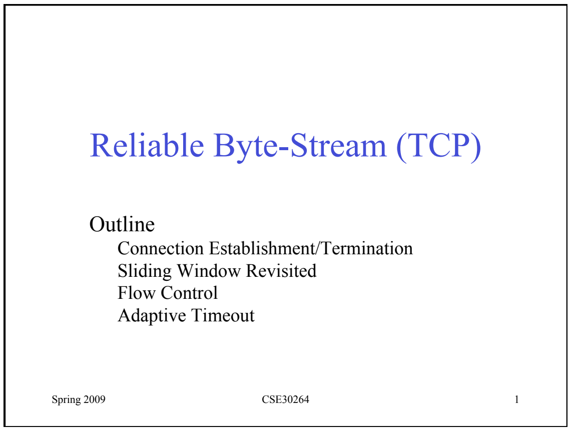
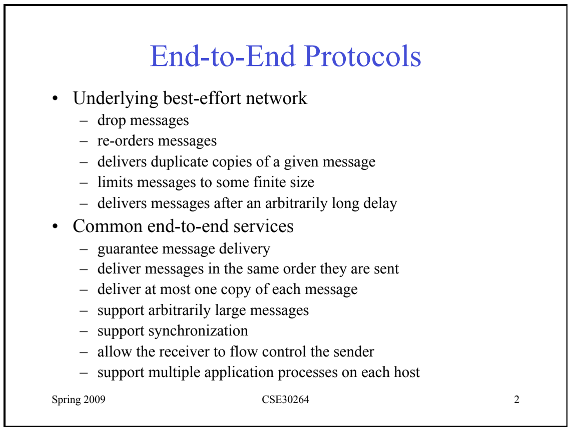


Reliable Byte-Stream (TCP)

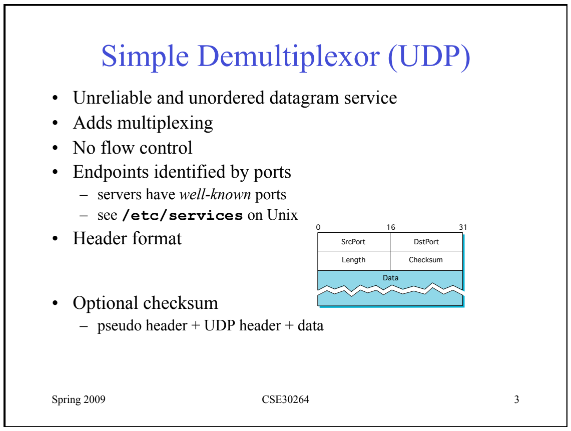
Outline

Connection Establishment/Termination
Sliding Window Revisited
Flow Control
Adaptive Timeout



End-to-End Protocols

- 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

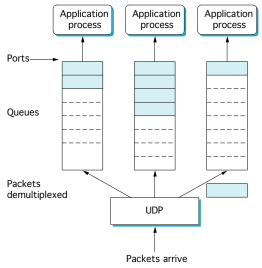


Simple Demultiplexor (UDP)

- 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

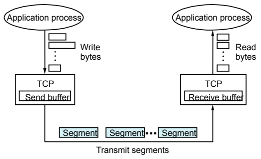
0
16
31
SrcPort
DstPort
Length
Checksum
Data

UDP

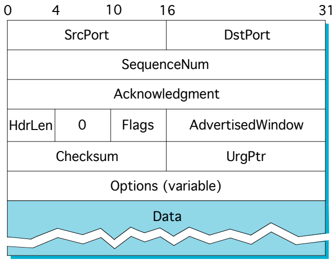


TCP Overview

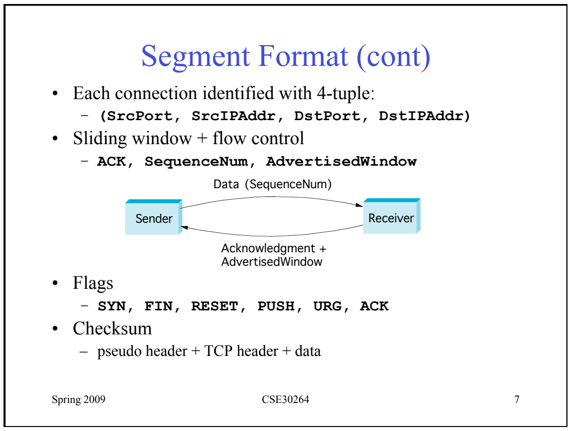
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Segment Format



0	4	10	16 31
SrcPort			DstPort
SequenceNum			
Acknowledgment			
HdrLen	0	Flags	AdvertisedWindow
Checksum			UrgPtr
Options (variable)			
Data			



Segment Format (cont)

- Each connection identified with 4-tuple:
  - `(SrcPort, SrcIPAddr, DstPort, DstIPAddr)`
- Sliding window + flow control
  - `ACK, SequenceNum, AdvertisedWindow`

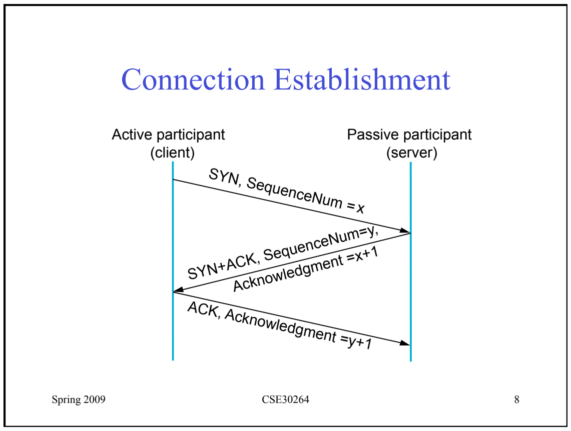
Data (SequenceNum)

Sender

Receiver

Acknowledgment +
AdvertisedWindow

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



Connection Establishment

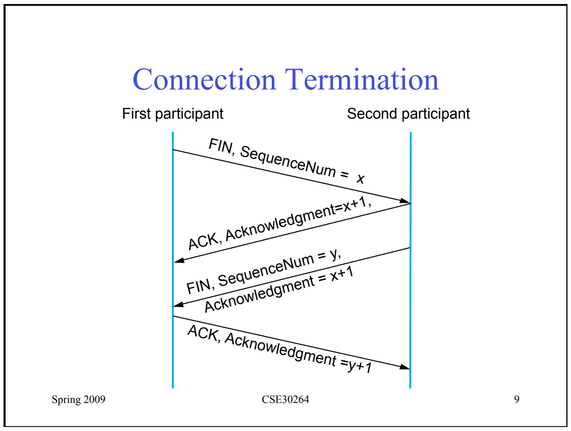
Active participant
(client)

Passive participant
(server)

SYN, SequenceNum = x

SYN+ACK, SequenceNum=y,
Acknowledgment =x+1

ACK, Acknowledgment =y+1



Connection Termination

First participant

Second participant

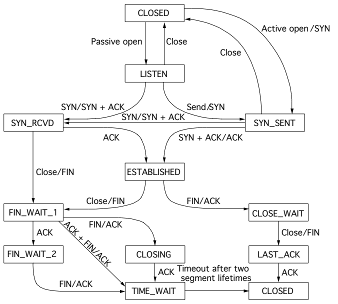
FIN, SequenceNum = x

ACK, Acknowledgment=x+1,

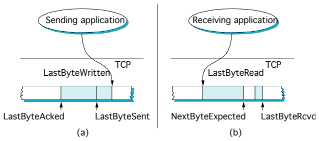
FIN, SequenceNum = y,
Acknowledgment = x+1

ACK, Acknowledgment =y+1

State Transition Diagram



Sliding Window Revisited



- Sending side
  - **LastByteAcked <=** **LastByteSent**
  - **LastByteSent <=** **LastByteWritten**
  - buffer bytes between **LastByteAcked** and **LastByteWritten**
- Receiving side
  - **LastByteRead <** **NextByteExpected**
  - **NextByteExpected <=** **LastByteRcvd +1**
  - buffer bytes between **LastByteRead** and **LastByteRcvd**

Flow Control

- Send buffer size: **MaxSendBuffer**
- Receive buffer size: **MaxRcvBuffer**
- Receiving side
  - **LastByteRcvd** - **LastByteRead** < = **MaxRcvBuffer**
  - **AdvertisedWindow** = **MaxRcvBuffer** - ((**NextByteExpected** - 1) - **LastByteRead**)
- Sending side
  - **LastByteSent** - **LastByteAcked** < = **AdvertisedWindow**
  - **EffectiveWindow** = **AdvertisedWindow** - (**LastByteSent** - **LastByteAcked**)
  - **LastByteWritten** - **LastByteAcked** < = **MaxSendBuffer**
  - block sender if (**LastByteWritten** - **LastByteAcked**) + *y* > **MaxSenderBuffer**
- Always send ACK in response to arriving data segment
- Persist when **AdvertisedWindow = 0**

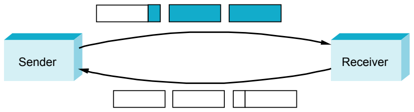
Protection Against Wrap Around

- 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

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

Nagle’s Algorithm

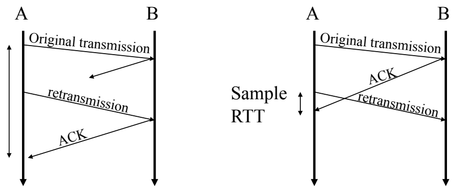
- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking

```
when application produces data to send
 if both the available data and the window >= MSS
   send a full segment
 else
   if there is unACKed data in flight
     buffer the new data until an ACK arrives
   else
     send all the new data now
```

Adaptive Retransmission

- Round-Trip Time Estimation:
  - wait at least one RTT before retransmitting
  - importance of accurate RTT estimators:
    - Low RTT -> unneeded retransmissions
    - High RTT -> poor throughput
  - RTT estimator must adapt to change in RTT
    - But not too fast, or too slow!
  - problem: If the instantaneously calculated RTT is 10, 20, 5, 12, 3 , 5, 6; what RTT should we use for calculations?
  - EstimatedRTT = α * EstimatedRTT + (1 - α) SampleRTT
  - recommended value for α: 0.8 - 0.9
  - retransmit timer set to β RTT, where β = 2

Retransmission Ambiguity



Karn/Partridge Algorithm

- Accounts for retransmission ambiguity
- If a segment has been retransmitted:
  - don’t count RTT sample on ACKs for this segment
  - reuse RTT estimate only after one successful transmission
  - double timeout after each retransmission

Jacobson/Karels Algorithm

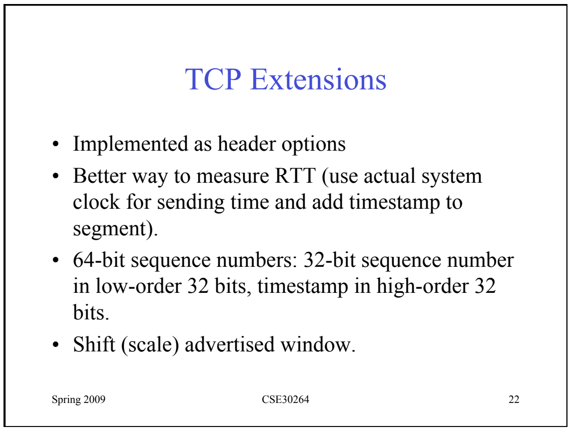
- Key observation:
  - using β RTT for timeout doesn’t work
  - at high loads round trip variance is high
- Solution:
  - if D denotes mean variation
  - timeout = RTT + 4D

Jacobson/Karels Algorithm

- New Calculations for average RTT
- Diff = SampleRTT - EstimatedRTT
- EstimatedRTT = EstimatedRTT + (d * Diff)
- Dev = Dev + d * (|Diff| - Dev)
  - where d is a factor between 0 and 1
- Consider variance when setting timeout value
- TimeOut = m * EstimatedRTT + f * Dev
  - where m = 1 and f = 4

Record Boundaries

- Byte-stream protocol: write 8+2+20 bytes and read 5+5+5+5+5+5 (loop).
- TCP offers two features to insert record boundaries:
  - URG flag
  - push operation



TCP Extensions

- Implemented as header options
- Better way to measure RTT (use actual system clock for sending time and add timestamp to segment).
- 64-bit sequence numbers: 32-bit sequence number in low-order 32 bits, timestamp in high-order 32 bits.
- Shift (scale) advertised window.