Reliable Byte-Stream (TCP)

Outline

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

1

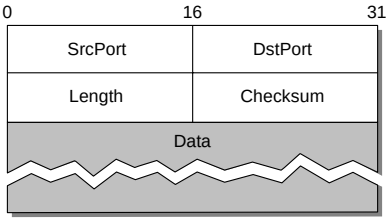
End-to-End Protocols

- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits packet (not message) 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 between sender and receiver
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

2

Simple Demultiplexor (UDP)

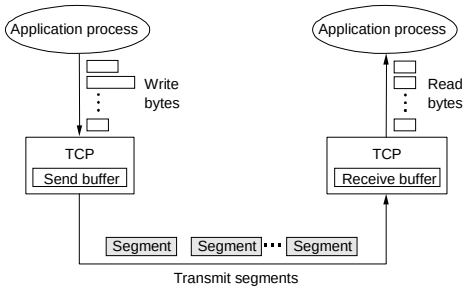
- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control or error control
  - no need for sender-side buffer)
- Endpoints identified by ports
  - servers listens at *well-known* ports!
  - see **`/etc/services`** on Unix
- Header format



- Optional checksum
  - psuedo header (IP.src, IP.dsest, IP.proto, UDP.len) + 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

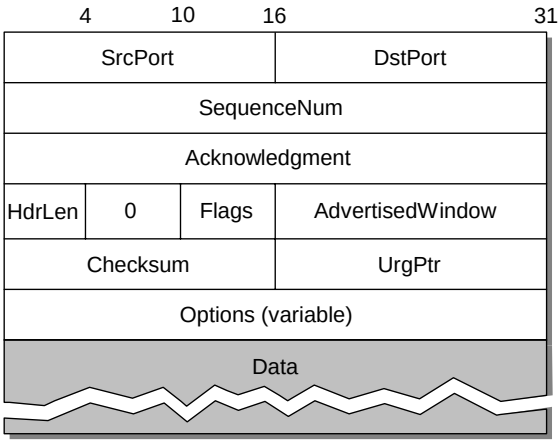


Data Link Versus End-to-End Transport

- 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

5

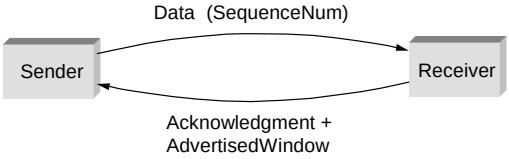
Segment Format



6

Segment Format (cont)

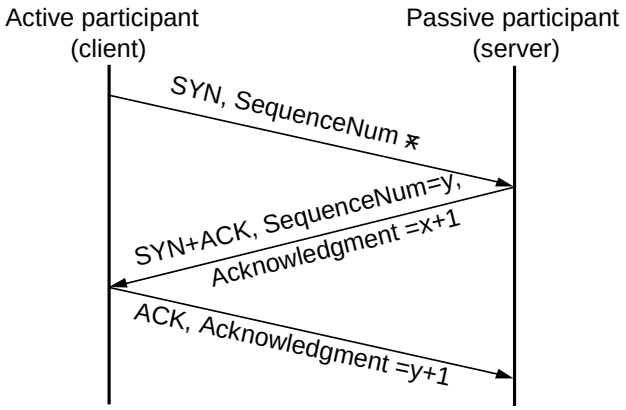
- Each connection identified with 4-tuple:
  - **(SrcPort, SrcIPAddr, DsrPort, DstIPAddr)**
- Sliding window + flow control
  - **acknowledgment, SequenceNum, AdvertisedWinow**



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

7

Connection Establishment and Three-Way Handshake

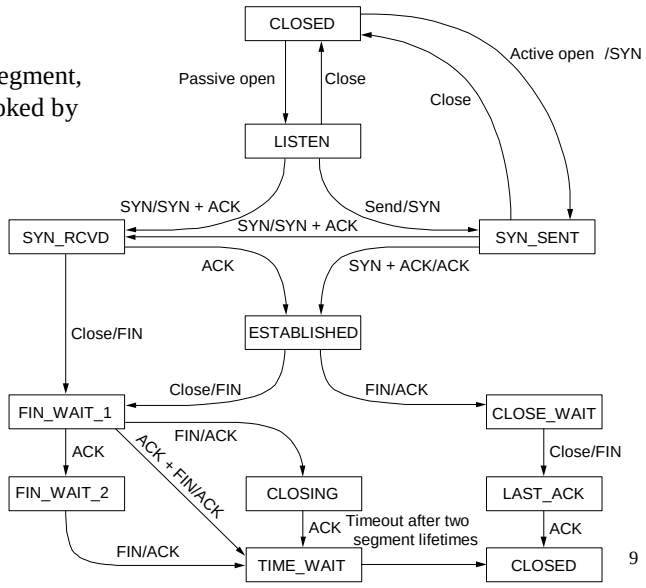


8

State Transition Diagram

event / action

event: receiving a segment, or an operation invoked by application



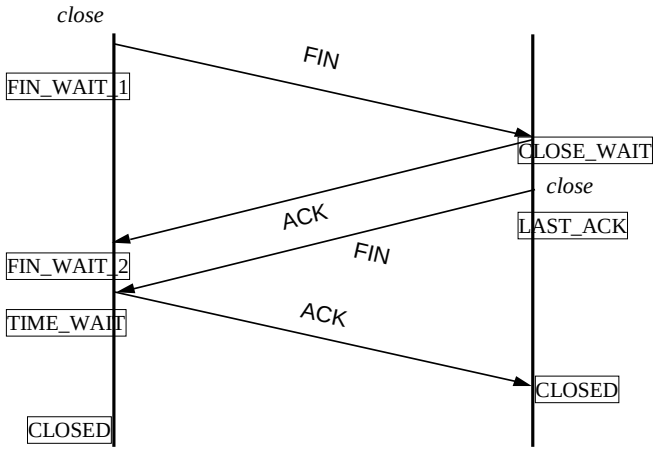
9

State Transition Diagram (cont)

- Data transfer occur in the ESTABLISHED state
- Open a connection
  - Server listens and waits for SYN.
  - If the client's ACK to the server is lost, connection is still established, due to cumulative ACKs
- Terminate a connection
  - Both sides can terminate
    - Case 1: one side closes first
    - Case 2: both sides close at the same time
  - TIME_WAIT to CLOSED: wait for 120 seconds
    - The other side might retransmit FIN while waiting for ACK
    - The next TCP connection might reuse the same port.

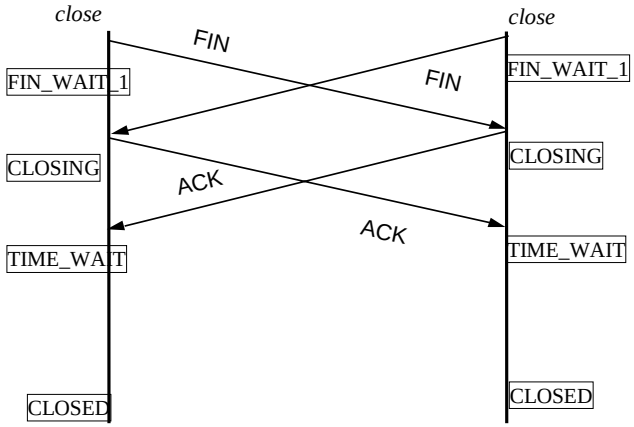
10

Connection Termination – One Side Closes First



11

Connection Termination – Both Sides Close



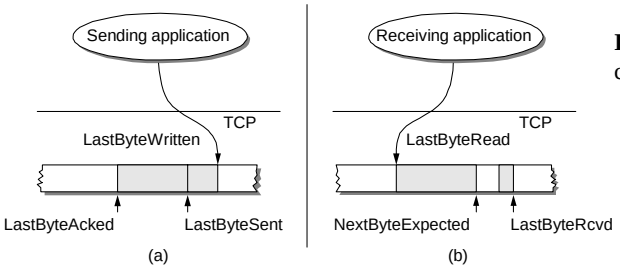
12

Sending Buffer and Receiving Buffer

- The receiver's buffer has two purposes
  - Reorder segments received out of order
  - Hold data unread by the application
- The receiver sends *AdvertisedWindow* in ACK
- The sender cannot send more than *AdvertisedWindow* bytes of unacknowledged data at any given time (Flow Control).
- The receiver selects a suitable *AdvertisedWindow* based on the available memory and application reading speed.

13

Sliding Window Revisited



Implementation: circular buffers

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

14

Flow Control

- **MaxSendBuffer** and **MaxRcvBuffer**
- Receiving side
  - **LastByteRcvd** - **LastByteRead** < = **MaxRcvBuffer**
  - **AdvertisedWindow** = **MaxRcvBuffer – ((NextByteExpected – 1) - LastByteRead)**
- Sending side
  - **LastByteWritten** - **LastByteAcked** < = **MaxSendBuffer**
    - block sender if (**LastByteWritten** - **LastByteAcked**) + *y* > **MaxSenderBuffer**
  - **LastByteSent** - **LastByteAcked** < = **AdvertisedWindow**
  - **EffectiveWindow** = **AdvertisedWindow** - (**LastByteSent** - **LastByteAcked**) (how much more original data can be sent)
- Always send ACK in response to arriving data segment
- Persist when **AdvertisedWindow = 0**
  - Sender sends 1 byte of data every so often.

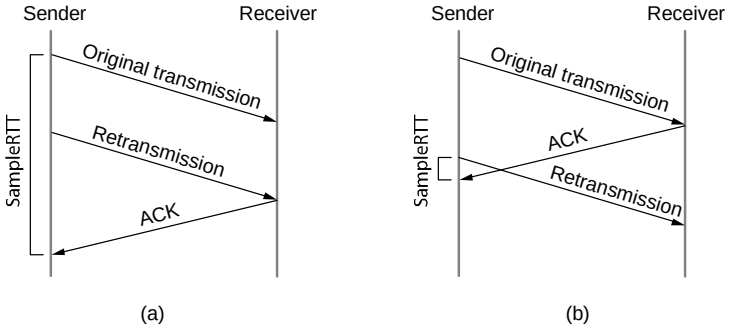
15

Adaptive Retransmission (Original Algorithm)

- Measure **SampleRTT** for each segment / ACK pair
- Compute weighted average of RTT
  - **EstRTT** = α x **EstRTT** + β x **SampleRTT**
  - where $\alpha + \beta$ **= 1**, **0.8** $\leq$ **a** $\leq$ **0.9**, **0.1** $\leq$ **b** $\leq$ **0.2**
  - Smooth noisy measurements
- Set timeout based on **EstRTT**
  - **TimeOut** = **2** x **EstRTT**
  - 2: to be conservative

16

Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission
  - When the retransmitted segment is ACKed, timeout value is reduced to **2** x **EstRTT**

17

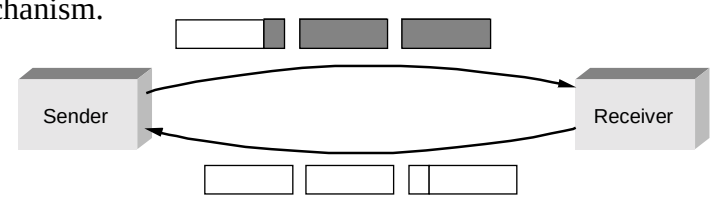
Jacobson/ Karels Algorithm

- Takes the variances of sampled RTT into account
  - if the var is small, no need to multiply **EstRTT** by 2.
- **Diff** = **SampleRTT** - **EstRTT**
- **EstRTT = EstRTT + (δ** x **Diff)**
  = (1 - δ) x **EstRTT + δ** x **SampleRTT**
- **Dev = Dev + δ(|Diff| - Dev)**
  = (1 - δ) x **Dev + δ|Diff|**
  - where δ is a factor between 0 and 1
- **TimeOut** = μ x **EstRTT** + ϕ x **Dev**
  - where $\mu = 1$ and $\phi = 4$
- Notes
  - algorithm only as good as granularity of timer (500ms on Unix, 100ms on Linux)
  - accurate timeout mechanism important to congestion control (later)

18

Silly Window Syndrome

- MSS (Max Segment Size) is set to (local MTU – TCP/IP header)
- The TCP sender may sends tiny segment into networks
  - if the effective window is less than MSS
  - if the application generates data one byte at a time
- Inefficient use of bandwidth : 4000% overhead of TCP/IP header
- Not aggregates afterwards due to the ACK self-clocking mechanism.



19

Nagle's Algorithm

- How long does sender delay sending data?
  - too short: poor network utilization
  - too long: hurts interactive applications
  - how long? utilize ACK self-clocking to simulate a timer
- If there is unACKed data in transit: buffer it until ACK arrives; else send it

20

Message Boundaries

- UDP socket API is message-oriented (datagram sockets)
  - Individual datagrams (sent with separate calls) will be kept separate when they are received. A revcfrom() call on a datagram socket will only return the next datagram.
  - Applications picks the segment size.
    - Could be segmented by IP.
- TCP socket API is byte-oriented (stream sockets)
  - Message boundaries addressed by the application layer protocol.

21

Problem: Keeping the Pipe Full

- 16-bit **`AdvertisedWindow`** allows 64KB

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

22

Problem: Protection Against Wrap Around

- 32-bit **`SequenceNum`**
  - 16-bit **`AdvertisedWindow`**: $2^{32} >> 2 \cdot 2^{16}$
- Another byte with the same sequence number x could be sent once again, if window size is large enough (e.g 1GB)

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

23

TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
  - for fine-grained RTT measurements
- Extend sequence space with 32-bit timestamp (PAWS)
  - for packet differentiation
  - not for reordering or acknowledging
- Shift (scale) advertised window

24