Transport Layer

Anand Baswade

anand@iitbhilai.ac.in

TCP Sender (simplified)

event: data received from application

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **`TimeOutInterval`**

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

TCP Receiver: ACK generation [RFC 5681]

Event at receiver	*TCP receiver action*

Rules for Generating the ACKs

1. When one end sends a data segment to the other end, it must include an ACK. That gives the next sequence number it expects to receive. (Piggyback)

2. The receiver needs to delay sending (until another segment arrives or 500ms) an ACK segment if there is only one outstanding in-order segment. It prevents ACK segments from creating extra traffic.

3. There should not be more than 2 in-order unacknowledged segments at any time. It prevent the unnecessary retransmission

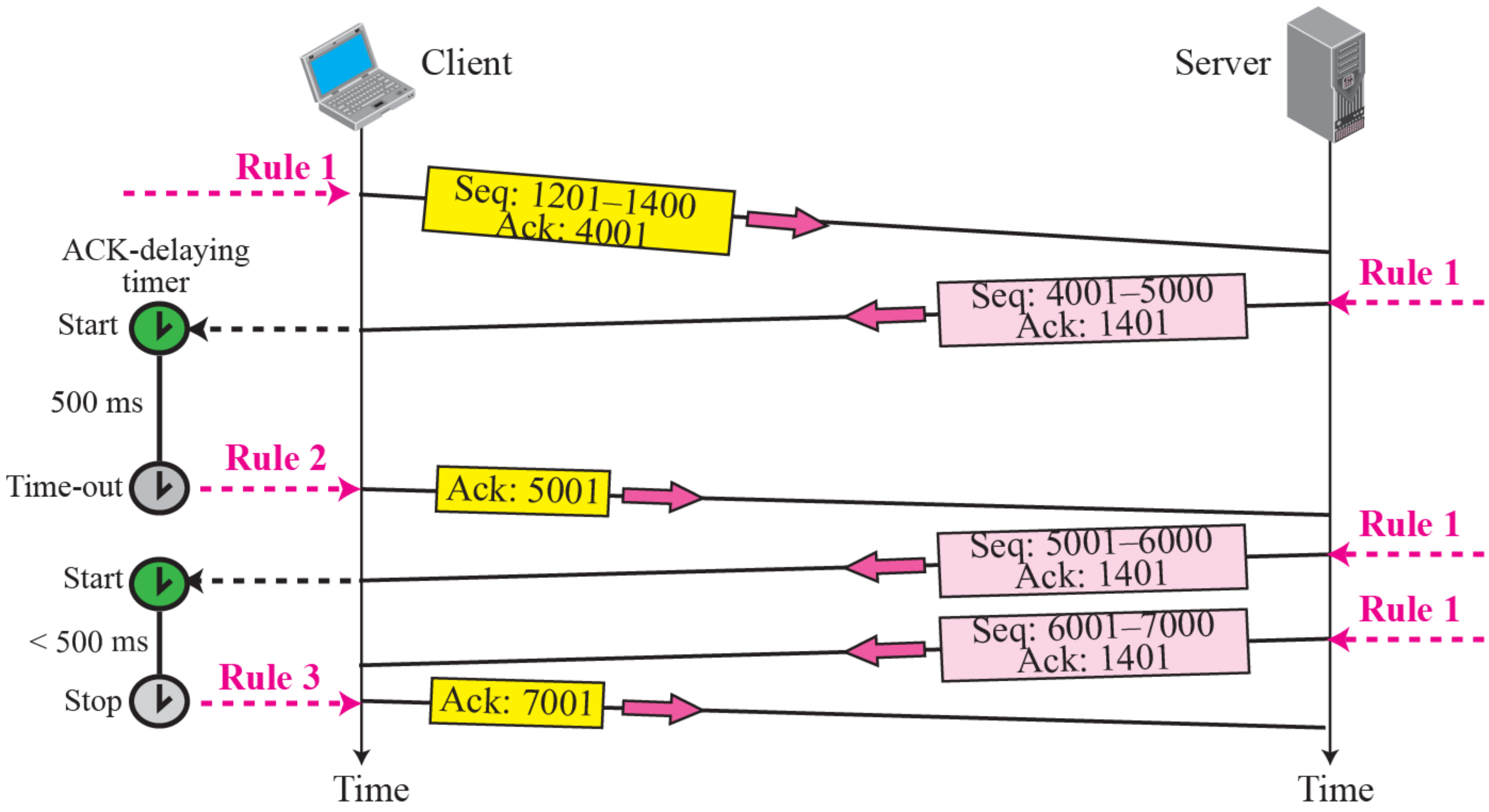
Rules for Generating the ACKs Cont..

4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. (for fast retransmission)

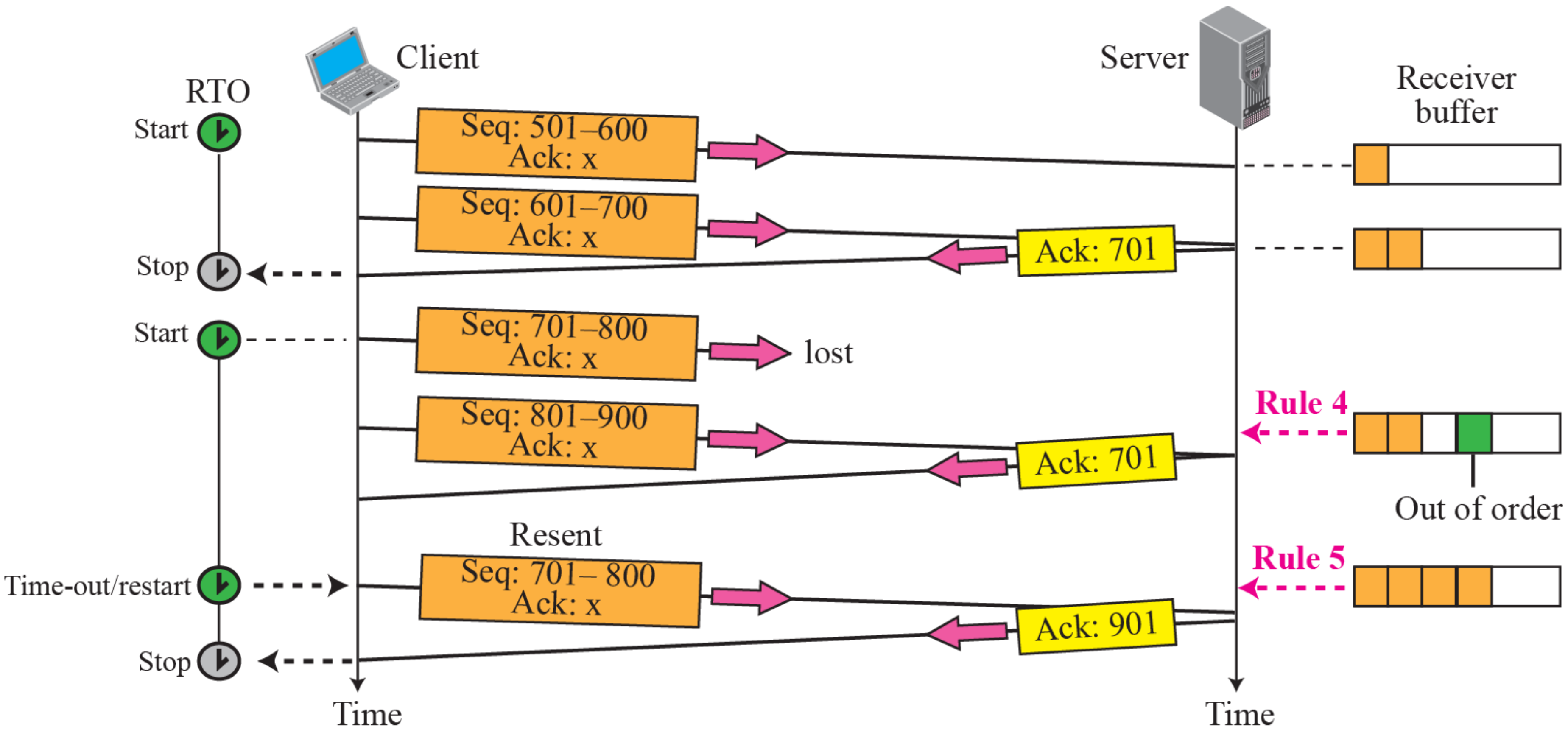
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected.

6. If a duplicate segment arrives, the receiver immediately sends an ACK.

Some Scenarios: Normal operation



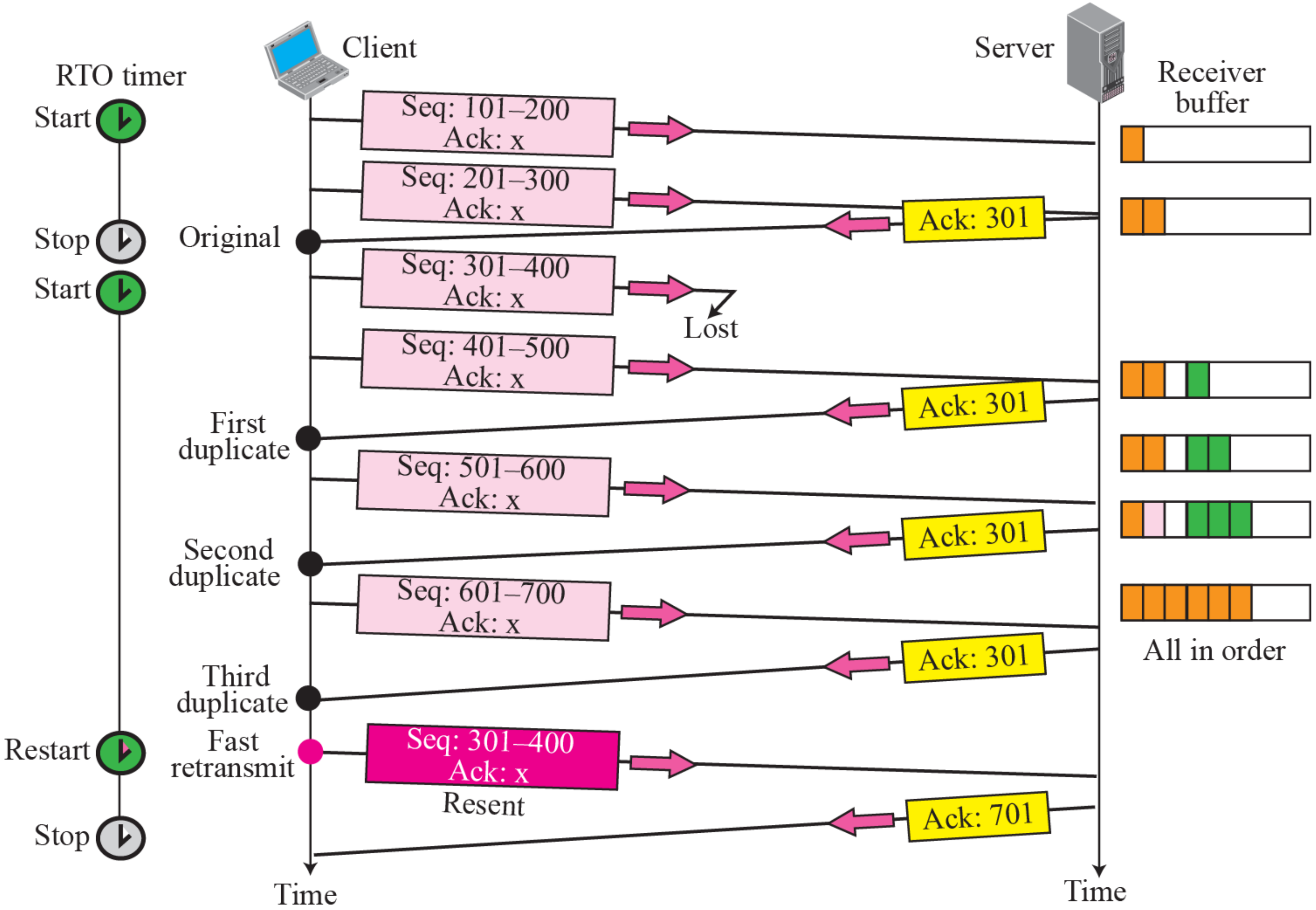
Lost segment



The receiver TCP delivers only ordered data to the process.

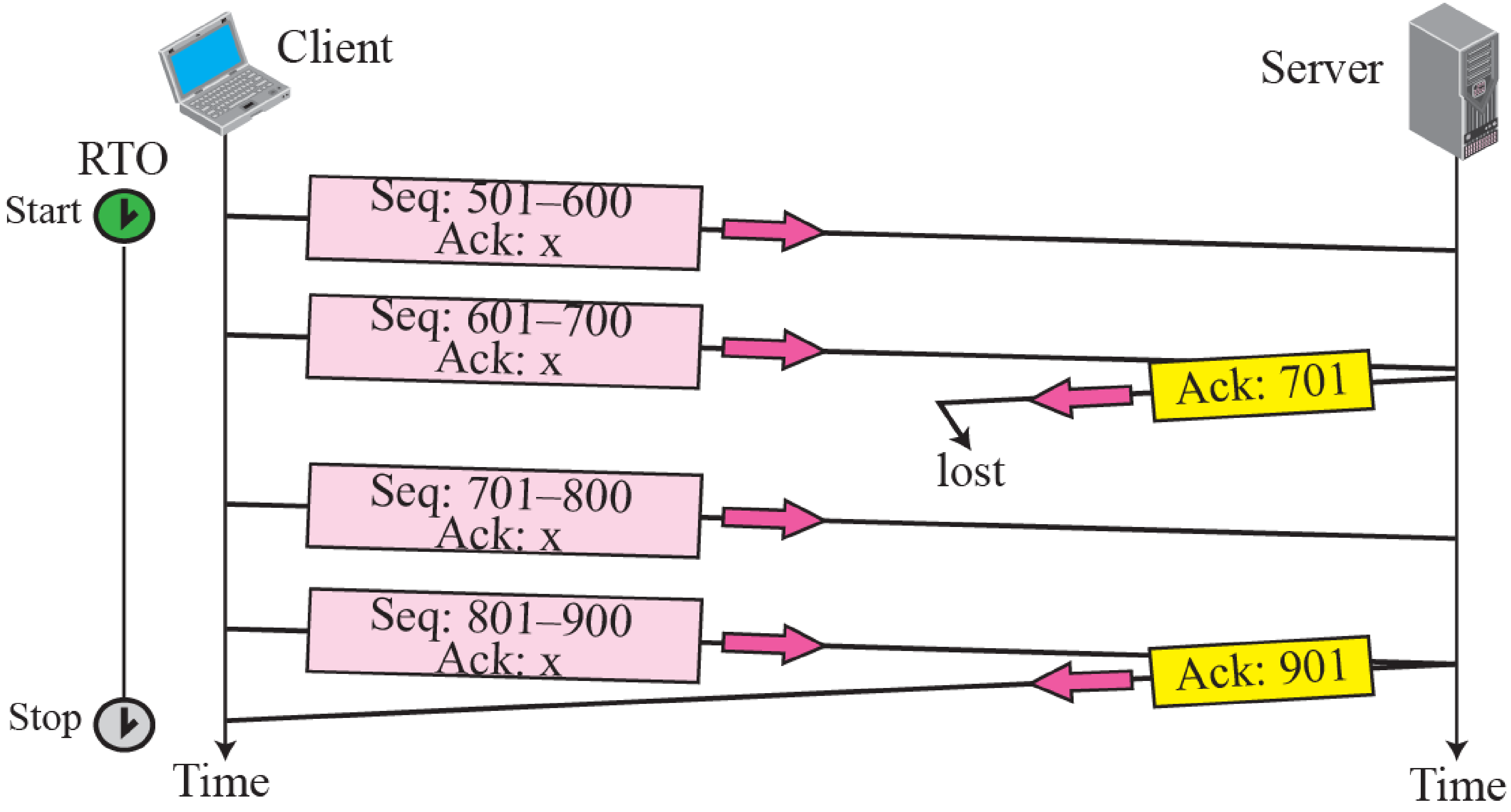
Fast retransmission

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

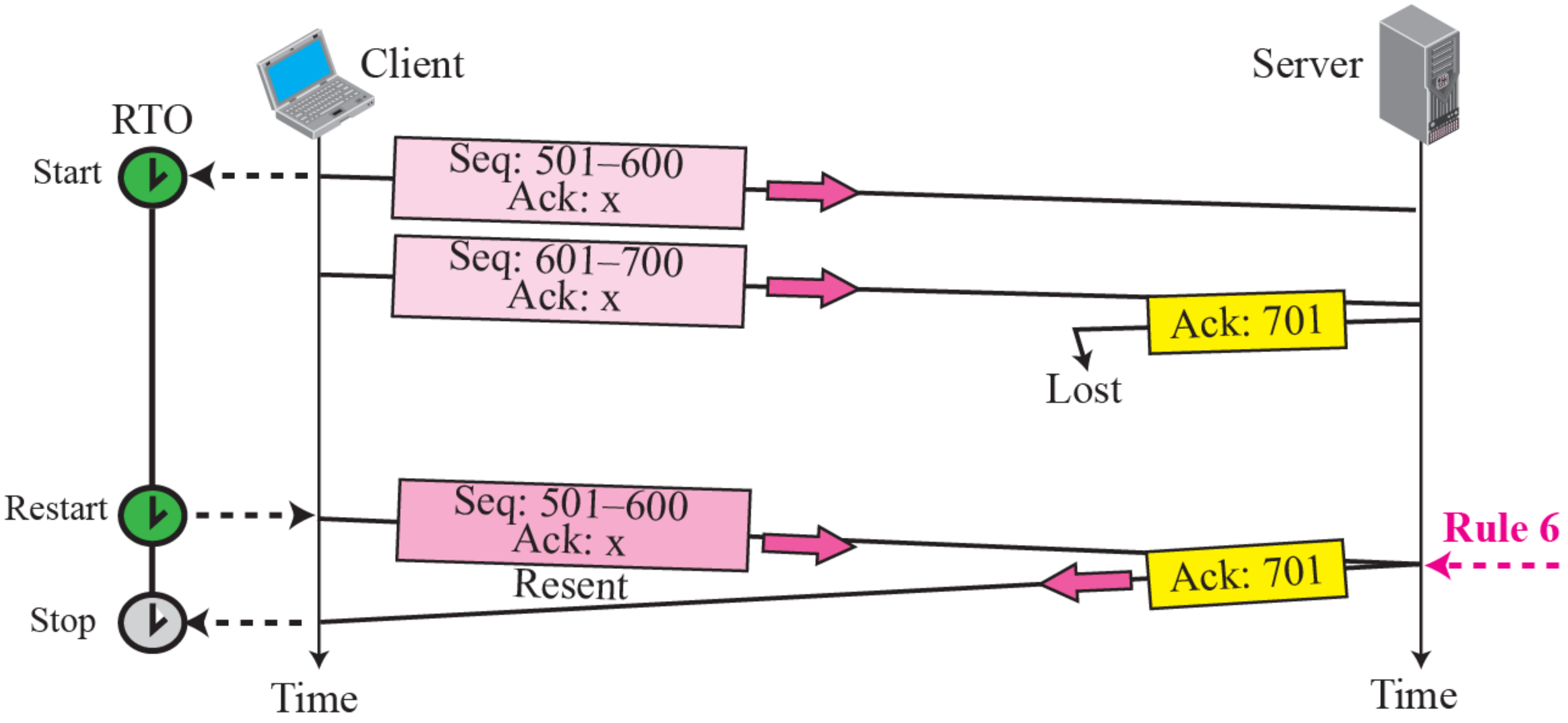


TCP/IP Protocol Suite

Lost acknowledgment



Lost acknowledgment corrected by resending a segment



ACK and Out of Order Handling in TCP

Acknowledgement in TCP – Cumulative acknowledgement

Receiver has received bytes 0, 1, 2, _, 4, 5, 6, 7

- TCP sends a cumulative acknowledgement with ACK number 3, acknowledging everything up to byte 2
- Once 4 is received, a duplicate ACK with ACK number 3 (next expected byte) is forwarded
- After timeout, sender retransmits byte 3
- Once byte 3 is received, it can send another cumulative ACK with ACK number 8 (next expected byte)

TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

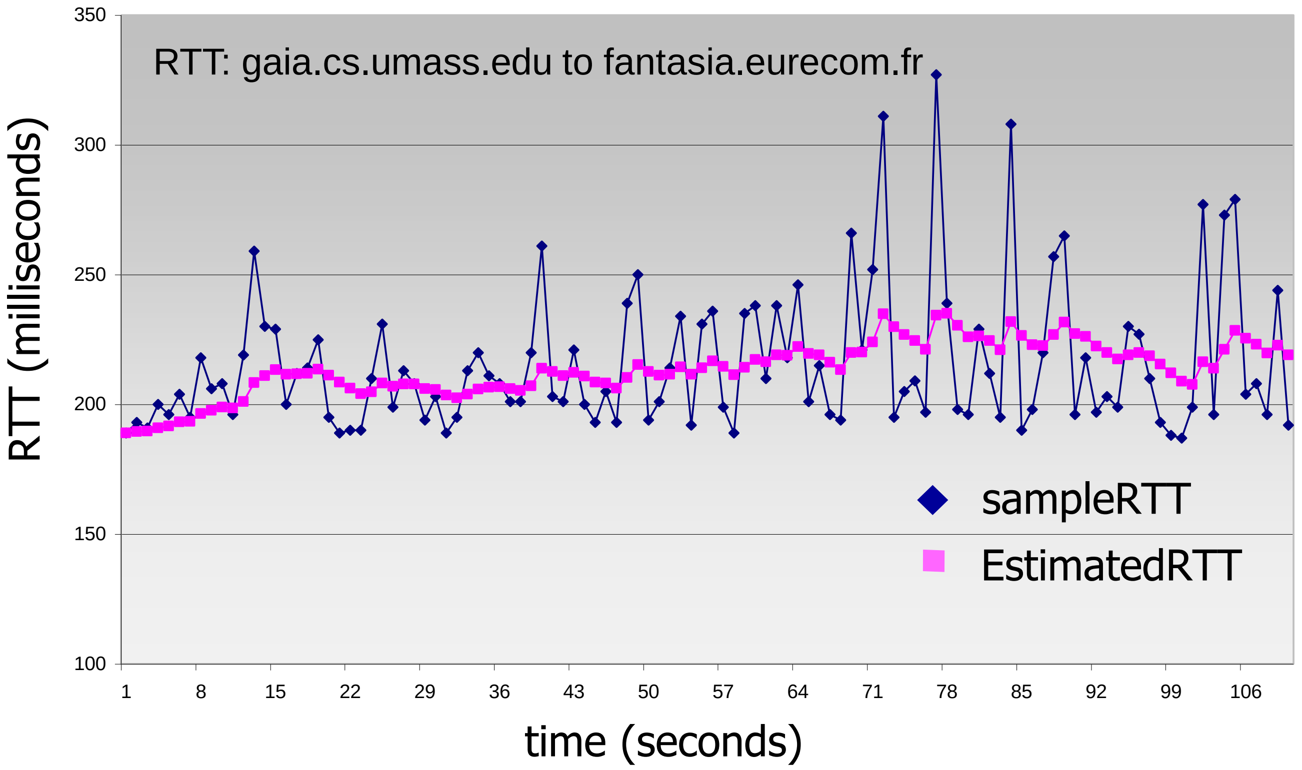
Q: how to estimate RTT?

- `SampleRTT` : measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

TCP round trip time, timeout

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
```

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus “safety margin”
  - large variation in **`EstimatedRTT:`** want a larger safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

“safety margin”

- **`DevRTT`**: EWMA of **`SampleRTT`** deviation from **`EstimatedRTT`**:

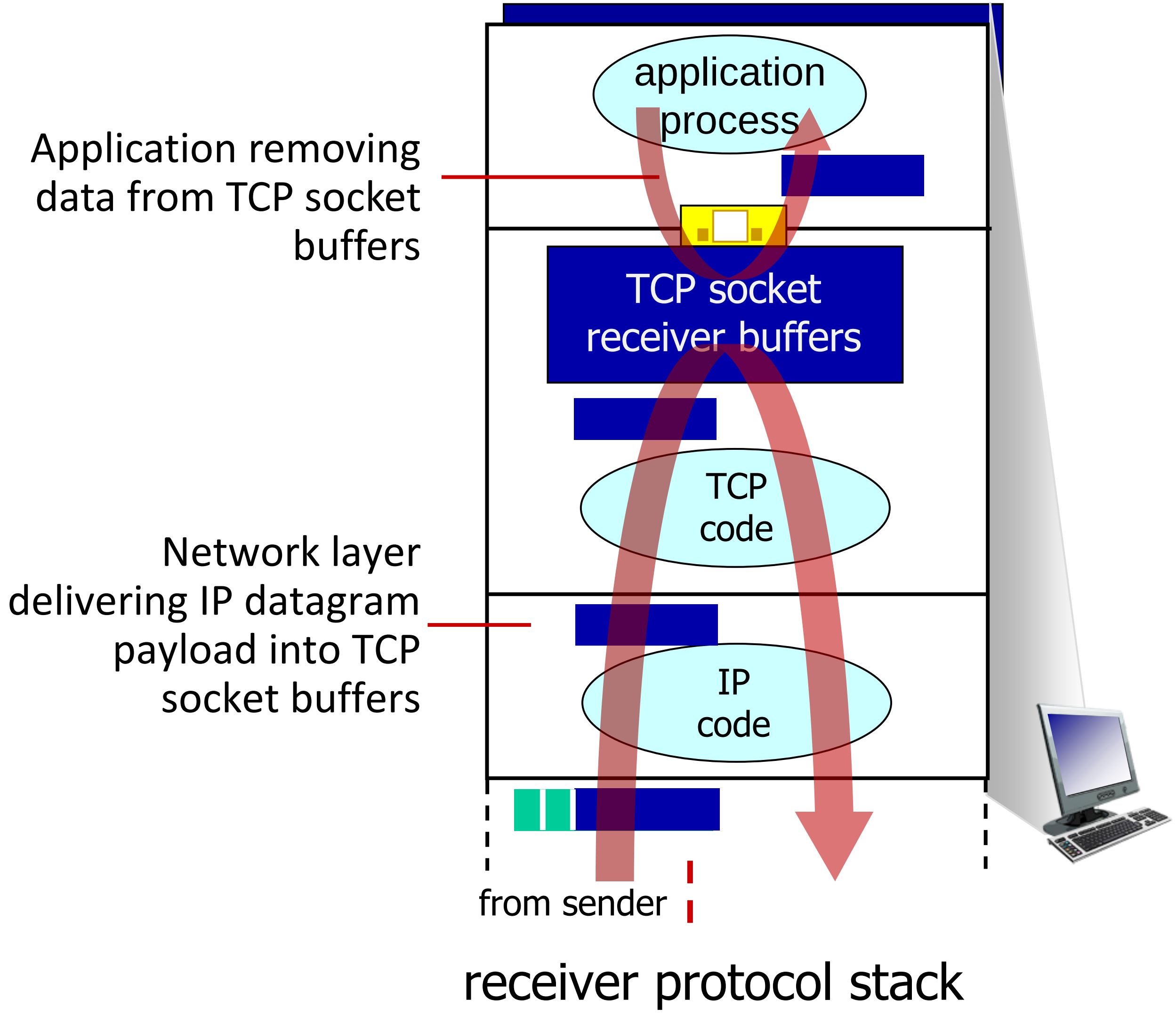
```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

(typically, β = 0.25)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

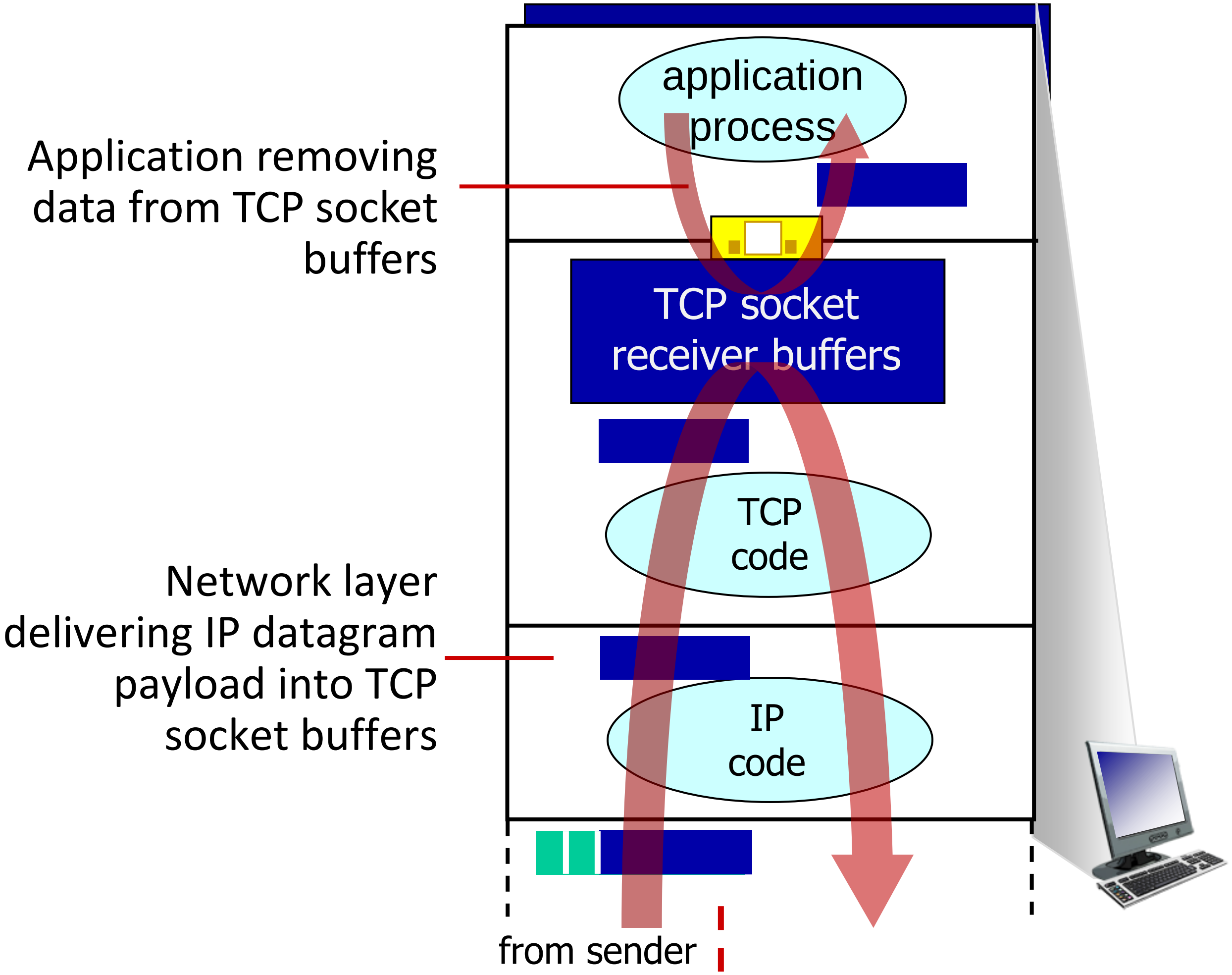
TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



TCP flow control

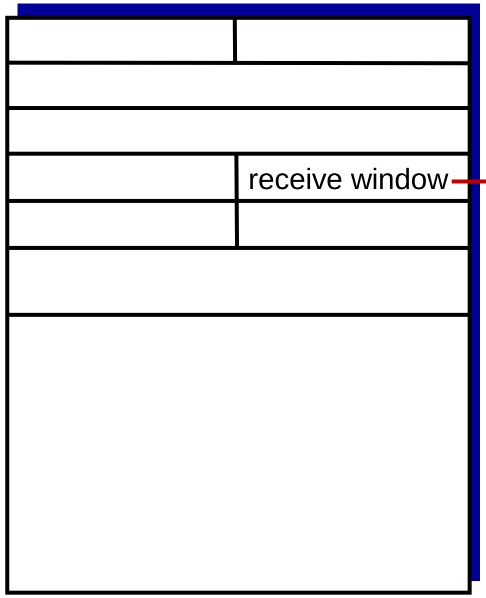
Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



receiver protocol stack

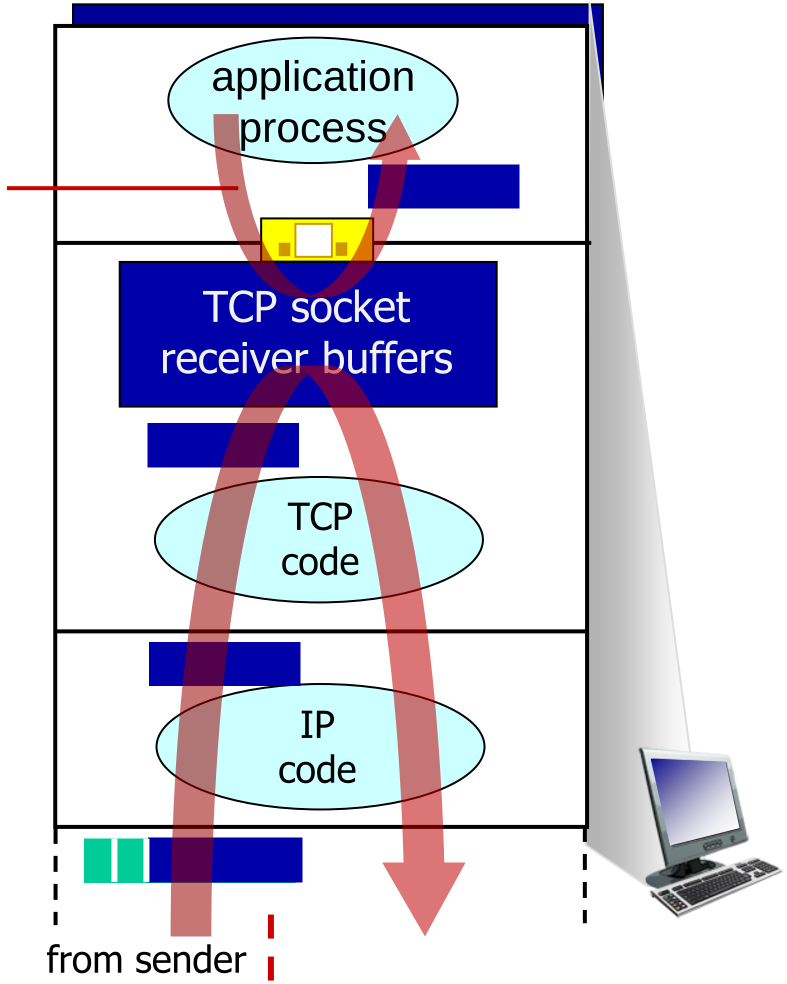
TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



flow control: # bytes receiver willing to accept

Application removing data from TCP socket buffers



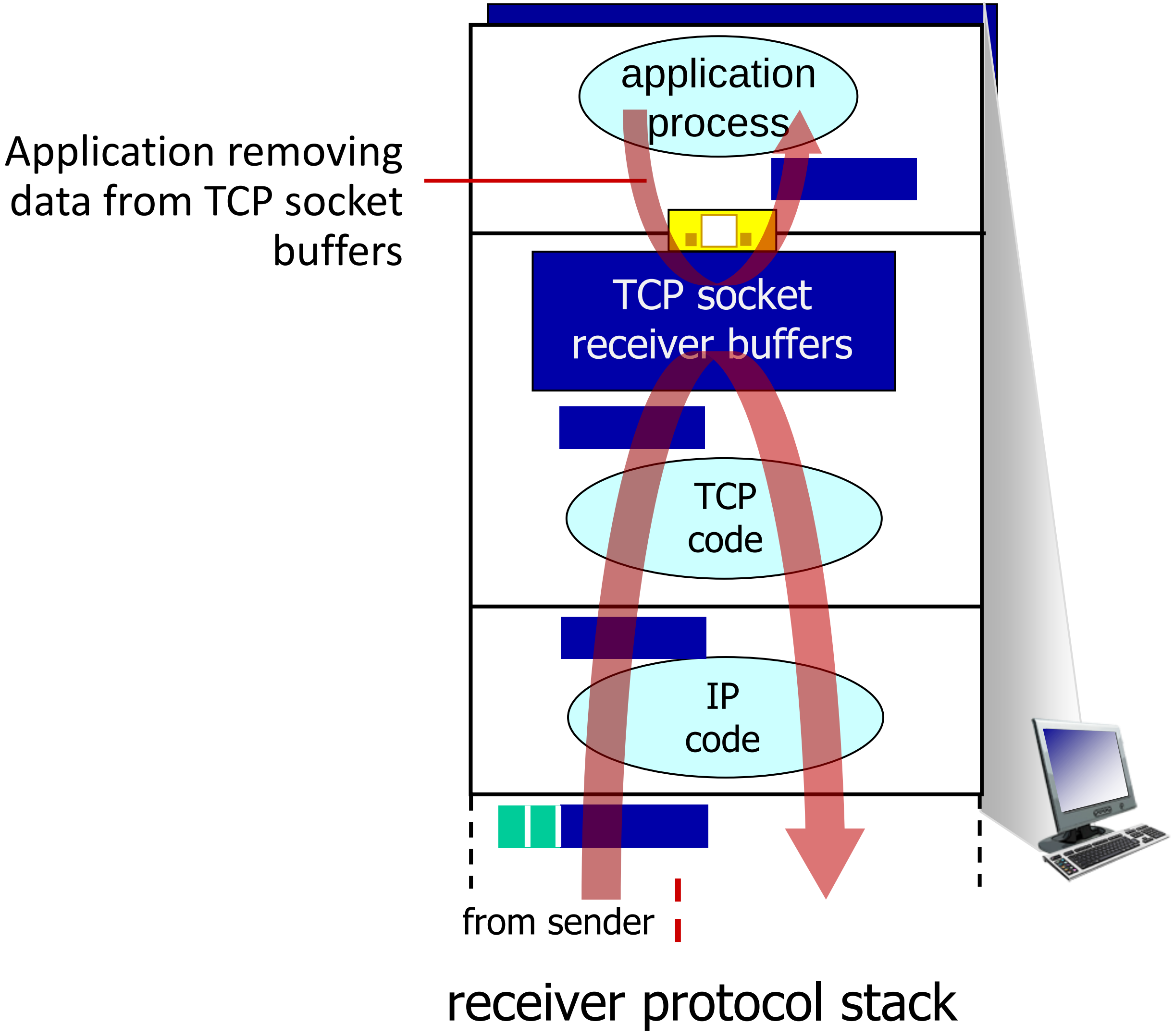
receiver protocol stack

TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

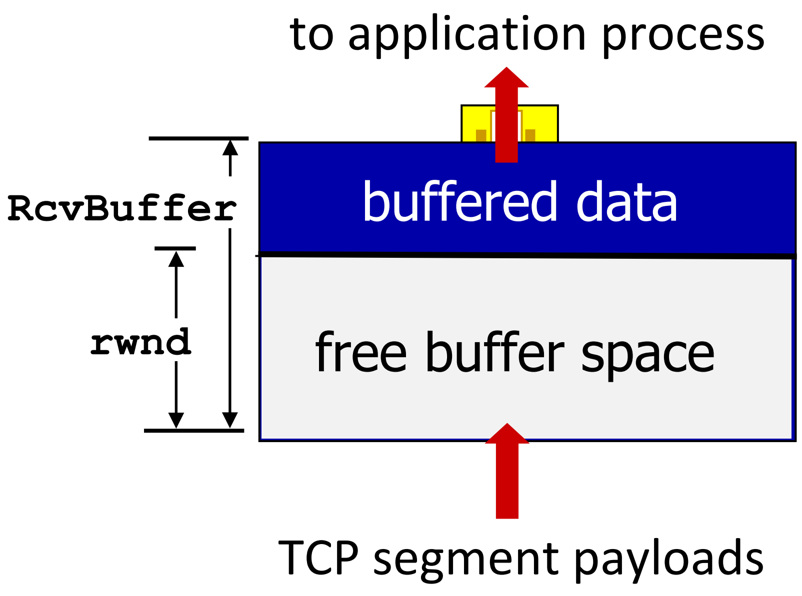
flow control

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



TCP flow control

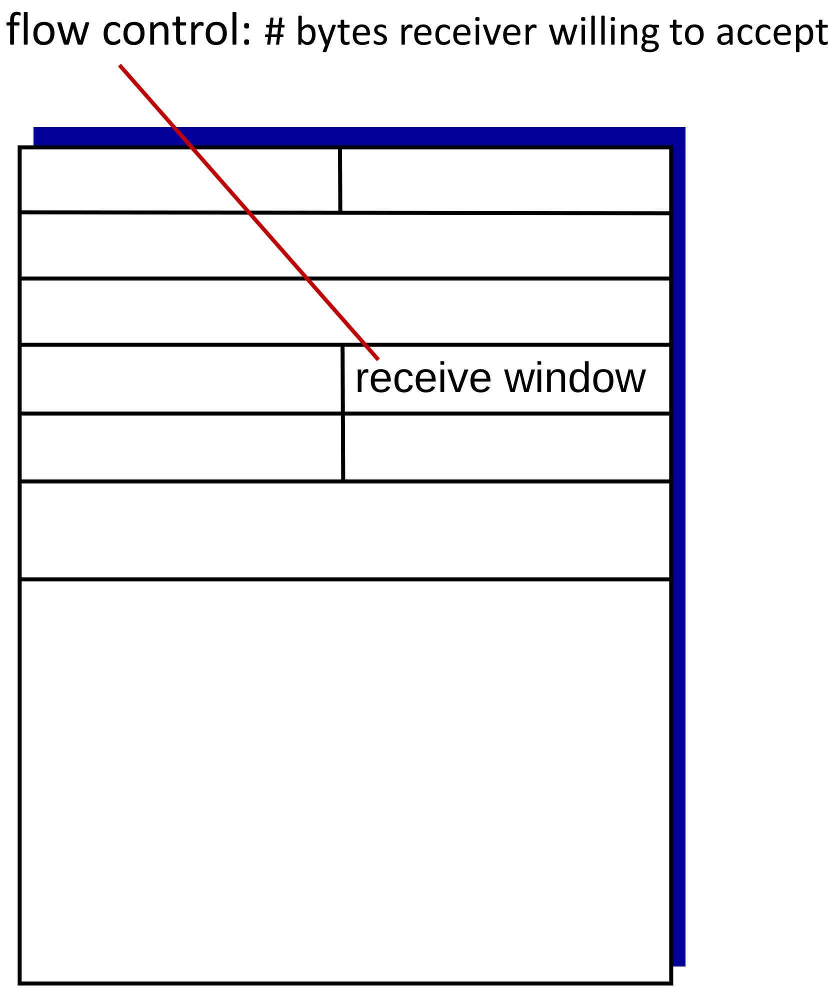
- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- guarantees receive buffer will not overflow



TCP receiver-side buffering

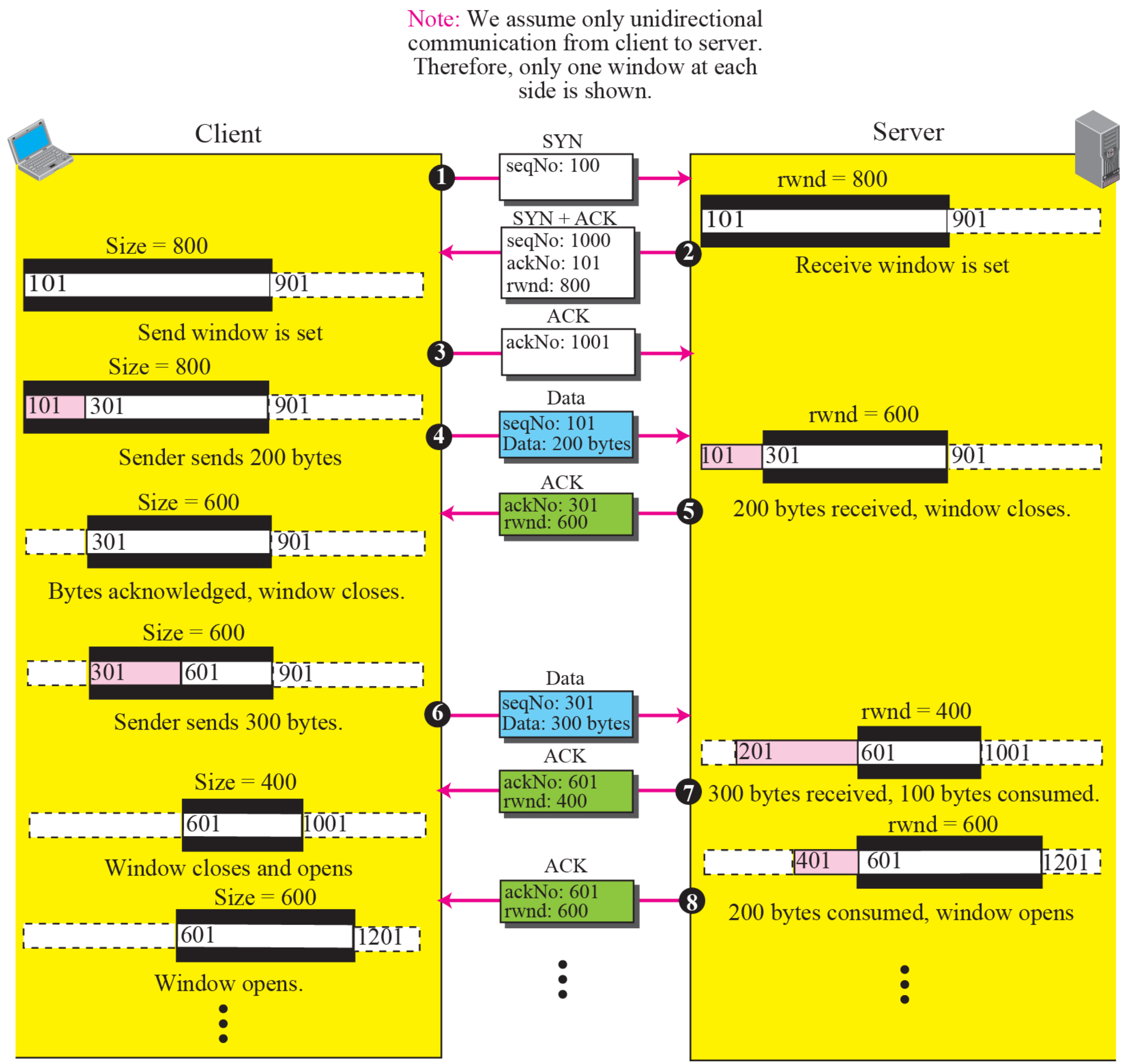
TCP flow control

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



TCP segment format

An example of flow control



TCP/IP Protocol Suite

Principles of congestion control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)

- different from flow control!
- a top-10 problem!

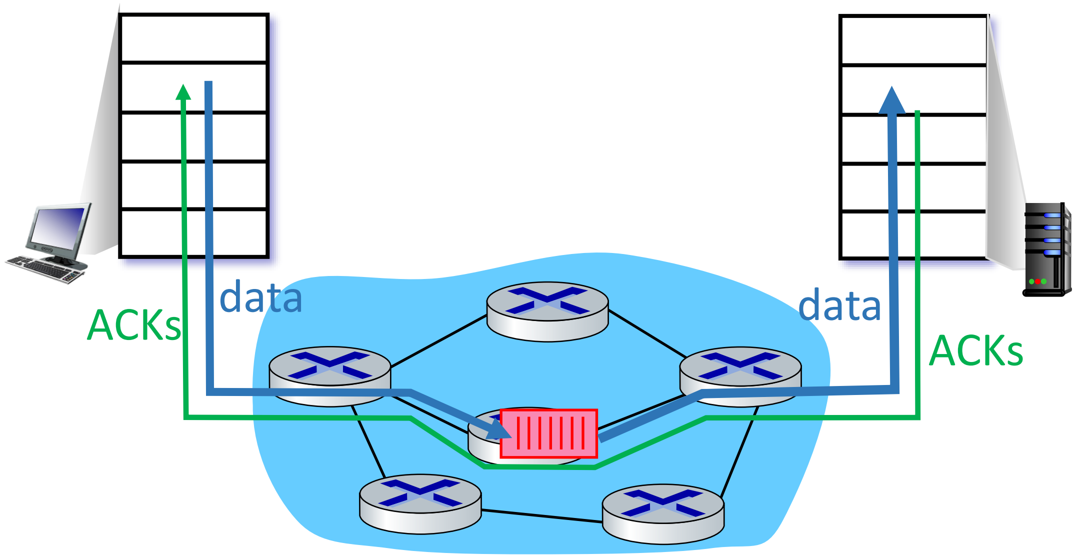
congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

Approaches towards congestion control

End-end congestion control:

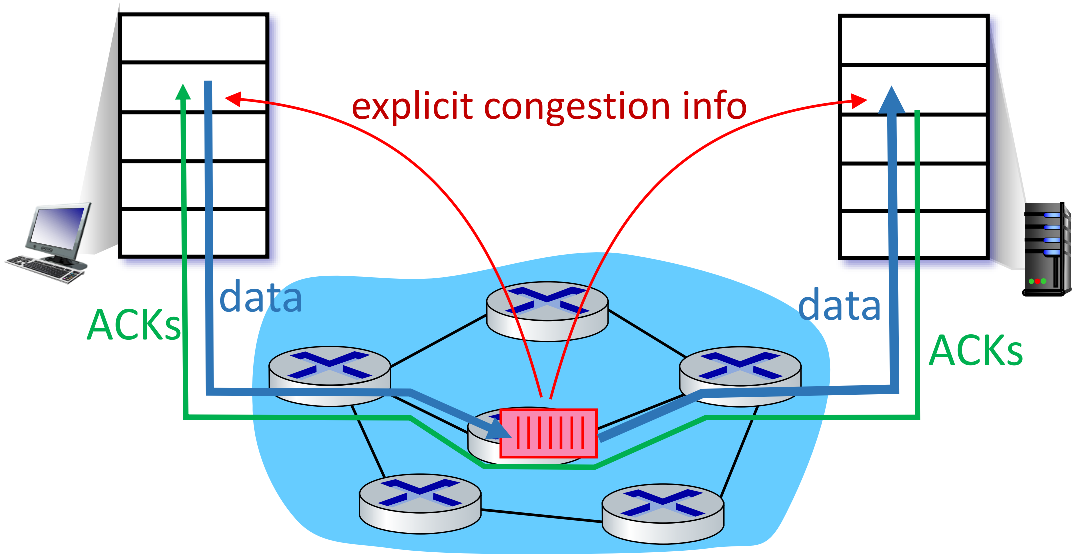
- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

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



- TCP ECN (Explicit Congestion Notification), ATM (Asynchronous Transfer Mode)

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

TCP: Triggering congestion control

- Two ways to trigger a congestion notification in TCP – (1) RTO, (2) Duplicate ACK
- RTO: A sure indication of congestion, however time consuming
- Duplicate ACK: Receiver sends a duplicate ACK when it receives out of order segment
  - A loose way of indicating congestion
  - TCP arbitrarily assumes that THREE duplicate ACKs (DUPACKs) imply that a packet has been lost – triggers congestion control mechanism
  - Retransmit the lost packet and trigger congestion control

TCP congestion control: AIMD

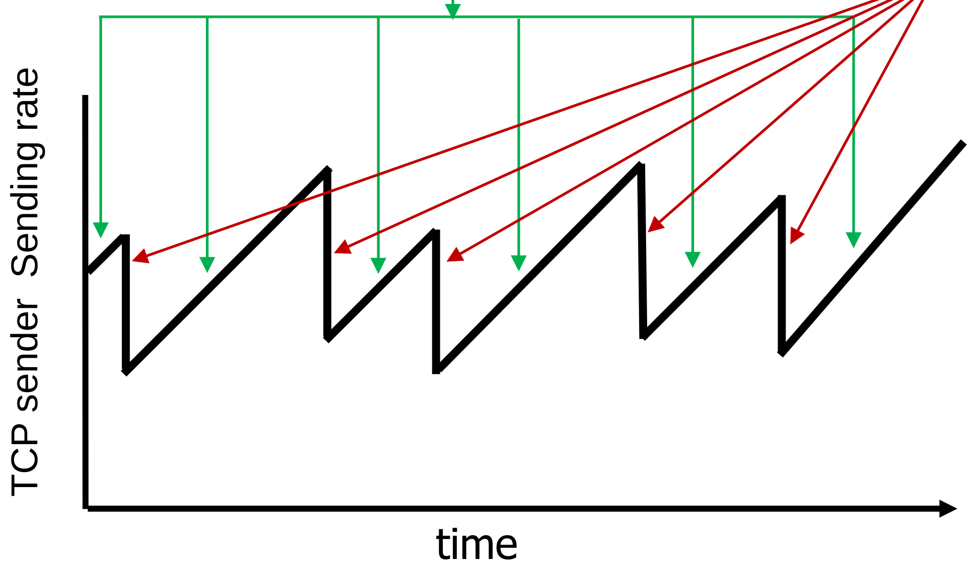
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event.

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



- Chiu and Jain (1989): Let *w(t)* be the sending rate. *a (a > 0)* is the additive increase factor, and *b (0<b<1)* is the multiplicative decrease factor

$$w(t+1) = \begin{cases} w(t) + a & \text{if congestion is not detected} \\ w(t) \times b & \text{if congestion is detected} \end{cases}$$