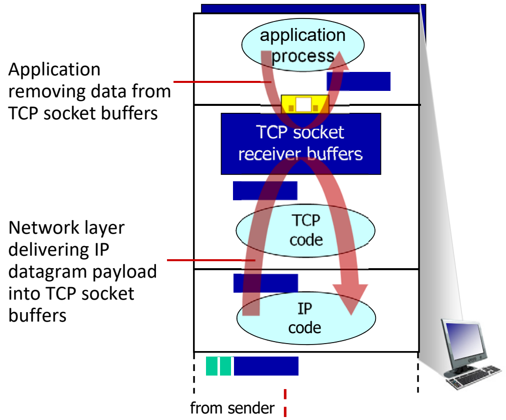
Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

TCP flow control

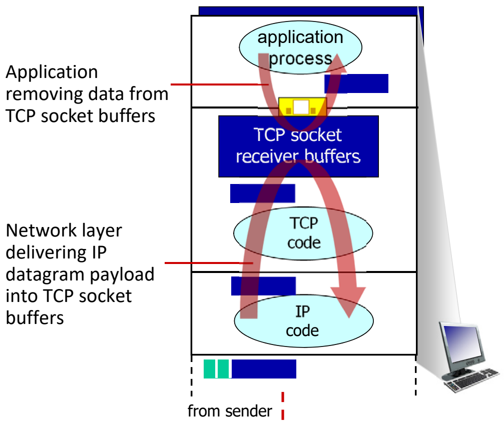
- At network and transport layers: payloads of new coming (3^{rd}-layer) datagrams are
  - brought up to the transport layer
  - saved into TCP socket buffer
- at application layer: an application process
  - performs socket reads
  - removes data from TCP socket buffer



receiver protocol stack

TCP flow control

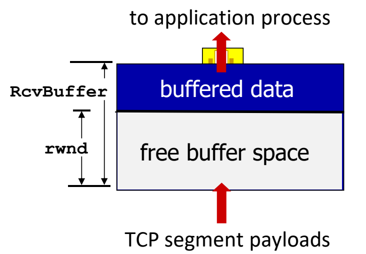
- what if network layer delivers data faster than application layer removes data from socket buffers?
  - buffer overflow
    - retransmissions
- flow control
  - receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



receiver protocol stack

TCP flow control

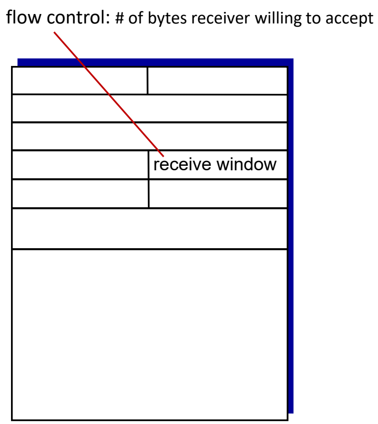
- **`RcvBuffer`** (receive buffer)
  - buffered data
  - free buffer space or called **`rwnd`** (receive window)
- the size of **`rwnd`** changes dynamically
  - buffer overflow should be avoided
  - but how?



TCP receiver-side buffering

TCP flow control

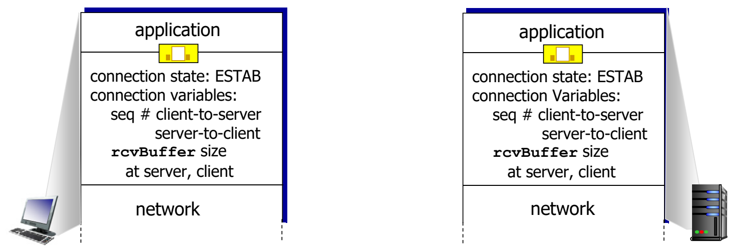
- TCP receiver "advertises" its free buffer space in the **receive window** field in TCP header
  - `RcvBuffer` size is set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust RcvBuffer
- sender limits the amount of unACKed ("in-flight") data to **rwnd** it received
  - LastByteSent – LastByteAcked ≤ rwnd
  - this guarantees receive buffer will not overflow



TCP segment format

TCP connection management

before exchanging data, sender/receiver "handshake":
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., initial seq #)

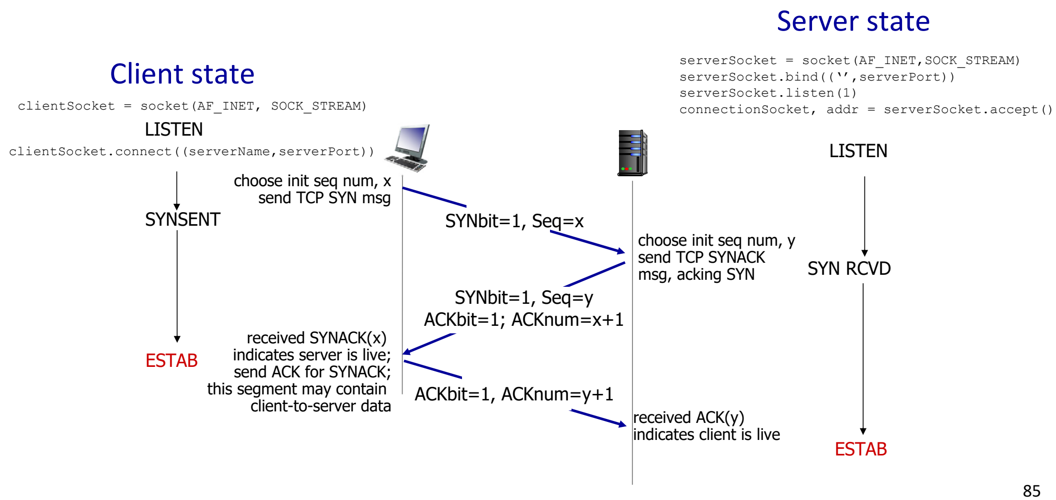



```
Socket clientSocket =
  newSocket("hostname","port number");
```

```
Socket connectionSocket =
  welcomeSocket.accept();
```

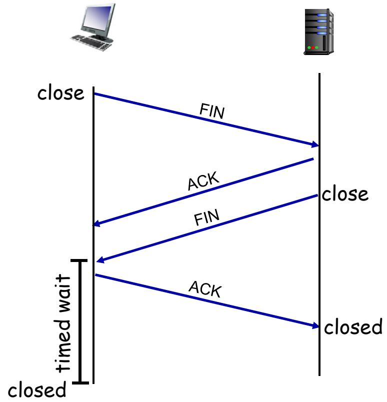

A human 3-way handshake protocol

TCP 3-way handshake



Closing a TCP connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled



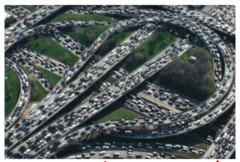
Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- **Principles of congestion control**
- TCP congestion control
- Evolution of transport-layer functionality

Principles of congestion control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!



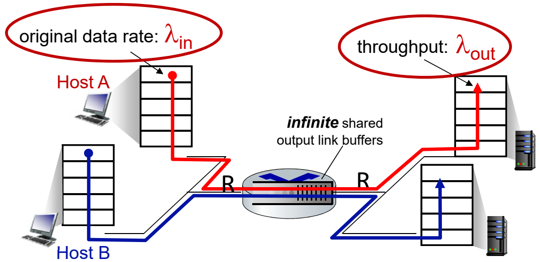
congestion control:
too many senders, sending too fast

flow control:
one sender too fast for one receiver

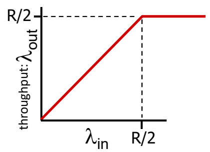
Causes/costs of congestion: scenario 1

Simplest scenario:

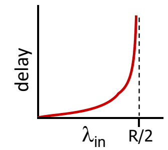
- one router, infinite buffers
- input, output link capacity: R
- two flows
- no retransmissions needed
  - $\lambda'_{in} = \lambda_{in}$



Q: What happens as arrival rate λ_{in} approaches R/2?



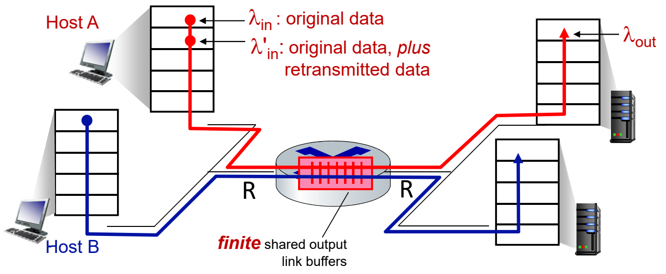
maximum per-connection throughput: R/2



large delays as arrival rate λ_{in} approaches capacity

Causes/costs of congestion: scenario 2

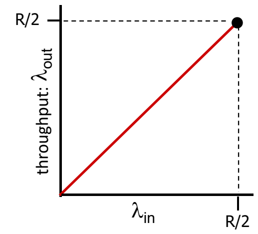
- one router, *finite* buffers
- sender retransmits lost, timed-out packet
  - original data rate: λ_{in}
    - throughput (at receiver): $\lambda_{out} \leq \lambda_{in}$
  - offered load (data rate including *retransmissions*): $\lambda'_{in} \geq \lambda_{in}$

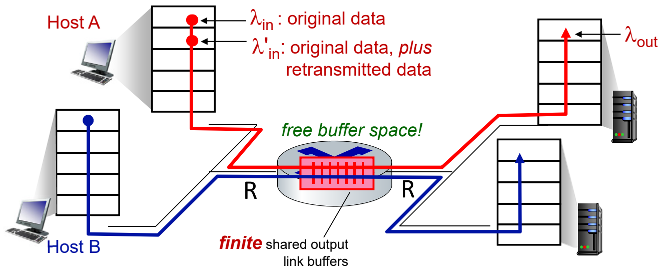


Causes/costs of congestion: scenario 2

Idealization: perfect knowledge

- sender sends only when router buffers available
  - no packet drop at router, no retransmission at sender
  - $\lambda'_{in} = \lambda_{in}$
  - $\lambda_{out} = \lambda'_{in} = \lambda_{in}$

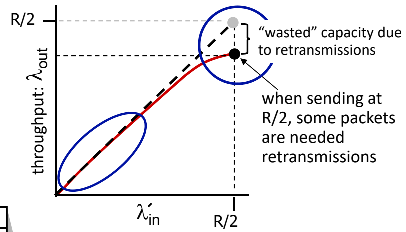


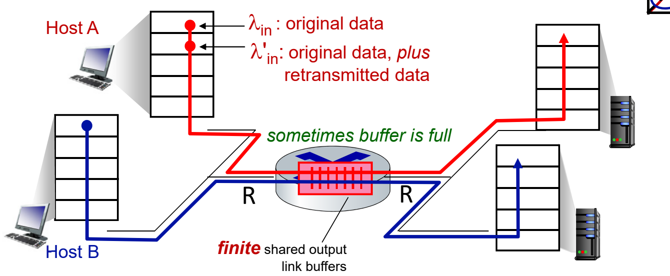


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- a part of packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends the packets *known* to be lost

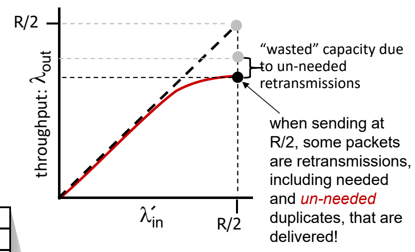


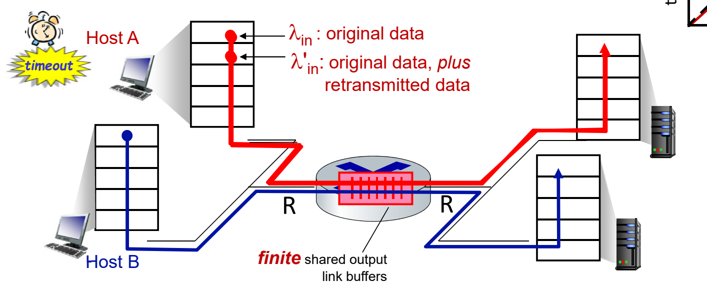


Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

- Besides retransmissions caused by packet drops at router due to buffer overflow,
- sender sometimes can time out prematurely, sending *two* copies, *both* of which are delivered



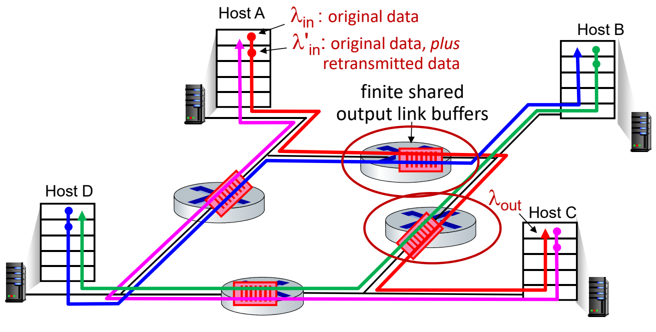


Causes/costs of congestion: scenario 3

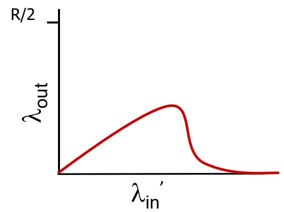
- *four* senders
- *multi-hop* paths
- timeout/retransmit

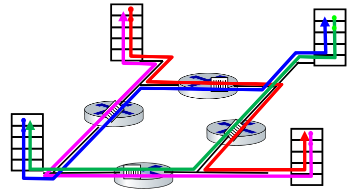
Q: what happens as λ_{in} and λ'_{in} increase ?

A: as red λ'_{in} increases, all blue pkts arriving at upper queue are dropped, blue throughput → 0



Causes/costs of congestion: scenario 3



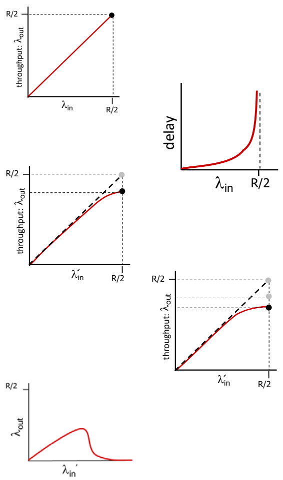


another "cost" of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

Causes/costs of congestion: insights

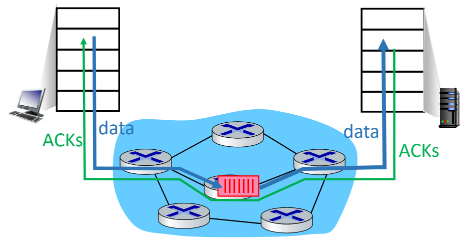
- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- (upstream) transmission capacity / buffering wasted for packets lost in the downstream



Approaches towards congestion control

End-to-end congestion control:

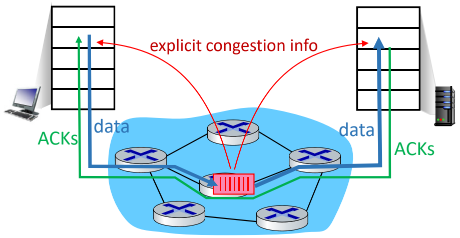
- no explicit feedback from network
- congestion *inferred* from
  - observed loss, overlong delay
- approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

- congested routers provide *direct* feedback to sending/receiving hosts
  - may indicate congestion level or explicitly set sending rate



- IP ECN (explicit congestion notification) & TCP ECE (ECN-Echo)
  - router sets a mark (in IP header) to signal congestion
  - receiver echoes back (in TCP header) to sender
  - sender reduces its transmission rate as for a packet drop