Transport Layer: MPTCP

CS 204: Advanced Computer Networks

Oct 16, 2023

Adapted from Jiasi's CS 204 slides for Spring 23

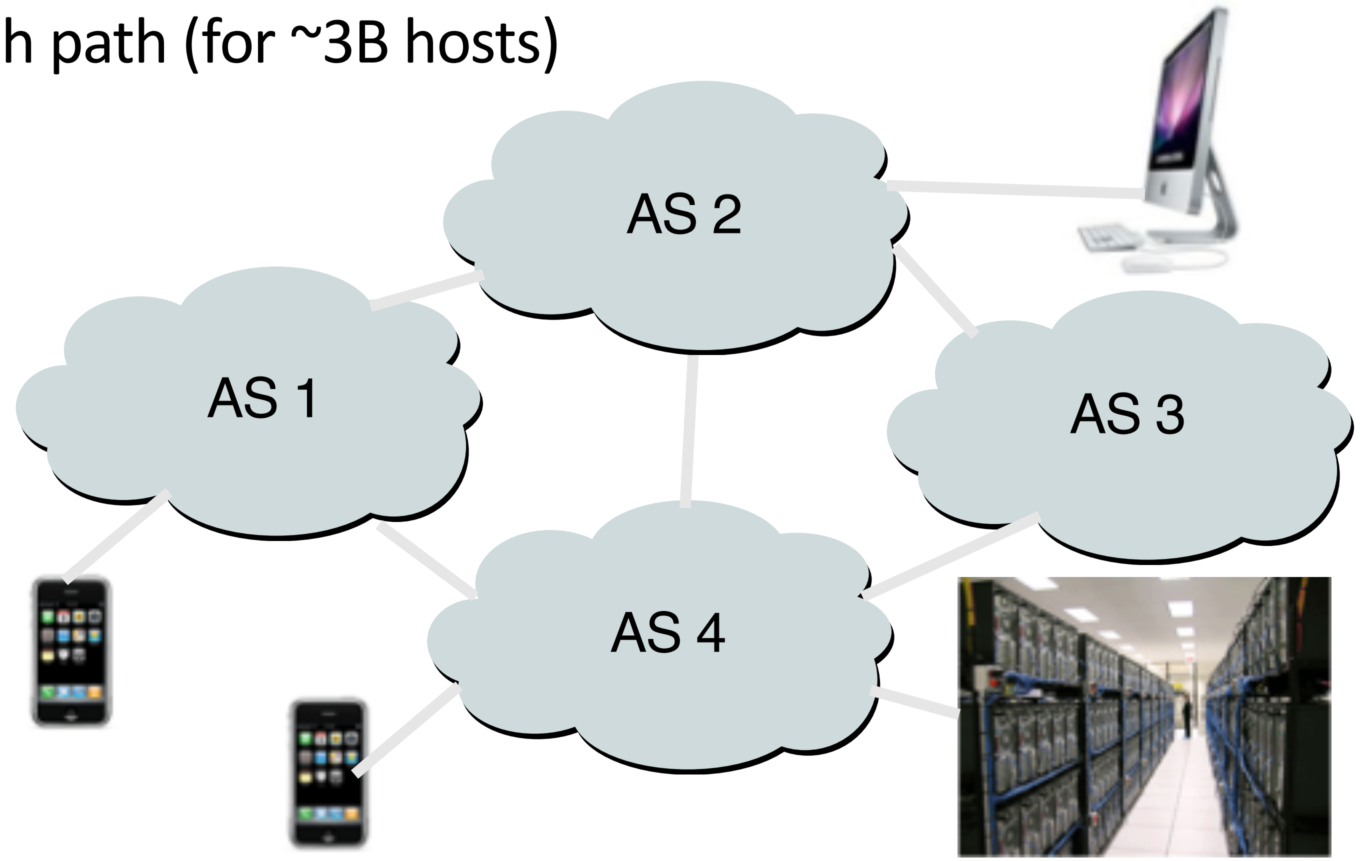
Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
  - CUBIC
  - BBR

Q: How should flows compete for bandwidth when there is congestion in the network?

Holding the Internet Together

- Distributed cooperation for resource allocation
  - BGP: what end-to-end *paths* to take (for ~60K ASes)
  - TCP: what *rate* to send over each path (for ~3B hosts)



Approaches towards congestion control

two broad approaches towards congestion control:

end-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at

TCP seq. numbers, ACKs

sequence numbers:

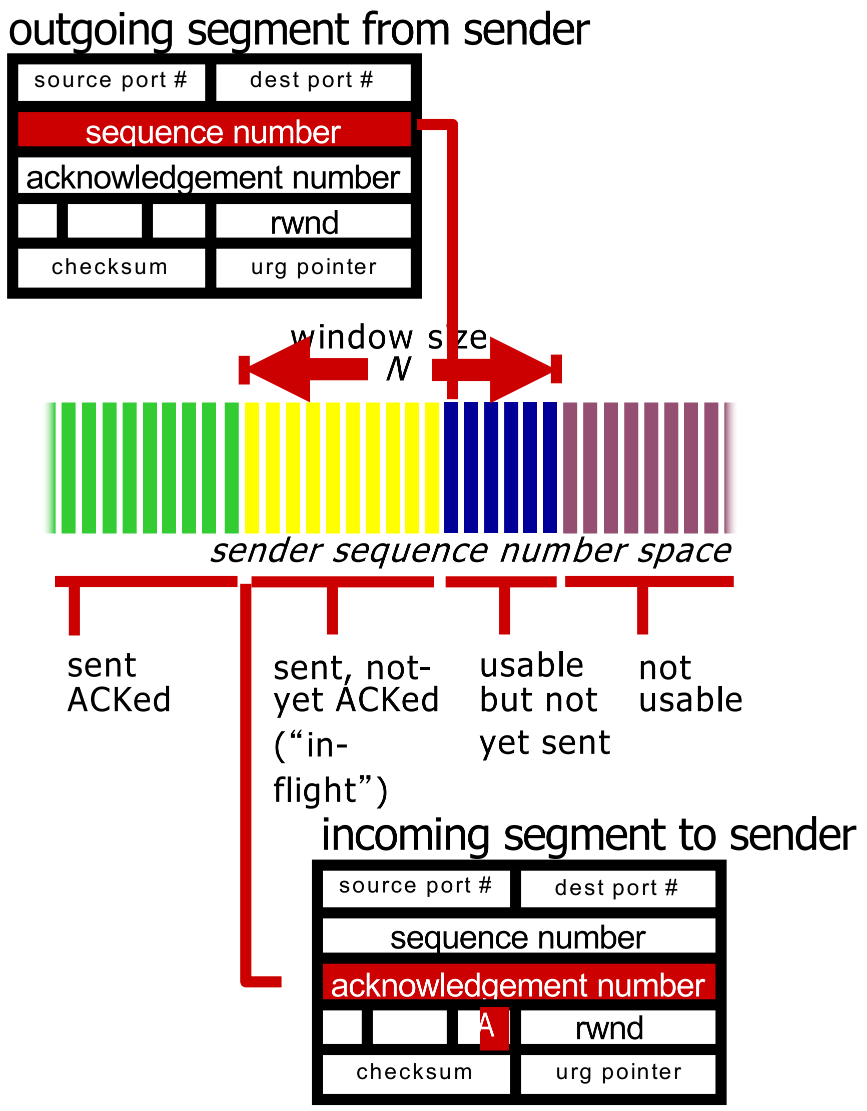
- byte stream "number" of first byte in segment's data

acknowledgements:

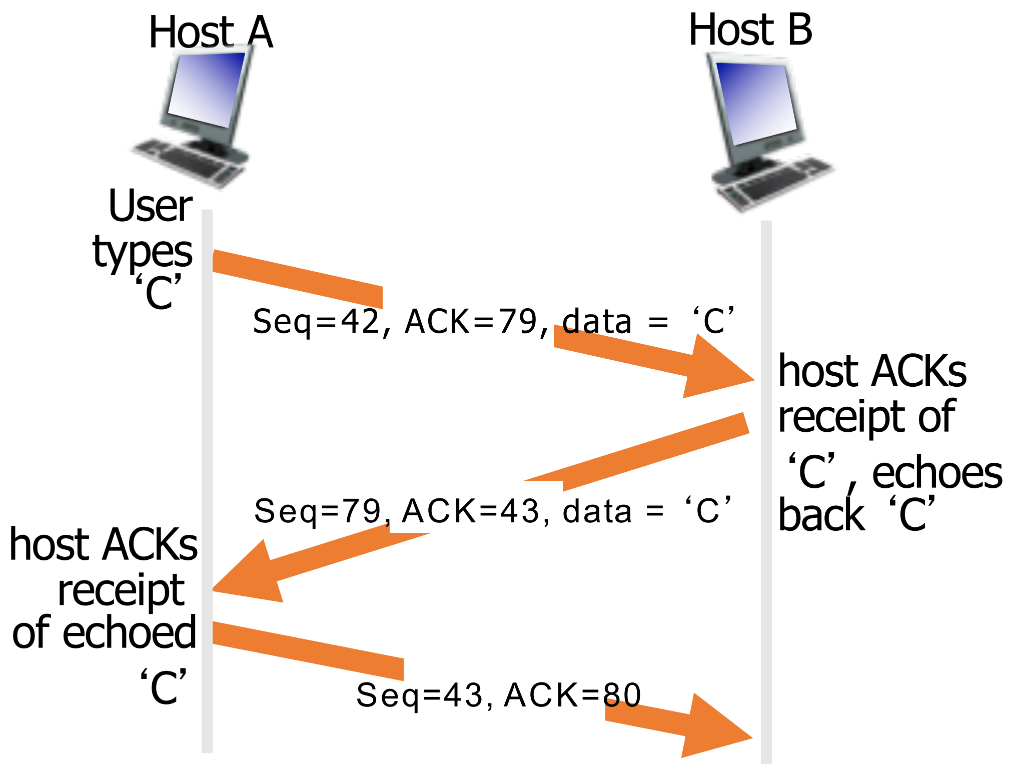
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor

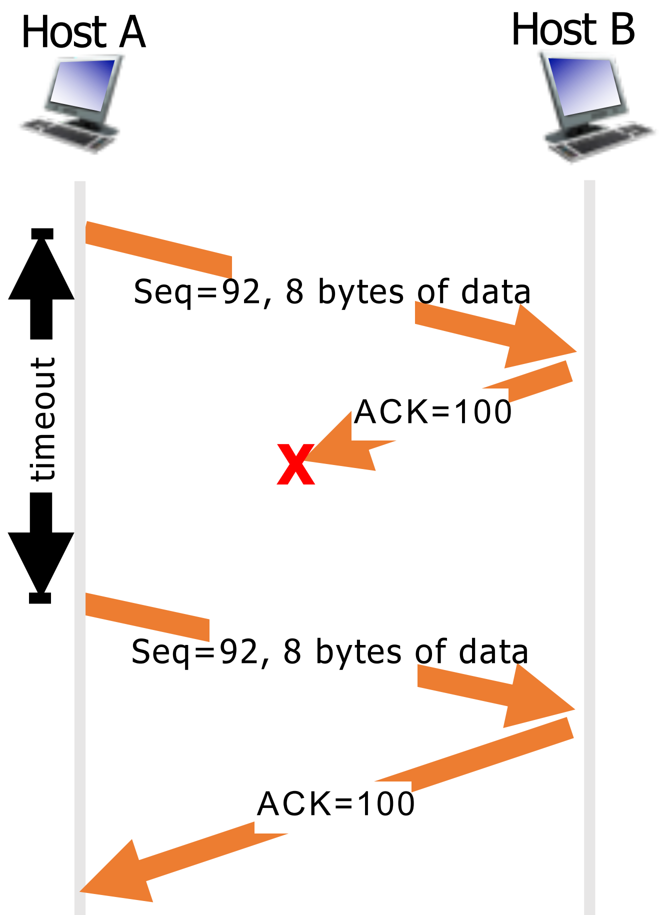


TCP seq. numbers, ACKs

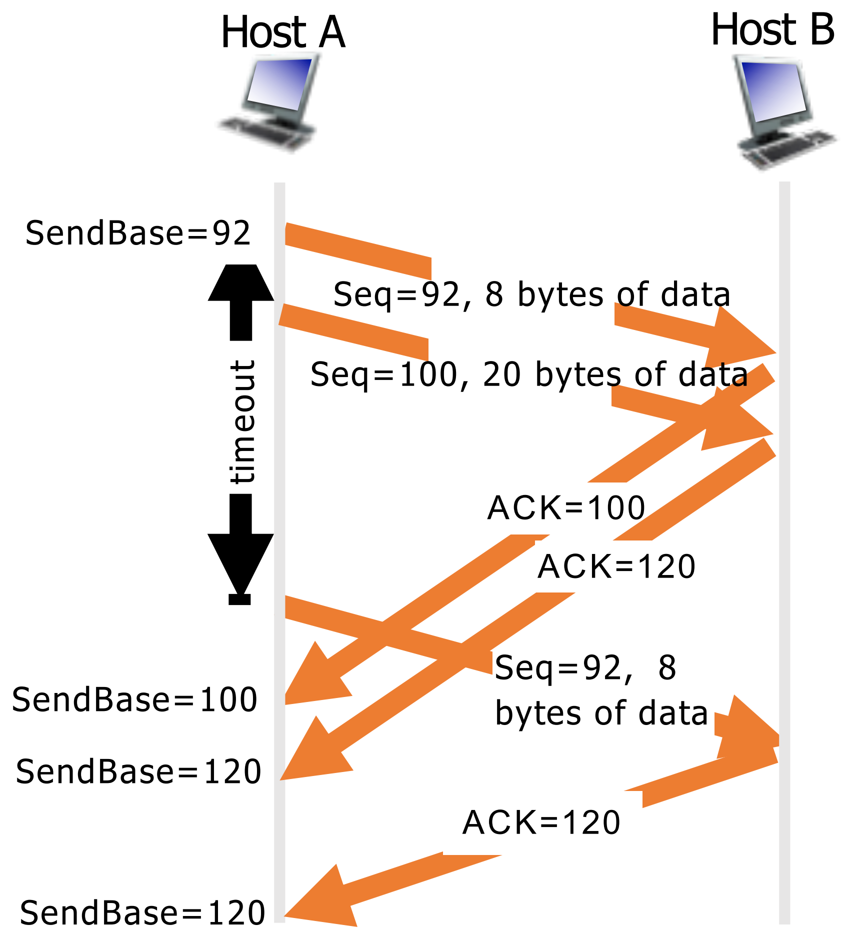

Host A
Host B
User types 'C'
Seq=42, ACK=79, data = 'C'
host ACKs receipt of 'C', echoes back 'C'
Seq=79, ACK=43, data = 'C'
host ACKs receipt of echoed 'C'
Seq=43, ACK=80

simple telnet scenario

TCP: retransmission scenarios



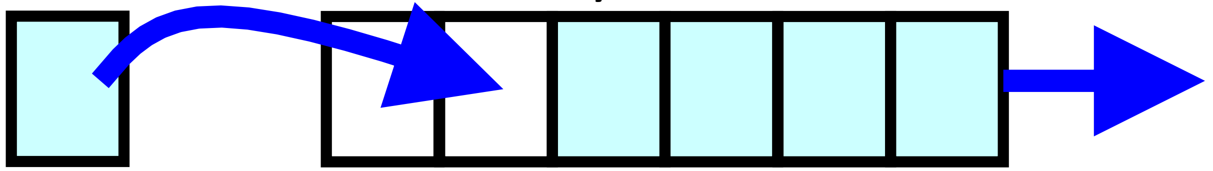
lost ACK scenario



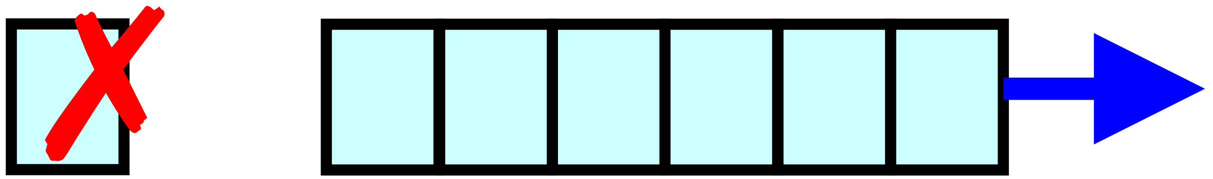
premature timeout

Congestion in Drop-Tail FIFO Queue

- Access to the bandwidth: first-in first-out queue
  - Packets transmitted in the order they arrive



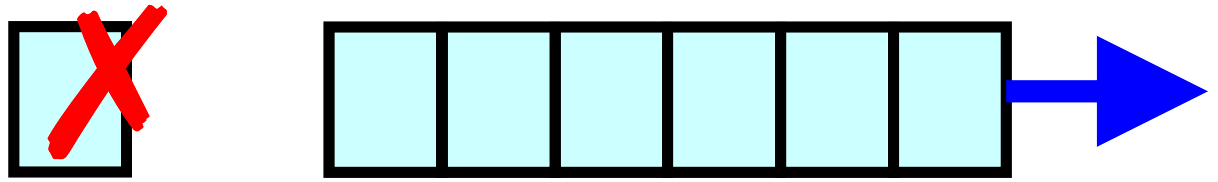
- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



How it Looks to the End Host

- Delay: Packet experiences high delay
- Loss: Packet gets dropped along path

- How can TCP sender learn this?
  - Delay: Round-trip time estimate
  - Loss: Timeout and/or duplicate acknowledgments
  - Mark: Packets marked by routers with large queues



TCP Congestion Window

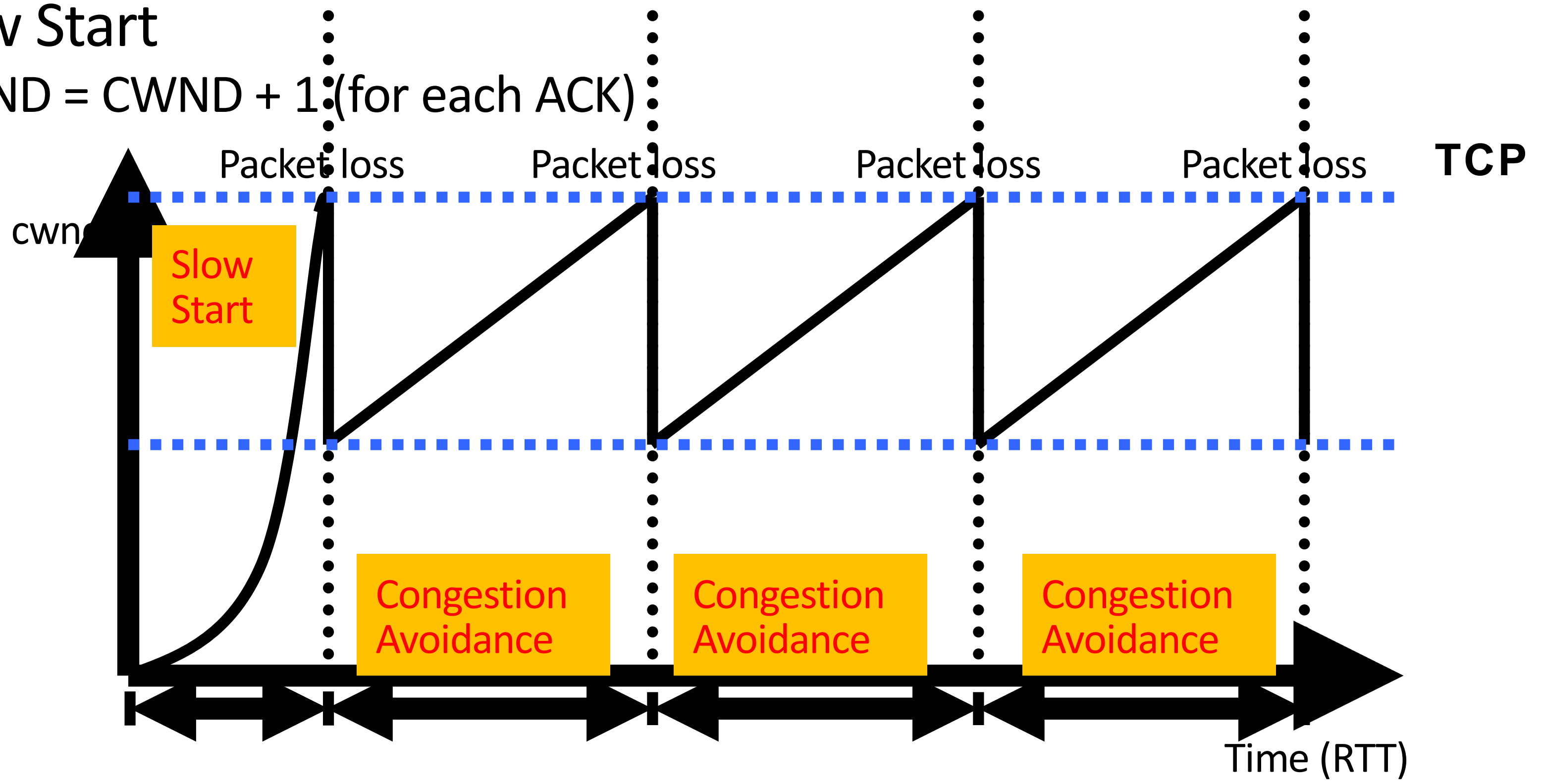
- Each sender maintains congestion window
  - Max number of bytes to have in transit (not ACK’d)

- Adapting the congestion window
  - Decrease upon losing a packet: backing off
  - Increase upon success: optimistically exploring
  - Always struggling to find right transfer rate

- Tradeoff
  - Pro: avoids needing explicit network feedback
  - Con: continually under- and over-shoots “right” rate

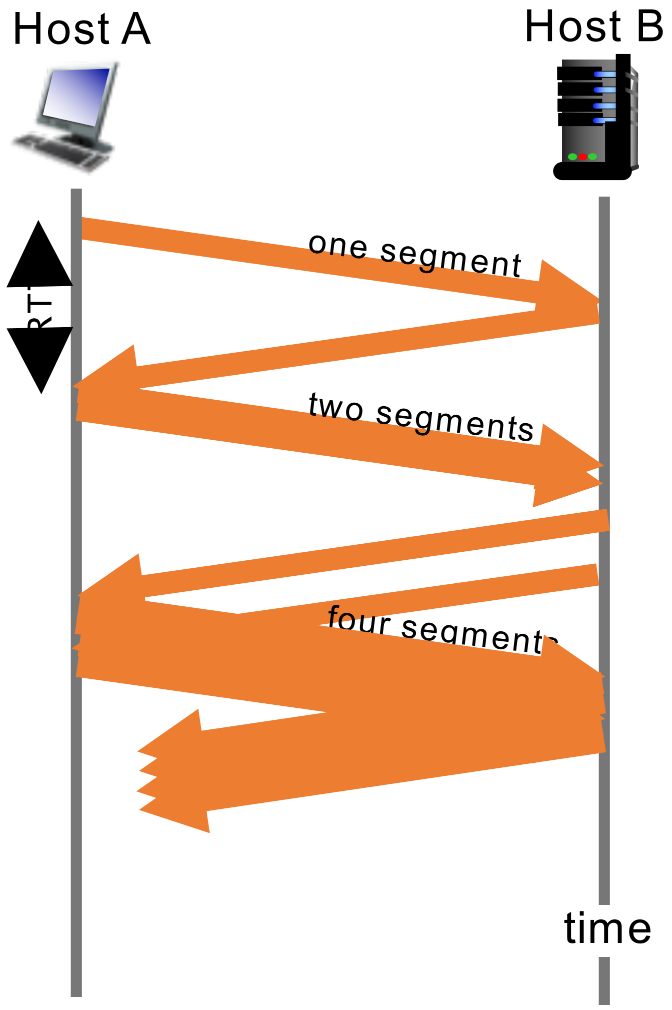
TCP Congestion Control

- Two parts in TCP congestion control
  - (1) Congestion Avoidance
    - CWND = CWND + 1/CWND (for each ACK)
  - (2) Slow Start
    - CWND = CWND + 1 (for each ACK)



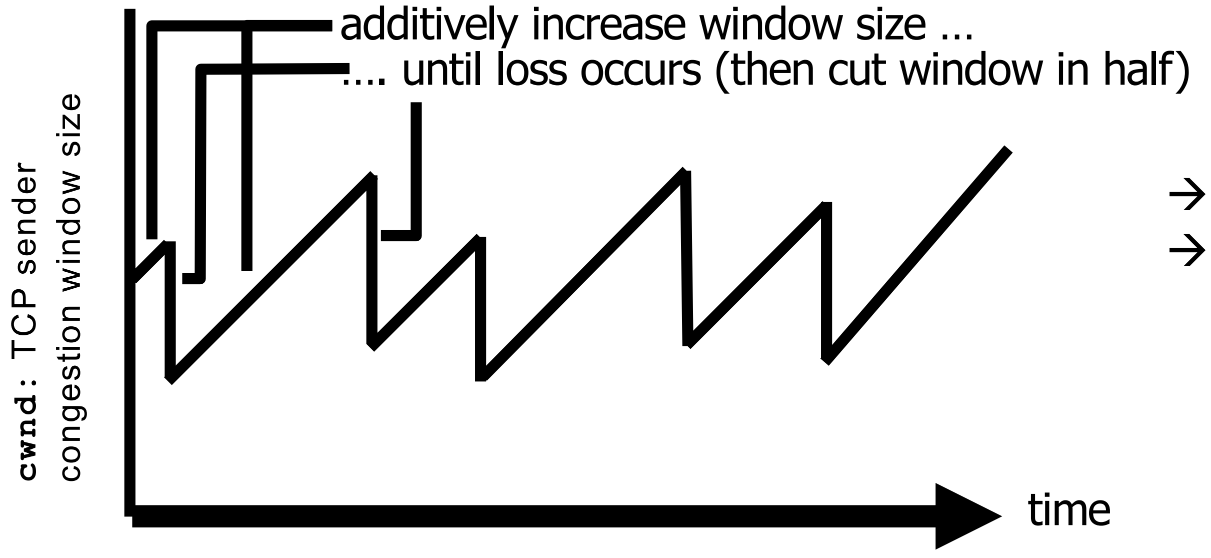
TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



TCP Congestion Avoidance

- sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - additive increase: increase cwnd by 1 MSS every RTT until loss detected
  - multiplicative decrease: cut cwnd in half after loss



→ AIMD saw tooth behavior: probing for bandwidth

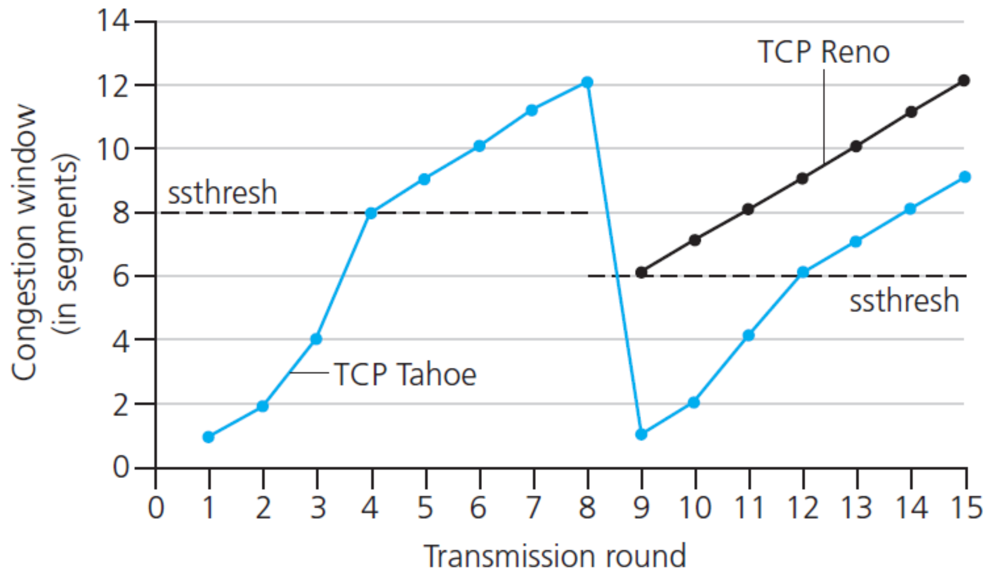
→ Much quicker to slow than speed up!

- Over-sized windows (causing loss) are much worse than under-sized windows (causing lower throughput)
- AIMD: A necessary condition for stability of TCP

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

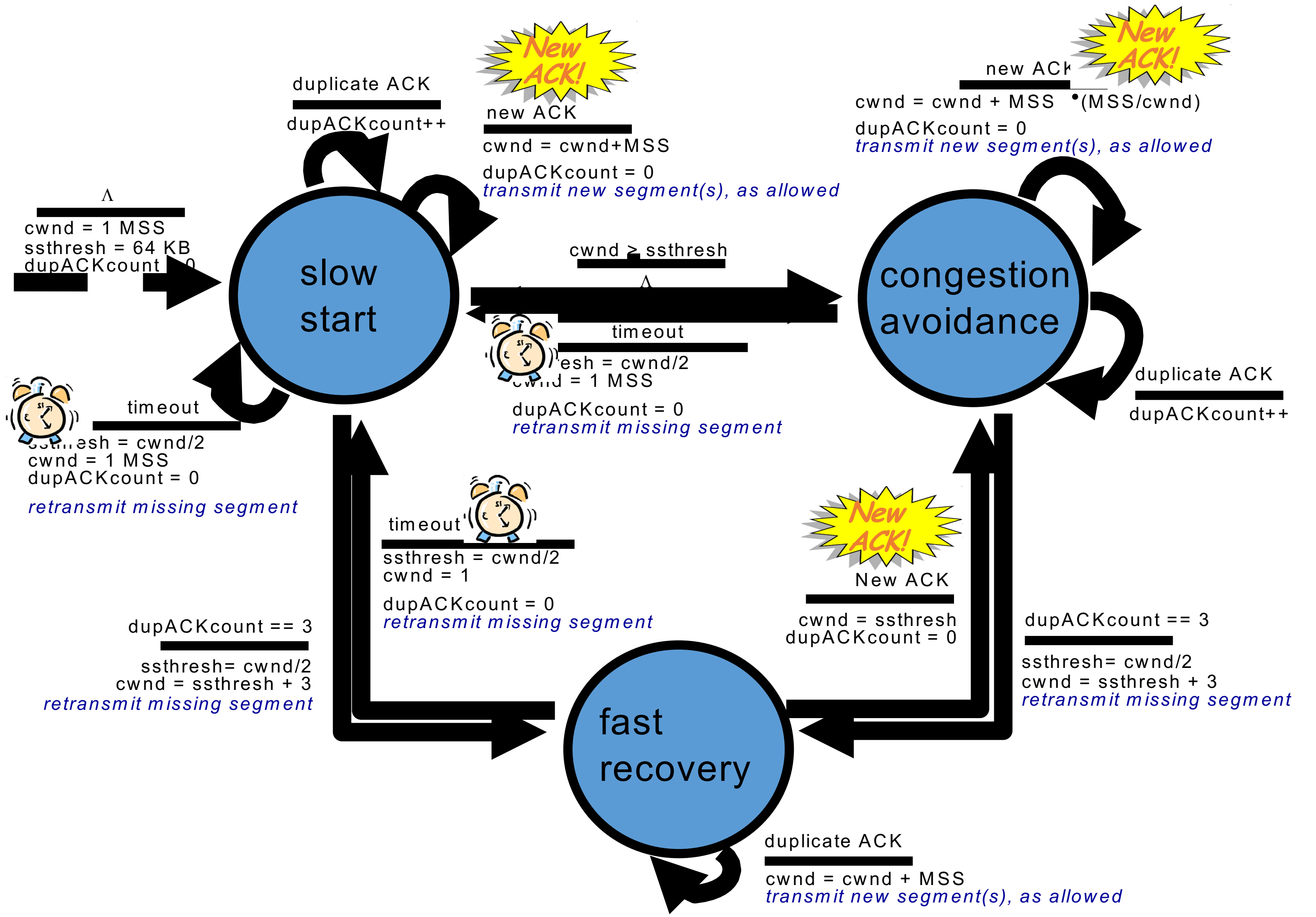
A: when **cwnd** gets to 1/2 of its value before timeout.



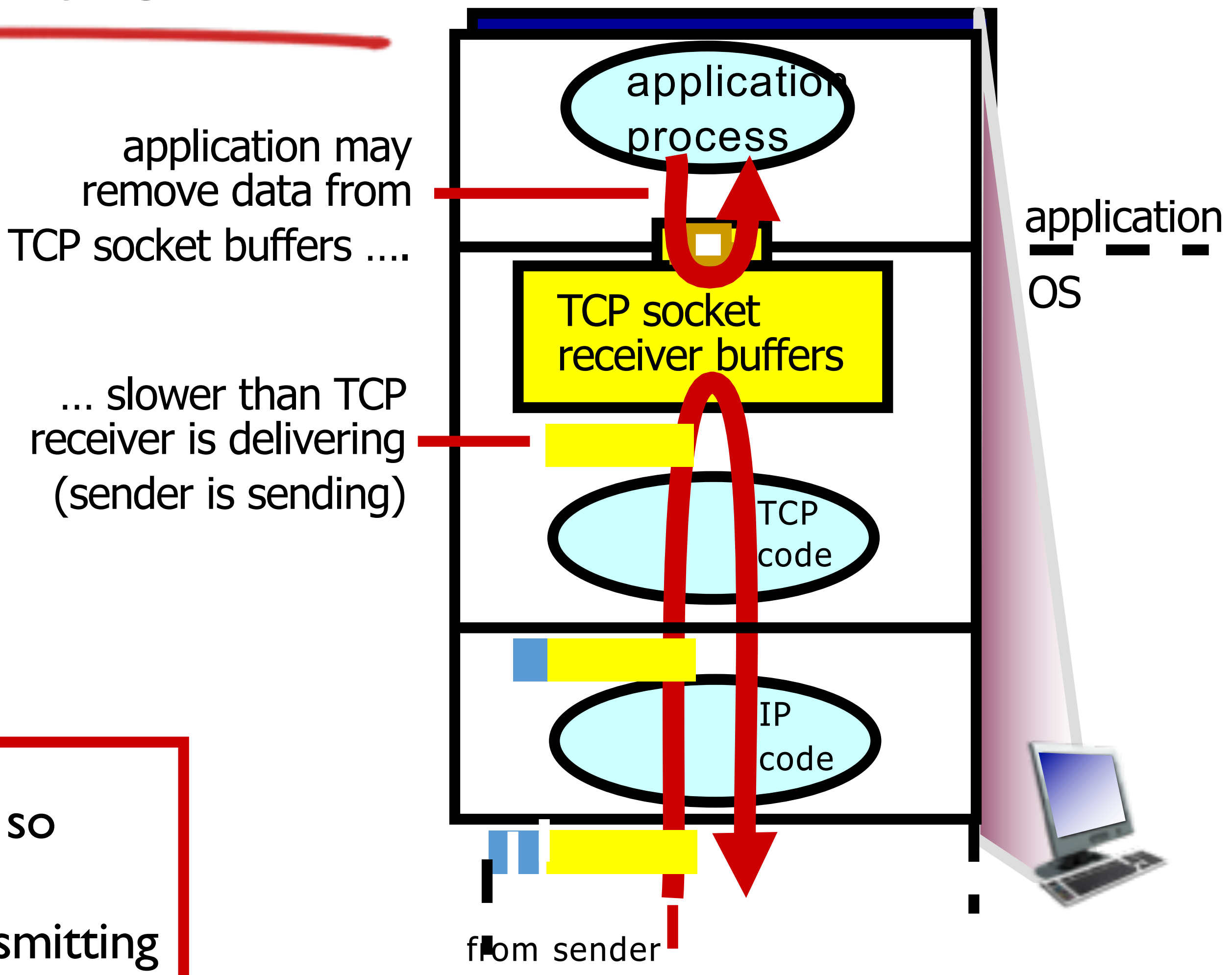
Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

Summary: TCP Congestion Control



TCP flow control


application may remove data from TCP socket buffers ….
… slower than TCP receiver is delivering (sender is sending)
application process
TCP socket receiver buffers
TCP code
IP code
application
OS
from sender

receiver protocol stack

flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

Receiver Window vs. Congestion Window

- Flow control
  - Keep a *fast sender* from overwhelming *slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - Sender TCP window =

    min { congestion window, receiver window }

TCP Throughput

- Throughput ≤ min(cwnd, rwnd) / RTT

- Desired properties of a TCP congestion control algorithm
  - TCP-friendliness
    - Only use as much rate as a regular TCP flow would
  - RTT-fairness
    - Congestion windows only increase with each RTT
    - What if one flow with big RTT, one flow with small RTT?

Outline

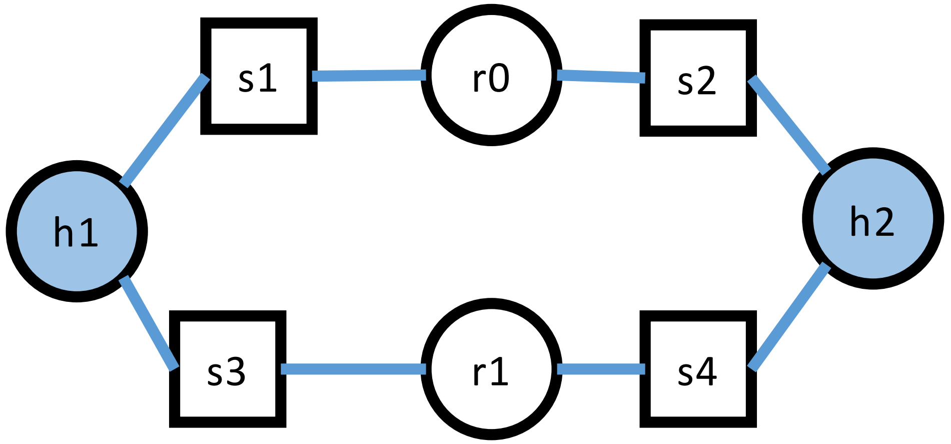
- TCP Review
- New TCP flavors
  - Multipath-TCP
    - Basics
    - Sequence numbers
    - Congestion control
  - CUBIC
  - BBR

Q: How should flows compete for bandwidth when there is congestion in the network?

Why have multiple paths?

- Mobile user
  - WiFi and cellular at the same time
- High-end servers
  - Multiple Ethernet cards
- Data centers
  - Rich topologies with many paths

- Benefits of multipath
  - Higher throughput
  - Failover from one path to another
  - Seamless mobility

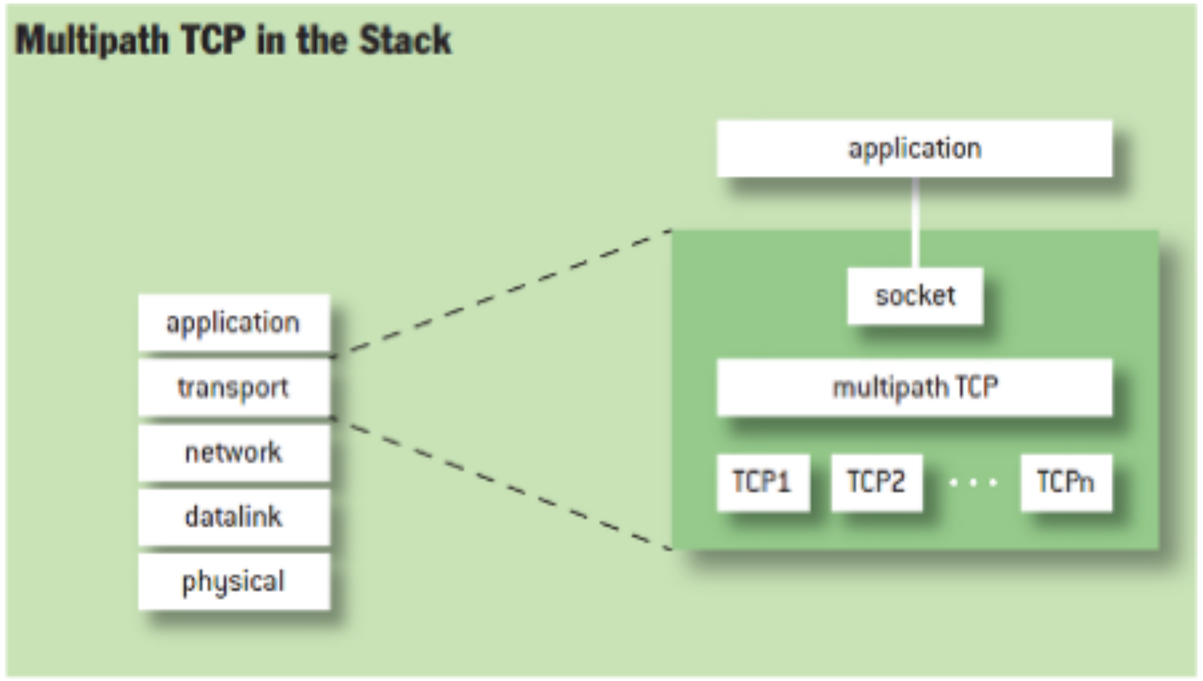


Key Design Goals

- Use the available network paths at least as well as regular TCP, but without starving TCP
- Usable as regular TCP for existing applications
- Enabling MPTCP must not prevent connectivity on a path where regular TCP works

Working With Unmodified Apps

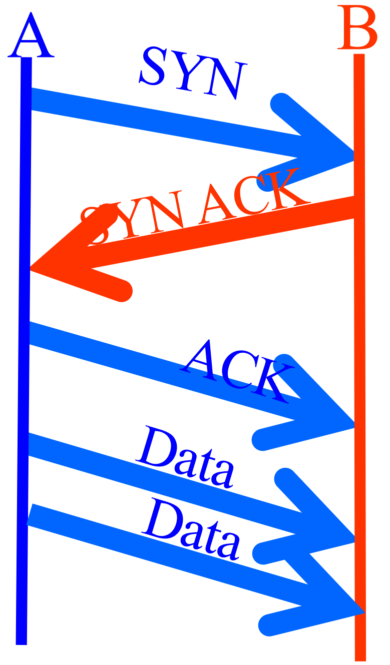
- Present the same socket API and expectations
  - Identified by the “five tuple” (IP address, port #, protocol)



From http://queue.acm.org/detail.cfm?id=2591369

Working With Unmodified Hosts

- Establish the TCP connection in the normal way
  - Create a socket to a single remote IP address/port

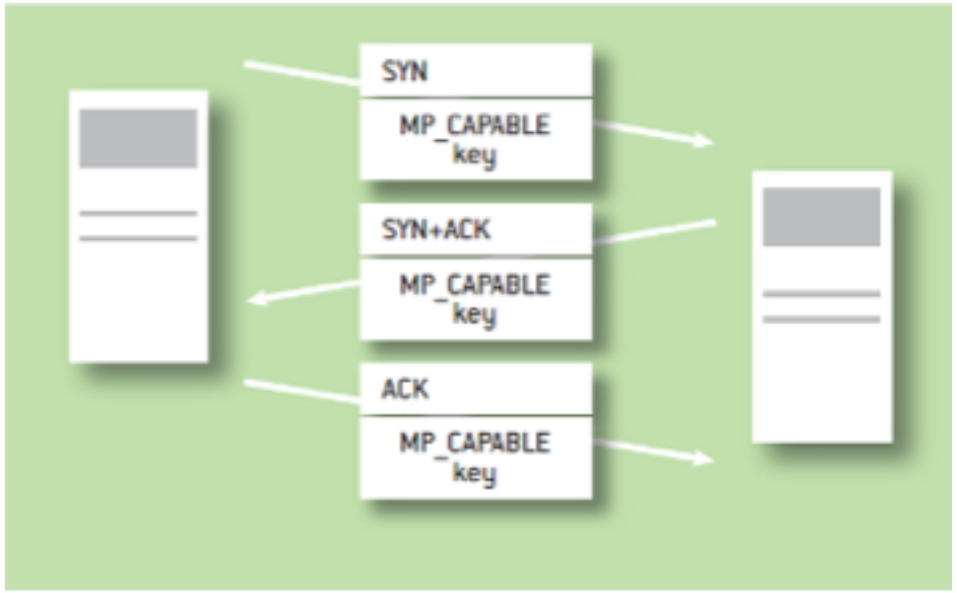


Each host tells its *Initial Sequence Number (ISN)* to the other host.

- And then add more subflows, if possible

Negotiating MPTCP Capability

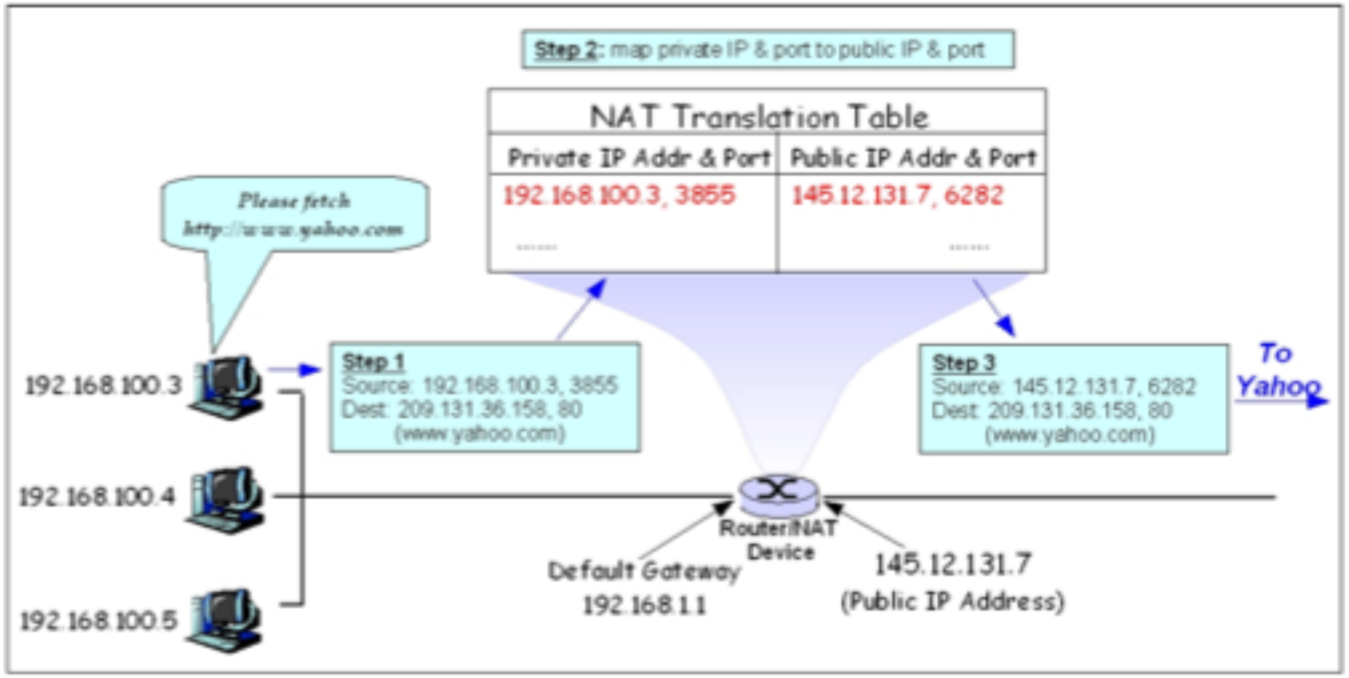
- How do hosts know they both speak MPTCP?
  - During the 3-way SYN/SYN-ACK/ACK handshake



- If SYN-ACK doesn’t contain MP_CAPABLE
  - Don’t try to add any subflows!

Detour: Middleboxes

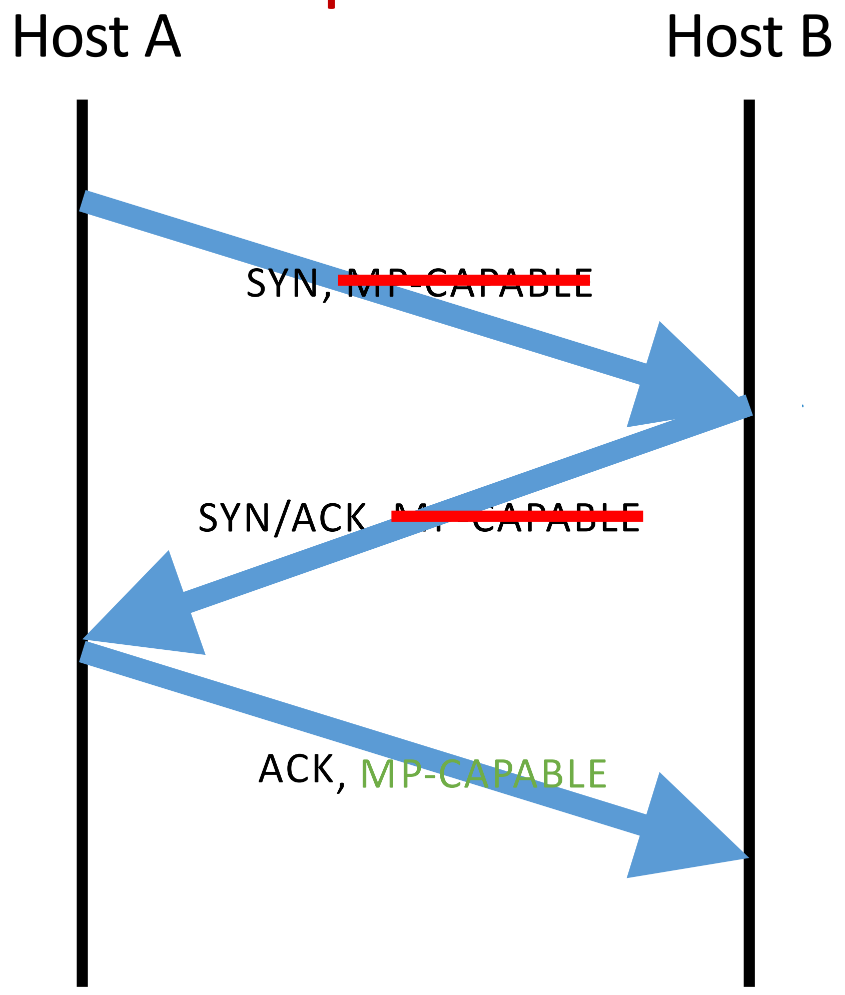
- In-network services, e.g.,
  - Firewall
  - Network address translator
  - Transparent proxy
  - Intrusion detection system
- Interaction with TCP
  - Change IP addresses and port numbers
  - Change TCP initial sequence number
  - Remove TCP options
  - Dividing large block of data into smaller packets
  - Expect to see all packets of the connection
  - Etc.


Step 2: map private IP & port to public IP & port
NAT Translation Table
Private IP Addr & Port
Public IP Addr & Port
192.168.100.3, 3855
145.12.131.7, 6282
Please fetch http://www.yahoo.com
Step 1 Source: 192.168.100.3, 3855 Dest: 209.131.36.158, 80 (www.yahoo.com)
Step 3 Source: 145.12.131.7, 6282 Dest: 209.131.36.158, 80 (www.yahoo.com)
To Yahoo
192.168.100.3
192.168.100.4
192.168.100.5
Router/NAT Device
Default Gateway 192.168.1.1
145.12.131.7 (Public IP Address)

Source: https://en.wikibooks.org/wiki/Communication_Networks/NAT_and_PAT_Protocols

Master Connection Setup: Example

- Use MP-CAPABLE flag to indicate sender has MPTCP capability
- Problem: Middleboxes remove TCP options
- Option removed on msg 1?
- Option removed on msg 2?
- If all goes correctly, add MP-CAPABLE



Master Connection Setup: Summary

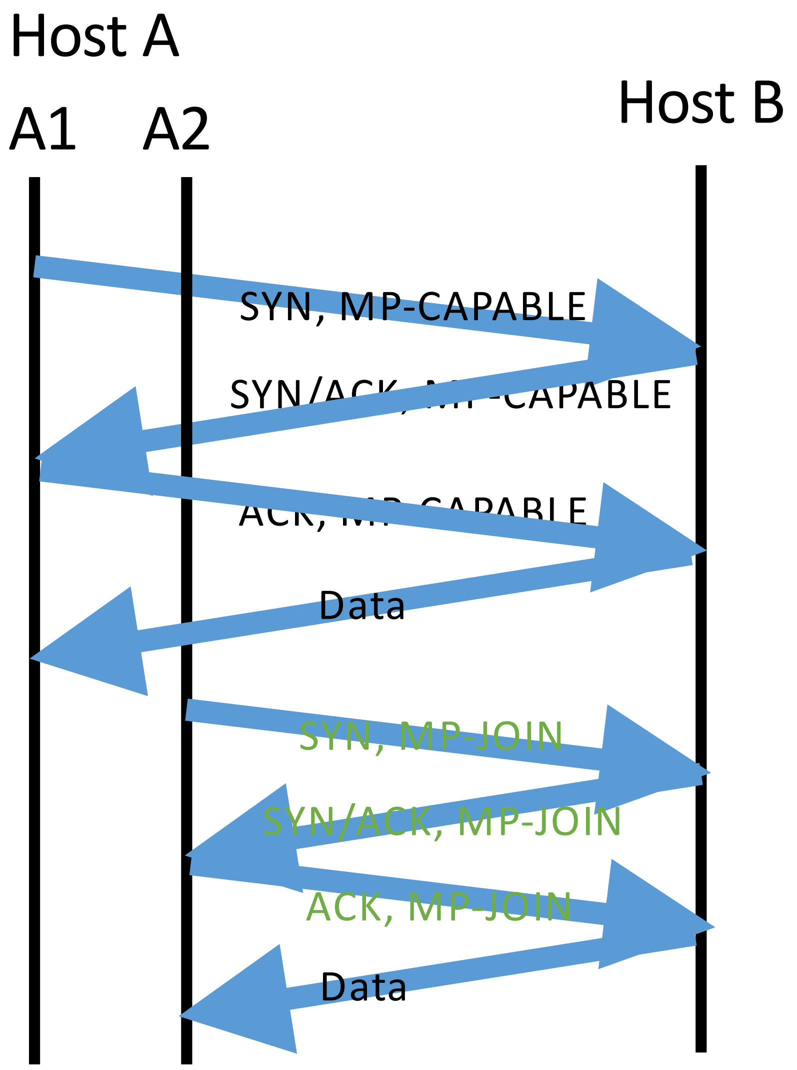
- What if middleboxes strip the MPTCP_CAPABLE option?
  - On the SYN? On the SYN-ACK?
- Include MPTCP_CAPABLE on the ACK of the SYN-ACK?
  - What if the ACK is lost?
  - Carry on all subsequent packets
- What if the middlebox *drops* SYN packets with unfamiliar options?
  - Sender can retransmit lost SYN without the option
  - … and fall back to regular TCP behavior

Adding New Subflows

- How to establish new subflows?

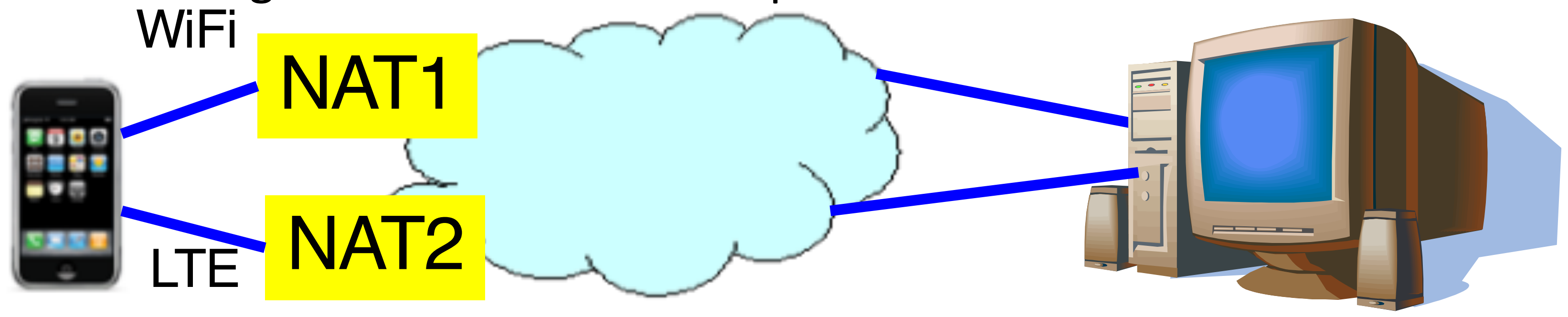
- Host A has addresses A1 and A2
- Assume Host B knows these addresses and starts sending data to both

- Problem: Middleboxes will not allow data to be sent without SYN
  → need 3-way handshake for new subflows



Adding New Subflows: Challenges

- Network Address Translators (NAT)
  - Problem: NAT changes the IP address and port number

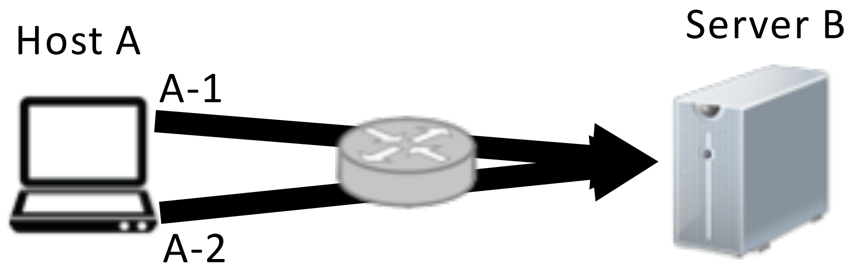


1. How to establish new subflows?
   - Allow one end-point to tell another about its addresses
2. How to identify which connection the subflow belongs to?
   - Using a token established during connection set-up

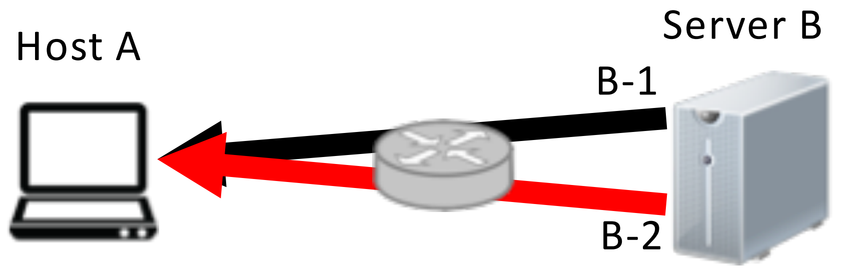
1. Adding New Subflows: Addresses

How to establish new subflows?

- Implicit address
  - Host A directly sends from A-2
  - Server B implicitly realizes that host A has 2 addresses
    - Using tokens (next slide)



- Explicit address
  - Server B tries to initiate a connection to Host A from B-2
    - Problem: B-2 can't reach client because of host A's NAT
  - Solution
    - Server sends ADD_ADDR option with B-2's address
    - Then Host A initiates connection to B-2

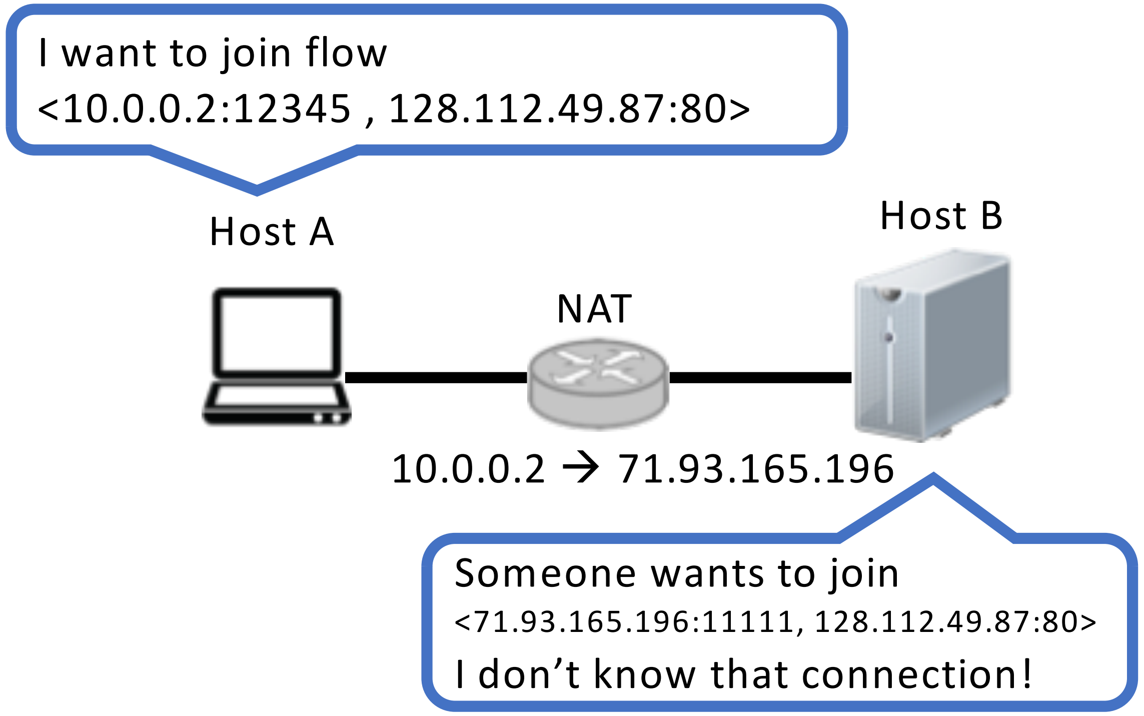


2. Adding New Subflows: Identification

How to identify which connection a subflow belongs to?

- TCP flows traditionally identified by
  <source IP:source port, dest IP:dest port>

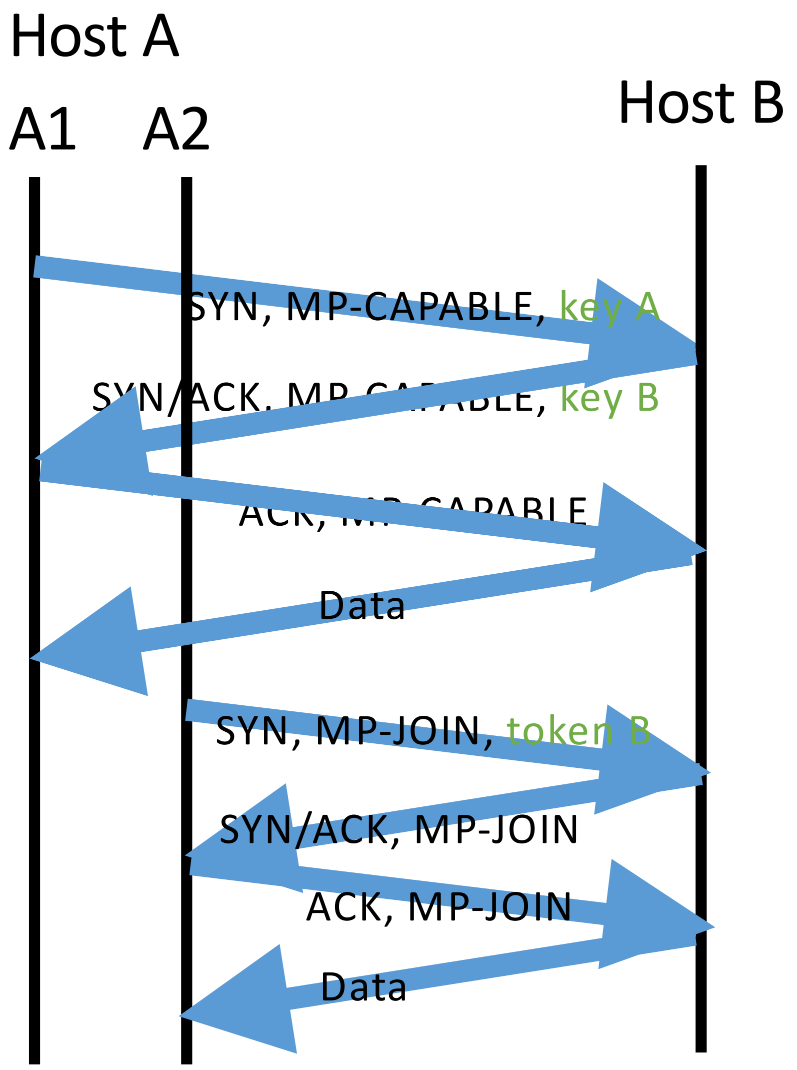
- Problem: Middleboxes may change host A's source IP



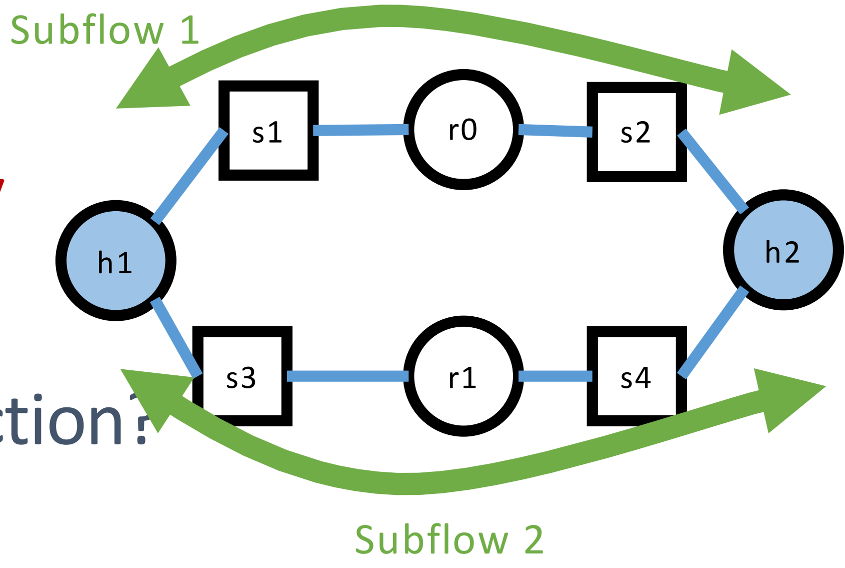
2. Adding New Subflows: Identification

- TCP flows traditionally identified by <source IP:source port, dest IP:dest port>

- Problem: Middleboxes may change host A's source IP
  → add a token to identify the connection
  - token B = hash(key B)



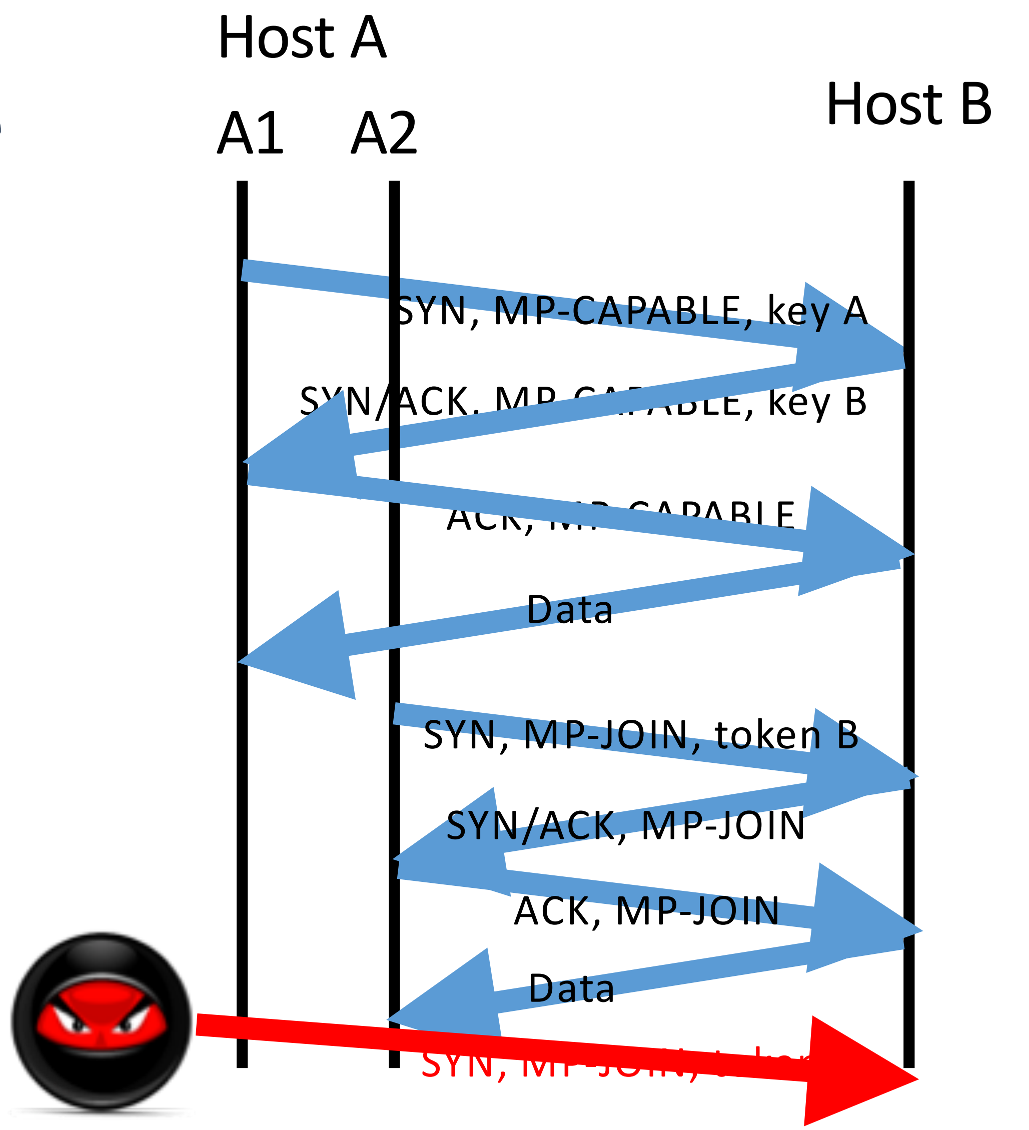
Adding New Subflows: Summary



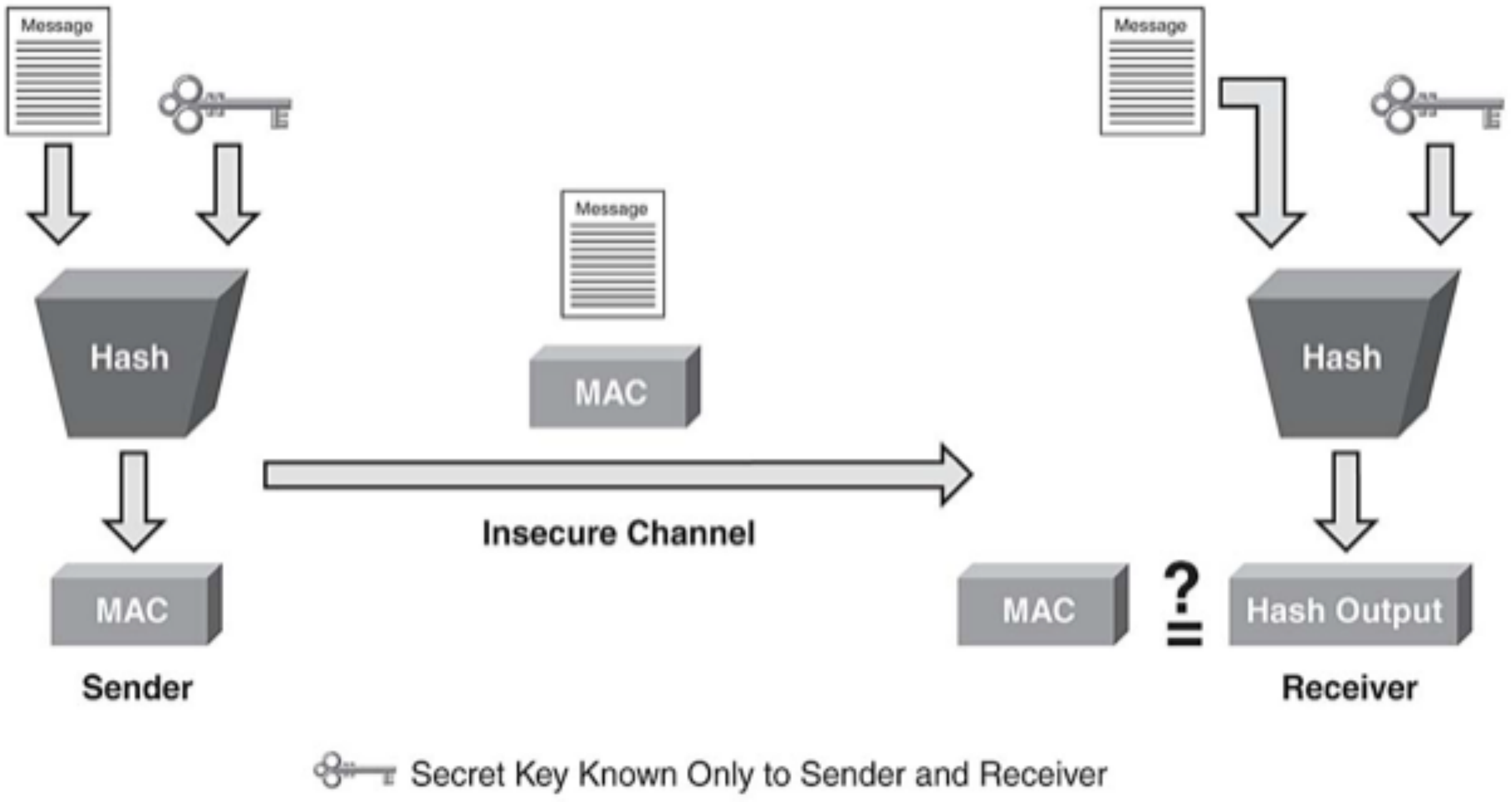
- How to associate a new subflow with the connection?
  - Use a token generated from original subflow set-up
- How to start using the new subflow?
  - Simply start sending packets with new IP/port pairs
  - … and associate them with the existing connection
- How could two end-points learn about extra IP addresses for establishing new subflows?
  - One end-point establishes a new subflow, to already-known address(es) at the other end-point

Adding New Subflows: Authentication

- Problem: attacker could use the same token
  → authentication using HMAC



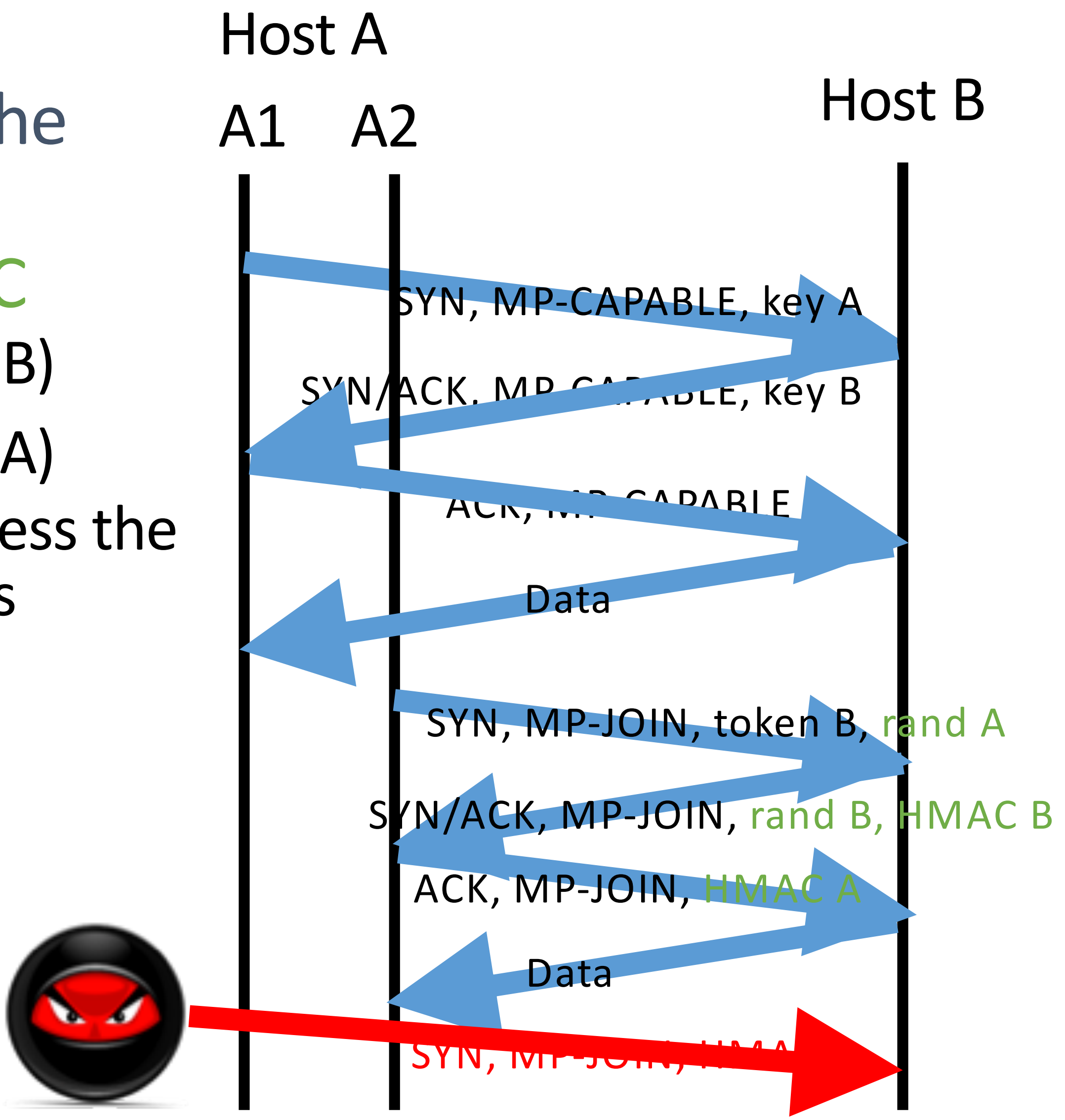
Hash-based Message Authentication Code (HMAC)



Source: http://www.networkworld.com/article/2268575/lan-wan/chapter-2--ssl-vpn-technology.html

Adding New Subflows: Authentication

- Problem: attacker could use the same token
  → authentication using HMAC
  - HMAC A = f(key A, key B , rand B)
  - HMAC B = f(key A, key B , rand A)
  - Attacker gets one chance to guess the HMAC, otherwise rand changes



Adding New Subflows: Security Summary

- Security
  - Malicious parties creating subflows
  - To hijack (part of) the connection
- How to bootstrap security?
  - Include a random key during connection set-up
  - … and use it to verify authenticity of new subflows
- How to identify the connection on new subflows?
  - A token generated from the key
- How to authenticate the addition of subflows?
  - Exchanging nonces and computing message authentication codes using the keys

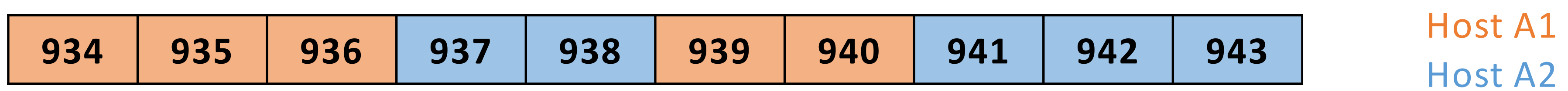
Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
    - Basics
    - Sequence numbers
    - Congestion control
  - CUBIC
  - BBR

Q: How should flows compete for bandwidth when there is congestion in the network?

Sequence Numbers

- Per-flow or per-subflow sequence #s?
- Naïve: Use one sequence of numbers, send a subset of those numbers on each subflow

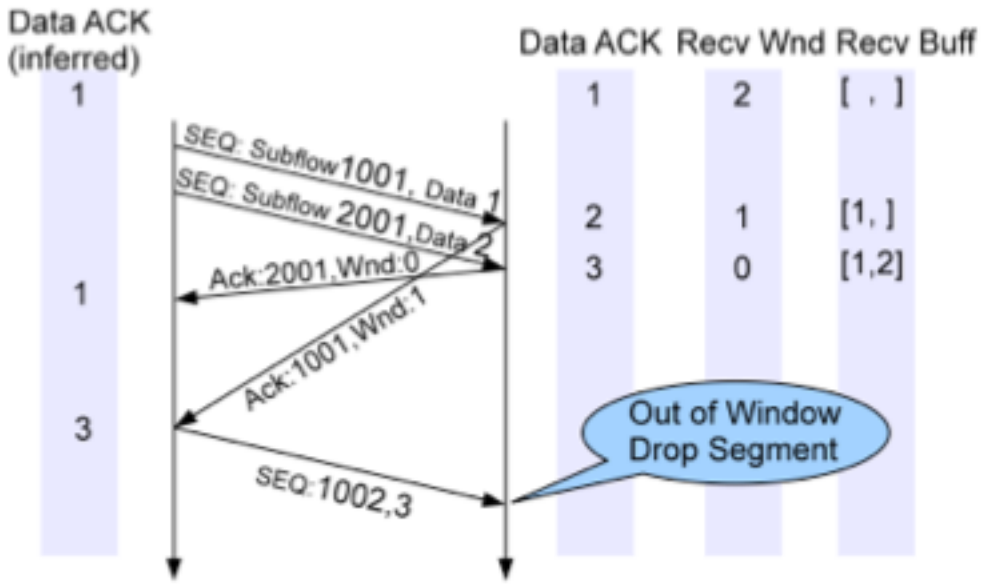


- Problem: middleboxes re-initialize sequence numbers
- Problem: middleboxes don't like gaps in sequence numbers

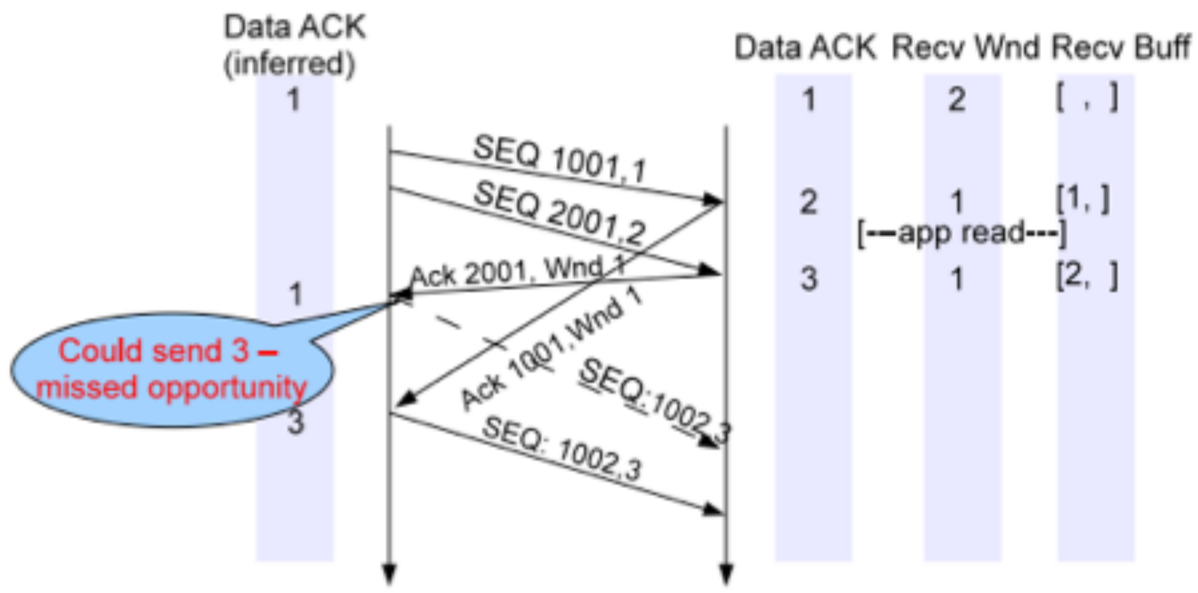
→ use flow-level sequence numbers ALONG WITH per-subflow sequence numbers

Sequence Numbers: ACKs

- Flow-level sequence numbers needed
- Are flow-level ACKs needed? Can we infer them from subflow ACKs?
- Examples below: no flow level ACK, receive buffer size 2



(a) Drops due to incorrect inference

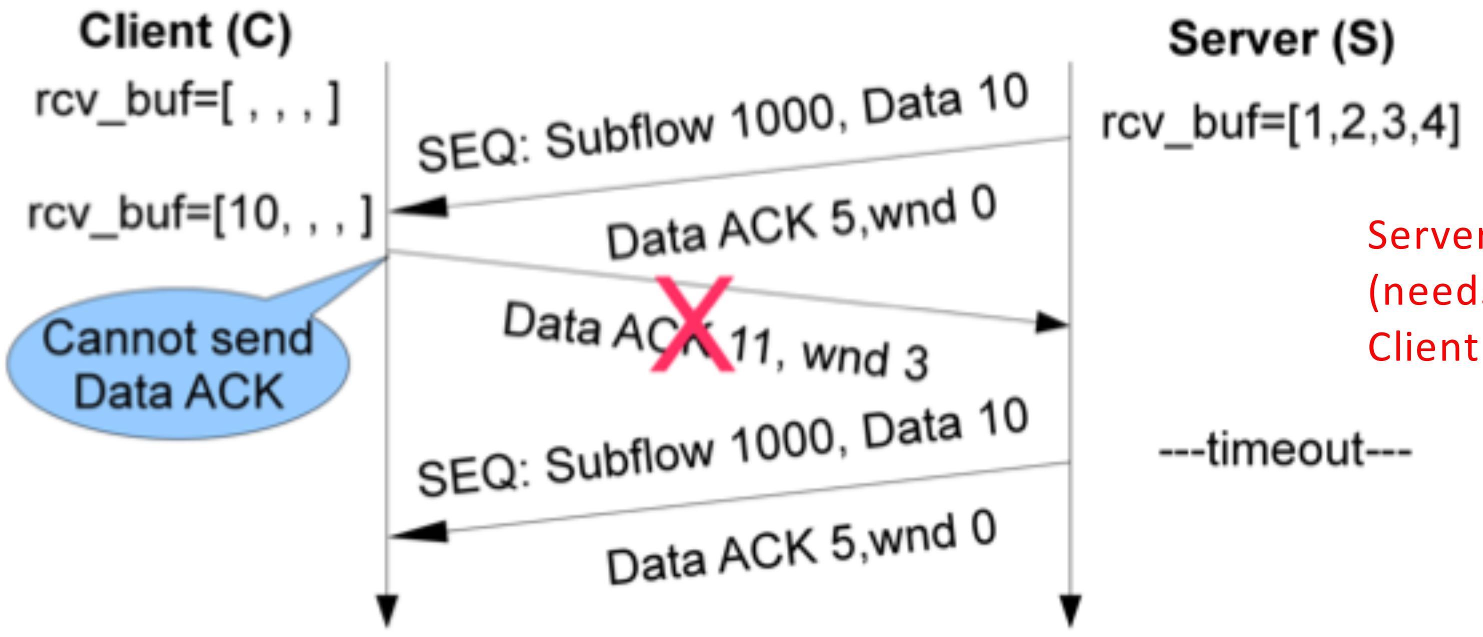


(b) Stalls due to incorrect inference

→ use flow-level ACKs along with per-subflow ACKs

Subflow Sequence Numbers: Where to encode?

- Naïve solution: Encode in data payload
- Problem: Data ACKs can get stuck from flow control



Server won't read until finished sending (needs to receive ACK to finish sending)
Client can't ACK until S reads

Source: [3]

→ Encode data sequence numbers and ACKs in TCP options

Sequence Numbers: Mapping

- Mapping from subflow sequence number to flow sequence number

- Naïve: On each packet, record absolute value of corresponding flow sequence number

- TCP segmentation offload (TSO)
  - Divide large segments into smaller chunks
  - Performed by NICs to save CPU resources

- Problem: TSO copies same flow sequence number onto multiple packets

  → Solution: record initial offset of subflow sequence number to flow sequence number
  - Doesn’t matter if this number is copied to multiple packets

Retransmissions

- What if data on a subflow times out?
  - Can resend on a different subflow

- Still need to retransmit on the original subflow
  - No holes in subflow sequence numbers for middlebox compatibility
  - Wastes bandwidth

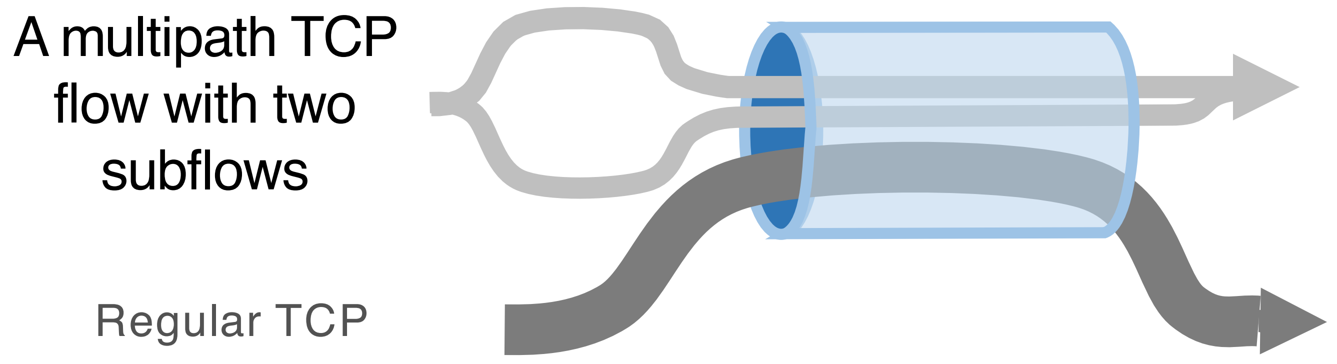
- Protocol not defined by RFC
  - *Aggressive*: Re-transmit every packet not received on a different subflow
  - *Conservative*: Re-transmit after fixed number of retries on the original subflow

Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
    - Basics
    - Sequence numbers
    - Congestion control
  - CUBIC
  - BBR

Q: How should flows compete for bandwidth when there is congestion in the network?

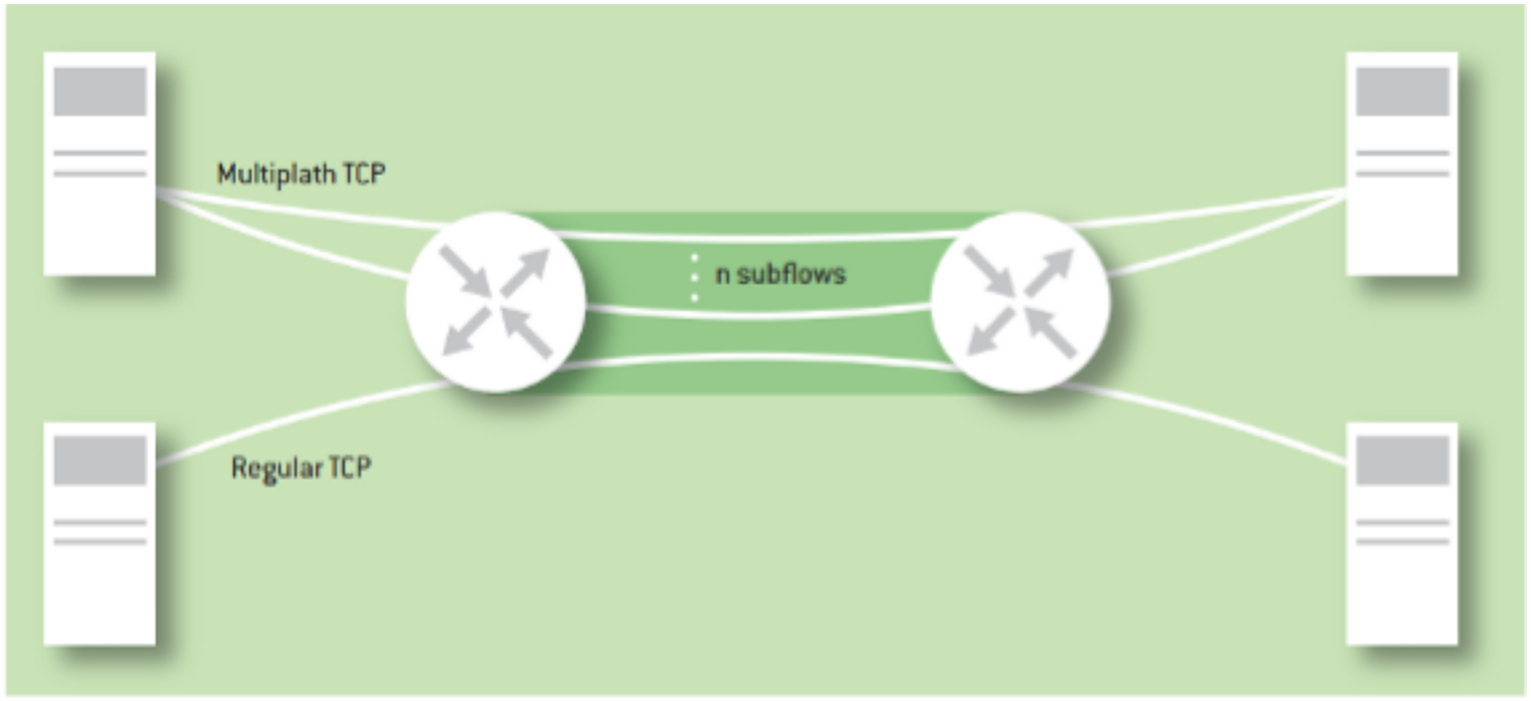
Goal #1: Fairness at Shared Bottlenecks



To be fair, Multipath TCP should take as much capacity as TCP at a bottleneck link, no matter how many paths it is using.

Congestion Control

- Naïve: use TCP congestion control separately on each path
- Problem: Not TCP-friendly



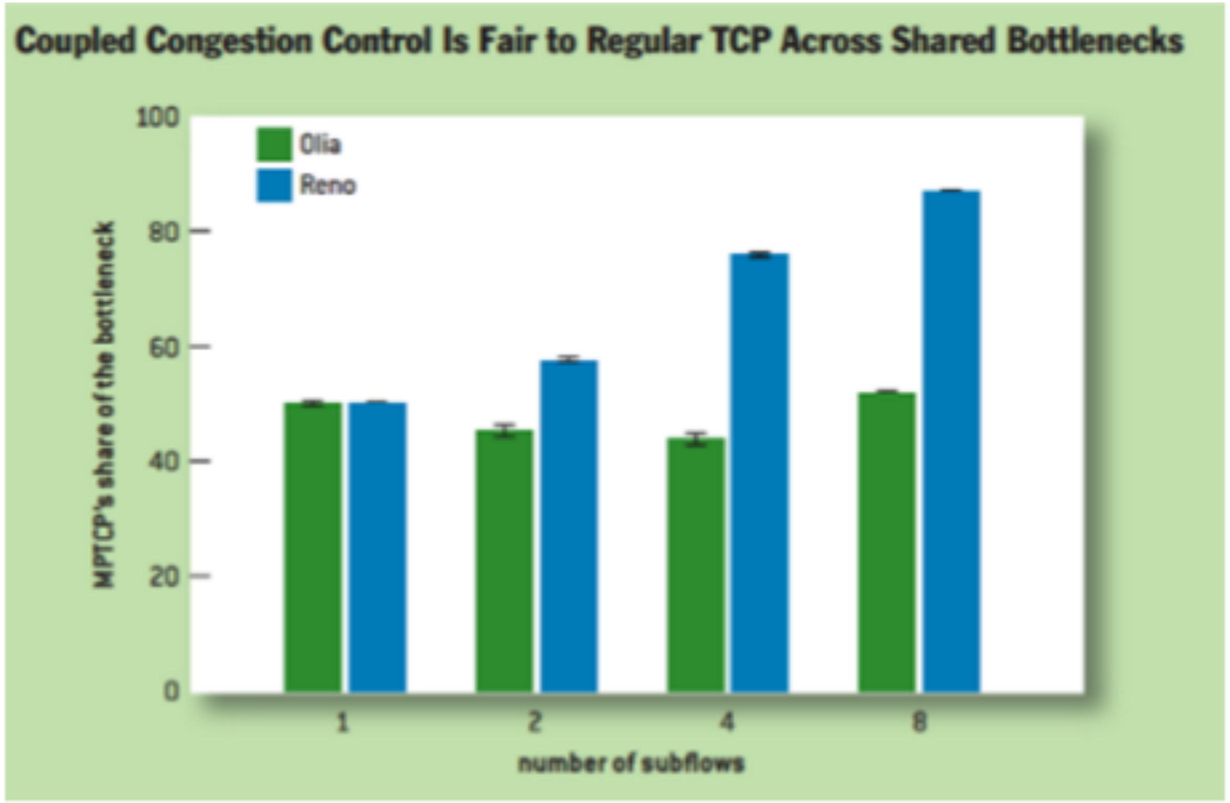
For example:
2 clients
Client A has 2 MPTCP subflows
Client B is regular TCP

How much will client A receive?

Source: [2]

Congestion Control

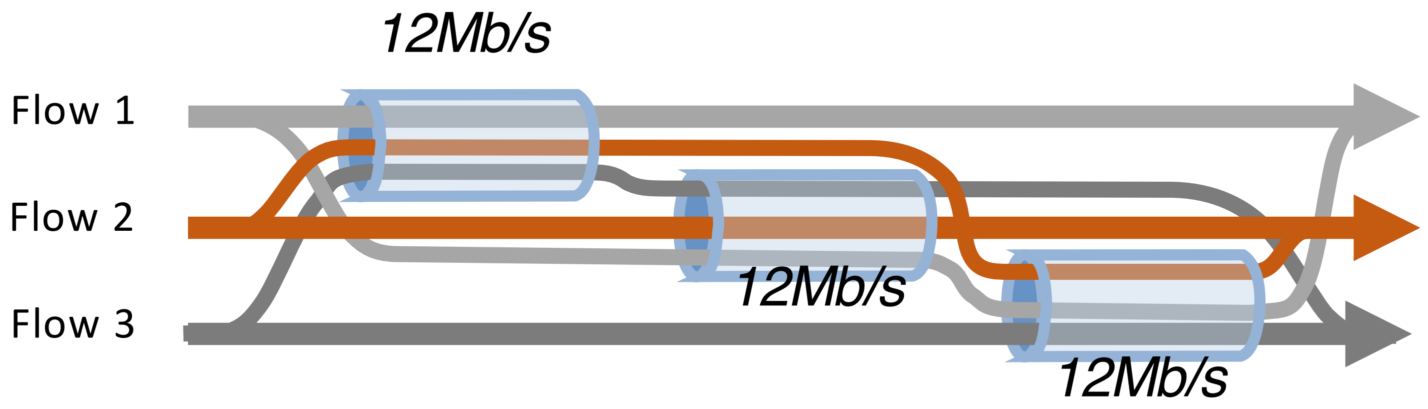
- Solution: Congestion control coupled across subflows
  - Many proposed MPTCP algorithms



- OLIA = new MPTCP congestion control
- Reno = using regular RENO on each MPTCP subflow

Source: [2]

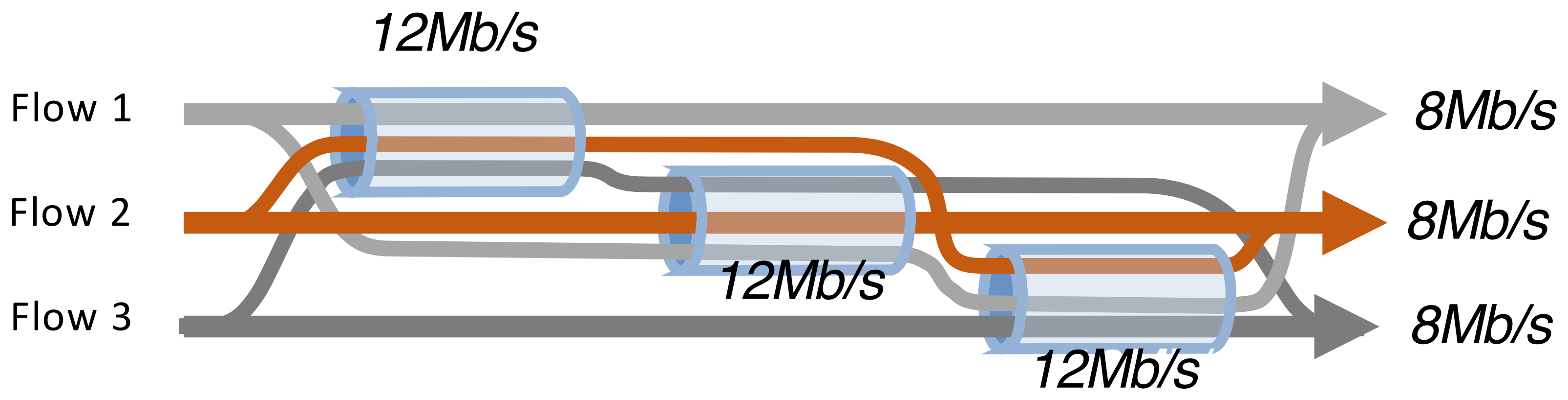
Goal #2: Use Efficient Paths



Each flow has a choice of a 1-hop and a 2-hop path.

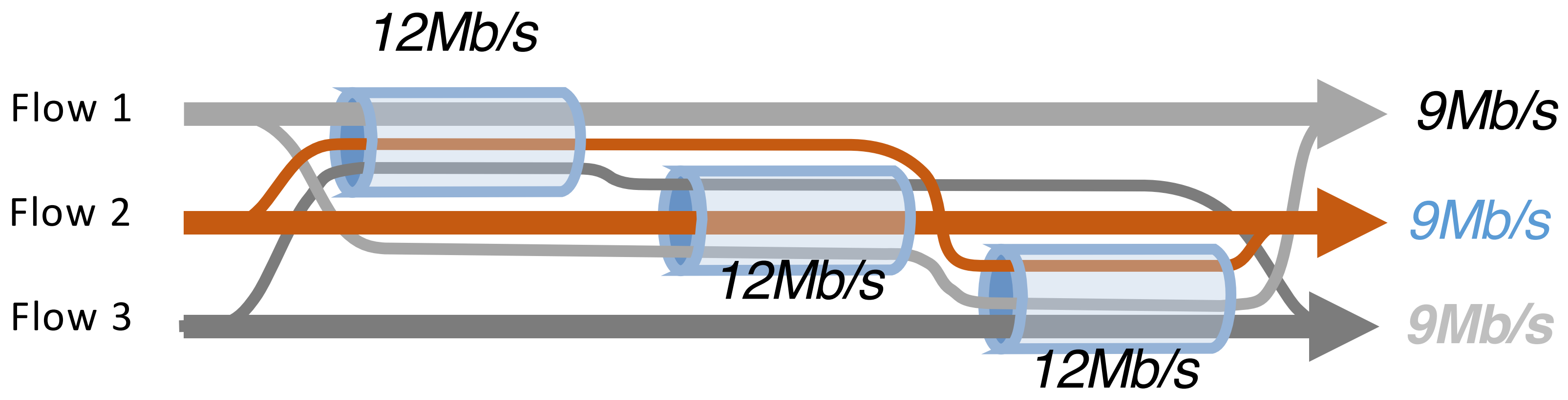
How should it split its traffic?

Use Efficient Paths



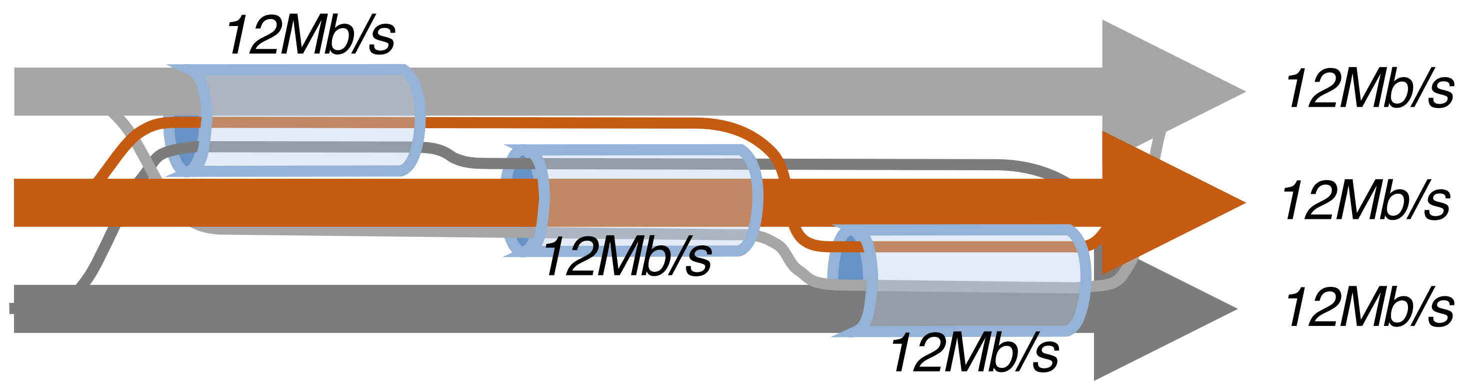
If each flow split its traffic 1:1 ...

Use Efficient Paths



If each flow split its traffic 2:1 ...

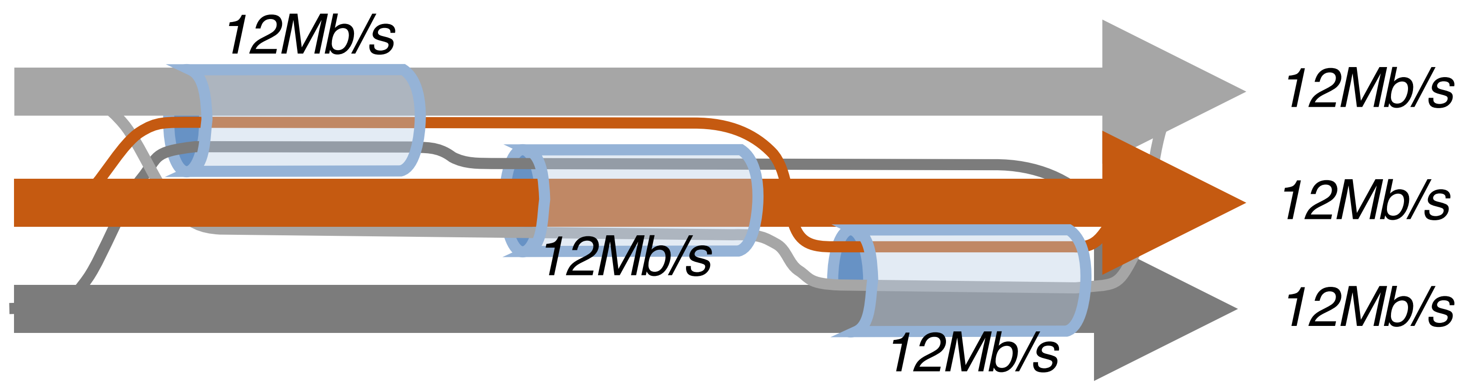
Use Efficient Paths



Better: Each connection on a one-hop path

Each connection should send all traffic on the least-congested paths

Use Efficient Paths



Better: Each connection on a one-hop path

Each connection should send all traffic on the least-congested paths

But keep some traffic on the alternate paths as a probe

Use Efficient Paths

- Least-congested paths may not be best!
  - Due to differences in round-trip time

- Example
  - Two paths
    - WiFi: high loss, low RTT
    - Cellular: low loss, high RTT
  - Using the least-congested path
    - Choose the cellular path, due to low loss
    - But, the RTT is high
    - So throughput is low!

- Thinking: Need a more rigorous way of adapting subflow rates...

Goal #3: Be Fair Compared to TCP

w_r: cwnd of path r
RTT_r: round-trip time of path r
R: set of all paths
S: subset of all paths
w^{TCP}_r: cwnd of single-path TCP

- To be fair, Multipath TCP should give a connection at least as much throughput as it would get with a single-path TCP on the best of its paths.
  - Ensure incentive for deploying MPTCP

$$\sum_{r \in R} \frac{\hat{w}_r}{\mathsf{RTT}_r} \geq \max_{r \in R} \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r}$$

- An Multipath TCP flow should take no more capacity on any path (or collection of paths) than if it was a single-path TCP flow using the best of those paths.
  - Do no harm!

$$\sum_{r \in S} \frac{\hat{w}_r}{\mathsf{RTT}_r} \leq \max_{r \in S} \frac{\hat{w}_r^{\mathsf{TCP}}}{\mathsf{RTT}_r} \quad \text{for all } S \subseteq R$$

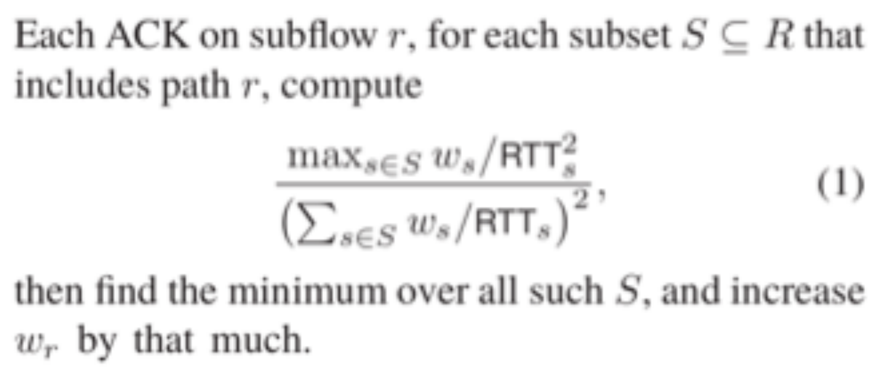
Achieving These Goals

- Regular TCP
  - Maintain a congestion window w
  - On an ACK, increase by 1/w (increase 1 per window)
  - On a loss, decrease by w/2
- MPTCP
  - Maintain a congestion window **per path** w_r
  - On an ACK on path r, increase w_r
  - On a loss on path r, decrease by $w_r/2$
- How much to increase w_r on an ACK?
  - If r is the only path at that bottleneck, increase by $1/w_r$
  - Otherwise...?

If Multiple Paths Share Bottleneck?

- Recall Goal #3: Don’t take any more bandwidth on a link than the best of the TCP paths would
  - But, where might the bottlenecks be?
  - Multiple paths might share the *same* bottleneck

- So, consider all possible *subsets* of the paths
  - Set R of paths used by a given MPTCP flow
  - Subset S of R that includes path r
- E.g., consider r = path 3
  - Suppose paths 1, 3, and 4 share a bottleneck
  - … but, path 2 does not
  - Then, we care about S = {1,3,4}


Each ACK on subflow r, for each subset $S \subseteq R$ that includes path r, compute

$$\frac{\max_{s \in S} w_s/\mathsf{RTT}_s^2}{\left(\sum_{s \in S} w_s/\mathsf{RTT}_s\right)^2}, \qquad (1)$$

then find the minimum over all such S, and increase w_r by that much.

Achieving These Goals

w_r: cwnd of path r
RTT_r: round-trip time of path r
R: set of all paths
S: subset of all paths
w^{TCP}_r: cwnd of single-path TCP

- What is the *best* of these subflows achieving?
  - Path s is achieving throughput of w_s/RTT_s
  - So best path is getting $\max_s(w_s/RTT_s)$
- What *total* bandwidth are these subflows getting?
  - Across *all* subflows sharing that bottleneck
  - Sum over s in S of w_s/RTT_s
- Consider the *ratio* of the two
- And pick the results for the subset S with min ratio
  - There is some subset S where the best subflow is not doing that great (numerator small) $\rightarrow$ extra capacity for r
  - Many subflows are sharing the bottleneck (denominator large) $\rightarrow$ less capacity for r

Scheduling

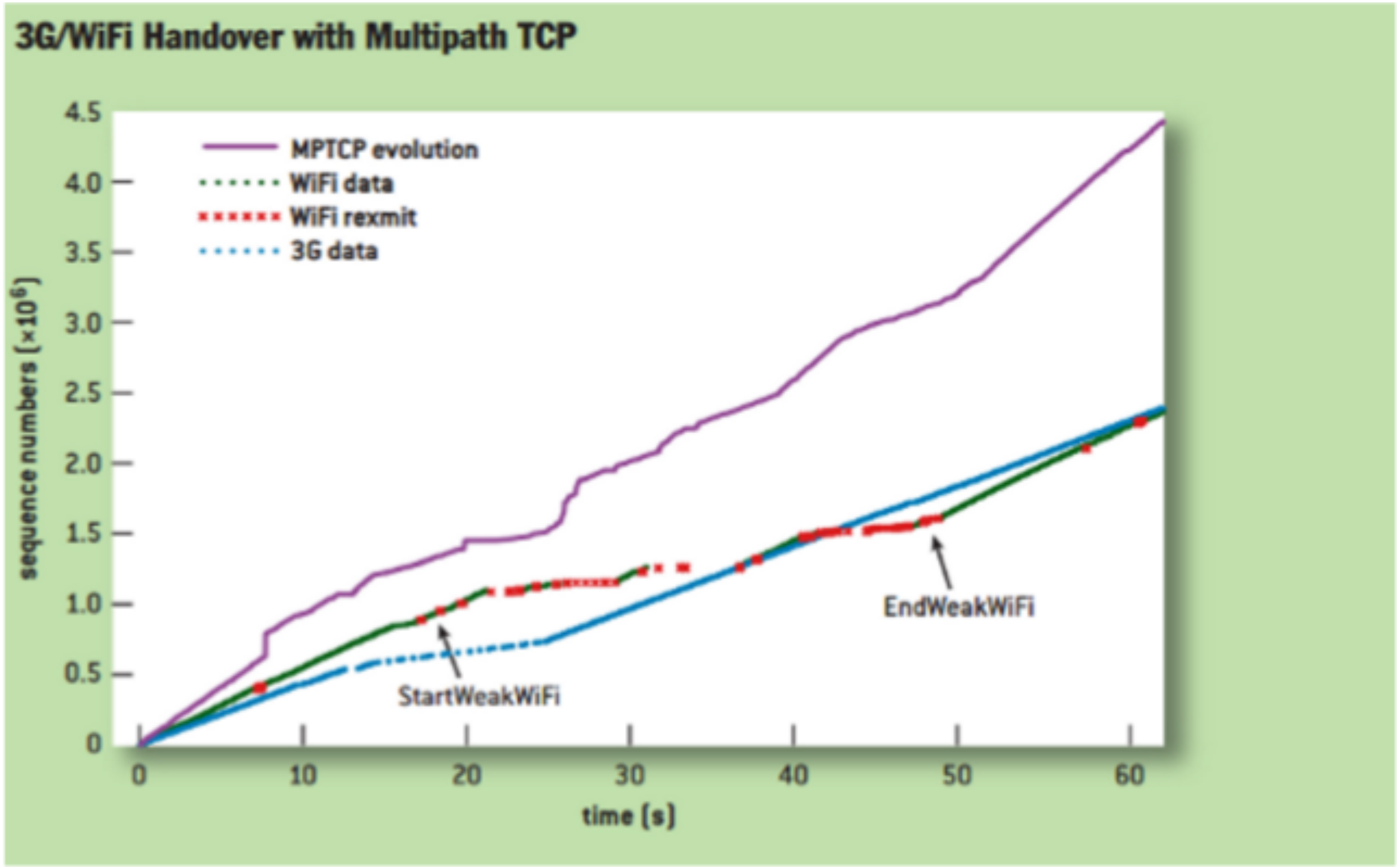
- When there is space in the congestion and receive windows, which subflow to transmit on?
  - Round-robin
  - Lowest-RTT first

- Issue: congestion window is ACK-clocked
  - Round-robin: if cwnd has space, send even if out of round-robin order?
  - Lowest-RTT first: if cwnd has space, send on higher-RTT subflow?

Use of Multipath TCP in iOS 7

- Multipath TCP since iOS 7 (fall 2013)
  - Primary TCP connection over WiFi
  - Backup TCP connection over cellular data
- Failover
  - If WiFi becomes unavailable…
  - … iOS 7 will use the cellular data connection
- For destinations controlled by Apple
  - E.g., Siri
- See https://support.apple.com/en-us/HT201373

MPTCP Example in Practice



Source: [2]

Sources

1. "Multipath TCP," Christoph Pasch and Olivier Bonaventure, *ACM Queue*, 2014.
2. TCP Extensions for Multipath Operation with Multiple Addresses, RFC 2684.
3. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," Raiciu et al., *USENIX NSDI* 2012.
4. "Design, implementation and evaluation of congestion control for multipath TCP", *USENIX NSDI* 2011.
5. Jennifer Rexford, COS 561