Chair for Network Architectures and Services – Prof. Carle
Department of Computer Science
TU München

Master Course Computer Networks IN2097

Prof. Dr.-Ing. Georg Carle
Christian Grothoff, Ph.D.
Dr. Nils Kammenhuber

Chair for Network Architectures and Services
Institut für Informatik
Technische Universität München
http://www.net.in.tum.de

Chapter 3: Transport Layer

Our goals:

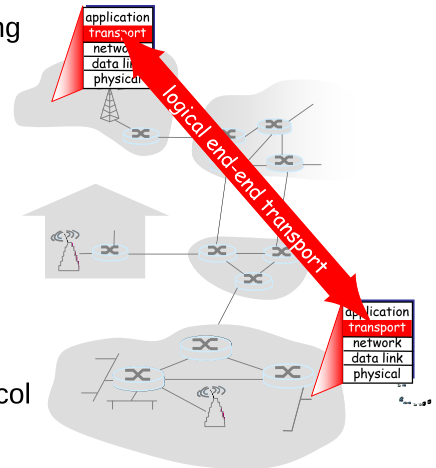
- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

Chapter 3 outline

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

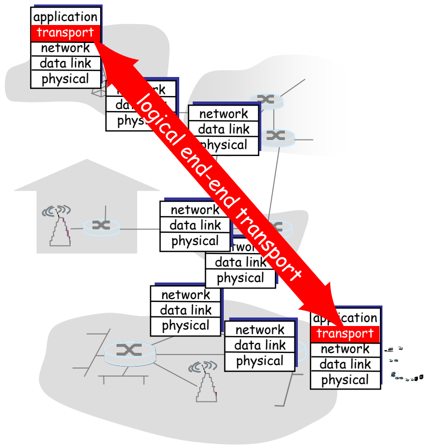
Transport services and protocols

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



Internet transport-layer protocols

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



Chapter 3 outline

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

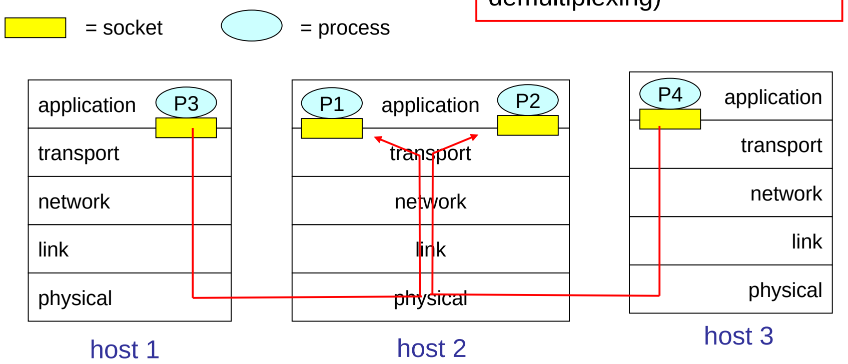
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

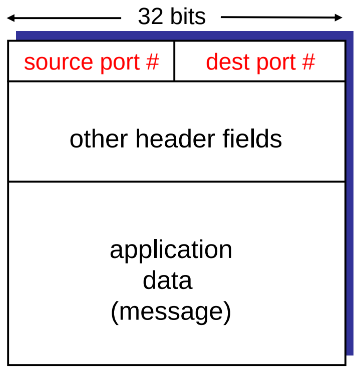
Multiplexing at send host:

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



How demultiplexing works

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



TCP/UDP segment format

Connectionless demultiplexing

- Create sockets with port numbers:

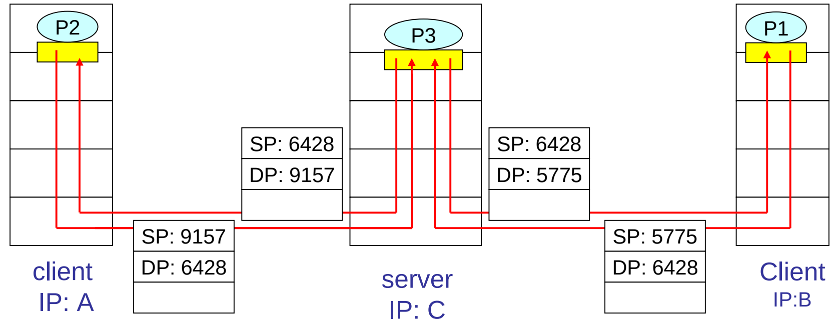
```
DatagramSocket mySocket1 = new DatagramSocket(12534);
DatagramSocket mySocket2 = new DatagramSocket(12535);
```

- UDP socket identified by two-tuple:

  (dest IP address, dest 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 different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

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

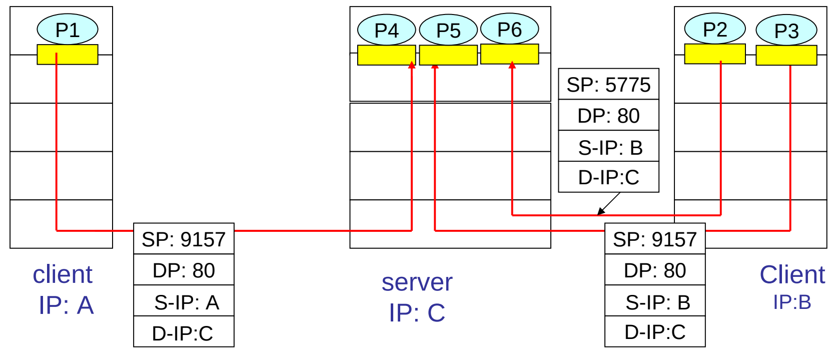


Source Port (SP) provides "return address"

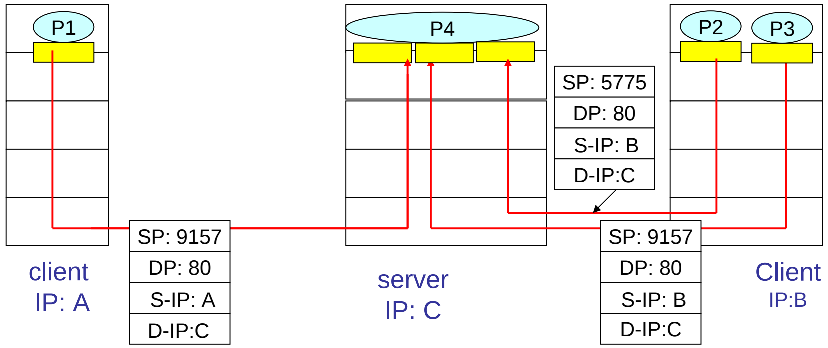
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv 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
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)



Connection-oriented demux: Threaded Web Server



Chapter 3 outline

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

UDP: User Datagram Protocol [RFC 768]

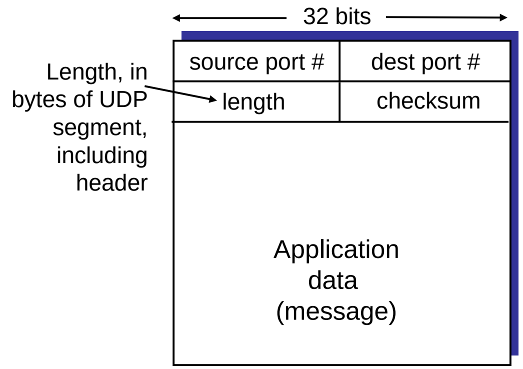
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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

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. *But maybe errors nonetheless?* More later ….

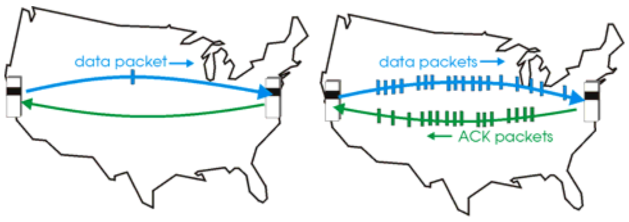
Chapter 3 outline

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

Pipelined protocols

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

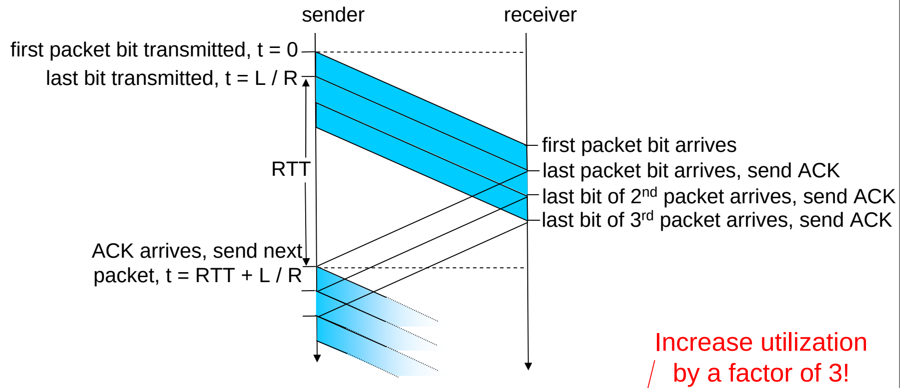
- range of sequence numbers must be increased
- buffering at sender and/or receiver



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

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelining: increased utilization



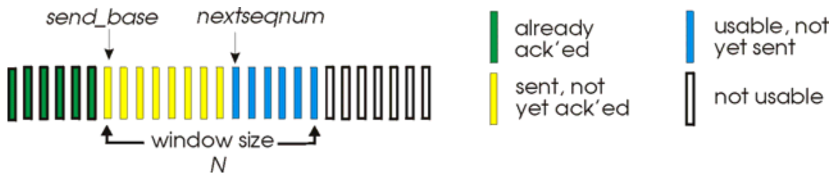
Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Go-Back-N

Sender:

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



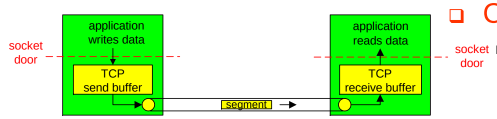
- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

Chapter 3 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
- Reliable Multicast

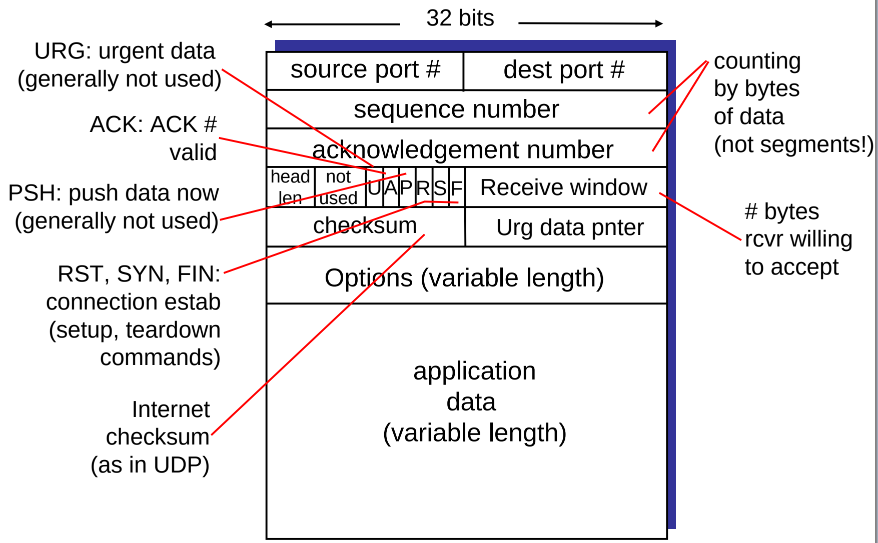
TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver
- Congestion controlled:
  - Will not overwhelm network

TCP segment structure

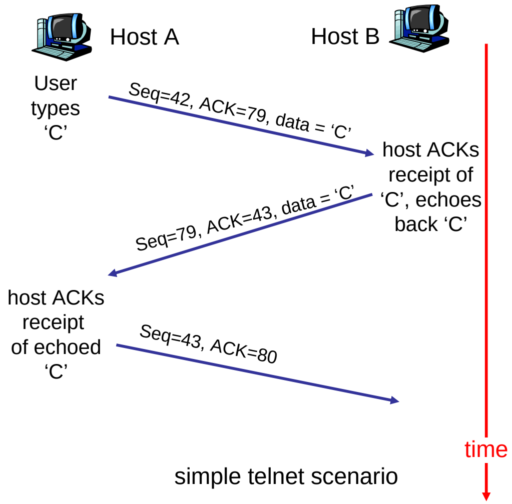


TCP seq. #'s and ACKs

Seq. #'s:
- byte stream "number" of first byte in segment's data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
- A: TCP spec doesn't say, - up to implementor



simple telnet scenario

Chapter 3 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
- Reliable Multicast

TCP sender events:

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

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks
- Initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

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?

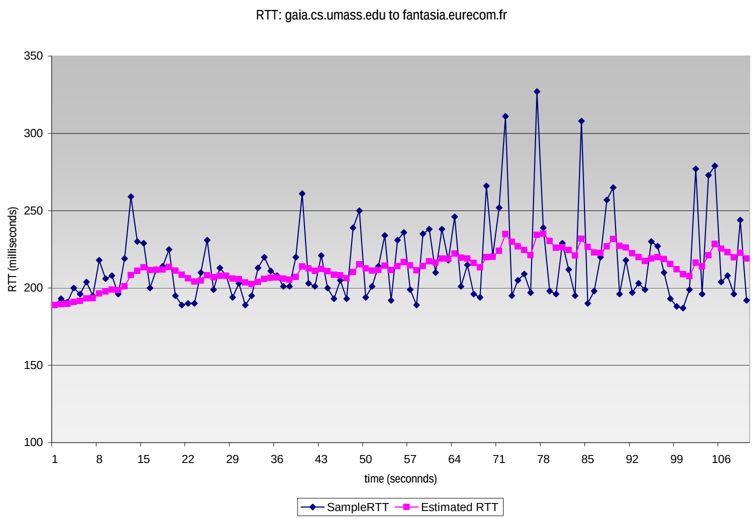
- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current **SampleRTT**

TCP Round Trip Time and Timeout

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

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125

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|

(typically, β = 0.25)
```

Then set timeout interval:

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

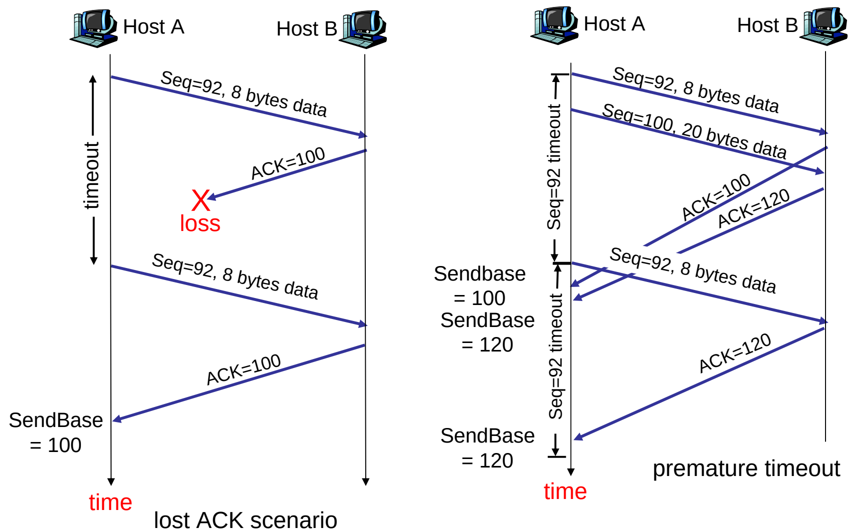
TCP sender (simplified)

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
 loop (forever) {
   switch(event)

   event: data received from application above
      create TCP segment with sequence number NextSeqNum
      if (timer currently not running)
           start timer
      pass segment to IP
      NextSeqNum = NextSeqNum + length(data)

   event: timer timeout
      retransmit not-yet-acknowledged segment with
           smallest sequence number
      start timer

   event: ACK received, with ACK field value of y
      if (y > SendBase) {
           SendBase = y
          if (there are currently not-yet-acknowledged segments)
               start timer  }
 }  /* end of loop forever */
```

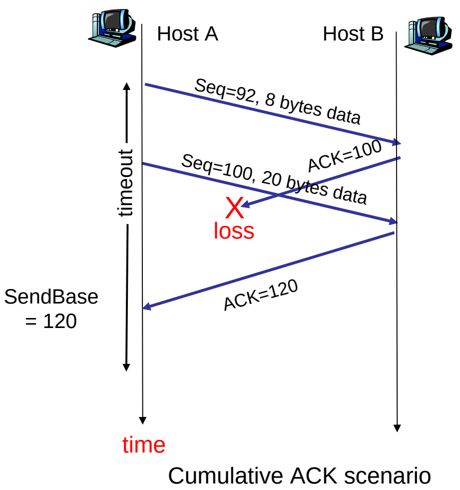
Comment:
- SendBase-1: last cumulatively ack'ed byte

Example:
- SendBase-1 = 71; y= 73, so the rcvr wants 73+ ; y > SendBase, so that new data is acked

TCP: retransmission scenarios



TCP retransmission scenarios (more)



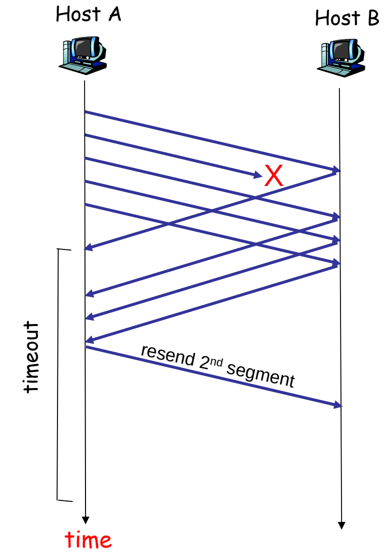
TCP ACK generation [RFC 1122, RFC 2581]

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 500ms 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 starts at lower end of gap

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

Resending a segment after triple duplicate ACK



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

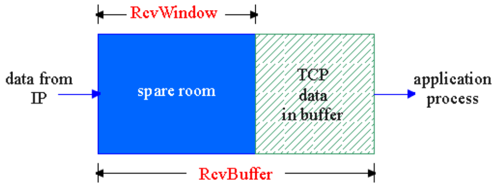
fast retransmit

Chapter 3 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
- Reliable Multicast

TCP Flow Control

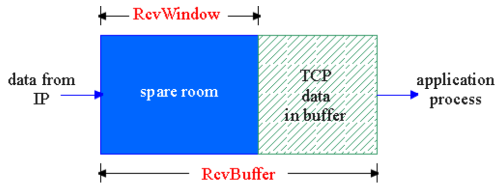
- receive side of TCP connection has a receive buffer:



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

- app process may be slow at reading from buffer
- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= **`RcvWindow`**

= **`RcvBuffer-[LastByteRcvd - LastByteRead]`**

- Rcvr advertises spare room by including value of **`RcvWindow`** in segments
- Sender limits unACKed data to **`RcvWindow`**
  - guarantees receive buffer doesn't overflow

Chapter 3 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
- Reliable Multicast

TCP Connection Management

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

- initialize TCP variables:
  - seq. #s
  - 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();
```

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

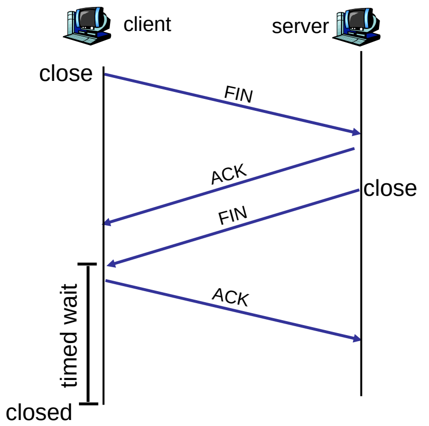
TCP Connection Management (cont.)

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.



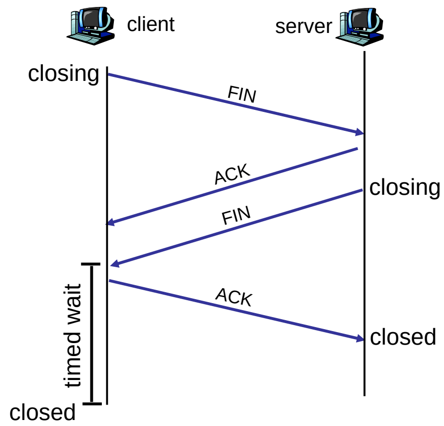
TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

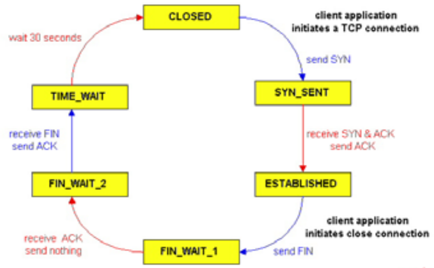
- Enters “timed wait” - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Note: with small modification, can handle simultaneous FINs.

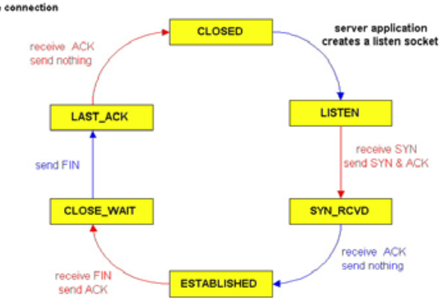


TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



Chapter 3 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
- Reliable Multicast

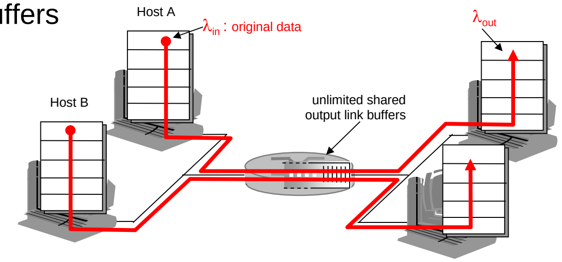
Principles of Congestion Control

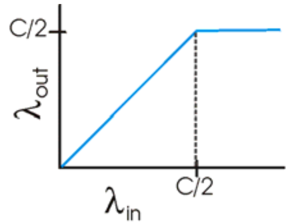
Congestion:

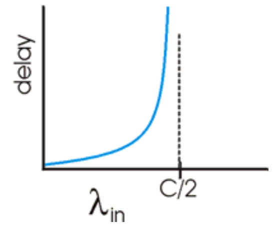
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



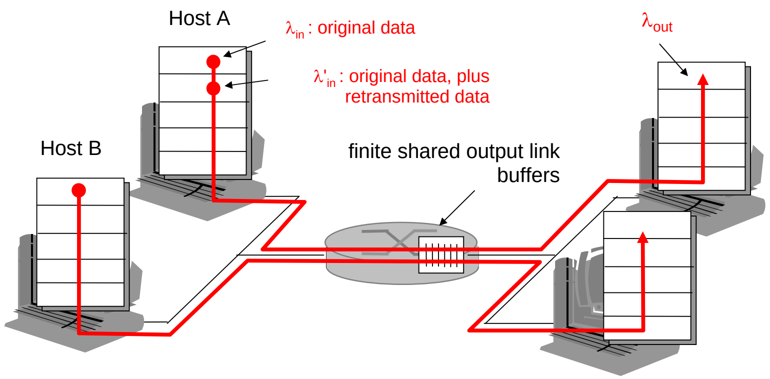




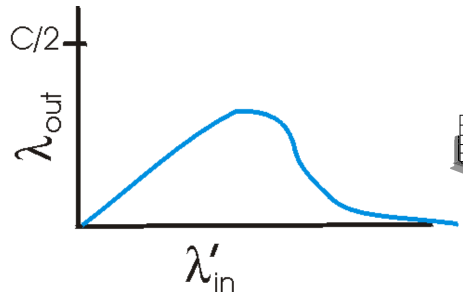
- large delays when congested
- maximum achievable throughput

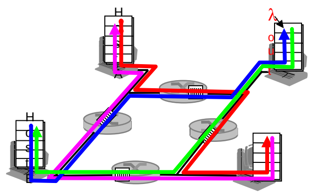
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet



Causes/costs of congestion: scenario 3





Another “cost” of congestion:

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

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

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

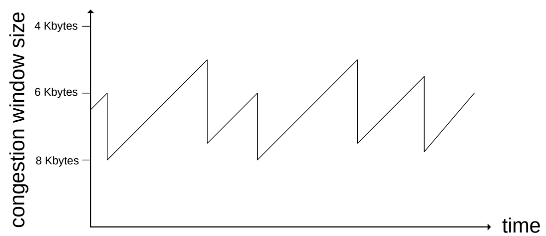
Chapter 3 outline

- 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
- Reliable Multicast

TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`CongWin`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **`CongWin`** in half after loss

Saw tooth behavior: probing for bandwidth



TCP Congestion Control: details

- sender limits transmission:

  the amount of unacked data at sender is limited by `CongWin`:

  `LastByteSent-LastByteAcked`

  ≤ `CongWin`
- Roughly:

$$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$$

- **`CongWin`** is dynamic, function of perceived network congestion
- **Self-clocking**: sender may send additional segments when acks arrive

How does sender perceive congestion?
- loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**`CongWin`**) after loss event

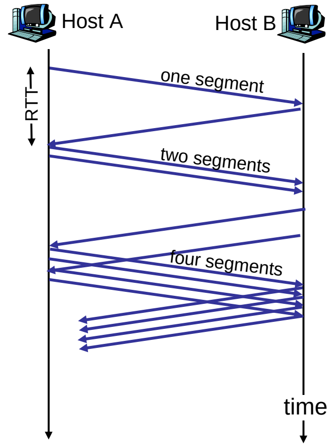
three mechanisms:
- AIMD
- slow start
- conservative after timeout events

TCP Slow Start

- When connection begins, **`CongWin`** = 1 MSS
  - Example:
    MSS = 1000 bytes, RTT = 100 msec
  - initial rate $\approx$ CongWin/RTT = 80 kbit/s
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event

TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double **`CongWin`** every RTT
  - done by incrementing **`CongWin`** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement: inferring loss

- After 3 dup ACKs:
  - **`CongWin`** is cut in half
  - window then grows linearly
- But after timeout event:
  - **`CongWin`** instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

Philosophy:

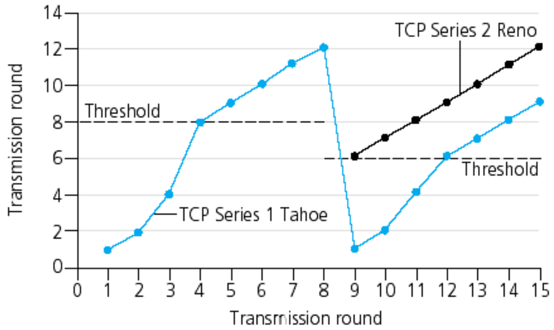
- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

Refinement

- Q: When should the exponential increase switch to linear?
- A: When **`CongWin`** gets to 1/2 of its value before timeout.

Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of **`CongWin`** just before loss event



- TCP Reno: Fast Recovery after 3 dup Acks

Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs,
  Threshold set to **CongWin/2**
  and **CongWin** set to **Threshold** (Fast Recovery, TCP Reno)
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

TCP sender congestion control

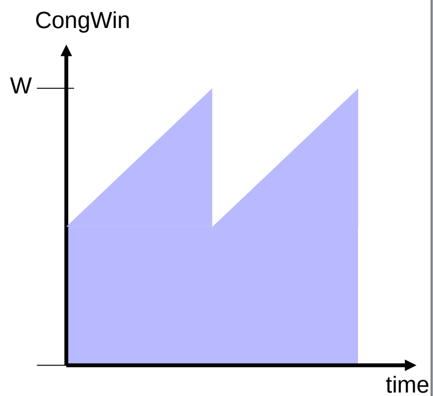
State	Event	TCP Sender Action	Comment
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP summary

- Connection-oriented: SYN, SYNACK; FIN
- Retransmit lost packets; in-order data: sequence no., ACK no.
- ACKs: either piggybacked, or no-data pure ACK packets if no data travelling in other direction
- Don't overload receiver: **RcvWin**
  - **RcvWin** advertised by receiver
- Don't overload network: **CongWin**
  - **CongWin** affected by receiving ACKs
- Sender buffer = min {**RcvWin**, **CongWin** }
- Congestion control:
  - Slow start: exponential growth of **CongWin**
  - Congestion avoidance: linear growth of **CongWin**
  - Timeout; duplicate ACK: shrink **CongWin**
- Continuously adjust RTT estimation

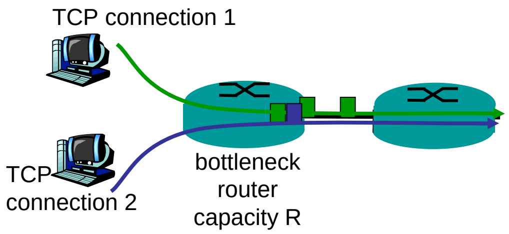
TCP throughput

- What's the average throughput of TCP as a function of window size and RTT?
  - Ignore slow start
  - Let W be the 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: 0.75 W/RTT



TCP Fairness

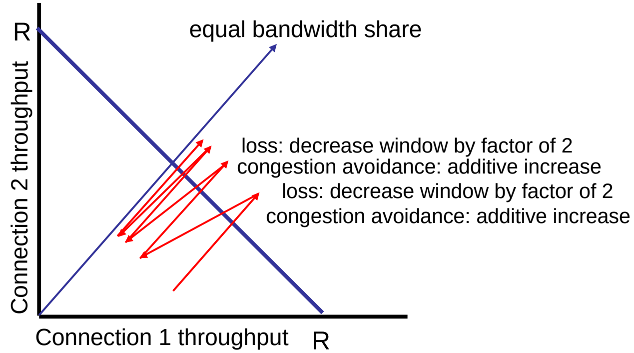
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



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 !

Chapter 3 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**
- Reliable Multicast

Internet Protocol Stack

- **The Internet Protocol Stack**

Session, Presentation, Application Layer	Application
Transport Layer	UDP / TCP / SCTP
Network Layer	IP
Physical + Data Link Layer	Network Interface (Ethernet, PPP, …)

- **Why another transport layer protocol?**

Contents

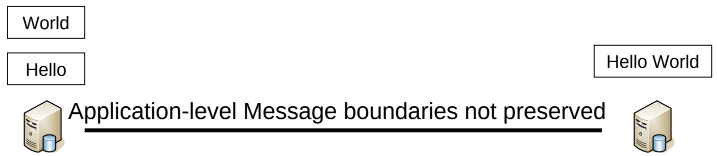
- **Limitations of UDP and TCP**
- **The Stream Control Transmission Protocol (SCTP)**
  - Association setup / stream setup
  - Message types
  - Partial Reliability
  - Multi-Homing support
  - Congestion control

User Datagram Protocol

- **Message oriented**
  - Sending application writes a N byte message
  - Receiving application reads a N byte message
- **Unreliable**
  - Lost packets will not be retransmitted
- **Unordered delivery**
  - Packets may be re-ordered in the network

Transmission Control Protocol

- Connection/Stream oriented. Not message oriented



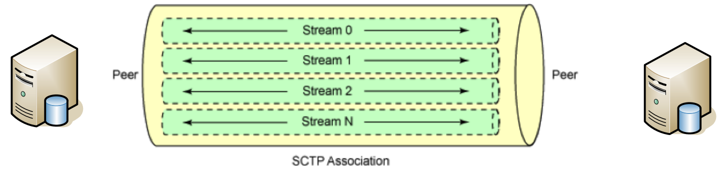
- Reliable transmission
  - Lost packets are retransmitted
  - Retransmission will be repeated until acknowledgment is received
- In-order delivery
  - Segments n + 1, n + 2, n + 3, will be delivered after segment n
- Congestion control
  - TCP tries to share bandwidth equally between all end-points

Problems

- Certain applications pose problems to UDP and TCP
- TCP: Head-of-line blocking with video streaming
  - Frames 2,3,4 arrived but cannot be shown because frame 1 is missing
  - ⇨ Video will stop until frame 1 is delivered
- UDP:
  - Unordered delivery: Second image is delivered after first image
  - Packet loss: Certain frames get lost ⇨ low video/audio quality
  - No congestion control
- Example: Internet-Telephony
  - Two types of traffic:
    - Signalling traffic: should be delivered reliable + in-order (TCP)
    - Voice traffic: should not suffer from head-of-line blocking (UDP)
  - Need to manage two sockets
- SCTP can deal with these problems

SCTP Features at a glance

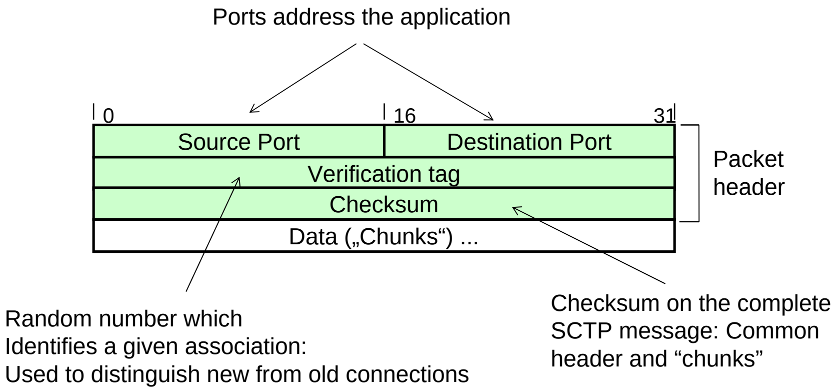
- **Connection and message oriented**
  - SCTP builds an "association" between two peers
  - Association can contain multiple "streams"
  - Messages are sent over one of the streams



- **Partial reliability**
  - "Lifetime" defined for each message
    - Retransmission of a message is performed during its lifetime
  - Messages can be delivered unreliable, full reliable or partial reliable
- **Multi-Homing**
  - SCTP can use multiple IP addresses

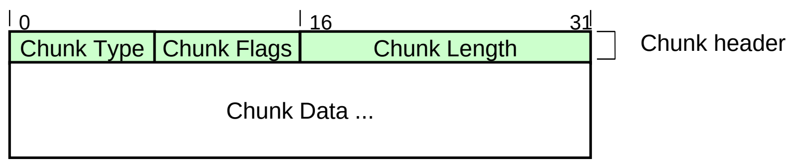
SCTP Message Format

- **Common header format**
  - 12 byte header
  - included in every SCTP message



SCTP Chunk Format

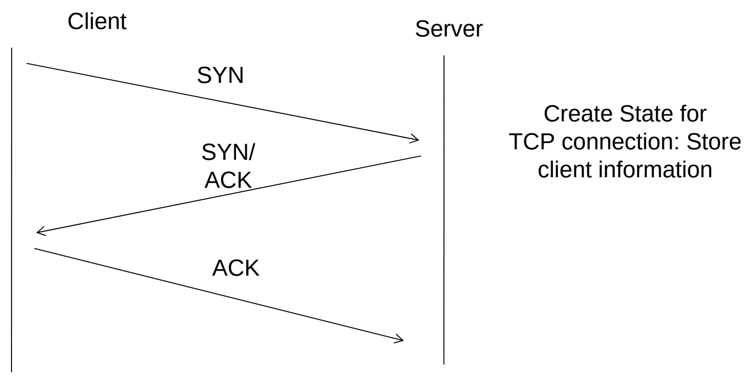
- Data and signaling information is transported in chunks
  - One or more chunks in a SCTP message
  - Each chunk type has a special meaning:
    - INIT, INIT-ACK, COOKIE, COOKIE-ACK ⇨ Connection setup
    - DATA ⇨ Transports user data
    - SACK ⇨ Acknowledge Data
- Common chunk format



- Additional formats are defined for specific chunk types

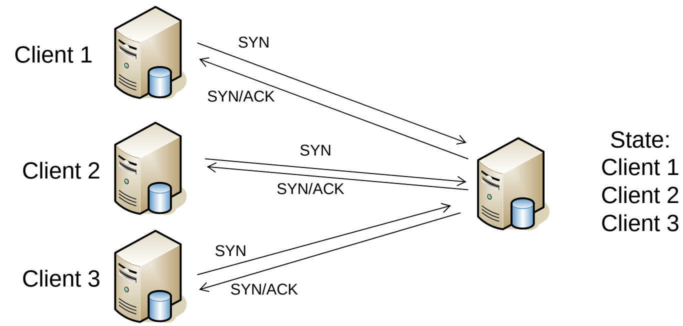
Connection Setup

- **TCP connection setup**



- **Known Problem: TCP SYN-Flooding**

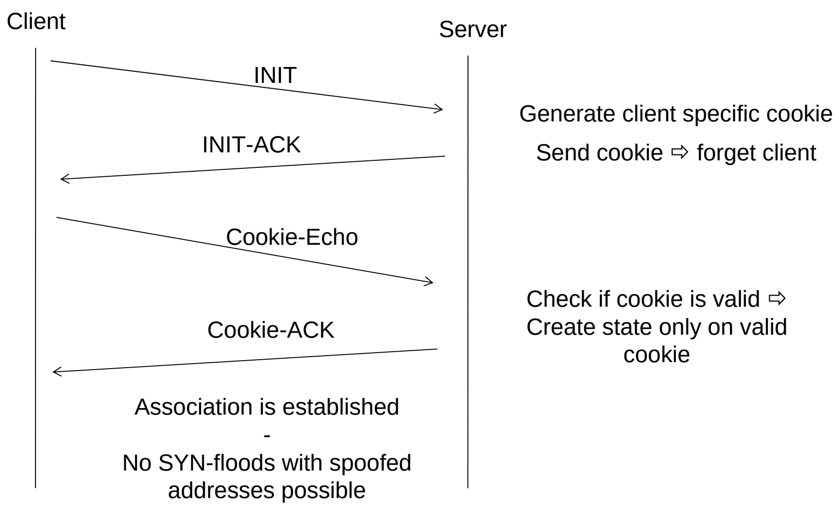
SYN Flodding



- Clients send SYN-Packets but do not respond to SYN-ACK
  - Usually done by a single client that performs IP address spoofing
  - Works because only a single forged packet is necessary
- ⇨ Server has to store state until a TCP timeout occurs
  - Leads to resource exhaustion
    - ⇨ Server cannot accept any more connections

SCTP Association Setup

- **Solution to SYN-Flood problem: Cookies**



Data Transmission

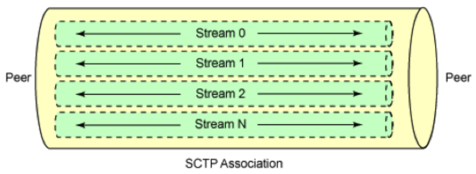
- Application data is transmitted in Data Chunks
  - A data chunk is associated to a stream (Stream Identifier S)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 0    | Reserved|U|B|E|          Length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              TSN                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Stream Identifier S      |   Stream Sequence Number n    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Payload Protocol Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               /
/                 User Data (seq n of Stream S)                 /
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- TSN (Transport Sequence Number)
  - Global Sequence Number
  - Similar to TCP sequence number, used for retransmissions
- Stream sequence number
  - Necessary for per-stream transmission reliability

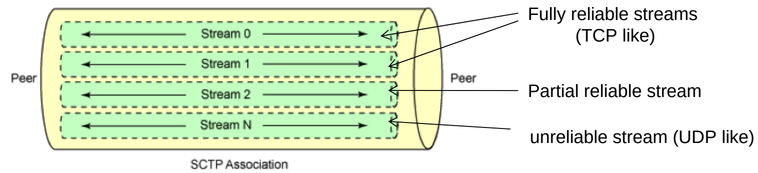
Transmission reliability (1)

- TCP
  - Packets are transmitted fully reliable ⇨ retransmitted until received
  - Packets are delivered in-order to the application
  - Slow start and congestion avoidance for congestion control
- UDP
  - Packets are transmitted fully unreliable ⇨ never retransmitted
  - No re-ordering ⇨ packet order may be changed at the receiver
  - No congestion control
- SCTP can do both and more, in a stream-specific way



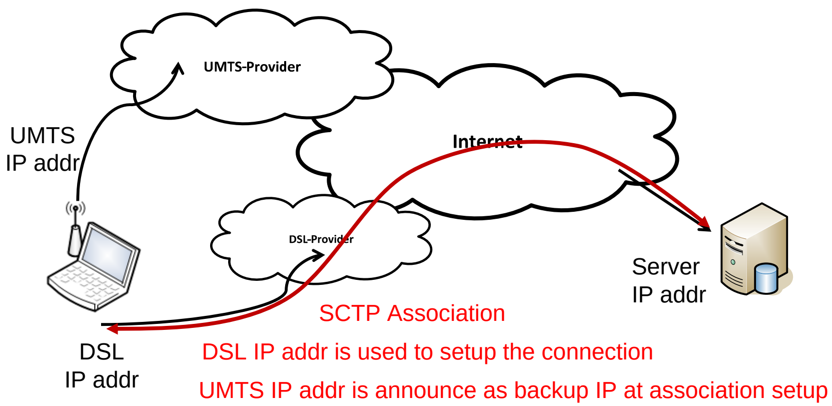
Transmission reliability (2)

- Why multiple streams?
  - Solves head of line blocking
  - No firewall issues (only one port for several streams)
  - Partial Reliability Extension (PR-SCTP) for different reliability levels
- PR-SCTP
  - Allows to set a lifetime parameter for each stream
  - Lifetime specifies how long the sender should try to retransmit a packet
  - Allows to mix reliable and unreliable streams



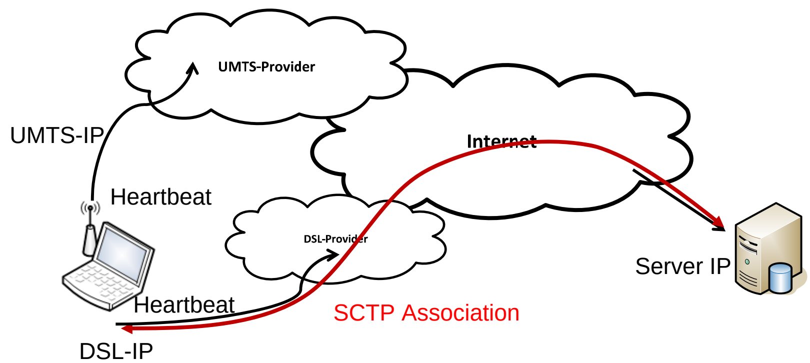
Multi-Homing: Association setup

- SCTP chooses one IP address at association setup
  - IP address can be specified by user



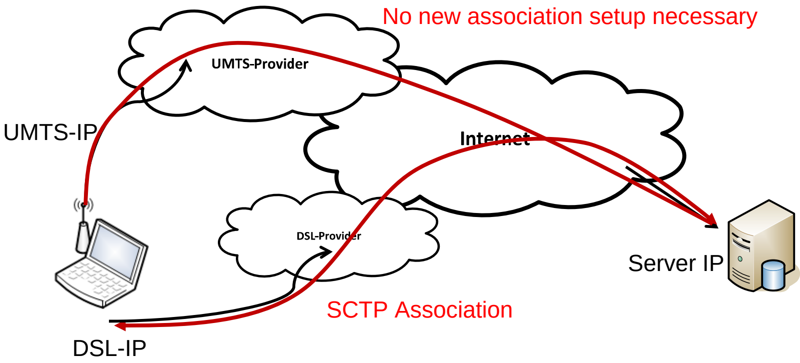
Multi-Homing

- Heartbeat messages are periodically sent to check link availability



Multi-Homing

- Changes occur when the default link is found to be broken
  - Is identified because of packet loss (data or heartbeat)
  - Consequence: SCTP will resume on the backup link



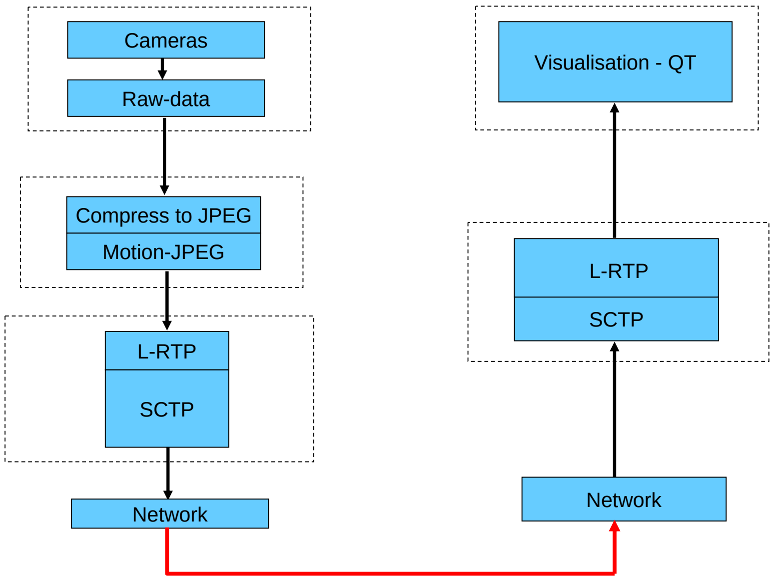
SCTP Example Scenario

- Real-time transmission of video streams and control data in vehicular scenario

Server: vehicle with embedded PC (Linux)

Client: Unix/Windows

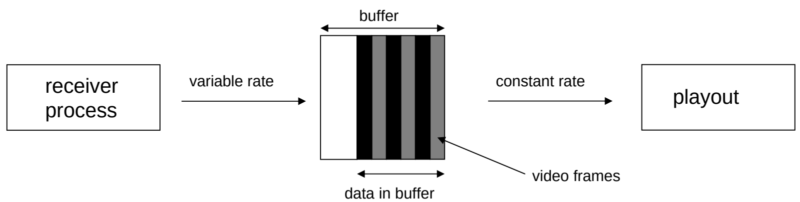
Protocol Architecture



Leightweight - RTP

L-RTP implementation:

- Timestamping for synchronisation
- Packet loss detection
- Buffering



SCTP Deployment

- SCTP has attractive features
  - but to which extent is it used?
- Why do we use HTTP over TCP for Video Streaming?
- Why is IP Multicast not generally deployed?
  - Because HTTP over TCP streaming just works „good enough“
- Firewall and NAT issues
  - Most home routers simply can‘t translate SCTP
- Implementations
  - Currently no native Windows support (only userspace lib)
- BUT: mandatory for some newly developed protocols such as IPFIX (IP Flow Information Export)

Chapter 3 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
- **Reliable Multicast**

Many Uses of Multicasting

- Teleconferencing
- Distributed Games
- Software/File Distribution
- Video Distribution
- Replicated Database Updates

⇨ multicast transport is done differently for each application

Multicast Application Modes

- Point-to-Multipoint:
  Single Source, Multiple Receivers
- Multipoint-to-Multipoint:
  Multiple Sources, Multiple Receivers
- Sources are receivers
- Sources are not receivers

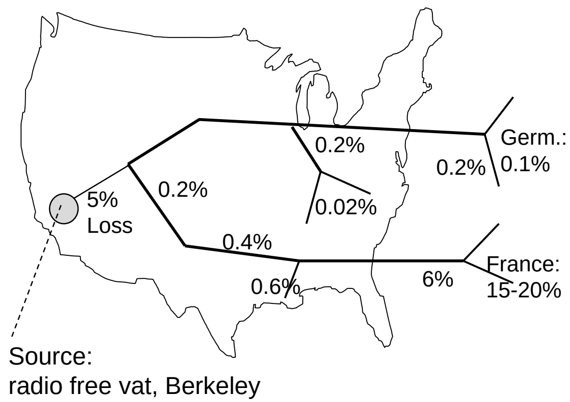
Classification of Multicast Applications

Transport service type	*Fully reliable multicast*	*Real-time multicast*
Single source: 1:N	Multicast-FTP; Software update	Audio-visual conference; Continuous Media Dissemination
Multiple Sources M:N	CSCW; Distributed computing	DIS; VR

- CSCW: Computer Supported Cooperative Work
- DIS: Distributed Interactive Simulation
- VR: Virtual Reality

Where Does Multicast Loss Occur

- Example measurements
  (April 96, Yajnik, Kurose, Towsely, Univ. Mass., Amherst)



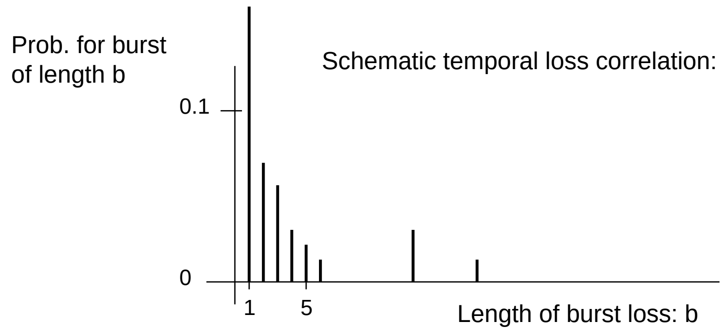
Simultaneous Packet Loss

- Q: distribution of number of receivers losing packet?
- Example dataset:
  47% packets lost somewhere
  5% shared loss
- Similar results across different datasets
- Models of packet loss (for protocol design, simulation, analysis):
  - star: end-end loss independent
  - full topology: measured per link loss independent
  - modified star: source-to-backbone plus star
    ⇨ good fit for example data set

Temporal Loss Correlation

Q: do losses occur singly or in “bursts”?

- occasional long periods of 100% loss
- generally isolated losses
- occasional longer bursts



Reliable Multicast Challenge

- How to transfer data reliably from source to R receivers
- scalability: 10s - 100s - 1000s - 10000s - 100000s of receivers
- heterogeneity
  - different capabilities of receivers (processing power, buffer, protocol capabilities)
  - different network conditions for receivers (bottleneck bandwidths, loss rates, delay)
- feedback implosion problem

ARQ: Alternatives for Basic Mechanisms

- Who retransmits
  - source
  - network
  - other group member.
- Who detects loss
  - sender based: waiting for all ACKs
  - receiver based: NACK, more receivers - faster loss detection.
- How to retransmit
  - Unicast
  - Multicast
  - Subgroup-multicast

Approaches

- shift responsibilities to receivers (in contrast to TCP: sender is responsible for large share of functionality)
- feedback suppression (some feedback is usually required)
- multiple multicast groups (e.g. for heterogeneity problems; can be used statically or dynamically)
- local recovery (can be used to reduce resource cost and latency)
- server-based recovery
- forward error correction (FEC)
  - FEC for unicast: frequently no particular gain
  - FEC for multicast: gain may be tremendous!

Classification of Multicast Error Control

Multicast Error Recovery

Centralized Error Recovery (CER): *Source* retransmits

Distributed Error Recovery (DER): retransmission by server or receiver

grouped (local): Multicast group is *partitioned into subgroups*

ungrouped (global): *All* group members participate in error recovery

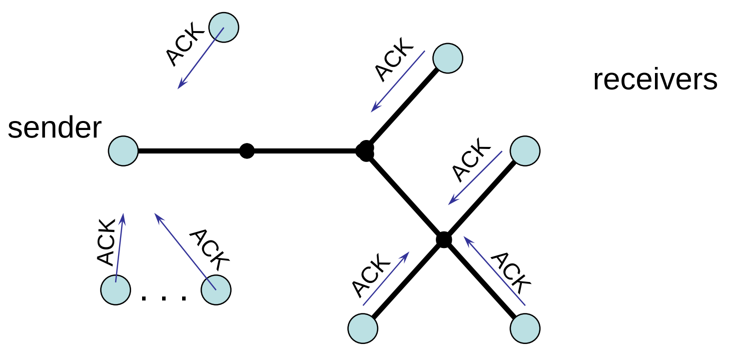
Reliable Multicast: Building Blocks

- Elements from Unicast:
  - Loss detection
    - Sender-based (ACK): 1 ACK per receiver and per packet; Sender needs a table of per-receiver ACK
    - Receiver-based (NAK): distributed over receivers; potentially only 1 NAK per lost packet
  - Loss recovery: ARQ vs. FEC
- Additional new Elements for Multicast:
  - Mechanisms for control message **Implosion Avoidance**
  - Mechanisms to deal with *heterogeneous receivers*

Feedback Processing

- Assume: R Receivers, independent packet loss probability p
- Calculate feedback per packet:
  - average number of ACKs: R - pR
  - average number of NAKs: pR
  - ⇨ more ACKs than NAKs
- Processing: higher throughput for receiver-based loss detection
- Reliability needs ACKs
  (No NAK does not mean successful reception)
  - ⇨ use NAK for loss signalling
  - ⇨ use ACKs at low frequency to ensure reliability

Multicast Challenge: Feedback Implosion Problem



NAK Implosion

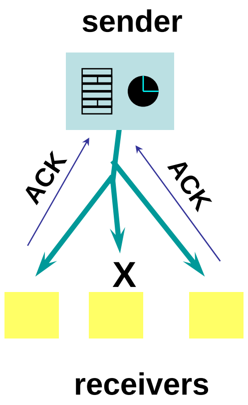
- Shared loss: All receivers loose same packet: All send NAK
  ⇨ NAK implosion
- Implosion avoidance techniques
  - Cluster/Hierarchy
  - Token
  - Timers

  For redundant feedback additionally:
  - Feedback suppression (e.g. multicast NAKs, receiver back off randomly)

  Drawback of implosion avoidance techniques : delay
- Fast NAKs (risk of NAK implosion):
  - Fast retransmission
  - Smaller sender/receiver buffer

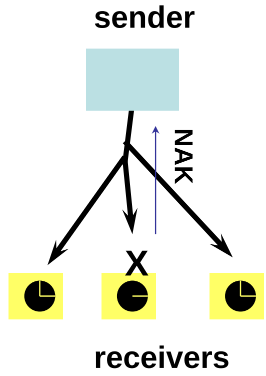
Sender Oriented Reliable Multicast

- Sender:
  multicasts all (re)transmissions
  - selective repeat
  - use of timeouts for loss detection
  - ACK table
- receiver: ACKs received packets
- Note: group membership important
- Example (historic):
  Xpress Transport Protocol (XTP)
  - extension of unicast protocol



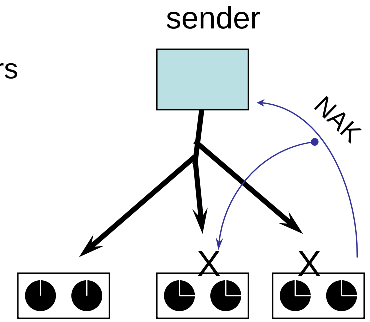
Receiver Oriented Reliable Multicast

- Sender: multicasts (re)transmissions
  - selective repeat
  - responds to NAKs
- Receiver: upon detecting packet loss
  - sends pt-pt NAK
  - timers to detect lost retransmission
- Note: easy to allow joins/leaves



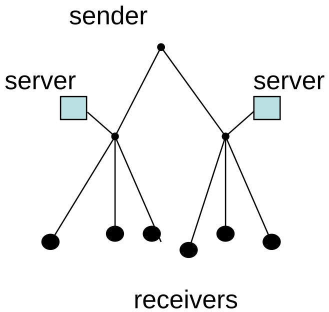
Feedback Suppression

- randomly delay NAKs
- multicast to all receivers
  + reduce bandwidth
  - additional complexity at receivers (timers, etc)
  - increase latencies (timers)
- similar to CSMA/CD (⇨ later)



Server-based Reliable Multicast

- first transmisions: multicast to all receivers and servers
- each receiver assigned to server
- servers perform loss recovery
- servers can be subset of receivers or provided by network
- can have more than 2 levels



Assessment:
- clear performance benefits
- how to configure
  - static/dynamic
  - many-many

Local Recovery

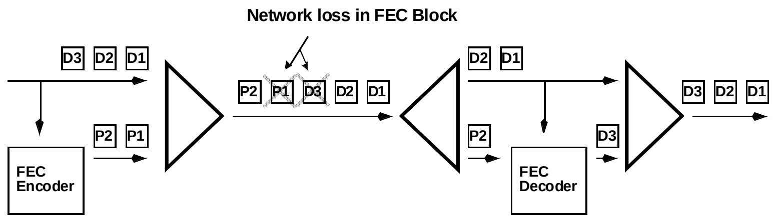
- lost packets recovered from nearby receivers
- deterministic methods
  - impose tree structure on receivers with sender as root
  - receiver goes to upstream node on tree
- self-organizing methods
  - receivers elect nearby receiver to act as retransmitter
- hybrid methods

Issues with Server- and Local Based Recovery

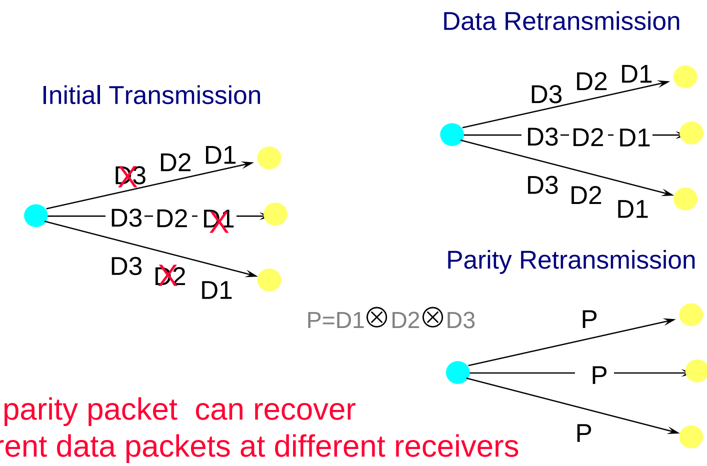
- how to configure tree
- what constitutes a local group
- how to permit joins/leaves
- how to adapt to time-varying network conditions

Forward Error Correction (FEC)

- k original data packets form a **Transmission Group (TG)**
- h parity packets derived from the k data packets
- any k received out of k+h are sufficient
- Assessment
  + allows to recover lost packets
  - overhead at end-hosts
  - increased network load may increase loss probability



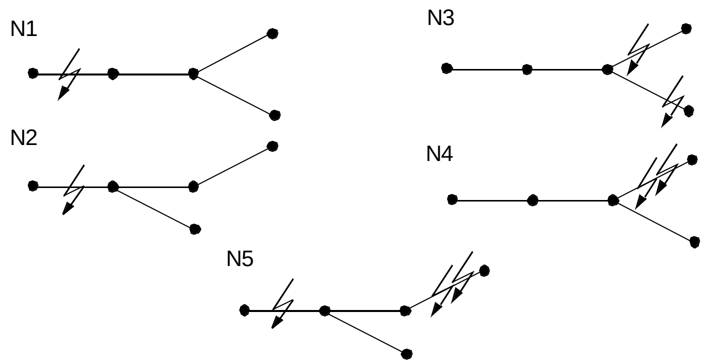
Potential Benefits of FEC



One parity packet can recover different data packets at different receivers

Influence of topology: Selected Scenarios for Modeling Heterogeneity

- Loss: on shared links / on individual links
- Loss: homogeneous/heterogeneous probability
- RTT: homogeneous/heterogeneous.



Scenario-specific Selection of Mechanisms

- FEC is of particular benefit in the following scenarios:
  - Large groups
  - No feedback
  - Heterogeneous RTTs
  - Limited buffer.
- ARQ is of particular benefit in the following scenarios:
  - Herterogeneous loss
  - Loss in shared links of multicast tree dominates
  - Small groups (Statistic by AT&T: on average < 7 participants in conference)
  - Non-interactive applications.
- ARQ by local recovery:
  - large groups (good for individual losses, heterogeneous RTT).

Chapter 3: Summary

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