TCP Strategies

Keepalive Timer

- Yet another timer in TCP is the *keepalive*
- This one is not required, and some implementations do not have it as it is occasionally regarded as controversial
- When a TCP connection is idle no data flows between source and destination
- So part of the path could break and be restored and the connection is none the wiser

TCP Strategies

Keepalive Timer

- On the other hand, sometimes the server wants to know if the client is still alive: perhaps each client uses some resources in the server
- If the client has crashed these resources could better be used elsewhere
- To do this the server sets the keepalive timer when the connection goes idle
- Typical values are 2 hours

TCP Strategies

Keepalive Timer

- When the timer expires, the server sends a *keepalive probe*
- This is simply an empty segment
- If the server gets an ACK, everything is OK
- If not, the server might conclude the client is no longer listening

TCP Strategies

Keepalive Timer

- There are four cases
  1. the client is up and running: the keepalive probe is ACKed and everybody is happy. The keepalive timer is reset to 2 hours
  2. the client has crashed or is otherwise not responding to TCP: the server gets no ACK and resends after 75 seconds. After 10 probes, 75 seconds apart, if there is no response, the server terminates the connection with “connection timed out”

TCP Strategies

Keepalive Timer

- There are four cases

3. the client has crashed and rebooted. The client gets the probe and responds with a RST. The server gets the RST and terminates the connection with “connection reset by peer”

4. the client is up and running, but is unreachable, e.g., a broken router. This is indistinguishable from case 2, so the same events ensue

TCP Strategies

Keepalive Timer

- If a host is shut down normally, rather than crashing, the operating system will send FINs for all currently open connections. This allows the server to close the connections

TCP Strategies

Keepalive Timer

- There are several reasons not to use keepalive
  - they can cause a good connection to be closed because of an intermittent failure of a router
  - they use bandwidth
  - some network operators charge per packet

TCP Strategies

Keepalive Timer

- The latter two are not particularly good arguments as the cost is just a couple of packets every 2 hours
- It is usually possible to disable keepalive in the application: some people think that keepalive should not be in the TCP layer, but should be handled by the application layer

TCP Strategies

Path MTU Discovery

- The *path MTU* is the size of the largest segment that can be sent from source to destination without being fragmented
- Actually the smallest MTU of *any* path, as a segment might take one of several routes
- Sending packets of size not larger than the path MTU will not be fragmented: this is good as the costs of fragmentation are high

TCP Strategies

Path MTU Discovery

- Path MTU discovery tries to find the path MTU by adjusting the sizes of the packets being sent
- Initially, send packets of MSS the minimum of the MTU of the interface and the MSS announced by the other end (or 536 if the other end does not give an MSS)
- Packets of this size will not be fragmented by the source or destination

TCP Strategies

Path MTU Discovery

- Subsequent segments are sent with the IP flag DF (Don't Fragment) set
- If a router need to fragment, it drops the segment and returns and ICMP “fragmentation needed by DF set”
- The source can then retry with a smaller MSS
- This is very much like the approach of IPv6

TCP Strategies

Path MTU Discovery

- Modern versions of ICMP return the *next hop MTU* in the ICMP error and the source can use that value directly
- Older versions do not and the source simply picks a value from a table

TCP Strategies

65535	Hyperchannel, maximum MTU	RFC1044, RFC791
32000	just in case	
17914	16Mb/sec Token Ring	
8166	Token Passing Bus 802.4	RFC1042
4464	4Mb/sec Token Ring (maximum MTU)	
4352	FDDI	RFC1188
2048	Wideband Network	RFC907
2002	4Mb/sec Token Ring (recommended MTU)	
1500	Ethernet	RFC894
1492	IEEE 802.3	RFC1042
1006	SLIP	RFC1055
576	X.25	RFC877
512	NETBIOS	RFC1088
296	point-to-point (low delay)	RFC1144
68	minimum MTU	RFC791

TCP Strategies

Path MTU Discovery

- If such an ICMP happens in a TCP connection, the congestion window should remain unchanged, but a slow start should begin
- It is recommended you try larger MTU once in a while (e.g., 10 minutes) as routes can vary dynamically

TCP Strategies

Path MTU Discovery

- Some poorly configured routers never return ICMP messages, allegedly for security
- Such routers fail with path MTU discovery if they have a small MTU: packets are dropped and no ICMP error is returned
- Turning off DF and allowing fragmentation will let the packet get through the router

TCP Strategies

Path MTU Discovery

- So sometimes you can only get a connection if you turn off path MTU discovery!
- This is a poor state of affairs as the administrators of the routers should know better

TCP Strategies

Long Fat Pipes

- Wide area networks have quite different problems to LANs
- Bandwidth is not the limiting factor, but the *speed of light*: the time the bits take to get to the destination now dominates

TCP Strategies

Long Fat Pipes

- The *bandwidth-delay product* is a measure of the capacity of a network:

capacity in bits =

bandwidth in bits/sec x round trip time in seconds

TCP Strategies

Long Fat Pipes

- Suppose we have an ATM connection across the Atlantic, running at 155Mb/s with a round trip of 144ms
- The bandwidth-delay product is roughly 22Mb (2.8MB)
- This is total number of bytes that can be in flight in the connection

TCP Strategies

Long Fat Pipes

- An Ethernet running at a reasonable 5Mb/s and 500µs RTT has a capacity of 312 bytes
- A gigabit network over a satellite network (delay 0.5s each way) has a capacity of 125MB
- A network with a large bandwidth-delay product is called a *long fat network* (LFN, or “elephant”)
- A TCP connection over a LFN is a *long fat pipe*

TCP Strategies

Long Fat Pipes

- These have many problems
  - To get efficiency you must have a huge advertised window, much bigger than the 65535 that TCP initially gives us
  - The *window scale* option exists to address this
  - This is because we can't afford to wait for ACKs with such a large RTT

TCP Strategies

Long Fat Pipes

- These have many problems
  - Packet loss is disastrous as the subsequent reduction in congestion window size hits us badly

TCP Strategies

Long Fat Pipes

- These have many problems
  - TCP counts bytes using a 32 bit counter: this wraps around after 4GB
  - In a 10GB network we can easily transmit this in under 10 seconds and a packet could feasibly be lost for this amount of time and reappear to clash with a later packet

TCP Strategies

Long Fat Pipes

- These have many problems
  - *Protection against wrapped sequence numbers* (PAWS) addresses this

TCP Strategies

Long Fat Pipes

- These have many problems
  - The biggest problem is latency: the time the packet takes to get to its destination

TCP Strategies

Long Fat Pipes: Latency

- To send a megabyte across the Atlantic takes about 76ms on a 155Mb/s network
  - the first bit arrives after about 70ms
  - the last bit arrives 6ms later

TCP Strategies

Long Fat Pipes: Latency

- To send a megabyte across the Atlantic takes about 70.6ms on a 1550Mb/s network
  - the first bit *still* arrives after about 70ms
  - the last bit arrives 0.6ms later
- A 10-fold increase in network speed only reduces the latency by 7%
- We can't get the first bit there any faster, it's the speed of light!

TCP Strategies

Long Fat Pipes: Latency

- Thus Long Fat Pipes are (not necessarily) bandwidth limited, but *latency* limited
- The problem is much worse with satellite links, with delays of 0.5 sec

TCP Strategies

Timestamps

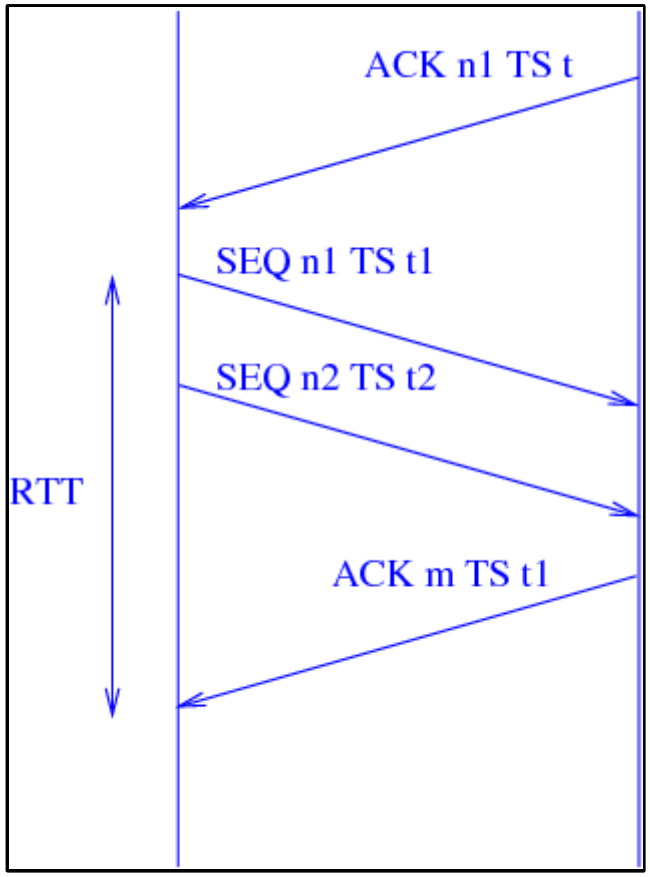
- The TCP header can contain an optional *timestamp*
- This is a 32-bit value, not necessarily a time, that should be echoed back to the sender in the ACK
- Use of timestamps is negotiated in the SYN: if both ends have a timestamp in their SYNs then timestamps can be used in this connection

TCP Strategies

Timestamps

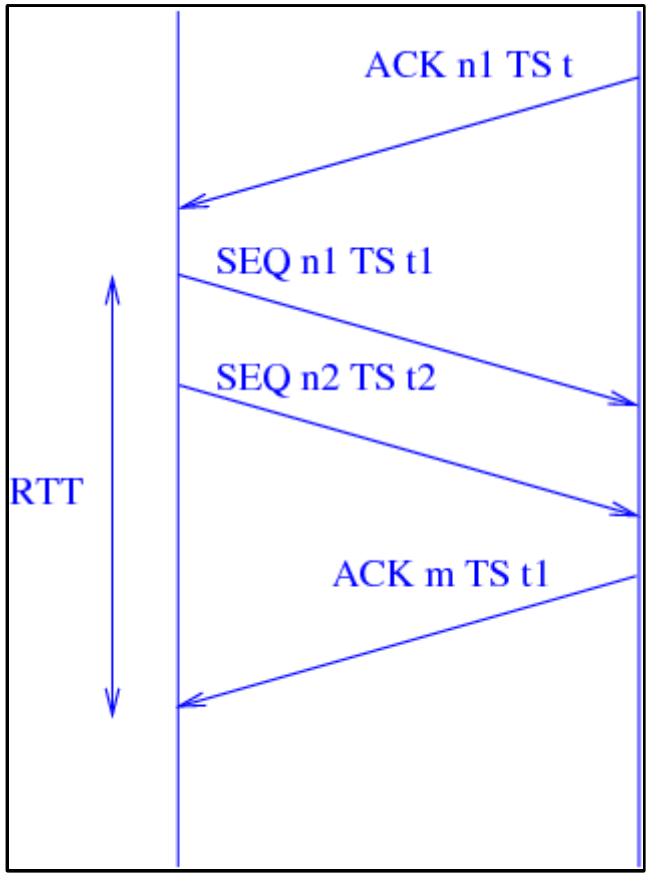
- This was originally used as a mechanism to get accurate RTTs and so better estimates for retransmission timeouts
- But a host can ACK several segments in a single reply: which timestamp should be used?

TCP Strategies



- The one for the next expected segment

TCP Strategies



- So if two segments are sent and one ACK returned, use the timestamp from the *first* segment

TCP Strategies

Timestamps

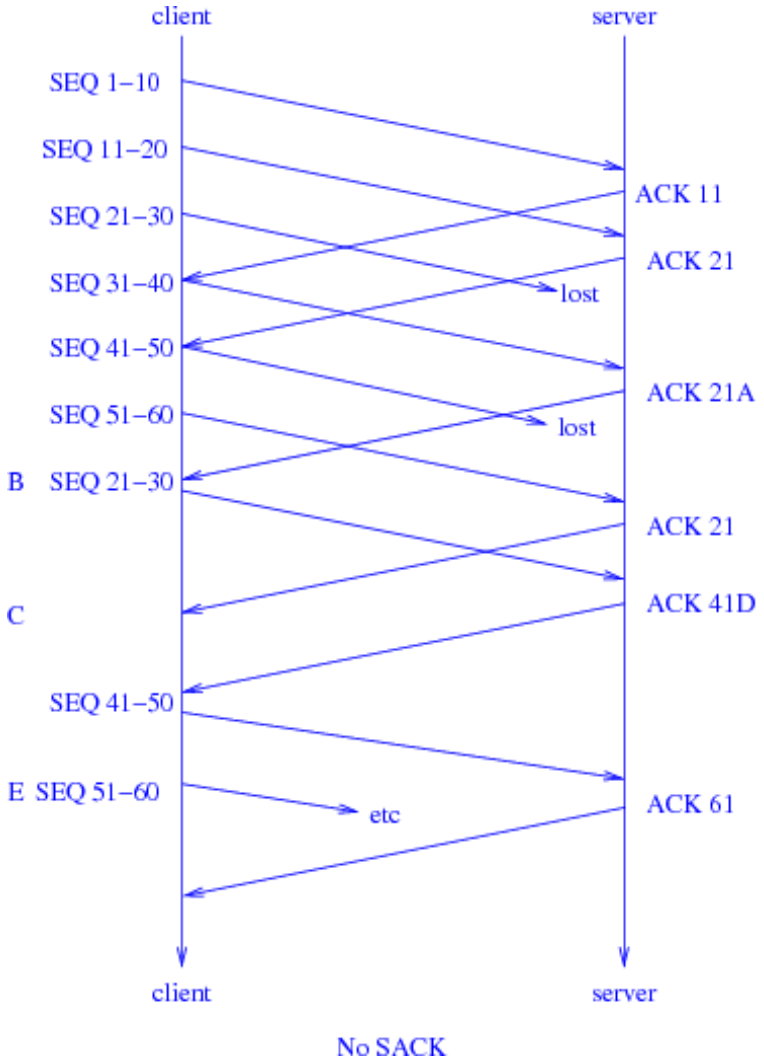
- Timestamps are also used by PAWS

TCP Strategies

SACK

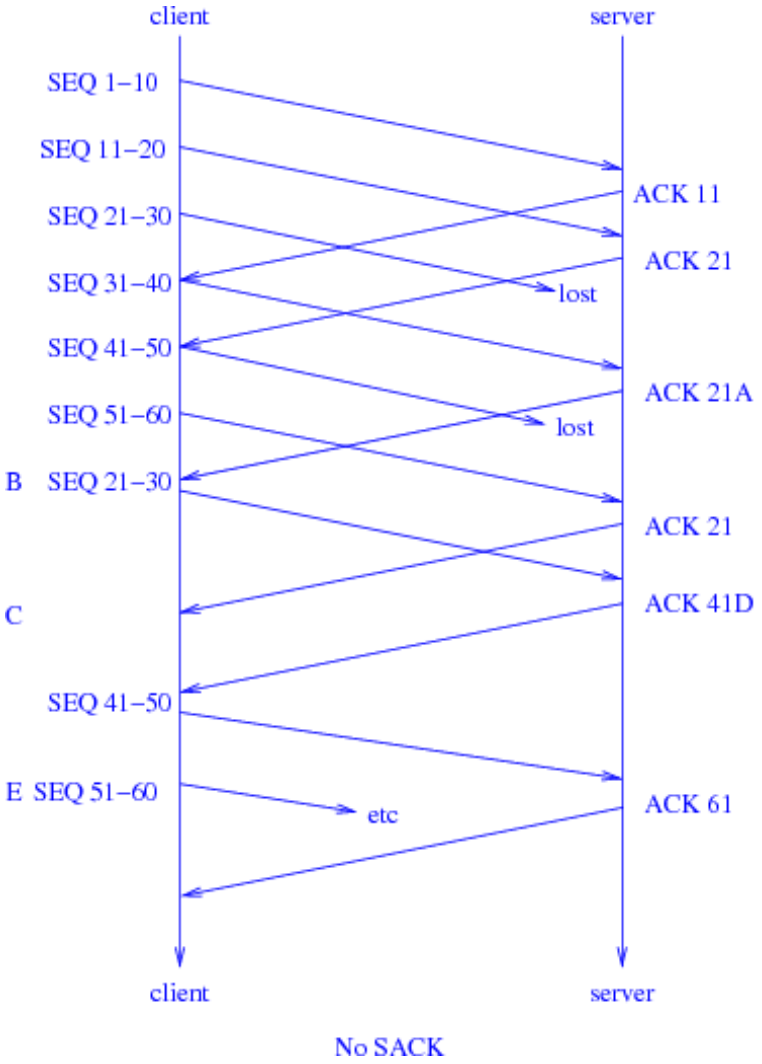
- *Selective Acknowledgement* (SACK) is an optional mechanism for finer control of acknowledgements than simple ACKs provide
- It extends the ACK by sending a list of blocks that have been safely received

TCP Strategies



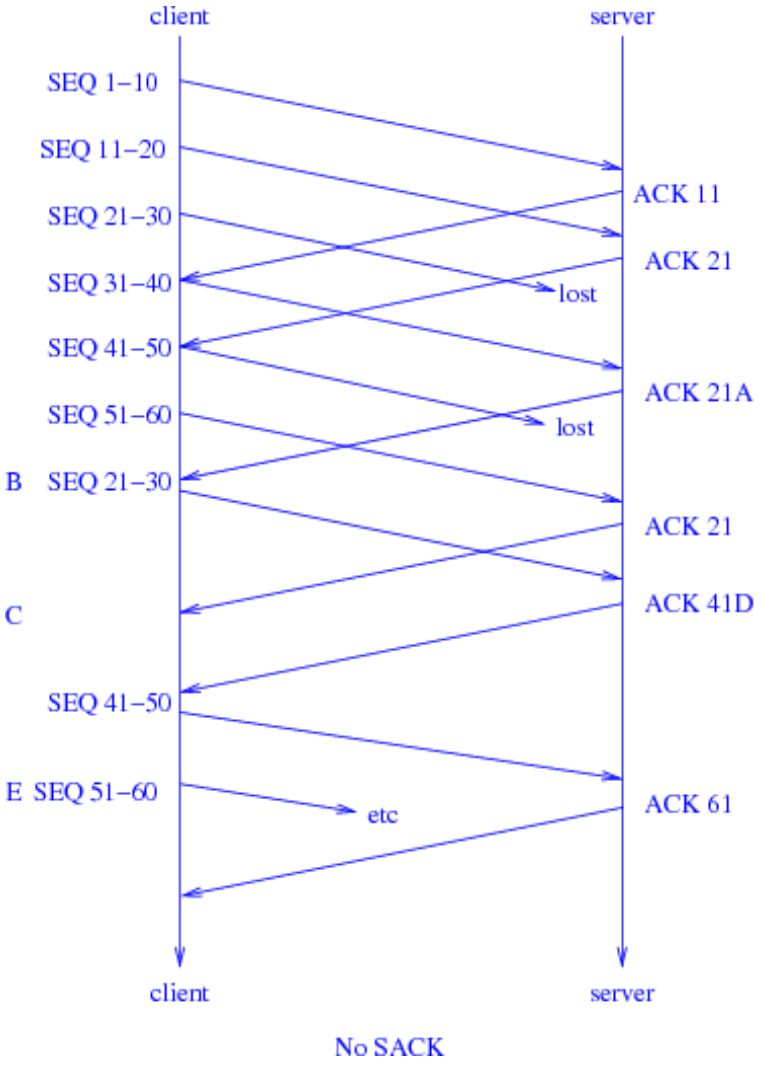
- A client is sending to a server, but a couple of segments are lost
- A point A the server gets an unexpected sequence number, so it sends an immediate duplicate ACK

TCP Strategies



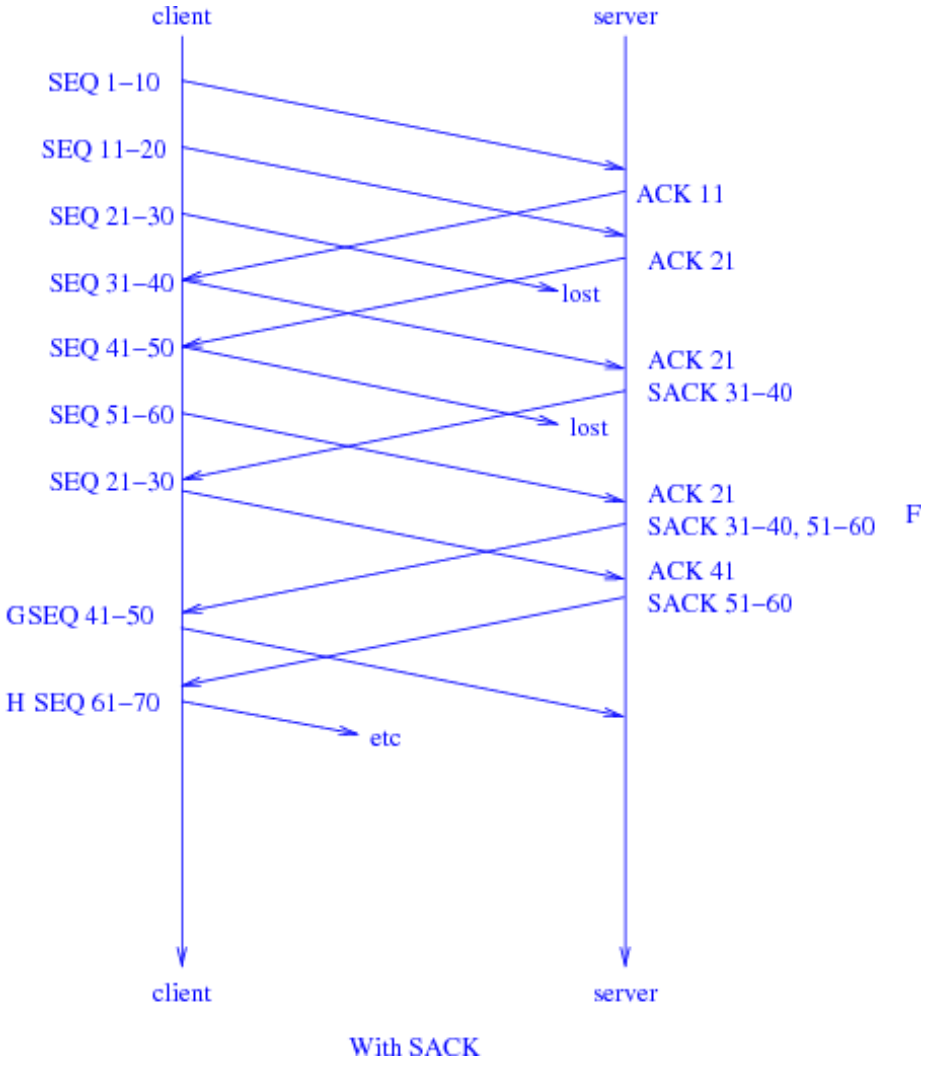
- The client eventually gets this (after sending more data), resends, and slows down
- Fast retransmit and recovers happens at C, the third duplicate ACK

TCP Strategies



- The data arrives safely at D, and is ACKed
- Server can ACK up to byte 41
- Cannot ACK that it also has bytes 51-60 and client doesn't need to resend them at E

TCP Strategies



With SACK

- SACK allows this: ACK as usual but also send SACK at F
- At G, client sees that 51-60 are OK, and can continue from 61 at H

TCP Strategies

SACK

- So fewer resends, and faster recovery
- Whether SACK can be used is negotiated in the SYN, using the *SACK permitted* optional header

TCP Strategies

Theoretical Throughput

- Standard Ethernet is rated at 10Mb/s
- Taking physical, IP and TCP headers, minimum inter-packet gap, ACK packets and so on into account, the theoretical maximum throughput of TCP on 10Mb Ethernet is calculated as 1,183,667 bytes/sec
- Implementations have been measured at 1,075,000 bytes/sec: very close!

TCP Strategies

Theoretical Throughput

- Faster networks are correspondingly better, but the limitations are:
  - the speed of light
  - the TCP window size
- We can tweak the latter, but not the former!
- If your network is slower than the above, it's the hardware or the implementation's fault, not TCP

TCP Strategies

Theoretical Throughput

- TCP is a huge success: from 1200 bits/sec telephone lines to gigabit networks and beyond it has turned out to be massively flexible and scalable
- It took a lot of work, though!