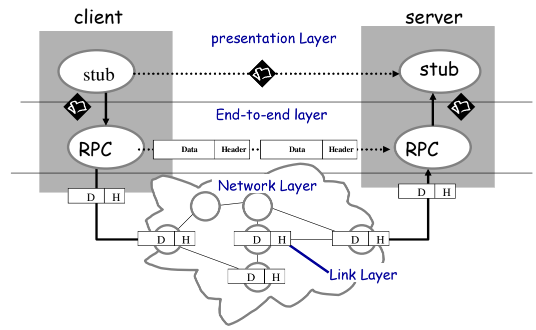
L12: end to end layer

6.033 Spring 2007

http://web.mit.edu/6.033

Slides from many folks

End-to-end layer



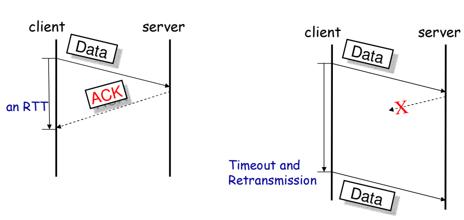
Network layer provides best effort service

- Packets may be:
  - Lossed
  - Delayed (jitter)
  - Duplicated
  - Reordered
  - …
- Problem: Inconvenient service for applications
- Solution: Design protocols for E2E modules
  - Many protocols/modules possible, depending on requirements

This lecture: some E2E properties

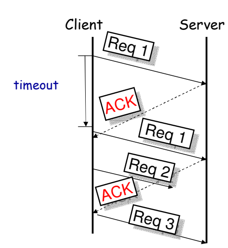
- At most once
- At least once
  - Exactly once?
- Sliding window
- Case study: TCP
- Tomorrow: Network File System (NFS)

At Least Once



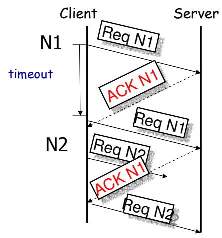
- Sender persistently sends until it receives an ack
- Challenges:
  - Duplicate ACKs
  - What value for timer

Duplicate ACK problem



- Problem: Request 2 is not delivered
  - violates at-least once delivery

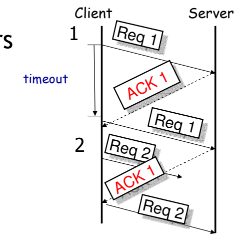
Solution: nonce



- Label request and ack with unique identifier that is never re-used

Engineering a nonce

- Use sequence numbers
- Challenges:
  - Wrap around?
  - Failures?

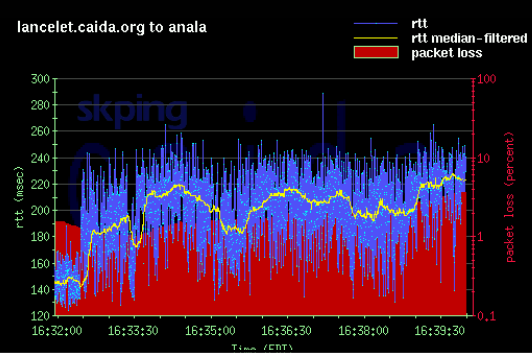


Timer value

- Fixed is bad. RTT changes depending on congestion
  - Pick a value that's too big, wait too long to retransmit a packet
  - Pick a value too small, generates a duplicate (retransmitted packet).
- Adapt the estimate of RTT à adaptive timeout

RTT Measurements

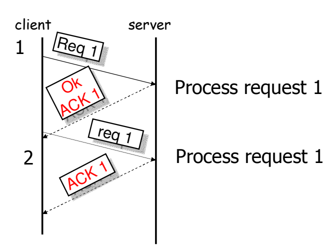
(collected by Caida)



Adaptive Timeout: Exponential weighted moving averages

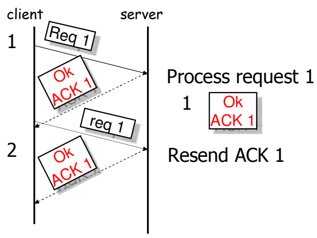
- Samples S_1, S_2, S_3, ..
- Algorithm
  - EstimatedRTT = T_0
  - EstimatedRTT = α S + (1- α) EstimatedRTT
  - where $0 \le \alpha \le 1$
- What values should one pick for α and T_0?
  - Adaptive timeout is also hard

At Most Once Challenges



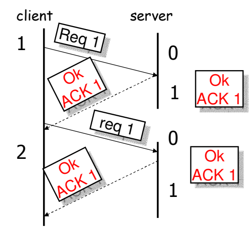
- Server shouldn't process req 1
- Server should send result preferably

Idea: remember sequence number



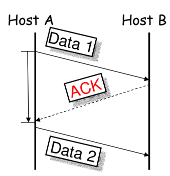
- Server remembers also last few responses

Problem: failures



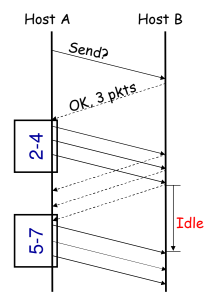
- Performed request 1 twice!
- How to maintain the last nonce per sender (tombstone)?
  - Write to non-volatile storage?
  - Move the problem? (e.g., different port number)
  - Make probability of mistake small?
- How about exactly once? (Need transactions)

How fast should the sender sends?



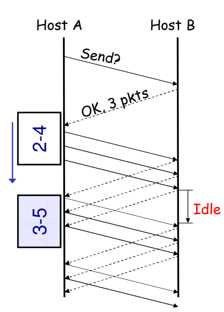
- Waiting for acks is too slow
- Throughput is one packet/RTT
  - Say packet is 500 bytes
  - RTT 100ms
  - à Throughput = 40Kb/s, Awful!
- Overlap pkt transmission

Send a window of packets



- Assume the receiver is the bottleneck
  - Maybe because the receiver is a slow machine
- Receiver needs to tell the sender when and how much it can send
- The window advances once all previous packets are acked à too slow

Sliding Window

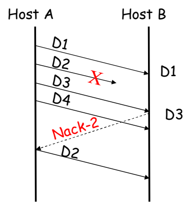


- Senders advances the window whenever it receives an ack à sliding window
- But what is the right value for the window?

The Right Window Size

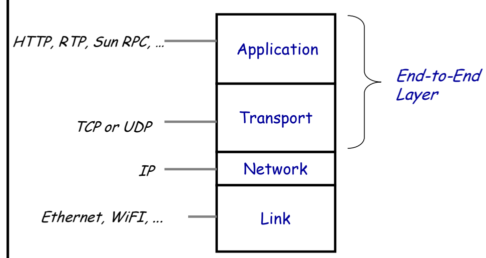
- Assume server is bottleneck
  - Goal: make idle time on server zero
  - Assume: server rate is B bytes/s
  - Window size = B x RTT
  - Danger: sequence number wrap around
- What if network is bottleneck?
  - Many senders?
  - Sharing?
  - Next lecture

“Negative” ACK



- Minimize reliance on timer
- Add sequence numbers to packets
- Send a Nack when the receiver finds a hole in the sequence numbers
- Difficulties
  - Reordering
  - Cannot eliminate acks, because we need to ack the last packet

E2E layer in Internet



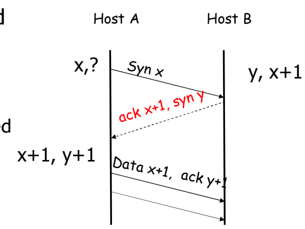
The 4-layer Internet model

UDP

	0–15	16–31
udp_hdr	Source Port udp_sport	Destination Port udp_dport
	Length udp_ulen	Checksum udp_sum
udp_data		

Transmission Control Protocol (TCP)

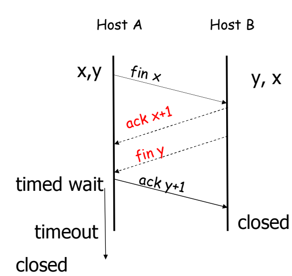
- Connection-oriented
- Delivers bytes at-most-once
- Bidirectional
  - ACKs are piggybacked

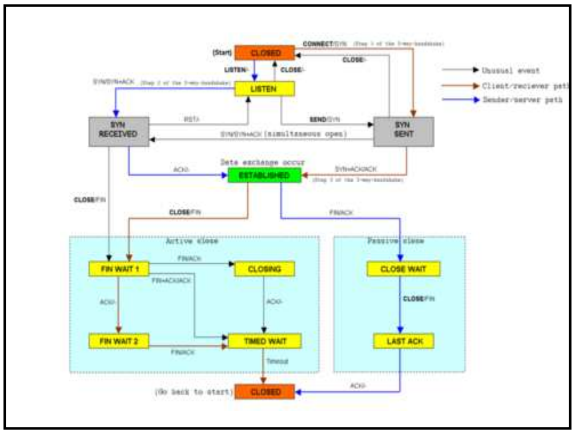


TCP header

	0–15			16–31
tcp_hdr	Source Port tcp_sport			Destination Port tcp_dport
	Sequence Number tcp_seq			
	Acknowledgment Number tcp_ack			
	Offset tcp_off	Reserved —	Flags (below)	Window tcp_win
	Checksum tcp_sum			Urgent Pointer tcp_urp
tcp_options	TCP options …			
tcp_data				

Closing a TCP connection




(Start)
CLOSED
CONNECT/SYN
LISTEN/-
CLOSE/-
CLOSE/-
LISTEN
Unusual event
Client/reciever path
Sender/server path
SYN/SYN+ACK
RST/-
SEND/SYN
SYN RECEIVED
SYN SENT
SYN/SYN+ACK (simultaneous open)
Data exchange occur
ACK/-
ESTABLISHED
SYN+ACK/ACK
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active close
Passive close
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
ACK/-
CLOSE/FIN
FIN WAIT 2
TIMED WAIT
LAST ACK
FIN/ACK
Timeout
(Go back to start)
CLOSED
ACK/-