The Transport Layer: Flow Control, Congestion Control

CS 352, Lecture 9, Spring 2020

http://www.cs.rutgers.edu/~sn624/352

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Course announcements

- Quiz 3 will go online later today: covers this lecture only
  - Due Tuesday 03/03
- Project 1 due today
  - Project 2 will go online this weekend

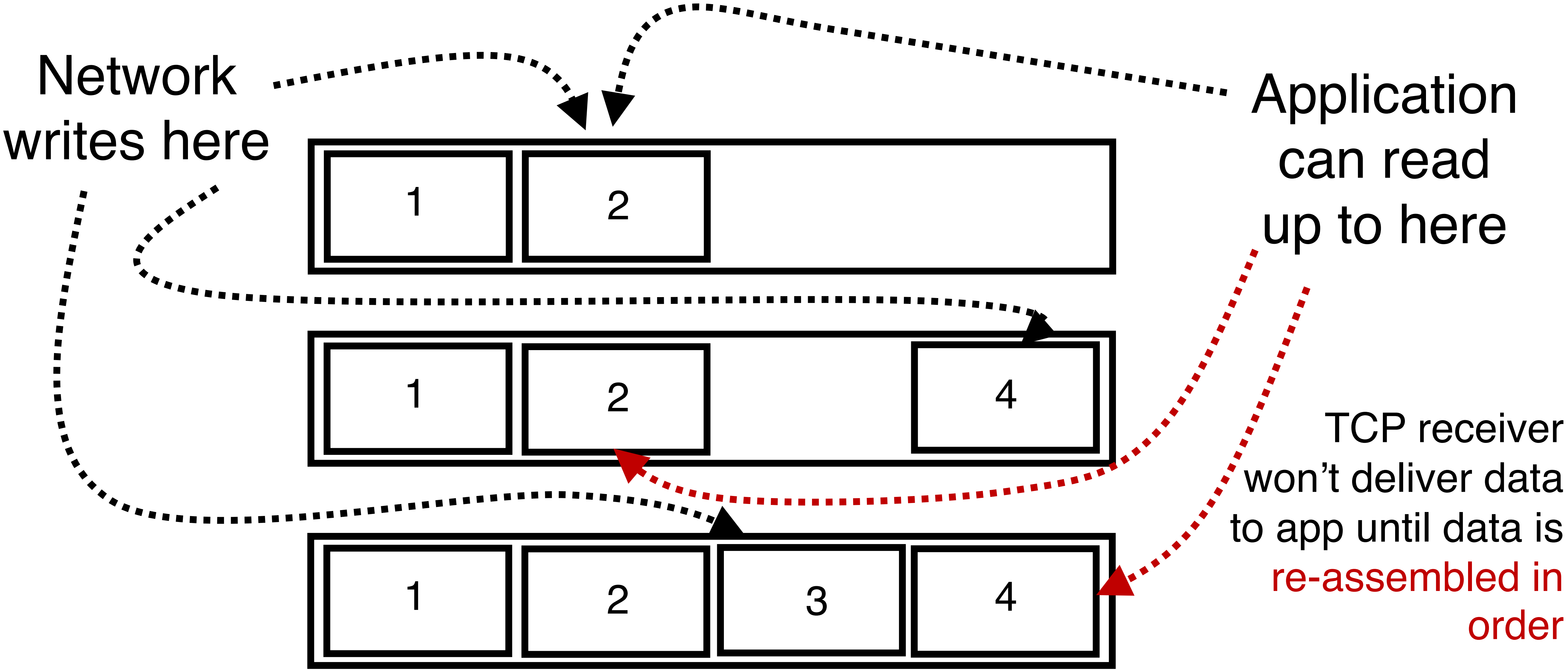
- I hear you:
  - More practice on calculation-style questions in recitations
  - More time for project questions during recitations
  - Suggested textbook problems close to exams
  - More time for class activities and demonstrations

Review of concepts

- Stop-and-wait reliability: ACK, RTO, sequence numbers
- Pipelined reliability: selective repeat vs. go-back-N
    - Sequence numbers even more important in pipelined reliability
    - Cumulative versus selective ACKs
- Sliding window, window sizes
- Need buffers on the receiver side: why?
    - Avoid needless sender retransmission
    - Keep data in order to deliver to application

Ordered Delivery

Ordering at the receiver side



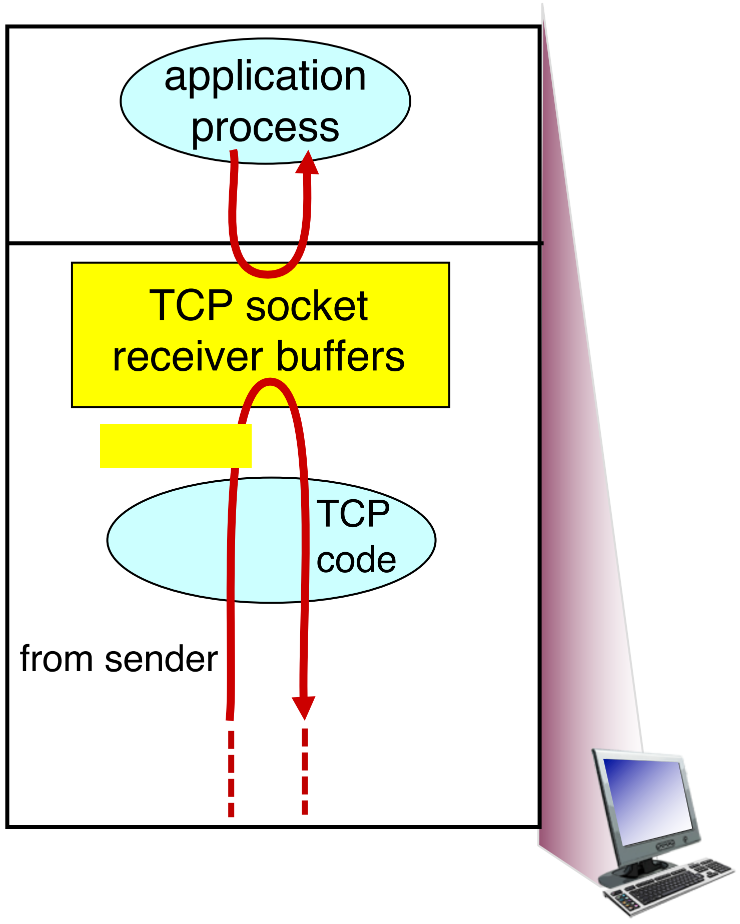
Socket buffer memory on the receiver

Flow control

Implications of buffering at the receiver

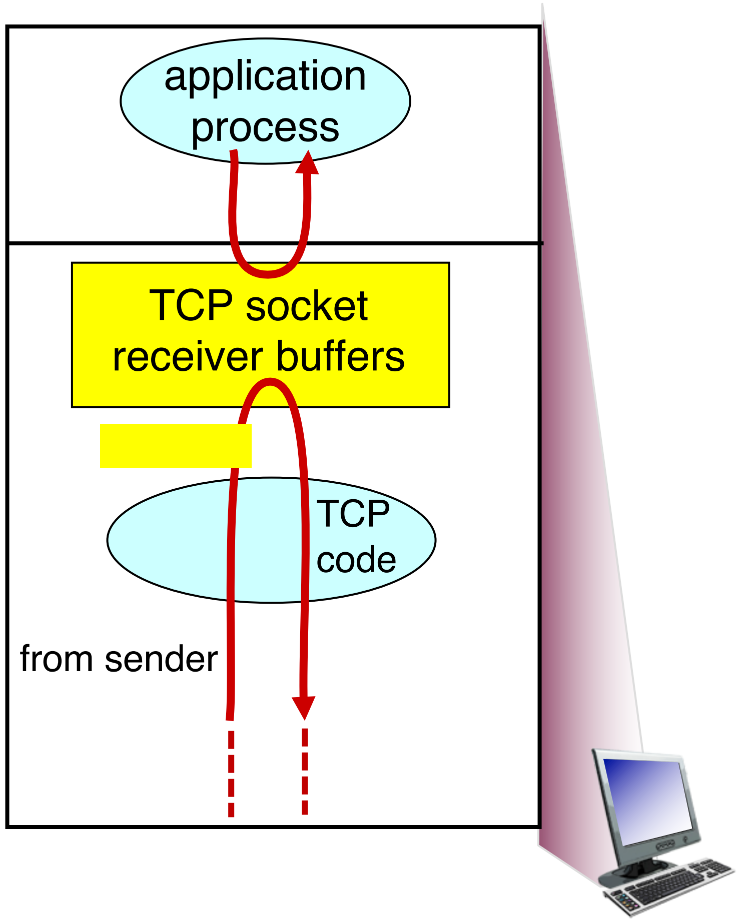
- Applications may read data slower than the sender is pushing data in
  - e.g., what if you never called `recv()`?

- Hence, the permissible window size may vary over time.



Implications of buffering at the receiver

- A TCP sender can only send as much as the free receiver buffer space available, before packets are dropped at the receiver
- This number is called the receiver window size or advertised window size
- TCP is said to implement flow control
- Sender's window size is bounded by the advertised window size.



receiver protocol stack

TCP headers

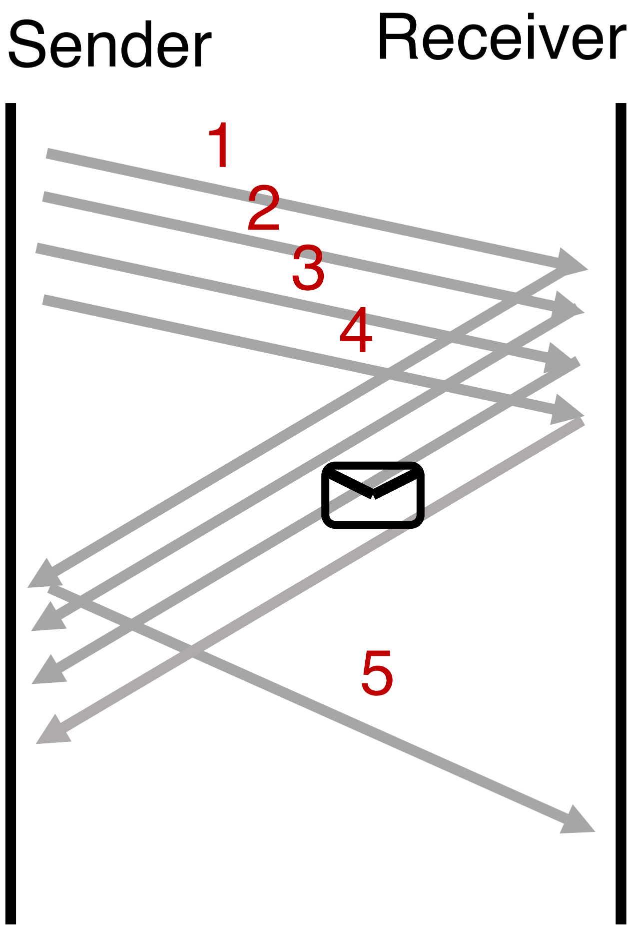
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                        TCP Header Format

      Note that one tick mark represents one bit position.
```

Sizing the receiver socket buffer

- For each socket, there is a default size for the memory allocated to the receiving socket buffer
    - Unimaginatively called the receiver socket buffer size
- If this number is too small, sender can't keep too many packets in flight ➔ lower throughput
- If the number is too large, consumes too much memory
- How big should the receiver socket buffer be?

Sizing the receiver socket buffer

- Assume that, on average, receiver can read data as fast as sender sends
- Further, suppose data is received in order
- Receiver still needs to buffer data: why?
  - Sender can send data in bursts
- Buffer needed is the size of the largest possible burst
  - What is this value?



Bandwidth-delay product

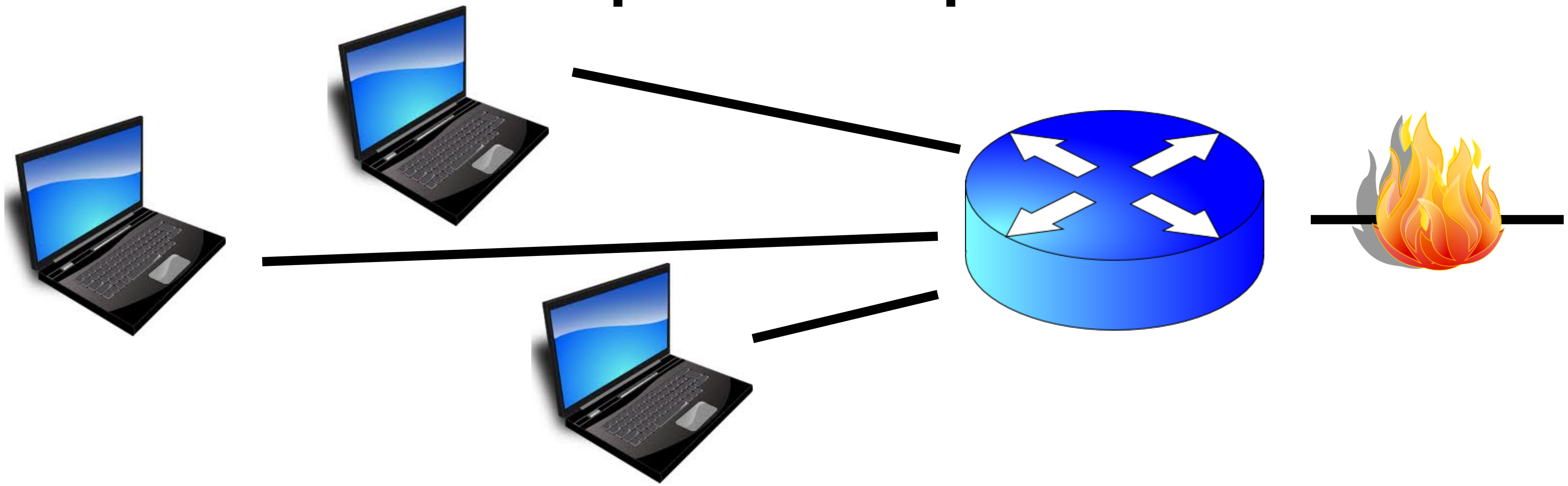
- The maximum amount of in-flight data the sender can have
- Bandwidth between the sender and receiver * round-trip time

- Example:
  - Sender can send data at 1 Mbit/s
  - Round-trip time is 100 ms
  - Maximum amount of in-flight data == maximum burst == 100 Kbit

- What if bandwidth was 100 Gbit/s and RTT was 500 ms?
  - Need a large socket buffer to capture the burst
  - Smaller socket buffer would degrade throughput to well below 100G

Congestion Control

How should multiple endpoints share net?



- It is difficult to know where the bottleneck link is
- It is difficult to know how many other endpoints are using that link
- Endpoints may join and leave at any time
- Network paths may change over time, leading to different bottleneck links (with different link rates) over time

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

No one can centrally view or control all the endpoints and bottlenecks in the Internet.

Every endpoint must try to reach a globally good outcome by itself: i.e., in a distributed fashion.

This also puts a lot of trust in endpoints.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

If there is spare capacity in the bottleneck link, the endpoints should use it.

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

If there are N endpoints sharing a bottleneck link, they should be able to get equitable shares of the link's capacity.

For example: 1/N'th of the link capacity.

Flow and Congestion control

- Flow control tries not to overwhelm the receiver
- Congestion control tries not to overwhelm routers

- Flow control manages the receiver’s buffers
- Congestion control manages the bottleneck link’s capacity and the bottleneck router’s buffers

Feedback and Actions

The signals and knobs of congestion control

The approach that the Internet takes is to use a distributed algorithm to converge to an efficient and fair outcome.

So, how to achieve this?

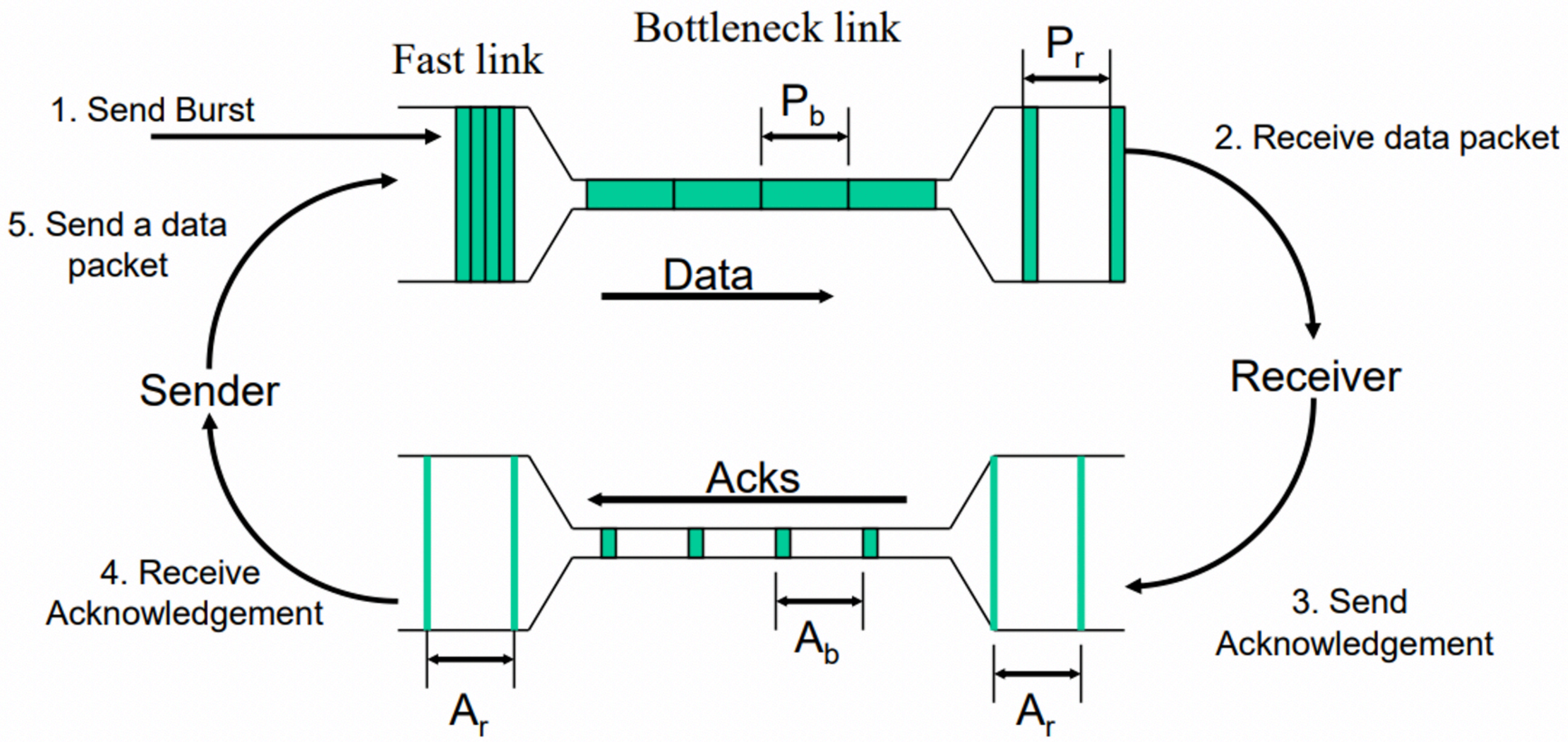
Feedback from network offers clues…

- Signals
  - Packets being dropped (ex, RTO fires)
  - Packets being delayed
  - Rate of incoming ACKs

"Implicit" feedback signals (more on explicit signals later)

- Knobs
  - What can you change to "probe" the sending rate?
  - Suppose receiver buffer is unbounded:
  - Let's call the amount of in-flight data per RTT the congestion window
  - Increase congestion window: e.g., by x or by a factor of x
  - Decrease congestion window: e.g., by x or by a factor of x

Steady state: Self clocking/ACK clocking

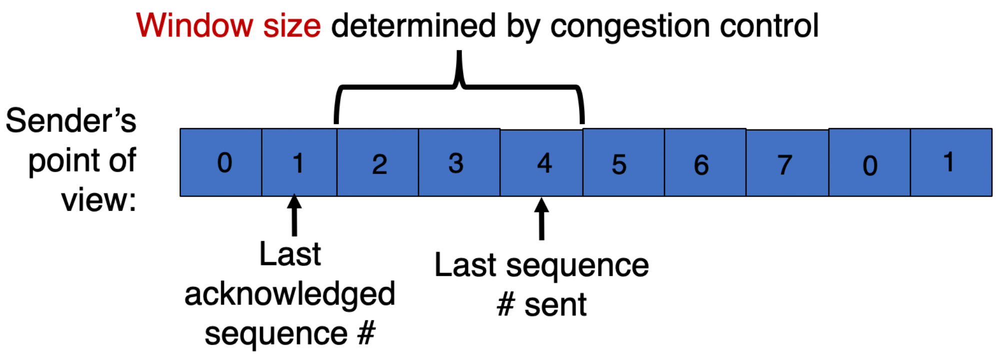


Given: $P_b = P_r = A_r = A_b = A_r$ (in units of time)
Sending a packet on each ACK keeps the bottleneck link busy

Time for an activity

TCP congestion window

- Congestion window (cwnd): an estimate of in-flight data needed to keep the pipe full and achieve self-clocking
- Sending window = min(congestion window, receiver advertised window). Why min?
  - Overwhelm neither the receiver nor network routers
- Use sliding window concept
  - Window size is adjusted by congestion



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
- Initial rate is slow but ramps up exponentially fast
- On loss, restart from cwnd := 1 MSS
- Is this good enough?