# TCP and Congestion Control

EECS 122
Valentine's Day, 2006



- HW 2: Ethereal Labs

Lecture today:

Wrap up on reliable data transfer.

See how principles are applied to TCP

Talk about congestion control.



# Forward Erasure/Error Correction: A Different Approach to RDT

- Our approach to reliable data delivery is based on ACKs and retransmissions, i.e. feedback.
- Long RTTs => long delays and/or low throughput
- An alternative approach is via forward corrections for errors and losses.



# Error Dectection and Loss Recovery

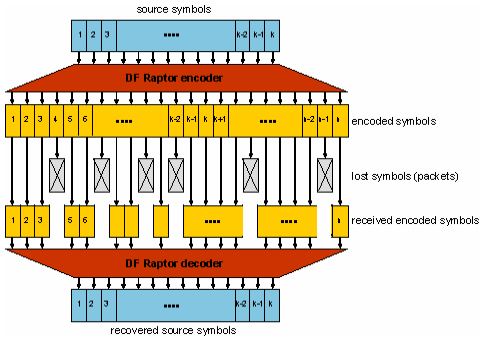
Message to Hong Kong:

Hope this will be our last Valentine's Day apart.

One extra parity-check "word" can detect error.
It can also recover from a single loss.
With more parity-check "words", one can recover from multiple losses.



# Example: Fountain Codes






# TCP

- Overview
- Reliable data transfer
- Flow control
- Congestion control

## 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*

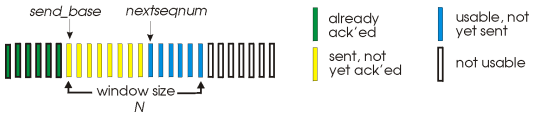
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver




## TCP reliable data transfer

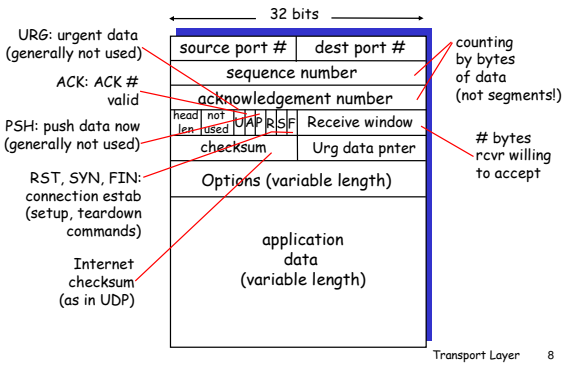
- TCP creates rdt service on top of IP's unreliable service
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer

- Retransmissions are triggered by:
  - timeout events
  - duplicate acks (fast retransmit)
- TimeOut intervals often doubled after a timeout.






## TCP segment structure






## 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 seq. #'s and ACKs

Seq. #'s:

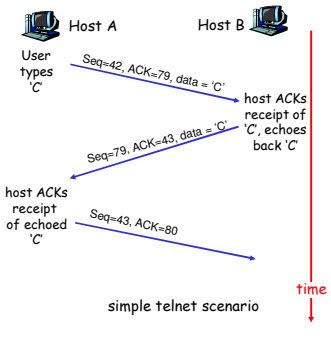
- byte stream "number" of first byte in segment's data

ACKs:

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

Full-duplex:

ACK's for one direction are piggybacked on data segments in the other direction






## TCP= Hybrid Go-Back-N and Selective Repeat

- Cumulative ACK (like GBN)
- Out-of-order segments often buffered at receiver and not discarded (but no individual ACK sent)

## 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**



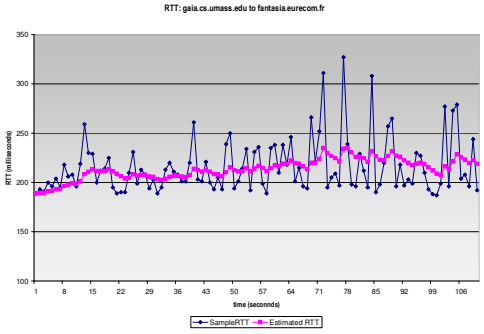
## TCP Round Trip Time and Timeout

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

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



## 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
```



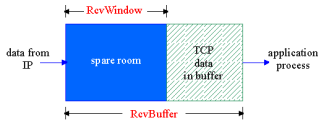
## Fast Retransmit

- 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



## TCP Flow Control

- receive side of TCP connection has a receive buffer:

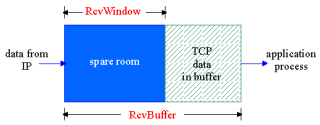



- app process may be slow at reading from buffer

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




(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