

## End-to-End Protocols

### Reliable Byte-Stream (TCP)

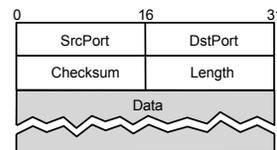
#### Outline

- Connection Establishment/Termination
- Sliding Window Revisited
- Flow Control
- Adaptive Timeout

- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

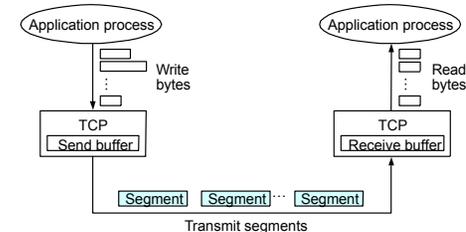
### Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see **/etc/services** on Unix
- Header format
- Optional checksum
  - pseudo header + UDP header + data



### TCP Overview

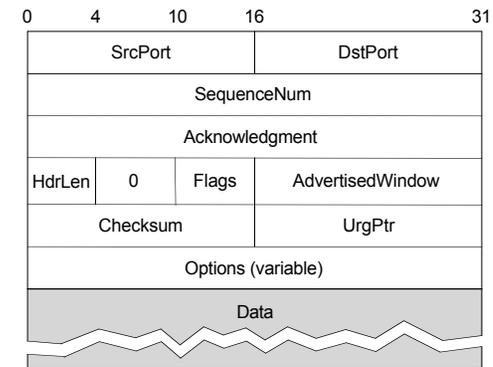
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



## Data Link Versus Transport

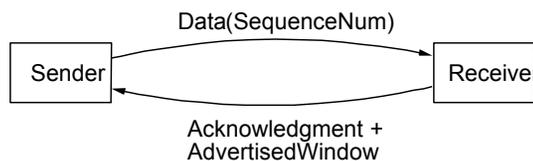
- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

## Segment Format



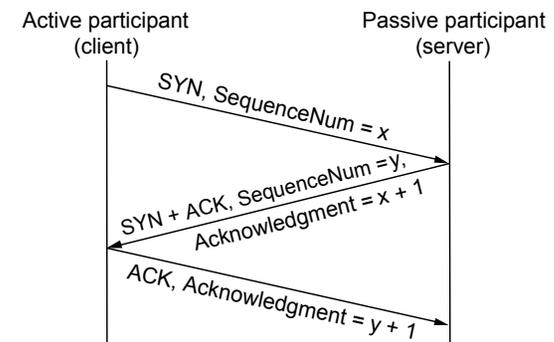
## Segment Format (cont)

- Each connection identified with 4-tuple:
  - (SrcPort, SrcIPAddr, DsrPort, DstIPAddr)
- Sliding window + flow control
  - acknowledgment, SequenceNum, AdvertisedWinow

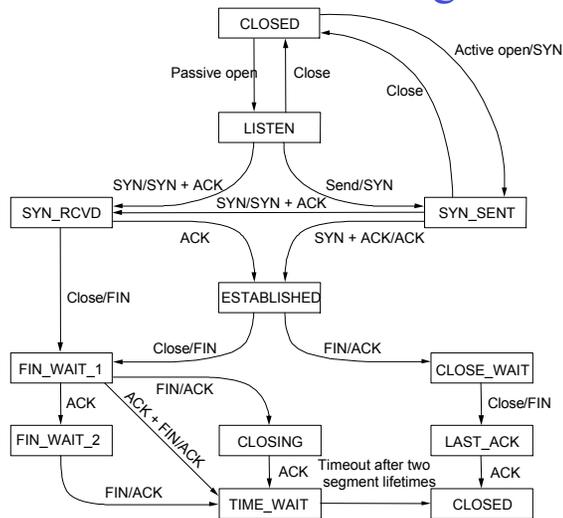


- Flags
  - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
  - pseudo header + TCP header + data

## Connection Establishment and Termination



## State Transition Diagram

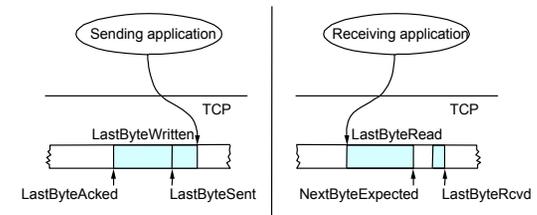


Spring 2005

CS 461

9

## Sliding Window Revisited



- Sending side
  - $\text{LastByteAcked} \leq \text{LastByteSent}$
  - $\text{LastByteSent} \leq \text{LastByteWritten}$
  - buffer bytes between **LastByteAcked** and **LastByteWritten**
- Receiving side
  - $\text{LastByteRead} < \text{NextByteExpected}$
  - $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
  - buffer bytes between **NextByteRead** and **LastByteRcvd**

Spring 2005

CS 461

10

## Flow Control

- Send buffer size: **MaxSendBuffer**
- Receive buffer size: **MaxRcvBuffer**
- Receiving side
  - $\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$
  - $\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{NextByteExpected} - \text{NextByteRead})$
- Sending side
  - $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$
  - $\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$
  - $\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$
  - block sender if  $(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSenderBuffer}$
- Always send ACK in response to arriving data segment
- Persist when **AdvertisedWindow = 0**

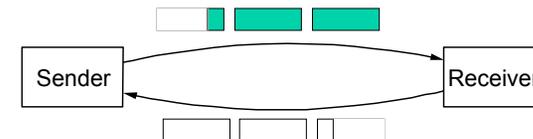
Spring 2005

CS 461

11

## Silly Window Syndrome

- How aggressively does sender exploit open window?



- Receiver-side solutions
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements

Spring 2005

CS 461

12

## Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking
- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack'd data in transit: buffer it until ACK arrives
    - else: send it

## Protection Against Wrap Around

- 32-bit **SequenceNum**

Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

## Keeping the Pipe Full

- 16-bit **AdvertisedWindow**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

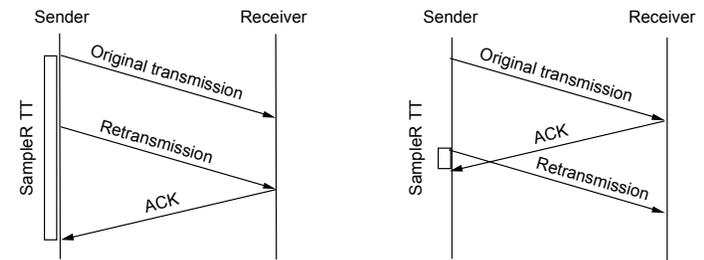
## TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS)
- Shift (scale) advertised window

## Adaptive Retransmission (Original Algorithm)

- Measure **SampleRTT** for each segment / ACK pair
- Compute weighted average of RTT
  - $\mathbf{EstRTT} = \alpha \times \mathbf{EstRTT} + \beta \times \mathbf{SampleRTT}$
  - where  $\alpha + \beta = 1$
  - $\alpha$  between 0.8 and 0.9
  - $\beta$  between 0.1 and 0.2
- Set timeout based on **EstRTT**
  - $\mathbf{TimeOut} = 2 \times \mathbf{EstRTT}$

## Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission

## Jacobson/ Karels Algorithm

- New Calculations for average RTT
- $\mathbf{Diff} = \mathbf{SampleRTT} - \mathbf{EstRTT}$
- $\mathbf{EstRTT} = \mathbf{EstRTT} + (\delta \times \mathbf{Diff})$
- $\mathbf{Dev} = \mathbf{Dev} + \delta (|\mathbf{Diff}| - \mathbf{Dev})$ 
  - where  $\delta$  is a factor between 0 and 1
- Consider variance when setting timeout value
- $\mathbf{TimeOut} = \mu \times \mathbf{EstRTT} + \phi \times \mathbf{Dev}$ 
  - where  $\mu = 1$  and  $\phi = 4$
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)