



CIS 553: Networked Systems

Congestion control

March 24, 2021



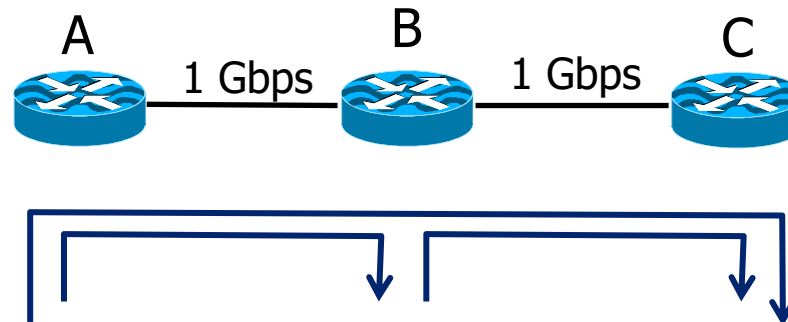
Agenda

- Transmission Control Protocol ✓
 - Congestion control ✓
 - Fairness ← NEXT
 - ACK Clocking
 - Router-assisted congestion control
- HTTP



Efficiency vs. Fairness

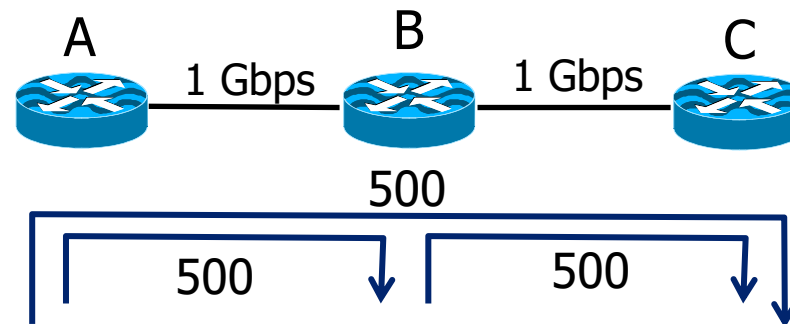
- Cannot always have both!
 - Example network with traffic $A \rightarrow B$, $B \rightarrow C$ and $A \rightarrow C$
 - All three flows want to use 1 Gbps
 - How much traffic can we carry?





Fairness?

- How would you allocate if you only cared about **fairness**?
 - Give equal bandwidth to each flow
 - A→B: 500 Mbps unit, B→C: 500 Mbps, and A→C, 500 Mbps
 - Total traffic carried is **1.5 Gbps**





Max-Min fairness

- For a single link, given set of bandwidth demands r_i and total bandwidth C
 - Allocation $a_i = \min(f, r_i)$
 - where f is the unique value such that $\text{Sum}(a_i) = C$
- If you don't get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size



Computing Max-Min Fairness

- To find it given a network, imagine “pouring water into the network”
 1. Start with all flows at rate 0
 2. Increase the flows until there is a new bottleneck in the network
 3. Hold fixed the rate of the flows that are bottlenecked
 4. Go to step 2 for any remaining flows



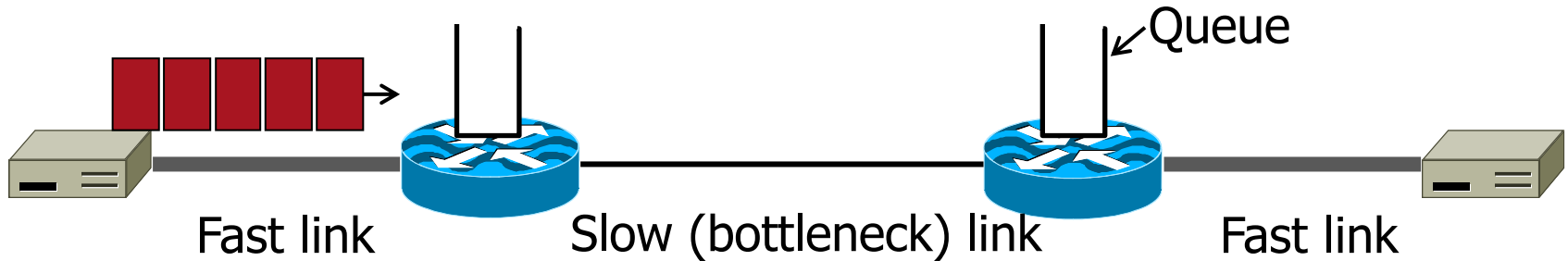
Agenda

- Transmission Control Protocol ✓
 - Congestion control ✓
 - Fairness ✓
 - ACK Clocking ← NEXT
 - Router-assisted congestion control
- HTTP



ACK Clocking

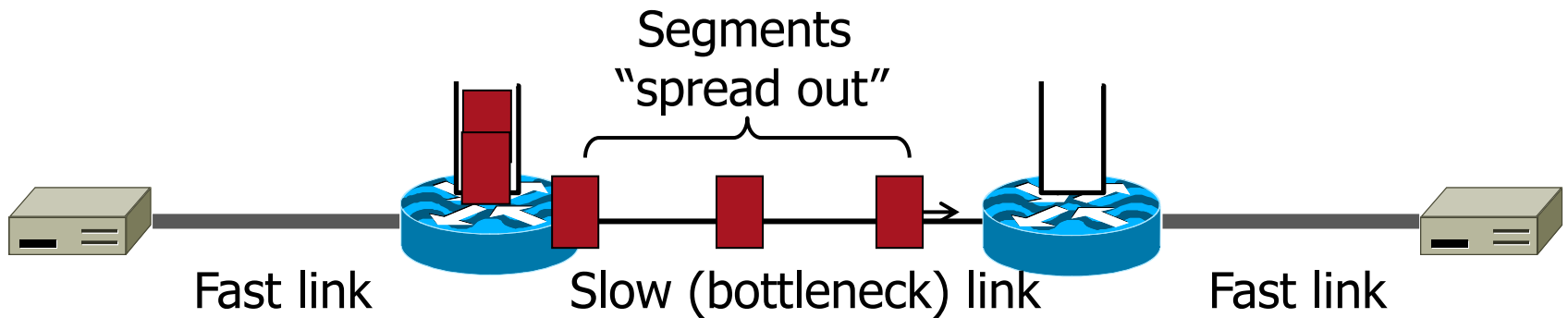
- Consider what happens when sender injects a burst of segments into the network





ACK Clocking

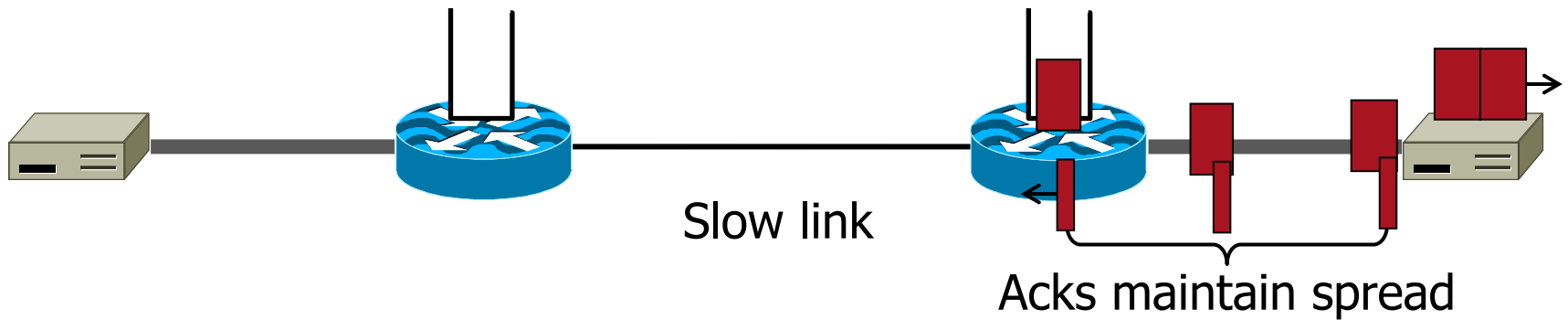
- Segments are buffered and spread out on slow link





ACK Clocking

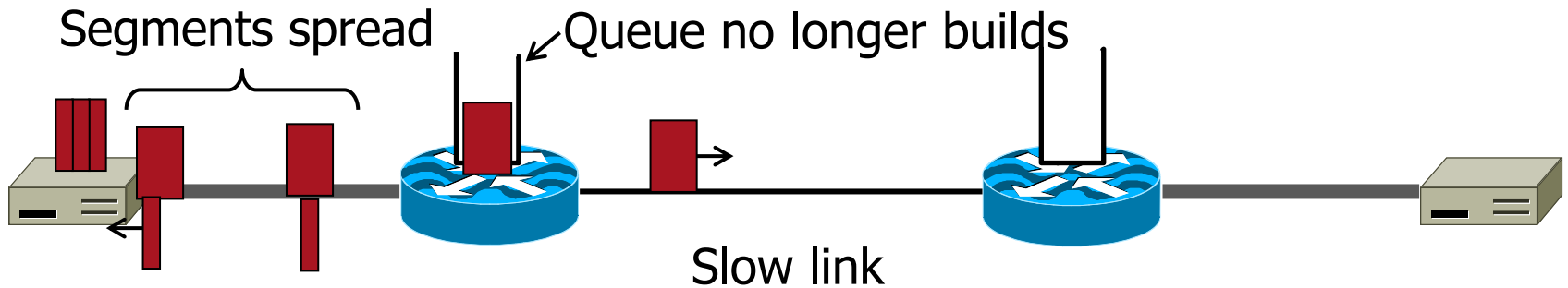
- ACKs maintain the spread back to the original sender





ACK Clocking

- Sender clocks new segments with the spread
 - Now sending at the bottleneck link without queuing!





ACK Clocking

- Helps the network run with low levels of loss and delay!
- The network has smoothed out the burst of data segments
- ACK clock transfers this smooth timing back to the sender
- Subsequent data segments are not sent in bursts so do not queue up in the network



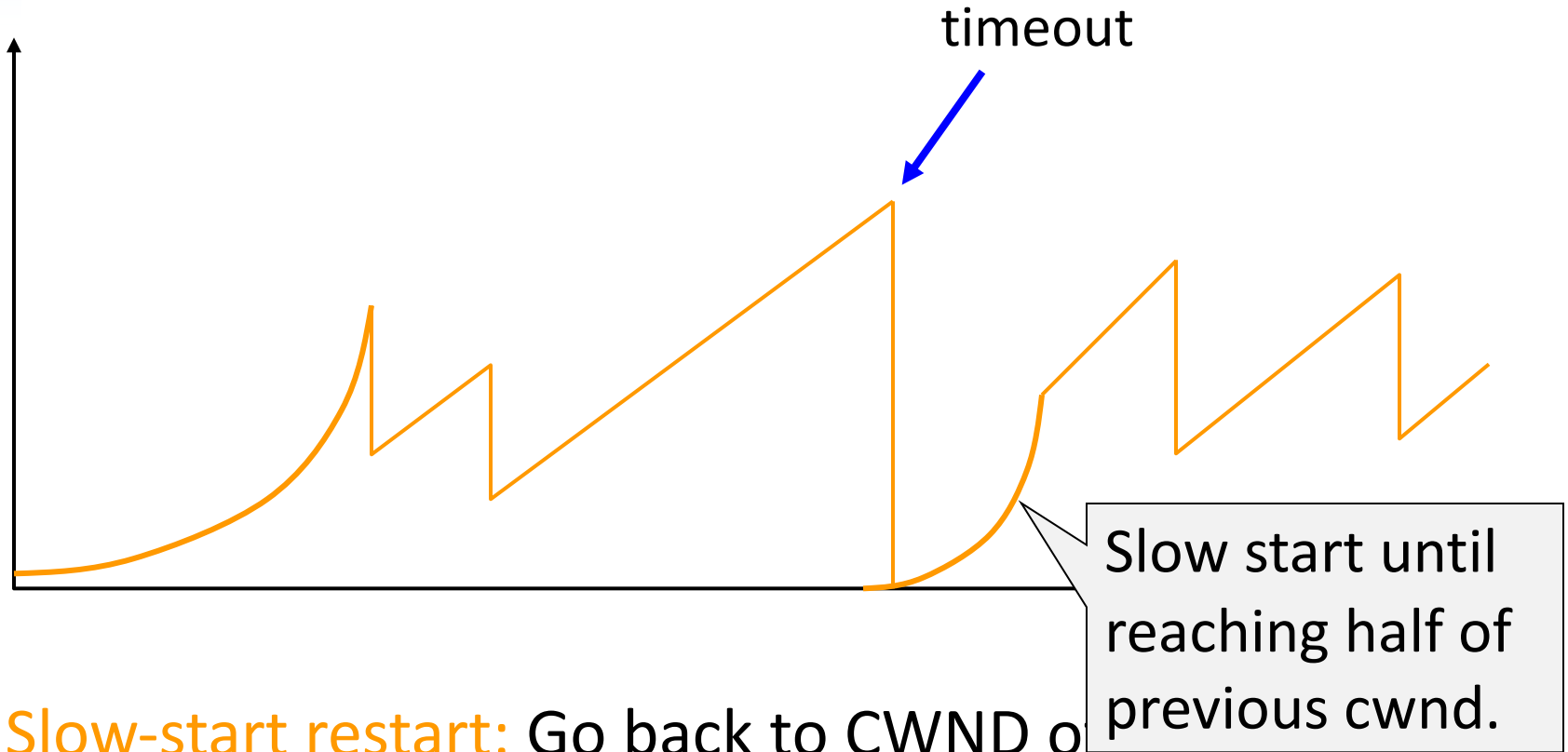
Timeouts and Idle Periods

- After a timeout or idle period:
 - We lose ACK clocking!
 - Also, network conditions change
 - Maybe many more flows are traversing the link
- Dangerous to start transmitting at the old rate
 - Previously-idle TCP sender might blast network
 - ... causing excessive congestion and packet loss
- So, some TCP implementations repeat slow start
 - Slow-start restart after an idle period



Repeating Slow Start After Idleness

Window



Slow-start restart: Go back to CWND of $\frac{1}{2}$ previous cwnd. Advantage of knowing the previous value of CWND.



TCP flavors

- TCP-Tahoe
 - $CWND = 1$ on 3 dupACKs
- TCP-Reno
 - $CWND = 1$ on timeout
 - $CWND = CWND/2$ on 3 dupACKs
- TCP-newReno
 - TCP-Reno + fast recovery
- TCP-SACK
 - Incorporates selective acknowledgements

Our default assumption



How can they coexist?

- All follow the same principle
 - Increase CWND on good news
 - Decrease CWND on bad news
- Notion of TCP-friendliness