Transport and Congestion Control in the Internet

Chadi BARAKAT

INRIA Sophia Antipolis, France
DIANA group

Email: Chadi.Barakat@inria.fr
WEB: http://planete.inria.fr/chadi/

References - Sources

- Slides of Prof. Jim Kurose, Umass Univ at Amherst.
- "Computer Networking book" by Jim Kurose and Keith Ross
- Computer Networks, 4th edition, Andrew S. Tanenbaum, 2003
  - The reference introduction to computer networks

Our goals

- Understand principles behind transport layer services:
  - Multiplexing and demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- Learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control
  - SCTP stream control transport protocol
  - DCCP Datagram Congestion Control Protocol

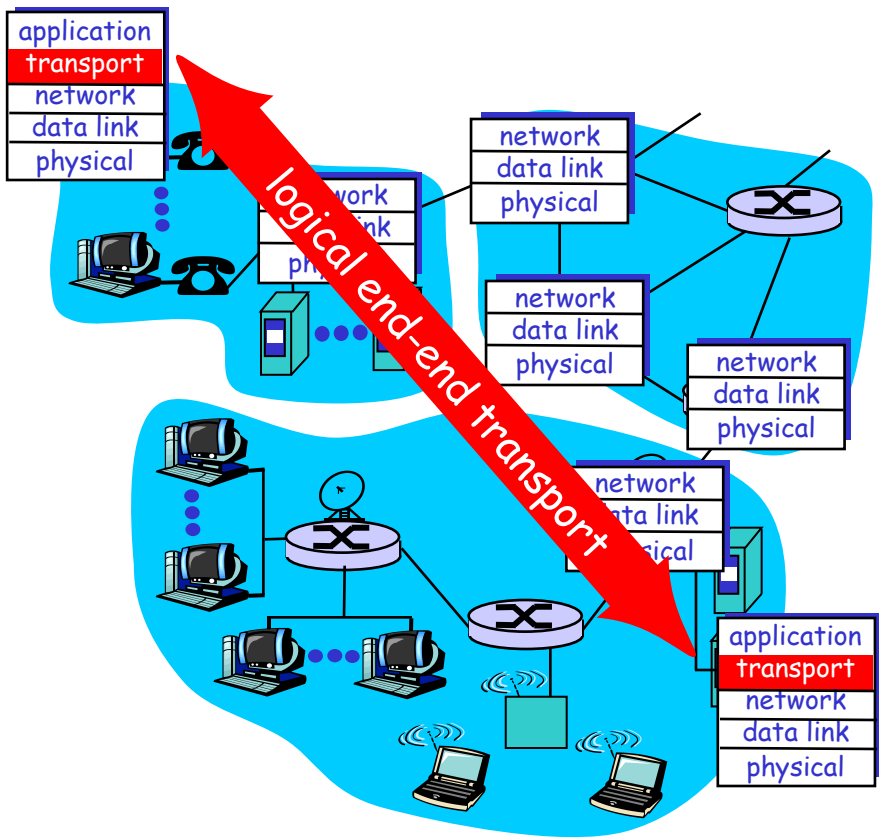
Highlight the evolution of this layer and the challenges it faces

Outline

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

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - receiver side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP
  - Recently DCCP and SCTP

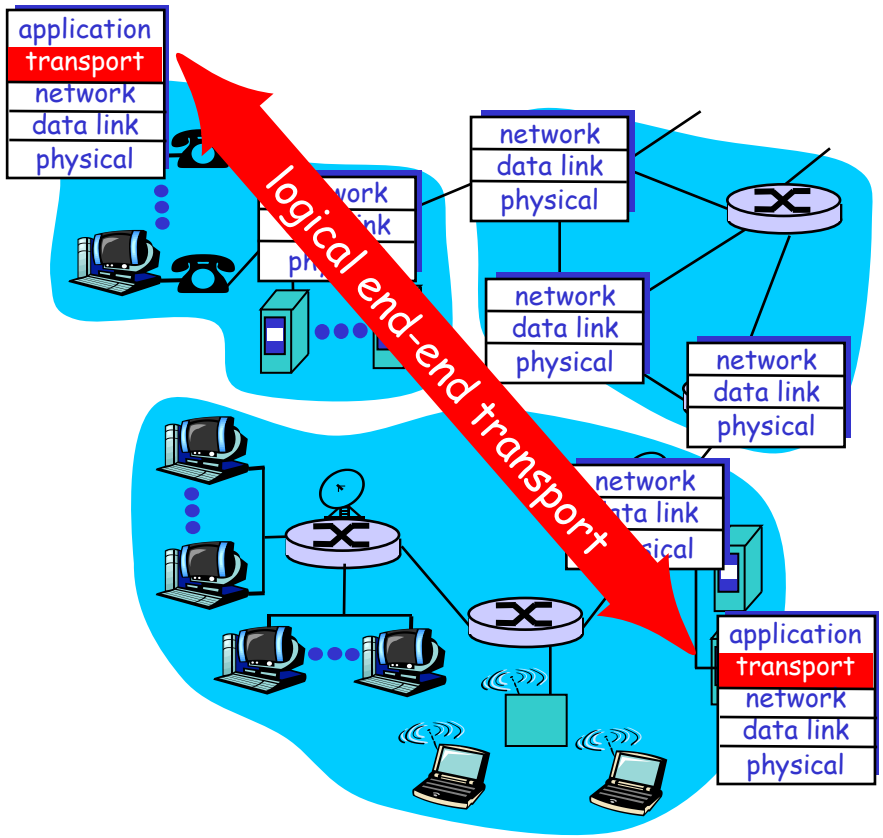


Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhanced, network layer services

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - No functionality extension of "best-effort" IP
- services not available in the Internet
  - delay guarantees
  - bandwidth guarantees



Outline

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

Socket

- A socket is a software abstraction, designed to provide a standard application programming interface (API) for sending and receiving data across a computer network
  - Specific method/function with many options

Why do we need multiplexing

- Consider that on your host you do Web browsing + file transfer using FTP + a remote session using ssh
  - How do you send data from these very different applications on the network?
  - How do you know to which application the data received from the network should be passed?
- The solution is
  - Multiplexing: gather data from applications and send them to the network
  - Demultiplexing: receive data from the network and pass them to the applications

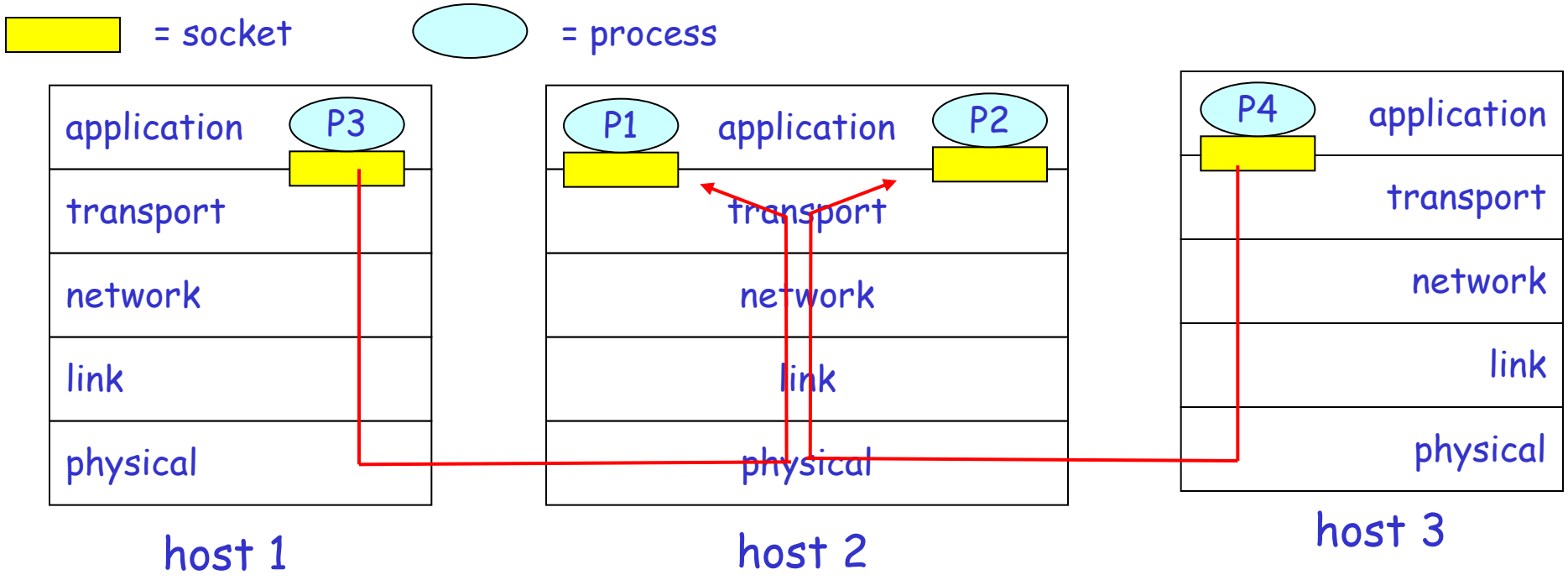
Multiplexing/demultiplexing

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

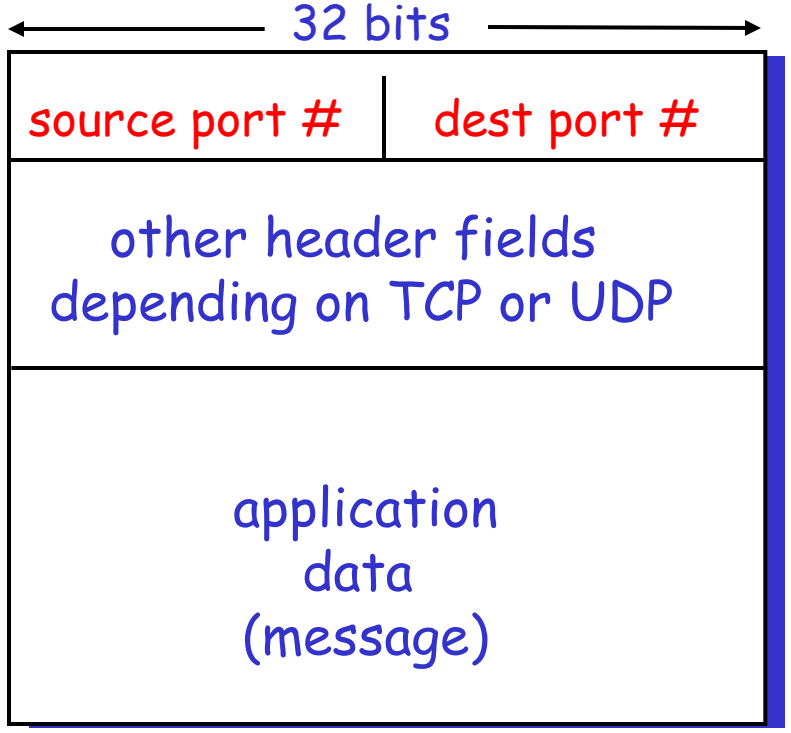
Demultiplexing at rcv host:

delivering received segments to correct socket



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries only 1 transport-layer segment
  - each segment corresponds to a single socket
  - each segment has source and destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP generic segment format

Port Numbers

- For both TCP and UDP they are on 16 bits
  - From 0 to 65535
- Ports from 0 to 1023 are reserved ports
  - Called *well known port numbers*
  - *80: HTTP*
  - *20, 21: FTP*
  - *110: POP3*
  - *143: IMAPv2*

UDP Connectionless demultiplexing

- Create sockets with port numbers (example in java):

```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

- UDP socket at destination identified by two-tuple:

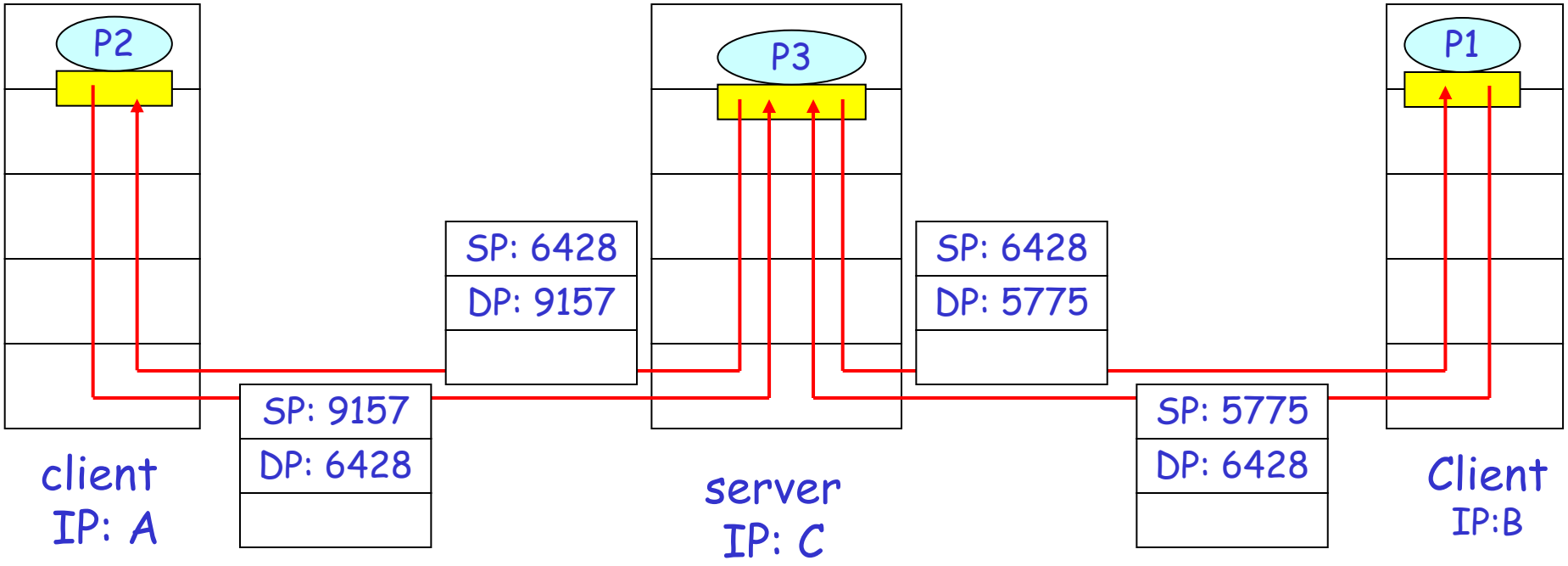
(dest IP address, dest port number)

- UDP socket at source only created with source @ and source port number

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with same (dest IP, dest port #) is directed to same socket
  - The (src IP, src port #) is not used to identify the UDP socket

UDP Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

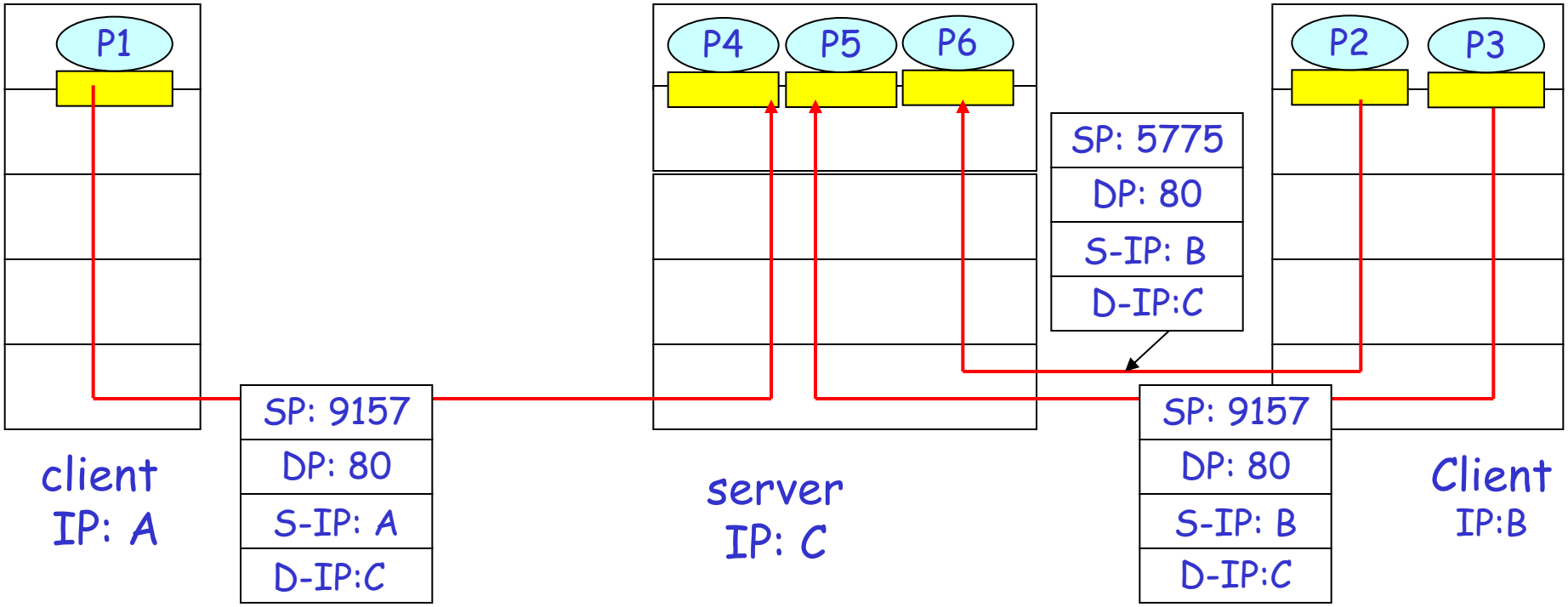


Source Port (SP) provides "return address" at the socket level

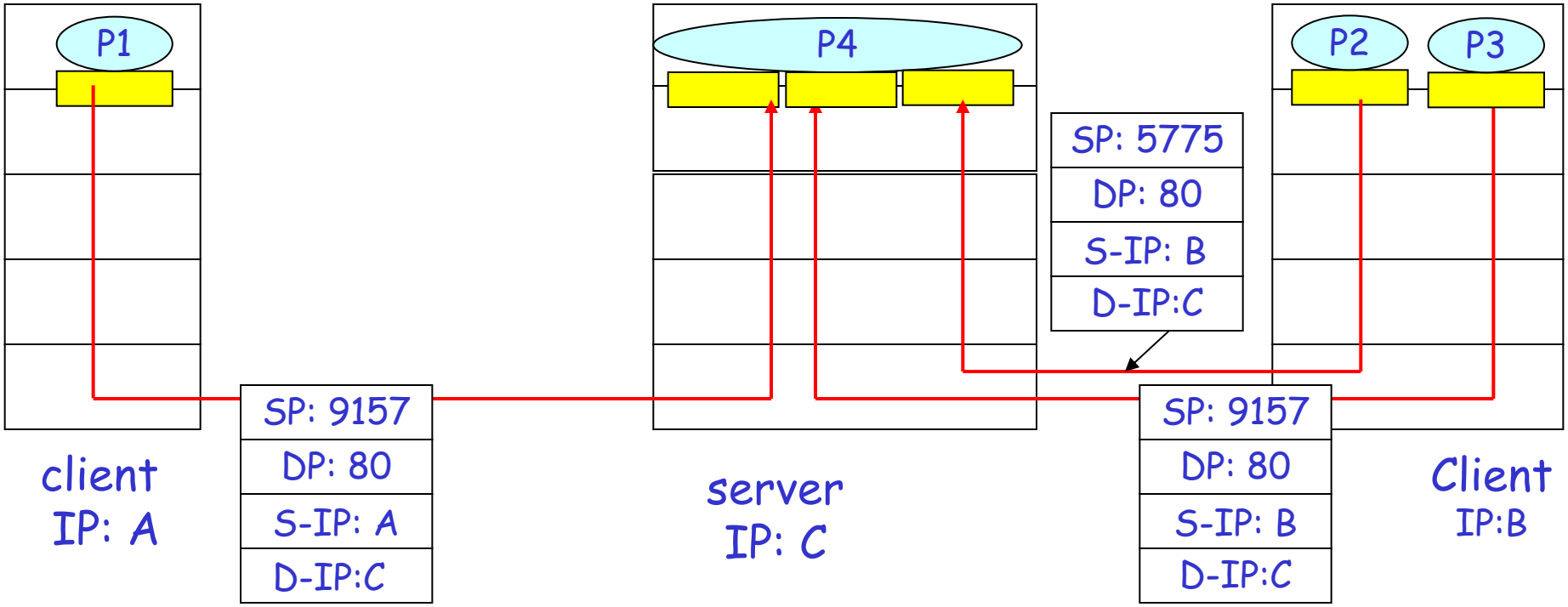
TCP Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- receiving host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client (all uses same port number e.g. 80)
  - non-persistent HTTP will have different socket for each request

TCP Connection-oriented demux (cont)



TCP Connection-oriented demux: Threaded Web Server



Part 2 outline

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

UDP: User Datagram Protocol [RFC 768]

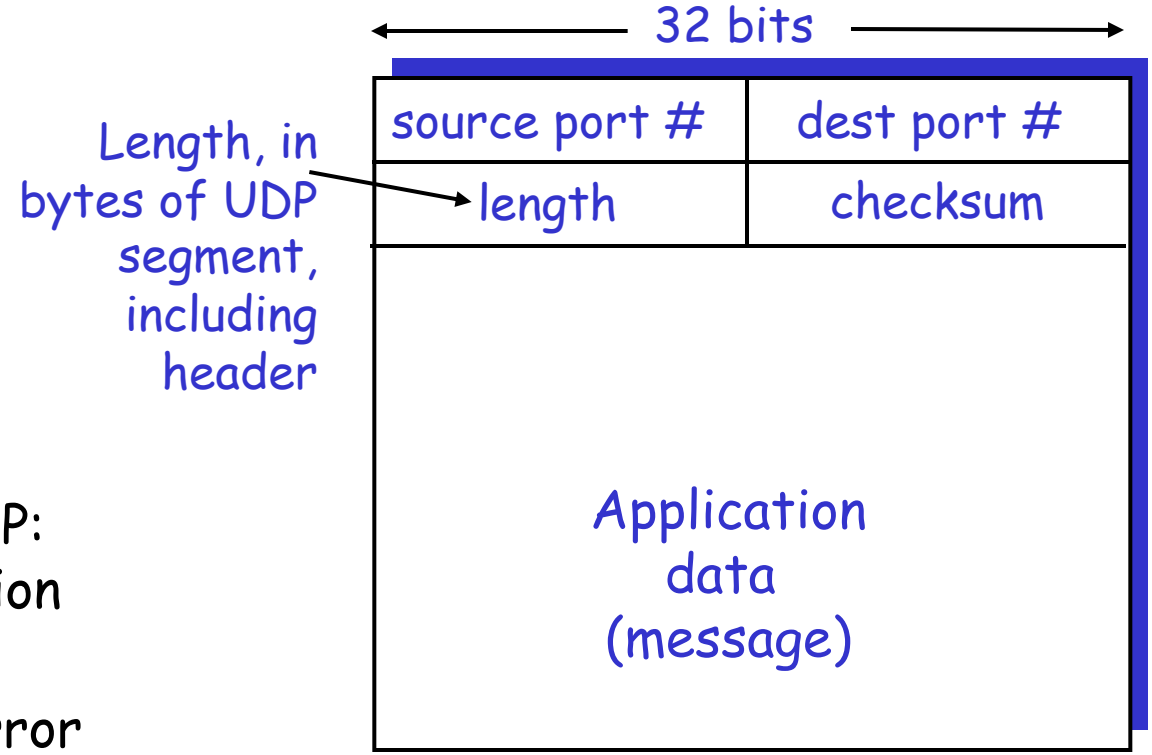
- Simplest Internet transport protocol
  - Multiplexing/demultiplexing
  - Basic error checking
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired
- Fine control on what is sent on the network (no automatic retransmissions)

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected.

Internet Checksum Example

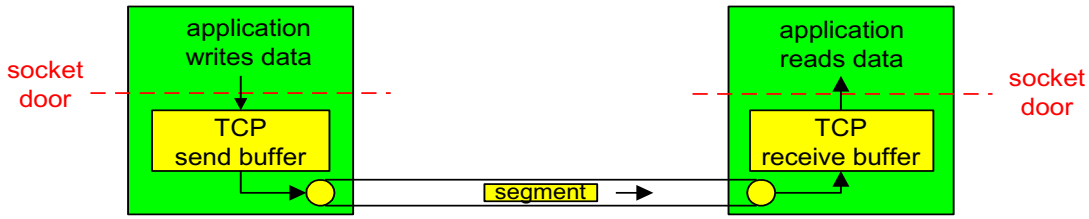
- Note
  - When adding numbers, a reminder from the most significant bit needs to be added to the result. The checksum is the 1 complement of the final sum.
- Example: add two 16-bit integers

Outline

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

TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

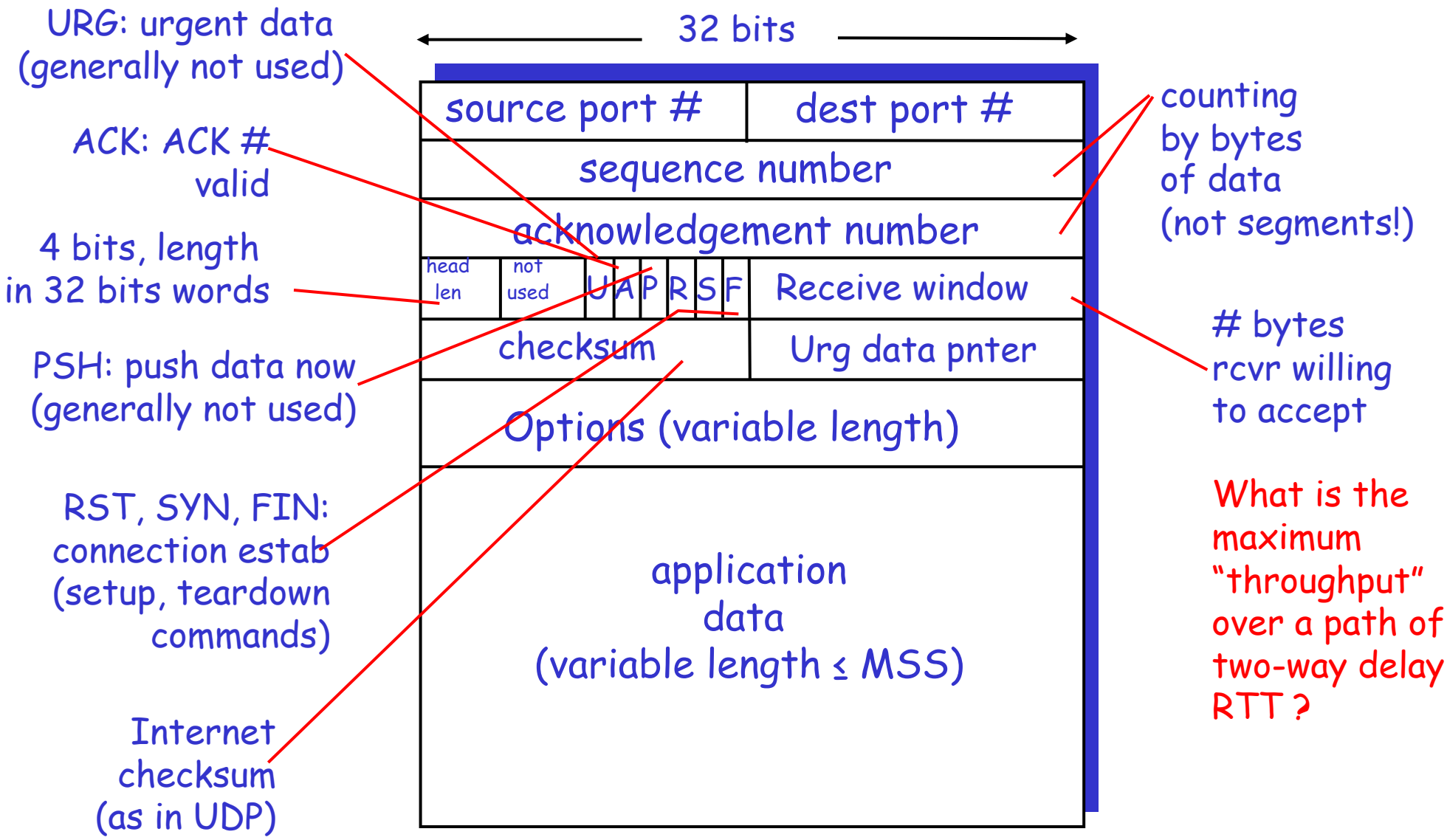
- point-to-point:
  - one sender, one receiver
  - No multicast support
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - Can send several segments back-to-back
  - TCP congestion and flow control set window size, i.e., number of segments that can be sent without being acknowledged
- *send & receive buffers*



TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size (MSS=maximum TCP payload size, misleading term. This is not the full TCP segment including header)
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
  - Three-way handshake
  - No state in the routers, only on the sender and receiver
- flow controlled:
  - sender will not overwhelm receiver
- Congestion control:
  - Sender will not overwhelm network

TCP segment structure



TCP segment structure

- Source port (16 bits)
- Destination port (16 bits)
- Sequence number (32 bits)
  - In bytes
  - What is the maximum throughput (2 minutes segment lifetime) ?
- ACK number (32 bits)
  - In bytes
  - Cumulative ACK: acknowledge all the bytes up to the ACK number (minus 1)
- Header length (4 bits)
  - Count header length in 32 bits (4 bytes) words
  - Maximum header length $2^4 * 32 = 512$ bits

TCP segment structure

- Flag fields (6 bits, 1 bit per flag)
  - ACK bit set: the value contained in the ACK field is valid. Allows piggybacked ACK
  - RST/SYN/FIN: used for connection establishment and teardown (discussed later)
  - PSH bit set: receiver should pass data to the upper layer immediately
  - URG bit set: specify that this data is urgent. The location of the last byte of this urgent data is specified by the 16 bits field *Urgent Data Pointer.* In practice the URG pointer is not used
- Urgent Data Pointer (16 bits): see above the URG flag

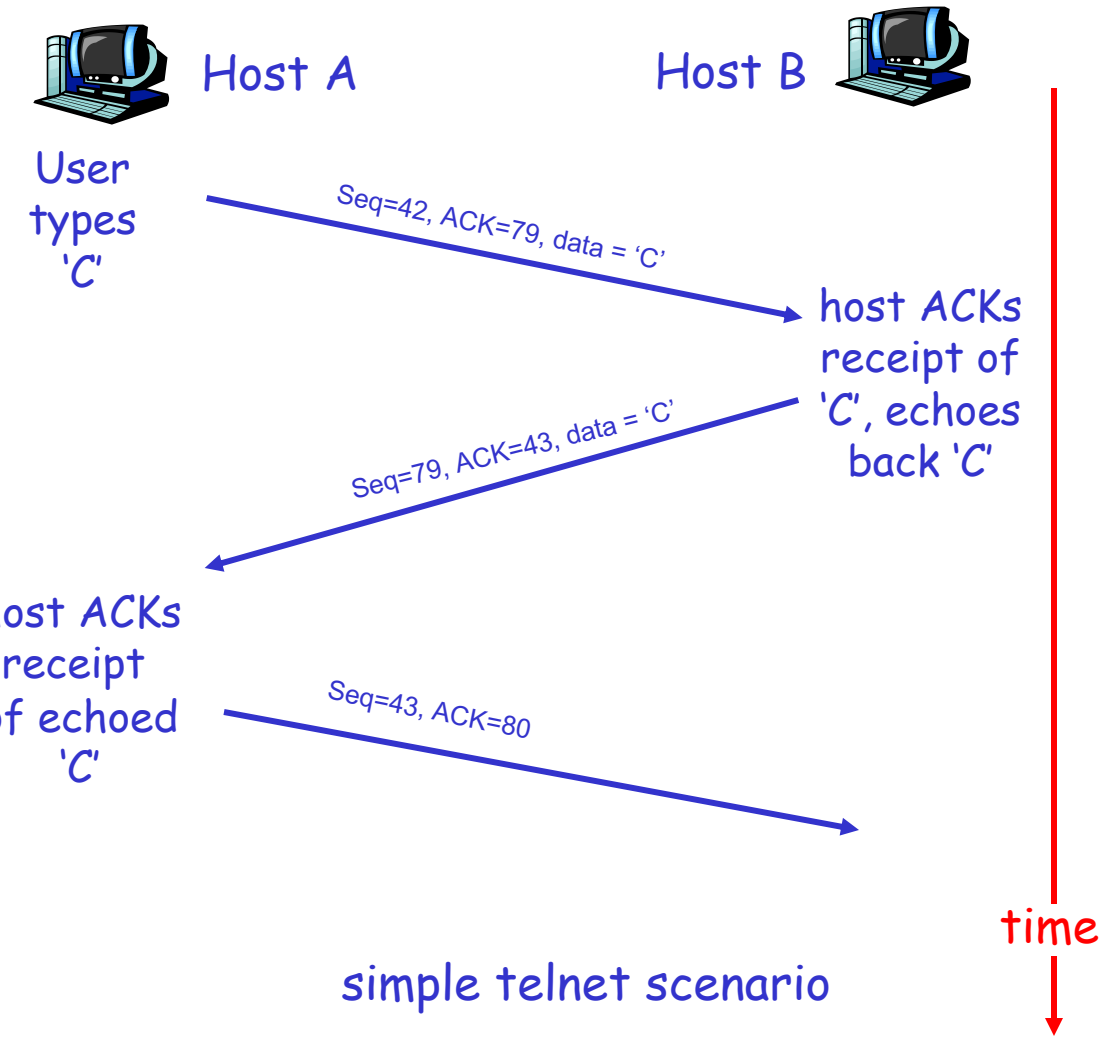
TCP seq. #'s and ACKs

Seq. #'s:

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

ACKs:

- Seq # of next byte expected from other side
- Cumulative ACK
  - Received all bytes up to (seq # -1)
- Piggybacked
  - Ack info sent in a segment



simple telnet scenario

Out of Order Segments

- How receiver handles out-of-order segments?
  - TCP spec doesn't say, - up to implementor
- Two approaches possible
  - Discard at the receiver all the out of order segments
    - Not efficient because the segments are correctly received, but needs to be retransmitted simply due to an out of order reception
  - Keep the out of order segments and wait for the missing segments to deliver the data to the application
    - Approach taken in practice in TCP

Outline

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

TCP reliable data transfer

- TCP creates a reliable service on top of IP's unreliable service
- Pipelined segments
  - Multiple transmitted segments not yet ACKed at the same time
- Cumulative positive ACKs
- TCP uses single retransmission timer
  - One timer for the oldest unacknowledged segment
  - Multiple retransmission timers (1 per unacknowledged segment) is too expensive

- Retransmissions are triggered by:
  - timeout events
  - duplicate ACKs

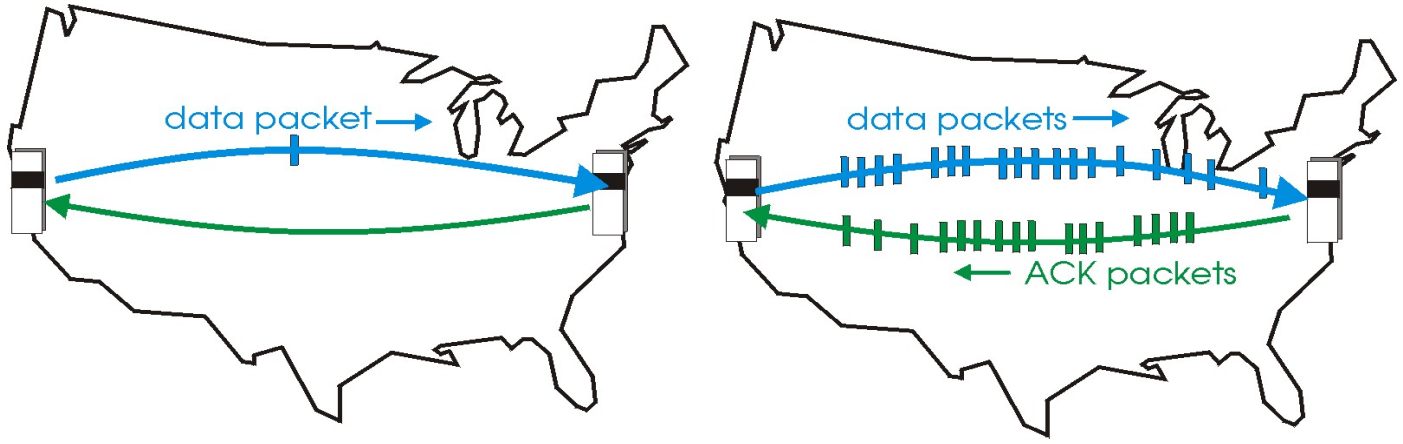
Let's first consider a simplified TCP sender:

- ignore duplicate ACKs
- ignore flow control, congestion control

Pipelined protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

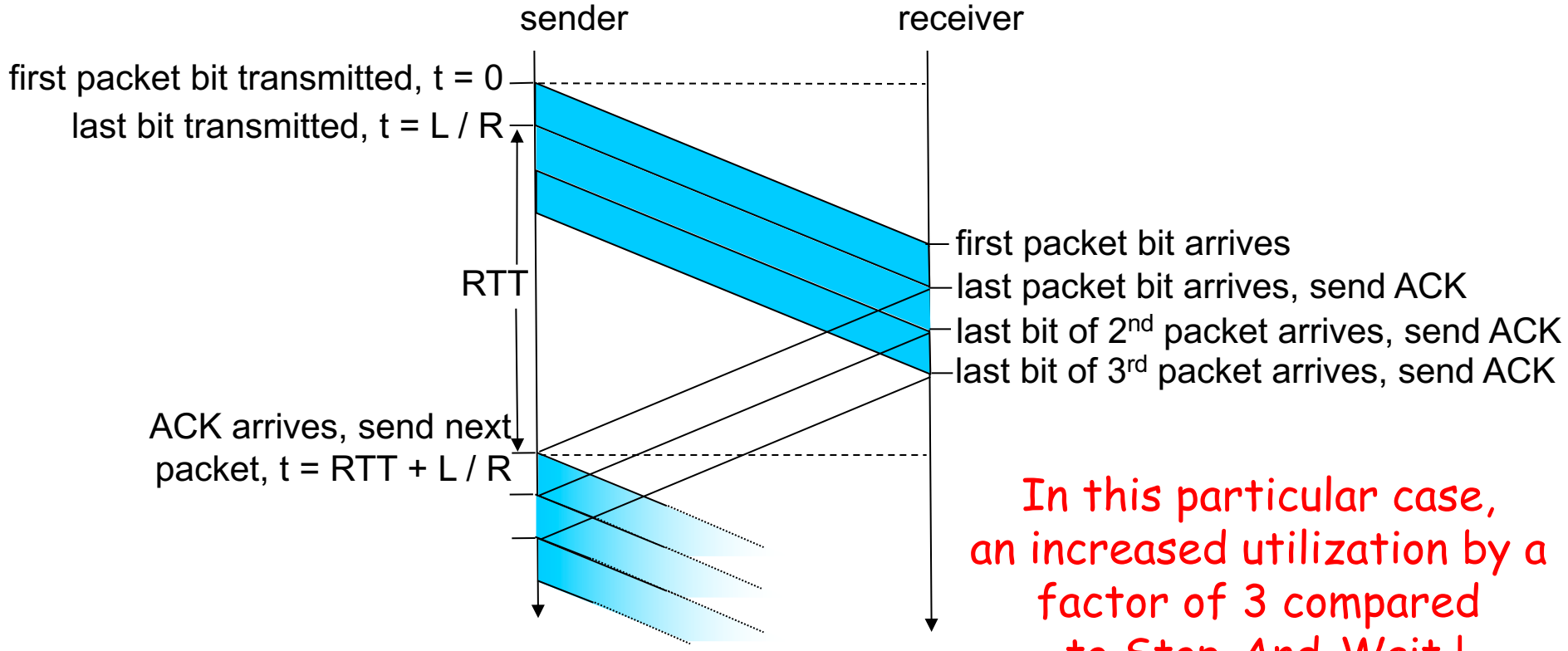
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

Pipelining: increased utilization



In this particular case, an increased utilization by a factor of 3 compared to Stop-And-Wait !

$$\text{Throughput} = \frac{\text{Window Size (bytes, bits, packets)}}{\text{RTT}}$$

TCP sender events

There are 3 events that trigger the transmission of a segment

Data rcvd from app:

- 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`

Timeout:

- retransmit segment that caused timeout
- restart timer

Ack rcvd:

- If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments

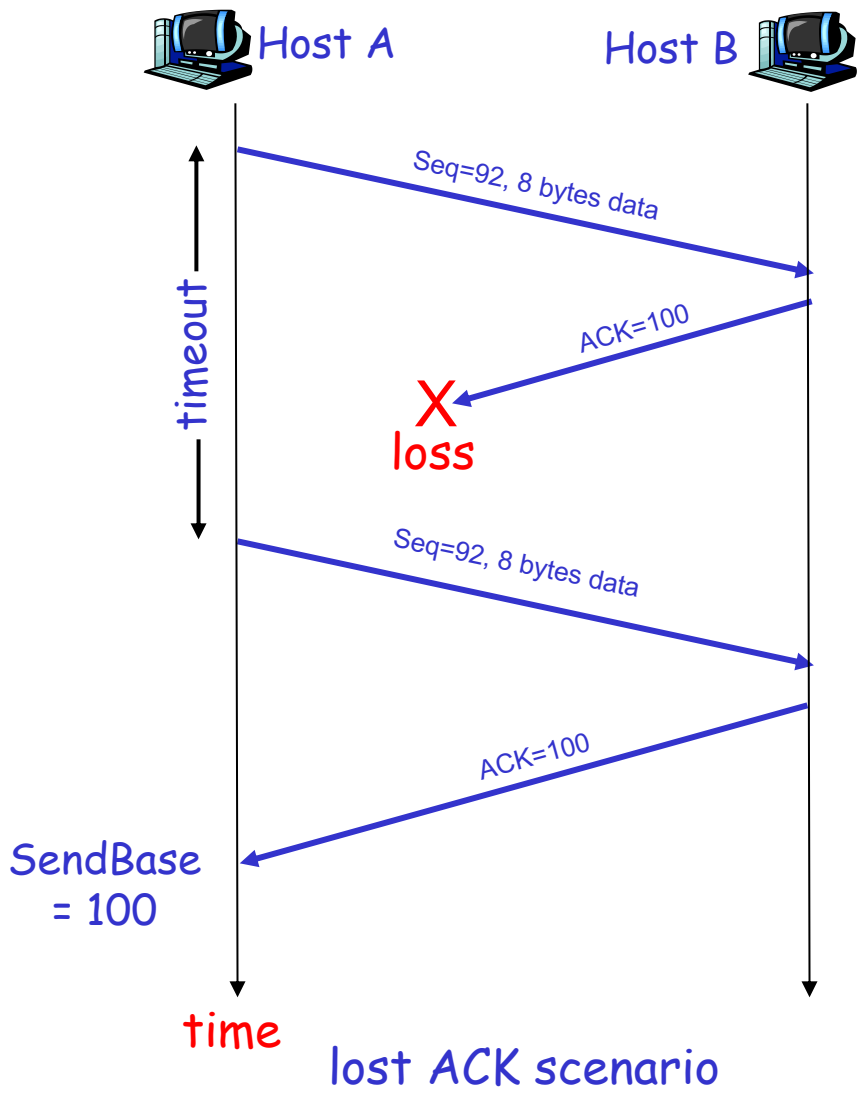
Exponential Back off Timer

- The *TimeoutInterval* is a function of *EstimatedRTT* and *DevRTT*
  - See in the following
- In case the *TimeoutInterval* is exceeded
  - The corresponding segment is resent
  - The *TimeoutInterval* is doubled
  - This is repeated until
    - ACK received, or a maximum value is reached
  - Then the *TimeoutInterval* is resumed to a function of *EstimatedRTT* and *DevRTT*

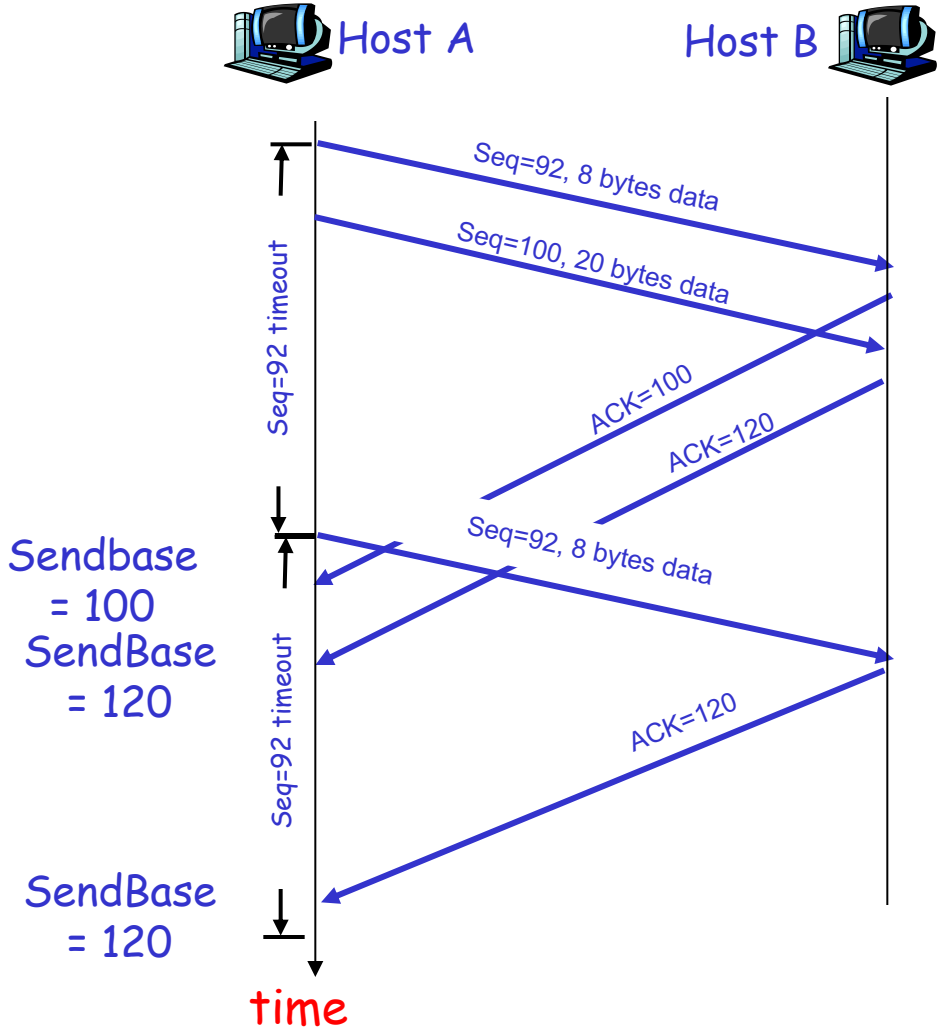
Exponential Back off Timer

- Provide basic congestion control
  - Slow down the sending rate as long as the segment is not ACKed by doubling the time before sending the lost segment

TCP: retransmission scenarios

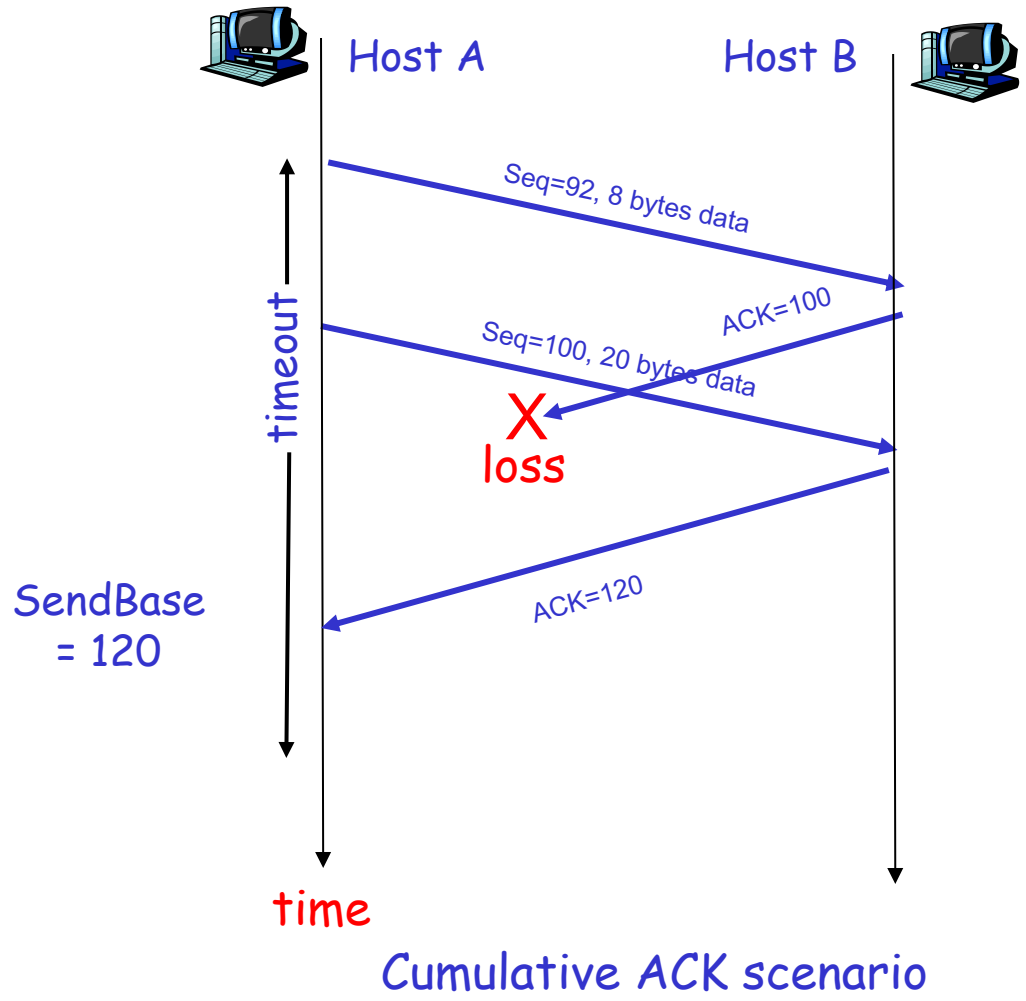


lost ACK scenario



premature timeout

TCP retransmission scenarios (more)



Cumulative ACK scenario

TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 200ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment startsat lower end of gap

TCP Round Trip Time and 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**

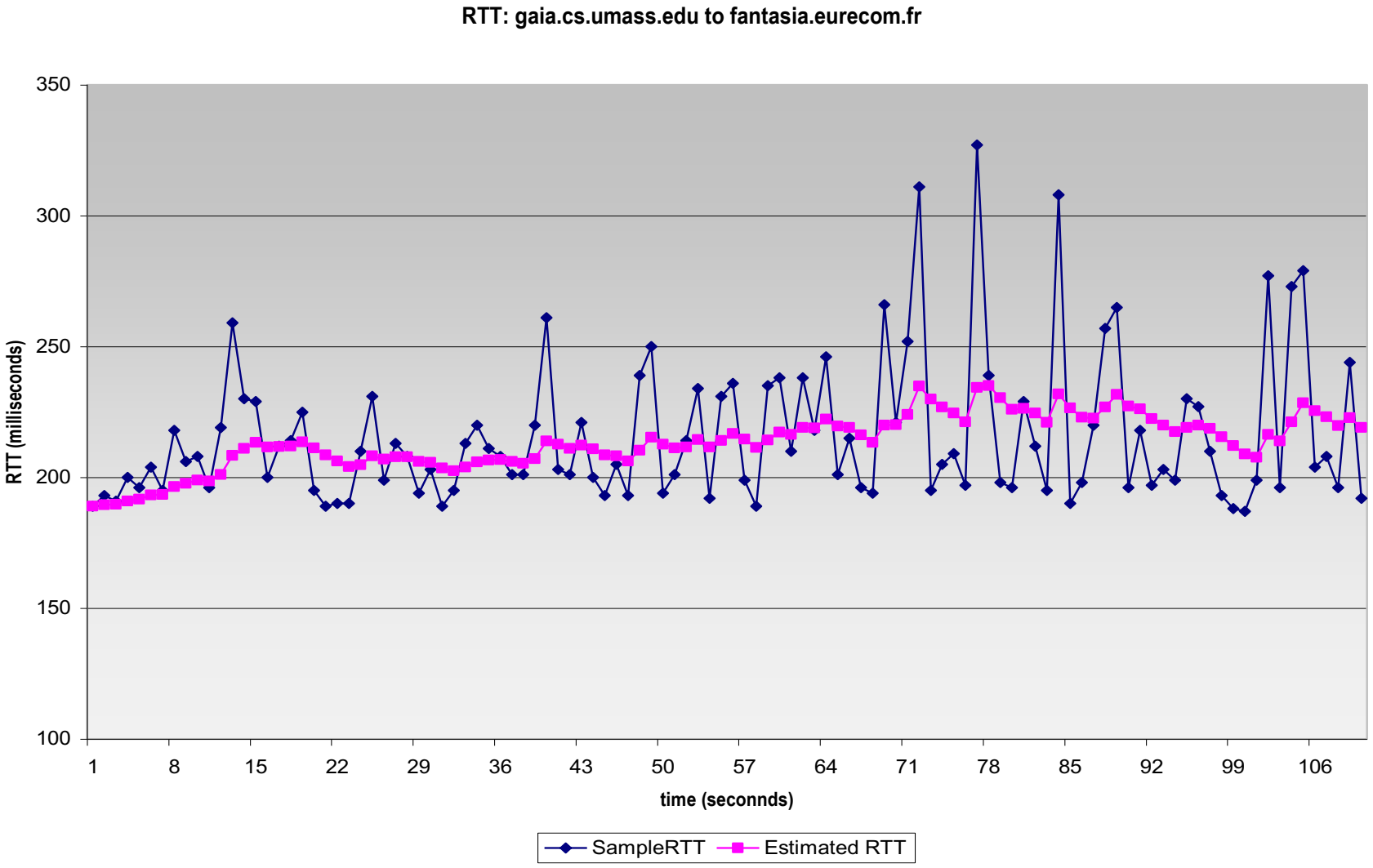
TCP Round Trip Time and Timeout

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

- Exponential weighted moving average
  - Influence of past sample decreases exponentially fast
  - More weight on recent samples than on old ones
- Recommended value: $\alpha = 0.125$

EstimatedRTT = (0.875)*EstimatedRTT + 0.125*SampleRTT

Example RTT estimation:



TCP Round Trip Time and Timeout

Setting the timeout

- **`EstimtedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** -> larger safety margin
- First estimate of how much SampleRTT deviates from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
(Recommended value β = 0.25)
```

Then set timeout interval:

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

- Close the EstimatedRTT when they are few variations (DevRTT small).
- Larger than RTT when they are large variations (DevRTT large)

Fast Retransmit

- Time-out period often relatively long:
  - Long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - Fast retransmit: resend segment before timer expires

Fast retransmit algorithm:

```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase = y
                if (there are currently not-yet-acknowledged segments)
                    start timer
              }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                 }
```

a duplicate ACK for already ACKed segment

fast retransmit

Loss recovery phase: how to retransmit packets ?

- Since ACKs are cumulative, the sender cannot know in one round-trip which segments have been lost in a window
  - It only knows the sequence number of the first lost segment
- Retransmit the first loss segment
  - Transmit new packets if the flow control/congestion control allows
- Wait for a round-trip till the ACK for the retransmitted packet arrives
- Detect if any other lost packet and retransmit it

Selective ACKs: Recover faster

- receiver *individually* acknowledges all correctly received pkts
  - and it buffers out of order pkts for eventual in-order delivery to upper layer
- sender resends pkts for which ACK not received
- more than one lost packets can be detected and retransmitted within the same round-trip
- implemented as a TCP option
  - Cumulative positive ACK
  - Plus an option that contains the gaps in the buffer

Part 2 outline

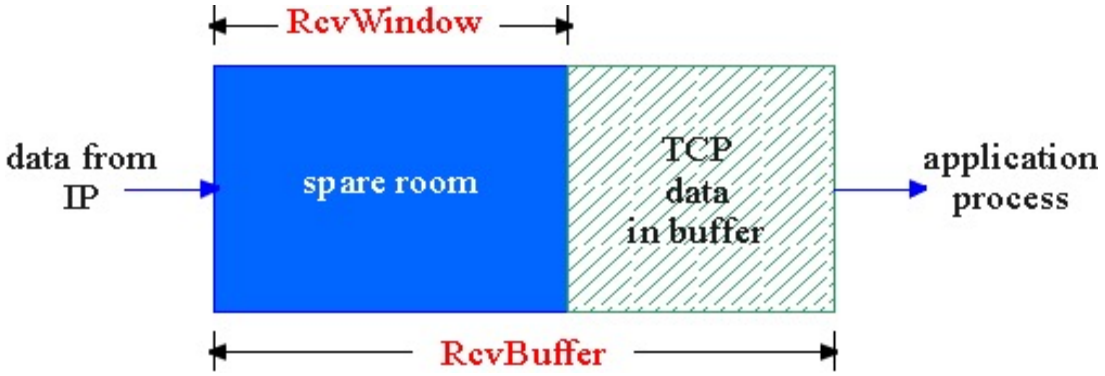
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control
- SCTP / DCCP

TCP Flow Control

- Received in-order data are put in a receiver buffer to wait to be read by the application
  - If the application is slow to read, the receiver buffer may overflow
- This problem is solved by a flow control mechanism
  - To avoid overflow at the receiver buffer
- This is different from congestion control
  - To avoid overflow in the network

TCP Flow Control

- receive side of TCP connection has a receive buffer:



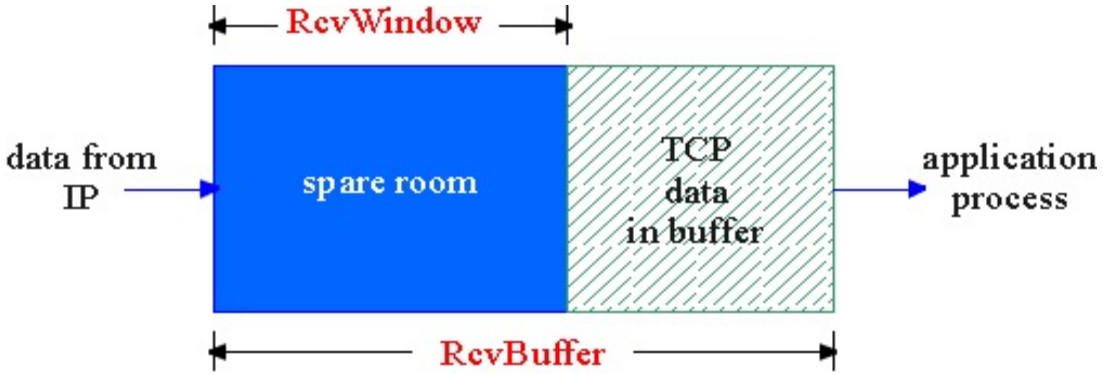
- app process may be slow at reading from buffer

flow control

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

- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



- Rcvr advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow
- RcvWindow is dynamic, it changes with time

- spare room in buffer

= RcvWindow

= RcvBuffer-[LastByteRcvd - LastByteRead]

What if the window shrinks to zero ? How the sender can know it is again open ?

Outline

- 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
- SCTP / DCCP

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- Aim of the connection establishment
  - initialize TCP variables
    - Initialize client and server seq #
    - buffers, flow control info (e.g. `RcvWindow`)
- *client:* connection initiator

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

- *server:* contacted by client

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

TCP Connection Management

Three way handshake:

Step 1: client host sends TCP SYN segment (SYN bit set to 1) to server

- specifies initial seq # for the client
- no data

Step 2: server host receives SYN, replies with SYNACK segment (SYN + ACK bit set to 1)

- server allocates buffers
- specifies initial seq. # for the server in the seq # field
- put the received client seq # +1 in the ACK field

Step 3: client receives SYNACK, replies with ACK segment, which may contain data (SYN bit set to 0, ACK bit set to 1)

- the ACK field is set to the server seq # + 1

TCP Connection Management

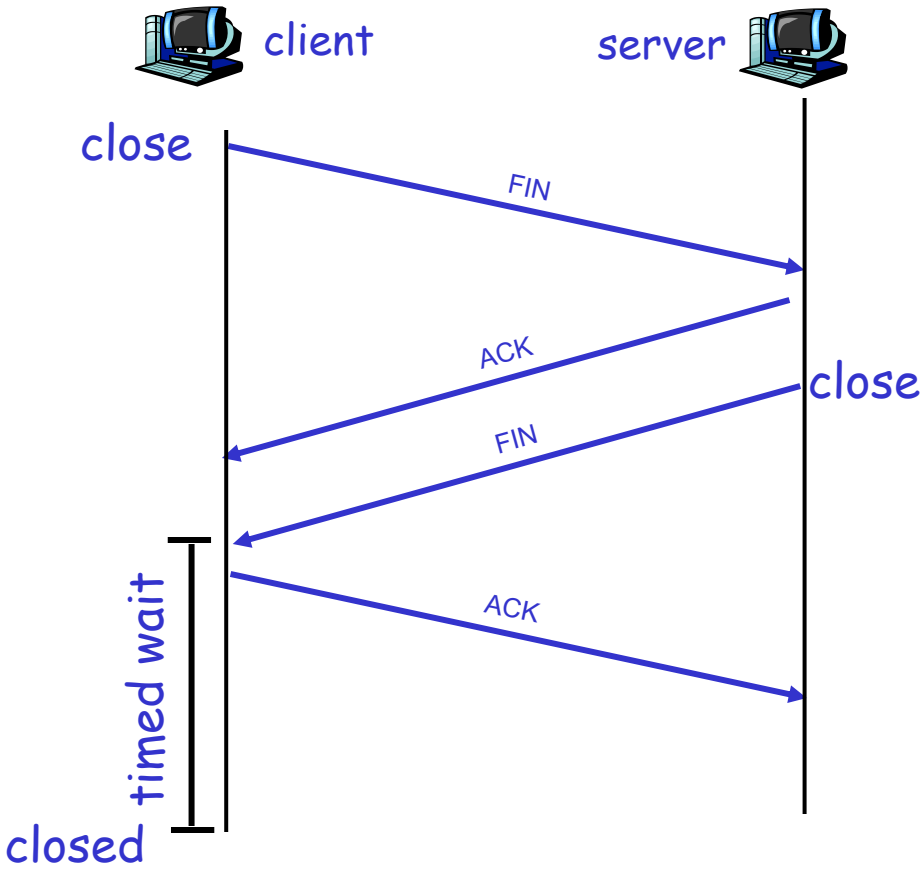
Closing a connection:

client closes socket:

```
clientSocket.close();
```

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



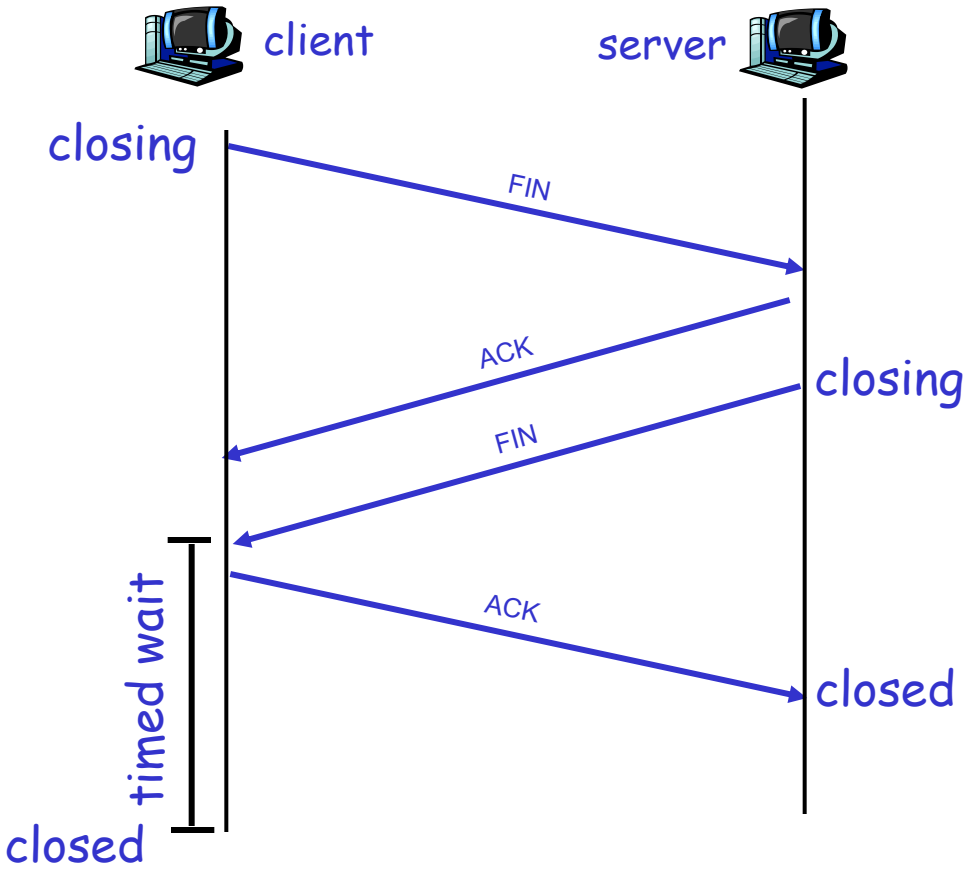
TCP Connection Management

Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Timed_Wait depends on implementations (30s, 60s, 120s)



What's About the RST flag?

- Suppose a host receive a TCP SYN segment for a given destination port, but the host does not have a socket on this port
  - Send a TCP RST (reset) segment to the sender of the TCP SYN
  - Means "I don't have a socket for this segment, do not resend it"
- Note that the in case of UDP, an ICMP message is sent (remind traceroute)

Outline

- 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
- SCTP / DCCP

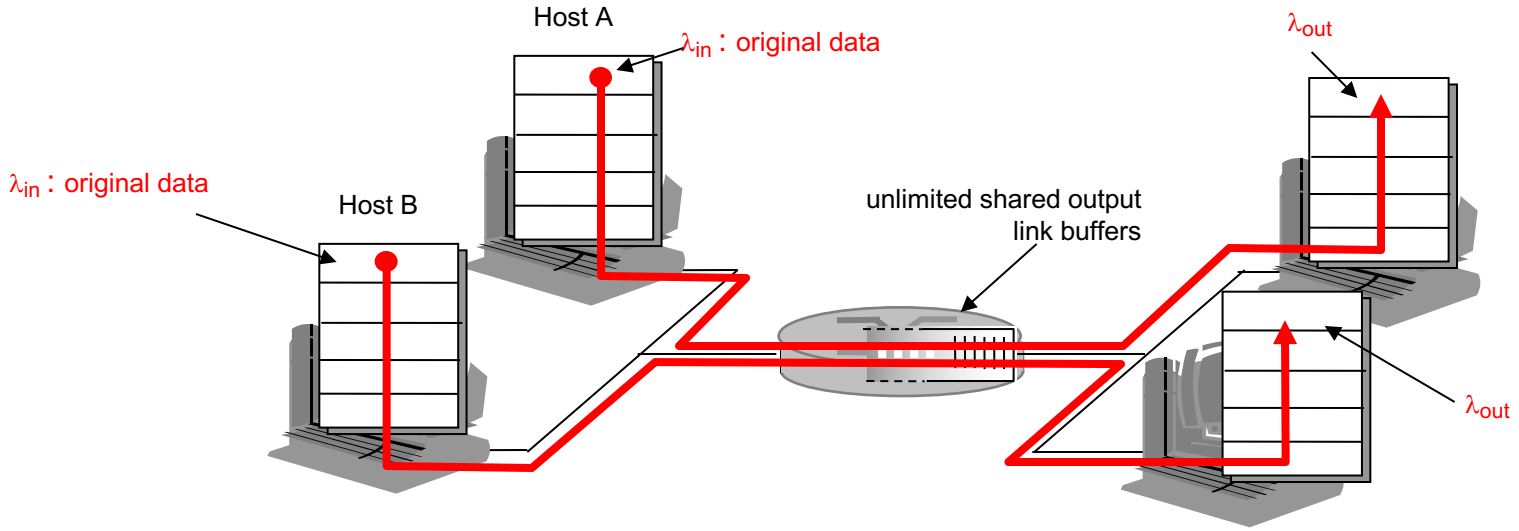
Principles of Congestion Control

Congestion:

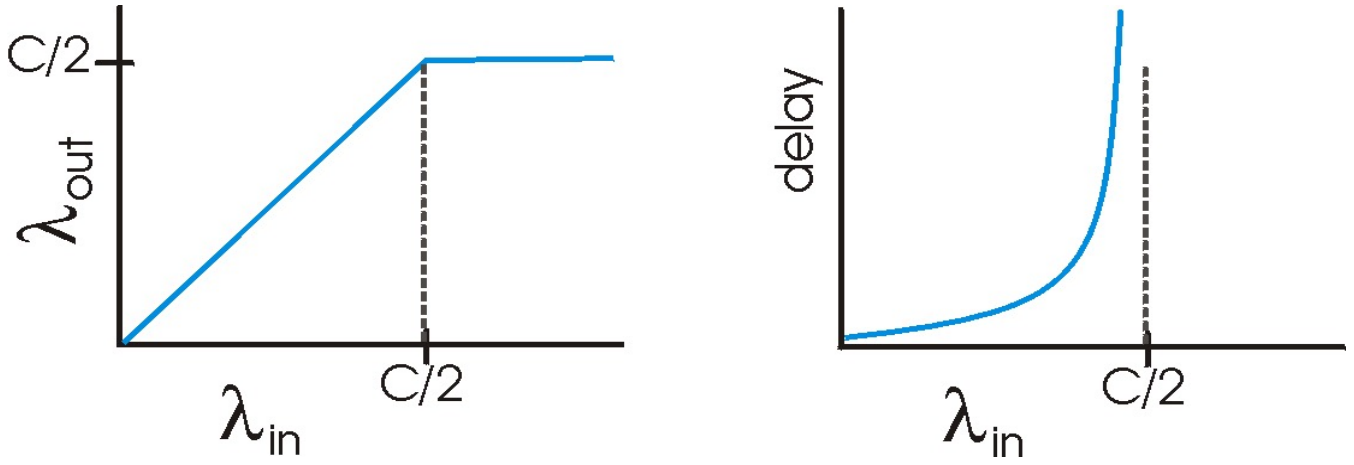
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control (which is end-to-end) !
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!
- Let's consider 3 scenarios

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router of capacity C, infinite buffers
- no retransmission



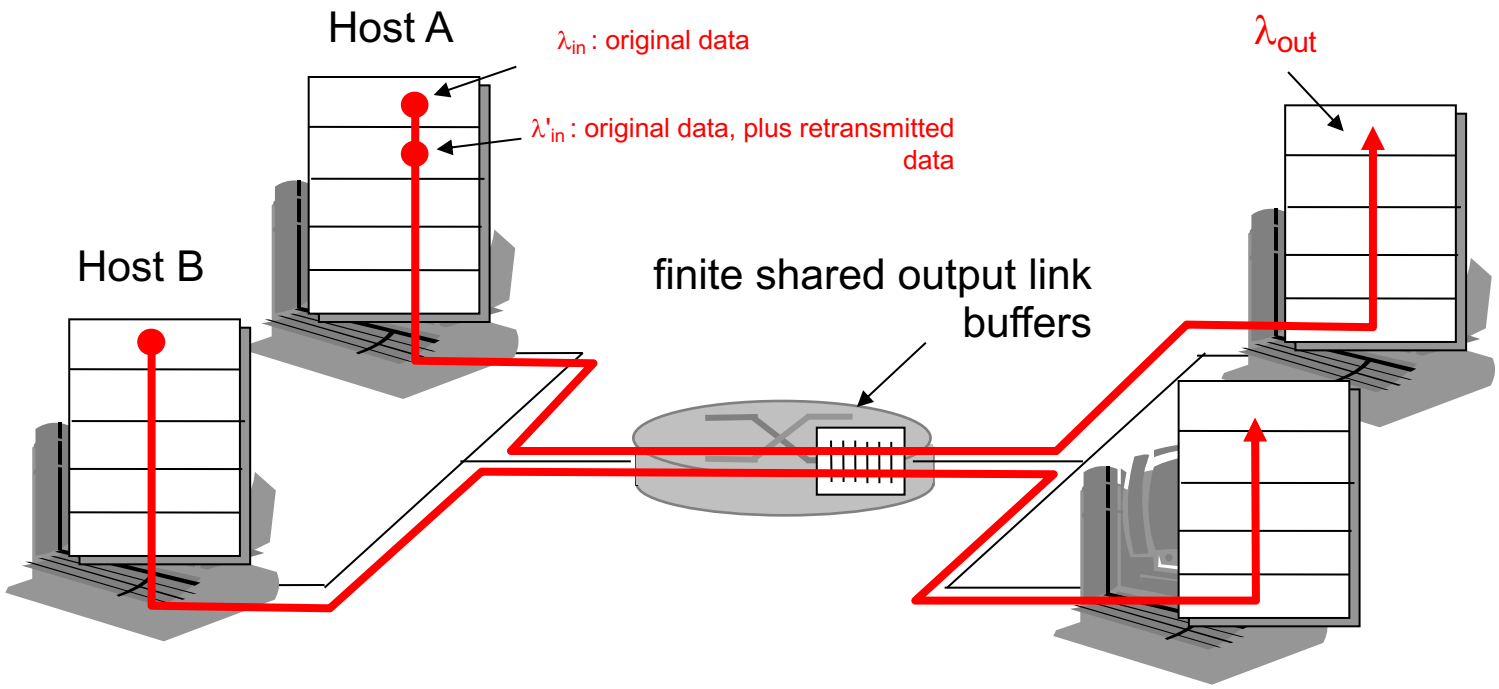
Causes/costs of congestion: scenario 1



- Maximum achievable throughput
  - At most C/2 per TCP session
  - May seem a good utilization of the resources
- But, large delays when sending rate comes close the C/2, and infinite when larger to C/2
- Take as example the M/M/1 queue (λ_{out}? Mean Delay?)

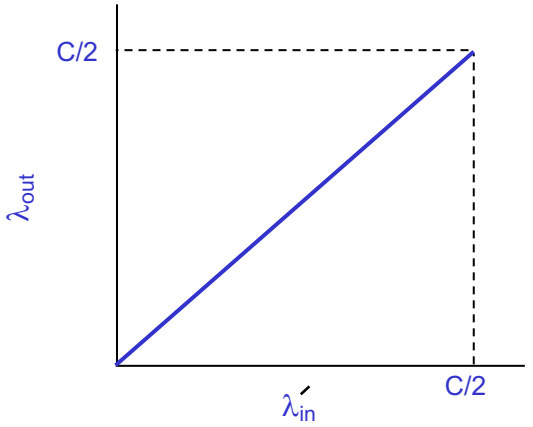
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet
- Sending rate of original data λ_{in}
- Sending rate of original data and retransmission λ'_{in}

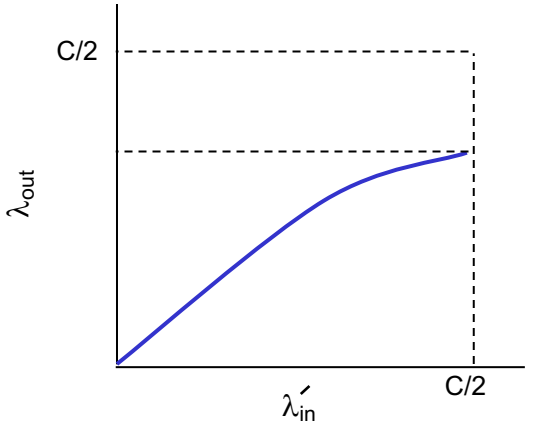


Causes/costs of congestion: scenario 2

- $\lambda_{in} = \lambda_{out}$ (goodput) when link is not fully utilized.
- Otherwise $\lambda_{in} > \lambda_{out}$
- Retransmissions make $\lambda'_{in} > \lambda_{in}$. Two cases:
- 1) "perfect" retransmission only when loss:
  - Achieving a goodput R/2 per connection is possible



- 2) retransmission of delayed (not lost) packet:
  - can cause false timeouts and spurious retransmissions
  - makes λ'_{in} larger (than perfect case) for same λ_{out}
  - Useless retransmissions consume bandwidth and prohibit the goodput from reaching C/2



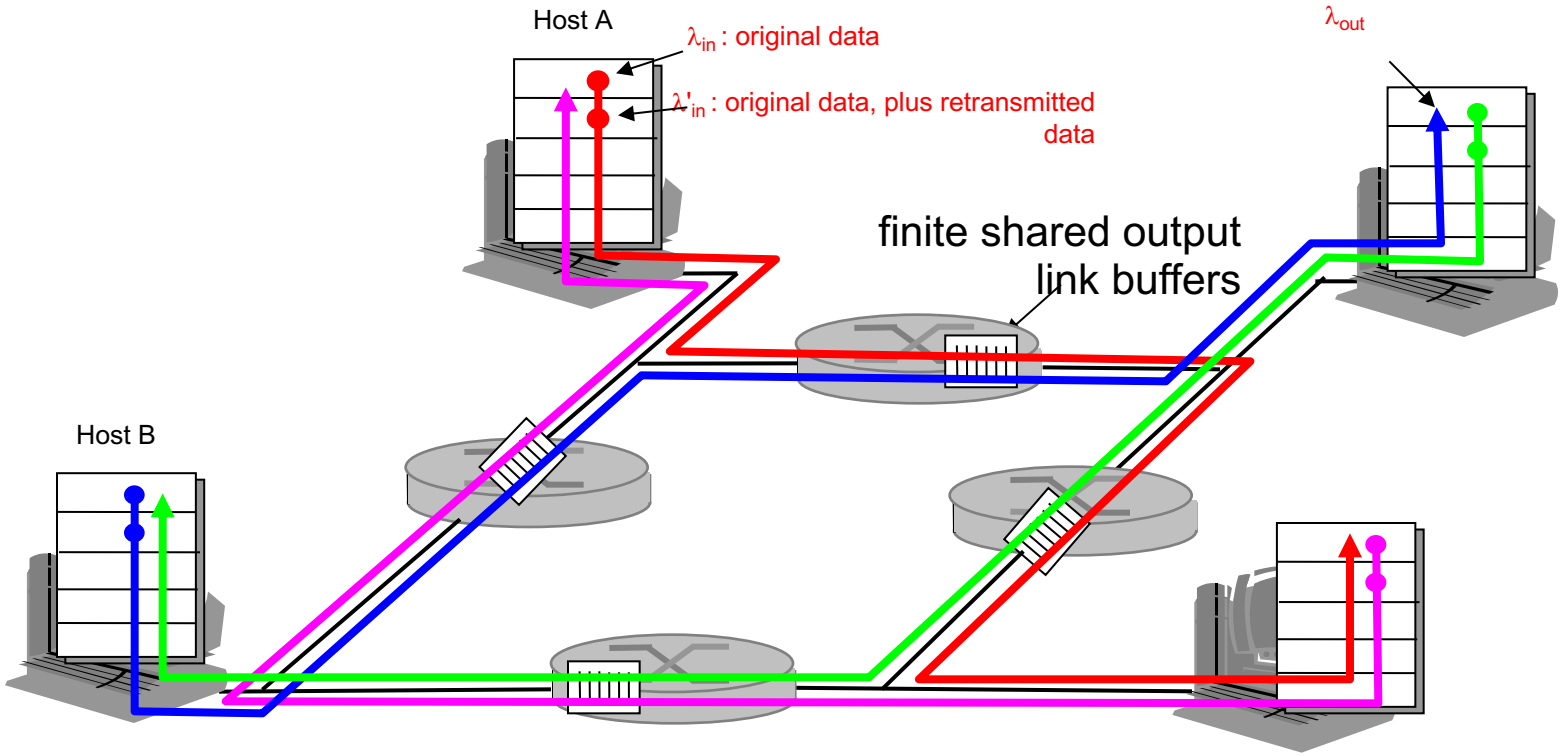
"costs" of congestion:

- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt

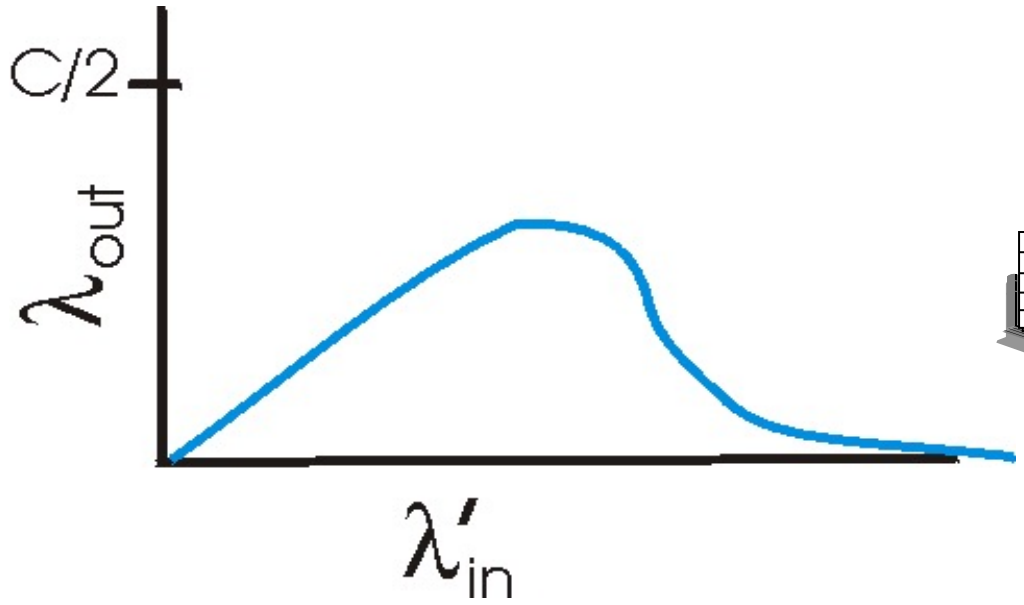
Causes/costs of congestion: scenario 3

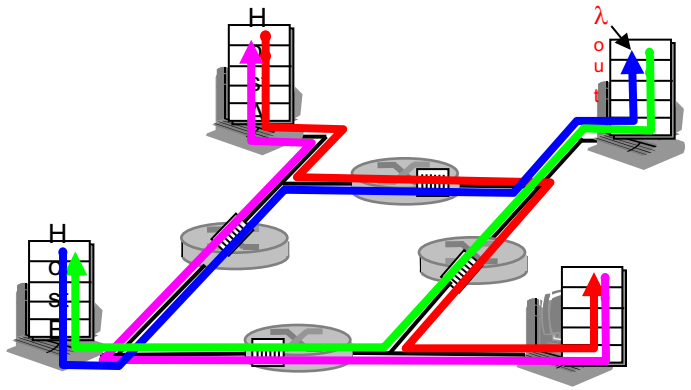
- four senders
- multihop paths
- timeout/retransmit

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



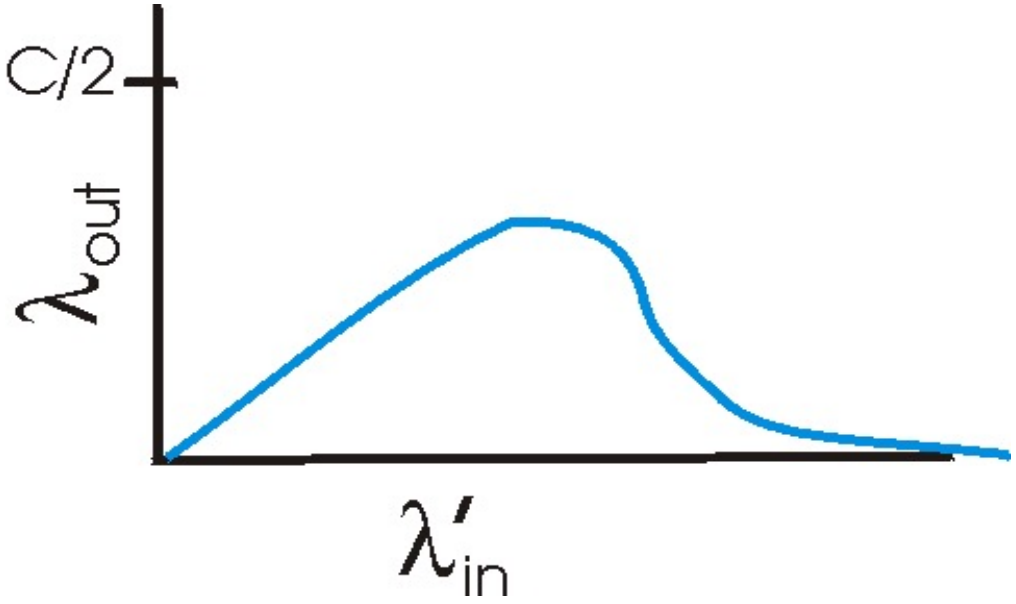
Causes/costs of congestion: scenario 3

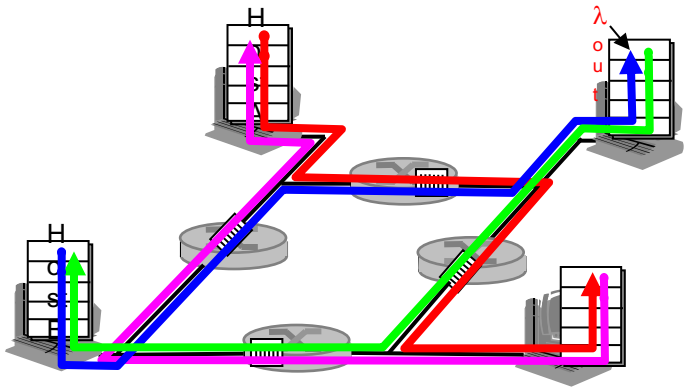




- When λ'_{in} is small, there is no congestion, thus λ_{out} increases linearly with λ'_{in}

Causes/costs of congestion: scenario 3





- When λ'_{in} is large the red traffic leaves the first router at most C, but when it crosses the green traffic at the second router, it competes with a flow at λ'_{in}. As the throughput of the red traffic is independent of λ'_{in} at the second router, it gets less and less bandwidth as λ'_{in} increases.
- In this case λ_{out} decreases exponentially with λ'_{in}

Causes/costs of congestion: scenario 3

Another "cost" of congestion:

- When a packet is dropped, any upstream transmission capacity used for that packet is wasted
- Congestion decreases fast the utility of real networks
  - Goodput is low
  - Congestion must be avoided
- The congestion collapse of the 80s
  - Before TCP congestion control
  - The goodput of the Internet dropped to zero because of TCP many retransmissions.

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
- Does not require any change to intermediate routers

Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at (XCP)
- Its major drawback is that it requires changing everything

Outline

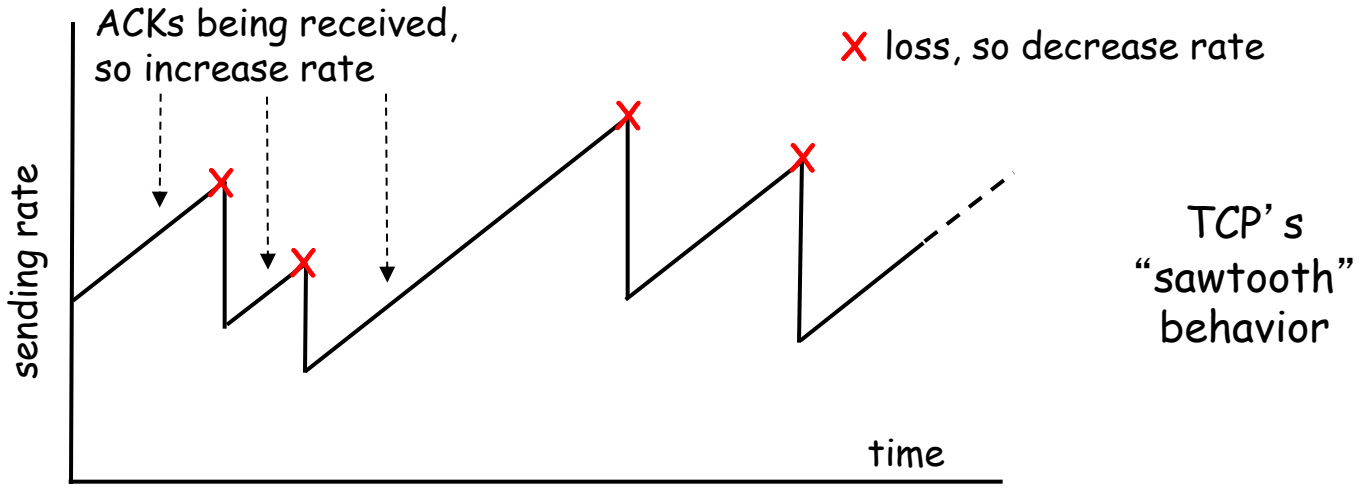
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP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

TCP congestion control:

- *goal:* TCP sender should transmit as fast as possible, but without congesting network
  - Q: how to find rate *just* below congestion level
- decentralized: each TCP sender sets its own rate, based on *implicit* feedback:
  - *ACK:* segment received (a good thing!), network not congested, so increase sending rate
  - *lost segment:* assume loss due to congested network, so decrease sending rate

TCP congestion control: bandwidth probing

- "probing for bandwidth": increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate
  - continue to increase on ACK, decrease on loss (since available bandwidth is changing, depending on other connections in network)



- Q: how fast to increase/decrease?
  - details to follow

TCP Congestion Control: details

- sender limits rate by limiting number of unACKed bytes "in pipeline":

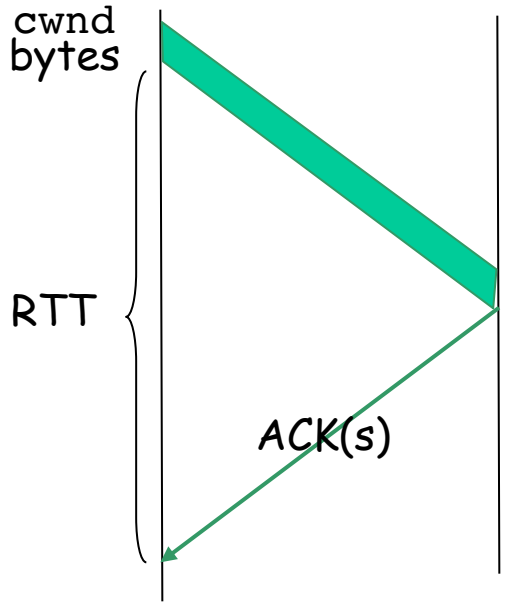
  LastByteSent-LastByteAcked ≤ cwnd

  - **cwnd:** differs from **rwnd** (how, why?)
  - sender limited by **min(cwnd,rwnd)**

- roughly,

$$\text{rate} = \frac{\min(\text{cwnd},\text{rwnd})}{\text{RTT}} \quad \text{bytes/sec}$$

- **cwnd** is dynamic, function of perceived network congestion



TCP Congestion Control: more details

segment loss event: reducing **cwnd**

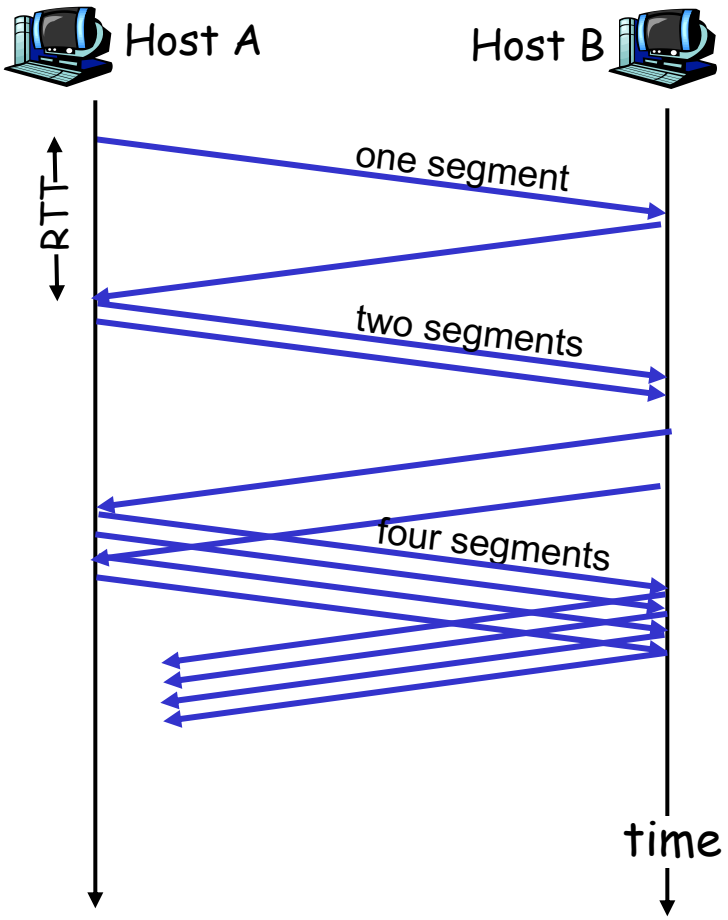
- timeout: no response from receiver
  - cut **cwnd** to 1 MSS
- 3 duplicate ACKs: at least some segments getting through (recall fast retransmit)
  - cut **cwnd** in half, less aggressively than on timeout

ACK received: increase **cwnd**

- slowstart phase:
  - increase exponentially fast (despite name) at connection start, or following timeout
- congestion avoidance:
  - increase linearly

TCP Slow Start

- when connection begins, **cwnd** = 1 MSS
  - example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- increase rate exponentially until first loss event or when threshold reached
  - double **cwnd** every RTT
  - done by incrementing **cwnd** by 1 MSS for every ACK received



TCP: congestion avoidance

- when **cwnd > ssthresh** grow **cwnd** linearly
  - increase **cwnd** by 1 MSS per RTT
  - approach possible congestion slower than in slowstart
  - implementation: **cwnd = cwnd + MSS/cwnd** for each ACK received

AIMD

- ACKs: increase **cwnd** by 1 MSS per RTT: additive increase
- loss: cut **cwnd** in half (non-timeout-detected loss): multiplicative decrease

AIMD: Additive Increase Multiplicative Decrease

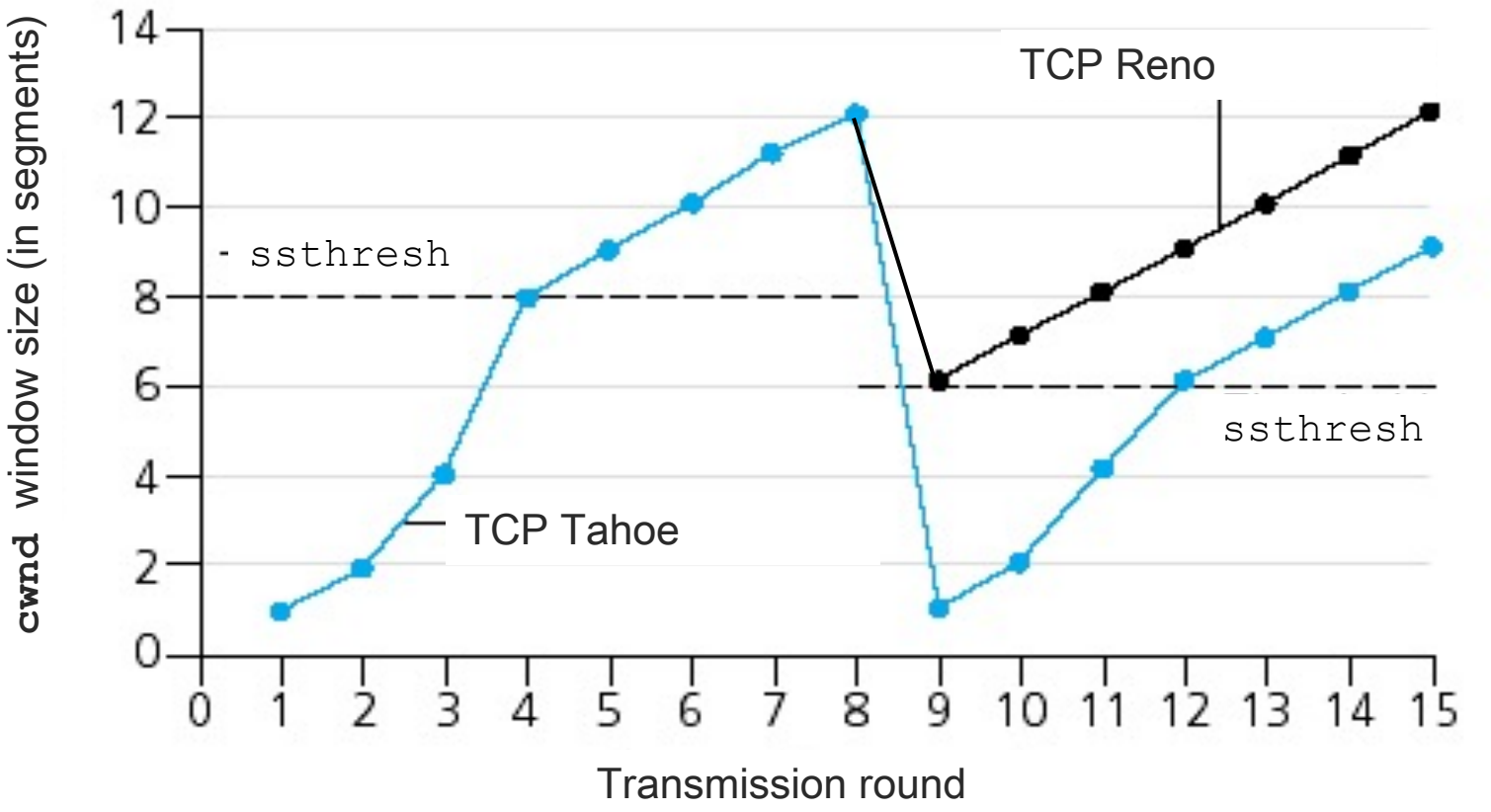
Refinement: inferring loss

- After timeout event
  - **Threshold** is set to half **CongWin**
  - **CongWin** set to 1 MSS
  - Window then grows exponentially up to **Threshold**
  - Then grows linearly
- After 3 dup ACKs:
  - **CongWin** is cut in half
  - window then grows linearly
  - **Threshold** is set to half **CongWin** before 3 dup ACKs

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

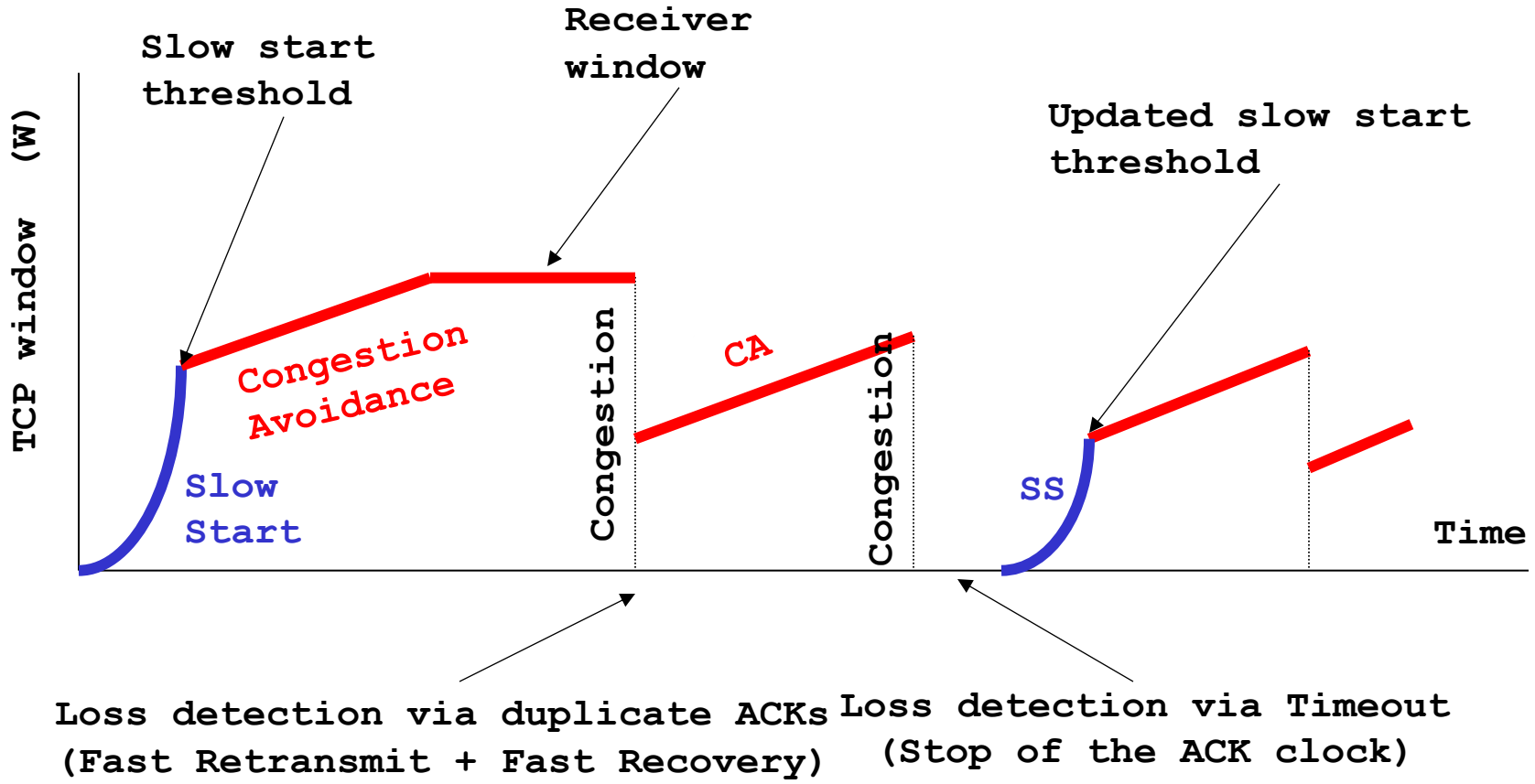
Popular "flavors" of TCP



Summary: TCP Congestion Control

- when **`cwnd < ssthresh`**, sender in slow-start phase, window grows exponentially.
- when **`cwnd >= ssthresh`**, sender is in congestion-avoidance phase, window grows linearly.
- when triple duplicate ACK occurs, **`ssthresh`** set to **`cwnd/2`**, **`cwnd`** set to ~ **`ssthresh`**
- when timeout occurs, **`ssthresh`** set to **`cwnd/2`**, **`cwnd`** set to 1 MSS.

Summary: TCP Congestion Control



TCP Congestion Control

- As TCP relies on the received ACKs to increase its congestion window, one say that TCP is self clocked
- Benefit: No need for timers to pace packets.
  - Will be very costly to implement.
- Problems:
  - Congestion increases by same amount every window size of packets whatever is the RTT value
  - What if different competing connections have different RTTs ?
  - What if ACKs are lost ?
  - Or delayed as with the delay ACK option ?

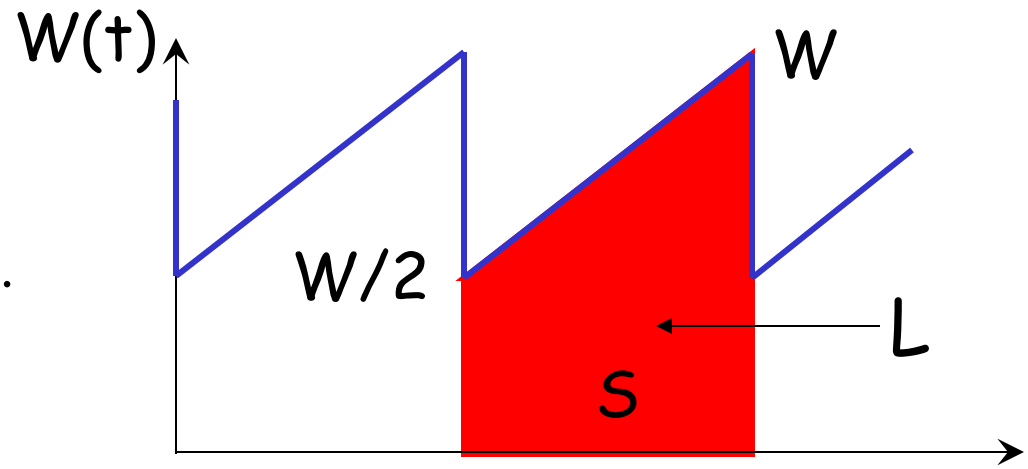
TCP performance

TCP throughput

- *Q:* what's average throughout of TCP as function of window size, RTT?
  - ignoring slow start
- let W be window size when loss occurs.
  - when window is W, throughput is W/RTT
  - just after loss, window drops to W/2, throughput to W/2RTT.
  - average throughout: X = 0.75 W/RTT

A simple model for TCP throughput

[FLO91,MSMO97]

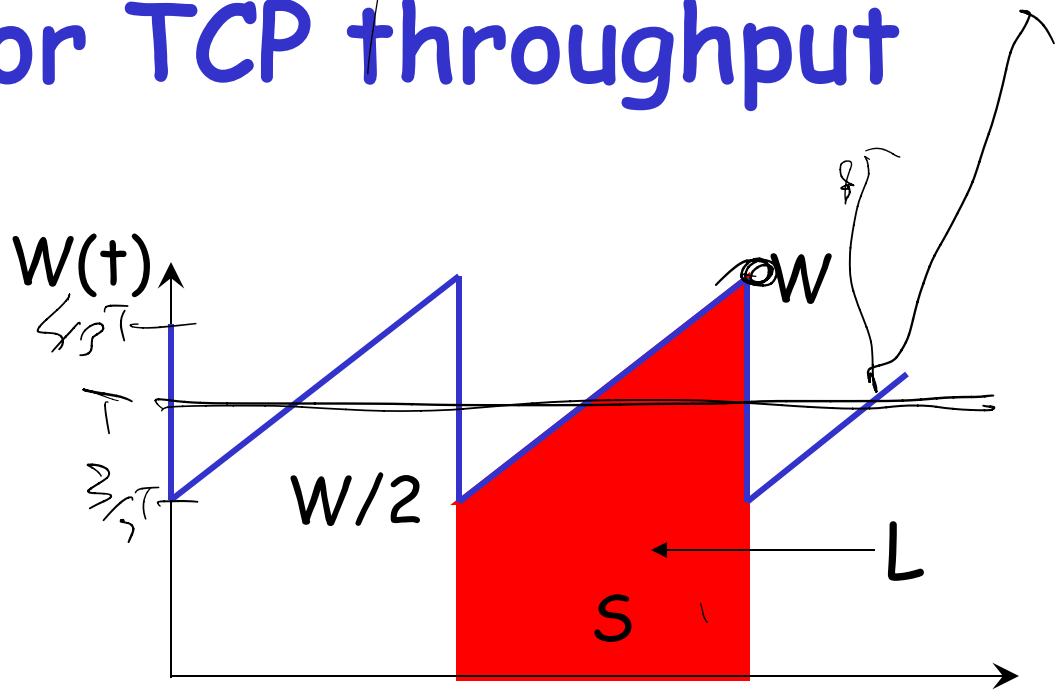


- Assumptions
  - Infinitely long TCP connection.
  - No Timeouts (no slow start).
  - Infinite receiver window.
  - Periodic evolution of the congestion window (time S constant).
  - Congestion window fluid and in packets.
  - Packet losses appear periodically with rate p (0 < p < 1).

- Throughput $X = \dfrac{\text{Number of packets transmitted per cycle } L}{\text{Cycle duration } S}$

A simple model for TCP throughput

- Each cycle: Congestion window to be increased by (W - W/2).
- In congestion avoidance phase, window increases by 1 packet every round trip time.
- Window size of packets are transmitted per round-trip time.



- Thus, the total number of packets sent per cycle is:
  - $L = W/2 + (W/2 + 1) + (W/2 + 2) + \ldots + (W/2 + W/2) \approx (3/8)W^2$
- And cycle duration $S = RTT.W/2$
- But, we have $L = 1/p = (3/8)\,W^2$
- Hence, $$X = \frac{L}{S} = \frac{1}{RTT}\sqrt{\frac{3}{2p}} \text{ packets/s}$$

Square Root Formula !

Comments on the SQRT formula

- The throughput of TCP is inversely proportional to the square root of p, the packet loss probability.
  - Bad performance of TCP over wireless links where packets are lost for other reasons than congestion, e.g. transmission errors.
- The throughput of TCP is inversely proportional to the round-trip time.
  - Bad performance of TCP over satellite links where the round-trip time is large, e.g. 500 ms over GEO satellite links.
  - TCP is unfair against connections with large round-trip times.
- The throughput of TCP is proportional to the packet size MSS.
  - A bulk-data TCP connection has interest to use large packets.

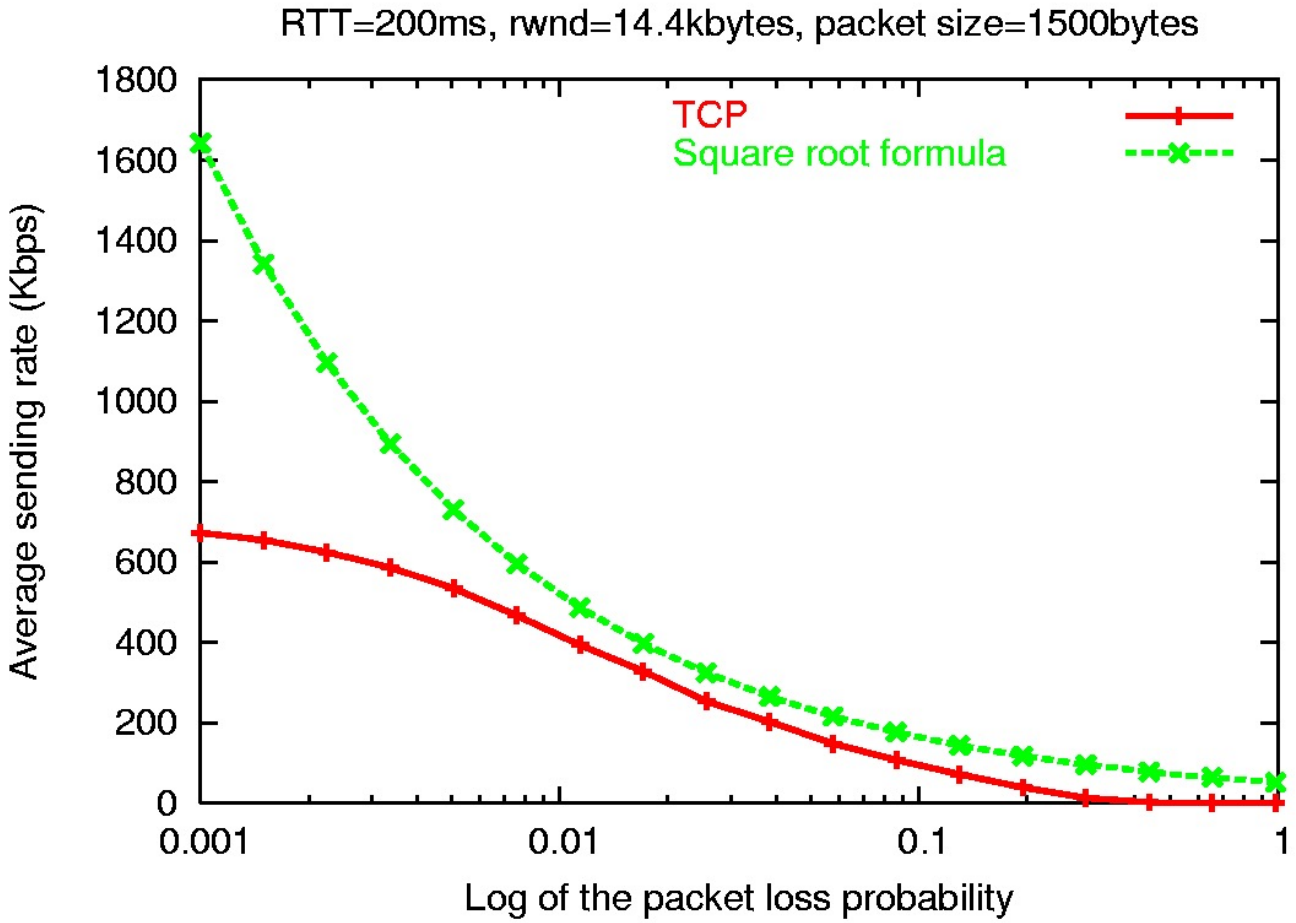
TCP over "fat pipes, large BDP"

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
  - If there is a packet loss, how much time to recover ?
- p should be as low as $2 \cdot 10^{-10}$ *Wow this is not realistic*
- New versions of TCP for high-speed needed!
  - Still a research area (High Speed TCP, Scalable TCP, Fast TCP, Binomial TCP, etc etc)

Problems with the SQRT formula

- Does not work when p is high:
  - The fluid assumption does not hold, i.e. the approximation $L \approx (3/8)W^2$.
  - Timeouts become frequent.
  - More than one packet can be lost in the same RTT, resulting in only one division of TCP window. Hence, p overestimates the congestion signal rate, and the SQRT formula underestimates the throughput.
- And it does not work when p is low:
  - At low p, the receiver window is often reached, which limits the throughput.
  - And at low p, the window is large, so it is very probable that the TCP protocol stops being linear increase with time !
- To all that, one has to add the problem with the assumption on the constancy of times between congestion events.

A simple simulation with ns-2



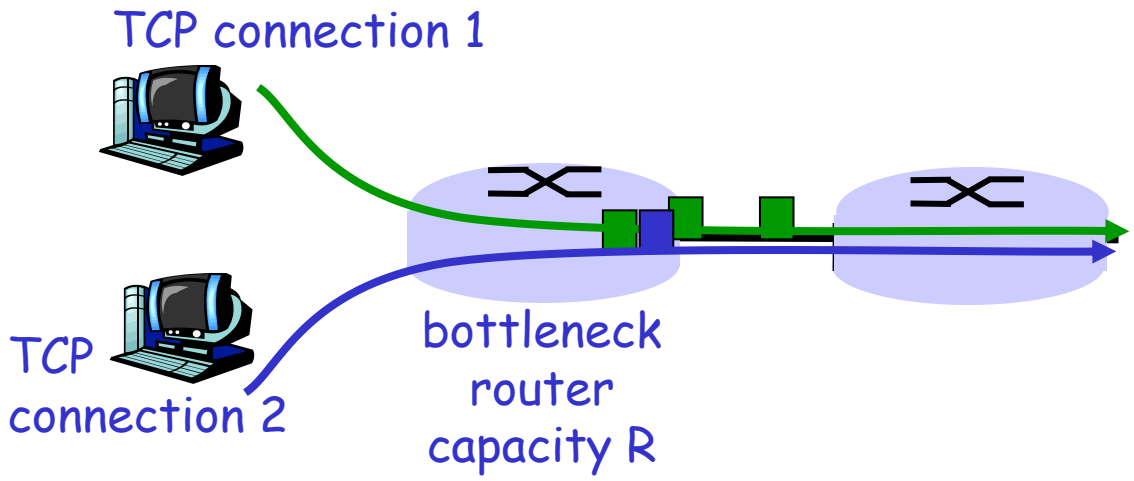
Look at the following document for a description on how to simulate such a scenario:

http://www.inria.fr/planete/chadi/NSCourse-2.pdf

TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K

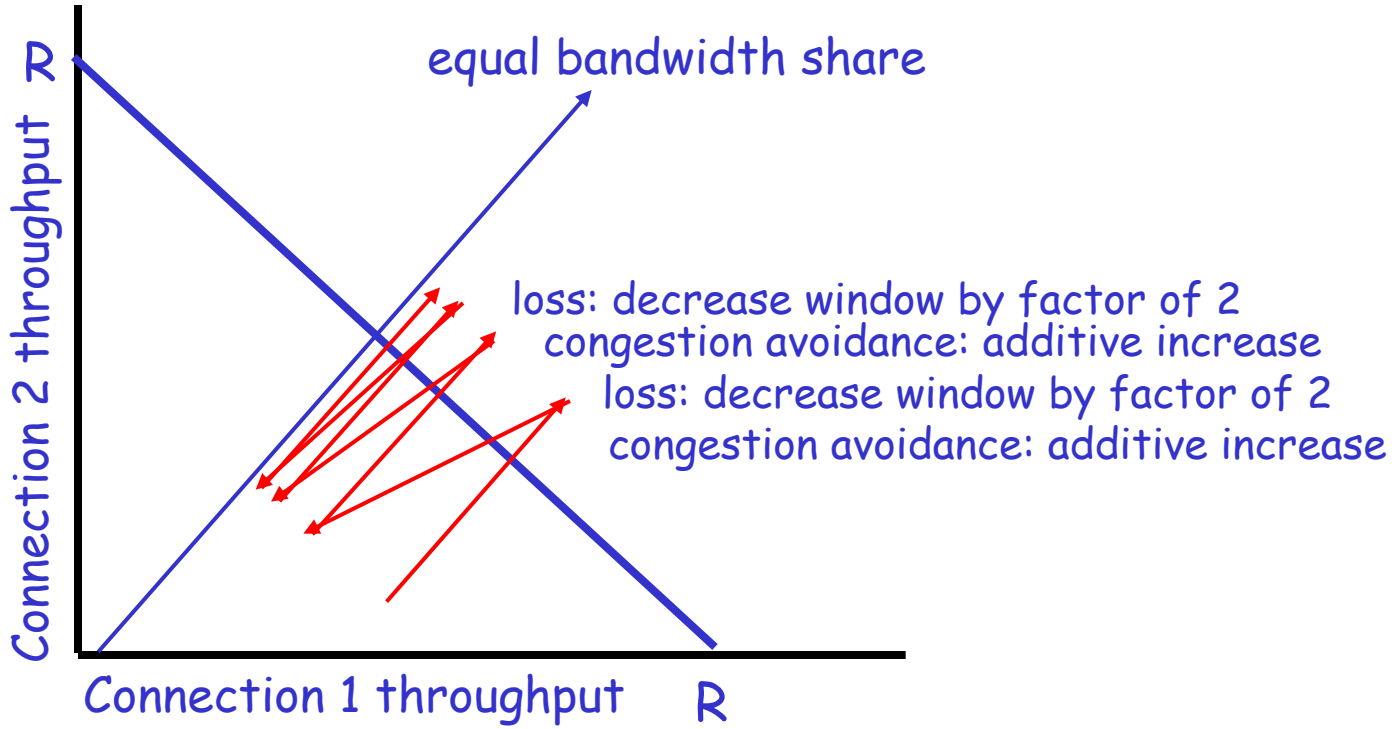
- If same round-trip time – performance inversely proportional to RTT



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

Outline

- 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
- SCTP / DCCP

SCTP : Stream Control Transport Protocol

- A kind of combination of both TCP and UDP
  - UDP not reliable but it is message oriented
  - TCP is reliable, offers congestion control, but it is byte oriented
- Plus the notion of multiple streams inside the same connection
  - A stream can be for example an HTTP object
  - A stream does not wait for the retransmission of a packet from other streams
- Originally proposed to carry audio signaling protocols that require reliability, the notion of streams and a message-oriented service
- Reliability in SCTP can be controlled

DCCP: Datagram Congestion Control Protocol

- RFC 4340
- Again, a combination of TCP and UDP
- TCP provides congestion control, but it is reliable and byte oriented
  - That's not good for multimedia applications for which reliability is not a big issue, whereas packet delay is a big issue
- UDP is fine but there is no congestion control
- DCCP fills the gap
  - As UDP, message oriented but no reliability
  - As TCP, provides congestion control for multimedia applications
  - TFRC a typical example !

Transport of multimédia applications over UDP

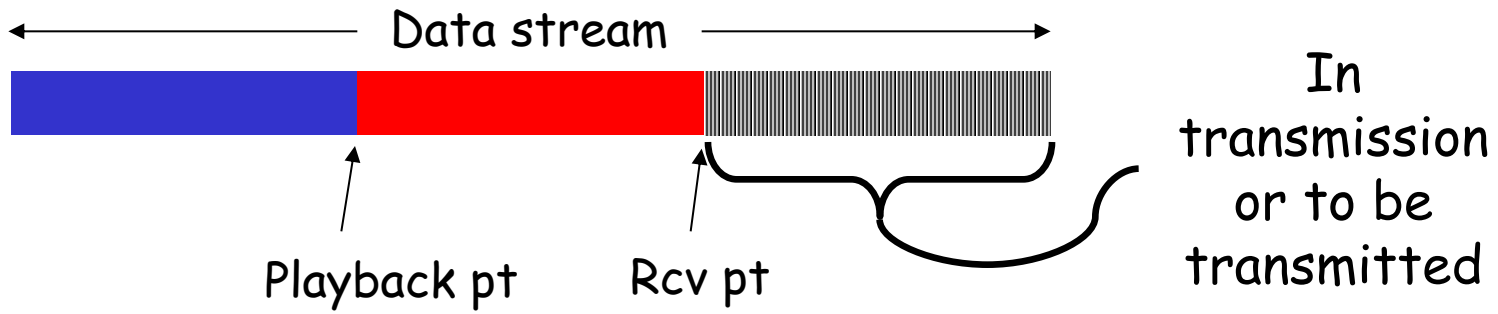
Case of Audio over IP

Classes of audio applications

- Three classes of applications:
  - Streaming
  - Unidirectional Real-Time
  - Interactive Real-Time
- Each class might be broadcast (multicast) or may simply be unicast (point-to-point).

Audio streaming class

- Streaming: Application usage of data during its transmission.



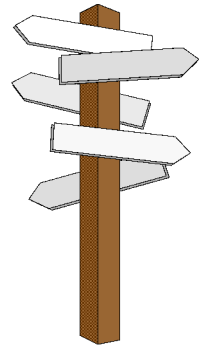
  - Clients request audio files from servers and listen to them during the download, e.g. RealPlayer, windows media player with the wmv files.
  - Interactive: User can control operation, e.g. pause, resume, fast forward.
- Important and growing application due to reduction of storage costs, increase in high speed net access from homes, enhancements to caching and introduction of QoS in IP networks.
- No strict constraints on the delay (5 to 10 seconds judged acceptable).

The other two classes

- Unidirectional Real-Time:
  - Similar to existing radio stations, but delivery on the network.
  - Non-interactive, just listen.
  - No strict constraints on the delay, but clearly more than for streaming.

- Interactive Real-Time :
  - Phone conversation.
  - More stringent delay requirement than Streaming and Unidirectional because of interactivity (good for less than 150 ms, acceptable for less than 400 ms, poor otherwise).
  - This class is what we call IP telephony.
  - Audio over the Internet or VoIP can be any one of the three classes.

Outline



Transport of audio over the Internet

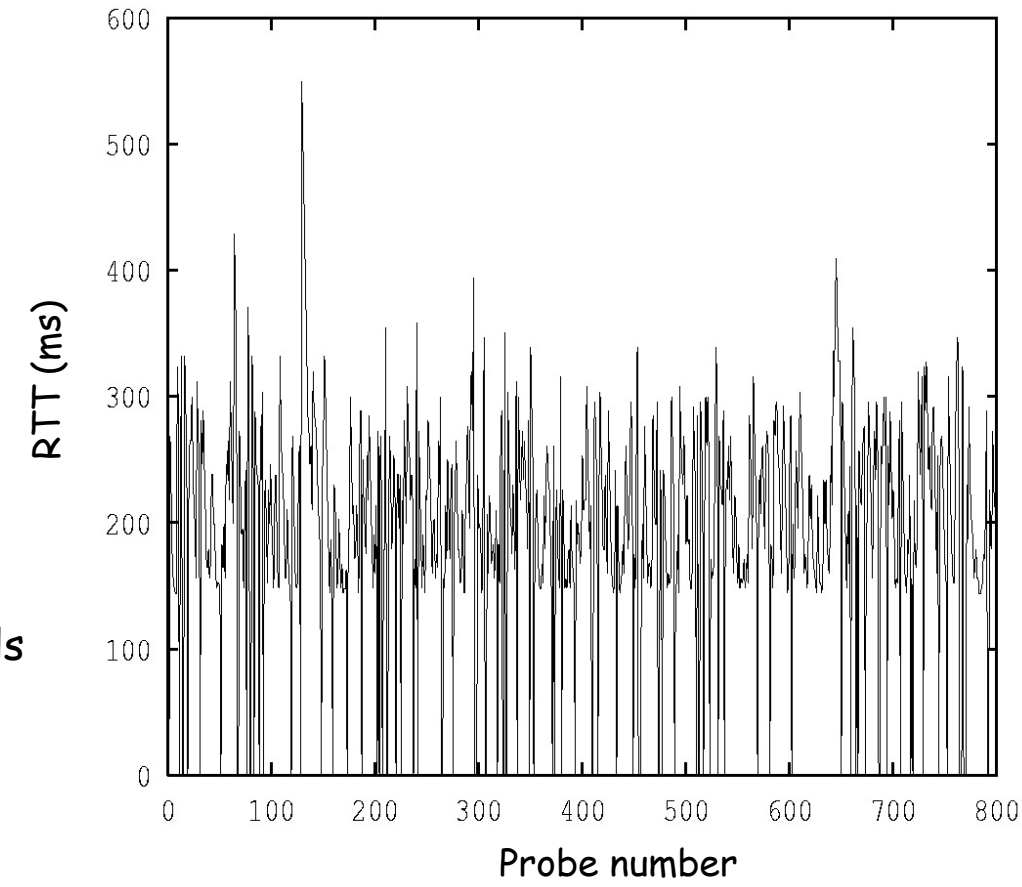
- Challenges: Delay, jitter, and losses
- Impact of delay and solutions
- Solve for jitter
- Impact of losses and solutions
- Rate adaptation: Equation-based congestion control

Transport of audio over the Internet

- These applications are not suited to TCP for different reasons:
  - TCP is byte oriented.
  - TCP introduces delay variability due to retransmissions and reordering.
  - TCP introduces rate variability due to congestion control.
- Moreover, audio applications can tolerate some losses and do not require full reliability:
  - Losing some packets from time to time is not very harmful for an audio conversation.
  - There are also different techniques to reconstruct the lost information without retransmissions.
- So the best choice is to support audio applications over UDP (could be SCTP for situations where there is enough bandwidth - no bit rate variability).
- But UDP does not provide any guarantee and the service can be very poor. Some measures must be taken !

Internet service can be bad

1	tom.inria.fr
2	t8-gw.inria.fr
3	sophia-gw.atlantic.fr
4	icm-sophia.icp.net
5	Ithaca.NY.NSS.NSF.NET
6	Ithaca1.NY.NSS.NSF.NET
7	nss-SURA-eth.sura.net
8	sura8-umd-c1.sura.net
9	csc2hub-gw.umd.edu
10	avwhub-gw.umd.edu



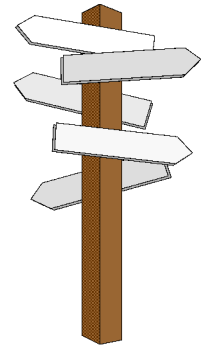
- UDP probes sent at regular time intervals between INRIA and the University of Maryland in July 1992.
  - A probe every 500 ms.
  - Round-trip time (RTT) is measured.
  - Round-trip time set to 0 when a packet is lost.

9 % of probes are lost in this experiment.

The situation today: Over-dimensioned network

- Before of reduced cost of bits/s (optical fiber, deployment of large infrastructures, etc), the actual Internet is over dimensioned in the core (link utilization can be less than 50%)
  - This implies a constant delay in the core
  - And almost no losses
- This has reduced the interest from using UDP
- To add to that the fact that UDP traffic is often blocked by ISPs
- More and more audio applications propose the utilization of TCP
- TCP solution will work as long as the Internet is over-dimensioned, both in the core and at the access, otherwise UDP is to be used
  - Do you think the wireless access is over dimensioned ?

Outline



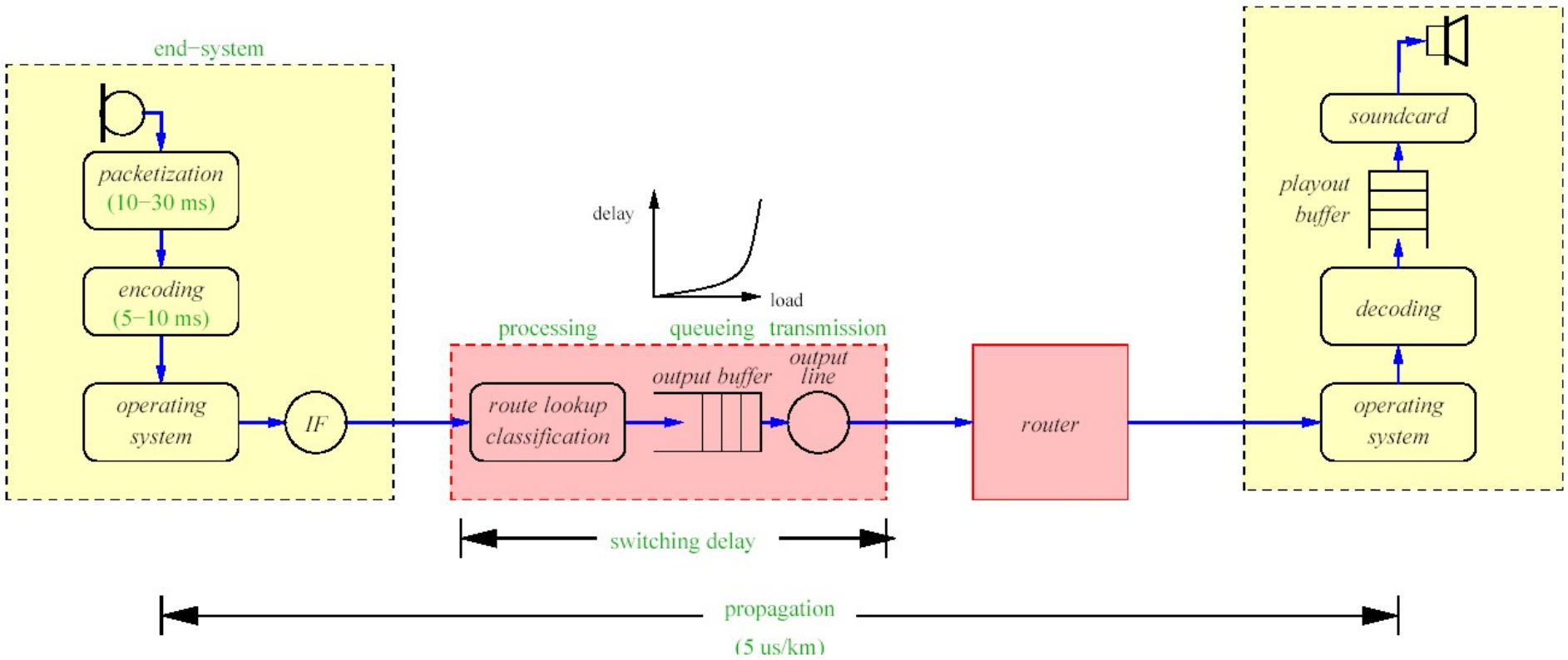
I- Transport of audio over the Internet

- Challenges: Delay, jitter, and losses
- Impact of delay and solutions
- Solve for jitter
- Impact of losses and solutions
- Rate adaptation: Equation-based congestion control

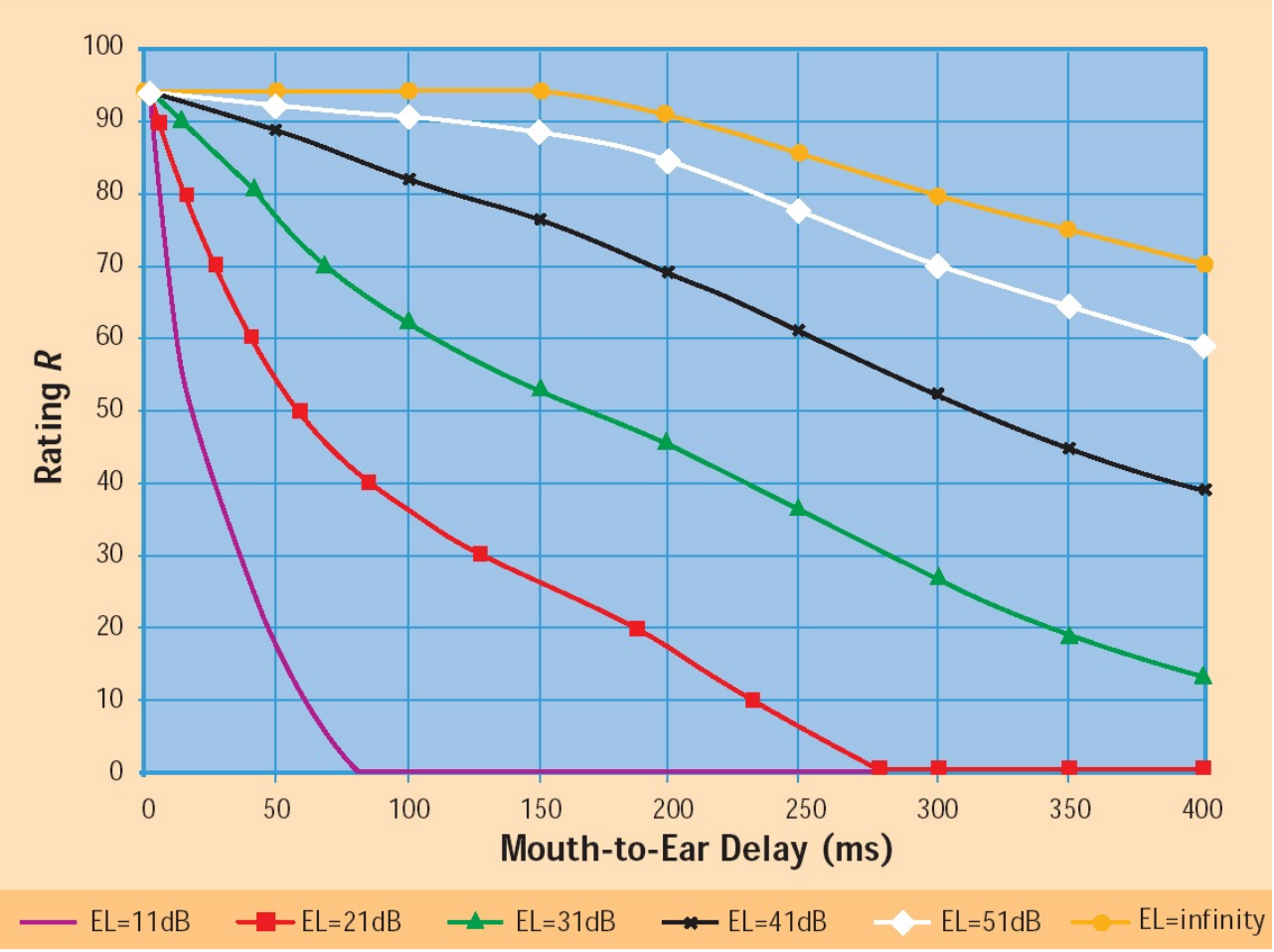
Challenges in supporting audio: Delay

- The delay can be large:
  - Large delay harmful for interactive audio:
    - One-way delay < 150 msec very good.
    - One-way delay < 400 msec acceptable, beyond 400 msec interactivity is poor.
  - And large delays exacerbate the problem of echo.
    - Echo cancellation algorithms are needed, similar to those deployed in PSTN networks incorporating satellite and transoceanic links.
  - The delay is not a problem for non-interactive audio e.g. radio, clips.
- Different components of delay:
  - Propagation delay on links: NOTHING TO DO.
  - Queuing delay in network routers: Something can be done.
    - Give audio packets priority over data packets in network routers: Quality of Service.

Delay sources



Impact of delay on audio quality (E-model)



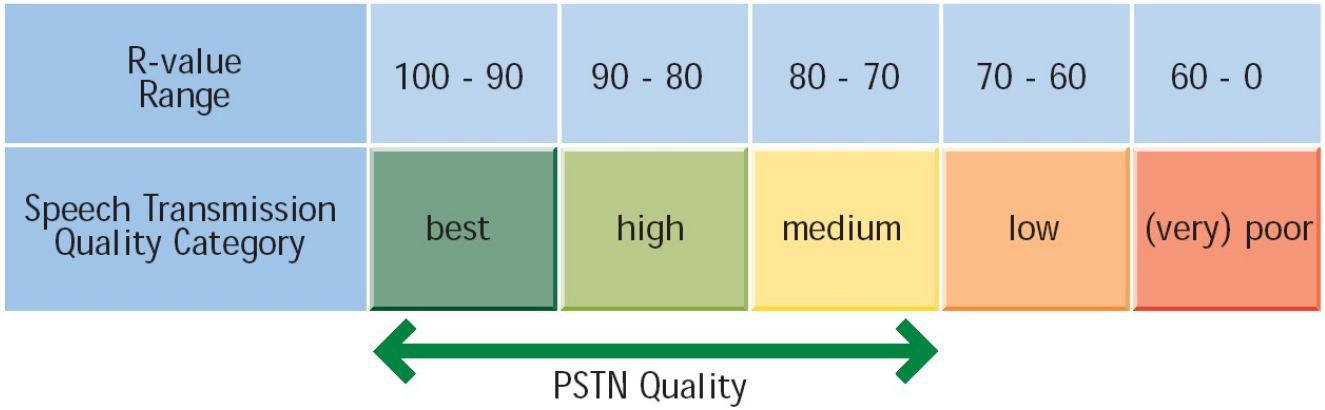
- Undistorted voice (G.711 codec).
- EL: Echo Loss measured in dB. The larger the EL, the lower the echo.
- Clearly, the higher the echo, the more the impact of the delay. Echo cancellation (increases the echo loss EL) allows the quality of the audio conversation to resist better to large delays.

E-model

- The E-model predicts the subjective quality of au audio conversation based on its characterizing transmission parameters. It combines the impairments caused by these transmission parameters into a rating R.
- The R scale was defined so that impairments are approximately additive in the R range of interest:

$$R = R_0 - I_d - I_s \ldots$$

R_0: Intrinsic quality, I_d: Impairment caused by delay, I_s: Impairment caused by external factors as noise around the talkers.



Intrinsic quality of main coders

Origin	Standard	Type	Codec bit rate (kbit/s)	Voice Frame(ms)	Look ahead(ms)	Algorithmic Delay(ms)	Ie	Instrinsic Quality
ITU-T	G.711	PCM	64	0.125	0	0.125	0	94.3
	G.726 G.727	ADPCM	16				50	44.3
			24				25	69.3
			32				7	87.3
			40	0.125	0	0.125	2	92.3
	G.728	LD-CELP	12.8	0.625	0	0.625	20	74.3
			16				7	87.3
	G.729(A)	CS-ACELP	8	10	5	15	10	84.3
	G.723.1	ACELP	5.3	30	7.5	37.5	19	75.3
		MP-MLQ	6.3				15	79.3
ETSI	GSM-FR	RPE-LTP	13	20	0	20	20	74.3
	GSM-HR	VSELP	5.6	20	0	20	23	71.3
	GSM-EFR	ACELP	12.2	20	0	20	5	89.3

Tolerable delays for different codecs

Origin	Standard	Codec Bit Rate (kbit/s)	M2E Delay Bound (ms)
ITU-T	G.711	64	400
	G.726 G.727	16	NA
		24	NA
		32	324
		40	379
	G.728	12,8	212
		16	324
	G.729(A)	8	296
	G.723.1	5,3	221
		6,3	253
ETSI	GSM-FR	13	212
	GSM-HR	5,6	180
	GSM-EFR	12,2	345

- The delay is tolerable if the rating R is above 70 (PSTN quality).
- NA: PSTN quality not attainable, the rating R is always below 70.

Alleviate queuing delay

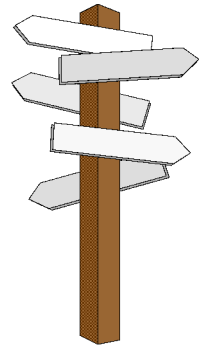
- Main responsible of queuing delay:
  - FIFO (First-In First-Out) policy implemented in Internet routers.
  - High load.
  - Audio packets wait as any other packet in routers.
  - Problem usually exacerbated at the edge of the network.
- Solutions:
  - Overprovisioning:

    Add more bandwidth to absorb the traffic and to make queuing delay disappear for everyone (this is almost the case nowadays, at least in the core of the network).

Alleviate queuing delay

- Give priority to audio packets over data packets in routers:
  - Mechanisms in routers: Priority queue, Weighted Fair Queuing, Class Based Queuing, Alternative Best Effort, etc.
  - Problems: Need major changes to the architecture of the Internet
    - Add resource reservation (bandwidth, buffering), and new scheduling policies into routers and domains.
    - Solutions to reserve across multiple domains (kind of Bandwidth Brokers) with all the complexity it induces (already hard to administrate one domain, how to do that across multiple domains !!).
    - Set up service level agreements with applications, monitor and enforce the agreements, charge accordingly.
  - There were efforts in this direction. Unfortunately, all failed ☹
    - IntServ: Reserve bandwidth everywhere (per connection state in routers).
    - DiffServ: Reserve at the edge of domains. Nothing in the middle.
    - Alternative Best Effort: Trade delay vs losses for audio applications in such a way that the concurrent traffic gets the same performance as before. but how to know it ???

Outline

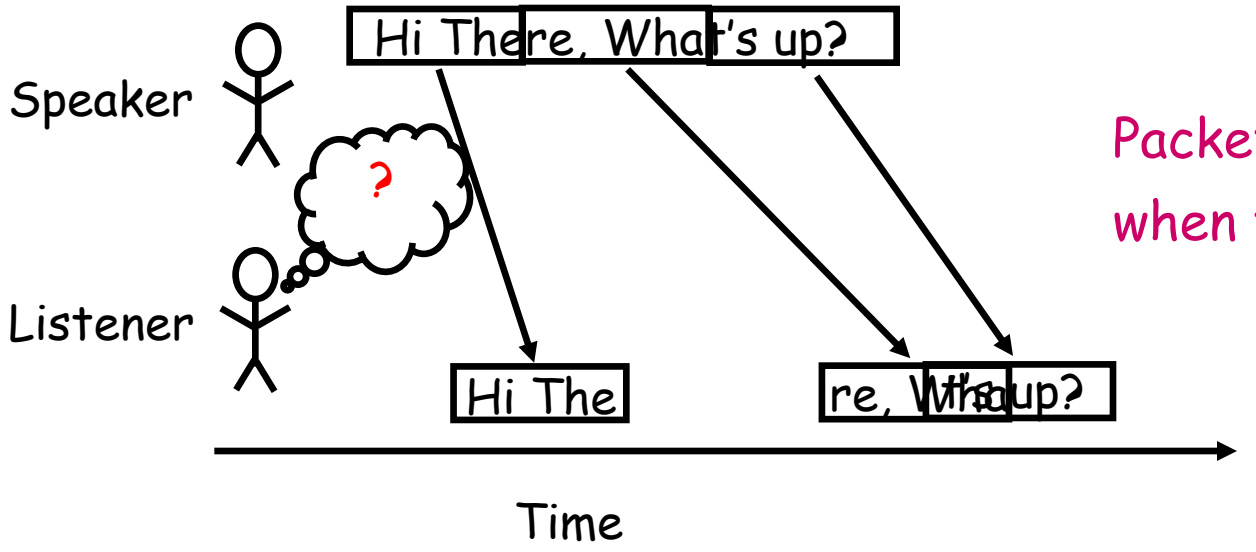


I- Transport of audio over the Internet

- Challenges: Delay, jitter, and losses
- Impact of delay and solutions
- Solve for jitter
- Impact of losses and solutions
- Rate adaptation: Equation-based congestion control

Problem of jitter

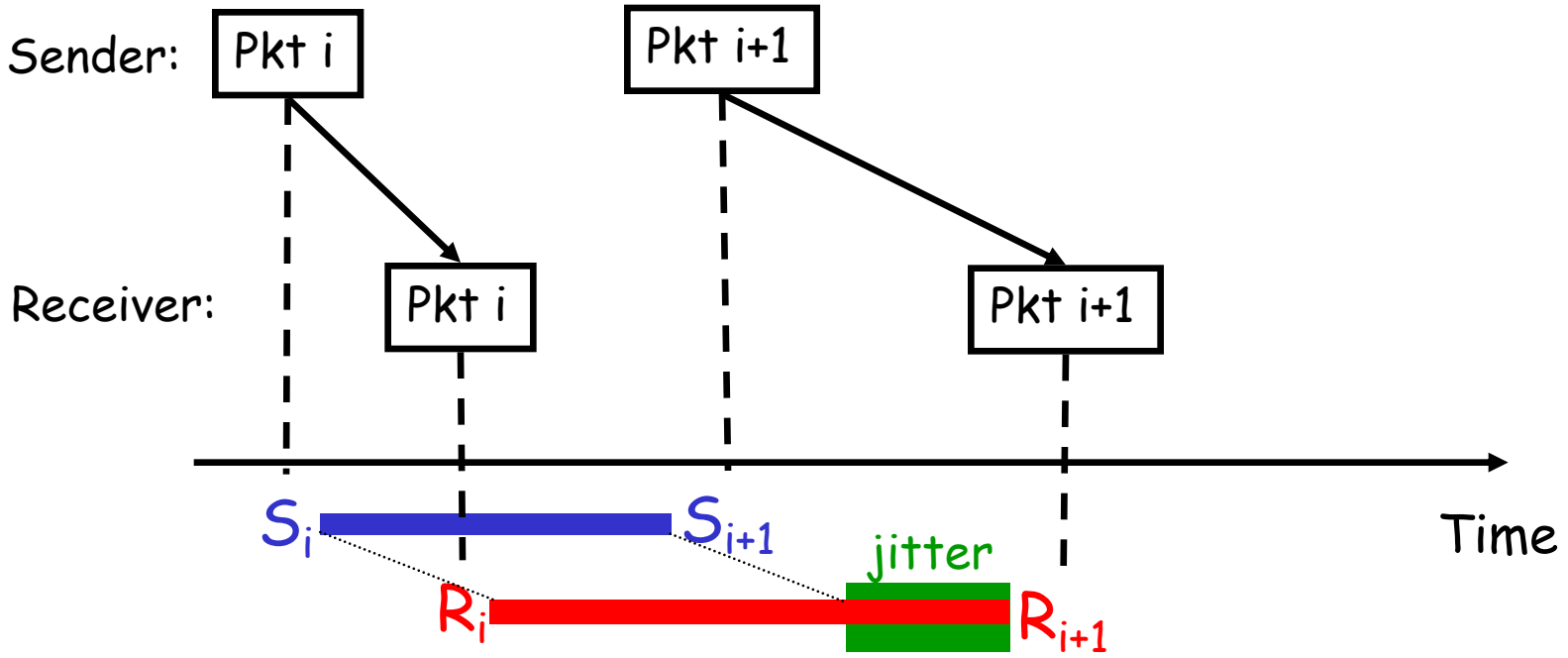
- Audio packets are sent at regular time intervals, e.g. one packet per 30ms.
- But the delay in the Internet is variable for different reasons:
  - Queuing, route changes, retransmissions at link layers, etc.
- So packets do not arrive at the receiver periodically as they left the sender. Moreover, they may arrive out of order, or may not arrive at all.



Packets cannot be played out when they arrive

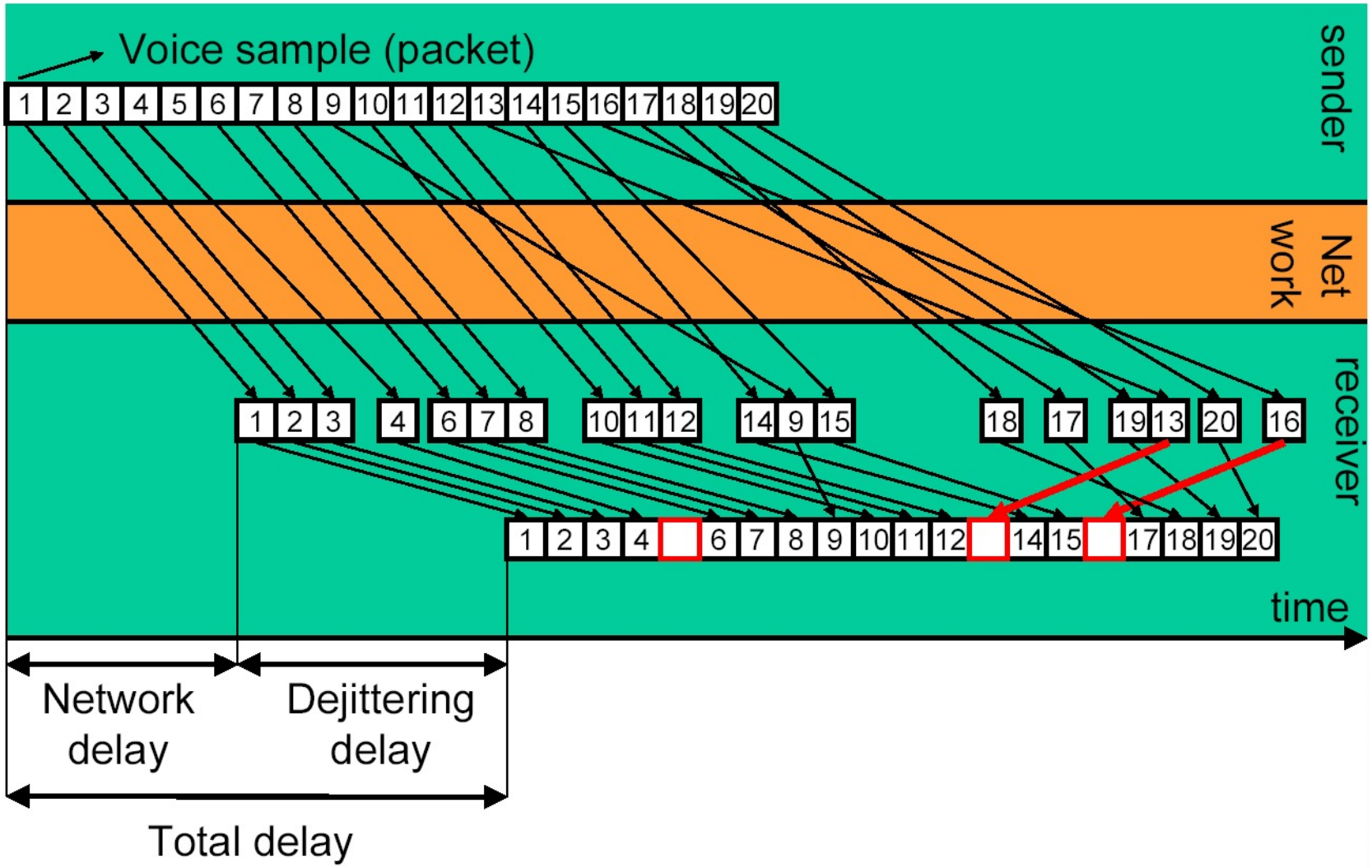
Problem of jitter

- A packet pair's jitter is the difference between the transmission time gap and the receive time gap



- Desired time-gap: $S_{i+1} - S_i$ Received time-gap: $R_{i+1} - R_i$
- Jitter between packets i and i+1: $(R_{i+1} - R_i) - (S_{i+1} - S_i)$

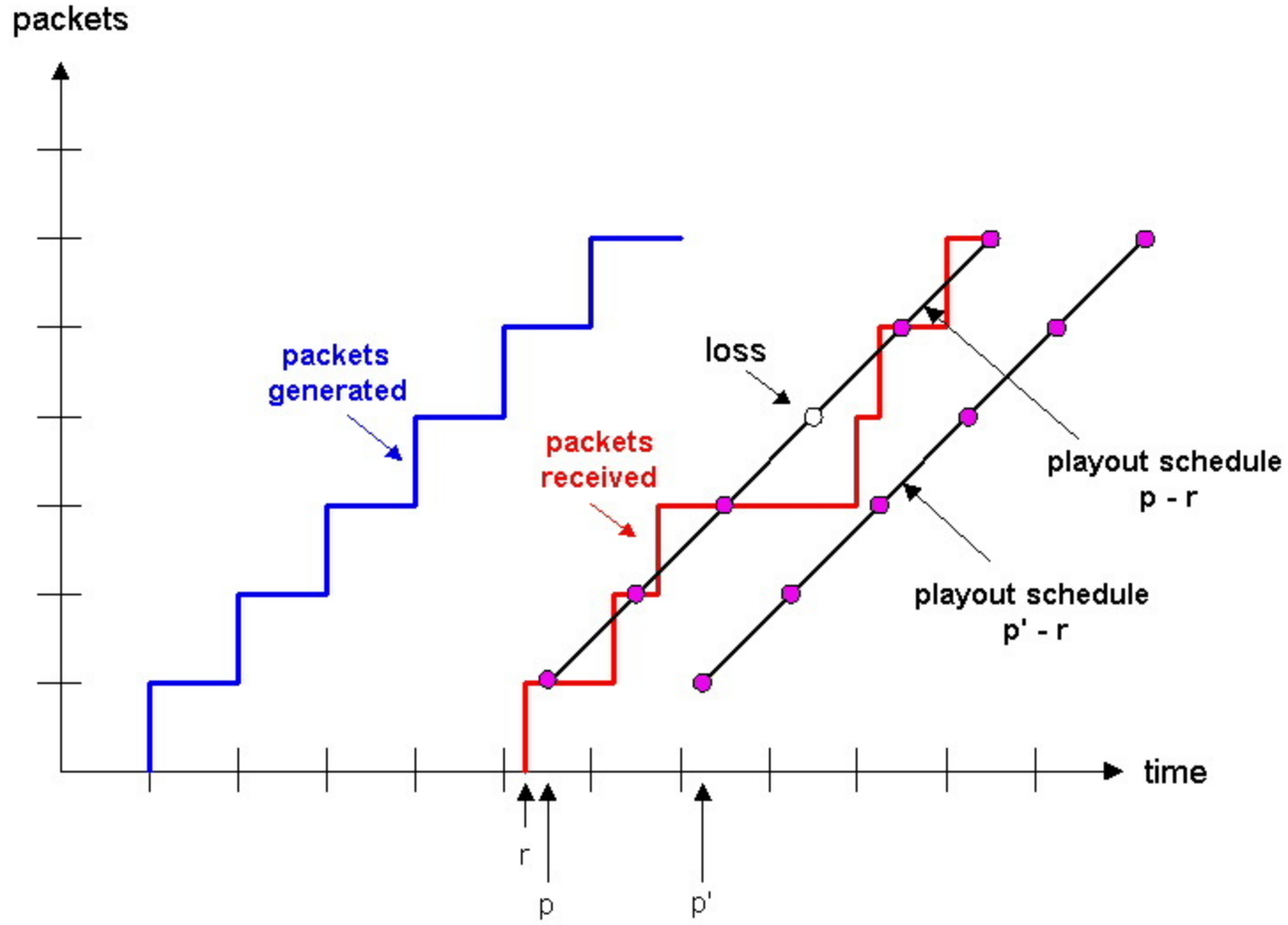
Solve jitter by delaying packets



Buffering: A remedy to jitter

- Delay playout of received packet i until time $S_i + C$ (C is some constant).
- If packet i arrives after its deadline, drop it, and wait for the next packet (too late to play out the packet).
- How to choose value for C?
  - Bigger jitter $\rightarrow$ need bigger C
  - Small C: More likely that $R_i > S_i + C \leftrightarrow$ missed deadline
  - Big C:
    - Requires more packets to be buffered.
    - Increased delay prior to playout.
  - Application timing requirement might limit C:
    - Interactive applications (IP telephony) cannot impose large playout delays
    - Non-interactive: More tolerant to delays, but still not infinite...

Playout tradeoff: Delay vs. loss



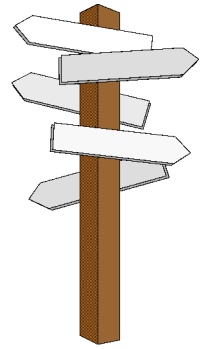
Playout tradeoff: Delay vs. loss

- In conclusion, the larger the playout delay, the lower the rate of packets that miss their deadlines.
- The opposite is also true.
- How to set it optimally ?
  - Audio quality is function of delay and loss, for example R(d,p).
  - Let p_0 be the loss rate of packets in the network.
  - The loss rate of packets after playout buffer $p = p_0+p_B$.
  - Delay after playout buffer is $d = d_0 + d_B$.
  - p_B function of d_B and function of the jitter.
  - The problem is then very simple: set d_B so that R(d,p) is maximum.

Adaptive playout

- When the network conditions change during an audio session (increase or decrease in the delay), the playout delay has to be changed as well.
- It is not easy to adapt the playout delay at every packet.
  - This can be done but requires a special and complex processing of packets at the receiver to stretch them or to lengthen them (audio rescaling).
- Adaptation can however be easily done between talkspurts:
  - A talkspurt is an activity period. Large silence periods separate talkspurts.
  - By adapting the playout delay at the beginning of talkspurts, we make small variations to the length of silence periods, but the user does notice that.

Outline

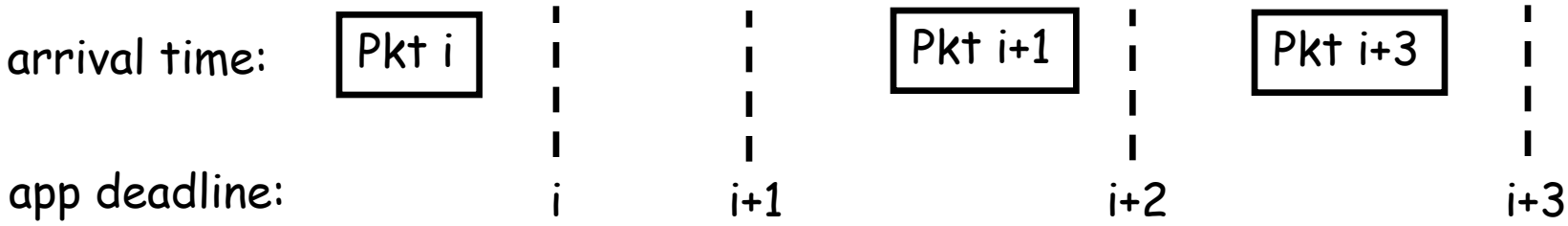


I- Transport of audio over the Internet

- Challenges: Delay, jitter, and losses
- Impact of delay and solutions
- Solve for jitter
- Impact of losses and solutions
- Rate adaptation: Equation-based congestion control

Packet loss

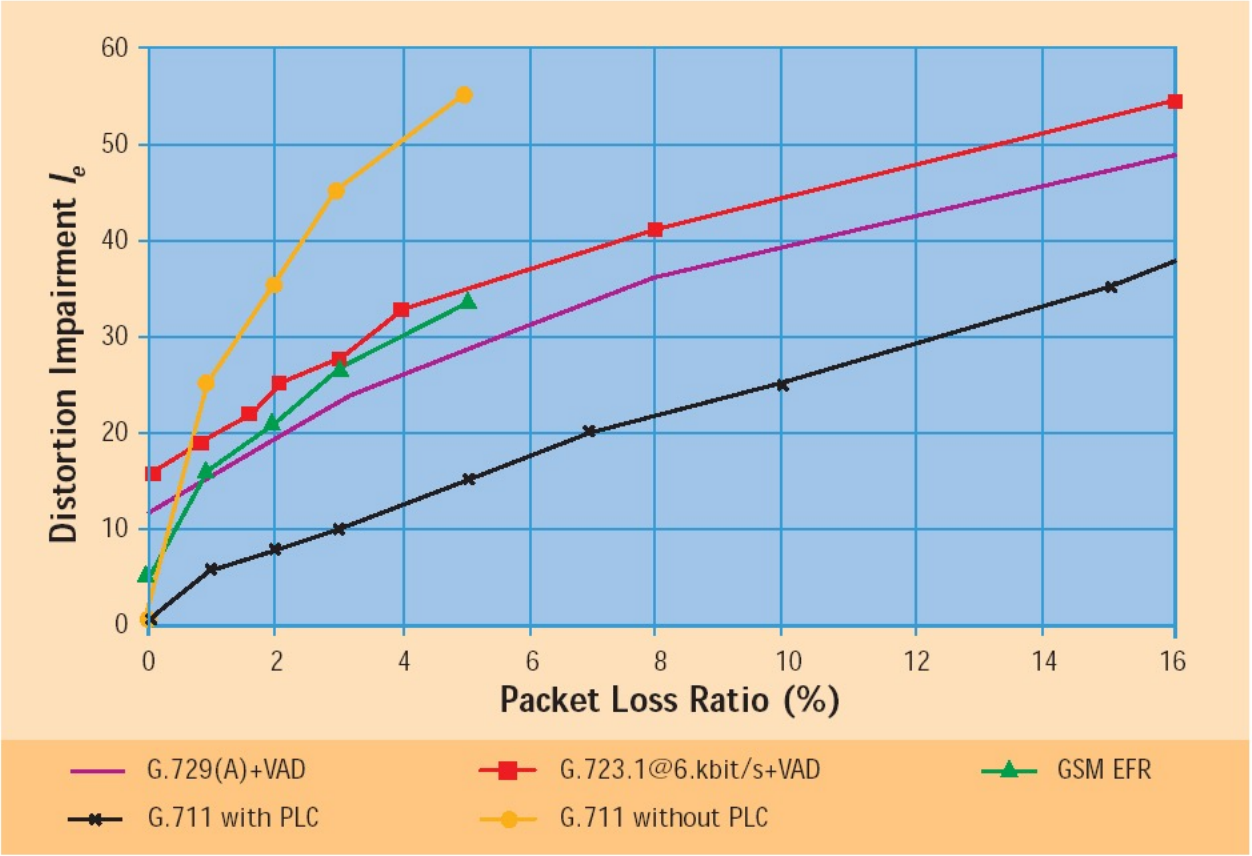
- Packets can be lost in the Internet for different reasons: Congestion in routers, transmission errors, routing loops, etc.
- Packets can also be excessively delayed making them unusable for the session.



usage status: …, i used, i+1 late, i+2 lost, i+3 used, ...

- Audio applications tolerate some losses. The maximum tolerate loss rate depends on the codec and on whether it implements loss concealment or not (substitute lost packets by white noise, replay past packets, etc.).

Impairment caused by losses (Emodel)



Distortion impairment as a function of the packet loss

VAD : Voice Activity Detection
PLC : Packet Loss Concealment
EFR : Enhanced Full Rate

Tolerable loss rates for some codecs

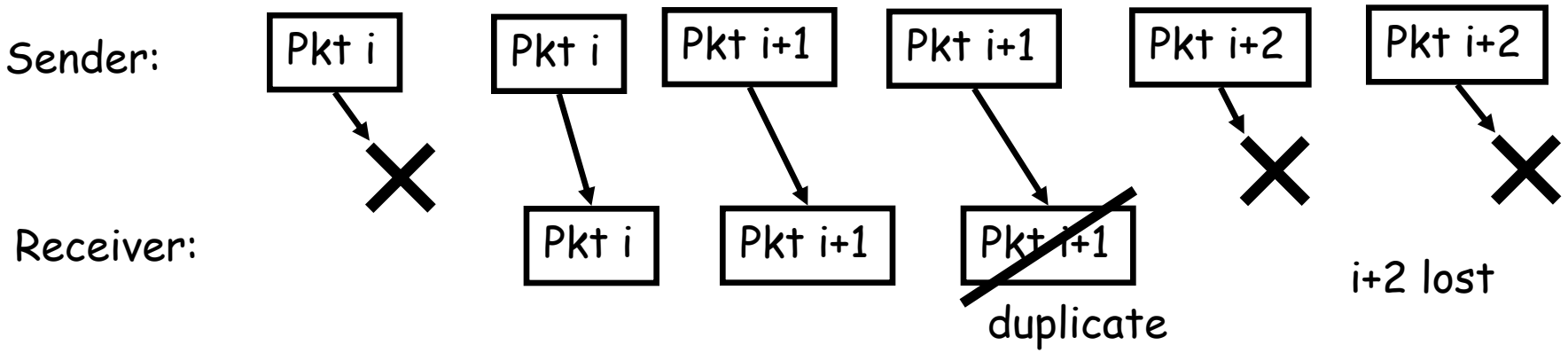
Origin	Standard	Codec Bit Rate (kbit/s)	PL Bound (%)
ITU-T	G.711 without PLC	64	1
	G.711 with PLC	64	10
	G.729(A)+VAD	8	3.4
	G.723.1@ 6.3 kbit/s+VAD	6.3	2.1
ETSI	GSM-EFR	12.2	2.7

Tolerable packet loss bounds for a mouth-to-ear delay below 150ms

PLC : Packet Loss Concealment
VAD : Voice Activity Detection

Reducing loss within application bounds

- Problem: Packets must be recovered prior to application deadline.
- Solution 1: Extend deadline, buffer correct packets at receiver, and use ARQ (Automatic Repeat Request: i.e., ACKs & NAKs) to recover from losses.
  - Recall: Unacceptable for many applications (e.g., interactive).
- Solution 2: Forward Error Correction (FEC) (Technically, we are using Erasure Codes, not FEC codes)
  - Send "repair" before a loss is reported.
  - Simplest FEC: Transmit redundant copies.



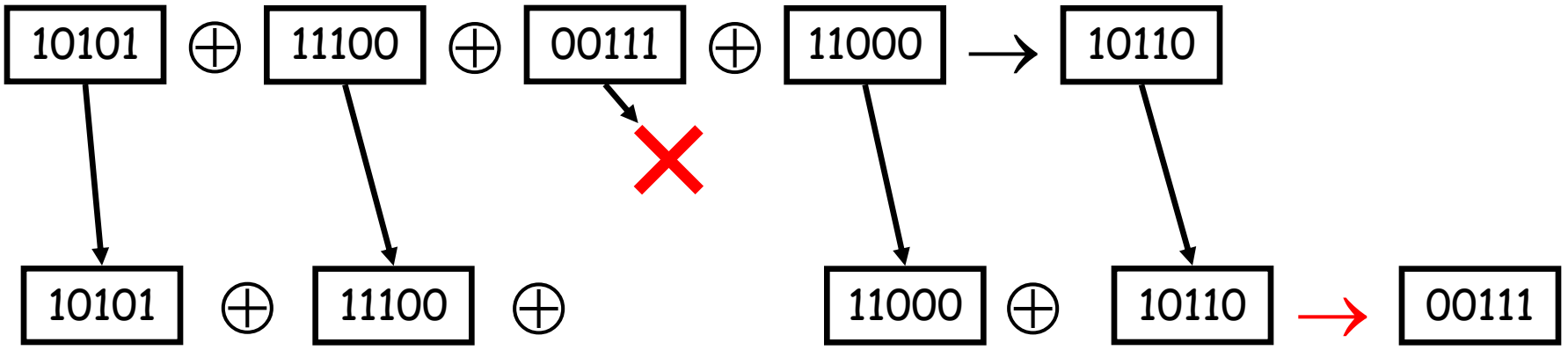
More advanced FEC techniques

- FEC often used at the bit-level to repair corrupt/missing bits (i.e., in the data-link layer):

- Here, we will consider using FEC (really Erasure Codes) at the packet layer (special repair packets):

A simple XOR code

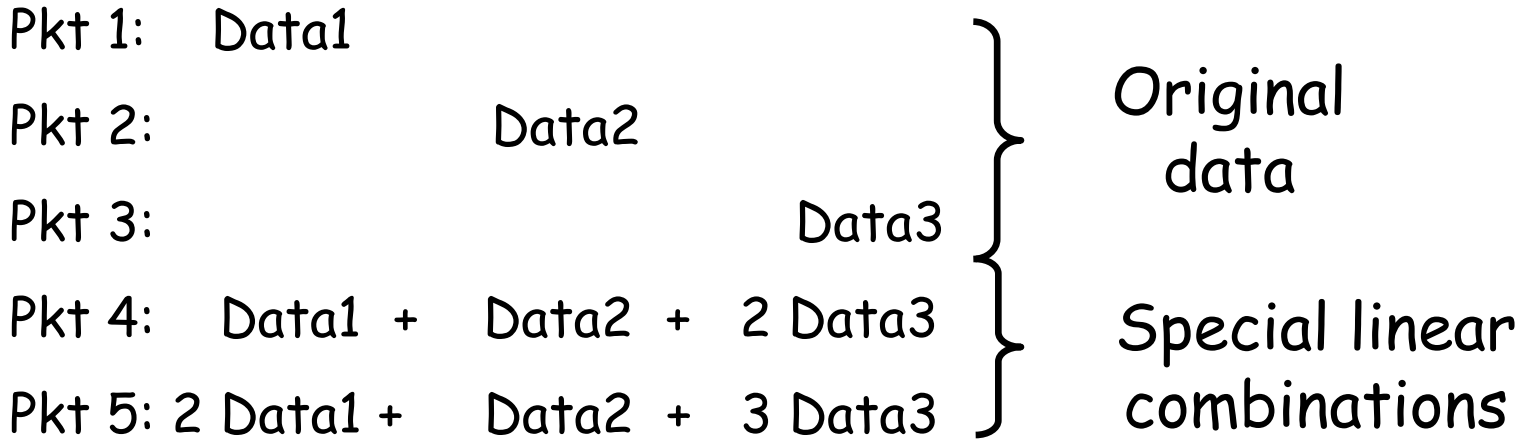
- For low packet loss rates (e.g. 5%), sending duplicates is expensive (wastes bandwidth).
- XOR code
  - XOR a group of data packets together to produce repair a packet.
  - Transmit data + XOR: Can recover 1 lost packet.



Reed-Solomon codes

- Based on simple linear algebra:
  - Can solve for "K" unknowns with "K" equations.
  - Each data packet represents a value.
  - Sender and receiver know which "equation" is in which packet (i.e., information in header).
  - Receiver can reconstruct "K" data packets from any set of "K" data + repair packets.
  - In other words, send "K" data packets + "R" repair packets, then if no more than any "R" packets are lost, then all data can be recovered.
- In practice and to reduce computation overhead, linear algebra is performed over finite fields that differ from the usual $\mathfrak{R}$, e.g, Galois field.

Reed-Solomon example over $\mathfrak{R}$



- Packets 1,2,3 are data (Data1, Data2, and Data3).
- Packets 4,5 are linear combinations of data.
- Assume 1-5 transmitted, packets 1 & 3 are lost:
  - Data1 = (2 * Pkt 5 - 3 * Pkt 4 + Pkt 2)
  - Data2 = Pkt 2
  - Data3 = (2 * Pkt 4 - Pkt 5 - Pkt 2)

More on Reed-Solomon

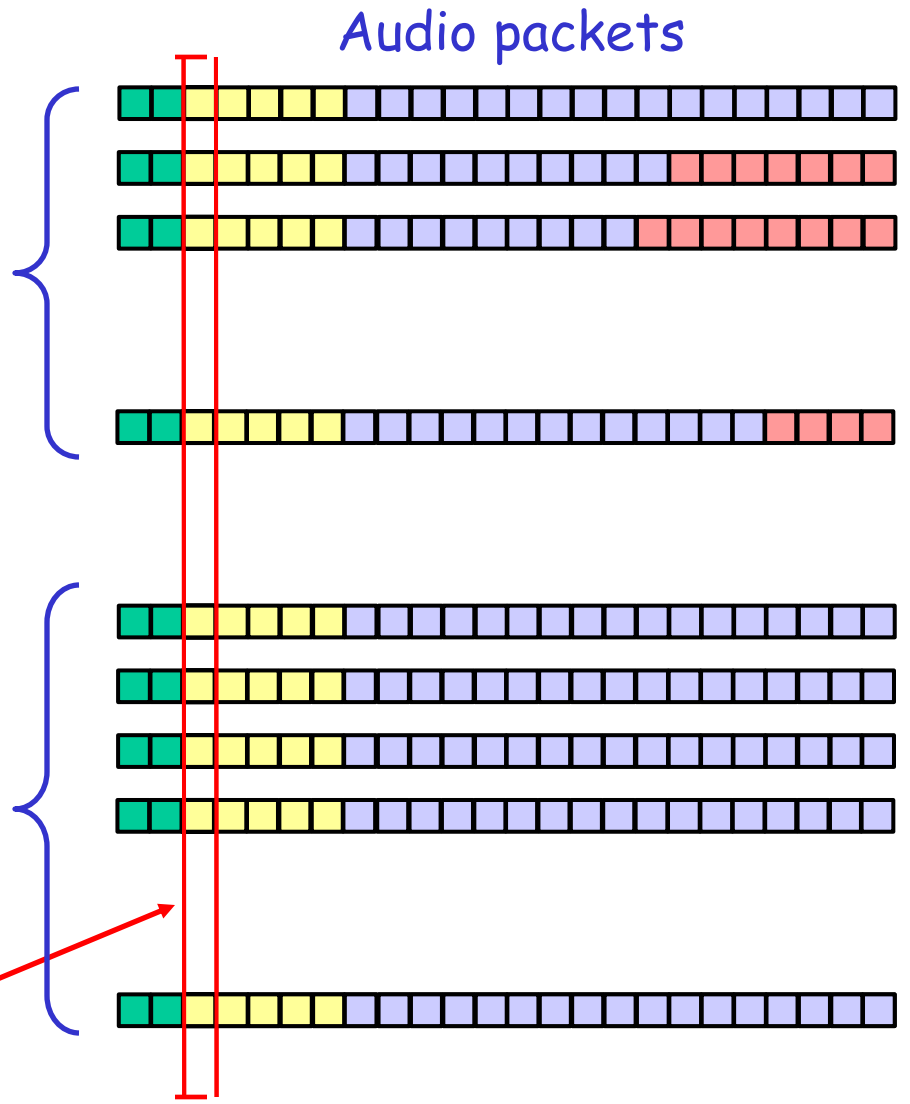
- A packet is a set of symbols (say bits).
- Take a finite field whose size should be larger than K+R.
- Find the coefficients in the field that form the set of independent linear equations.
- Multiply audio packets by the coefficients and form coded packets.
- The nice thing with finite fields is that coded packets are of the same length as original packets ! (not possible in $\Re$).
- One problem: complexity in case of large fields ☹ but in case of audio it is ok, there is no need for large fields.
- The large field problem is more for files (digital fountain).

Vertical implementation of RS

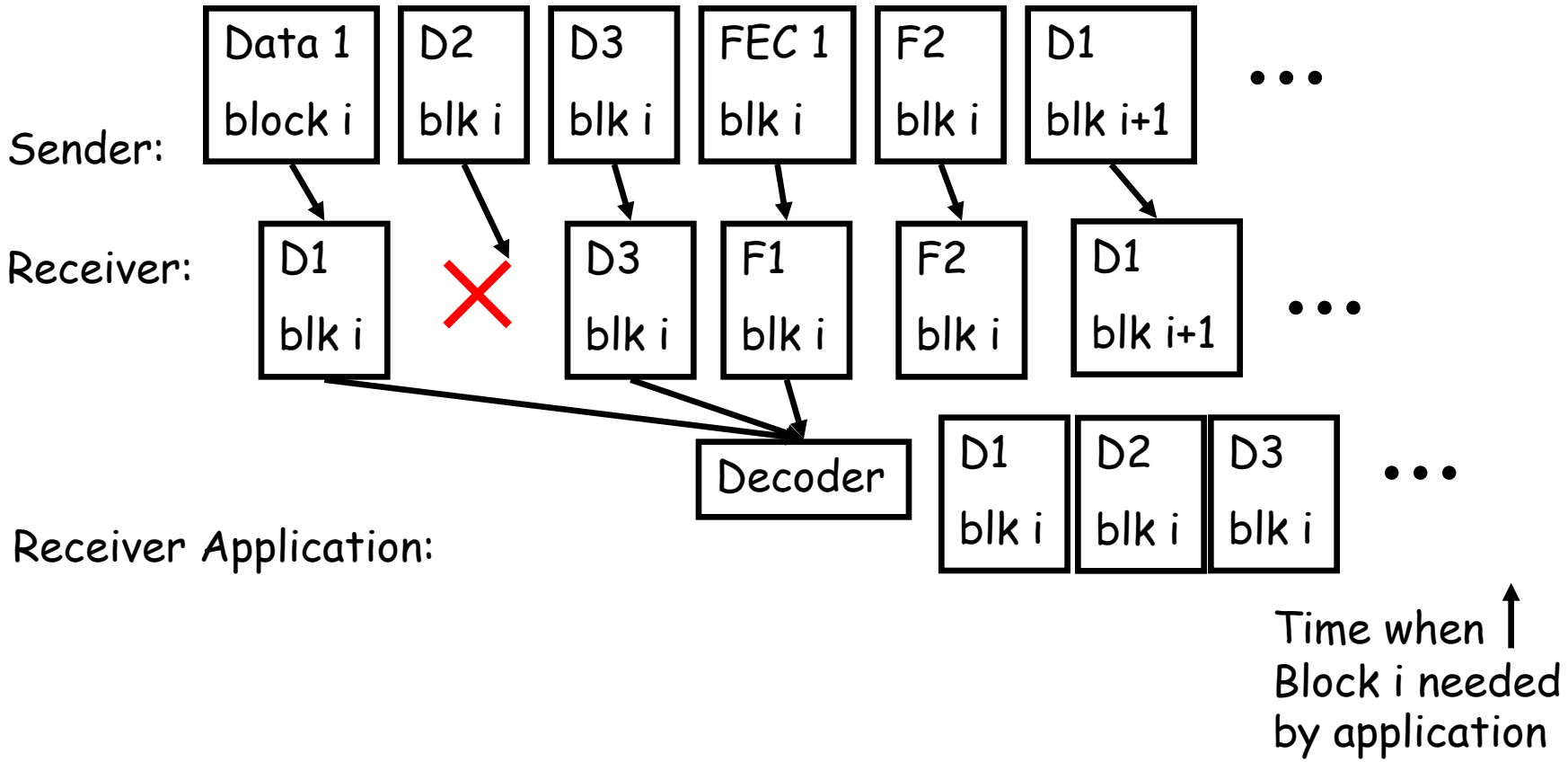
- FEC header
- Header of link-level units
- Payload of link-level units
- Virtual zero padding

FEC as implemented in the TRANSAT architecture:

- ALCATEL Space
- INRIA
- ENSICA
- University of Helsinki



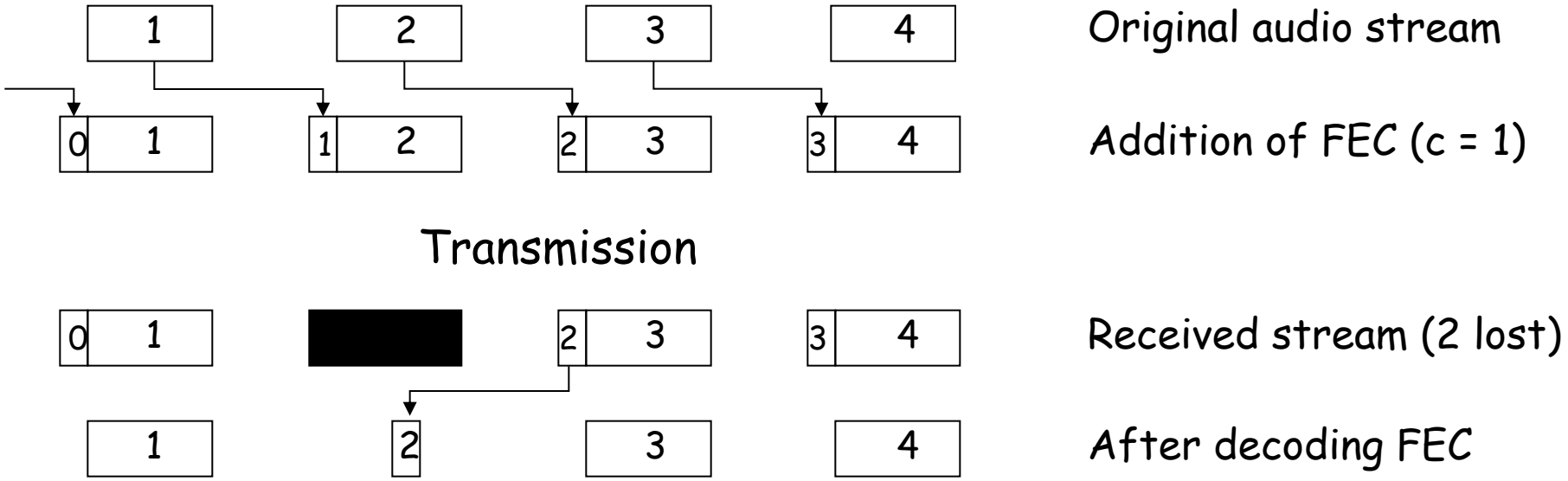
Using FEC for continuous-media



- Divide data packets into blocks.
- Send FEC repair packets after corresponding data block.
- Receiver decodes and supplies data to application before block i deadline

FEC via variable encodings

- Media-specific approach.
- Packet contents:
  - High quality version of media frame k.
  - Low quality version of media frame k-c (c is a constant called offset).
  - If packet i containing high quality frame k is lost, then one can use packet i+c with low quality frame k in place.



FEC tradeoff: Losses vs. rate

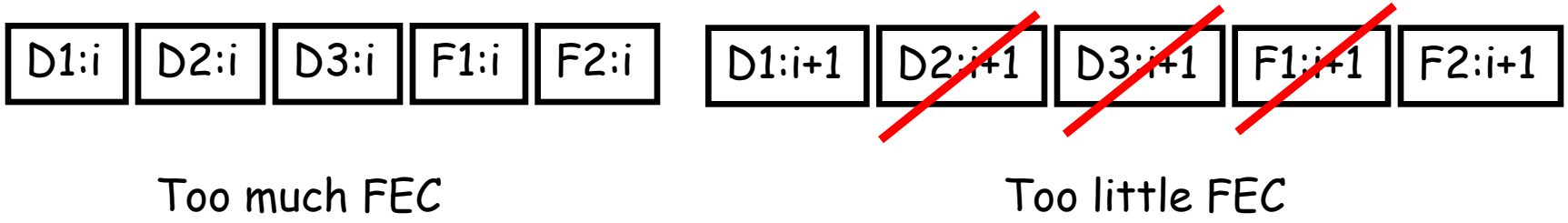
- Problems of FEC:
  - Consumes bandwidth, introduces processing overhead and additional delay.
  - Increases the volume of data injected into the network which increases the loss rate.
  - But, FEC is intended to reduce the loss rate ! So there is a tradeoff.
- For Reed-Solomon FEC
  - The loss rate usually increases with the addition of FEC.
  - It is not interesting to add FEC while keeping the original audio stream coded with the same rate.
  - However, the quality can be improved if we reduce the coding rate of the original audio stream and use the bandwidth we save to add FEC. FEC improves the loss rate in this case which improves the quality even though the coding rate decreases. The reason is that the audio quality is more sensitive to losses than to the coding rate.

FEC tradeoff: Losses vs. rate

- Concerning the FEC that uses a different coding:
  - It increases the loss rate before FEC decoding.
  - After FEC decoding, the loss rate decreases. Lost packets will be played out using a lower coding rate.
  - This again improves the quality since it is better to play out packets with low quality than to play nothing.
  - One must not add however FEC in large amounts, the quality may deteriorate since the loss rate may increase considerably (before FEC decoding).
- The problem in its general form is the following:
  - Given a budget of bits per second to use for audio and FEC.
  - And given a set of codecs.
  - What would be the best codec and the best FEC that lead to the best audio quality ?

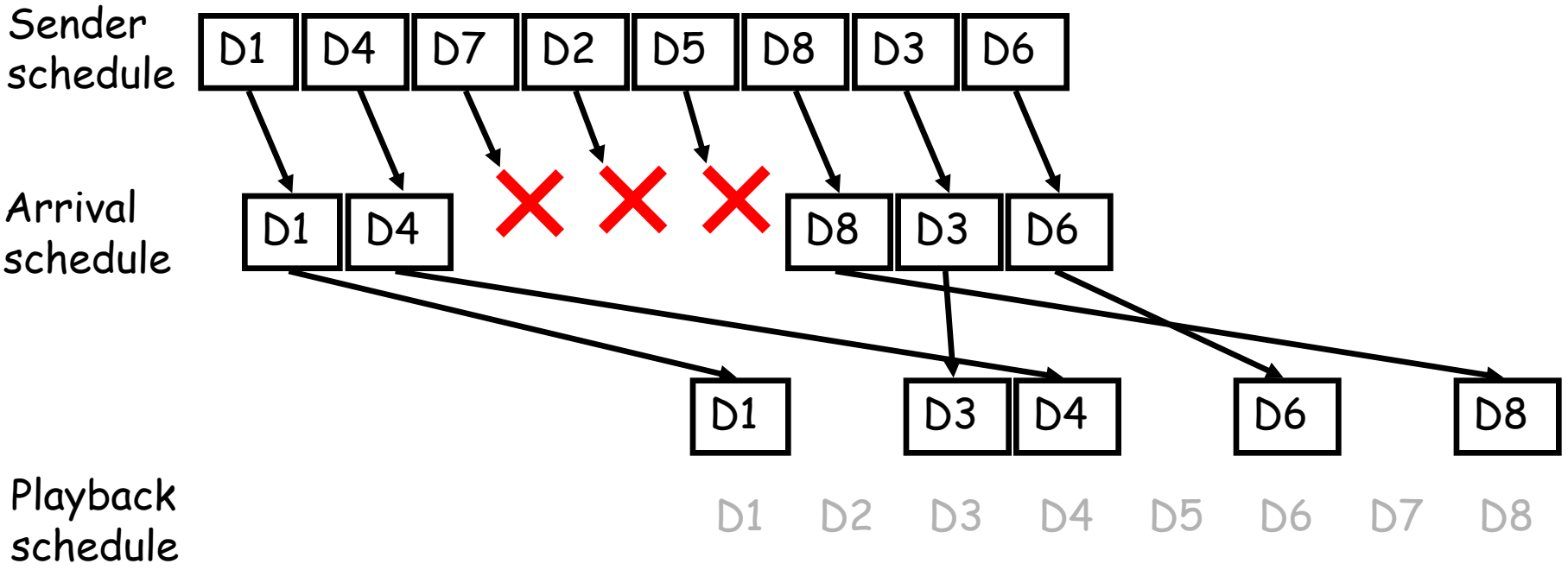
Bursty losses

- Many codecs can recover from short (1 or 2 packet) loss outages.
- Bursty losses (loss of many packets in a row) create long outages: Quality deterioration more noticeable.
- FEC provides less benefit in a bursty loss scenario:
  - Lot of FEC is needed, which is not efficiently used most of the time.

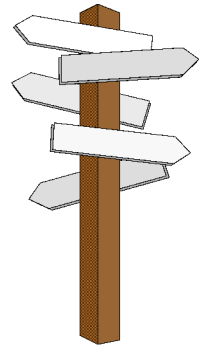


Interleaving

- To reduce effects of burstiness, reorder packet transmissions
- Drawback: Induces buffering and playout delay



Outline



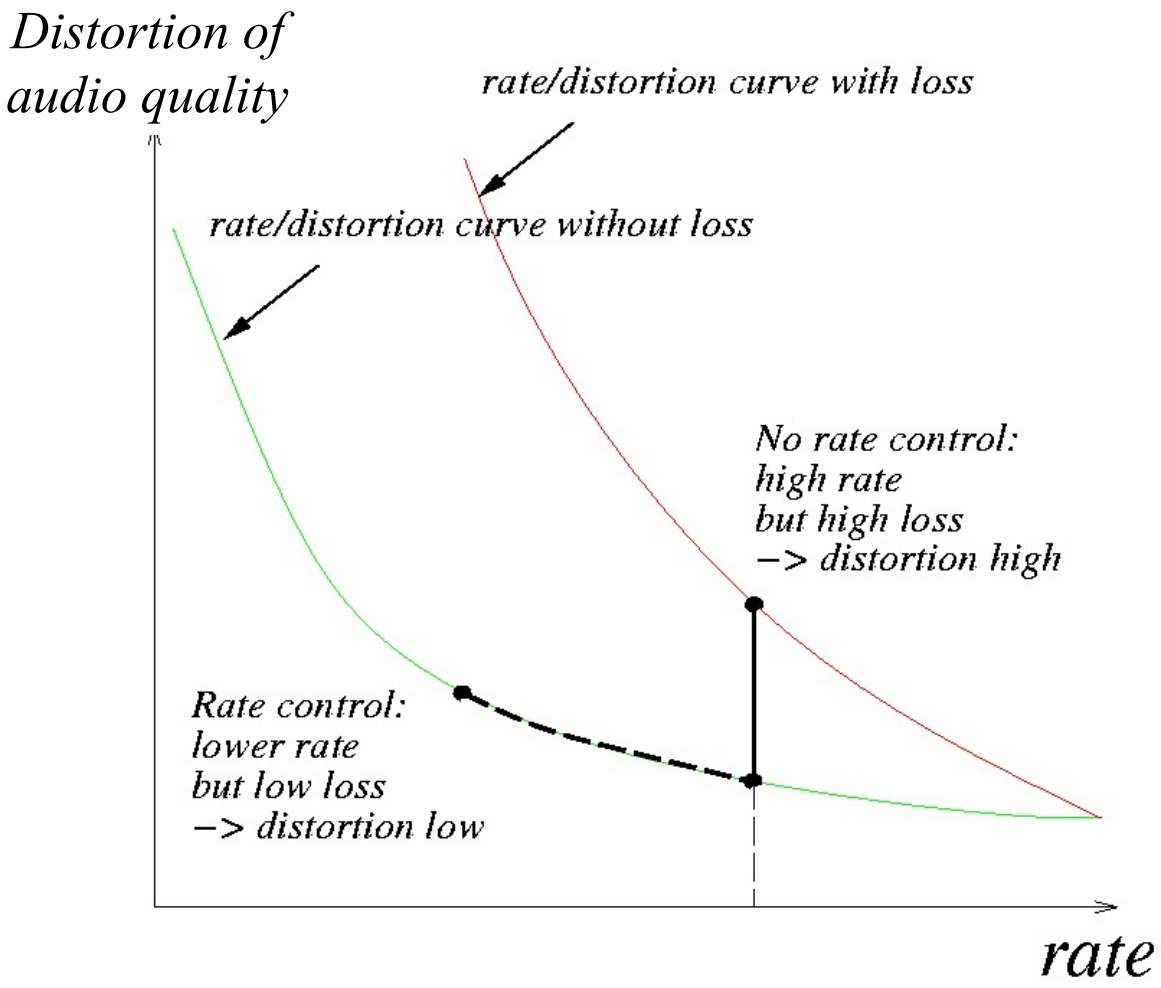
I- Transport of audio over the Internet

- Challenges: Delay, jitter, and losses
- Impact of delay and solutions
- Solve for jitter
- Impact of losses and solutions
- Rate adaptation: Equation-based congestion control

Congestion control for audio applications: TCP-friendliness

- An audio application shares the network with other applications mostly using the TCP protocol. The TCP protocol ensures that:
  - All applications share fairly the available resources.
  - Applications adapt their rates so as to avoid network congestion.
- There is a common interest that audio applications adapt their rates in the way TCP does:
  - Social gain:
    - An audio application that does not adapt its rate may harm the other applications using TCP (unfairness in the distribution of network resources).
    - Non-responsive audio applications may drive the network into congestion (high loss rate).
  - Individual gain: When implementing congestion control, audio applications will see their loss rate decreasing which is essential for good quality.

Individual gain from rate control



Equation-based congestion control

- Audio applications (especially interactive ones) cannot use the TCP protocol since the rate of TCP changes quickly.
  - Large playout delay is required to absorb these variations.
  - Delay variations exacerbated by retransmissions.
- Problem: Come up with another congestion control algorithm than TCP, where the rate changes slowly, but where the average rate is the same as that of a TCP connection running in the same network conditions.
- Mostly used solution: Equation based-congestion control (e.g. TFRC)
  - Let X be the average rate of a TCP connection as a function of
    - p: The packet loss rate.
    - RTT: The average round-trip time.
  - Estimate p and RTT during an audio session.
  - Set the transmission rate to X.

TCP average rate

Accounts for timeouts (not slow start !) and receiver window.

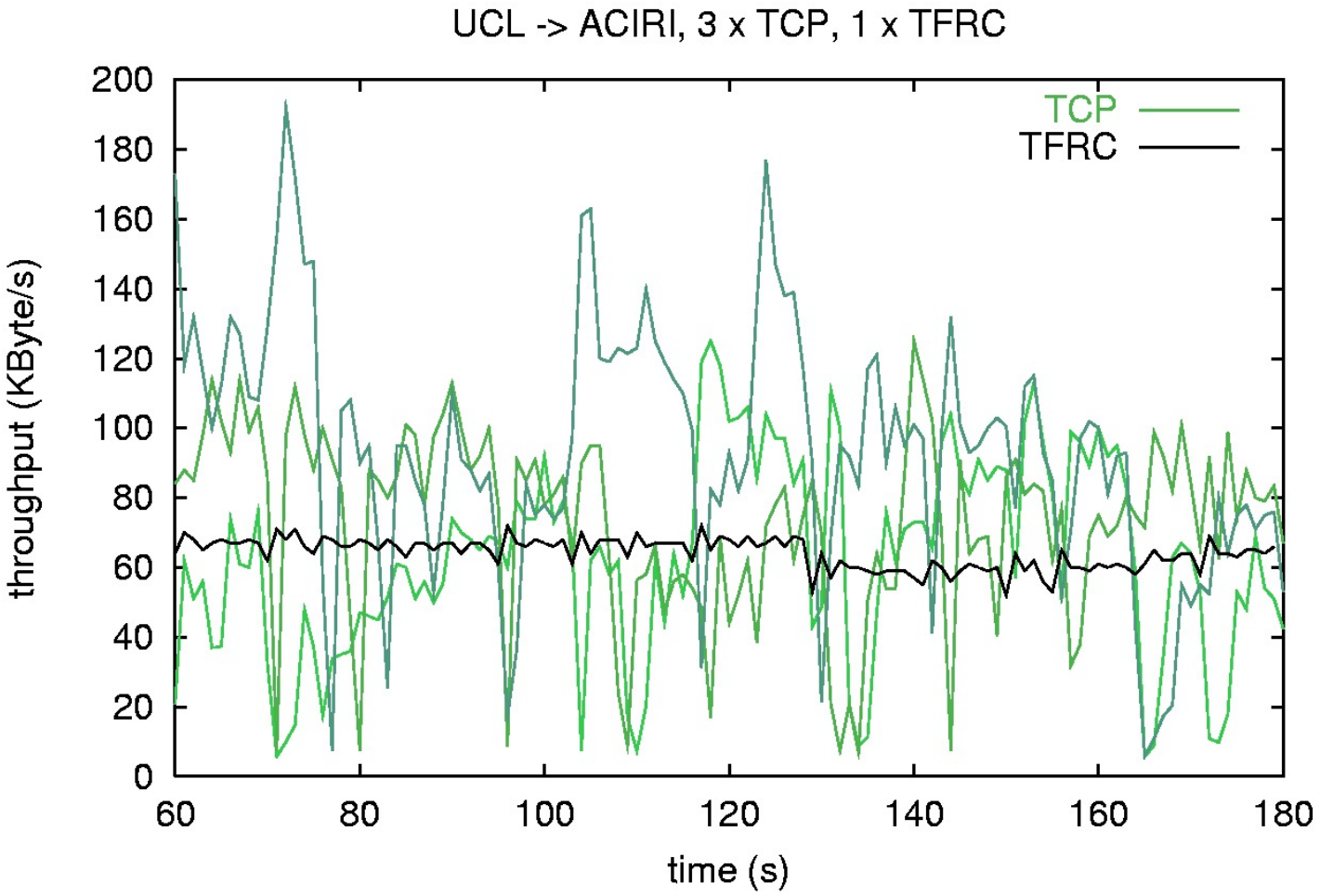
TCP average rate in packets/s:

$$X \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p\left(1 + 32p^2\right)}\right)$$

- W_{max} – maximum congestion window size (receiver window).
- T_0 – Length of timeout usually approximated by 4 RTT.

Padhye J., Firoiu V., Towsley D., and Kurose J., "Modeling TCP Throughput: a Simple Model and its Empirical Validation", in Proceedings of ACM SIGCOMM, August 1998.

Smoother throughput than TCP



TFRC: TCP-Friendly Rate Control protocol, RFC 3448

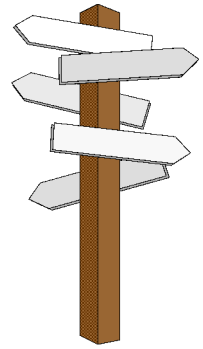
Plenty of other solutions

- Binomial TCP: increase the window as W^{-K} and decrease it as W^{L}.
  - Choose K and L so that to be TCP friendly.
  - They should satisfy K + L = 1.
  - Regular TCP is a particular case: K = 1 and L = 0.
- RAP: Rate Adaptation Protocol
  - Simulates TCP window adaptation at the source.
  - Adapts the transmission rate less often then the Round-Trip Time based on the value of the "simulated" congestion window.
- TEAR: TCP Emulation at Receivers
  - Do the same job as RAP, but at the receiver side.
  - Send back a smoothed version of the window.
- ATTENTION: In TFRC
  - The loss rate is calculated at the receiver, sent back to the sender.
  - The sender estimates the RTT and sets the TCP-friendly transmission rate.
- There is a serious problem in Multicast for all these protocols ……

Combined optimization of FEC and playout delay

- Congestion control for audio gives us the rate to use X.
- And we have at this rate some distribution of end-to-end delay D and of packet loss in the network p.
- So given this information, one can look at the optimal FEC code and the optimal playout delay to use while respecting the rate constraint X.
  - The quality is a function of the coding rate, the loss rate, and the delay.
  - Using FEC reduces the rate of the original audio stream and increases the delay, but reduces the loss rate (could be better).
  - The playout delay absorbs the jitter, reduces the losses, but increases the delay. Can this loss reduction be absorbed by FEC??
- This combined optimization is an interesting research topic …

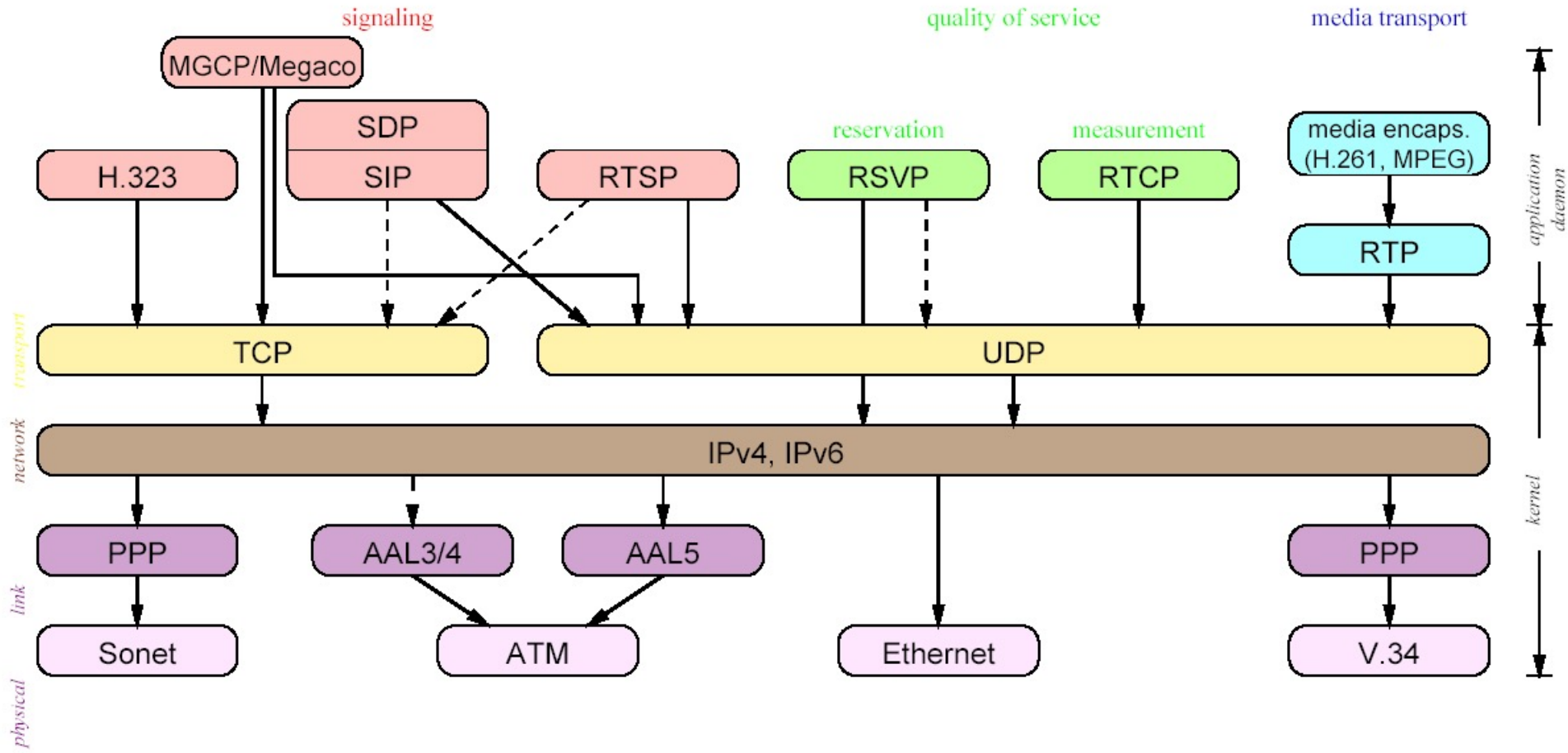
Outline



I- Internet audio protocols

- Multimedia protocol stack
- RTP/RTCP
- Session Initiation Protocol (SIP)

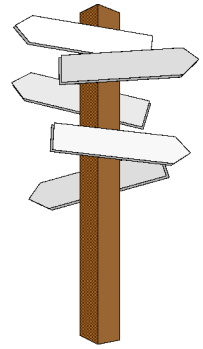
Internet multimedia protocol stack



Internet multimedia protocol stack

- Audio transport protocols: Provide the set of functions necessary to carry the audio frames (media identifier, timestamps, numbering, etc).
  - Real-Time Transfer Protocol (RTP) is a typical example and will be described.
- Signaling protocols: Provide the set of methods for the establishment of the audio session and the set of entities to provide an audio service over the Internet (gateways, directories, address translation, authentication).
  - Two typical examples: Session Initiation Protocol (SIP), H.323.
  - SIP will be described in this course. No H.323 …
- Quality of service protocols: The set of functions and entities that tune the Internet so as to provide a better quality to the audio conversation than the classical best effort service.
  - DiffServ, IntServ, RSVP for bandwidth reservation, etc. All not described.

Outline

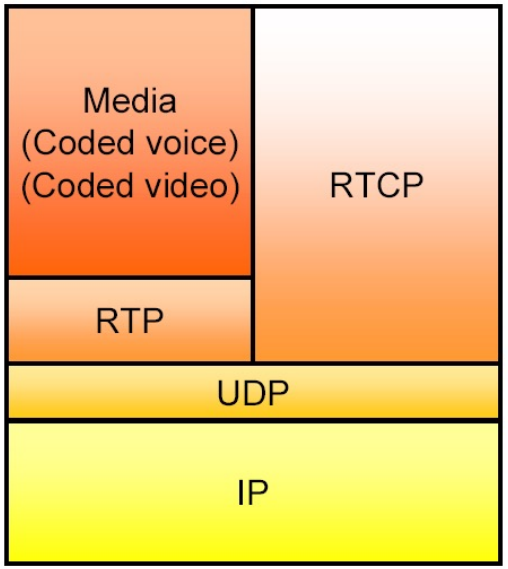


I- Internet audio protocols

- Multimedia protocol stack
- RTP/RTCP
- Session Initiation Protocol (SIP)

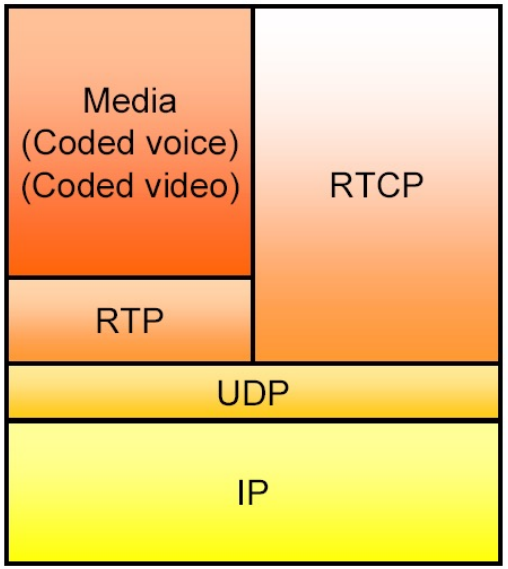
RTP/RTCP

- General purpose Real-Time Multimedia protocol (also video & others).
  - Scalable to large sessions (many senders, receivers).
  - Used by most of Internet audio tools: NeVot, Rat, Vat, Netmeeting, etc.
- Session data sent via RTP (Real-time Transfer Protocol).
- RTP components / support:
  - Sequence number and timestamps.
  - Intra-media synchronization (resequence packets, detect losses).
  - Inter-media synchronization (audio and video).
  - Unique source/session ID (SSRC or CSRC).
  - Encryption.
  - Payload type info (codec).
  - Aggregation of audio streams.
  - Translation of audio streams (change coding in the middle of the network).

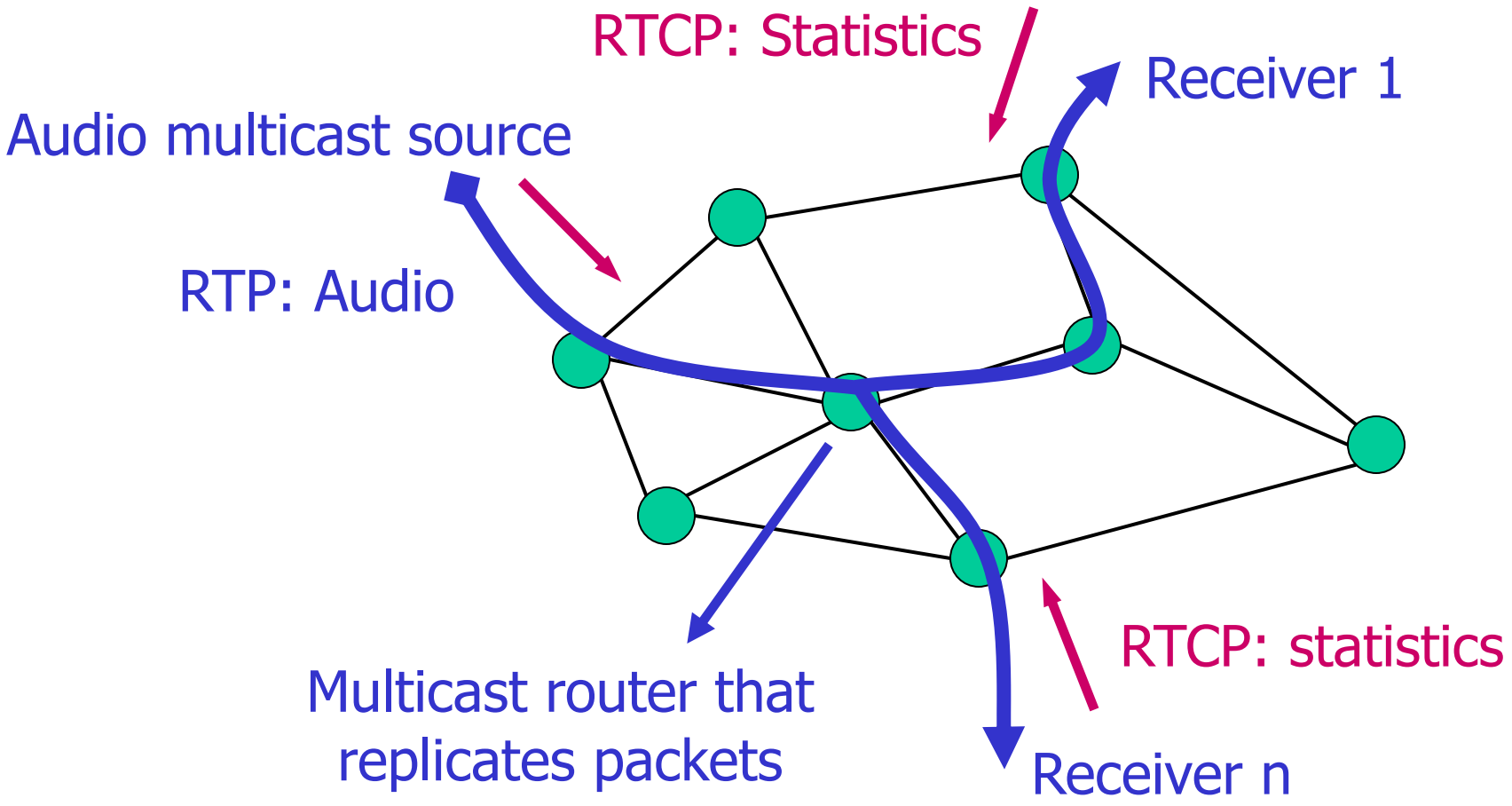


RTP/RTCP

- Receiver /Sender session status transmitted via RTCP (Real-time Transfer Control Protocol)
  - Timestamps in sender/receiver reports that allow the computation of the round-trip time.
  - Last sequence number received from various senders.
  - Observed loss rates from various senders.
  - Observed jitter info from various senders.
  - Member information (canonical name, e-mail, etc.).
  - Control algorithm (limits RTCP transmission rate).
  - Detection of collision in case two sources choose the same sequence number.



RTP/RTCP supports audio multicast

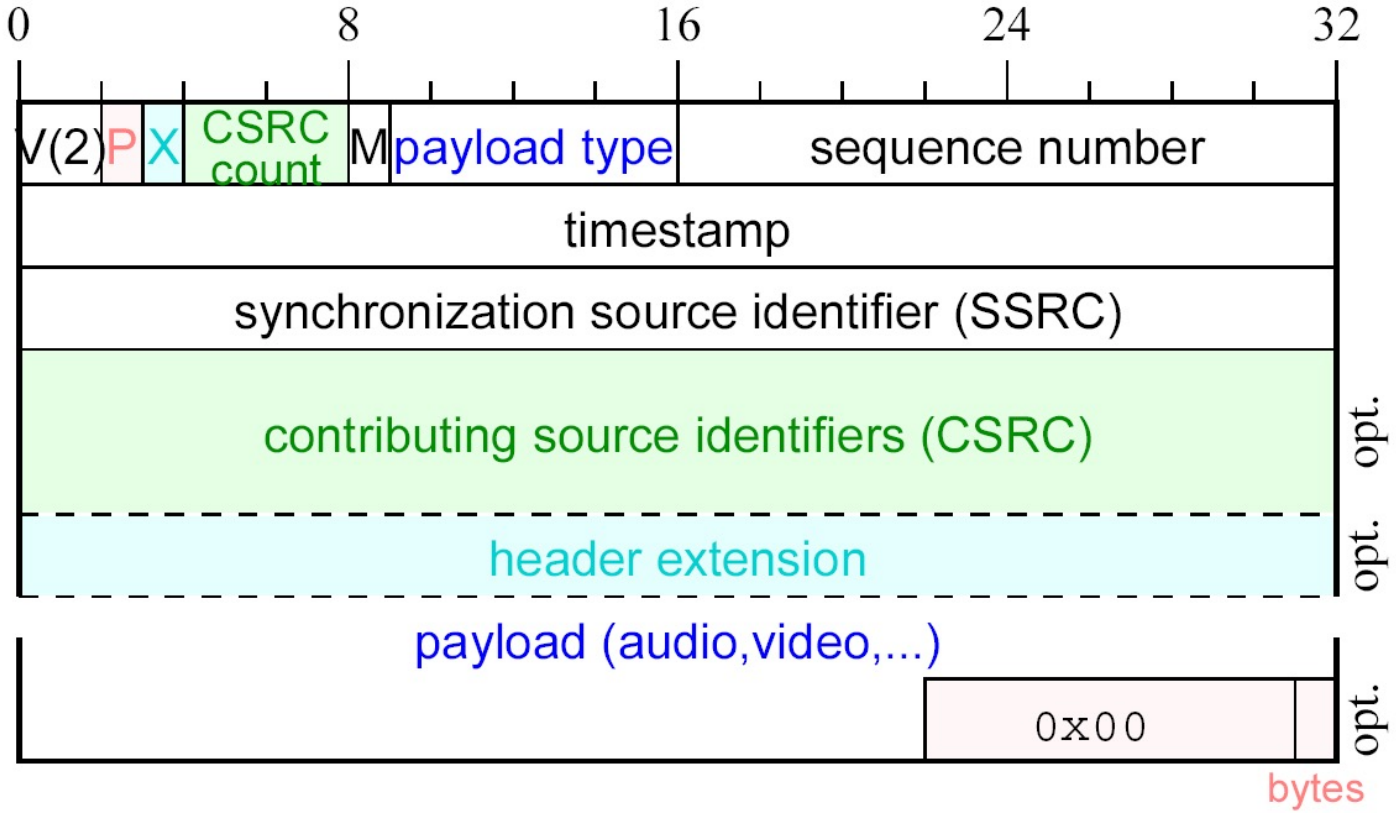


RTP/RTCP: More details

- All of a session's RTP/RTCP packets are sent to the same multicast group (by all participants):
  - All RTP packets sent to some even-numbered port, 2p.
  - All RTCP packets sent to port 2p+1.
- Only data senders send RTP packets, they are received by all participants. We distinguish sources based on their identifiers.
- All participants (senders/receivers) send RTCP packets.

Real-Time Protocol (RTP)

- Provides standard packet format for real-time application.
- Typically runs over UDP.
- Specifies header fields below.



RTP

- P: Last byte has padding count (for encryption).
- M: Marker bit; the current frame is the start of a talkspurt (playout delay adjustment).
- Payload Type: 7 bits, providing 128 possible different types of encoding, e.g. PCM, MPEG2 video, etc.
- Sequence Number: 16 bits; used to detect packet loss and to resequence packets at the receiver.
- Timestamp: 32 bytes; gives the sampling instant of the first audio byte in the packet;
  - Used to remove jitter introduced by the network (information required by the playout delay adaptation mechanism).
  - Also used for inter-media synchronization (audio and video session).

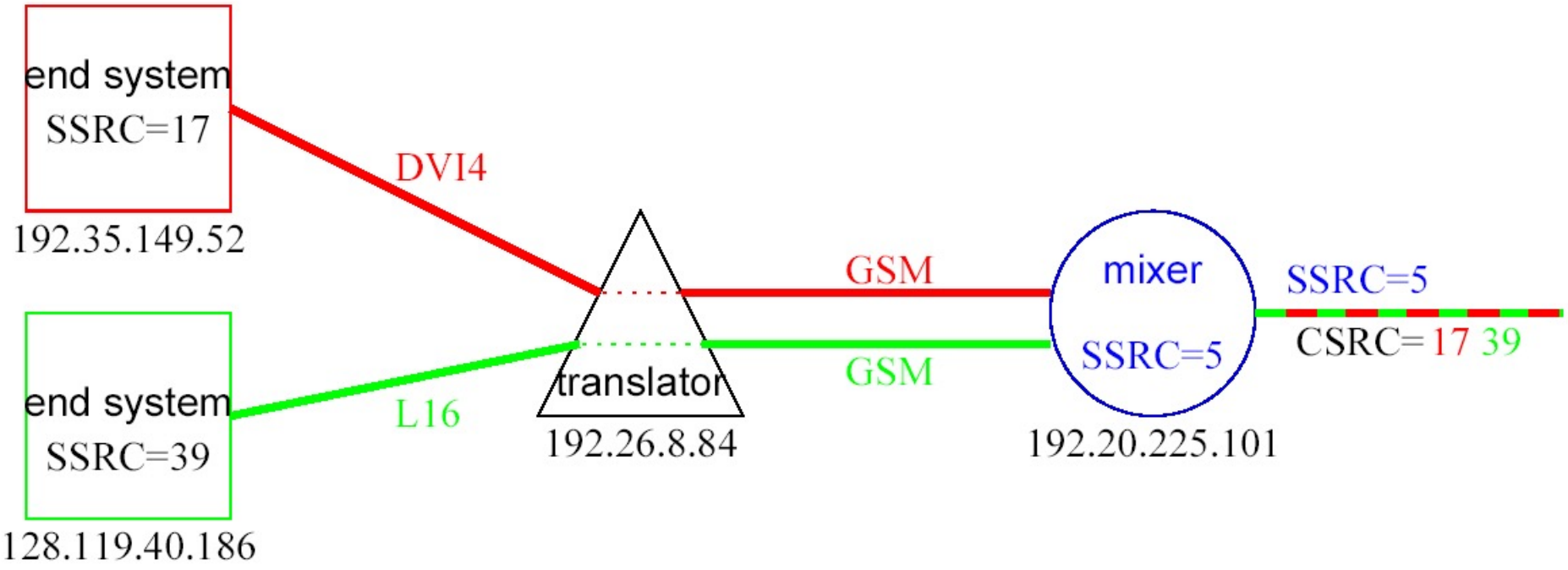
RTP

- Synchronization Source identifier (SSRC): 32 bits; an ID for the source of a stream; assigned randomly by the source.
  - In case a source picks a ID which already exists, it detects that, announces the death of the old ID and picks a new one.
  - The SSRC helps to distinguish an audio source within a session.
- Contributing Source identifiers (CSRC): Variable length field; used when multiple frames are multiplexed within the same RTP packet. It will include the SSRCs of the sources of all frames. The field CSRC count will indicate the number of multiplexed frames.
  - The SSRC of the aggregate packet will contain the ID of the entity that did the aggregation, i.e. the mixer.
- Payload: One or multiple audio frames.

Comments about RTP

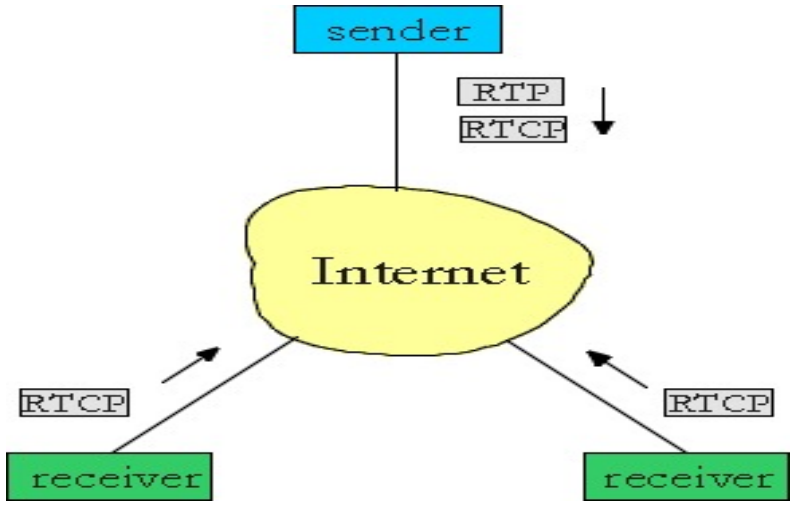
- RTP does NOT provide Quality of Service, it only provides a header that contains information about the content of an audio packet.
  - RTP can be seen as an enhanced version of UDP adapted to audio applications.
- Congestion control is implemented ABOVE RTP.
  - The rate of the congestion control mechanism allows to choose the codec and the amount of FEC (an optimization problem).
- RTP alone is not enough to provide an audio service worldwide. A signaling protocol is required, e.g. SIP, H.323.
  - Directories, authentication, billing, user profiles, gateways with PSTN.

Mixer / Translator



RTP Control Protocol (RTCP)

- Protocol specifies report packets exchanged between sources and destinations in an audio session.
- Four reports are defined: Receiver reception, Sender, Source description, and explicit leave (BYE).
- Reports contain statistics such as the number of packets sent, number of packets lost, inter-arrival jitter, etc.
- Used by application to modify sender transmission rates and for diagnostics purposes.



RTCP report types

- Sender report (SR):
  - Number of bytes sent (the receiver can then compute the receiving rate).
  - Universal time (allows the receiver to synchronize among senders).
- Reception report (RR):
  - Number of packets received.
  - Estimates of the average packet loss rate and of the jitter.
  - The timestamp of the SR packet to which the RR corresponds; this allows the sender to compute the round-trip time (estimate the average and variance).
- Source description (SDES): Name, email, location, . . .
  - CNAME (canonical name = user@host) identifies user across media.
- Explicit leave (BYE): Sent when a participant leaves the session.
  - Also used when there is a collision of IDs; the participant leaves then joins.

RTCP congestion control

- Problem: At which frequency reports have to be sent ?
  - Ideally, the best thing is to send lot of reports.
  - But this creates congestion in the network if the number of participants is very large.
- Simple rule: A session's aggregate RTCP bandwidth usage should be 5% of the session's RTP bandwidth:
  - 75% of the RTCP bandwidth goes to the receivers.
  - 25% goes to the senders.
- Reports are periodically sent every time T, with the time T adapted as a function of the number of participants in the audio session.

RTCP congestion control

- For receivers, set the period T to:

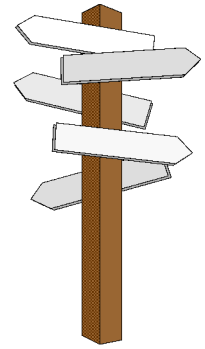
$$T = \frac{\text{\# of receivers}}{0.75 \cdot 0.05 \cdot \text{session bw}} \cdot \text{avg. RTCP packet size}$$

- And for senders:

$$T = \frac{\text{\# of senders}}{0.25 \cdot 0.05 \cdot \text{session bw}} \cdot \text{avg. RTCP packet size}$$

- Also, introduce some randomness in order to avoid synchronization of reports:
  - Set T to T $\times$ random(0.5,…,1.5).

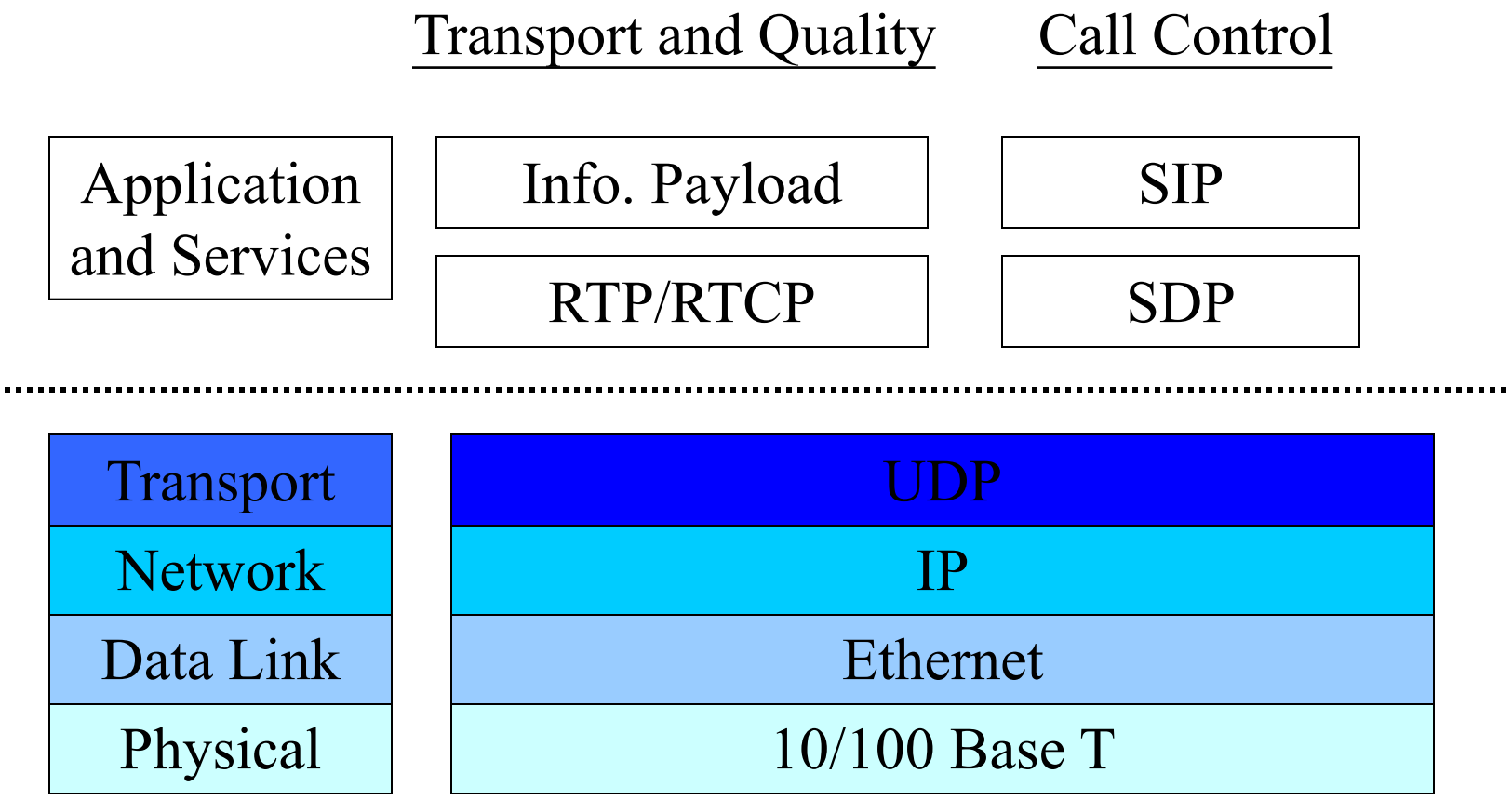
Outline



I- Internet audio protocols

- Multimedia protocol stack
- RTP/RTCP
- Session Initiation Protocol (SIP)

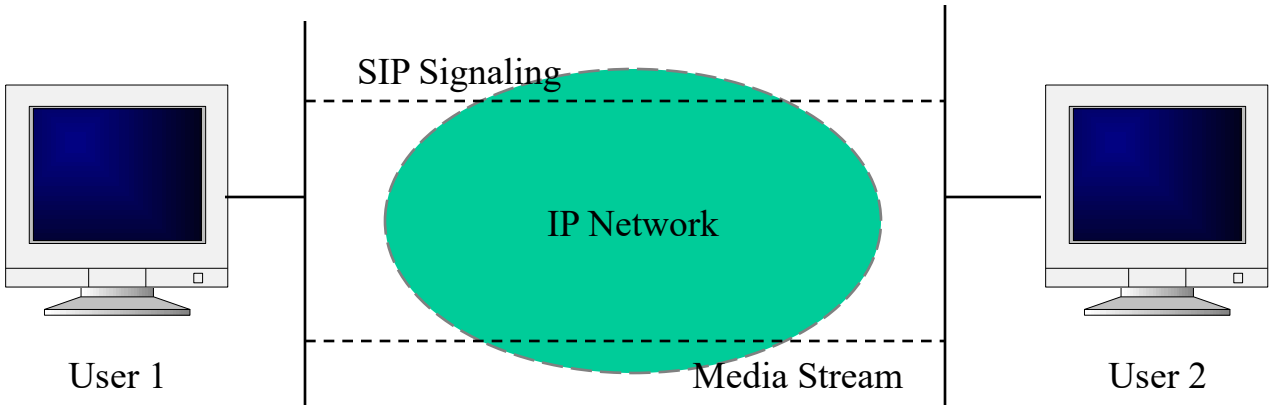
SIP: Session Initiation Protocol



What does SIP do?

- Handles the establishment of user sessions:
  - Does not define the actual session information.
  - Locate users in a very dynamic environment.
- Delivers a description of the session:
  - Uses common MIME (Multipurpose Internet Mail) extensions.
  - Can negotiate a common format.
- Allows users to change/control/terminate sessions.
- Once a Call Session has been established, RTP/RTCP are used to exchange media.

Example

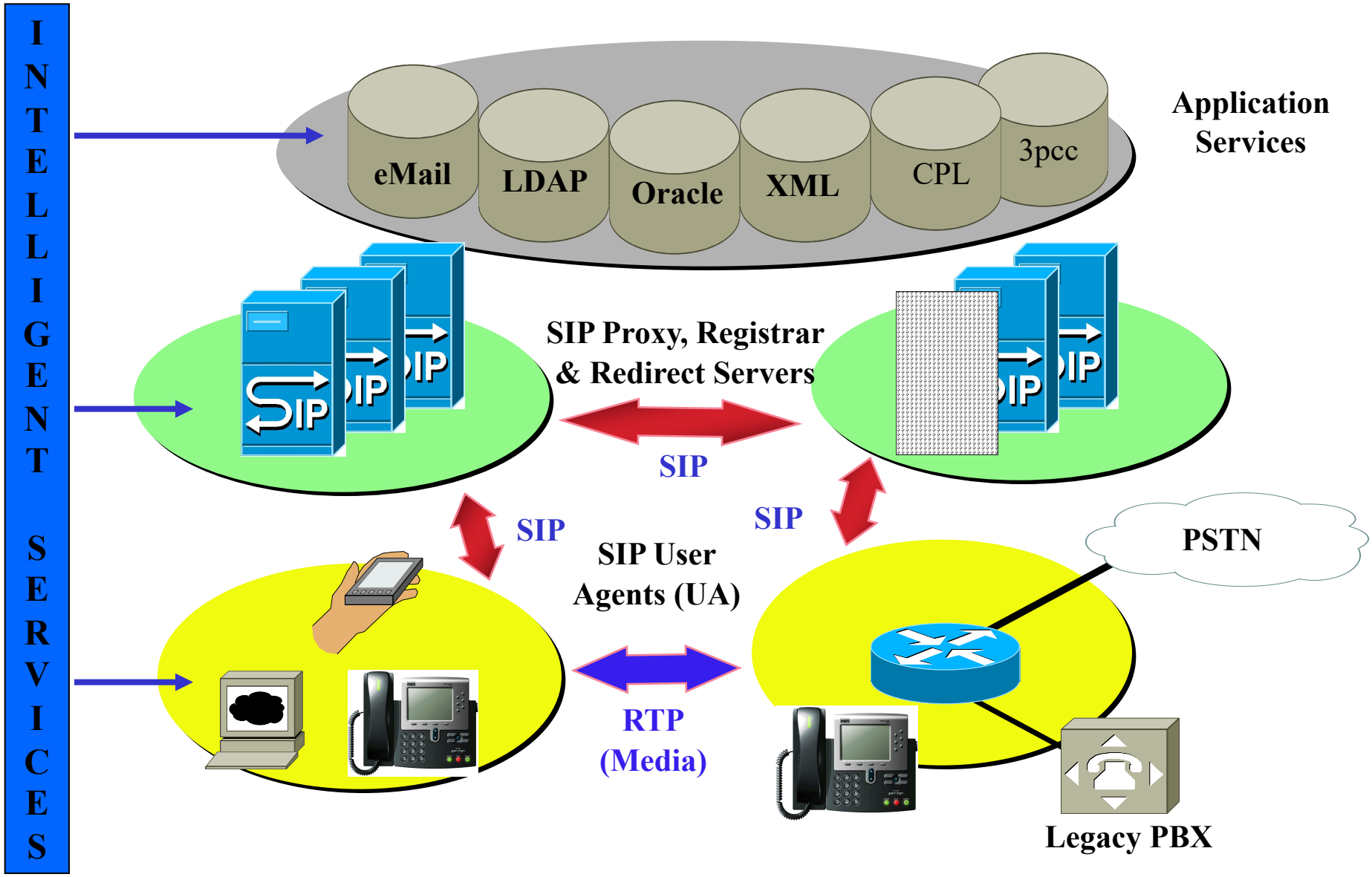


- User 1 requests a session, passing characteristics to other users.
- User 2 accepts invitation, modifying session as necessary.
- Session begins.
- Either user signals the other if changes are necessary, or when terminating the session.

SIP philosophy

- Internet Standard:
  - IETF - http://www.ietf.org
  - MMUSIC - Multiparty Multimedia Session Control Working Group
- Reuse Internet addressing (URLs, DNS, proxies).
- Reuse HTTP coding (Text based).
- Makes no assumptions about underlying protocol:
  - TCP, UDP, X.25, frame, ATM, etc.
  - Support of multicast.
- Used for more than audio, e.g. Internet messaging.

SIP basics - architecture



SIP basics - architectural elements

- Clients: SIP Phones, Softphones, PDAs, Robots.
  - User Agent Client (UAC) / User Agent Server (UAS).
  - Originate / Terminate SIP requests.
  - Typically an endpoint will have both UAC & UAS, UAC for originating requests, and UAS for terminating requests.
- Servers:
  - Registrar: Accepts REGISTER requests from clients.
  - Proxy: Decides next hop and forwards request.
  - Redirect: Sends address of next hop back to client.
- Gateways:
  - To PSTN for telephony interworking.
  - To H.323 for IP Telephony interworking.

SIP addressing

- Uses Internet URLs:
  - Uniform Resource Locators.
  - Supports both Internet and PSTN addresses.
  - General form is name@domain.
  - To complete a call, needs to be resolved down to User@Host.
  - Examples:

    sip:alan@wcom.com

    sip:J.T. Kirk <kirk@starfleet.gov>

    sip:+1-613-555-1212@wcom.com;user=phone

    sip:guest@10.64.1.1

SIP requests

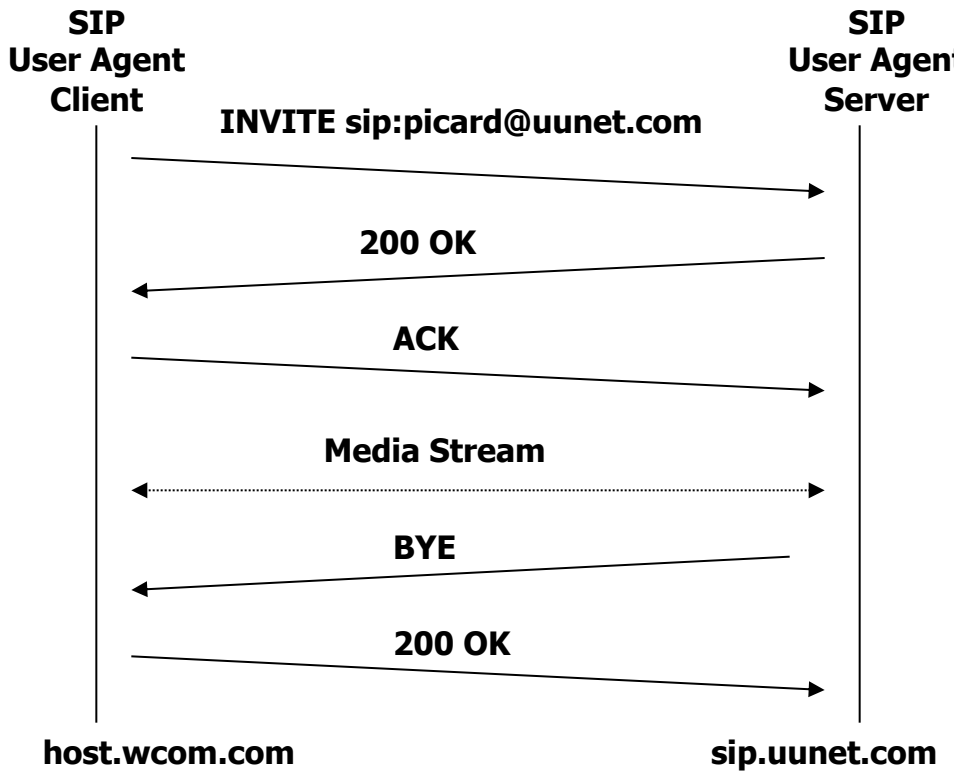
Method	Description
INVITE	**A session is being requested to be setup using a specified media**
ACK	**Message from client to indicate that a successful response to an INVITE has been received**
OPTIONS	**A Query to a server about its capabilities**
BYE	**A call is being released by either party**
CANCEL	**Cancels any pending requests. Usually sent to a Proxy Server to cancel searches**
REGISTER	**Used by client to register a particular address with the SIP server**

SIP responses

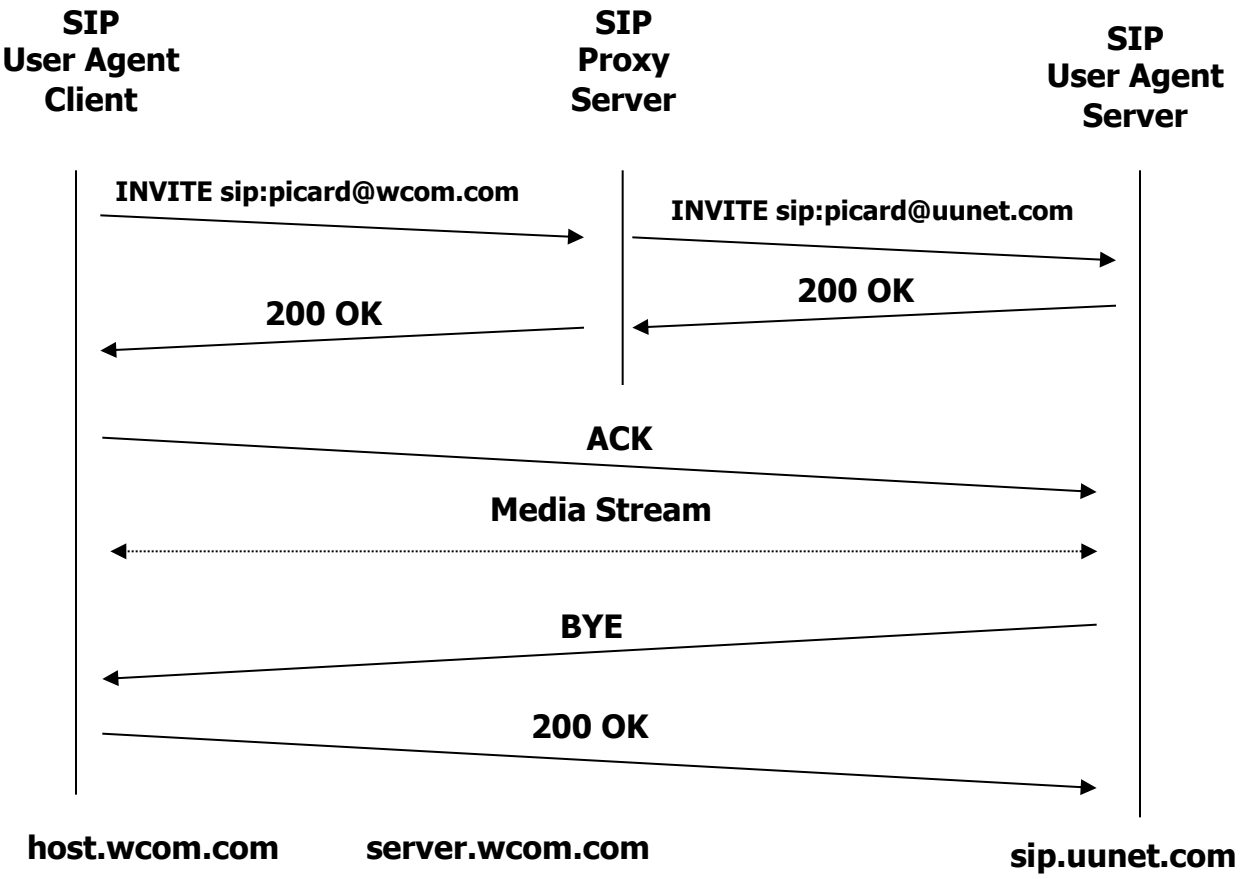
- First digit gives class of response.

	Description	Examples
1xx	Informational – Request received, continuing to process request.	180 Ringing 181 Call is Being Forwarded
2xx	Success – Action was successfully received, understood and accepted.	200 OK
3xx	Redirection – Further action needs to be taken in order to complete the request.	300 Multiple Choices 302 Moved Temporarily
4xx	Client Error – Request contains bad syntax or cannot be fulfilled at this server.	401 Unauthorized 408 Request Timeout
5xx	Server Error – Server failed to fulfill an apparently valid request.	503 Service Unavailable 505 Version Not Suported
6xx	Global Failure – Request is invalid at any server.	600 Busy Everywhere 603 Decline

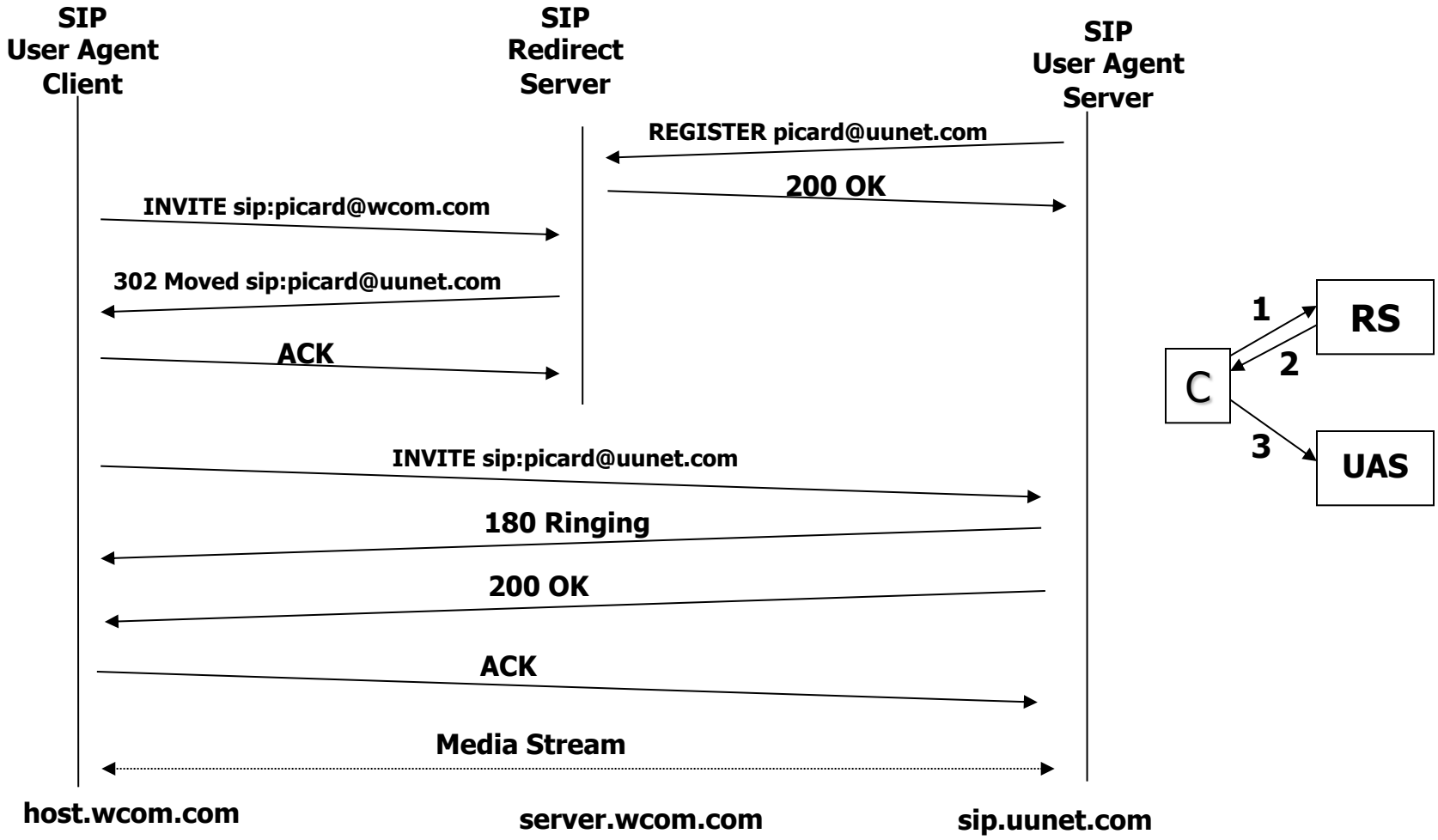
SIP session setup example



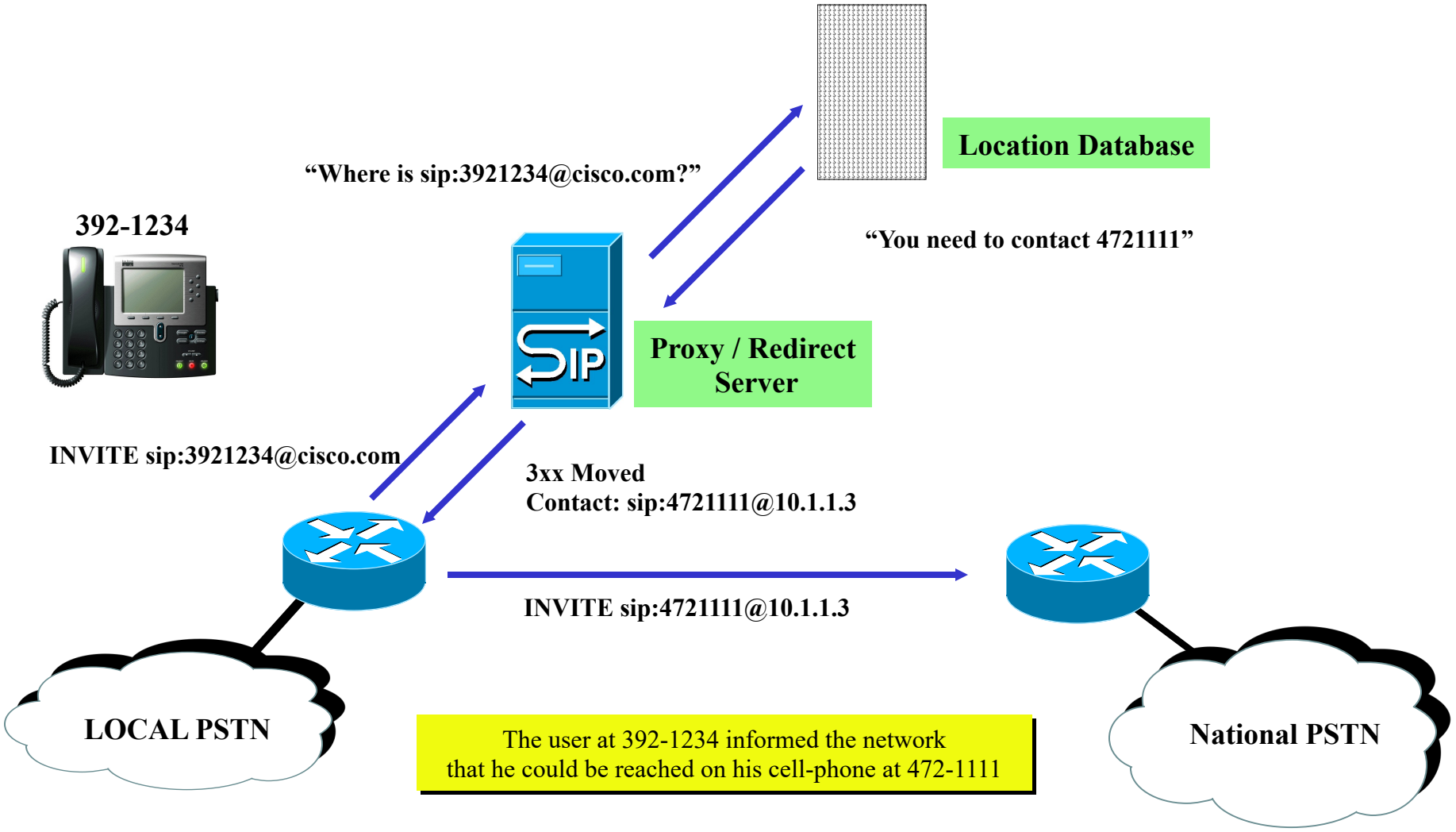
Proxy server example



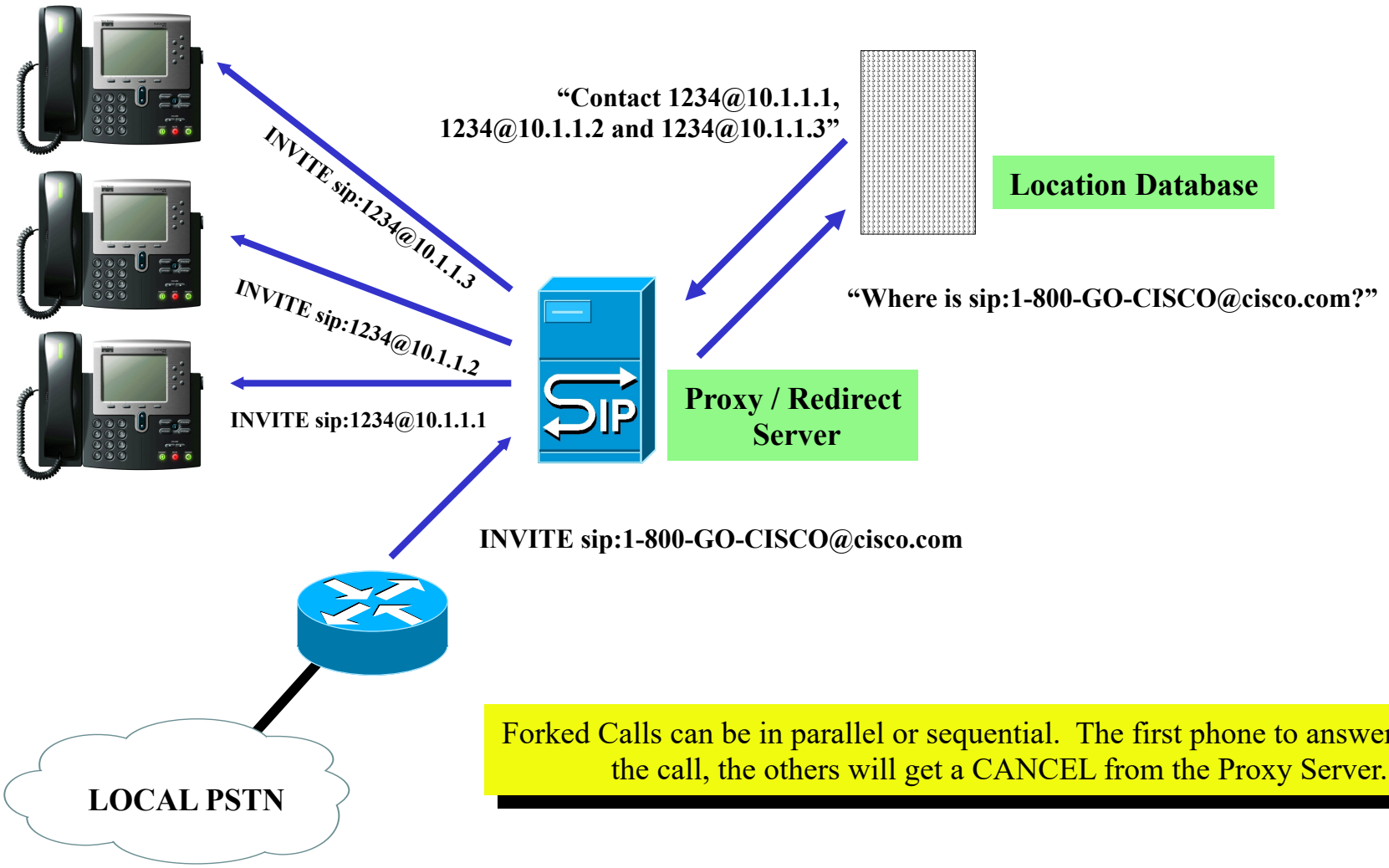
Redirect server example



Call redirection



Call forking



Forked Calls can be in parallel or sequential. The first phone to answer will get the call, the others will get a CANCEL from the Proxy Server.

SIP message body (Invite)

- Message body can be any protocol (as HTTP).
- Mostly used:
  - SDP - Session Description Protocol.
  - RFC 2327 4/98 by Handley and Jacobson.
    - http://www.ietf.org/rfc/rfc2327.txt
  - Used to specify info about a multi-media session.
  - SDP fields have a required order.
  - For RTP - Real Time Protocol Sessions: RTP Audio/Video Profile (RTP/AVP) payload descriptions are often used.

SDP examples

SDP Example 1

v=0
o=ajohnston +1-613-555-1212 IN IP4
host.wcom.com
s=Let's Talk
t=0 0
c=IN IP4 101.64.4.1
m=audio 49170 RTP/AVP 0 3

SDP Example 2

v=0
o=picard 124333 67895 IN IP4
uunet.com
s=Engage!
t=0 0
c=IN IP4 101.234.2.1
m=audio 3456 RTP/AVP 0

Field	Descripton
Version	v=0
Origin	o=<username> <session id> <version> <network type> <address type> <address>
Session Name	s=<session name>
Times	t=<start time> <stop time>
Connection Data	c=<network type> <address type> <connection address>
Media	m=<media> <port> <transport> <media format list>

References

- Jean-Chrysostome Bolot, "Characterizing end-to-end packet delay and loss in the Internet", Journal of High Speed Networks, 2(3):305-323, 1993.

- J-C. Bolot, "Adaptive Applications Tutorial", ADAPTS BOF, IETF Meeting, Washington DC, Dec. 1997.

- Brian Gracely, "Voice over IP (VoIP)", Cisco Systems Next Generation Internet Symposium, 2001.

- R. Ramjee, Jim Kurose, Don Towslet, and Henning Schulzerine, "Adaptive playout mechanisms for packetized audio applications in wide area networks," Proceedings of IEEE INFOCOMM, pp. 680--686, June 1994.

- Benjamin NICOLLE, Philippe GORGUEIRA, Olivier DERRIEN, "Introduction to Voice over IP", Journées Portes Ouvertes 2001, Université de Versailles St. Quentin-en-Yvelines, 2001.

- Prof. Dan Rubenstein, "Real time Internet lecture", columnbia university.

- Prof. Henning Schulzrinne, "Tutorial on Voice over IP ", ACM SIGCOMM, august 2000.

- Henry Sinnreich, Alan Johnston, "Internet Telephony based on SIP", SMU - Dallas, April 28, May 1, 2000.

-D. De Vleeschauwer, J. Janssen, G. H. Petit and F. Poppe, "Quality bounds for packetized voice transport", Alcatel Telecommunications Review (1st quarter 2000) 19-24.

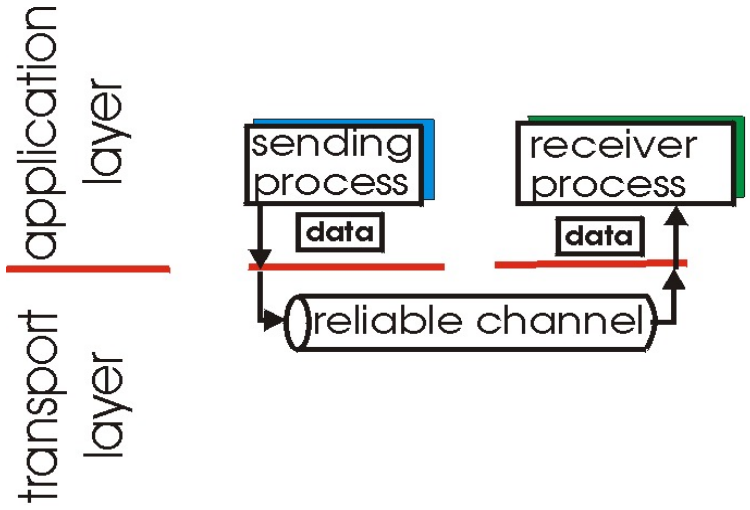
-V Ramos, C Barakat, E Altman, " A Moving Average Predictor for Playout Delay Control in VoIP", in proceedings of IWQoS 2003, Monterey, CA, June 2003.

Additional slides

- For further information and details
- Not included in the course

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

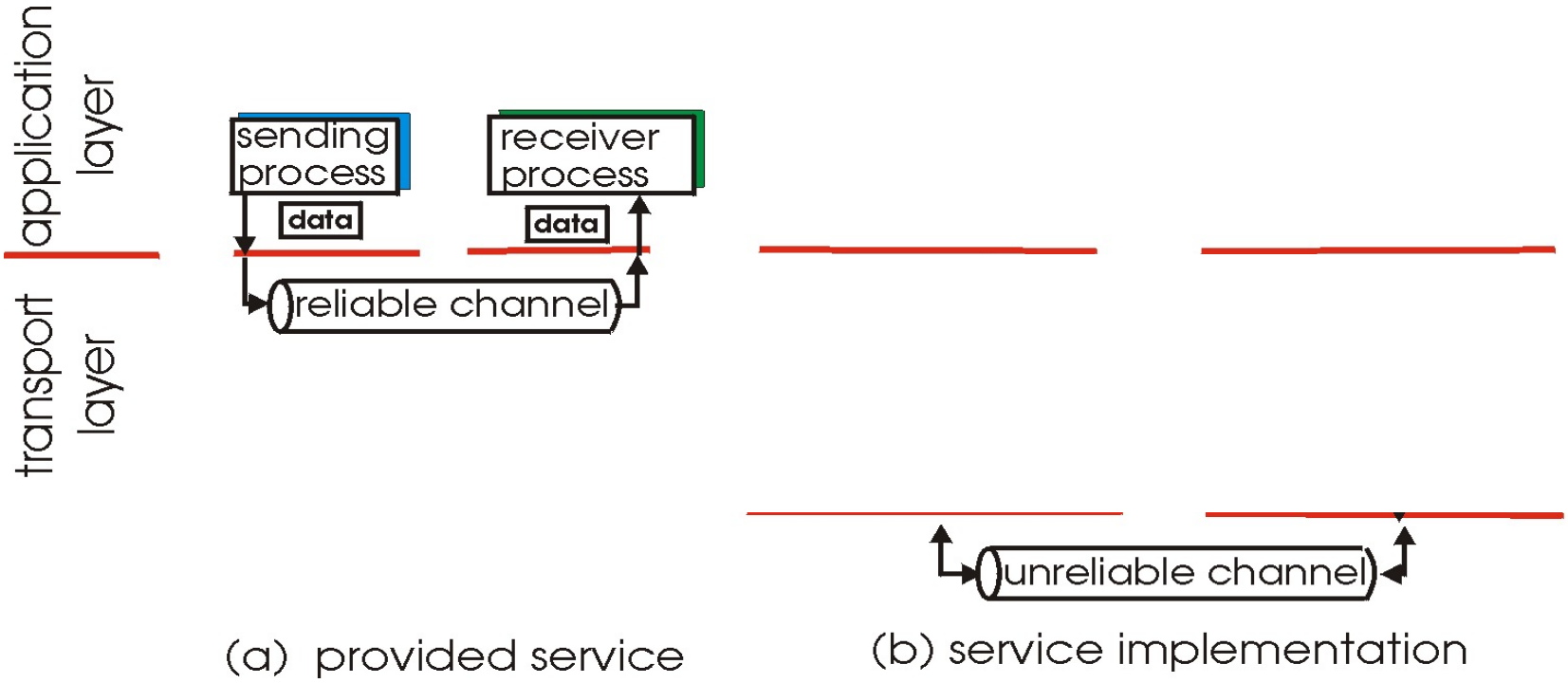


(a) provided service

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

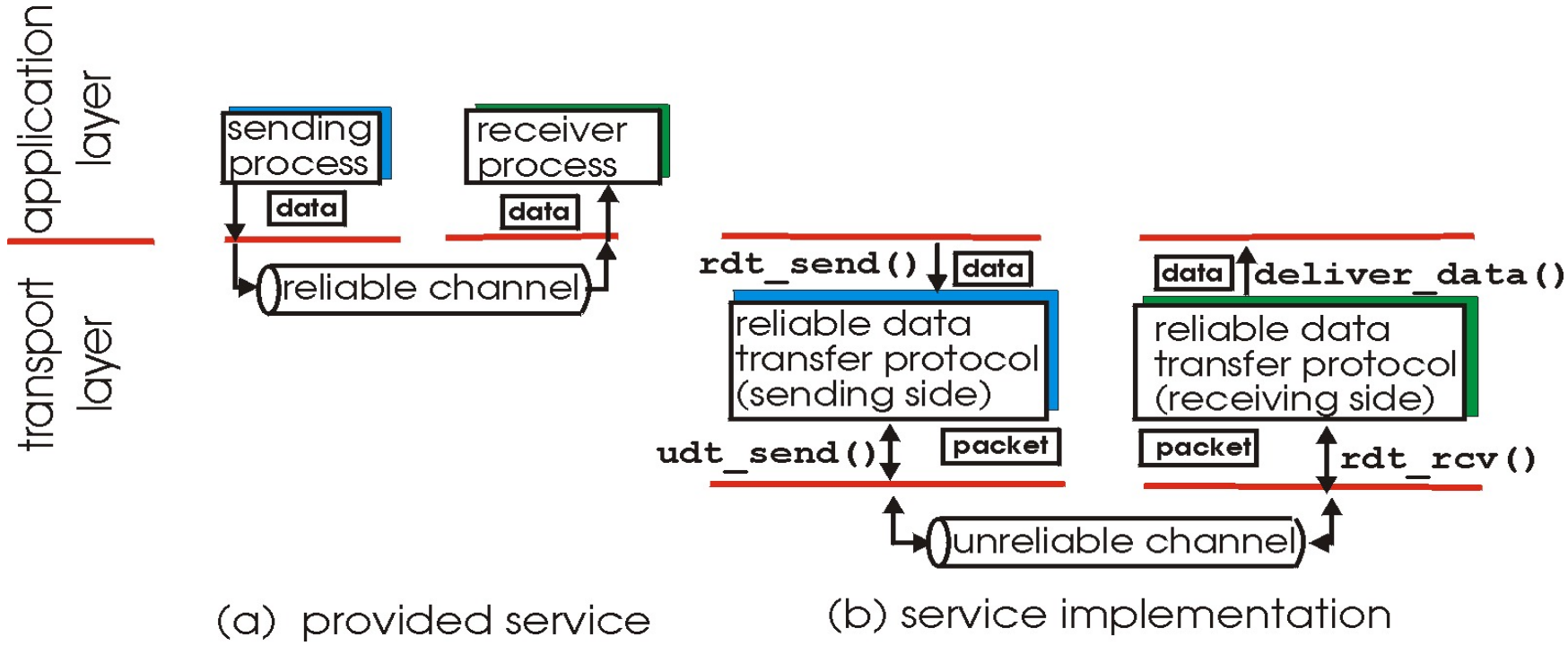
- important in app., transport, link layers
- top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



(a) provided service

(b) service implementation

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- Separate state machines for sender and receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel

Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- new mechanisms in `rdt2.0` (beyond `rdt1.0`):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender

rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

Handling duplicates:

- sender retransmits current pkt if ACK/NAK garbled
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

Sender sends one packet, then waits for receiver Response. This gives rdt2.1.

rdt2.1: discussion

Sender:

- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "current" pkt has 0 or 1 seq. #

Receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*
- Main advantage compared to previous one: Can recover from packets replicated inside the network.

rdt3.0: channels with errors *and* loss

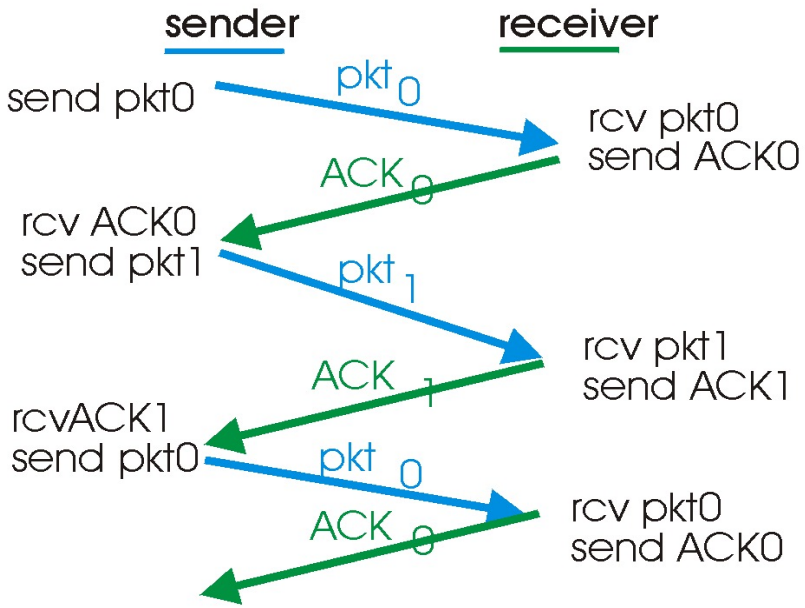
New assumption: underlying channel can also lose packets (data or ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

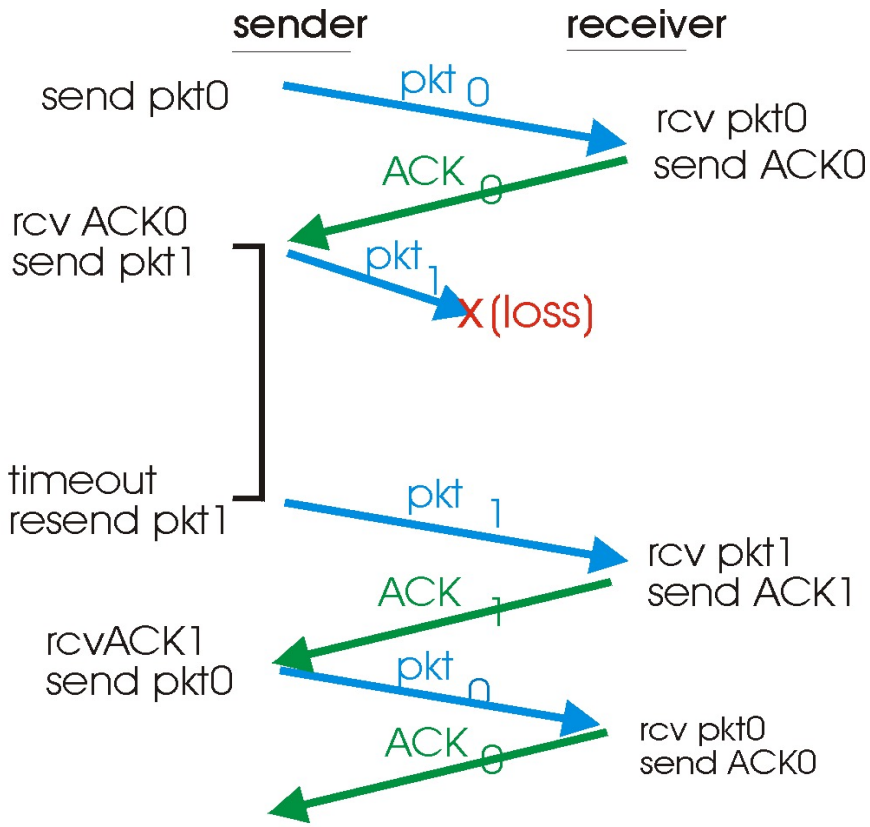
Approach: sender waits "reasonable" amount of time for ACK

- ❑ retransmits if no ACK received in this time
- ❑ if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- ❑ requires countdown timer

rdt3.0 in action



(a) operation with no loss



(b) lost packet

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10**9\ b/sec} = 8 \text{ microsec}$$

  - U_{sender}: utilization – fraction of time sender busy sending

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

  - 1KB pkt every 30 msec -> 33kB/sec throughput over 1 Gbps link
  - network protocol limits use of physical resources!

Generalities: Pipelining Protocols

Go-back-N: overview

- *sender:* up to N unACKed pkts in pipeline
- *receiver:* only sends cumulative ACKs
  - doesn't ACK pkt if there's a gap
- *sender:* has timer for oldest unACKed pkt
  - if timer expires: retransmit all unACKed packets

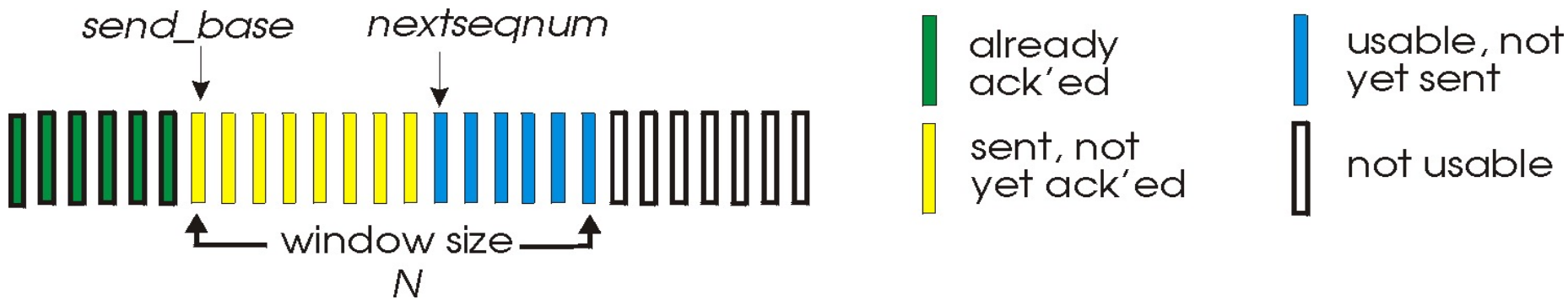
Selective Repeat: overview

- *sender:* up to N unACKed packets in pipeline
- *receiver:* ACKs individual pkts
- *sender:* maintains timer for each unACKed pkt
  - if timer expires: retransmit only unACKed packet

Generalities: Go-Back-N

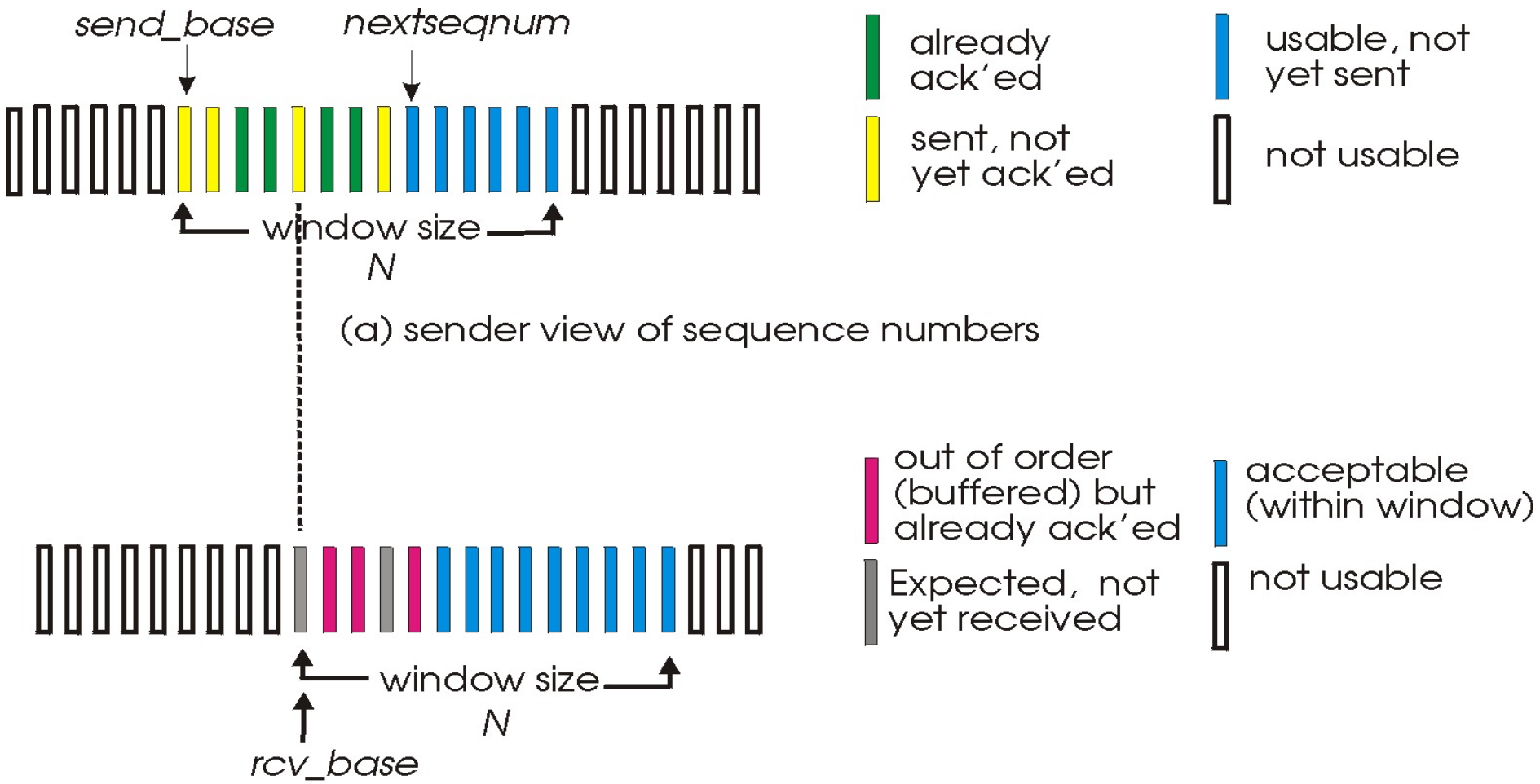
Sender:

- k-bit seq # in pkt header
- "window" of up to N, consecutive unACKed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

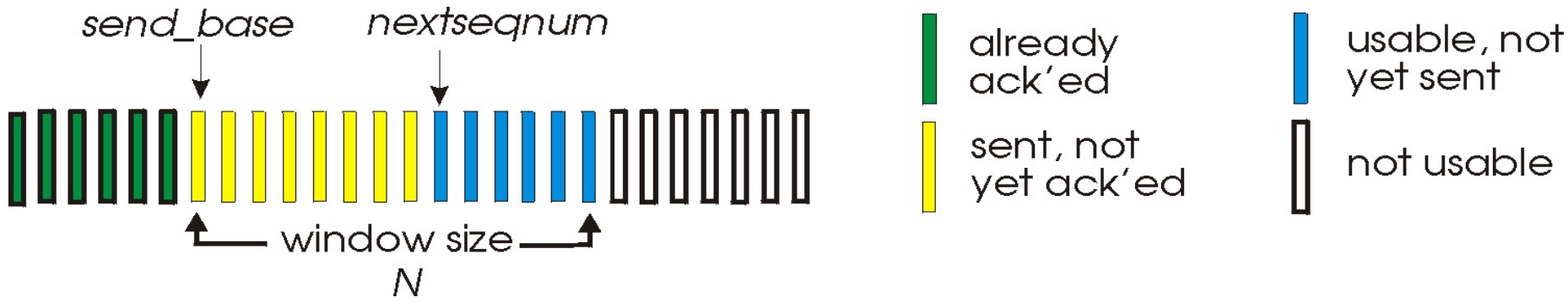
Generalities: Selective ACKs, sender, receiver



Go-Back-N

Sender:

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

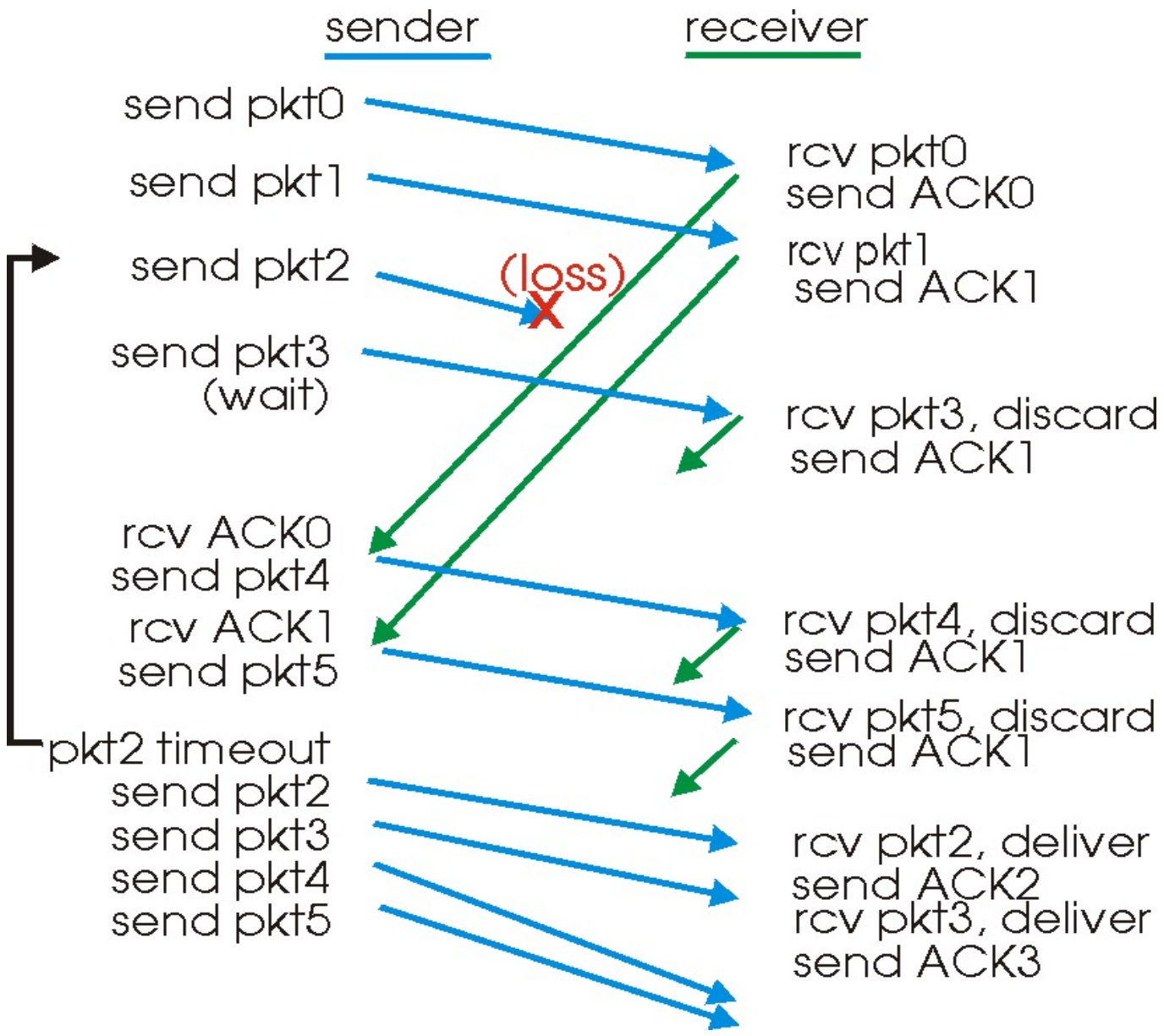
GBN: the receiver side

ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

❑ out-of-order pkt:

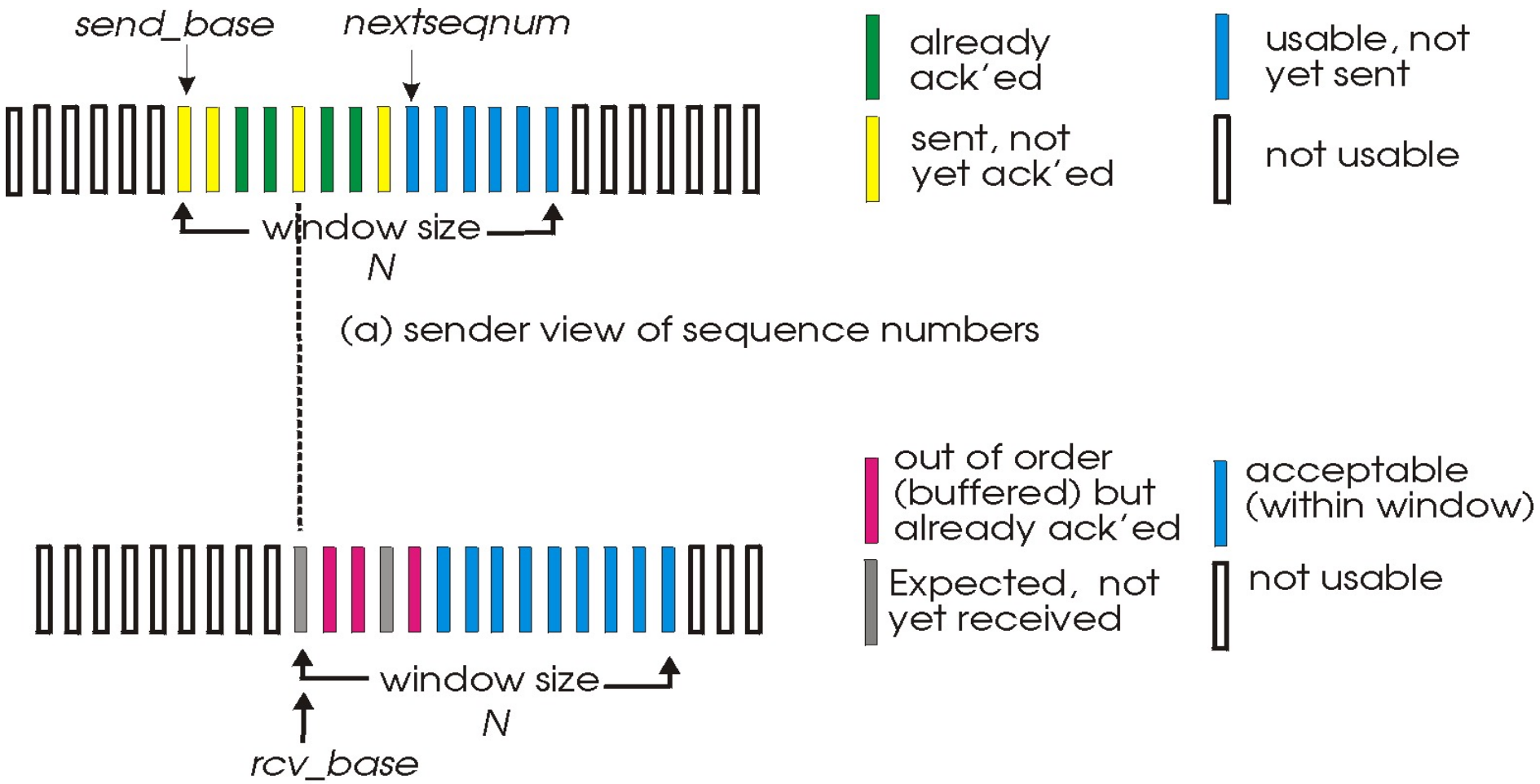
- You can discard (don't need to buffer)
- Re-ACK pkt with highest in-order seq #


sender
receiver
send pkt0
send pkt1
send pkt2
(loss)
send pkt3
(wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3

Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts

Selective repeat: sender, receiver windows



Selective repeat

sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat: dilemma

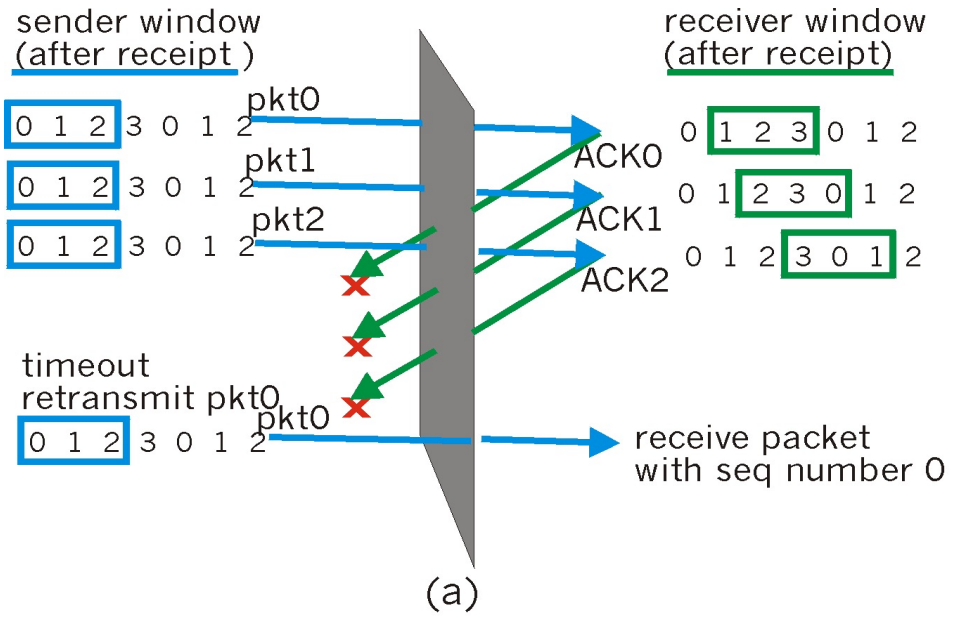
Example:

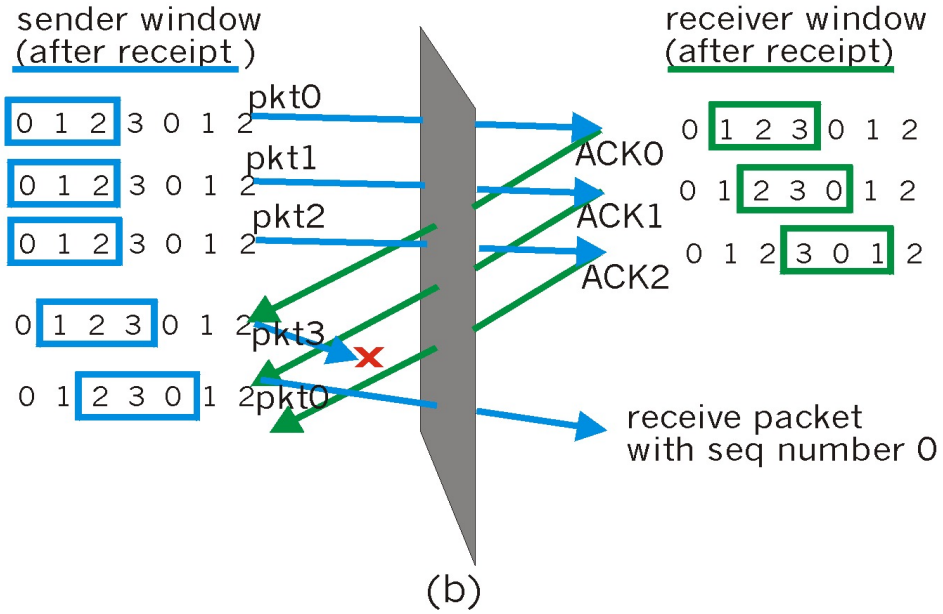
- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?

Q: And window size and throughput? And delai?





Selective repeat

sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat: dilemma

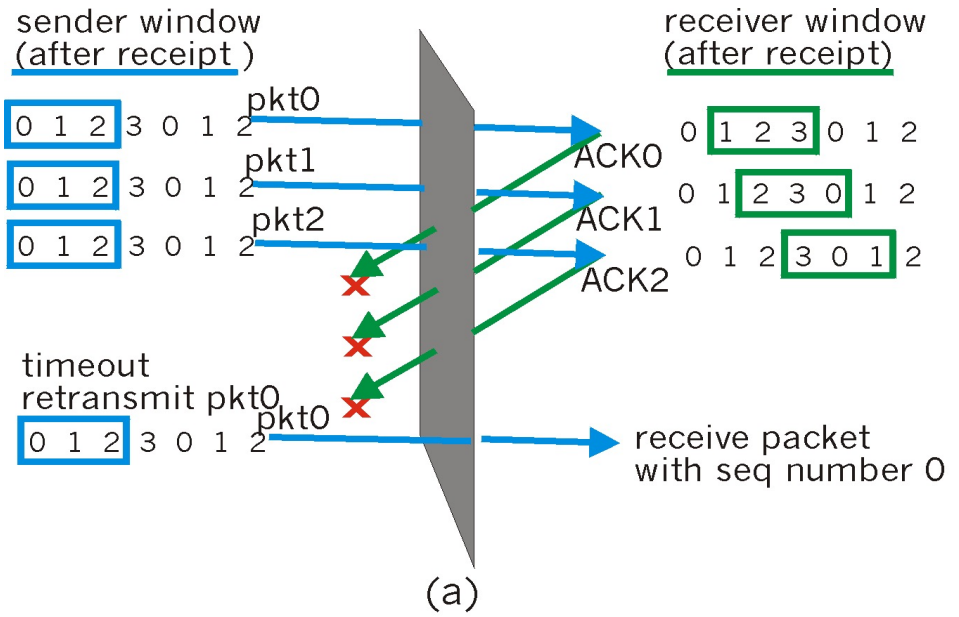
Example:

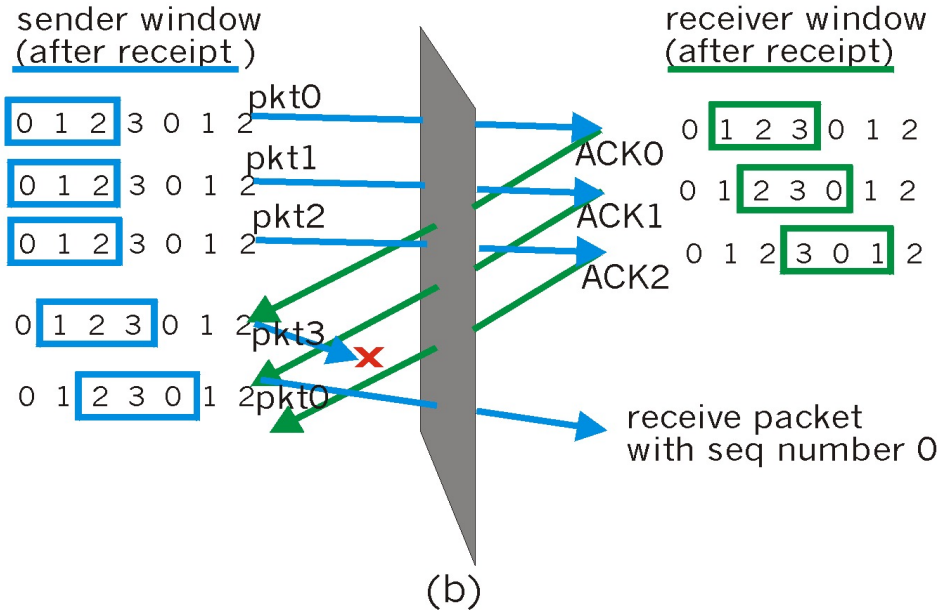
- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?

Q: And window size and throughput? And delai?





Three Way Handshake Example

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
3. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED
5. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED

- In line 2 TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100
- In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100
- At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN
- At line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!). Also note that the segment still have the ACK flag set, this is due to cumulative ACK and piggybacked ACKs

Three Way Handshake Example

- For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers
- This is done in an exchange of connection establishing segments carrying the flag bit SYN (for synchronize) and the initial sequence
- The synchronization requires each side to send its own initial sequence number and to receive a confirmation of it in acknowledgment from the other side.
- Each side must also receive the other side's initial sequence number and send a confirming acknowledgment
  - 1) A --> B SYN my sequence number is X
  - 2) A <-- B ACK your sequence number is X
  - 3) A <-- B SYN my sequence number is Y
  - 4) A --> B ACK your sequence number is Y

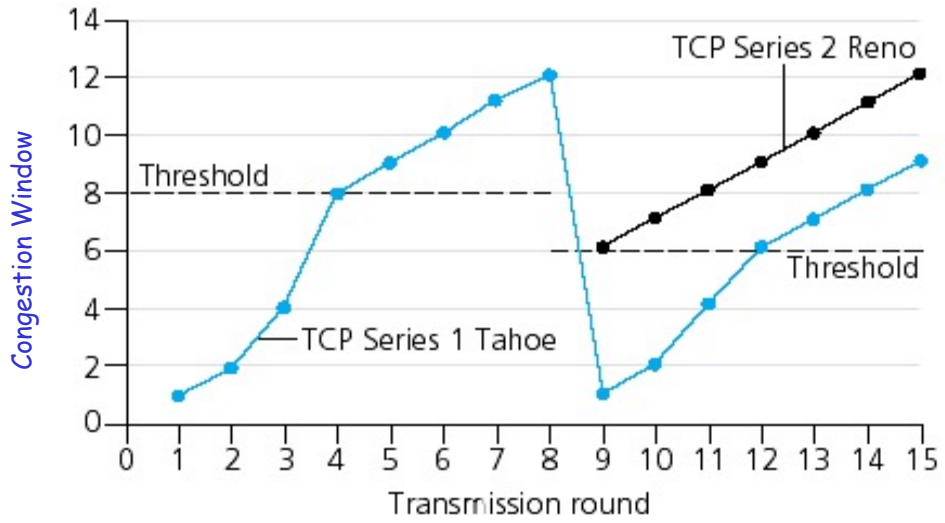
Three Way Handshake Example

- Because steps 2 and 3 can be combined in a single message this is called the three way handshake
- A three way handshake is necessary because the receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN

Refinement

Q: When should the exponential increase (*SS*) switch to linear (*CA*)?

A: When **CongWin** gets to 1/2 of its value before timeout, i.e., when it reaches **Threshold**



- TCP Tahoe (first version of TCP with congestion control, 1988) always cut its **CongWin** to 1 MSS
- Fast recovery was introduced in TCP Reno
  - Also introduces fast retransmit

Which version of TCP

- Congestion control mechanism introduced by Van Jacobson (1988)
- Versions of TCP
  - TCP Tahoe (first version with CC)
  - TCP Reno (fast retransmit, fast recovery)
  - TCP newReno (minor improvement)
  - TCP SACK (Reno + selective acknowledgements)
  - TCP Vegas (detect congestion before loss event by monitoring RTT increase)
  - Much more…

Which version of TCP

- Current implementations
  - windows Vista/XP/2003: TCP SACK with plenty of options
  - Current versions of linux/unix: TCP Reno/NewReno/SACK

Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and implementation in the Internet
  - UDP
  - TCP

Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

Notation, assumptions:

- Assume one link between client and server of rate R
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

Window size:

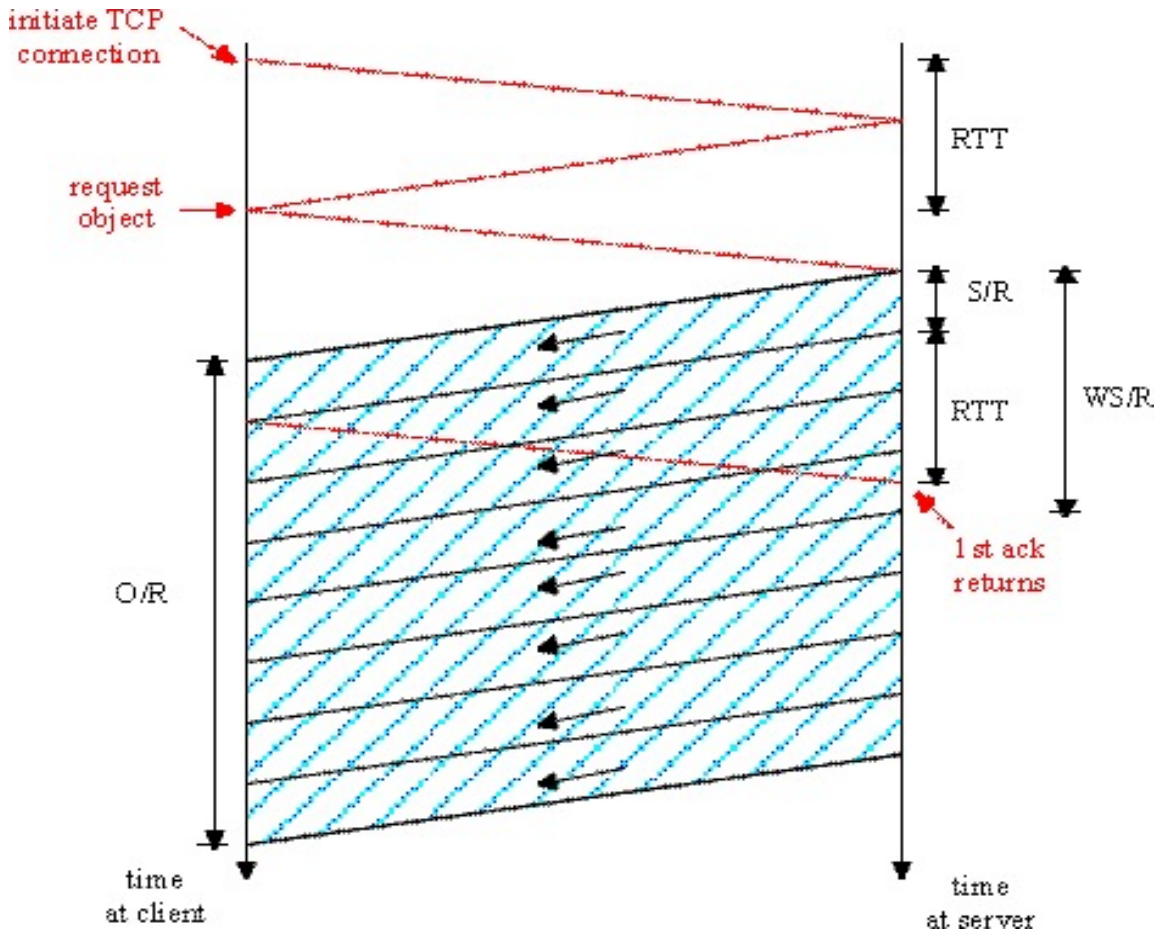
- First assume: fixed congestion window, W segments
- Then dynamic window, modeling slow start

Fixed congestion window (1)

First case:

WS/R > RTT + S/R: ACK for first segment in window returns before window's worth of data sent

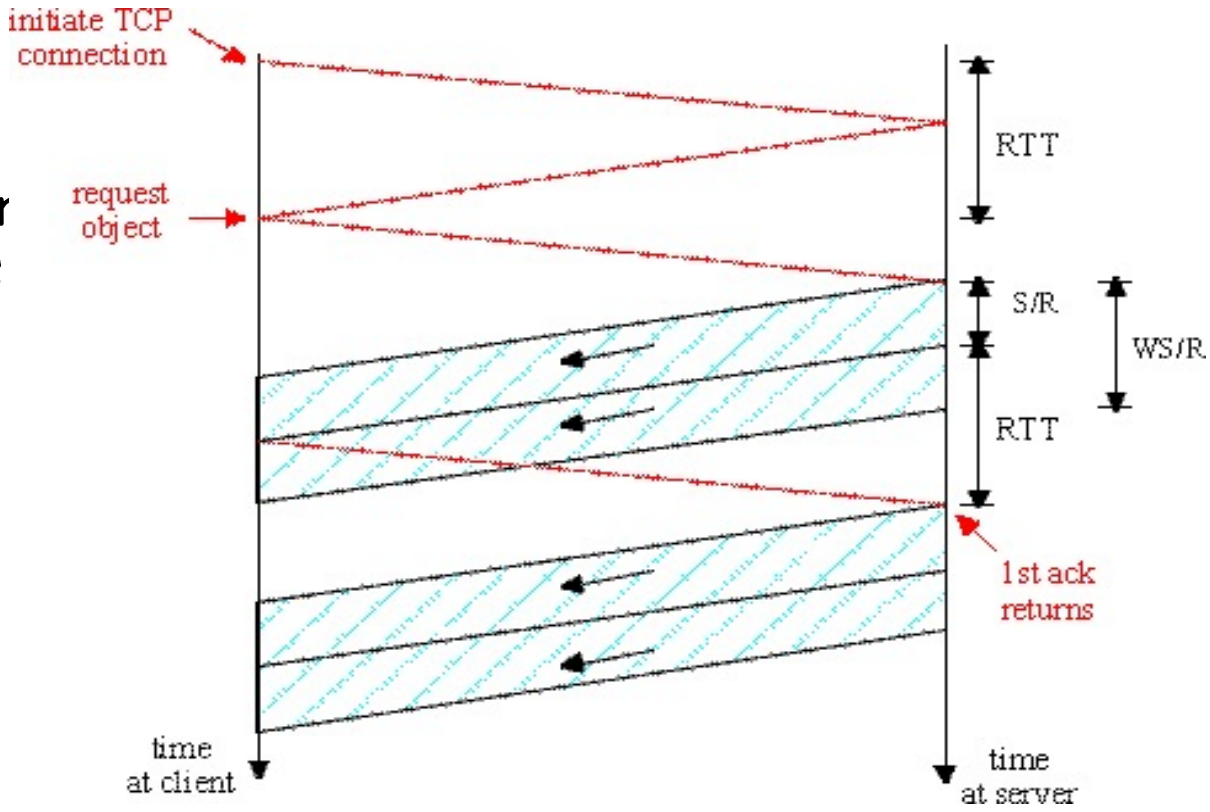
delay = 2RTT + O/R



Fixed congestion window (2)

Second case:

- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent



delay = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]

- Q: Find K

TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

$$Latency = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}$$

where P is the number of times TCP idles at server:

$$P = \min\{Q, K - 1\}$$

- where Q is the number of times the server idles if the object were of infinite size.

- and K is the number of windows that cover the object.

TCP Delay Modeling: Slow Start (2)

Delay components:

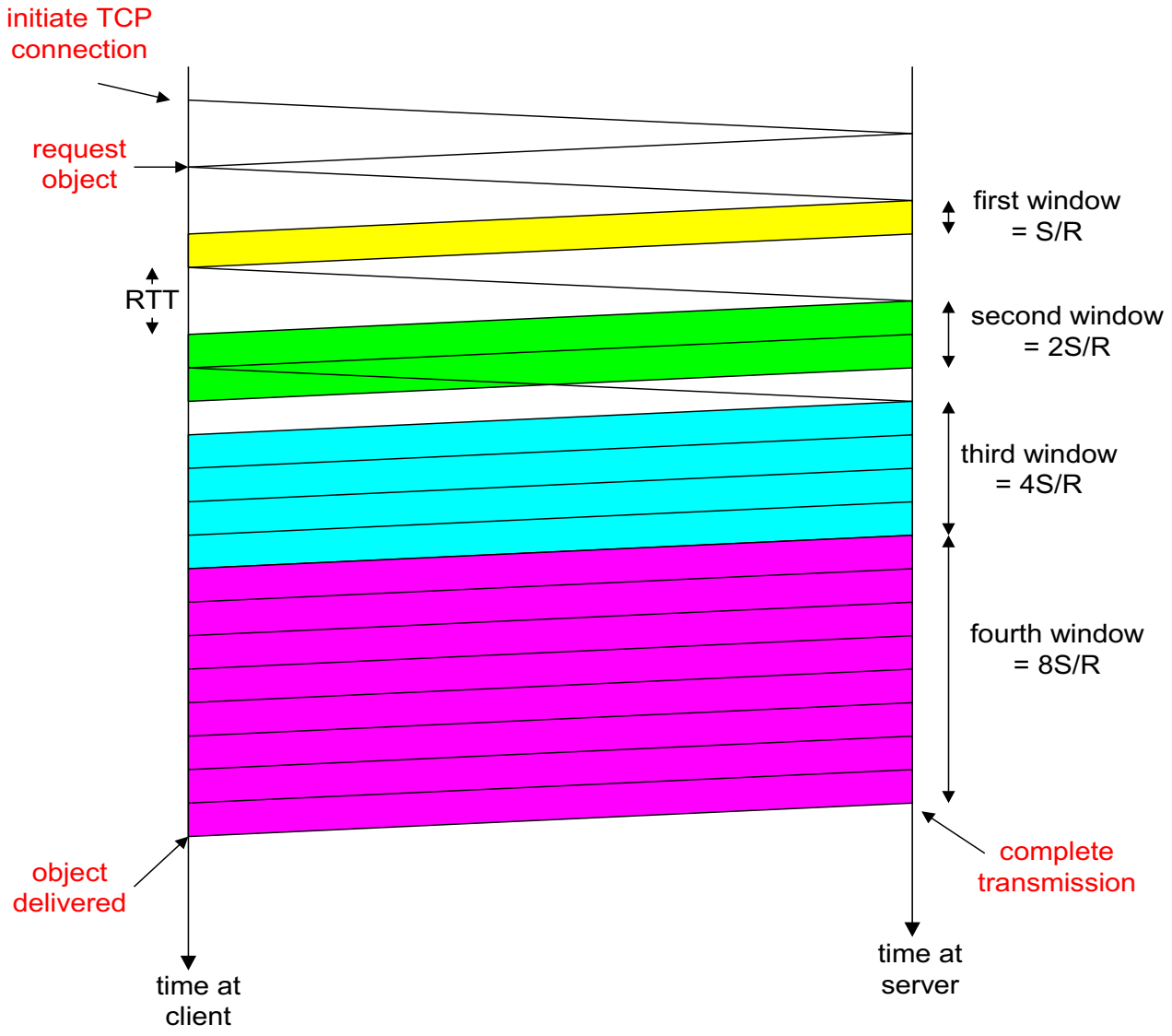
- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

Server idles:
P = min{K-1,Q} times

Example:

- O/S = 15 segments
- K = 4 windows
- Q = 2
- P = min{K-1,Q} = 2

Server idles P=2 times



TCP Delay Modeling (3)

$\frac{S}{R} + RTT =$ time from when server starts to send segment until server receives acknowledgement

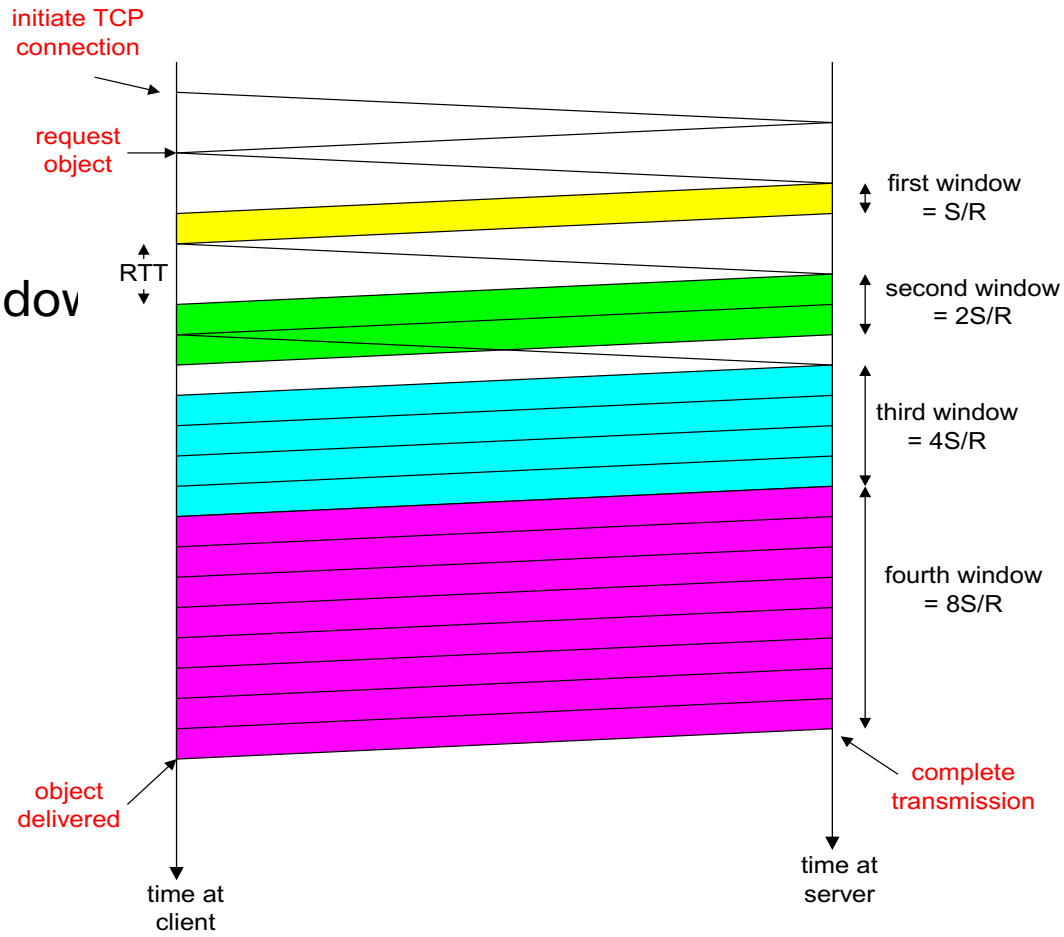
$2^{k-1}\frac{S}{R} =$ time to transmit the kth window

$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} =$ idle time after the kth window

$$\text{delay} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} idleTime_p$$

$$= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

$$= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P} - 1)\frac{S}{R}$$



TCP Delay Modeling (4)

Recall K = number of windows that cover object

How do we calculate K ?

$$
\begin{aligned}
K &= \min\{k: 2^0 S + 2^1 S + \mathrm{L} + 2^{k-1} S \geq O\} \\
&= \min\{k: 2^0 + 2^1 + \mathrm{L} + 2^{k-1} \geq O/S\} \\
&= \min\{k: 2^k - 1 \geq \frac{O}{S}\} \\
&= \min\{k: k \geq \log_2(\frac{O}{S} + 1)\} \\
&= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil
\end{aligned}
$$

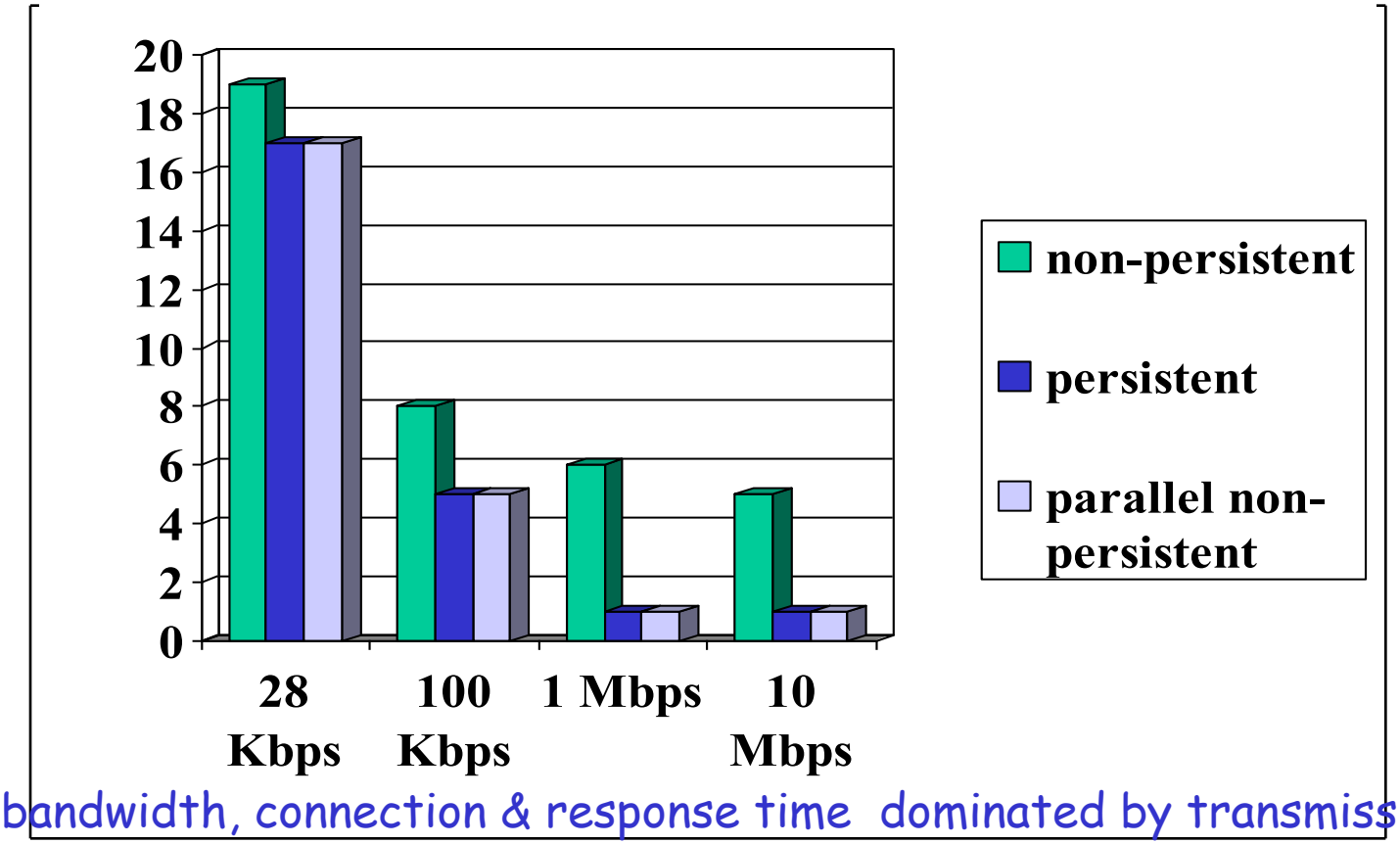
Calculation of Q, number of idles for infinite-size object, is similar.

HTTP Modeling

- Assume Web page consists of:
  - *1* base HTML page (of size *O* bits)
  - *M* images (each of size *O* bits)
- Non-persistent HTTP:
  - *M+1* TCP connections in series
  - *Response time = (M+1)O/R + (M+1)2RTT + sum of idle times*
- Persistent HTTP:
  - *2 RTT* to request and receive base HTML file
  - *1 RTT* to request and receive M images
  - *Response time = (M+1)O/R + 3RTT + sum of idle times*
- Non-persistent HTTP with X parallel connections
  - Suppose M/X integer.
  - 1 TCP connection for base file
  - M/X sets of parallel connections for images.
  - *Response time = (M+1)O/R + (M/X + 1)2RTT + sum of idle times*

HTTP Response time (in seconds)

RTT = 100 msec, O = 5 Kbytes, M=10 and X=5

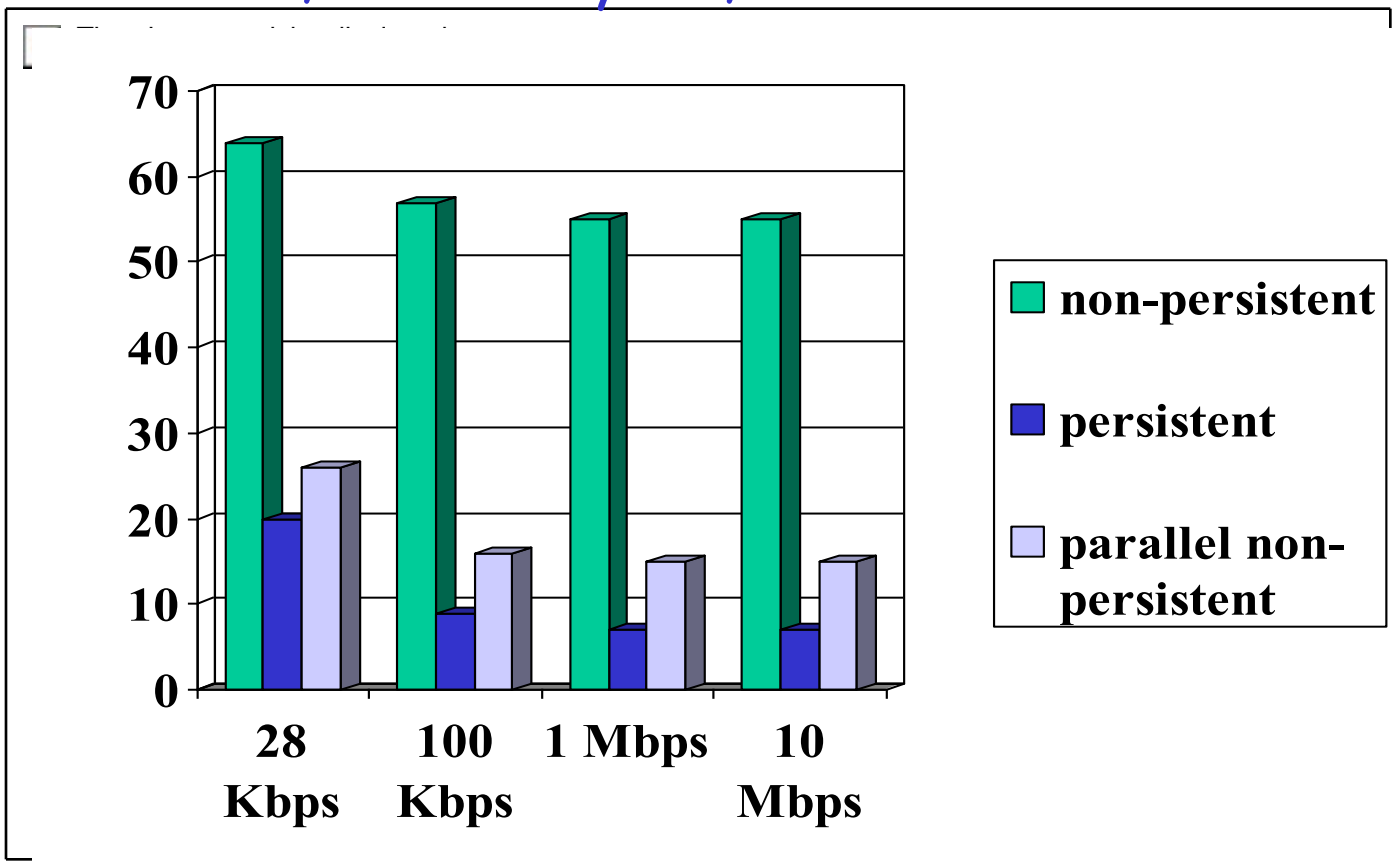


For low bandwidth, connection & response time dominated by transmission time.

Persistent connections only give minor improvement over parallel connections.

HTTP Response time (in seconds)

RTT =1 sec, O = 5 Kbytes, M=10 and X=5



For larger RTT, response time dominated by TCP establishment & slow start delays. Persistent connections now give important improvement: particularly in high delay•bandwidth networks.