Round Trip Time Estimation

- Useful to know when to retransmit
  - if have not received ACK within the RTT
    - (plus a bit)
    - then assume packet lost
- But how to measure the RTT?
  - Measure delay between packet and its ACK
    - easy
  - But
    - Send packet
      - wait ... wait ... wait (nothing)
    - Retransmit packet
    - ACK arrives
  - Which packet was acknowledged?
    - The initial packet
      - acknowledged slower than expected
    - Or the retransmit?

TCP RTT Measurement

TCP RTT Measurement (2)
TCP Timestamp Option

- Each TCP can add timestamp option to every packet
- Peer TCP sends back timestamp received with each ACK
- Allows TCP to determine which packet was ACK'd
- Better than that
  - no need to remember when packets were sent
  - returning timestamp contains that information
- Also used for long delayed old packet detection
  - extends the sequence number space

TCP RTT Measurement (tstamp)
TCP or UDP

- UDP is unreliable, not flow controlled
- TCP is reliable, has flow control
  - Often would prefer to use TCP to UDP
- But
  - Overheads are much greater

Typical TCP

![TCP Handshake](image)

- Start with 3 way handshake
- Server
- Client
- SYN
- One RTT
- Server
- Client
- SYN + ACK
- Server
- Client
- SYN + ACK
- ACK
  - DATA
  - Server delay before reply
  - SYN
  - Server
  - Client
  - SYN + ACK
  - ACK
    - DATA
  - ACK
    - DATA
  - ACK
    - FIN
    - ACK
  - Another RTT (but unimportant)
    - FIN
    - ACK
    - FIN
    - ACK
  - Done - 2*RTT + server delay (11 packets)

UDP Alternative

![UDP Alternative](image)

- Just send the data
- Server delay
  - Client
  - SYN
  - Server
  - Client
  - SYN + ACK
  - Server
  - Client
  - SYN + ACK
  - ACK
    - Data
    - Reply
    - Data
    - ACK
    - Data
    - ACK
    - FIN
    - ACK
    - FIN
    - ACK
- Done - 1*RTT + server delay (2 packets)

- 2 packets instead of 11
- 1 RTT instead of 2
Minimal TCP

Possible to merge in TCP?
- Can put data in SYN packet
- Can put FIN in SYN packet
- Server reply can carry FIN
- Server data reply can carry FIN
- Still 2*RTT + server delay (6 packets)

TCP Cannot Do

Server cannot return data with its SYN
- TCP prohibits passing any data in original SYN to application until 3-way handshake is complete
- Not known that SYN is new
- SYN+ACK acks only the SYN

Old Duplicate SYN

Consider a client sending a SYN which is lost in the net and retransmitted
- Transaction continues as normal
- "Lost" SYN arrives
- TCP sends normal SYN+ACK reply
- Client not expecting this RST
- What if data were already delivered?
TCP Can Sometimes Do

- TCP can sometimes do ACK (which ACKs data in SYN packet) and merge with DATA reply packet.
- TCP depends on server delay.
- TCP has a total of 5 packets in the transaction.

Maximum Transaction Rate

- Bounded by $2 \times$ RTT + server delay.
- But also limited by TIME WAIT state:
  - No more data on same connection for $2 \times$ MSL.
  - Client picks a different port number.
  - Different connection.
- If 1000 connections / second, and MSL == 2 mins (120 secs):
  - Then 240 * 1000 connections in TIME WAIT state.
  - Consumes lots of memory.
  - (If 40 bytes/connection, almost 10MB).
- Worse! Impossible, only 65536 ports!
  - Thus limited to about 270 trans/sec.

T/TCP Packet exchange

- T/TCP is a minimal T/TCP 3 packet transaction.
- Server Delay:
  - 4 packets added to SYN.
  - Old segment protection.
  - A new CC is added in all packets.
  - SYN has its own CC.
  - ACK also carries echo of received CC from SYN.
T/TCP transactions

Server Client

SYN + Data + FIN
CC=4

SYN + ACK + Data + FIN
ACK CC=4

SYN + Data + FIN
CC=5

SYN + ACK + Data + FIN
ACK CC=5

T/TCP Server choices

- Server can perform 3 way handshake whenever it needs to
  - Received lower CC than expected
  - Didn’t know what CC to expect
  - ...
  - (And if doesn’t implement T/TCP)

T/TCP Client choices

- Client needs to be able to force 3 way handshake
  - Could just omit CC option
    - But that tells server that T/TCP is not supported
  - Could deliberately send lower CC
    - Hard to know what is low enough, not too low
  - Could just not send data in SYN packet
    - That could mean slightly increased delays
Also need a way to resynchronise server with client’s CC

New option CCnew added

* Replaces CC option in initial SYN packet (only)
* Forces 3-way handshake before connection established
  » (before data handed to server application)
* Informs server of clients CC value
  » once 3 way handshake is complete

T/TCP Client Lost State

If client loses state and sends a lower CC number to server, 3-way handshake results.

For every new connection until CC is bigger than server’s memory

Server

Client

T/TCP CCnew

Client sends a new option whenever it has no state for server

Server always does 3 way handshake

Verifies CCnew is valid

Sets memory to CCnew value

Initial SYN packet uses CCnew
T/TCP Large Requests

- Can still use T/TCP for large requests
- Nothing new from TCP here
- Note, none of initial packets from client to server carry ACK
- Cannot ACK anything until ISN received from server in its SYN-ACK packet

T/TCP Large Replies

- Can also handle large replies
- This is nothing different from TCP

T/TCP Large Request & Reply

- And of course, both
- Apart from CC options (not shown)
- Just the same as TCP
Isn't there a problem here?

How does the client know the MSS?

It is normally transmitted with the SYN.

The client info cached server info cached

Includes server’s CC PMTU to server

MSS from previous PMTU to cache

T/TCP Large Requests Revisited

Isn't there another problem here?

How about the window size sent in all packets but relates only to one connection the client has received no packets on this connection when it is sending data

WINDOW = 16K

Define window before known = 4K

T/TCP Large Requests Again