CS/ECE 439: Wireless Networking

Transport Layer – TCP over Wireless

© CS 439 Staff, University of Illinois Fall 2026

Wireless Characteristics

- Variable bandwidth (rate adaptation, interference)
- Variable latency (retries, aggregation, handoffs)

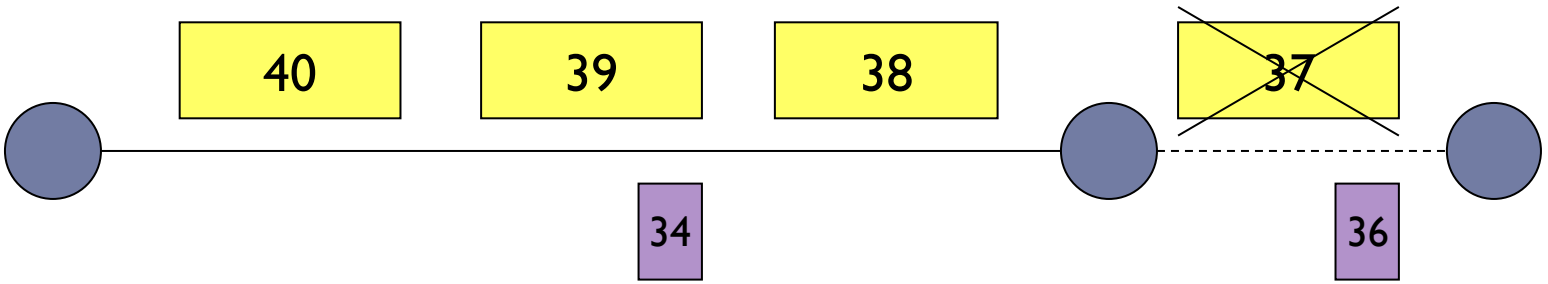
- Random Errors
  - If number of errors is small
    - May be corrected by an error correcting code
  - Excessive bit errors
    - Result in a packet being discarded, possibly before it reaches the transport layer





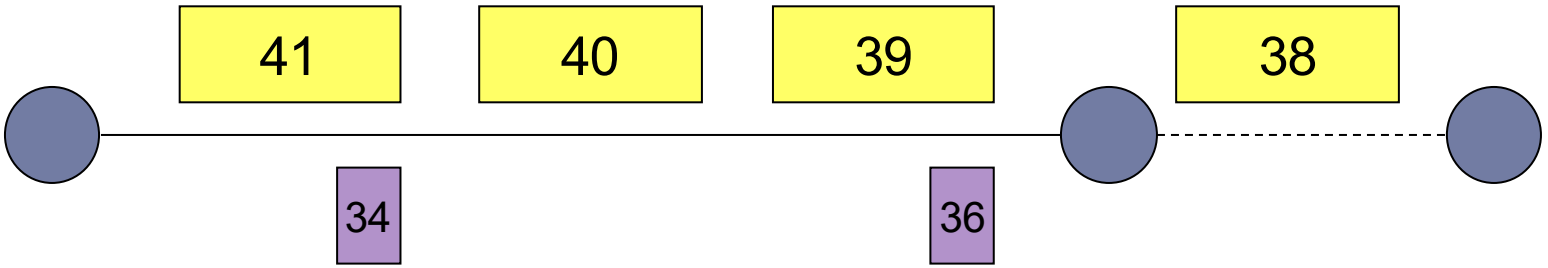
Random Errors

- May cause fast retransmit
  - Example assumes delayed ack - every other packet ack'd



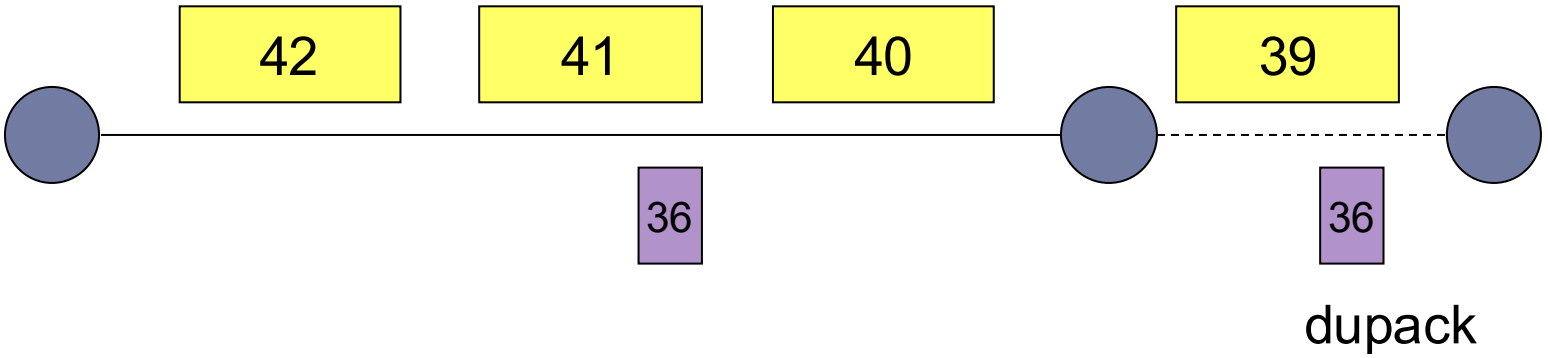
Random Errors

- May cause fast retransmit
  - Example assumes delayed ack - every other packet ack'd



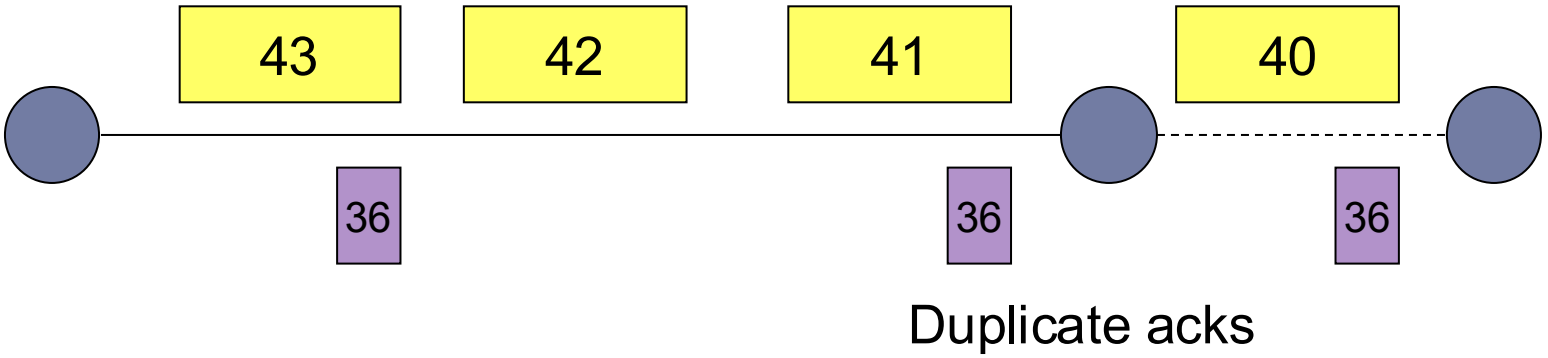
Random Errors

- May cause fast retransmit
  - Example assumes delayed ack - every other packet ack'd



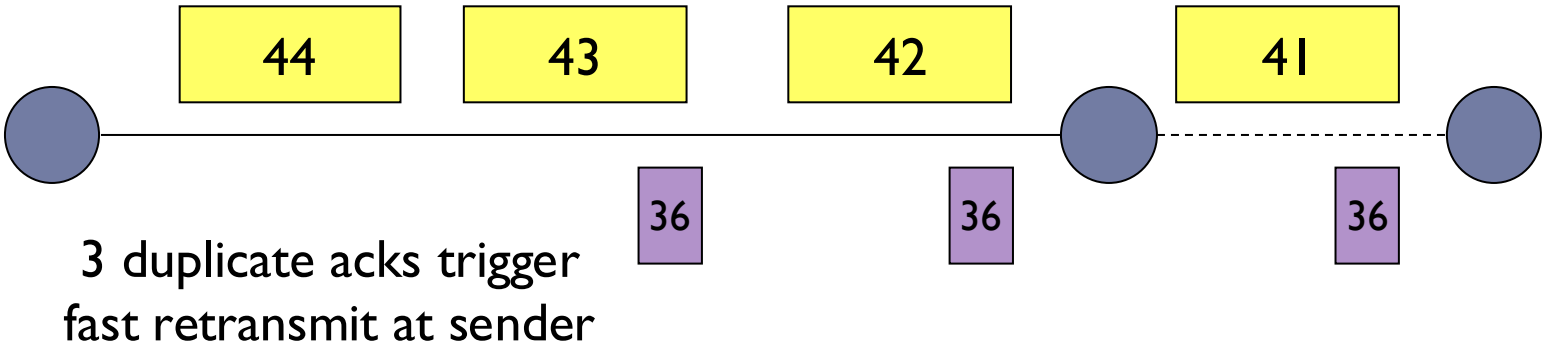
Random Errors

- May cause fast retransmit
  - Example assumes delayed ack - every other packet ack'd



Random Errors

- May cause fast retransmit
  - Example assumes delayed ack - every other packet ack'd



Random Errors

- Fast retransmit results in
  - Retransmission of lost packet
  - Reduction in congestion window
- Reducing congestion window
  - Unnecessary response to errors
  - Reduces the throughput



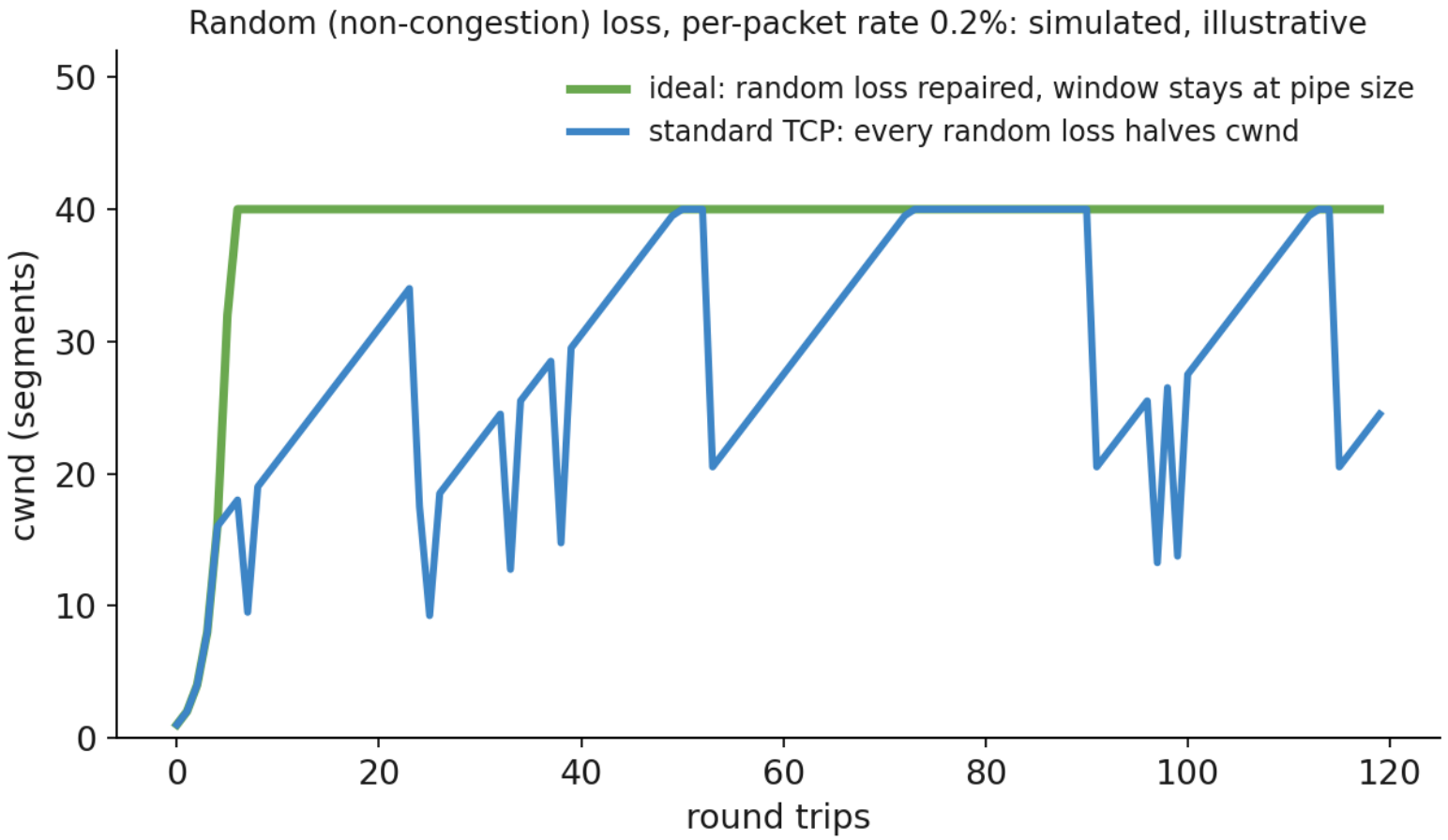
Random Errors

- Sometimes congestion response is appropriate
  - Interference due to other users
    - Reduce congestion window
  - Bad channel for a long duration
    - Let TCP backoff
    - Do not unnecessarily attempt retransmissions while the channel remains in the bad state
- But what about errors for which reducing congestion window is an inappropriate response?
  - Noise
    - Do not reduce window





Random Loss Keeps Cutting the Window







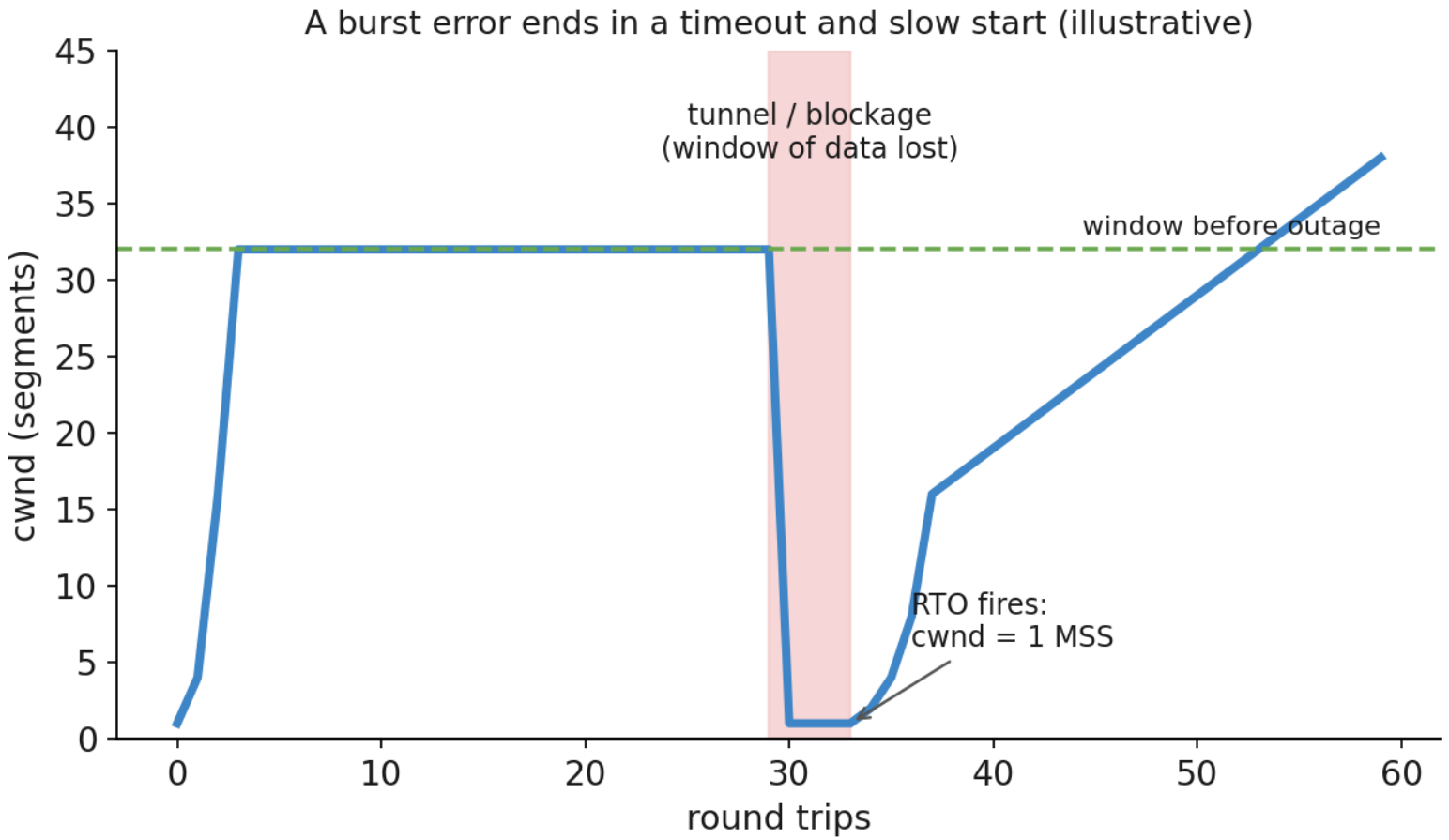
Timeouts

- 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 burst errors?
- Random errors may also cause timeouts
  - Multiple packet losses in a window can result in timeout when using TCP-Reno
    - And to a lesser extent when using SACK



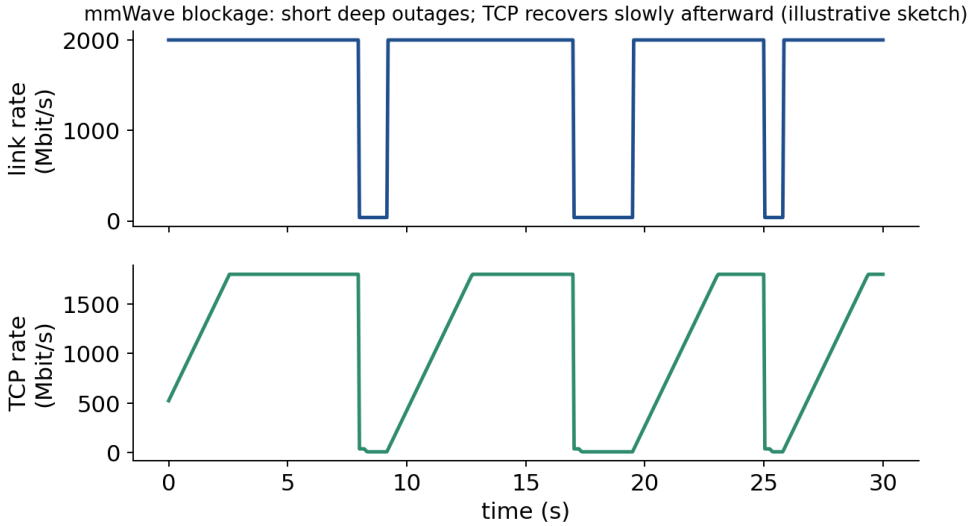


A Burst Error Ends in Timeout and Slow Start



Blockage and Bursty Loss (mmWave)

- Directional mmWave links can drop by tens of dB when a hand, body, or vehicle blocks the path
- Outages last from tens to hundreds of ms while the radio re-beams, switches cells, or falls back to LTE/sub-6 GHz
- TCP sees a burst loss or a long stall
  - Timeout, slow start from 1 MSS, then a slow climb on a multi-Gbit/s link
  - The large bandwidth-delay product makes recovery especially long



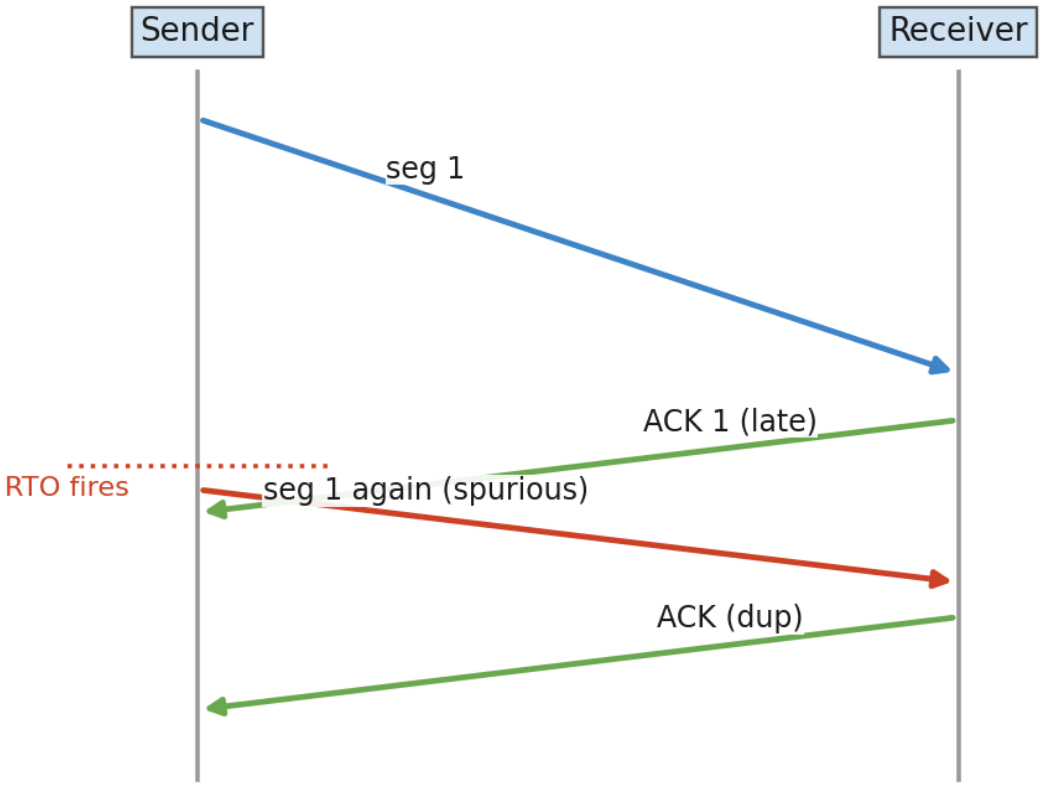
Spurious Timeouts

- A delay spike can outlast the RTO even when nothing was lost
  - Link-level retries, handoffs, or a sudden rate drop
- The sender retransmits needlessly and falls into slow start
- Remedies
  - F-RTO and Eifel
    - Detect that the timeout was spurious and undo the window cut
  - RACK-TLP
    - Time-based loss detection plus a tail-loss probe (Linux default)
  - SACK
    - Tells the sender exactly what arrived





Spurious Timeout Timeline



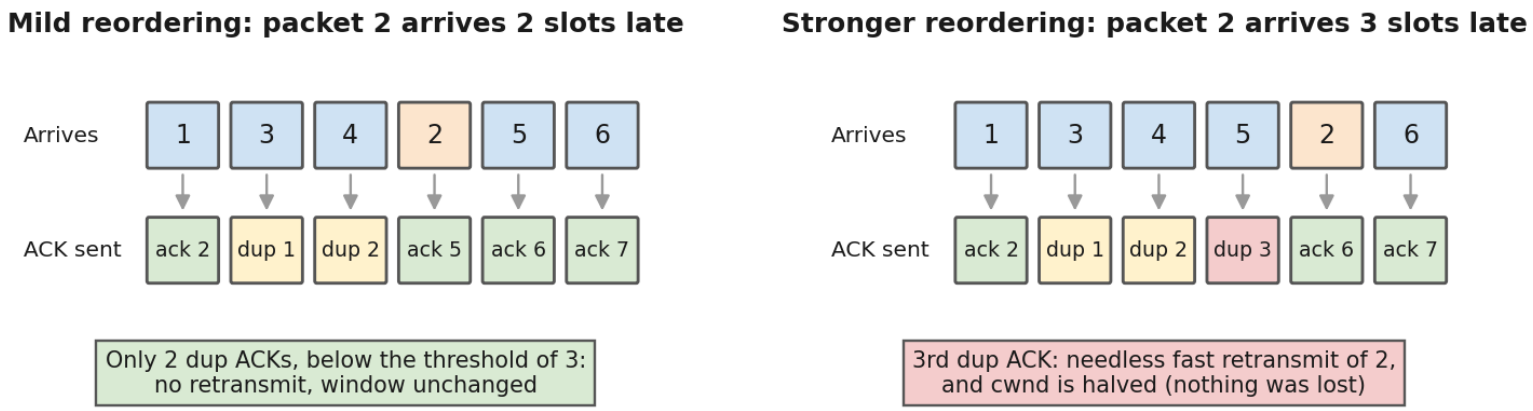
Delay spike: link retries, handoff, or rate drop

Nothing was lost, but the sender cuts cwnd and enters slow start

F-RTO / Eifel detect this and undo the cut

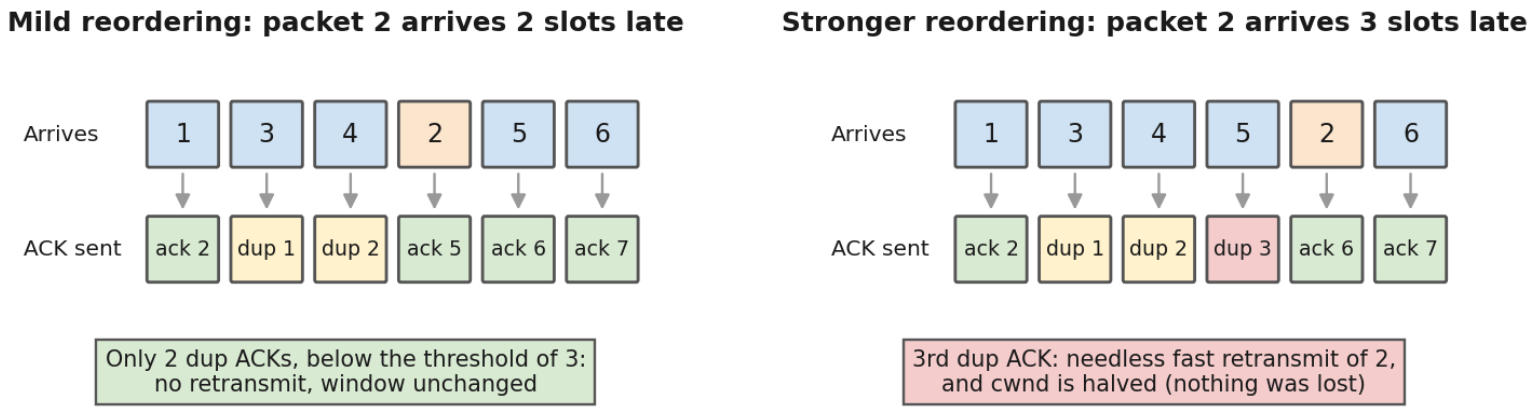
Impact of Packet Reordering

- Causes
  - Link-layer retransmission (parallel HARQ processes, Wi-Fi retries), handoffs, multipath



Impact of Packet Reordering

- Reno/NewReno - fixed threshold of 3 duplicate ACKs
  - Can trigger a needless fast retransmit and window cut
- Some lower layers can reorder before delivery
  - e.g., cellular RLC/PDCP, Wi-Fi block-ack reordering
  - Cost: extra delay while waiting for the missing packet
- RACK-TLP (RFC 8985)
  - Time-based loss detection with adaptive reordering window



Impact of Transmission Errors

- TCP cannot distinguish between packet losses due to congestion and transmission errors
  - Unnecessarily reduces congestion window
  - Throughput suffers





Ideal Behavior

- Ideal TCP behavior
  - Simply retransmit a packet lost due to transmission errors
  - Take no congestion control actions
    - Ideal TCP typically not realizable
- Ideal network behavior
  - Transmission errors should be hidden from the sender
  - Errors should be recovered transparently and efficiently
- Proposed schemes attempt to approximate one of the above two ideals





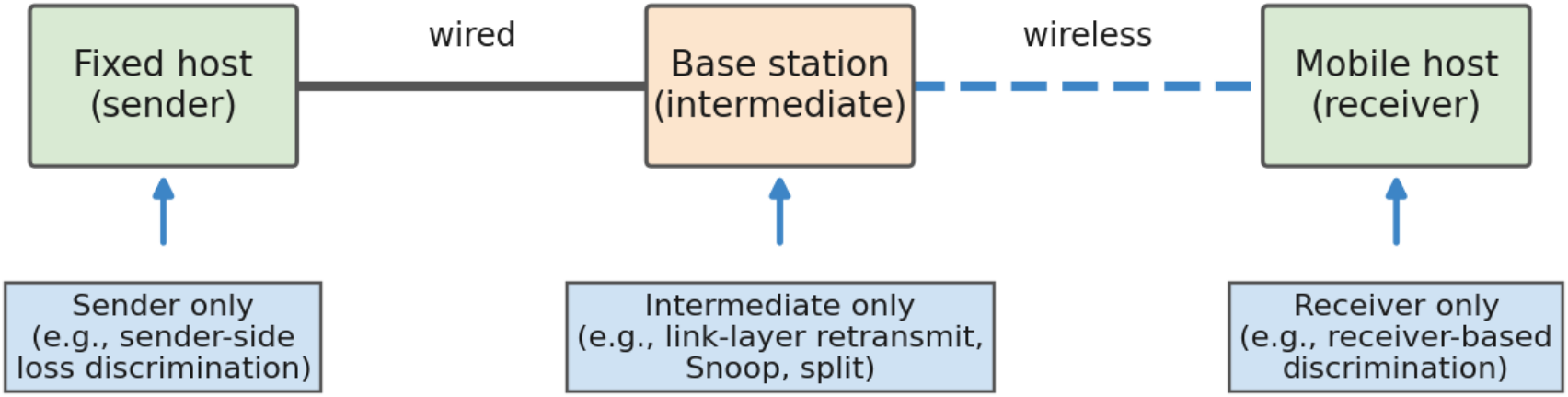
Techniques

- Nature of actions taken to improve performance
  - Hide error losses from the sender
    - Sender is unaware of error-based losses
      - Will not reduce congestion window
  - Let sender know, or determine, cause of packet loss
    - Sender knows about cause of packet loss
      - Will not reduce congestion window



Techniques

- Where modifications are needed
  - At the sender node only
  - At the receiver node only
  - At intermediate node(s) only
  - Combinations of the above



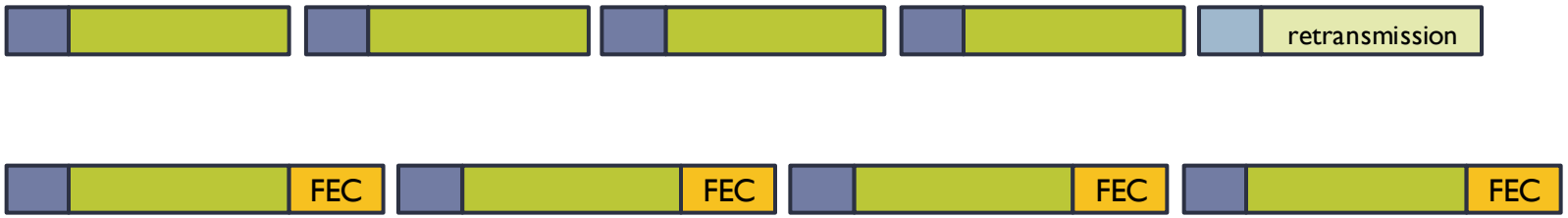
Schemes

- Link level mechanisms
- Split connection approach
- TCP-Aware link layer
- TCP-Unaware approximation of TCP-aware link layer
- Explicit notification
- Receiver-based discrimination
- Sender-based discrimination



Link Layer Mechanisms: Forward Error Correction

- Forward Error Correction (FEC) can be used 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

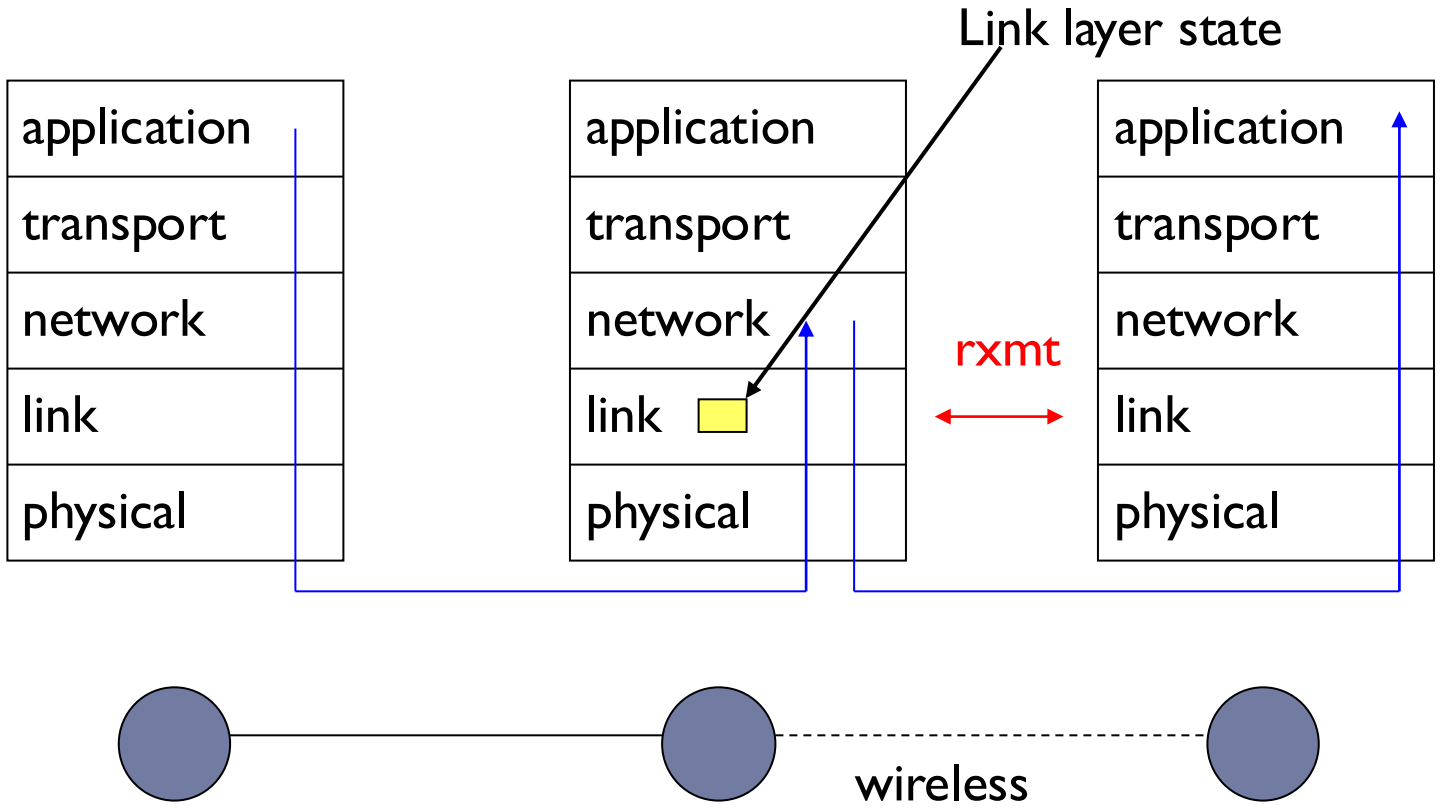


Link Layer Mechanisms: Link Level Retransmissions

- 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

- How many retransmissions at the link level before giving up?
  - Finite bound -- semi-reliable link layer
    - Wi-Fi defaults: retry limit of 7 for short frames, 4 for long frames
  - 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)



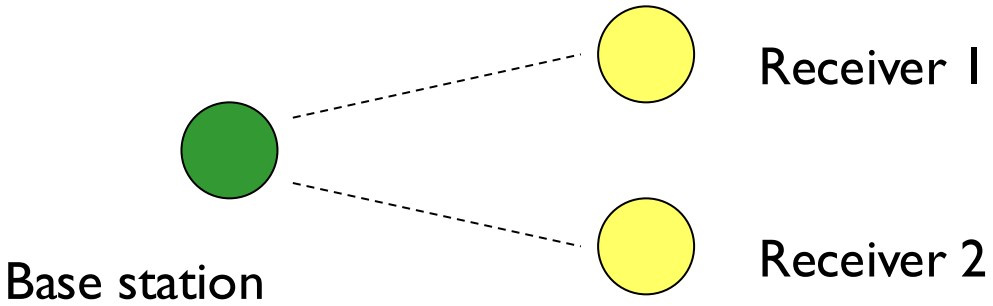
Link Level Retransmissions

- 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

- Retransmissions can cause
  - Head-of-the-line blocking
  - Congestion losses



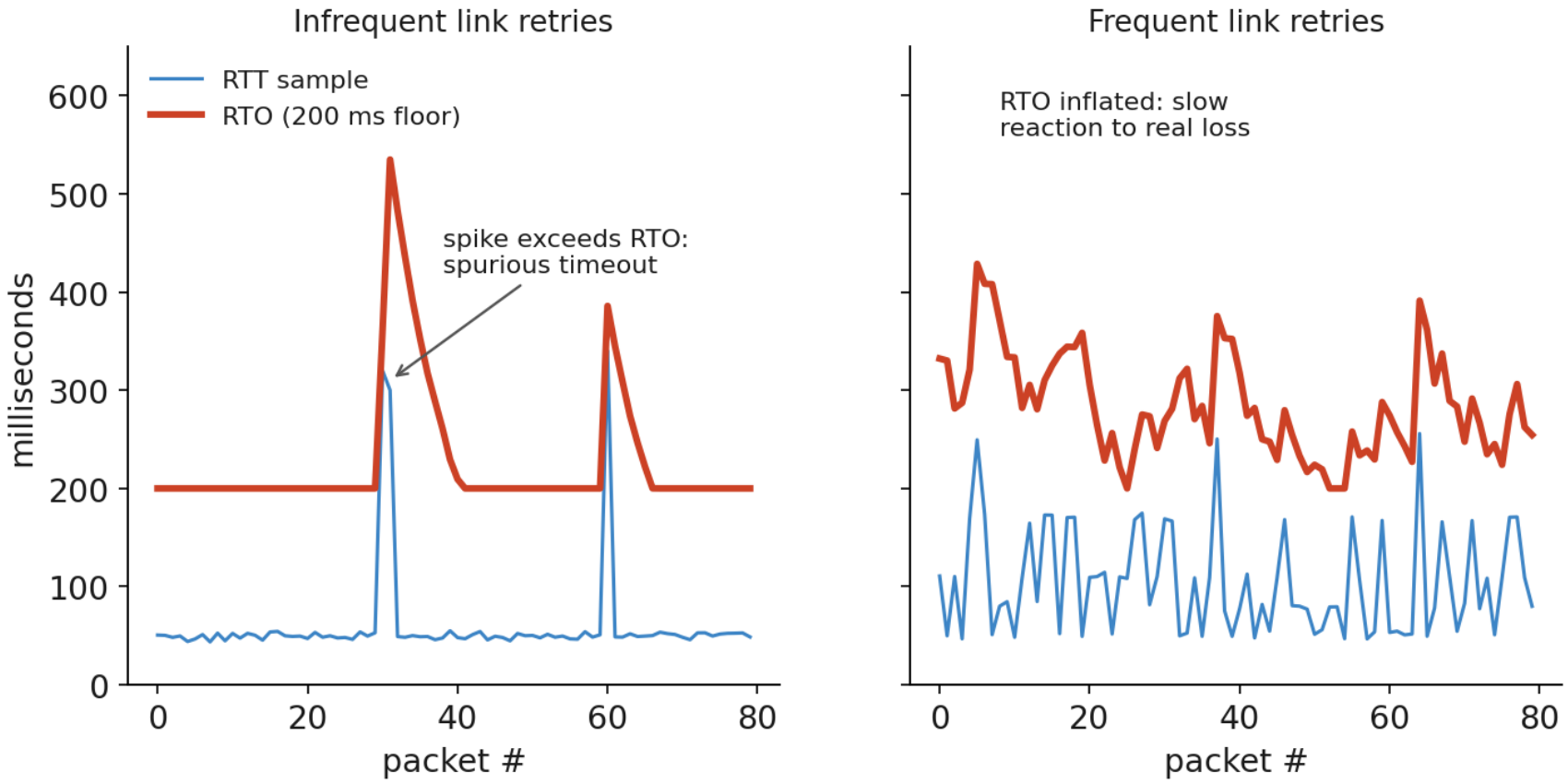
Link Level Retransmissions

- The sender's Retransmission Timeout (RTO)
  - Function of measured RTT (round-trip times)
  - Link level retransmits increase RTT, therefore, RTO
- Infrequent errors
  - RTO will not account for RTT variations due to link level retransmissions
- Frequent errors
  - Increase RTO significantly on slow wireless links





Link Retries Stretch RTT and RTO



Link Level Retransmissions

- Not all connections benefit from retransmissions
  - Audio
- Need to be able to specify requirements on a per-packet basis
  - Should the packet be retransmitted?
  - How many times?
- Need a standard mechanism to specify the requirements



Link Layer Schemes: Summary

- When is a reliable link layer beneficial to TCP performance?
  - If TCP retransmission timeout is large enough to tolerate additional delays due to link level retransmits



Link Layer Mechanisms: Hiding Losses

- Hide wireless losses from TCP sender
- Link layer modifications needed at both ends of wireless link
  - TCP need not be modified

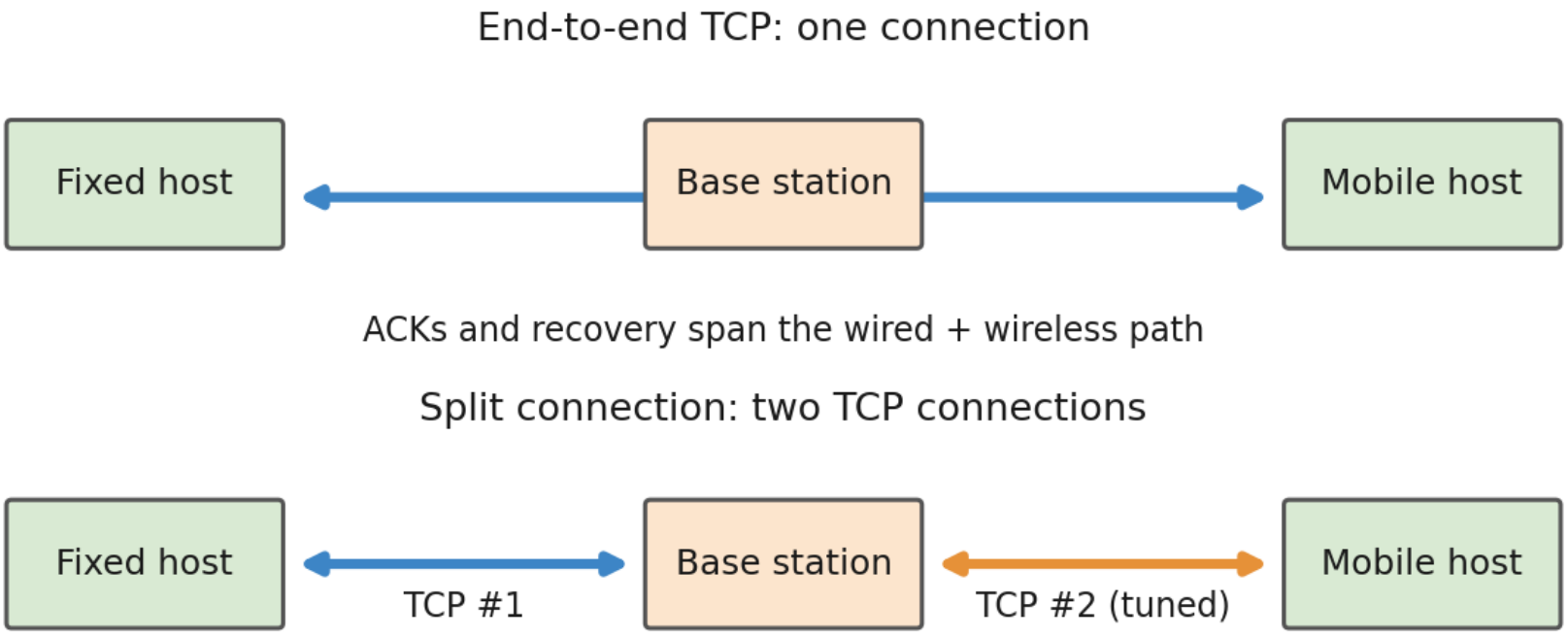


Split Connection Approach

- End-to-end TCP connection is broken into
  - One connection on the wired part of route
  - One over wireless part of the route
- A single TCP connection split into two TCP connections
  - If wireless link is not last on route
    - More than two TCP connections may be needed



Split Connection: One Path, Two Connections

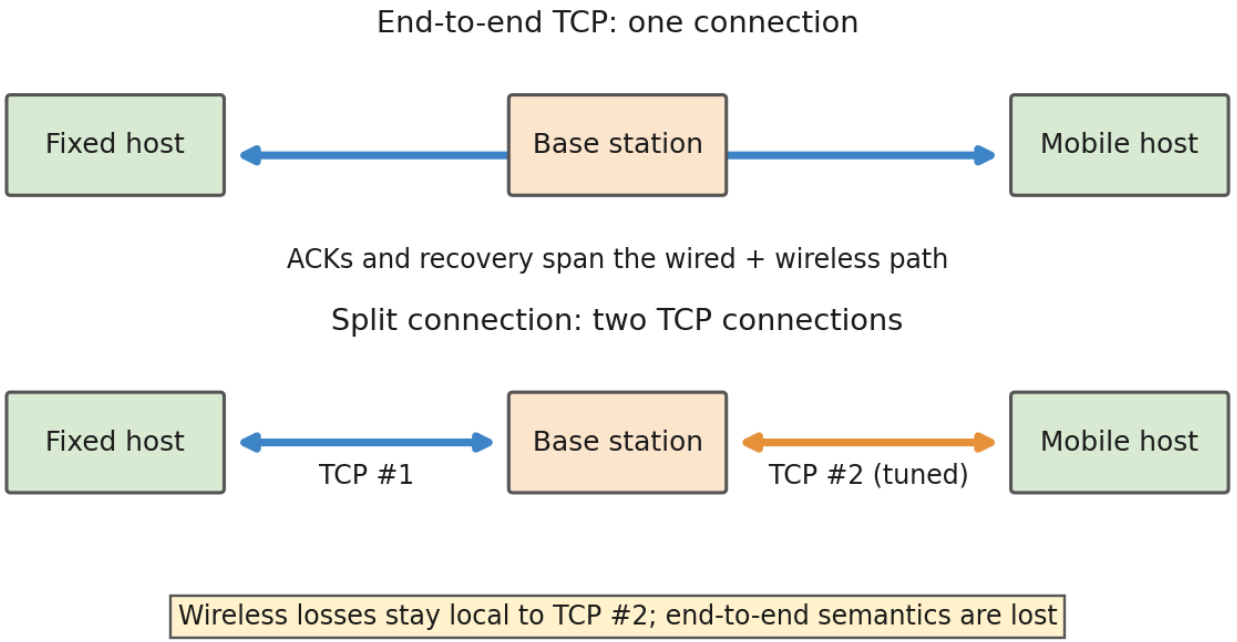


Wireless losses stay local to TCP #2; end-to-end semantics are lost

Splitting keeps wireless losses local, at the price of end-to-end semantics.

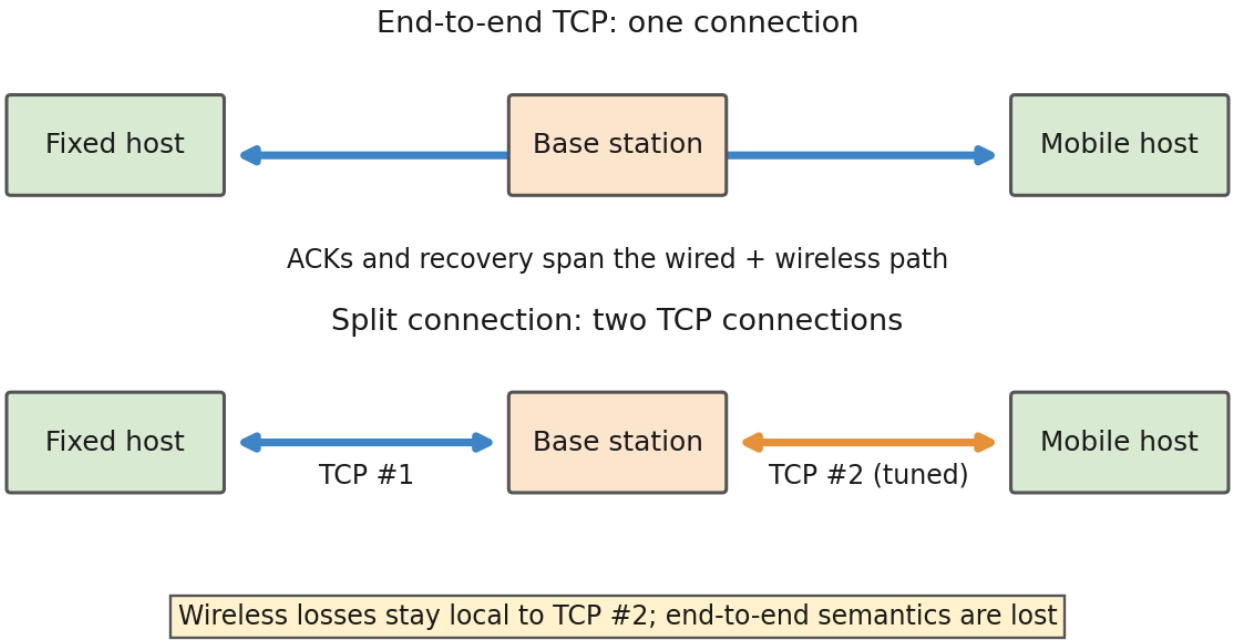
Split Connection Approach

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



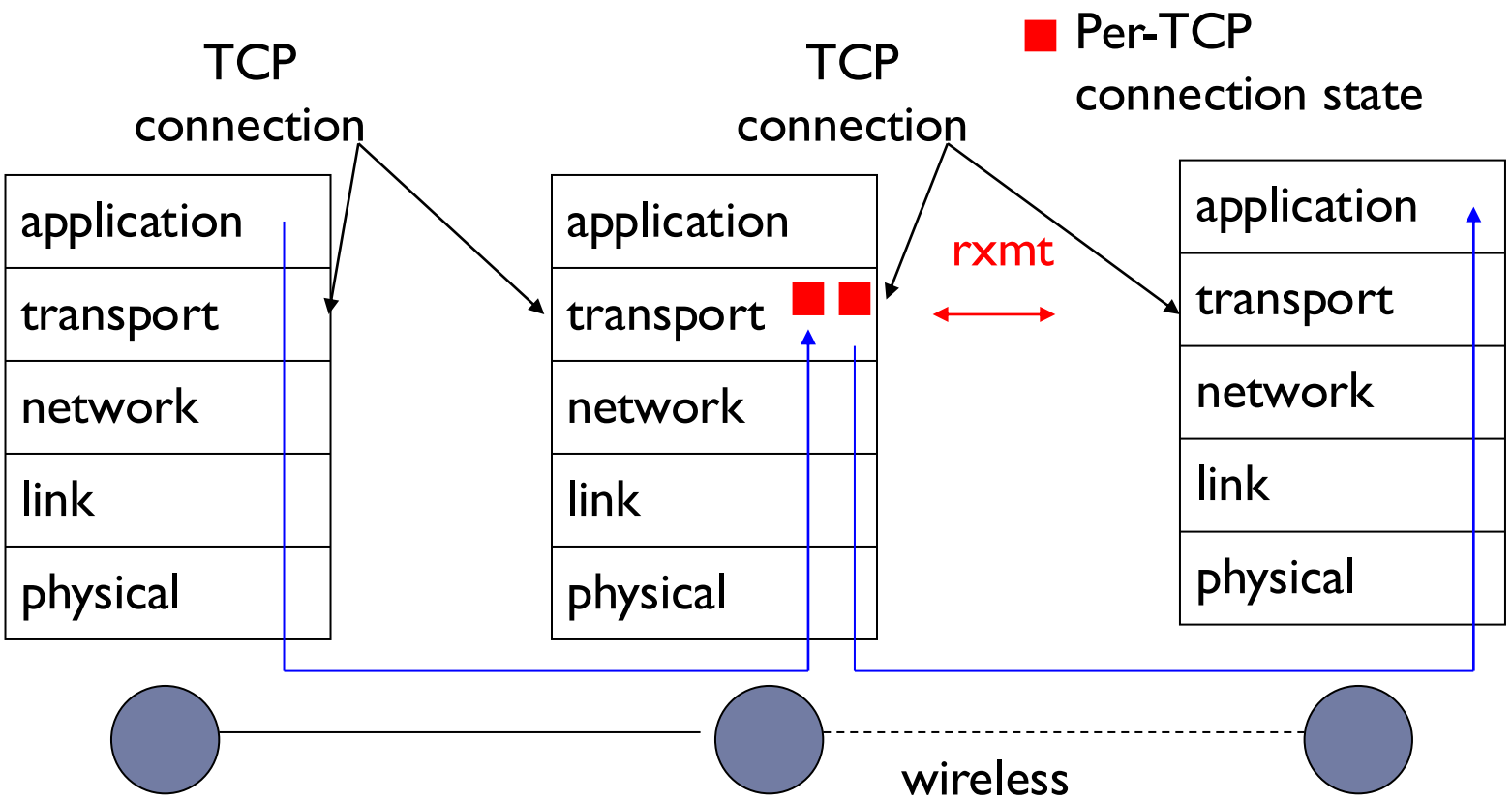
Split Connection Approach

- 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



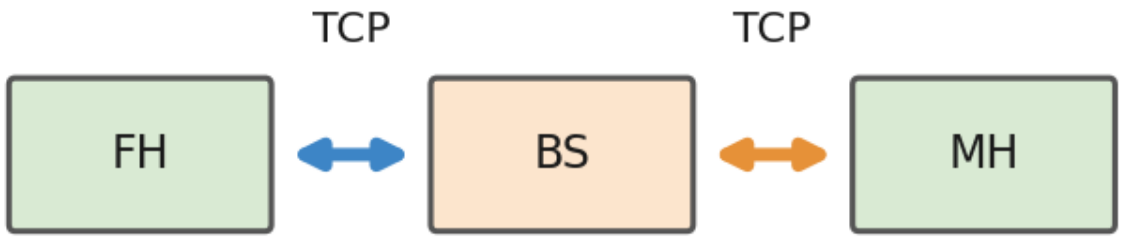
Split Connection Approach

1 TCP = ½ TCP + ½ (TCP or XXX)



Split Connection Approach

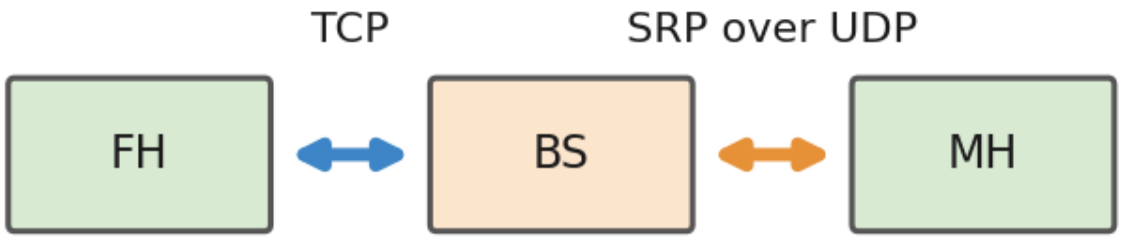
- Indirect TCP
  - FH → BS connection : Standard TCP
  - BS → MH connection : Standard TCP



Both hops run standard TCP
Wireless TCP still sees
loss as congestion

Split Connection Approach

- Selective Repeat Protocol (SRP)
  - FH → BS : standard TCP, BS → FH : SRP on UDP
  - Performance better than Indirect-TCP (I-TCP)
    - Wireless link can be tuned to wireless behavior



Wireless hop uses selective repeat:
tuned to wireless behavior,
better than I-TCP

Split Connection Approach: Advantages

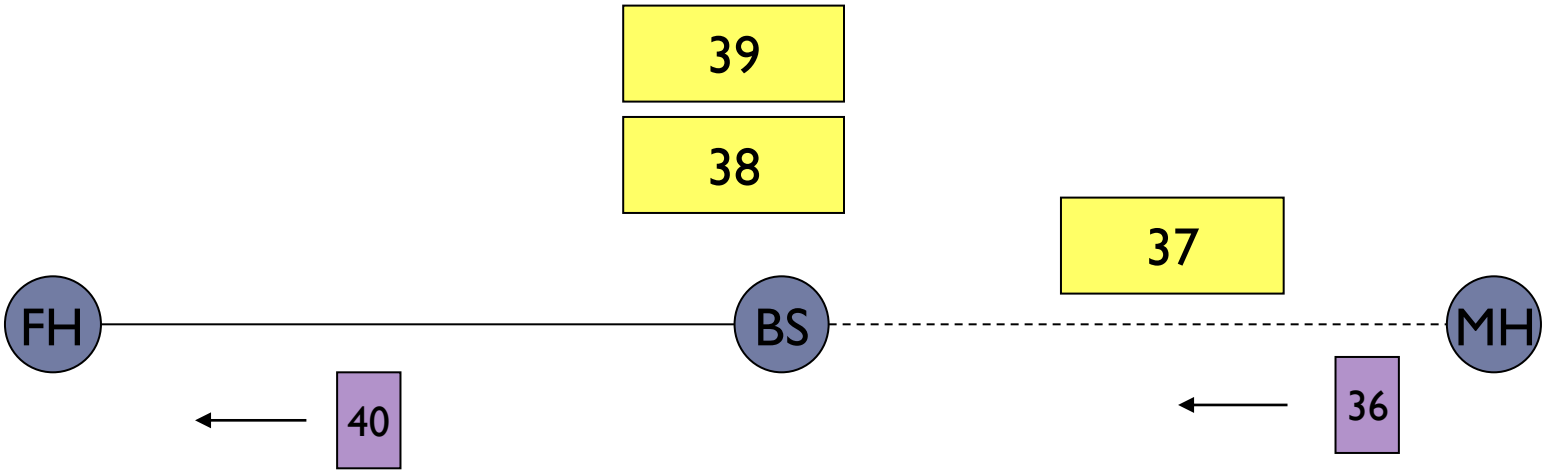
- BS → MH connection can be optimized independent of FH → BS connection
- Local recovery of errors
- Good performance achievable using appropriate BS → MH protocol
  - Standard TCP on BS → MH performs poorly
  - Selective acks improve performance





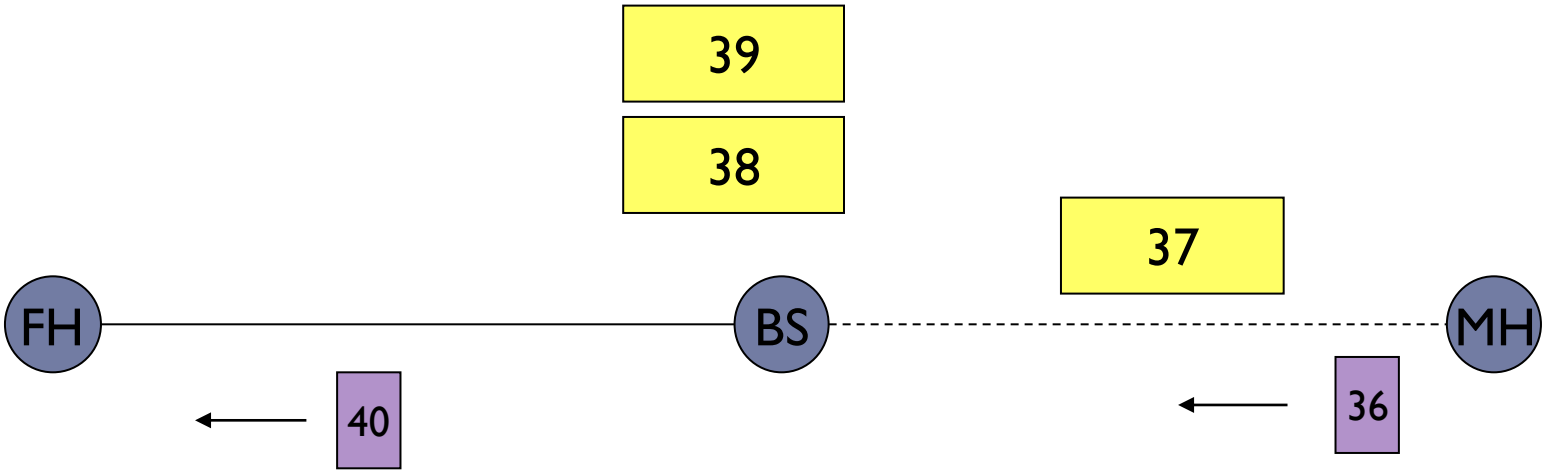
Split Connection Approach : Disadvantages

- 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



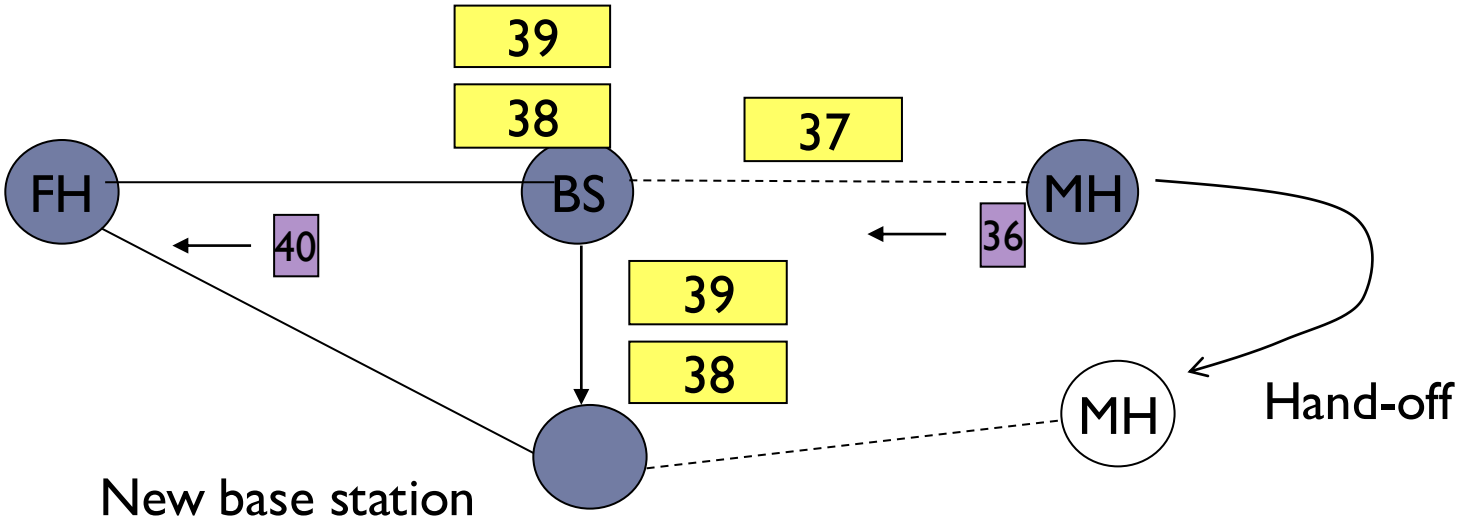
Split Connection Approach : Disadvantages

- BS 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



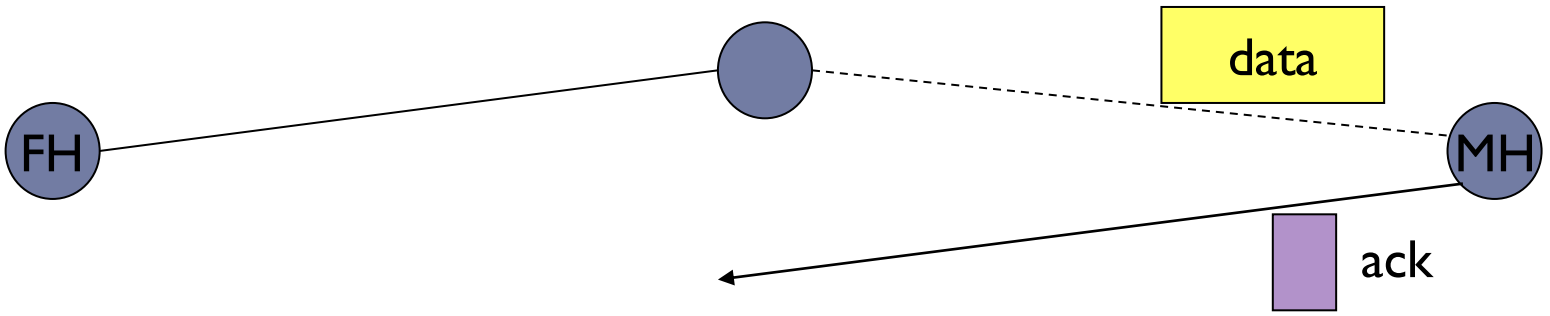
Split Connection Approach : Disadvantages

- Buffer space needed at BS for each TCP connection
  - BS buffers tend to get full with a slow wireless link slower
    - One window of data on wired connection could be stored at 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 one path (e.g., a cable or cellular downlink), acks returning on a different path



Split Connections Today

- The idea survives, in different forms
  - CDNs and edge proxies terminate TCP and TLS near the client
  - Cellular and satellite networks use performance-enhancing proxies (PEPs)
- Same trade-offs as before
  - End-to-end semantics are lost, and the proxy keeps state
- QUIC encrypts transport headers, so middlebox PEPs cannot see or split it





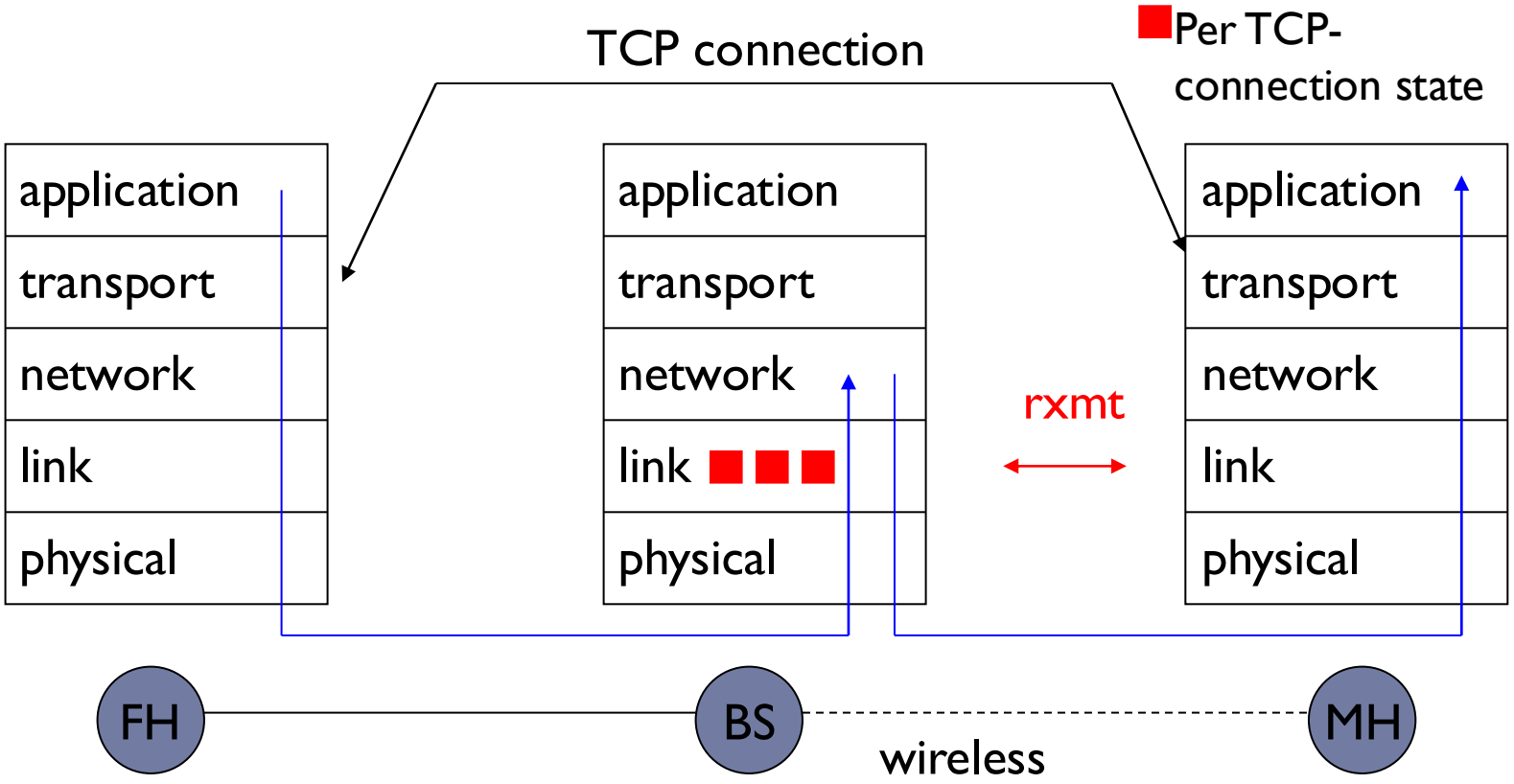
TCP-Aware Link Layer

- Snoop Protocol
  - Retains local recovery of Split Connection approach and link level retransmission schemes
  - Improves on split connection
    - End-to-end semantics retained
    - Soft state at base station, instead of hard state



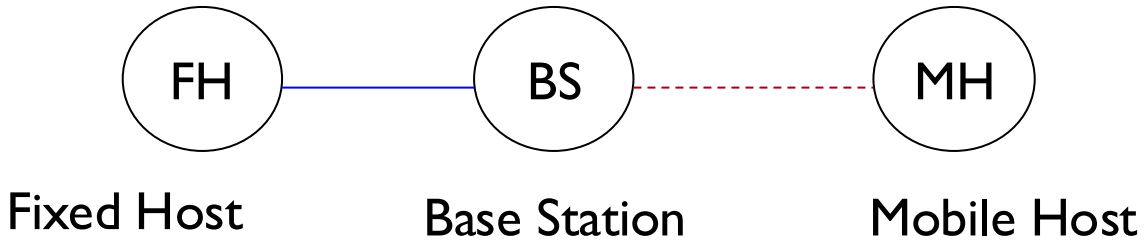


Snoop Protocol



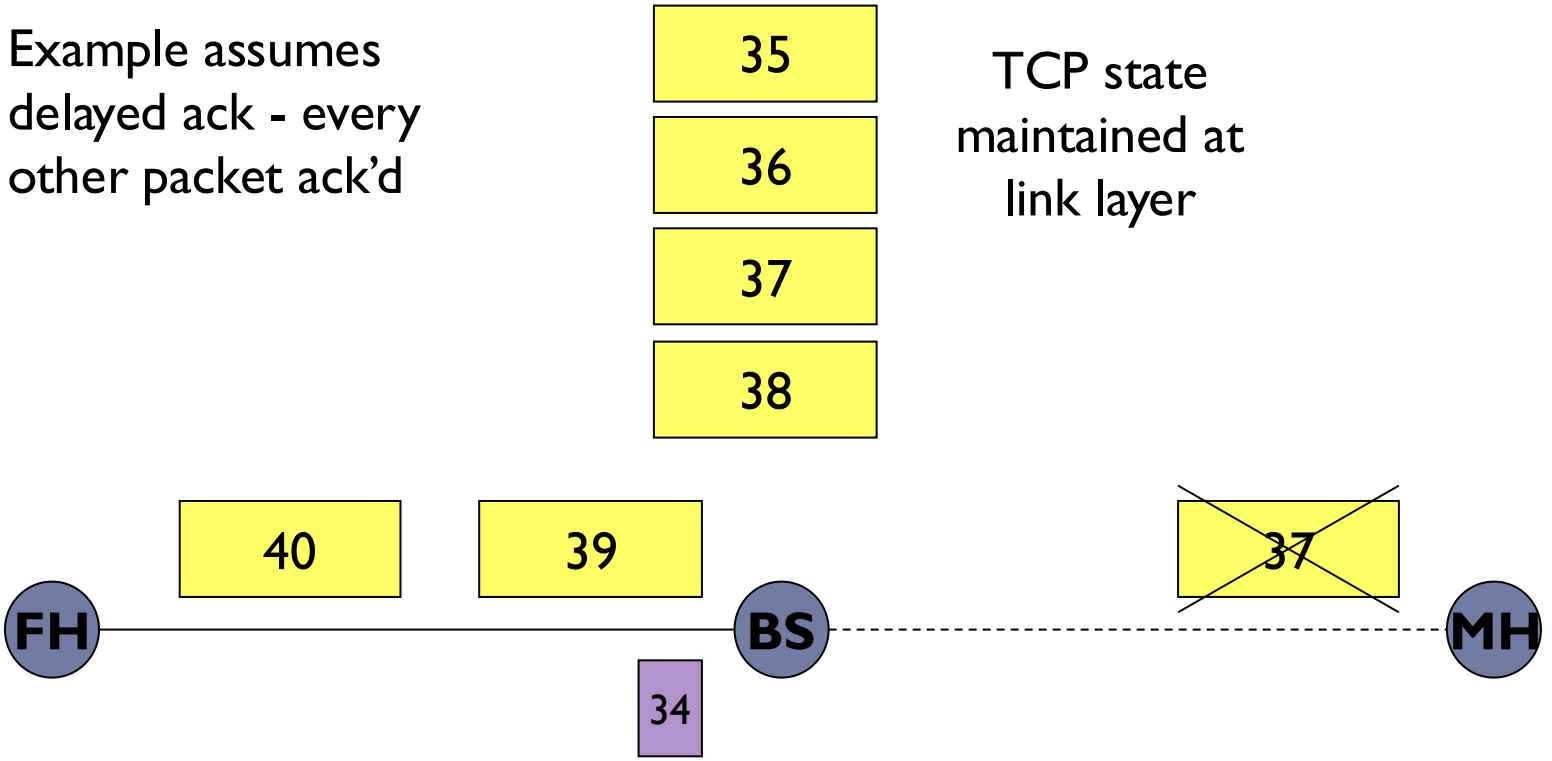
Snoop Protocol

- 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

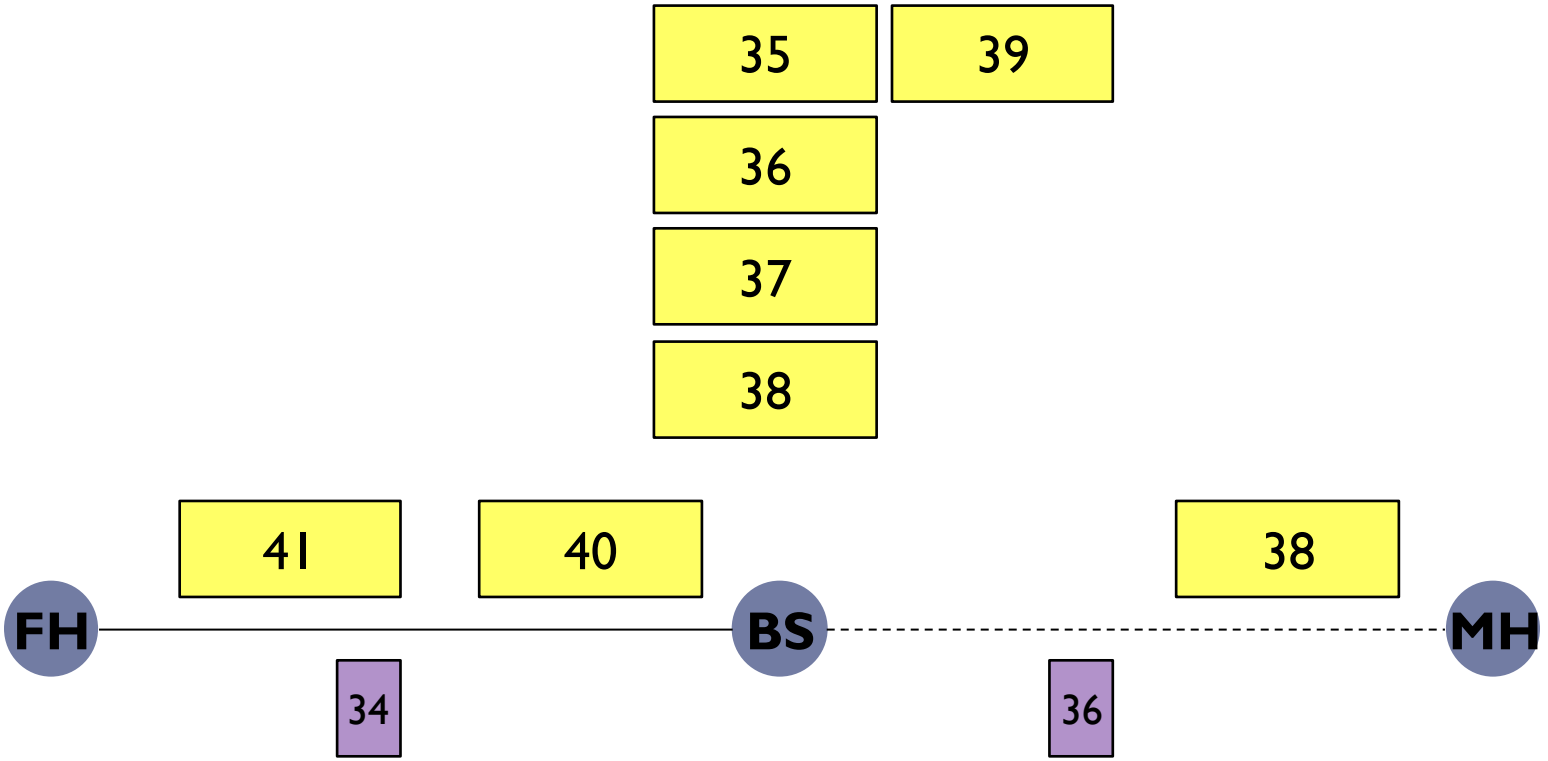


Snoop Protocol

Example assumes delayed ack - every other packet ack'd

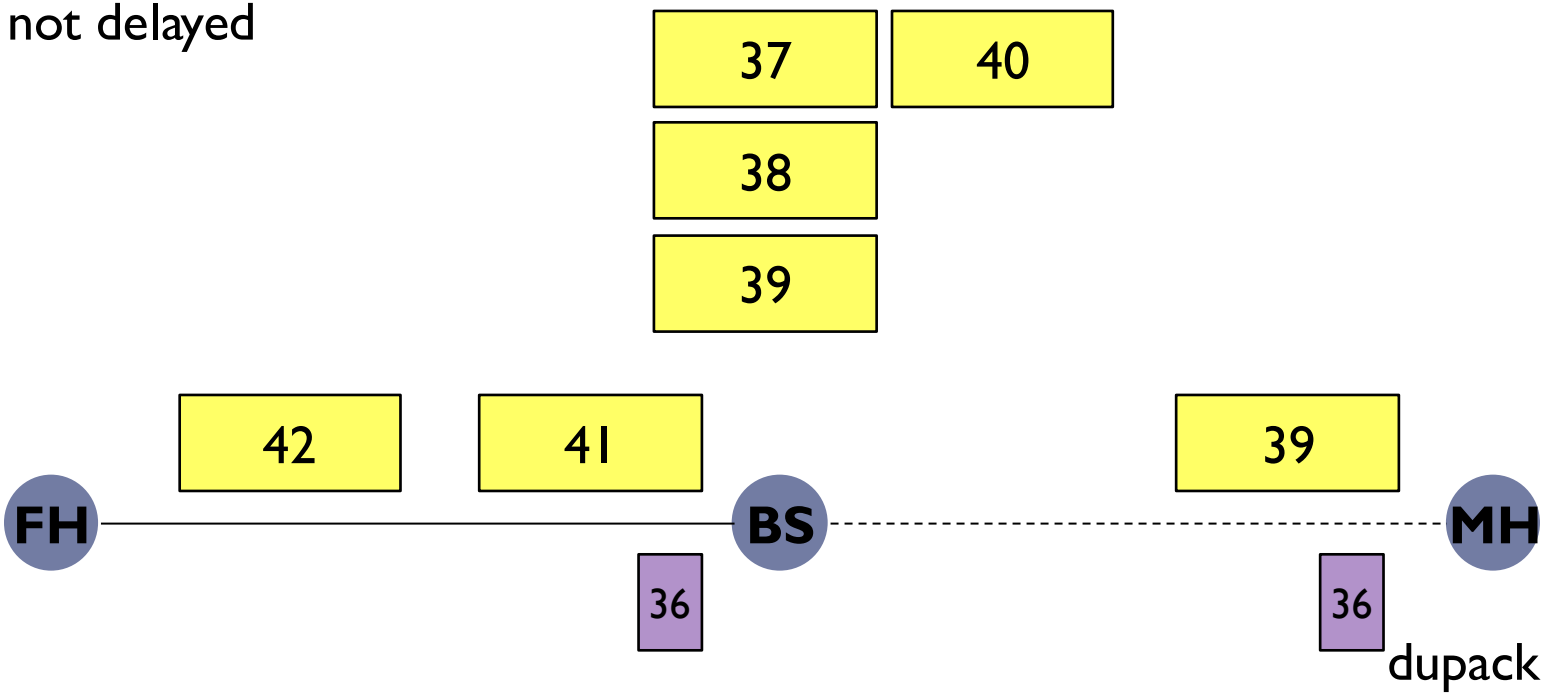


Snoop Protocol

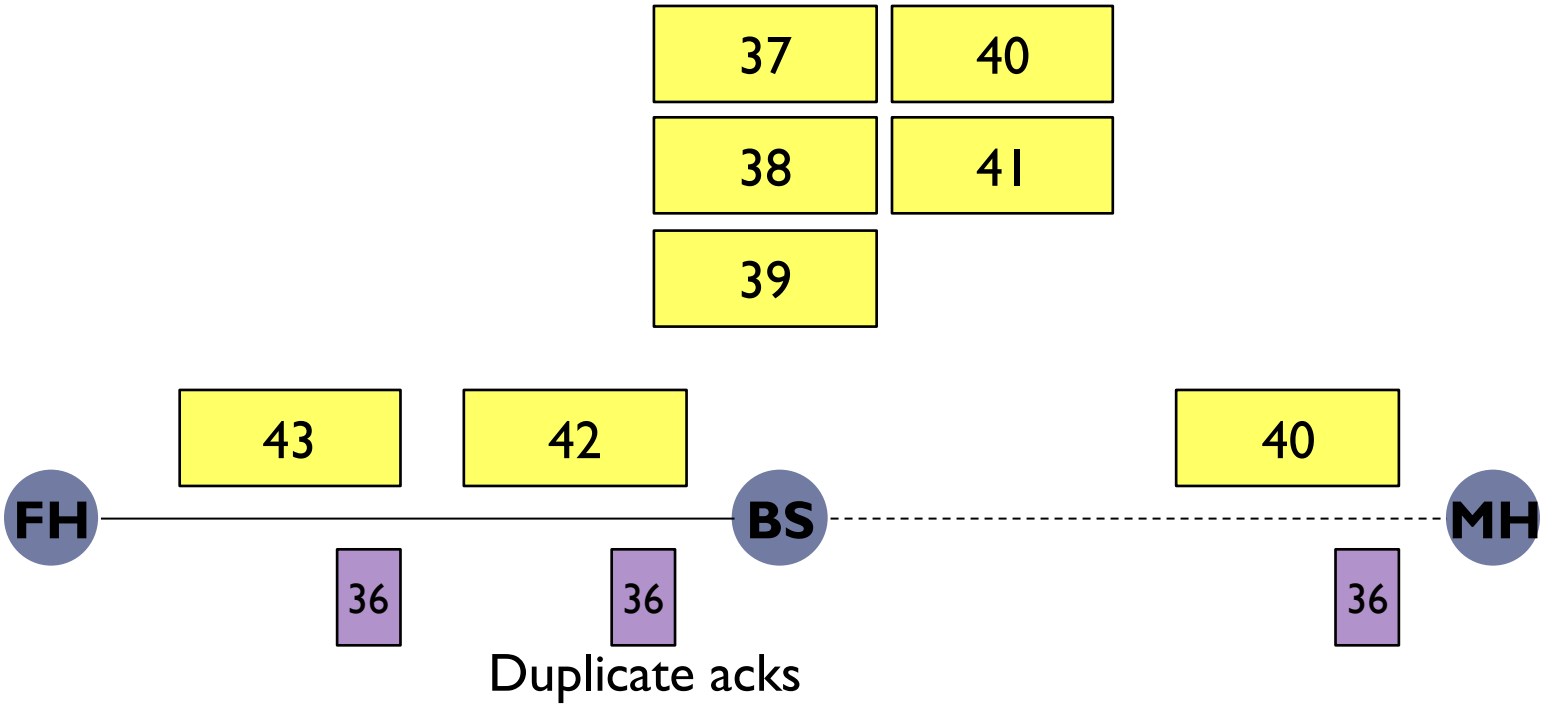


Snoop Protocol

Duplicate acks are not delayed



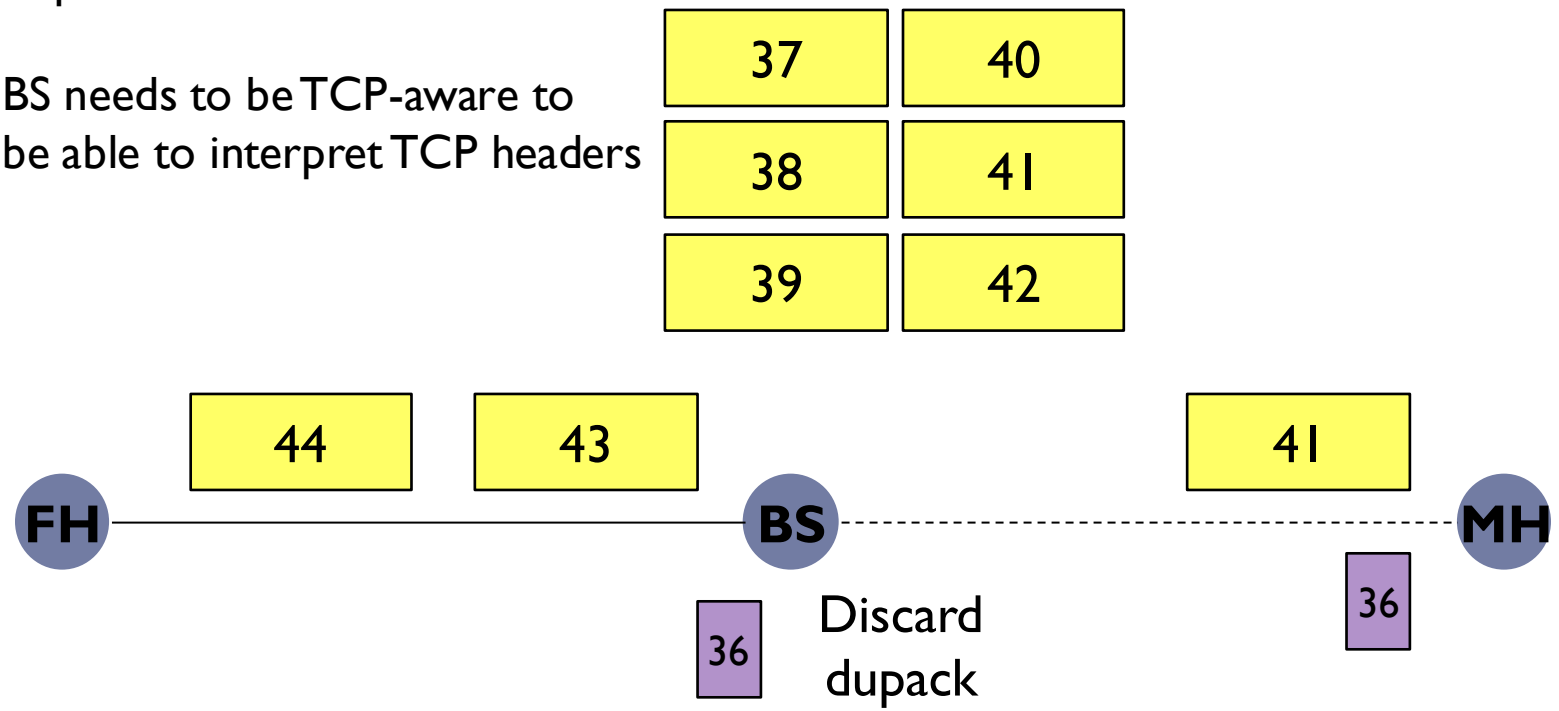
Snoop Protocol



Snoop Protocol

Dupack triggers retransmission of packet 37 from base station

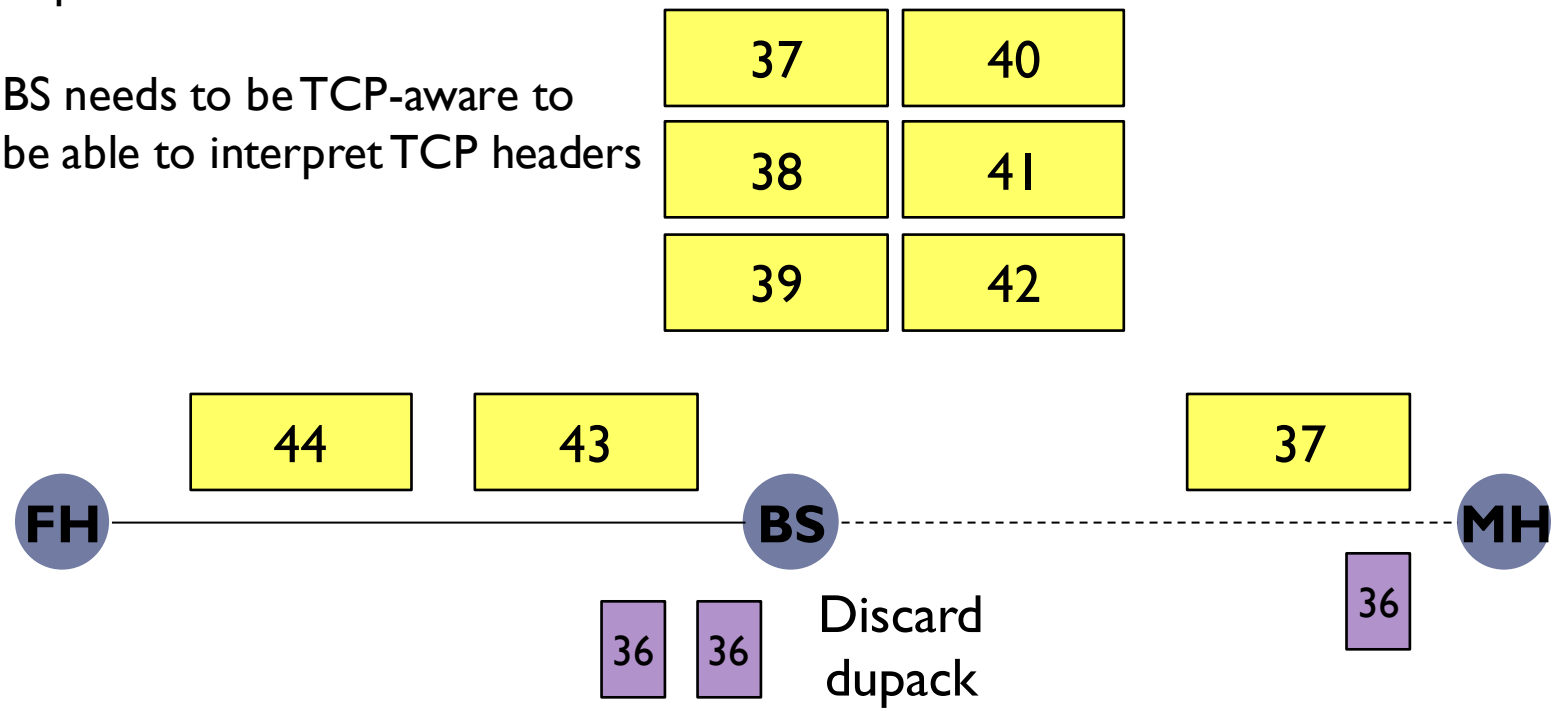
BS needs to be TCP-aware to be able to interpret TCP headers



Snoop Protocol

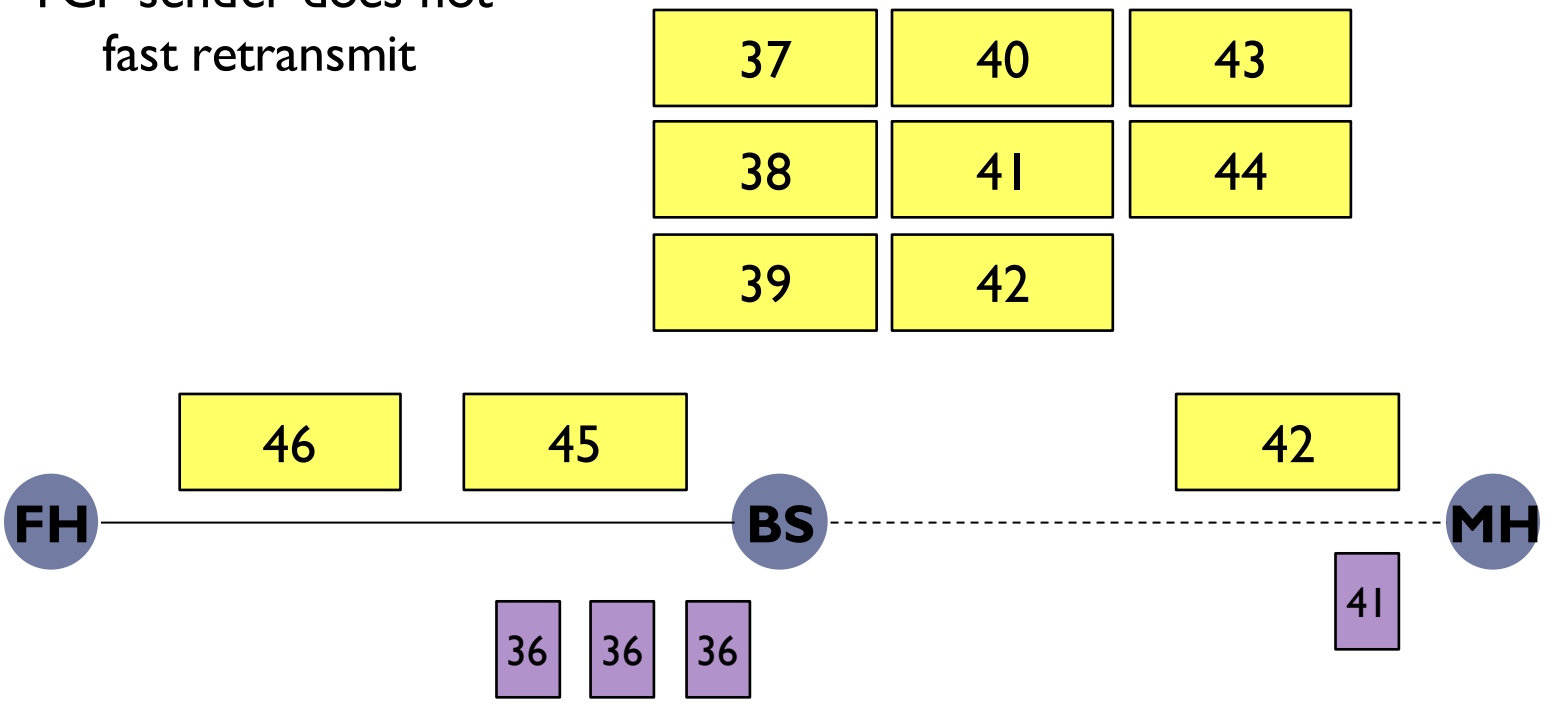
Dupack triggers retransmission of packet 37 from base station

BS needs to be TCP-aware to be able to interpret TCP headers



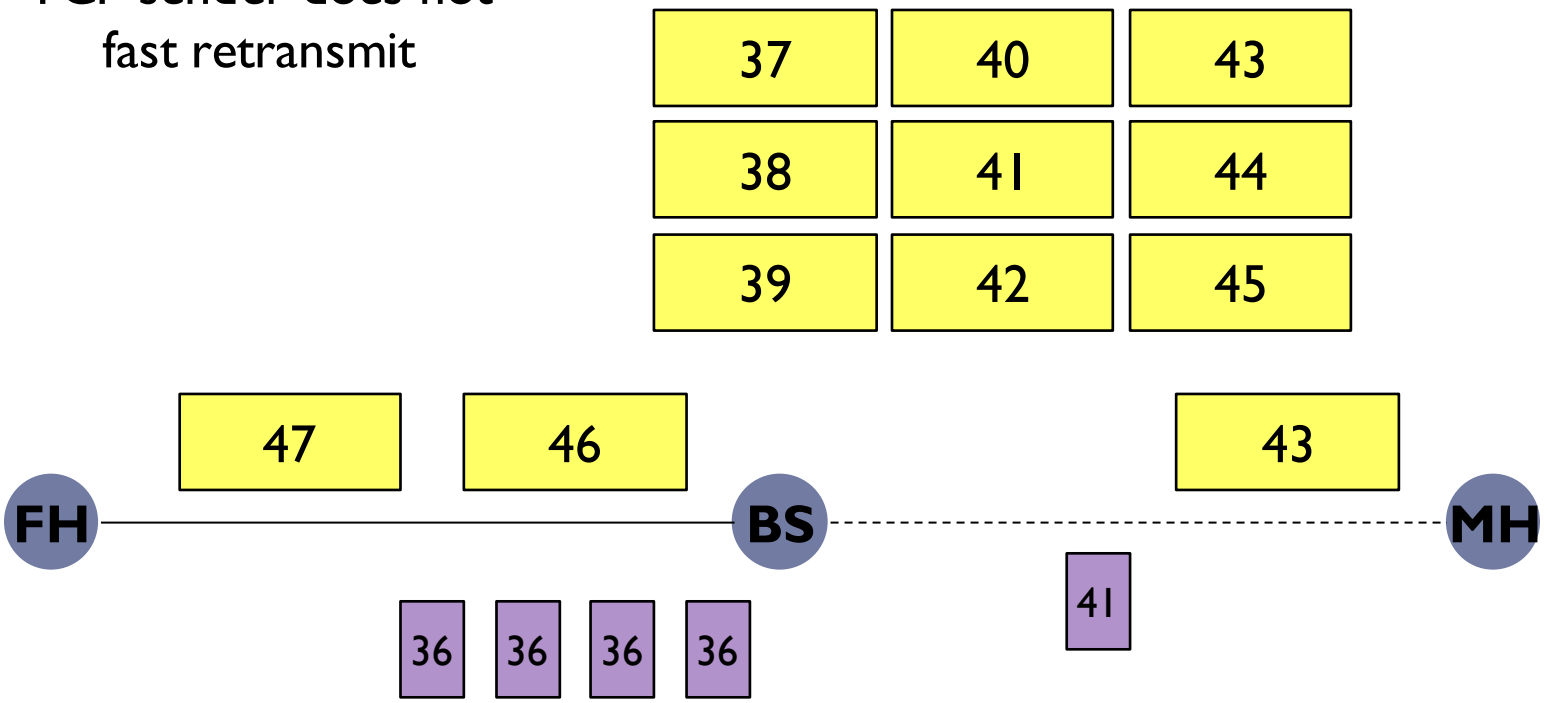
Snoop Protocol

TCP sender does not fast retransmit

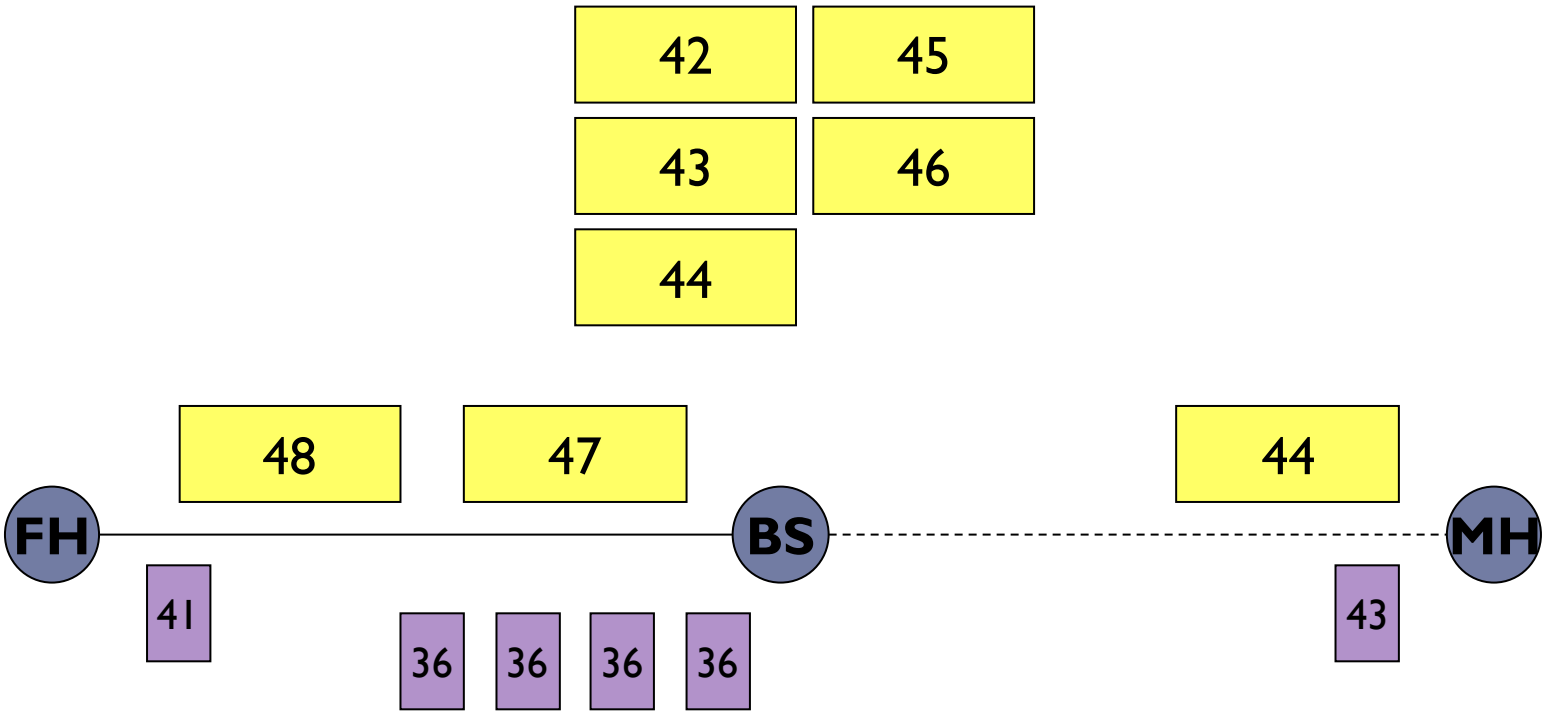


Snoop Protocol

TCP sender does not fast retransmit



Snoop Protocol



Snoop Protocol: When Beneficial?

- Snoop
  - Prevents fast retransmit despite transmission errors on the wireless link
- If wireless link level delay-bandwidth product is less than 4 packets
  - Simple (TCP-unaware) link level retransmission scheme can suffice
  - Since delay-bandwidth product is small
    - Retransmission scheme can deliver the lost packet without resulting in 3 dupacks from the TCP receiver



Snoop Protocol: Advantages

- High throughput
  - Performance further improved using selective acks
- Local recovery from wireless losses
- Fast retransmit not triggered at sender
- 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); QUIC encrypts its transport headers too
- Cannot be used if TCP data and TCP acks traverse different paths (both do not go through the base station)



Explicit Notification Schemes

- General Philosophy
  - Approximate Ideal TCP behavior
    - TCP sender should simply retransmit a packet lost due to transmission errors
    - No congestion control actions
  - Wireless node
    - Determines that packets are lost due to errors
    - Informs sender using an explicit notification
  - Sender - on notification
    - Does not reduce congestion window
    - Retransmits lost packet





Explicit Notification Schemes

- Motivated by the Explicit Congestion Notification (ECN) proposals
- Variations proposed in literature differ in
  - Who sends explicit notification
  - How they know to send the explicit notification
  - What the sender does on receiving the notification





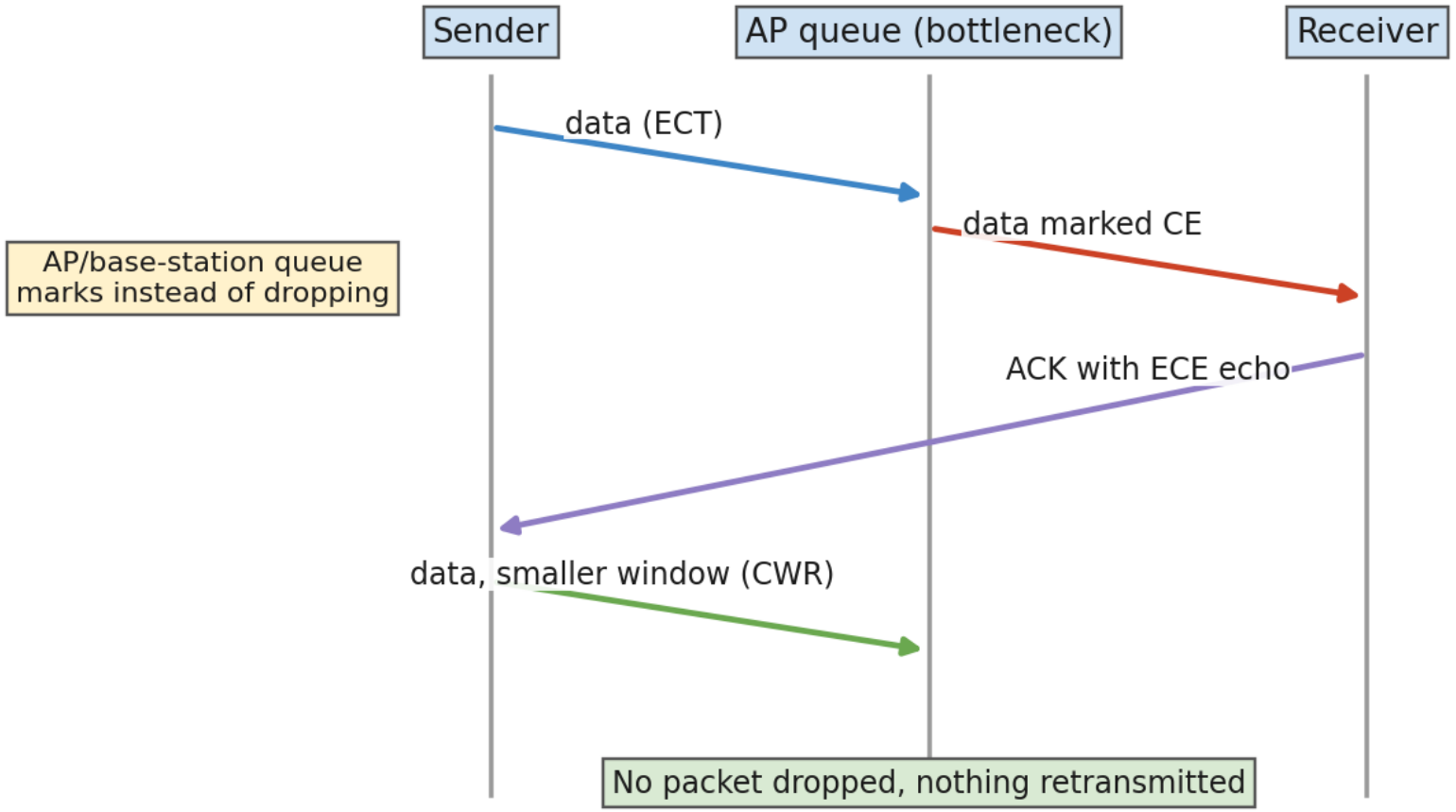
ECN and L4S: A Signal That Is Not Loss

- ECN (RFC 3168)
  - Bottleneck marks packets instead of dropping them
  - Receiver echoes the mark; sender reduces its window, with no loss and no retransmission
  - On wireless, marks cannot be confused with radio errors
- L4S (RFCs 9330-9332)
  - Scalable congestion control reacts in proportion to marks
  - Dual-queue AQM at the bottleneck keeps queueing delay at the millisecond level
- For wireless the marking must happen in the AP or base-station queue
  - Needs deployment support exactly where the rate varies





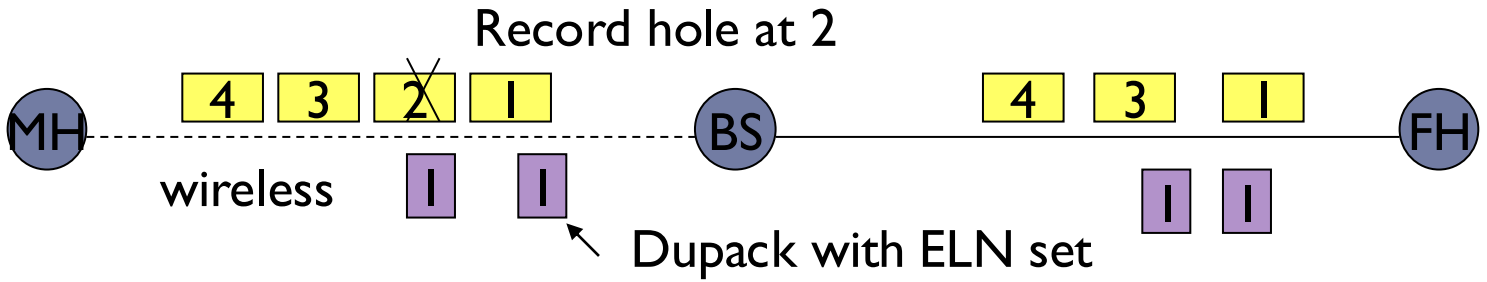
ECN: Marking Instead of Dropping



A mark signals congestion with no loss and no retransmission.

Explicit Loss Notification – MH as TCP Sender

- Wireless link first on the path from sender to receiver
- Base station
  - Keeps track of holes in the packet sequence
  - Dupack from receiver
    - Base station compares the dupack sequence number with recorded holes
    - If there is a match, an ELN bit is set in the dupack
- Sender - Dupack with ELN set
  - Retransmit packet
  - Do not reduce congestion window

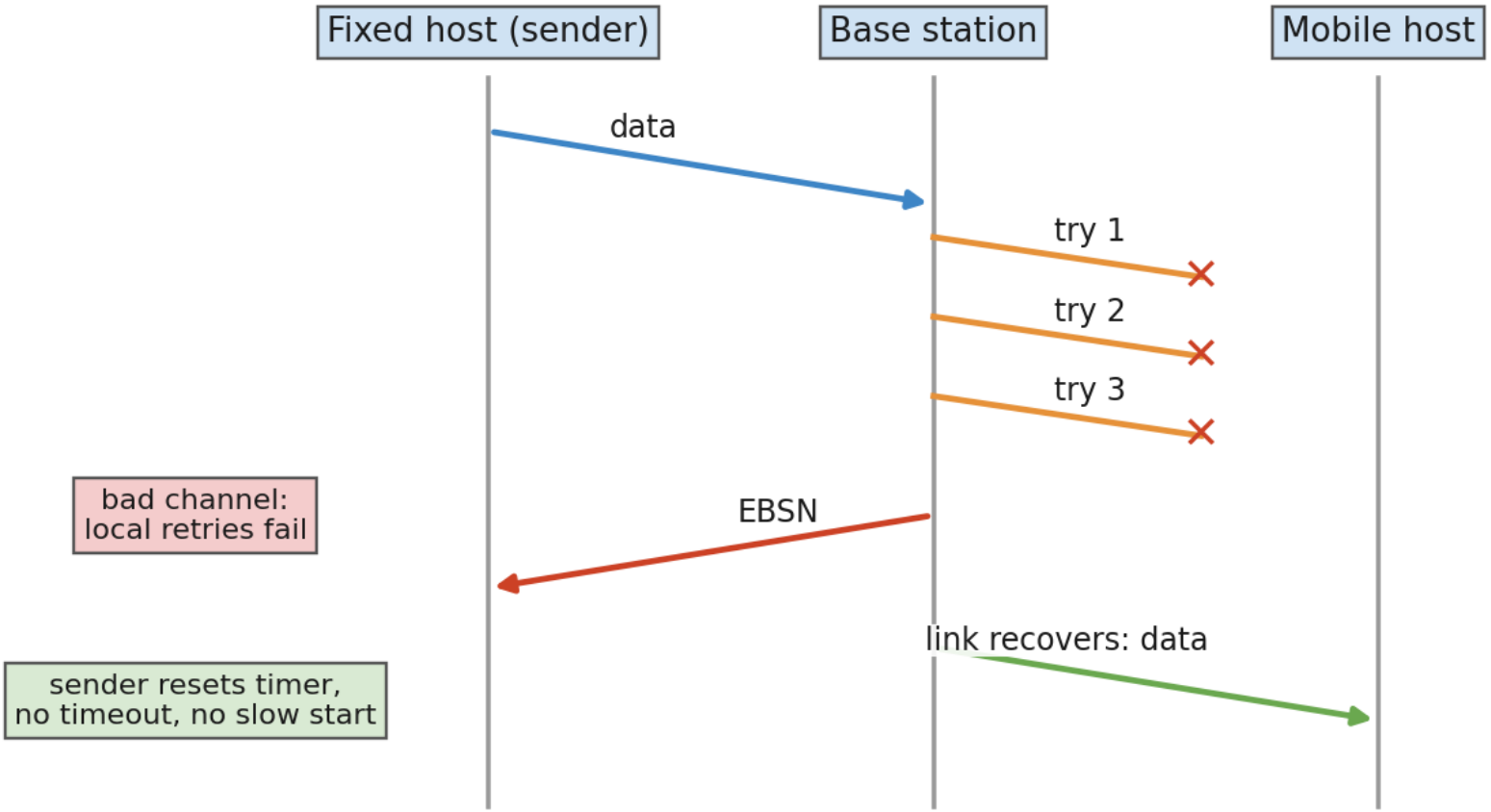


Explicit Loss Notification – MH as TCP Sender

- Base station
  - Attempts to deliver packets to the MH using a link layer retransmission scheme
  - If packet cannot be delivered using a small number of retransmissions
    - BS sends a Explicit Bad State Notification (EBSN) message to TCP sender
- When TCP sender receives EBSN, it resets its timer
  - Timeout delayed, when wireless channel in bad state



EBSN Timeline

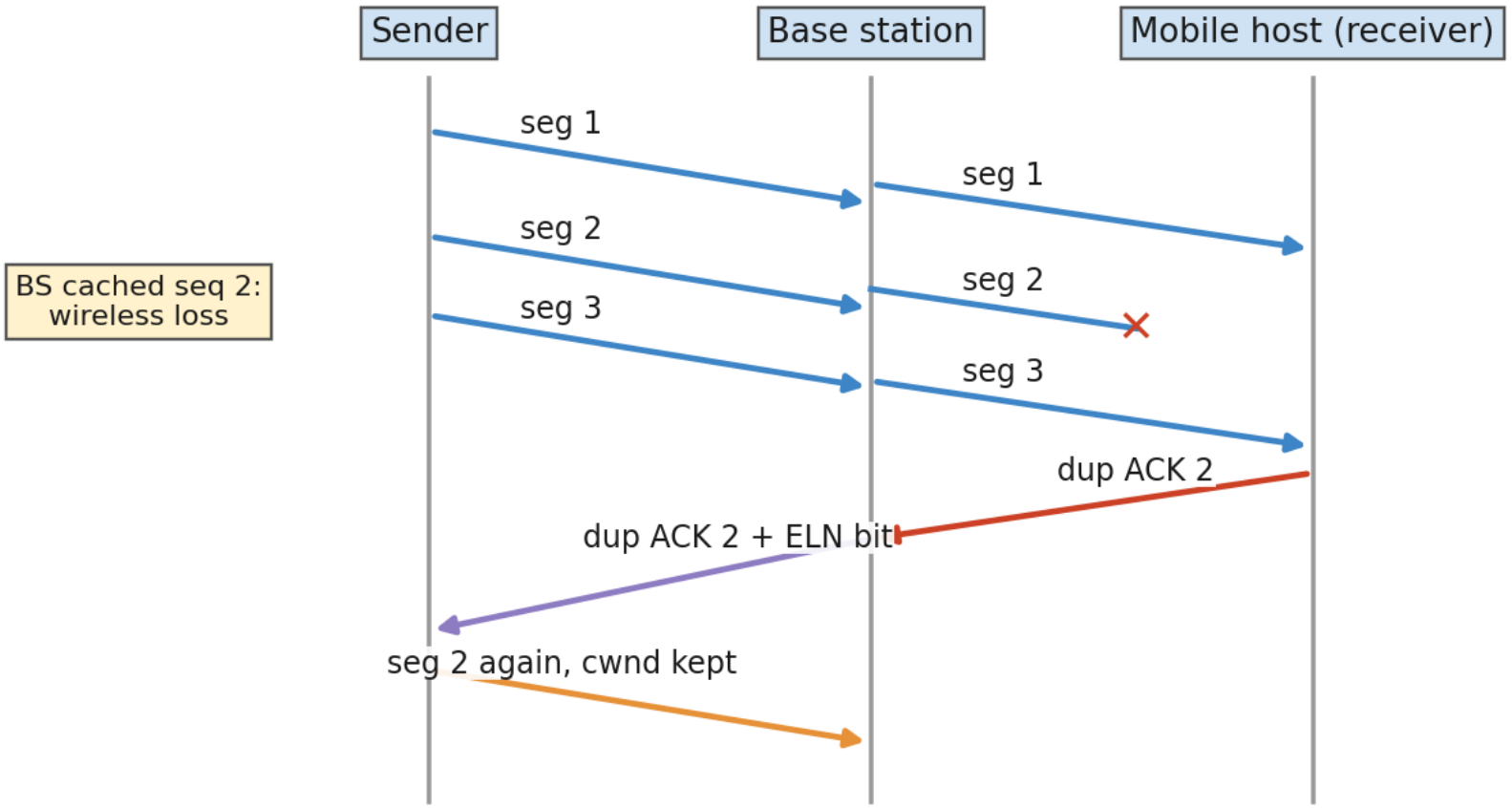


Explicit Loss Notification - MH as TCP receiver

- Approximate hypothetical ELN
- Base station
  - Caches TCP sequence numbers
  - Does not cache data packets
- If sequence number for lost packet is cached at the base station
  - Duplicate acks are tagged with ELN bit before being forwarded to sender
- Sender takes appropriate action on receiving ELN



ELN Timeline



Handoffs and TCP

- During an AP or cell handoff, packets in flight are lost or delayed
- No ACKs arrive, so the RTO fires
  - Each timeout doubles the RTO, so TCP can stay idle after the link returns
- Fast retransmit after handoff
  - When the mobile host is the receiver, it sends three duplicate ACKs once the handoff completes
  - The sender retransmits at once instead of waiting for the backed-off timer
- Better still: avoid the gap
  - Fast roaming (802.11r), make-before-break links, MPTCP and QUIC migration





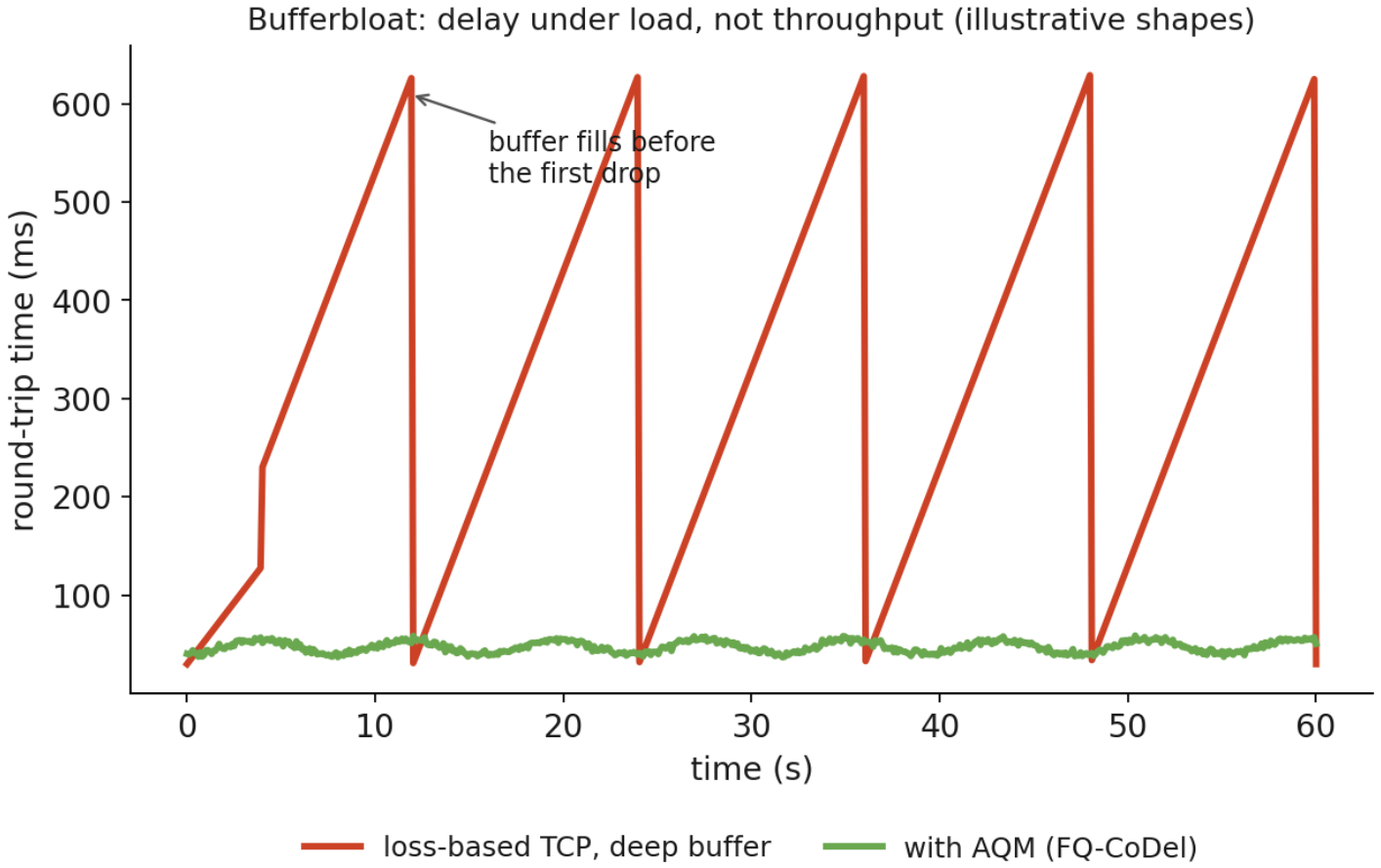
Bufferbloat on Wi-Fi and Cellular

- Deep buffers plus a variable link rate create long queues
  - Loss-based TCP fills any buffer before it sees a drop
  - Result: seconds of added delay under load, with little loss
- Fixes
  - Active queue management: CoDel and FQ-CoDel (RFC 8290)
  - Per-station queues and airtime fairness (Linux mac80211)
  - ECN, and congestion control that limits queueing (BBR)
- The symptom is latency under load, not throughput





Bufferbloat: Delay Under Load



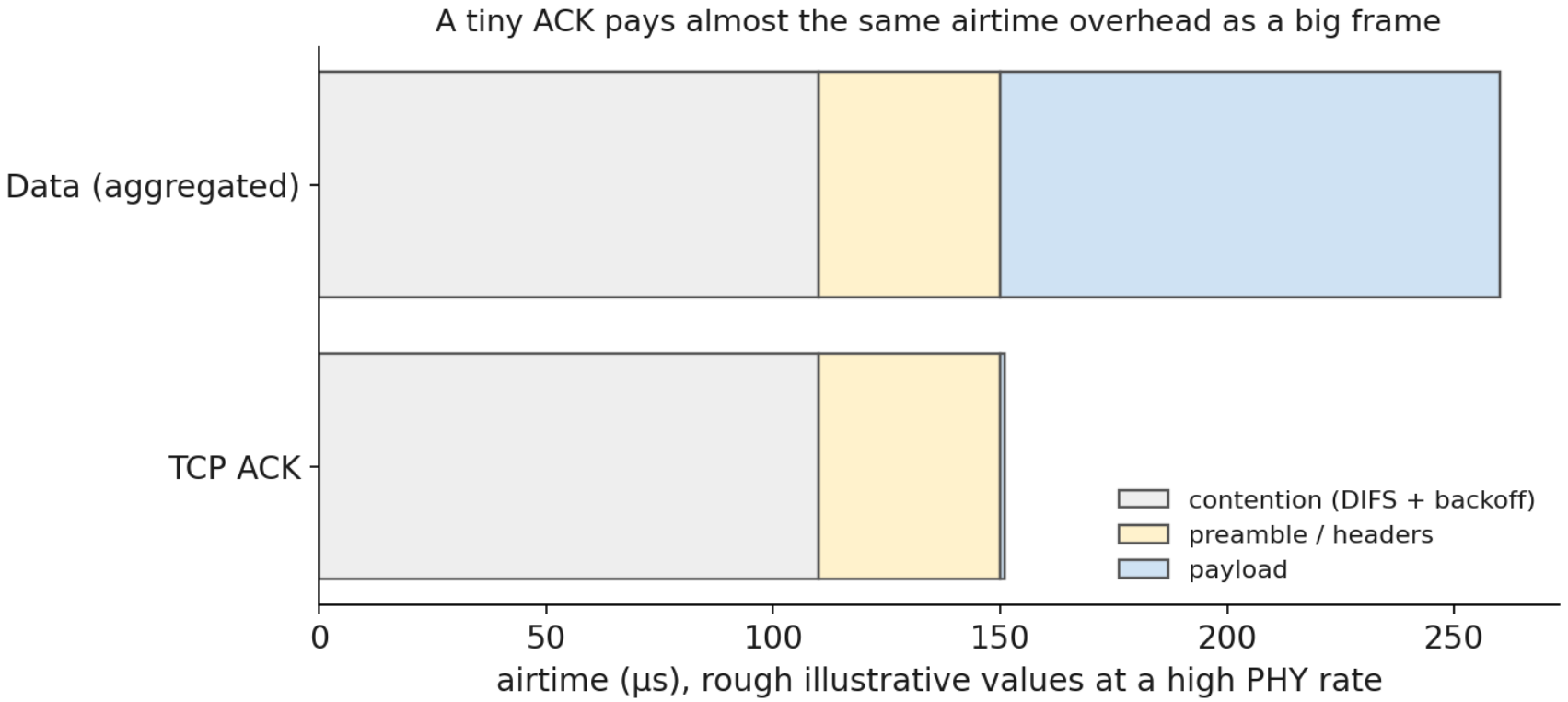
Illustrative shapes: a deep buffer fills before loss; AQM keeps delay low.

ACKs Compete for Airtime

- A TCP download sends ACKs on the uplink
  - On Wi-Fi each ACK needs its own channel access, preamble, and link-layer ACK
  - At high PHY rates a tiny ACK still pays full channel-access and preamble overhead: a large share of the airtime
- Delayed ACKs halve this cost
- ACK thinning and aggregation, in the client or the AP, cut it further
- Matters most on asymmetric or busy links

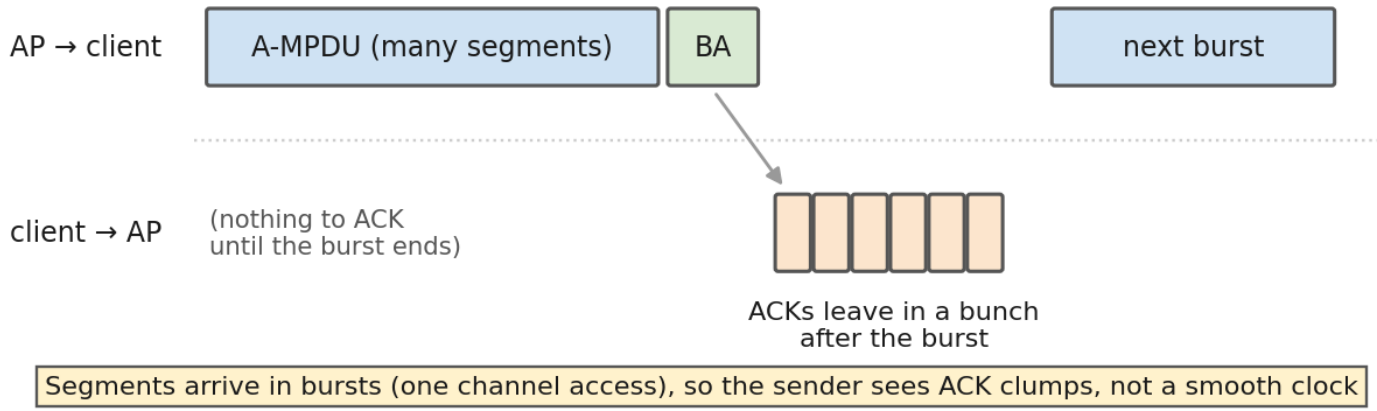


ACKs Cost Airtime



Frame Aggregation and TCP

- 802.11n and later aggregate many frames into one A-MPDU (one channel access, one block ACK)
- TCP data arrives in bursts, and ACKs return in bursts: "ACK clumping"
  - The sender's self-clocking is disturbed, so it sends in bursts and queues spike
- Larger aggregates raise throughput but add latency and jitter
- Smoothing: sender pacing, AP queue management, ACK thinning



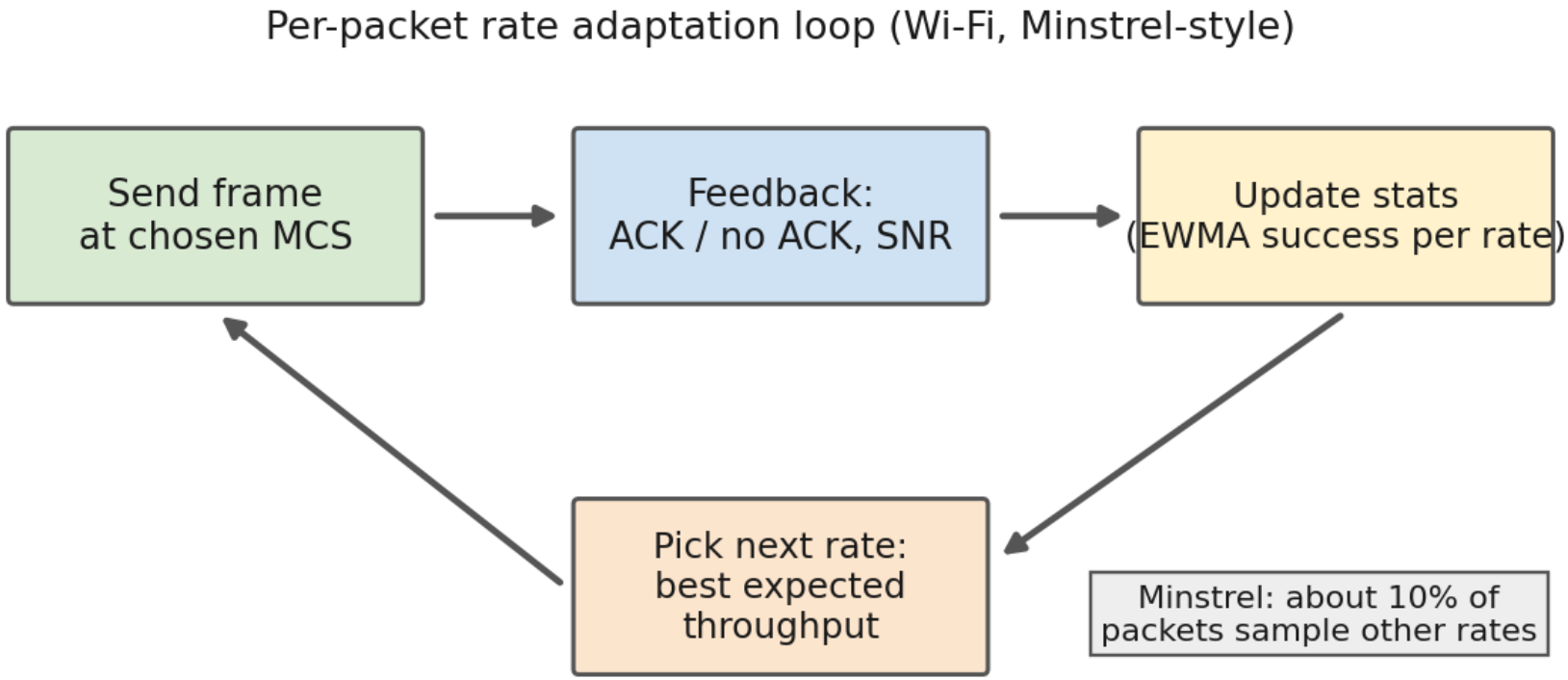
Rate Adaptation: What the Link Layer Does

- The radio picks a PHY rate (MCS) per packet from feedback: ACK success, SNR or CQI
- Wi-Fi example: Minstrel (Linux mac80211)
  - Keeps an EWMA of success probability per rate and picks the best expected throughput
  - Spends about 10% of packets sampling other rates; refreshes about 10 times per second
  - Retry chain sized so a packet is delivered or dropped within about 26 ms
- Cellular: link adaptation targets about 10% block error on the first transmission; HARQ handles the rest
- The goal is link throughput. The algorithm knows nothing about TCP



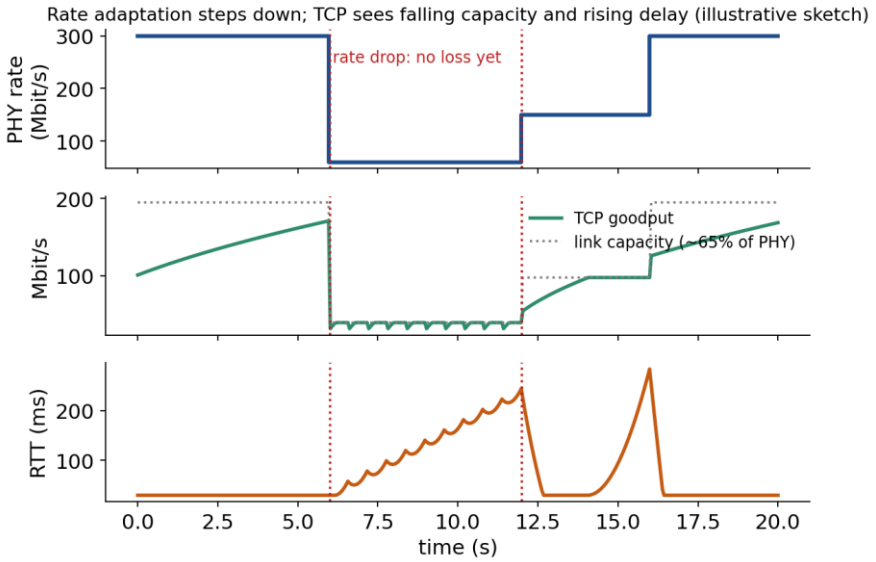


The Rate Adaptation Loop



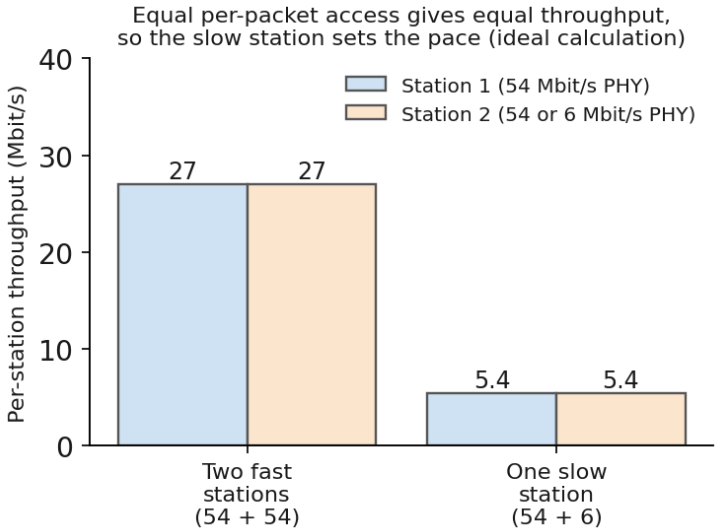
Rate Adaptation Meets TCP

- A rate drop is a capacity drop, with no loss at first
- TCP keeps sending at the old rate, so the AP queue grows and RTT rises
- Loss shows up late (queue overflow) or never (link retries hide it)
- Timescales differ
  - Rate adaptation reacts in milliseconds to 100s of ms
  - TCP reacts per RTT; delay-aware control (BBR) senses the drop sooner



Pathologies: Collisions and the Rate Anomaly

- Collisions look like channel errors
  - A lost ACK makes the algorithm lower the rate, so frames last longer and collisions rise
- Rate anomaly (Heusse et al.)
  - Equal packet opportunities mean a slow station takes most of the airtime
  - Fast stations are dragged down to the slow station's throughput
- TCP fairness does not fix it: each flow gets similar throughput, not similar airtime
- Fix at the AP: per-station queues and an airtime-fairness scheduler (Linux mac80211)



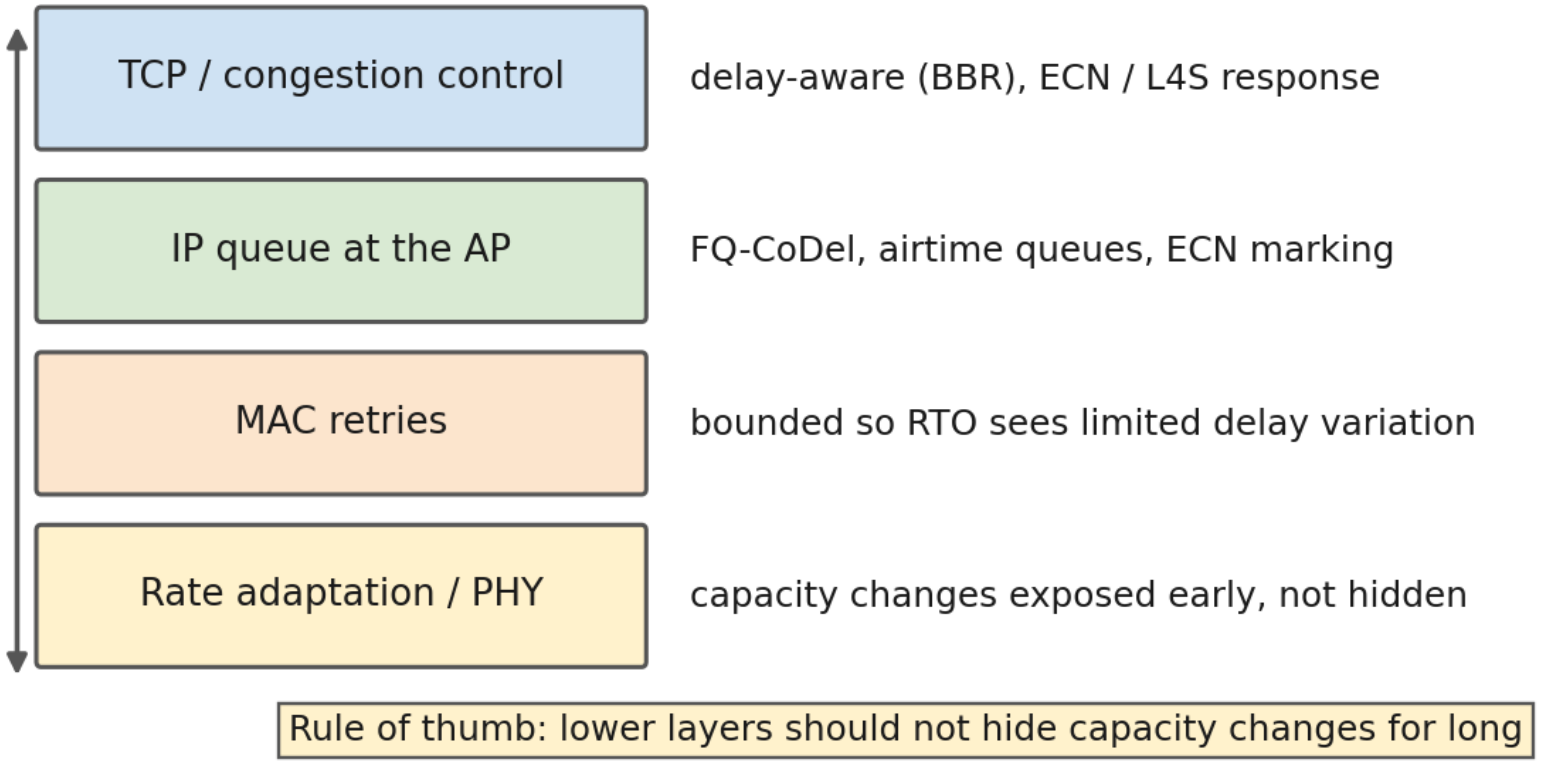
Making Rate Adaptation and TCP Cooperate

- Bound link-layer retries so TCP's RTO sees limited delay variation
- Queue management at the AP (FQ-CoDel, airtime queues) turns a capacity drop into an early signal instead of seconds of delay
- ECN/L4S marks signal queue growth without loss
- Delay-aware congestion control (BBR) tracks the changing bottleneck rate
- Cross-layer hints (rate or airtime info to TCP) have been proposed but are rarely deployed
- Rule of thumb: lower layers should not hide capacity changes for long; TCP should not read delay spikes as loss





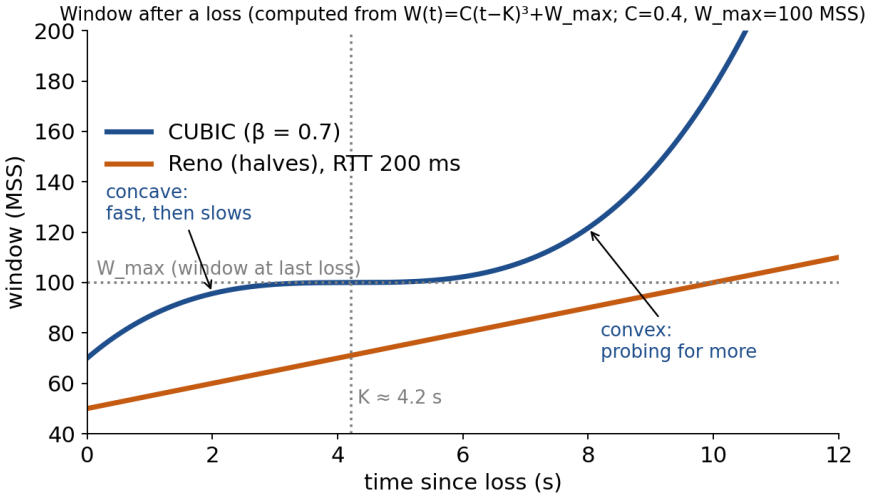
Rate Adaptation and TCP: Cooperating Layers



Each layer has a role in exposing capacity changes early.

CUBIC: New Congestion Control

- Window is a cubic function of time since the last loss
  - $W(t) = C \cdot (t - K)^3 + W_max$
  - On loss, cwnd drops to $\beta \cdot W_max$
    - $\beta = 0.7$ (Reno halves); $C = 0.4$ (RFC 9438)
- Growth depends on elapsed time, not RTT
  - Long-RTT flows are not penalized
  - Rcovery on large-BDP paths is fast
- Slow start exits with HyStart++ (RFC 9406)



CUBIC on Wireless: What Changes

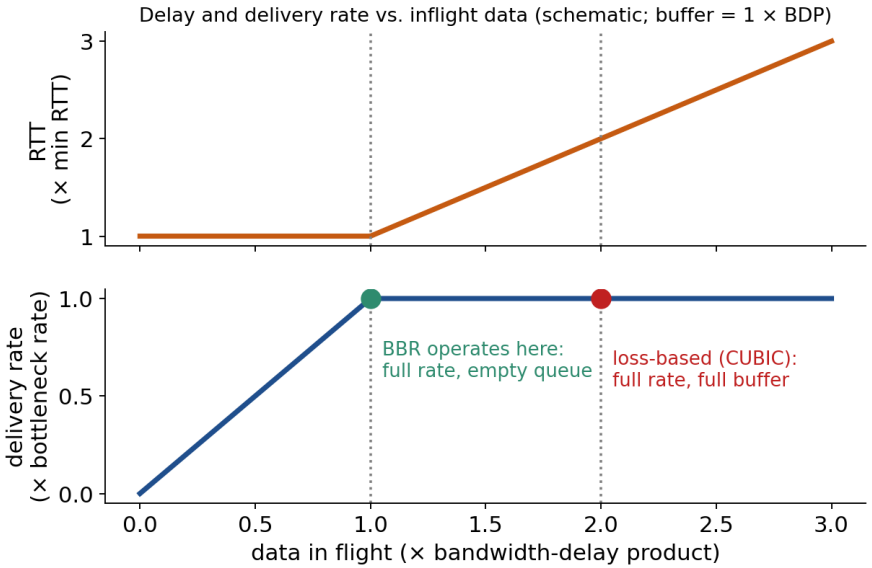
- Random loss still cuts window on every loss to 0.7 instead of 0.5
  - Recovers faster on large-BDP paths, but throughput is still loss-limited
- Fills the buffers
  - A standing queue in the cellular buffer causes bufferbloat
  - Bigger buffer gives CUBIC a bigger share against BBR
- HyStart can leave slow start too early when RTT varies,
  - Short connections underuse the link
  - HyStart++ adds Conservative Slow Start phase to test for early exit
- Capacity changes are hard
  - Real 5G tests found Reno, CUBIC and BBRv1 all slow to track carrier bandwidth
- Mobility
  - Similar to NewReno when radio quality falls, better when it improves (more efficient probing)





BBR: Model-Based Congestion Control

- Estimates bottleneck bandwidth (BtlBw) and minimum RTT (RTprop)
  - Paces at about BtlBw and keeps inflight near the BDP
  - Loss is not the main signal
- BBRv2 and v3 react to loss and ECN



BBR on Wireless: What Changes

- Helps with random loss
  - Not loss-triggered
  - On a lossy LTE high-speed-rail path BBR's goodput was about 70% higher than CUBIC's
- Keeps queues short when it runs alone
- Hurts when the model is stale
  - Link rate changes distort the estimates
  - CUBIC's queue can stop BBR measuring true min RTT
- Fairness and retransmissions
  - v1 can retransmit far more than CUBIC and be unfair to loss-based flows (IMC 2019)
  - v2/v3 respond to loss and ECN





TCP in Presence of Transmission Errors: Summary

- Many techniques have been proposed, and several approaches perform well in many environments
- Recommendation: Prefer end-to-end techniques
  - End-to-end techniques are those which do not require TCP-Specific help from lower layers
  - Lower layers may help improve TCP performance without taking TCP-specific actions.
    - Examples:
      - Semi-reliable link level retransmission schemes
      - Explicit notification





Why This Whole Problem Shrank: Link-Layer HARQ

- LTE and 5G bake aggressive link-layer retransmission directly into the radio protocol
- Wi-Fi does the same with MAC retries, block ACK, and rate adaptation; the cost shows up as delay variation (jitter), not loss

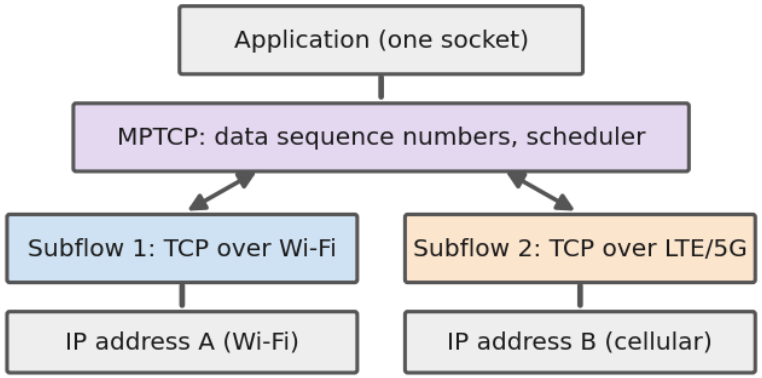
- Hybrid ARQ (HARQ)
  - Failed transmissions are retransmitted and combined with the original at the physical layer, before the loss ever reaches TCP
  - Typically resolves errors within a few milliseconds, well inside normal TCP RTT
- The practical effect
  - Uses pieces of Split Connection, Snoop, and the ELN
  - TCP now rarely sees any wireless loss at all
  - The remaining ambiguity
    - Was a loss really congestion, or one HARQ couldn't fix in time?





Multipath TCP

- Plain TCP is tied to one IP address pair
  - Switching between Wi-Fi and LTE breaks the connection
- MPTCP: one connection, several subflows over different interfaces
  - Each subflow looks like ordinary TCP to the network
- Wireless motivation: phones have Wi-Fi and cellular
  - Handover without breaking sessions, bandwidth aggregation, resilience to fade



SYN with MP_CAPABLE on first path; MP_JOIN adds more paths later

MPTCP: Congestion Control and Scheduling

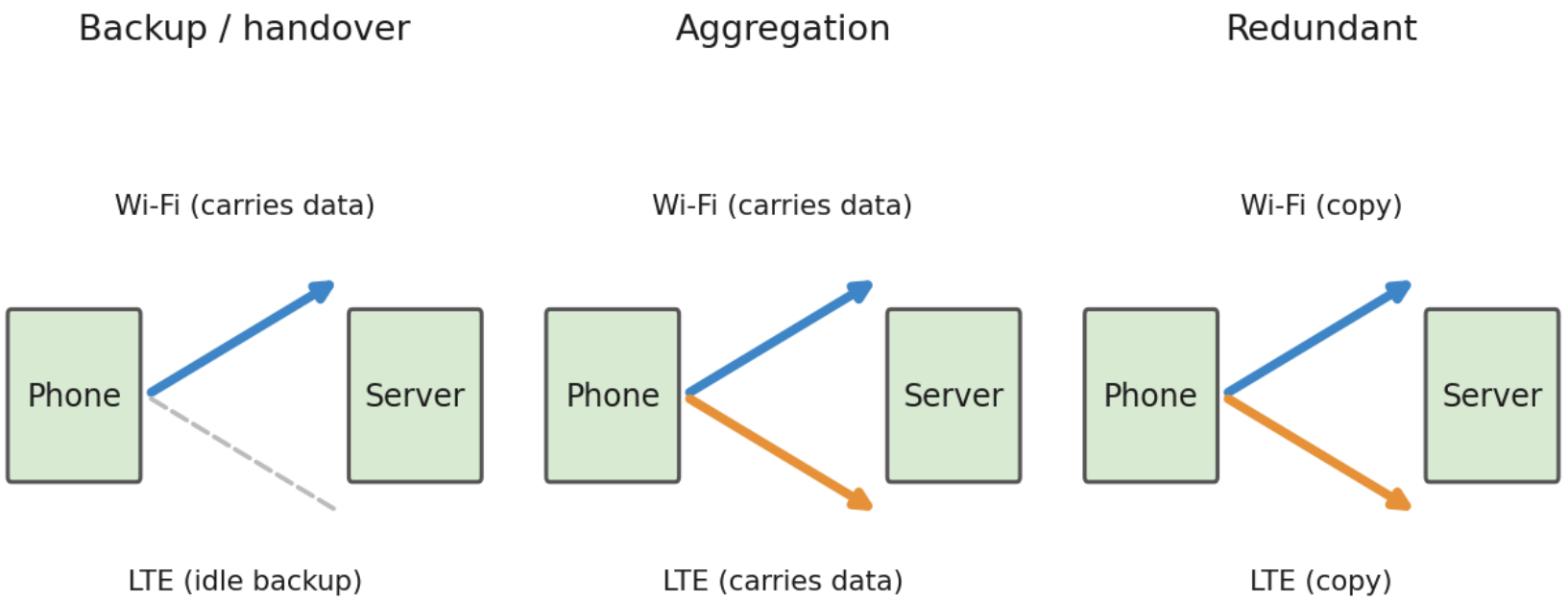
- Each subflow runs its own congestion window
  - Coupled congestion control (RFC 6356)
  - Keeps MPTCP from taking more than one TCP flow's share at a shared bottleneck
- The scheduler decides which subflow carries each segment
  - e.g., Lowest-RTT subflow with window space
  - e.g., Send on all for latency
- Heterogeneous paths cause trouble
  - e.g., Wi-Fi 20 ms, LTE 80 ms
  - Segments arrive out of order
  - Receive buffer fills
  - One slow path can stall the stream
  - Research schedulers try to avoid this by predicting arrival order





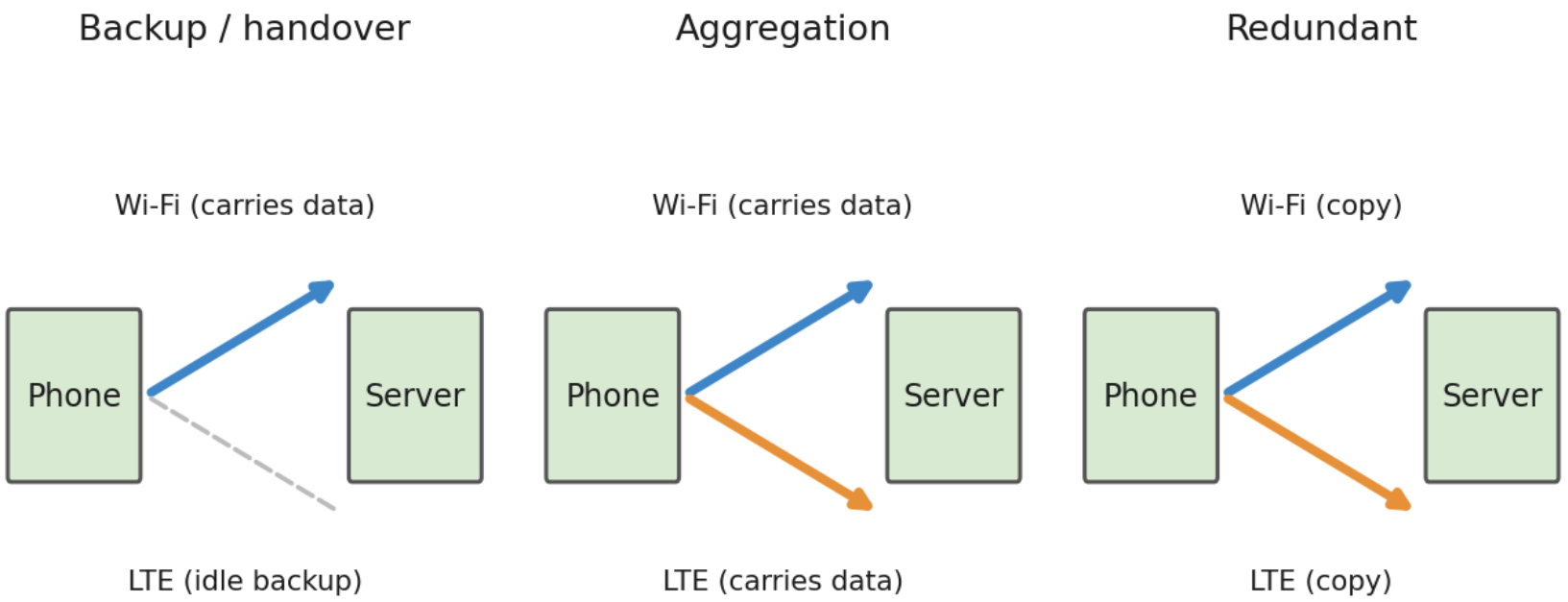
MPTCP Policies: Handover vs. Aggregation

- Backup (handover) mode
  - Wi-Fi is primary
  - Cellular is used only when Wi-Fi fades or drops
  - Saves cellular data and radio energy



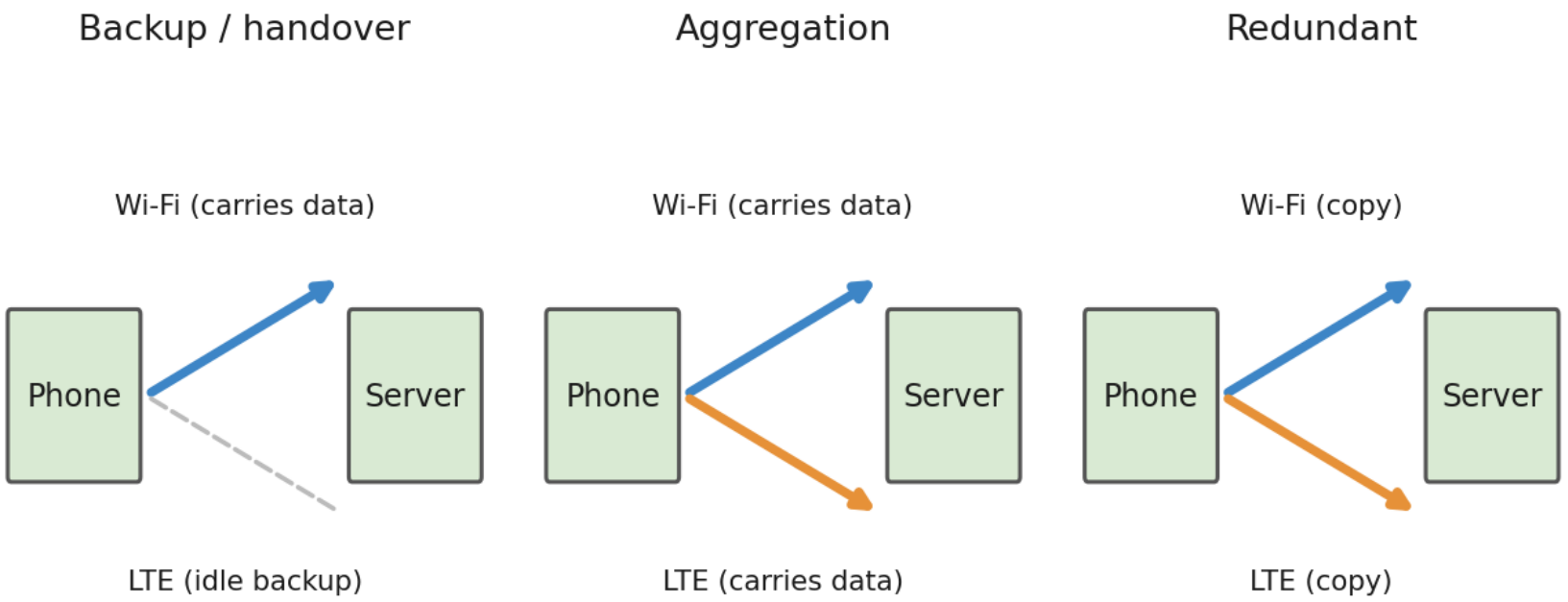
MPTCP Policies: Handover vs. Aggregation

- Aggregation mode
  - Both paths carry data
  - Higher throughput, higher cost and energy



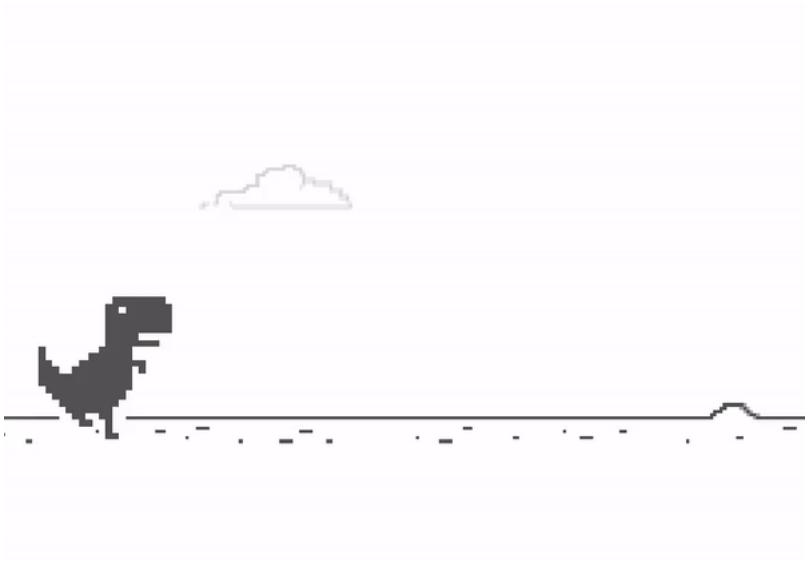
MPTCP Policies: Handover vs. Aggregation

- Redundant mode
  - Copy data on both paths for latency-critical traffic



MPTCP Policies: Handover vs. Aggregation

- A path manager creates and removes subflows as interfaces come and go
  - Typical trap: Wi-Fi with strong signal but no Internet



QUIC: a New Transport

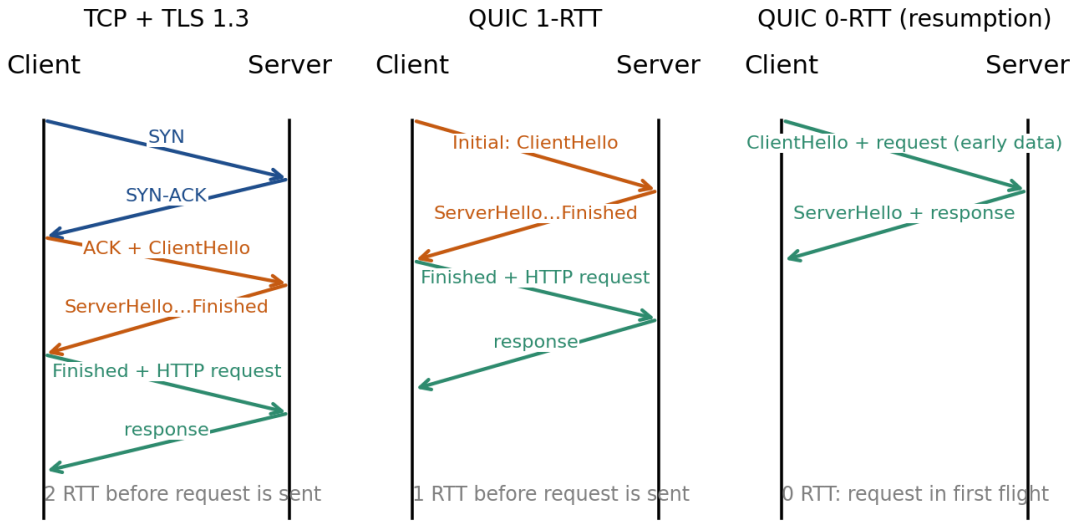
- QUIC (RFC 9000) runs over UDP, in user space, with TLS 1.3 built in (RFC 9001)
- Relevant problems with TCP it addresses
  - Separate transport and TLS handshakes cost extra round trips
  - Head-of-line blocking
    - One lost packet stalls every request on the connection
  - Connection is tied to the IP address and port
    - Breaks when the device moves
- Tries to get rid of
  - Long and variable RTTs
  - Frequent loss
  - Frequent handoffs





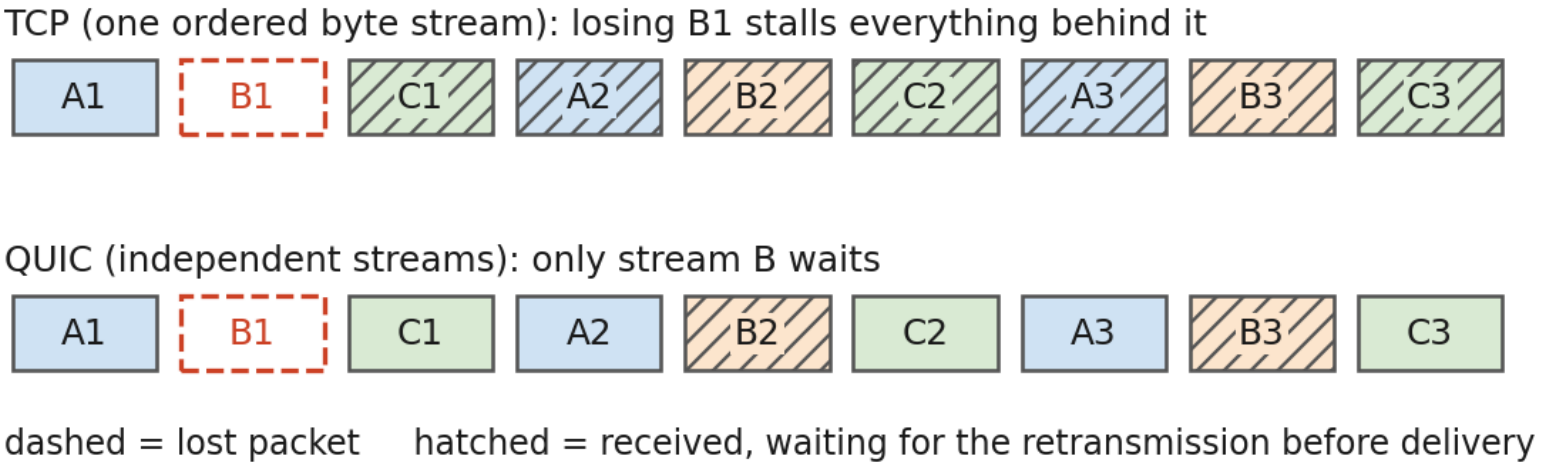
QUIC Handshake

- TCP + TLS 1.3
  - Two round trips before the first request (TCP handshake, then TLS)
- QUIC
  - Transport and crypto together, so one round trip
- On a high-RTT or lossy wireless link, saved round trips matter more



QUIC Streams and Head-of-Line Blocking

- TCP delivers one ordered byte stream
  - A lost packet delays everything behind it, even data for other requests
- QUIC multiplexes independent streams



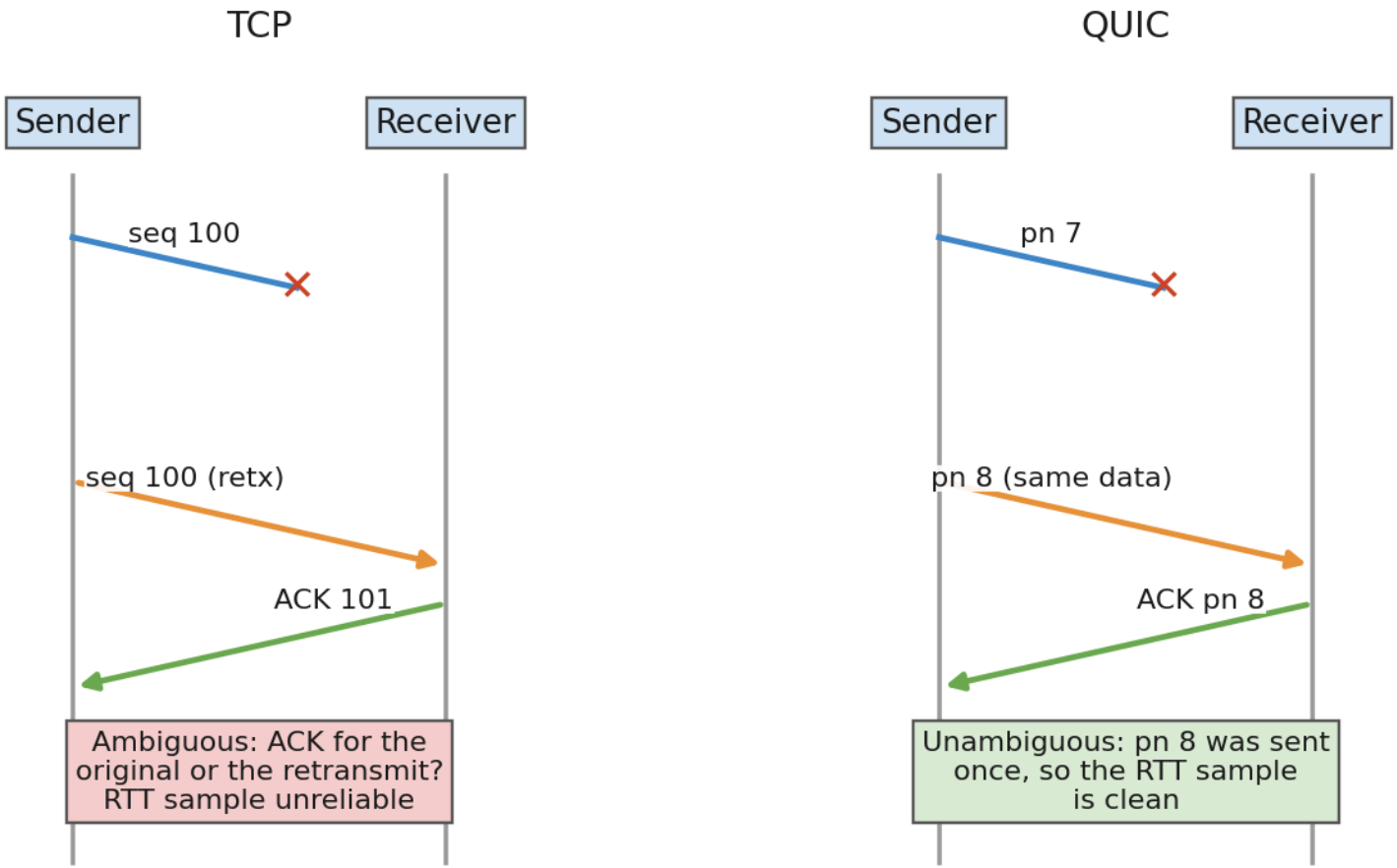
QUIC Packet Numbers

- Every packet gets a new, increasing packet number, including retransmissions
  - No retransmission ambiguity
  - Each ACK maps to exactly one transmission
  - RTT samples are clean
- ACKs carry ranges and the receiver's ACK delay
  - The sender subtracts the delay for more accurate RTT estimates





QUIC Packet Numbers vs. TCP Sequence Numbers



QUIC Loss Recovery and Congestion Control

- Loss detection (RFC 9002)
  - Packet threshold
    - Packet is lost when one sent 3 or more packets later is acknowledged
  - Time threshold, and a probe timeout (PTO) instead of an RTO
  - PTO sends probes and does not by itself mean loss
- Congestion control runs in user space
  - Default NewReno in the RFC; CUBIC and BBR in practice; pacing and ECN supported





QUIC Congestion Control on Wireless

- QUIC runs the same algorithms as TCP (NewReno, CUBIC, BBR), so the same wireless issues apply
- Where QUIC helps
  - Unique packet numbers and ACK delay give clean RTT samples
  - A probe timeout (PTO) sends probes without implying loss, so a delay spike is less damaging than a TCP timeout
- Where it varies
  - Implementations differ by library, and the choice of congestion control noticeably changes performance on real cellular networks
  - BBR is common in QUIC
  - Migration resets path state, so the new path is probed from scratch





QUIC Connection Migration

- A QUIC connection is identified by connection IDs, not the IP address and port
- After a Wi-Fi to cellular switch, the client keeps the connection
  - It validates the new path first, which prevents address spoofing
  - The congestion state of the new path is reset
- Also survives NAT rebinding
- Compare
  - TCP reconnects; MPTCP keeps both paths alive at once (previous section)
- Using several paths at once (multipath QUIC) is still being standardized





QUIC: Trade-offs and Reality

- Encrypted headers
  - Middleboxes cannot see or split QUIC (Snoop, PEPs)
  - nd hosts must do the work
- Costs and fallbacks
  - UDP is sometimes blocked, so clients fall back to TCP
  - User-space processing costs CPU
- Adoption
  - Roughly 35-39% of websites use HTTP/3
- Measured benefit on wireless is mixed
  - Similar to TCP, with QUIC coping better with channel variation



