling up reverse GW with data
  - Reverse connection sustains good throughput without growing window to more than 1-2 packets
  - Still a few side-by-side data packets on link
- ACC with acks-1st scheduling takes care of this problem
- AF with acks-1st scheduling
  - Starvation of data packets of reverse transfer
  - Always an ack waiting to be sent in queue

Results: latency in multi-hop network

- At link layer, piggy back acks with data frames
  - Avoids extra link-layer radio turnarounds
- With single and multiple transfers
  - AF/SA outperforms Reno
  - Fairness much better with AF/SA
  - Also better utilization of network
  - Due to fewer interfering packets

Results: combined technologies

- Getting a little exotic
- "Web-like" benchmark
  - Request followed by four large transfers back to client
  - 1 to 50 hosts requesting transfers
- ACC not as good as AF in overall transaction time
  - Shorter transfer lengths so sender's window not large
  - ACC can't be performed much
  - AF also reduces number of acks and hence removes the variability associated with those packets

Implementation

- Acks queued in on-board memory on modem rather than in OS
  - Makes AF hard

Real measurements of packet network

- Round-trip TCP delays from 0.2 seconds to several seconds
  - Even minimum delay is noticeable to users
  - Median delay about ½ second
- A lot of retransmissions (25.6% packet loss!)
  - 80% of requests transmitted only once
  - 10% retransmitted once
  - 2% retransmitted twice
  - 1 packet retransmitted 6 times
- Less packet loss in reverse direction (3.6%)
  - Mobiles finally get packet through to poletop when conditions are ok
  - Poletop likely to respond while conditions are still good

Packet reordering

- Packets arrive out of order
  - Different paths through the poletops
  - Average out-of-order distance > 3 so packets treated as lost
  - Fair amount of packet reordering: 2.1% to 5.1% of packets

Conclusions / questions

- Is it worth using severely asymmetric links?
- Header compression helps a lot in many circumstances
- Except for some bidirectional traffic problems