TCP Window Size Limitation

- At 100 M bits/second
  - (current network speeds)
  - sending 64K bytes
    - takes about 5.5 milliseconds
- So RTT must be less than 5.5 milliseconds
  - to avoid stall
- OK for LAN
- WAN might easily have RTT of 500ms
  - TCP can only use about 1% of available bandwidth
- Want to extend TCP
  - to allow higher throughput
  - Add an option

TCP Window Scale Option

- Allows TCP systems to
  - discover that both implement the option
  - specify that all window values
    - should be multiplied by a power of two
- Limited to 2^{14}
  - So, max window is (64K - 1) * 2^{14} ~= 1GB
  - Thus max of $^1/_4$ of the entire window space
- If both TCPs send wscale option
  - window scaling protocol is used
- Each TCP indicates
  - how much its window should be scaled
- If either does not send option
  - No window scaling
  - In either direction

TCP Option Usage

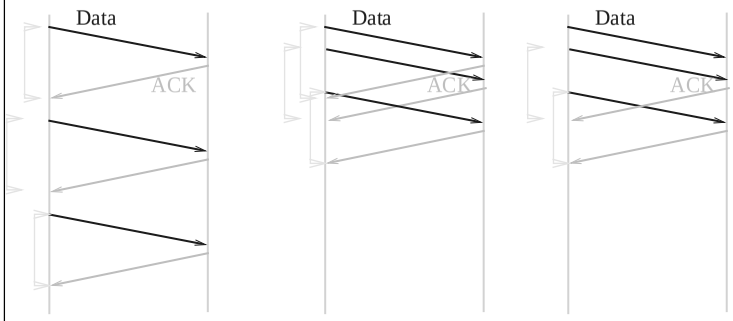
```
172.30.0.77.65486 > 172.30.0.161.9: S
  334775908:334775908(0) win 16384
  <mss 1460,nop,wscale 0,nop,nop,
                        timestamp 11188 0>
```

- WScale == 0
  - Scale window size by 2^0
    - 2^0 == 1
    - So, no scaling
- Why is option included?
  - So other host knows
    - this TCP supports the window scale option

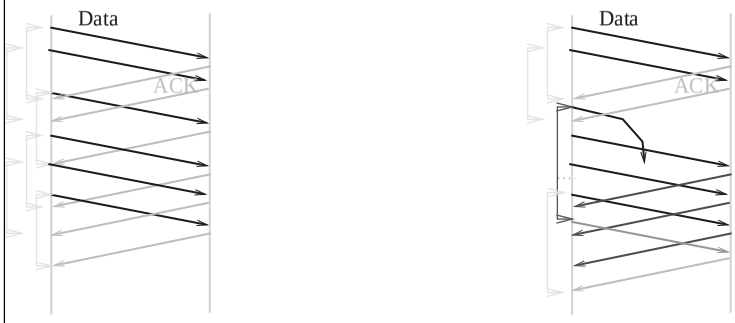
Round Trip Time Estimation

- 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

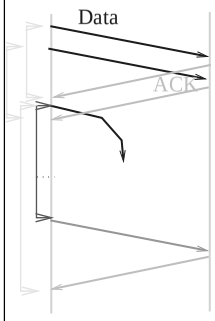
TCP RTT Measurement

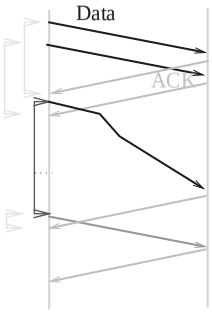


TCP RTT Measurement (2)



TCP RTT Measurement (3)

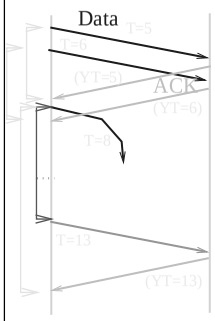


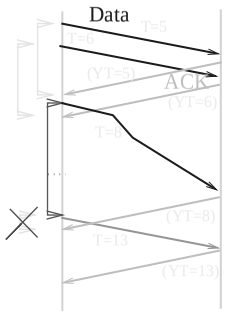


TCP Timestamp Option

- 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)

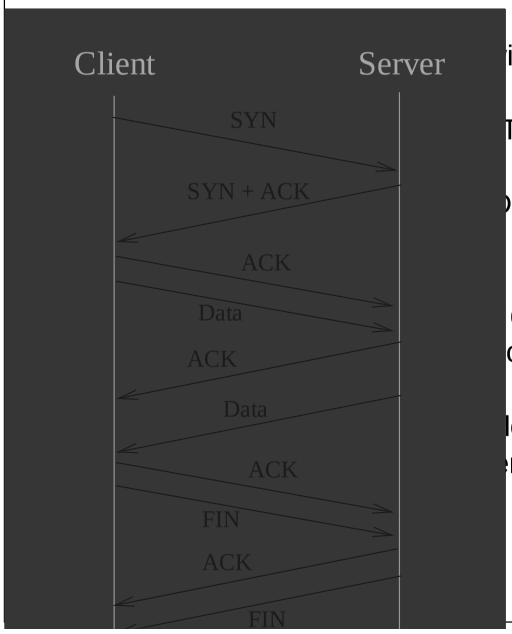




TCP or UDP

- 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



ith 3 way handshake

TT

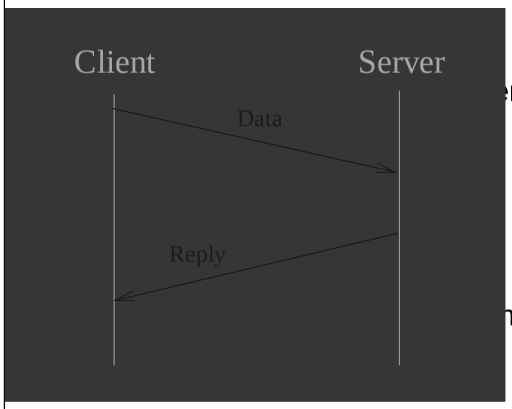
ATA

delay before reply
other RTT

lose connection
r RTT (but unimportant)

2*RTT + server delay
(11 packets)

UDP Alternative



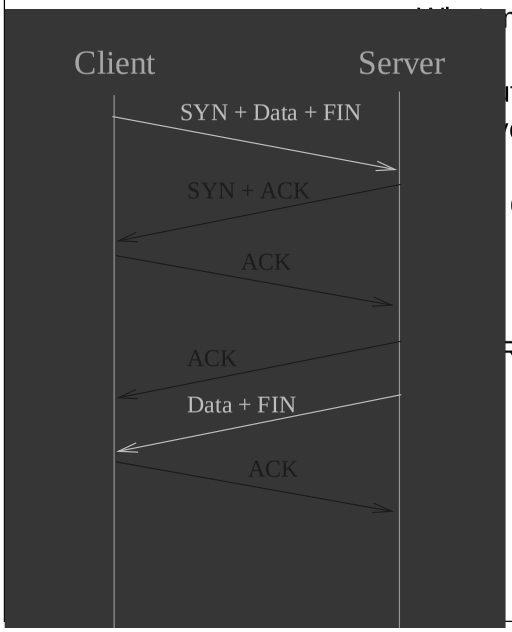
nd the data

Delay

ne RTT

Done - 1*RTT + server delay
(2 packets)

Minimal TCP



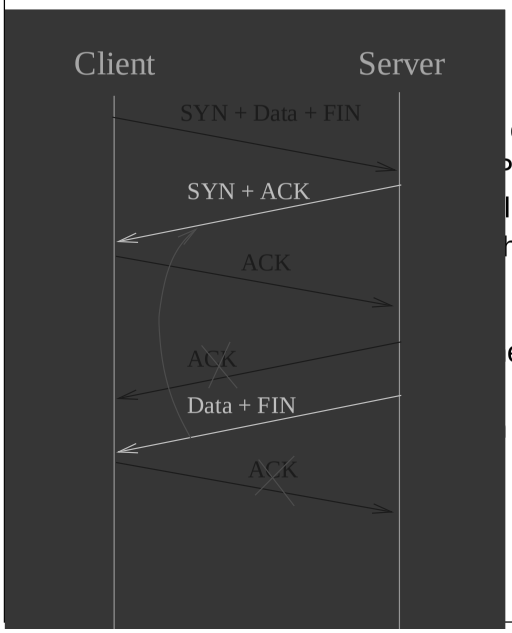
night be merged in TCP?

t data in SYN packet
ven put FIN in SYN packet

data reply can carry FIN

RTT + server delay
(6 packets)

TCP Cannot Do

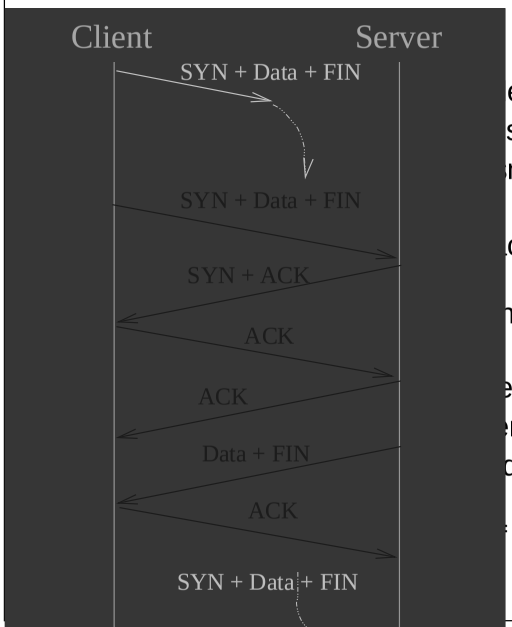


cannot return data with its SYN
P prohibits passing any data in
l SYN to application until
handshake is complete

en, not known that SYN is new

SYN+ACK acks only the SYN

Old Duplicate SYN



er a client sending a SYN
s lost in the net and
mitted

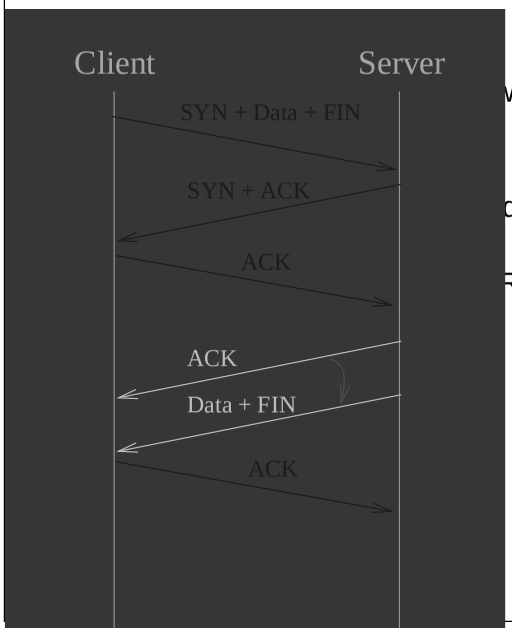
ction continues as normal

he "lost" SYN arrives

ends normal SYN+ACK reply
nt is not expecting this
ds a RST

data were already delivered?

TCP Can Sometimes Do



which ACKs data in SYN packet)
merged with DATA reply packet

ds on Server Delay

RTT + server delay
(5 packets)

Maximum Transaction Rate

- Bounded by 2 * RTT + server delay
- But also limited by TIME WAIT state
  - No more data on same connection for 2*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