Transport Layer

Outline

Reliable Byte-Stream (TCP) - continued

Sliding Window Revisited

Flow Control

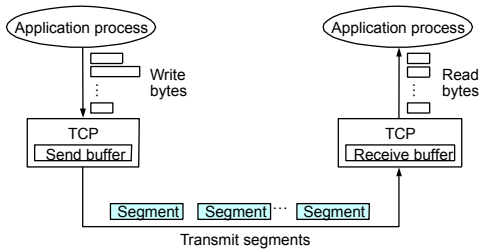
Adaptive Timeout

Housekeeping

- Turn in homework
- UDP Length field
  - Still a mystery
  - One speculation: it's there to make the header length a multiple of 4
  - "Jumbo packet" extension requires setting it to 0 and calculating the UDP data size based on the IP packet length

TCP Overview

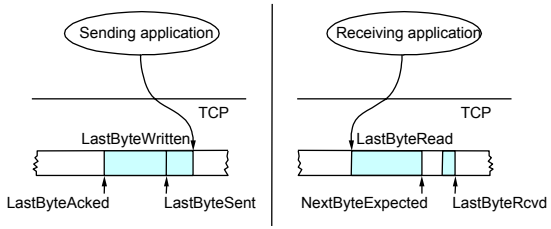
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Data Link Versus Transport

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

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 **`NextByteRead`** and **`LastByteRcvd`**

Flow Control

- Send buffer size: **`MaxSendBuffer`**
- Receive buffer size: **`MaxRcvBuffer`**
- Receiving side
  - **`LastByteRcvd`** - **`LastByteRead`** < = **`MaxRcvBuffer`**
  - **`AdvertisedWindow`** = **`MaxRcvBuffer`** - (**`NextByteExpected`** - **`NextByteRead`**)
- 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
- Sender persists 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

Keeping the Pipe Full

- Limits of 16-bit **`AdvertisedWindow: 100ms RTT`**

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

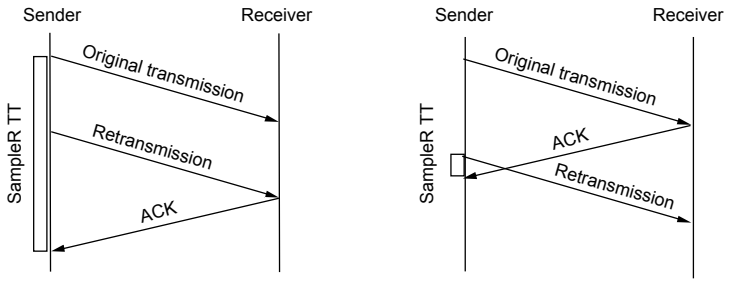
TCP Extensions – RFC 1323

- 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
  - **EstRTT** = α x **EstRTT** + β x **SampleRTT**
  - where $\alpha + \beta = 1$
  - α between 0.8 and 0.9
  - β between 0.1 and 0.2
- Set timeout based on **EstRTT**
  - **TimeOut** = **2** x **EstRTT**

Karn/Partridge Algorithm



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

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- **Diff** = **SampleRTT** - **EstRTT**
- **EstRTT = EstRTT + (δ x Diff)**
- **Dev = Dev + δ(|Diff| - Dev)**
  - where δ is a factor between 0 and 1
- Consider variance when setting timeout value
- **TimeOut** = μ x **EstRTT** + ϕ x **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)