Systems and Devices 2 (Network)
Lec 3b: Transport Layer

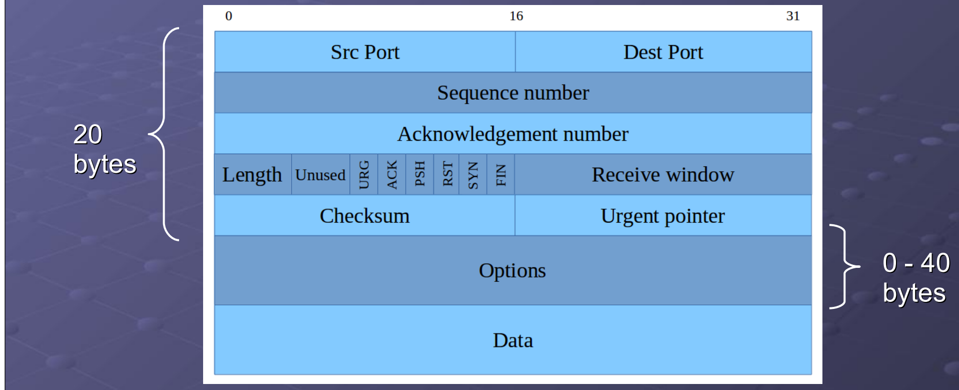
Before we get started ...

- Did you manage to teleport Bob (file transfer)?
  - Could you get my most excellent code working :)
- How did your research into TCP go?
  - Implementing a reliable communications protocol is tricky, need to consider all possible scenarios: lost / corrupt packets, packets received out of order, re-transmission of packets, fair usage of network bandwidth i.e. congestion control …
- Questions to consider :
  - How does TCP implement a reliable connection?
  - What are the differences between UDP and TCP?

TCP

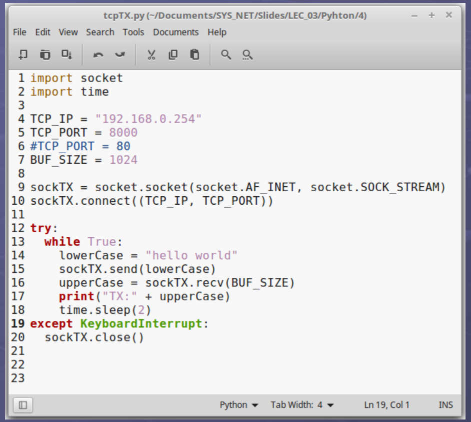
- Transmission Control Protocol
  - RFC 793 : https://tools.ietf.org/html/rfc793
  - Created in 1981 one of the core internet protocols.
  - TCP is a connection orientated protocol i.e. before an application process can send data it must first perform a handshake to ensure that a connection is possible
    - TCP is <u>not</u> an end-to-end protocol like circuit switching, it only existing in the transport layer of the communicating hosts, not in lower layers i.e. no reserved connections, “packets” can take different paths.
  - TCP uses full duplex connections. If there is a connection between process A and B, then we can send data A→B and B→A at the “same” time.
  - TCP implements a reliable, ordered, error-checked data stream between two hosts i.e. retransmission.

TCP



- TCP header, depending on option fields used can vary from 20-60B (UDP uses 8B).

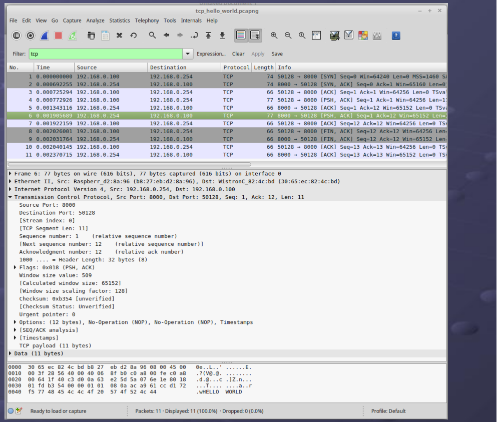
Demo



tcpTX.py (~/Documents/SYS_NET/Slides/LEC_03/Pyhton/4)

```python
import socket
import time

TCP_IP = "192.168.0.254"
TCP_PORT = 8000
#TCP_PORT = 80
BUF_SIZE = 1024

sockTX = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sockTX.connect((TCP_IP, TCP_PORT))

try:
  while True:
     lowerCase = "hello world"
     sockTX.send(lowerCase)
     upperCase = sockTX.recv(BUF_SIZE)
     print("TX:" + upperCase)
     time.sleep(2)
except KeyboardInterrupt:
  sockTX.close()
```

Python ▾ Tab Width: 4 ▾ Ln 19, Col 1 INS



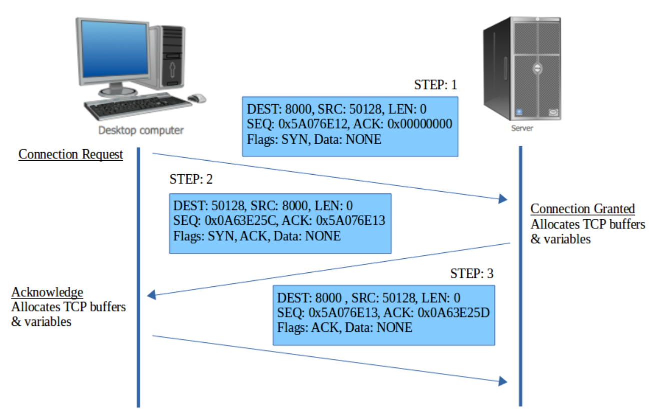
tcp_hello_world.pcapng

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply Save

```
No.  Time        Source         Destination    Protocol Length Info
   1 0.000000000 192.168.0.100  192.168.0.254  TCP        74 50128 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
   2 0.000692255 192.168.0.254  192.168.0.100  TCP        74 8000 → 50128 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
   3 0.000725294 192.168.0.100  192.168.0.254  TCP        66 50128 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
   4 0.000772926 192.168.0.100  192.168.0.254  TCP        77 50128 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=1
   5 0.001343116 192.168.0.254  192.168.0.100  TCP        66 8000 → 50128 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSva
   6 0.001905689 192.168.0.254  192.168.0.100  TCP        77 8000 → 50128 [PSH, ACK] Seq=1 Ack=12 Win=65152 Len=
   7 0.001922159 192.168.0.100  192.168.0.254  TCP        66 50128 → 8000 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TS
   8 0.002026001 192.168.0.100  192.168.0.254  TCP        66 50128 → 8000 [FIN, ACK] Seq=12 Ack=12 Win=64256 Len
   9 0.002031764 192.168.0.254  192.168.0.100  TCP        66 8000 → 50128 [FIN, ACK] Seq=12 Ack=12 Win=65152 Len
  10 0.002040145 192.168.0.100  192.168.0.254  TCP        66 50128 → 8000 [ACK] Seq=13 Ack=13 Win=64256 Len=0 TS
  11 0.002370715 192.168.0.254  192.168.0.100  TCP        66 8000 → 50128 [ACK] Seq=13 Ack=13 Win=65152 Len=0 TS

▸ Frame 6: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▸ Ethernet II, Src: Raspberr_d2:8a:96 (b8:27:eb:d2:8a:96), Dst: WistronC_82:4c:bd (30:65:ec:82:4c:bd)
▸ Internet Protocol Version 4, Src: 192.168.0.254, Dst: 192.168.0.100
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 50128, Seq: 1, Ack: 12, Len: 11
    Source Port: 8000
    Destination Port: 50128
    [Stream index: 0]
    [TCP Segment Len: 11]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 12    (relative sequence number)]
    Acknowledgment number: 12    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
    Checksum: 0xb354 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (11 bytes)
▸ Data (11 bytes)

0000  30 65 ec 82 4c bd b8 27  eb d2 8a 96 08 00 45 00   0e..L..' ......E.
0010  00 3f 28 56 40 00 40 06  8f b0 c0 a8 00 fe c0 a8   .?(V@.@. ........
0020  00 64 1f 40 c3 d0 0a 63  e2 5d 5a 07 6e 1e 80 18   .d.@...c .]Z.n...
0030  01 fd b3 54 00 00 01 01  08 0a ac a9 61 cc d1 72   ...T.... ....a..r
0040  f5 77 48 45 4c 4c 4f 20  57 4f 52 4c 44            .wHELLO  WORLD
```

- TCP server: convert lower case string to upper case.
  - An echo function : **single**, multiple and continuous.

Three-way handshake



STEP: 1

DEST: 8000, SRC: 50128, LEN: 0
SEQ: 0x5A076E12, ACK: 0x00000000
Flags: SYN, Data: NONE

STEP: 2

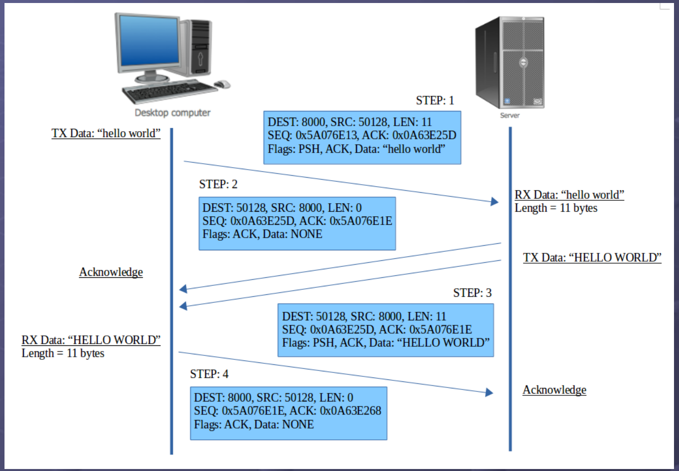
DEST: 50128, SRC: 8000, LEN: 0
SEQ: 0x0A63E25C, ACK: 0x5A076E13
Flags: SYN, ACK, Data: NONE

STEP: 3

DEST: 8000 , SRC: 50128, LEN: 0
SEQ: 0x5A076E13, ACK: 0x0A63E25D
Flags: ACK, Data: NONE

- To initialise a TCP connection the two communicating processes perform a three-way handshake.

TCP



STEP: 1

DEST: 8000, SRC: 50128, LEN: 11
SEQ: 0x5A076E13, ACK: 0x0A63E25D
Flags: PSH, ACK, Data: "hello world"

STEP: 2

DEST: 50128, SRC: 8000, LEN: 0
SEQ: 0x0A63E25D, ACK: 0x5A076E1E
Flags: ACK, Data: NONE

STEP: 3

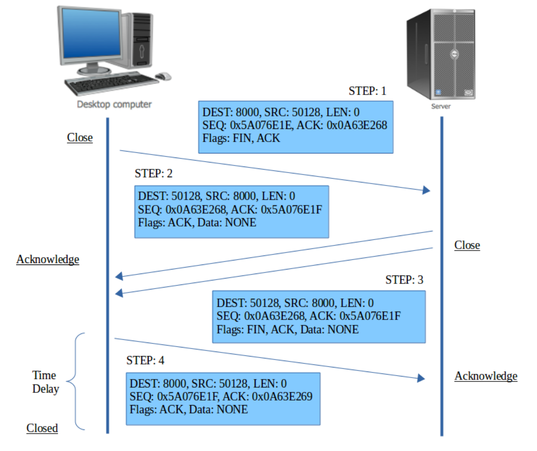
DEST: 50128, SRC: 8000, LEN: 11
SEQ: 0x0A63E25D, ACK: 0x5A076E1E
Flags: PSH, ACK, Data: "HELLO WORLD"

STEP: 4

DEST: 8000, SRC: 50128, LEN: 0
SEQ: 0x5A076E1E, ACK: 0x0A63E268
Flags: ACK, Data: NONE

- To transfer data the client uses the PSH flag

TCP



STEP: 1

DEST: 8000, SRC: 50128, LEN: 0
SEQ: 0x5A076E1E, ACK: 0x0A63E268
Flags: FIN, ACK

STEP: 2

DEST: 50128, SRC: 8000, LEN: 0
SEQ: 0x0A63E268, ACK: 0x5A076E1F
Flags: ACK, Data: NONE

STEP: 3

DEST: 50128, SRC: 8000, LEN: 0
SEQ: 0x0A63E268, ACK: 0x5A076E1F
Flags: FIN, ACK, Data: NONE

STEP: 4

DEST: 8000, SRC: 50128, LEN: 0
SEQ: 0x5A076E1F, ACK: 0x0A63E269
Flags: ACK, Data: NONE

- To close a TCP connection client/server use the FIN flag

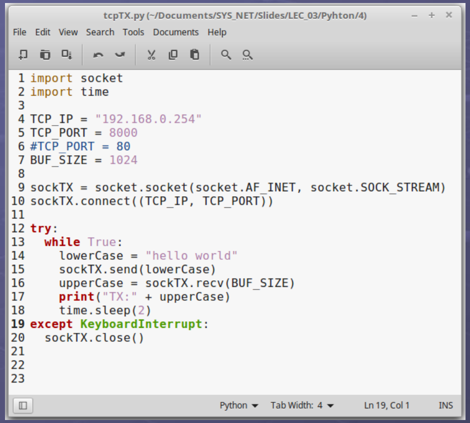
Pause to consider ...

- To ensure reliable connections TCP adds additional initialisation and acknowledgement segments. However, these incur significant overheads i.e. time.
  - To transfer the 10 characters UDP needs 1 segment, TCP needs 9 segments. Therefore, when using TCP we normally keep the link open and transfer multiple values across it.
- This highlights one of the main differences between TCP (stream) vs UDP (message) from the programming point of view i.e. how to separate out different values.
  - UDP: send individual messages i.e. paired send / receive
  - TCP: programmers responsibility to structure the data i.e. either by using a fixed length packets or delimiters such as newline characters.

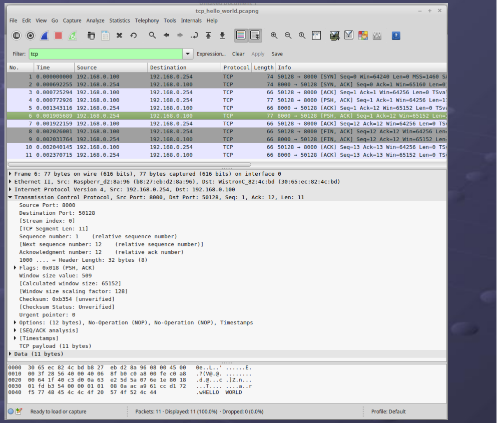
Points to note ...

- Reliable data transfer is implemented using positive acknowledgements and timers (discussed next).
- Establishing a TCP link takes time i.e. 3-way handshake. If the client / servers are situated on different continents this will result in significant delays.
- Therefore, web browsers will tend to keep open TCP connection to avoid having to re-establish a link e.g. HTTP communicates across TCP links, using:
  - Non-Persistent : closed after each object is transferred.
  - Persistent : multiple objects sent over the same link.
  - HTTP1.1 : all connections assumed persistent, however, a web server's default time-out can be 5 – 15 sec.

Demo


tcpTX.py (~/Documents/SYS_NET/Slides/LEC_03/Python/4)
File Edit View Search Tools Documents Help
Python
Tab Width: 4
Ln 19, Col 1
INS

```python
import socket
import time

TCP_IP = "192.168.0.254"
TCP_PORT = 8000
#TCP_PORT = 80
BUF_SIZE = 1024

sockTX = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sockTX.connect((TCP_IP, TCP_PORT))

try:
  while True:
    lowerCase = "hello world"
    sockTX.send(lowerCase)
    upperCase = sockTX.recv(BUF_SIZE)
    print("TX:" + upperCase)
    time.sleep(2)
except KeyboardInterrupt:
  sockTX.close()
```

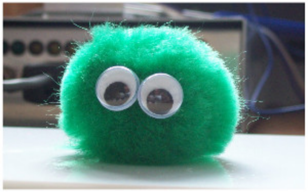

tcp_hello_world.pcapng
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

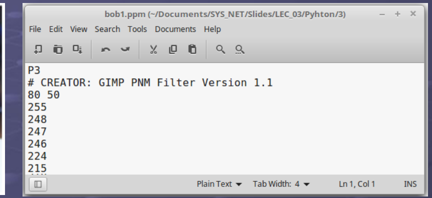
Filter: tcp

```
No.  Time         Source          Destination     Protocol Length Info
  1 0.000000000 192.168.0.100   192.168.0.254   TCP        74 50128 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
  2 0.000692255 192.168.0.254   192.168.0.100   TCP        74 8000 → 50128 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
  3 0.000725294 192.168.0.100   192.168.0.254   TCP        66 50128 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva
  4 0.000772926 192.168.0.100   192.168.0.254   TCP        77 50128 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=1
  5 0.001343116 192.168.0.254   192.168.0.100   TCP        66 8000 → 50128 [ACK] Seq=1 Ack=12 Win=65152 Len=0 TSva
  6 0.001905689 192.168.0.254   192.168.0.100   TCP        77 8000 → 50128 [PSH, ACK] Seq=1 Ack=12 Win=65152 Len=
  7 0.001922159 192.168.0.100   192.168.0.254   TCP        66 50128 → 8000 [ACK] Seq=12 Ack=12 Win=64256 Len=0 TS
  8 0.002026001 192.168.0.100   192.168.0.254   TCP        66 50128 → 8000 [FIN, ACK] Seq=12 Ack=12 Win=64256 Len
  9 0.002031764 192.168.0.254   192.168.0.100   TCP        66 8000 → 50128 [FIN, ACK] Seq=12 Ack=12 Win=65152 Len
 10 0.002040145 192.168.0.100   192.168.0.254   TCP        66 50128 → 8000 [ACK] Seq=13 Ack=13 Win=64256 Len=0 TS
 11 0.002370715 192.168.0.254   192.168.0.100   TCP        66 8000 → 50128 [ACK] Seq=13 Ack=13 Win=65152 Len=0 TS

▸ Frame 6: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▸ Ethernet II, Src: Raspberr_d2:8a:96 (b8:27:eb:d2:8a:96), Dst: WistronC_82:4c:bd (30:65:ec:82:4c:bd)
▸ Internet Protocol Version 4, Src: 192.168.0.254, Dst: 192.168.0.100
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 50128, Seq: 1, Ack: 12, Len: 11
    Source Port: 8000
    Destination Port: 50128
    [Stream index: 0]
    [TCP Segment Len: 11]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 12    (relative sequence number)]
    Acknowledgment number: 12    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
    Checksum: 0xb354 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (11 bytes)
▸ Data (11 bytes)

0000  30 65 ec 82 4c bd b8 27  eb d2 8a 96 08 00 45 00   0e..L..' ......E.
0010  00 3f 28 56 40 00 40 06  8f b0 c0 a8 00 fe c0 a8   .?(V@.@. ........
0020  00 64 1f 40 c3 d0 0a 63  e2 5d 5a 07 6e 1e 80 18   .d.@...c .]Z.n...
0030  01 fd b3 54 00 00 01 01  08 0a ac a9 61 cc d1 72   ...T.... ....a..r
0040  f5 77 48 45 4c 4c 4f 20  57 4f 52 4c 44            .wHELLO  WORLD

Ready to load or capture    Packets: 11 · Displayed: 11 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

- TCP server: convert lower case string to upper case.
  - An echo function : single, **multiple** and **continuous**.

Programming Task

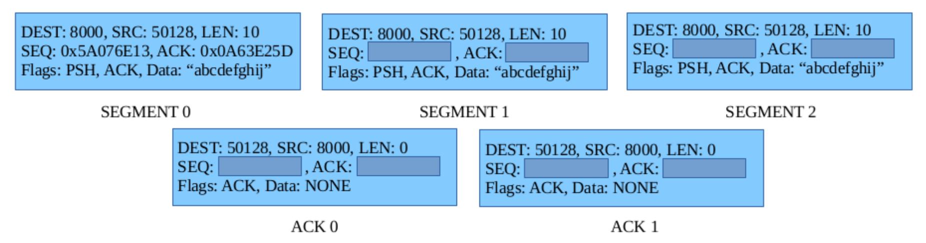



bob1.ppm (~/Documents/SYS_NET/Slides/LEC_03/Python/3)
File Edit View Search Tools Documents Help
Plain Text
Tab Width: 4
Ln 1, Col 1
INS

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
80 50
255
248
247
246
224
215
```

- Q: reimplement the ppm image file (text file) transfer problem (code on VLE) using TCP, that should be simple :).
  - You will find this tricky. You need to think in terms of a stream of data, rather than packets of data.
  - Consider how the name, number of segments and image data will be sent across the TCP connection. Do you need to send the number of segments?

Quick Quizzz



DEST: 8000, SRC: 50128, LEN: 10
SEQ: 0x5A076E13, ACK: 0x0A63E25D
Flags: PSH, ACK, Data: “abcdefghij”

SEGMENT 0

DEST: 8000, SRC: 50128, LEN: 10
SEQ: , ACK:
Flags: PSH, ACK, Data: “abcdefghij”

SEGMENT 1

DEST: 8000, SRC: 50128, LEN: 10
SEQ: , ACK:
Flags: PSH, ACK, Data: “abcdefghij”

SEGMENT 2

DEST: 50128, SRC: 8000, LEN: 0
SEQ: , ACK:
Flags: ACK, Data: NONE

ACK 0

DEST: 50128, SRC: 8000, LEN: 0
SEQ: , ACK:
Flags: ACK, Data: NONE

ACK 1

- The most important parts of the TCP header are the SEQ and ACK numbers i.e. to ensure reliability.
  - SEQ number is the byte stream number of the first byte in the segment.
  - ACK number is the sequence number of the next byte the RX host is expecting from the TX host.
- Q : what are the missing the SEQ and ACK numbers if:
  - RX in order segment 0,1,2 etc, or RX out of order 0,2,1?

Pause to consider ...

- That was the easy bit :), now need to consider the what ifs :(
- IpoAC: multiple packets in flight at the same time, these may be received out of order, packet lose may also occur ...

TCP

- To help understand how TCP operates consider the following simplified TX pseudo code.
- Three main events
  - Application data
  - Time-out
  - Acknowledgement

```
NextSeqNum ← RAN()
PrevSeqNum ← NextSeqNum
segment ← []

while TRUE do
   case EVENT of
      application_layer_data:
         segment.create( data, NextSeqNum )
         if timer.state = stop then
            timer.state ← start
         end if
         networkLayer( segment )
         NextSeqNum ← NextSeqNum + len( data )

      time_out:
         networkLayer( segment.select( notACK, smallestSEQ  )
         timer.state ← start

      rx_ACK:
         if ACK > PrevSeqNum then
            segment.remove( ACK )
            PrevSeqNum ← ACK
            if segment.NotACK() then
               timer.state ← start
            end if
         end if
      end case
end while
```

Time-outs

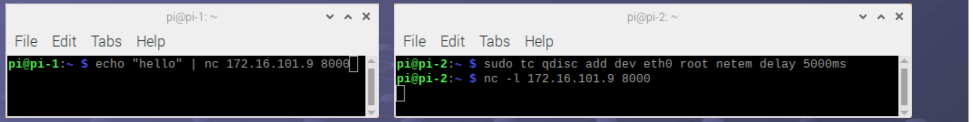
$$\alpha = 0.125$$
$$estRTT = (1 - \alpha) \times estRTT + \alpha \times curRTT$$

$$\beta = 0.25$$
$$devRTT = (1 - \beta) \times devRTT + \beta \times | curRTT - estRTT |$$

$$Timeout = estRTT + 4 \times devRTT$$

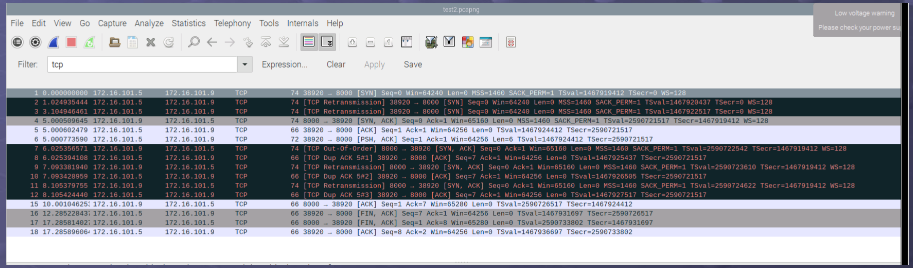
- Q: how do we choose the time-out value
  - Too short and we will reject valid segments in transit. Too long and we waste time waiting for segments that will never arrive.
- RFC 6298 : round trip time (RTT) delay. Initial time-out set to 1 sec. If time-out occurs, time-out value doubled, reset on next RX segment, do not use times from re-transmitted segments.

Demo



```
pi@pi-1:~ $ echo "hello" | nc 172.16.101.9 8000
```

```
pi@pi-2:~ $ sudo tc qdisc add dev eth0 root netem delay 5000ms
pi@pi-2:~ $ nc -l 172.16.101.9 8000
```



test2.pcapng
Low voltage warning
Please check your power su
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

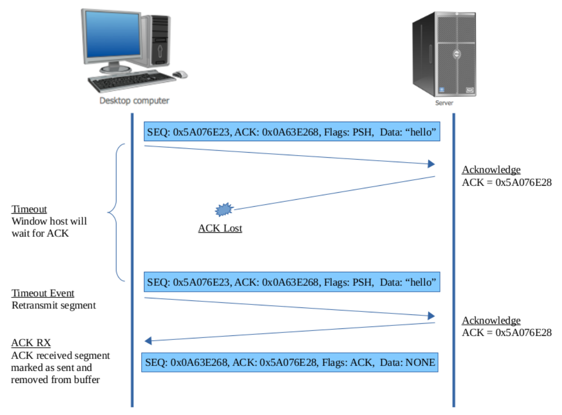
Filter: tcp

```
 1 0.000000000 172.16.101.5  172.16.101.9  TCP  74 38920 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1467919412 TSecr=0 WS=128
 2 1.024935444 172.16.101.5  172.16.101.9  TCP  74 [TCP Retransmission] 38920 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1467920437 TSecr=0 WS=128
 3 3.104946461 172.16.101.5  172.16.101.9  TCP  74 [TCP Retransmission] 38920 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1467922517 TSecr=0 WS=128
 4 5.000509645 172.16.101.9  172.16.101.5  TCP  74 8000 → 38920 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2590721517 TSecr=1467919412 WS=128
 5 5.000602479 172.16.101.5  172.16.101.9  TCP  66 38920 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1467924412 TSecr=2590721517
 6 5.000773590 172.16.101.5  172.16.101.9  TCP  72 38920 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=6 TSval=1467924412 TSecr=2590721517
 7 6.025356571 172.16.101.9  172.16.101.5  TCP  74 [TCP Out-Of-Order] 8000 → 38920 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2590722542 TSecr=1467919412 WS=128
 8 6.025394108 172.16.101.5  172.16.101.9  TCP  66 [TCP Dup ACK 5#1] 38920 → 8000 [ACK] Seq=7 Ack=1 Win=64256 Len=0 TSval=1467925437 TSecr=2590721517
 9 7.093381940 172.16.101.9  172.16.101.5  TCP  74 [TCP Retransmission] 8000 → 38920 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2590723610 TSecr=1467919412 WS=128
10 7.093428959 172.16.101.5  172.16.101.9  TCP  66 [TCP Dup ACK 5#2] 38920 → 8000 [ACK] Seq=7 Ack=1 Win=64256 Len=0 TSval=1467926505 TSecr=2590721517
11 8.105379755 172.16.101.9  172.16.101.5  TCP  74 [TCP Retransmission] 8000 → 38920 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2590724622 TSecr=1467919412 WS=128
12 8.105424440 172.16.101.5  172.16.101.9  TCP  66 [TCP Dup ACK 5#3] 38920 → 8000 [ACK] Seq=7 Ack=1 Win=64256 Len=0 TSval=1467927517 TSecr=2590721517
15 10.00104625: 172.16.101.9 172.16.101.5  TCP  66 8000 → 38920 [ACK] Seq=1 Ack=7 Win=65280 Len=0 TSval=2590726517 TSecr=1467924412
16 12.28522843: 172.16.101.5 172.16.101.9  TCP  66 38920 → 8000 [FIN, ACK] Seq=7 Ack=1 Win=64256 Len=0 TSval=1467931697 TSecr=2590726517
17 17.28581402: 172.16.101.9 172.16.101.5  TCP  66 8000 → 38920 [FIN, ACK] Seq=1 Ack=8 Win=65280 Len=0 TSval=2590733802 TSecr=1467931697
18 17.28589606: 172.16.101.5 172.16.101.9  TCP  66 38920 → 8000 [ACK] Seq=8 Ack=2 Win=64256 Len=0 TSval=1467936697 TSecr=2590733802
```

- Command line tool : tc (traffic control)
  - Add 5000ms delay from Pi-1 to Pi-2
  - Work through the time line of errors (black lines), what causes what, tricky :)

Pause to consider ...

- Time-out example 1
  - TX host sends a message, it does not receive an ACK, timeout triggered.
  - Q: what happens next i.e. to the second "hello" RX by the server?



Desktop computer

Server

SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: "hello"

Acknowledge
ACK = 0x5A076E28

Timeout
Window host will
wait for ACK

ACK Lost

SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: "hello"

Timeout Event
Retransmit segment

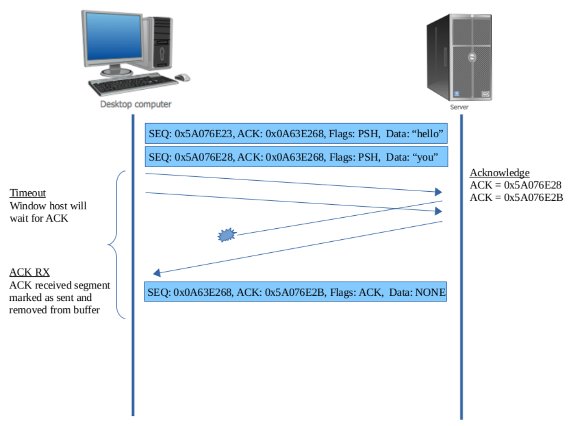
Acknowledge
ACK = 0x5A076E28

ACK RX
ACK received segment
marked as sent and
removed from buffer

SEQ: 0x0A63E268, ACK: 0x5A076E28, Flags: ACK, Data: NONE

Pause to consider ...

- Time-out example 2
  - TX host sends a message, but only one ACK received.
  - Q: what happens next i.e. does the client need the missing ACK?



Desktop computer

Server

SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: "hello"

SEQ: 0x5A076E28, ACK: 0x0A63E268, Flags: PSH, Data: "you"

Acknowledge
ACK = 0x5A076E28
ACK = 0x5A076E2B

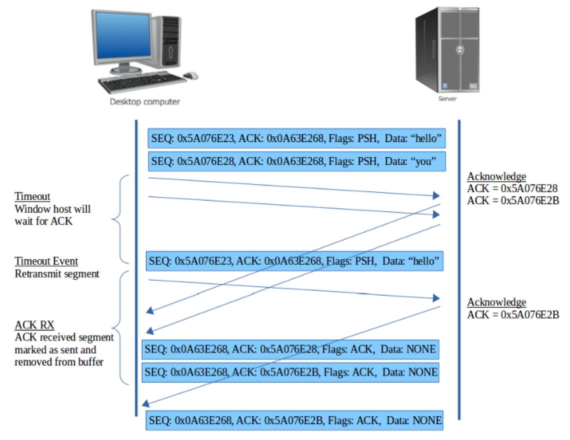
Timeout
Window host will
wait for ACK

ACK RX
ACK received segment
marked as sent and
removed from buffer

SEQ: 0x0A63E268, ACK: 0x5A076E2B, Flags: ACK, Data: NONE

Pause to consider ...

- Time-out example 3
  - TX host sends a message, but ACKs delayed, triggering a timeout.
  - Q: what happens next i.e. does the client know that all data has been RX?



Desktop computer

Server

SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: "hello"

SEQ: 0x5A076E28, ACK: 0x0A63E268, Flags: PSH, Data: "you"

Acknowledge
ACK = 0x5A076E28
ACK = 0x5A076E2B

Timeout
Window host will
wait for ACK

Timeout Event
Retransmit segment

SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: "hello"

Acknowledge
ACK = 0x5A076E2B

ACK RX
ACK received segment
marked as sent and
removed from buffer

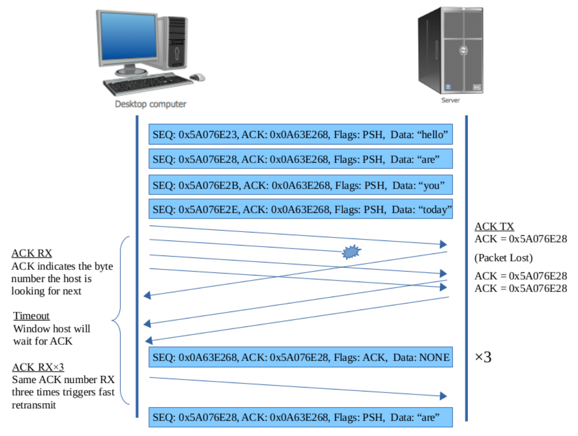
SEQ: 0x0A63E268, ACK: 0x5A076E28, Flags: ACK, Data: NONE

SEQ: 0x0A63E268, ACK: 0x5A076E2B, Flags: ACK, Data: NONE

SEQ: 0x0A63E268, ACK: 0x5A076E2B, Flags: ACK, Data: NONE

Pause to consider ...

- Time-out example 4
  - TX host sends a message but one of the “words” is lost.
  - Q: what happens next?
    - Fast retransmit
  - Q: what happens if only one word is sent?



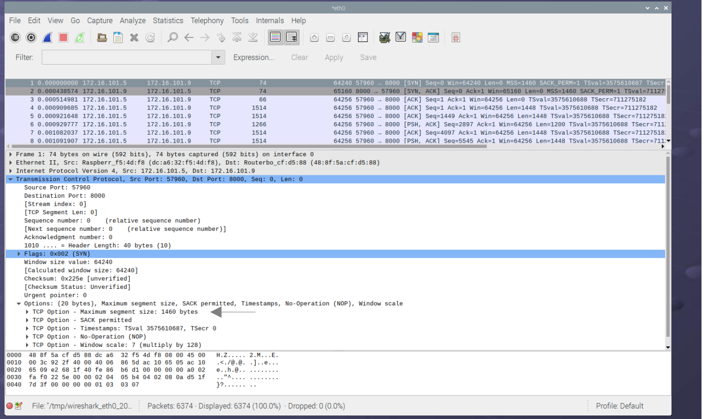
Desktop computer
Server
SEQ: 0x5A076E23, ACK: 0x0A63E268, Flags: PSH, Data: “hello”
SEQ: 0x5A076E28, ACK: 0x0A63E268, Flags: PSH, Data: “are”
SEQ: 0x5A076E2B, ACK: 0x0A63E268, Flags: PSH, Data: “you”
SEQ: 0x5A076E2E, ACK: 0x0A63E268, Flags: PSH, Data: “today”
ACK TX
ACK = 0x5A076E28
(Packet Lost)
ACK = 0x5A076E28
ACK = 0x5A076E28
ACK RX
ACK indicates the byte number the host is looking for next
Timeout
Window host will wait for ACK
ACK RX×3
Same ACK number RX three times triggers fast retransmit
SEQ: 0x0A63E268, ACK: 0x5A076E28, Flags: ACK, Data: NONE
×3
SEQ: 0x5A076E28, ACK: 0x0A63E268, Flags: PSH, Data: “are”

Pause to consider ...

- TX host transmits segments onto transport mechanism. Therefore, there can be multiple segments in flight.
  - $D_{End-End} = N_{stages}(D_{proc} + D_{Que} + D_{Tran} + D_{prop})$
  - Q : if we are transmitting 1000 byte segments to Australia at 1Mbps how many segments can be in flight?
- The RX host receives these segments, but may not process them immediately owing to:
  - Time taken to pass up the layers in the protocol stack
  - The OS is executing other processes.
  - There is a significant processing overhead for the application using these segments.
- Problem : RX host can be swamped with data i.e. data is coming in faster than it can process :(.

MSS

- During 3-way handshake hosts signal MSS, max encapsulated data size
  - not including IP or TCP headers
  - If not specified defaults to 536B

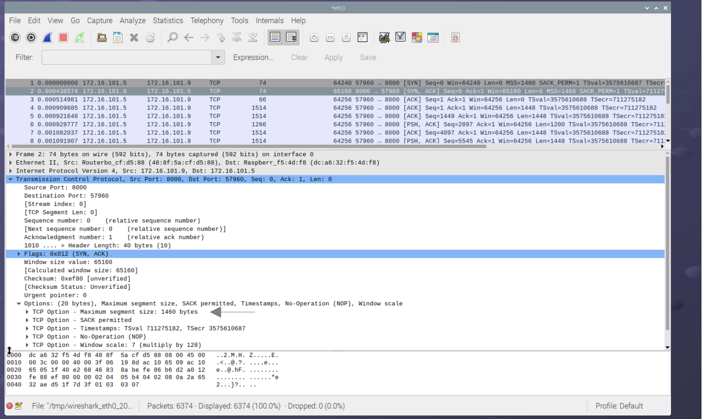


*eth0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
    1 0.000000000 172.16.101.5    172.16.101.9    TCP     74      64240 57960 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3575610687 TSecr
    2 0.000438574 172.16.101.9    172.16.101.5    TCP     74      65160 8000 → 57960 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=71127
    3 0.000514981 172.16.101.5    172.16.101.9    TCP     66      64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3575610688 TSecr=711275182
    4 0.000909685 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711275182
    5 0.000921648 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=1449 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    6 0.000929777 172.16.101.5    172.16.101.9    TCP   1266      64256 57960 → 8000 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=1200 TSval=3575610688 TSecr=711
    7 0.001082037 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=4097 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    8 0.001091907 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [PSH, ACK] Seq=5545 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711
▸ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: Raspberr_f5:4d:f8 (dc:a6:32:f5:4d:f8), Dst: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88)
▸ Internet Protocol Version 4, Src: 172.16.101.5, Dst: 172.16.101.9
▾ Transmission Control Protocol, Src Port: 57960, Dst Port: 8000, Seq: 0, Len: 0
    Source Port: 57960
    Destination Port: 8000
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x002 (SYN)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x225e [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    ▸ TCP Option - Maximum segment size: 1460 bytes
    ▸ TCP Option - SACK permitted
    ▸ TCP Option - Timestamps: TSval 3575610687, TSecr 0
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Window scale: 7 (multiply by 128)
0000  48 8f 5a cf d5 88 dc a6  32 f5 4d f8 08 00 45 00   H.Z..... 2.M...E.
0010  00 3c 92 2f 40 00 40 06  86 5d ac 10 65 05 ac 10   .</@.@. .]..e...
0020  65 09 e2 68 1f 40 fe 86  b6 d1 00 00 00 00 a0 02   e..h.@.. ........
0030  fa f0 22 5e 00 00 02 04  05 b4 04 02 08 0a d5 1f   .."^.... ........
0040  7d 3f 00 00 00 00 01 03  03 07                     }?...... ..
File: "/tmp/wireshark_eth0_20...   Packets: 6374 · Displayed: 6374 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

MSS

- During 3-way handshake hosts signal MSS, max encapsulated data size
  - not including IP or TCP headers
  - If not specified defaults to 536B



*eth0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
    1 0.000000000 172.16.101.5    172.16.101.9    TCP     74      64240 57960 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3575610687 TSecr
    2 0.000438574 172.16.101.9    172.16.101.5    TCP     74      65160 8000 → 57960 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=71127
    3 0.000514981 172.16.101.5    172.16.101.9    TCP     66      64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3575610688 TSecr=711275182
    4 0.000909685 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711275182
    5 0.000921648 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=1449 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    6 0.000929777 172.16.101.5    172.16.101.9    TCP   1266      64256 57960 → 8000 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=1200 TSval=3575610688 TSecr=711
    7 0.001082037 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [ACK] Seq=4097 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    8 0.001091907 172.16.101.5    172.16.101.9    TCP   1514      64256 57960 → 8000 [PSH, ACK] Seq=5545 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711
▸ Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88), Dst: Raspberr_f5:4d:f8 (dc:a6:32:f5:4d:f8)
▸ Internet Protocol Version 4, Src: 172.16.101.9, Dst: 172.16.101.5
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 57960, Seq: 0, Ack: 1, Len: 0
    Source Port: 8000
    Destination Port: 57960
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x012 (SYN, ACK)
    Window size value: 65160
    [Calculated window size: 65160]
    Checksum: 0xef80 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    ▸ TCP Option - Maximum segment size: 1460 bytes
    ▸ TCP Option - SACK permitted
    ▸ TCP Option - Timestamps: TSval 711275182, TSecr 3575610687
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Window scale: 7 (multiply by 128)
0000  dc a6 32 f5 4d f8 48 8f  5a cf d5 88 08 00 45 00   ..2.M.H. Z.....E.
0010  00 3c 00 00 40 00 3f 06  19 8d ac 10 65 09 ac 10   .<..@.?. ....e...
0020  65 05 1f 40 e2 68 46 83  8a be fe 86 b6 d2 a0 12   e..@.hF. ........
0030  fe 88 ef 80 00 00 02 04  05 b4 04 02 08 0a 2a 65   ........ ......*e
0040  32 ae d5 1f 7d 3f 01 03  03 07                     2...}?.. ..
File: "/tmp/wireshark_eth0_20...   Packets: 6374 · Displayed: 6374 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

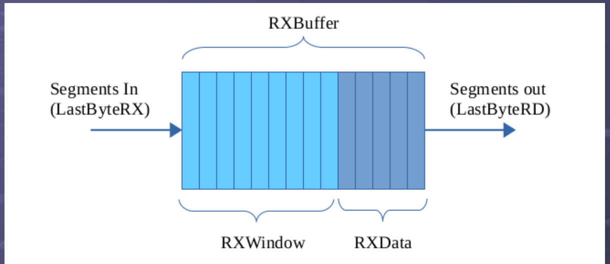
Flow control

Typical RXBuffer size set to Bandwidth Delay Product (BDP)

Typical LAN

1 Gbit/s * 1ms RTT
10^9 * 10^-3 = 125kB

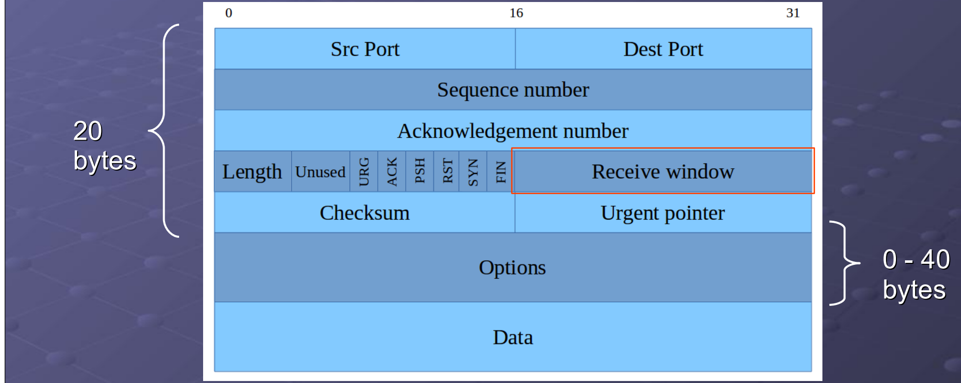
Needs to be larger for slower networks



```
RX Host : RXWindow = RXBuffer – (LastByteRX – LastByteRD)
TX Host : RXCapacity = RXWindow – (LastByteSend – LastByteACK)
```

- Both TX and RX hosts have buffers to store data when busy. The free space in the RX host’s buffer (RXWindow) is transmitted to TX host to ensure buffer overflow does not occur.
  - Data is temporarily stored on “wire” i.e. data in flight.
  - RX window size transmitted back to host in ACK packets.

Flow control



0 16 31

Src Port | Dest Port

Sequence number

Acknowledgement number

Length | Unused | URG | ACK | PSH | RST | SYN | FIN | Receive window

Checksum | Urgent pointer

Options

Data

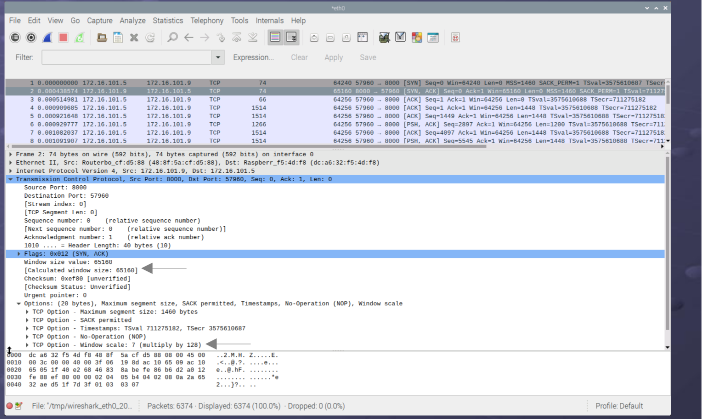
20 bytes

0 - 40 bytes

- Hosts inform each other how much data they can receive using the Receive Window field (16bits)

Flow control

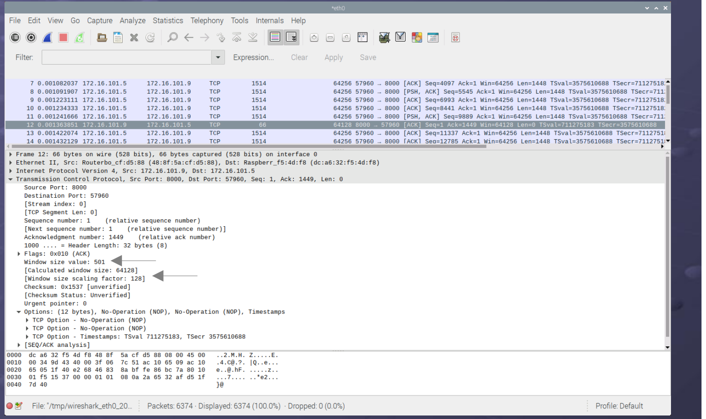
- Window scaling factor sent during 3-way handshake
- Allowing RX windows greater than 64KB


*eth0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
    1 0.000000000 172.16.101.5      172.16.101.9      TCP        74            64240 57960 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3575610687 TSecr
    2 0.000438574 172.16.101.9      172.16.101.5      TCP        74            65160 8000 → 57960 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=71127
    3 0.000514981 172.16.101.5      172.16.101.9      TCP        66            64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3575610688 TSecr=711275182
    4 0.000909685 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=1 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711275182
    5 0.000921648 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=1449 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    6 0.000929777 172.16.101.5      172.16.101.9      TCP      1266            64256 57960 → 8000 [PSH, ACK] Seq=2897 Ack=1 Win=64256 Len=1200 TSval=3575610688 TSecr=711
    7 0.001082037 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=4097 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    8 0.001091907 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [PSH, ACK] Seq=5545 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711
▸ Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88), Dst: Raspberr_f5:4d:f8 (dc:a6:32:f5:4d:f8)
▸ Internet Protocol Version 4, Src: 172.16.101.9, Dst: 172.16.101.5
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 57960, Seq: 0, Ack: 1, Len: 0
    Source Port: 8000
    Destination Port: 57960
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x012 (SYN, ACK)
    Window size value: 65160
    [Calculated window size: 65160]
    Checksum: 0xef80 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    ▸ TCP Option - Maximum segment size: 1460 bytes
    ▸ TCP Option - SACK permitted
    ▸ TCP Option - Timestamps: TSval 711275182, TSecr 3575610687
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Window scale: 7 (multiply by 128)
0000  dc a6 32 f5 4d f8 48 8f  5a cf d5 88 08 00 45 00   ..2.M.H. Z.....E.
0010  00 3c 00 00 40 00 3f 06  19 8d ac 10 65 09 ac 10   .<..@.?. ....e...
0020  65 05 1f 40 e2 68 46 83  8a be fe 86 b6 d2 a0 12   e..@.hF. ........
0030  fe 88 ef 80 00 00 02 04  05 b4 04 02 08 0a 2a 65   ........ ......*e
0040  32 ae d5 1f 7d 3f 01 03  03 07                     2...}?.. ..
File: "/tmp/wireshark_eth0_20...   Packets: 6374 · Displayed: 6374 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

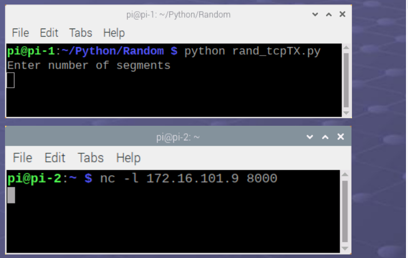
Flow control

- Receive window size : how much data can be in flight.
- Receive buffer size: how much data can be stored in RX host.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

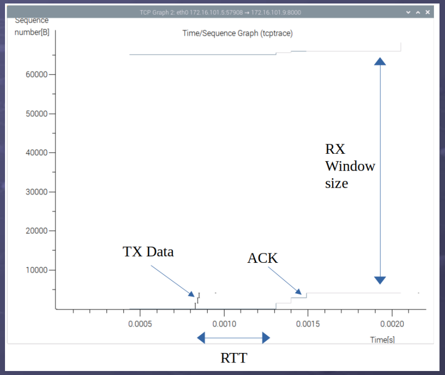
```
    7 0.001082037 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=4097 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
    8 0.001091907 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [PSH, ACK] Seq=5545 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711
    9 0.001223111 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=6993 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
   10 0.001234333 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=8441 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=71127518
   11 0.001241666 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [PSH, ACK] Seq=9889 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=711
   12 0.001363851 172.16.101.9      172.16.101.5      TCP        66            64128 8000 → 57960 [ACK] Seq=1 Ack=1449 Win=64128 Len=0 TSval=711275183 TSecr=3575610688
   13 0.001422074 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=11337 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=7112751
   14 0.001432129 172.16.101.5      172.16.101.9      TCP      1514            64256 57960 → 8000 [ACK] Seq=12785 Ack=1 Win=64256 Len=1448 TSval=3575610688 TSecr=7112751
▸ Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88), Dst: Raspberr_f5:4d:f8 (dc:a6:32:f5:4d:f8)
▸ Internet Protocol Version 4, Src: 172.16.101.9, Dst: 172.16.101.5
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 57960, Seq: 1, Ack: 1449, Len: 0
    Source Port: 8000
    Destination Port: 57960
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1449    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x010 (ACK)
    Window size value: 501
    [Calculated window size: 64128]
    [Window size scaling factor: 128]
    Checksum: 0x1537 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Timestamps: TSval 711275183, TSecr 3575610688
  ▸ [SEQ/ACK analysis]
0000  dc a6 32 f5 4d f8 48 8f  5a cf d5 88 08 00 45 00   ..2.M.H. Z.....E.
0010  00 34 9d 43 40 00 3f 06  7c 51 ac 10 65 09 ac 10   .4.C@.?. |Q..e...
0020  65 05 1f 40 e2 68 46 83  8a bf fe 86 bc 7a 80 10   e..@.hF. .....z..
0030  01 f5 15 37 00 00 01 01  08 0a 2a 65 32 af d5 1f   ...7.... ..*e2...
0040  7d 40                                              }@
File: "/tmp/wireshark_eth0_20...   Packets: 6374 · Displayed: 6374 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

Demo



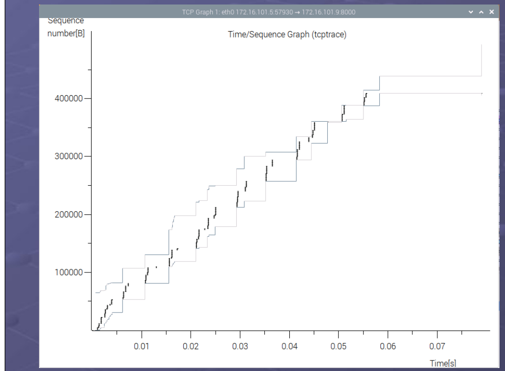
```
pi@pi-1:~/Python/Random $ python rand_tcpTX.py
Enter number of segments
```

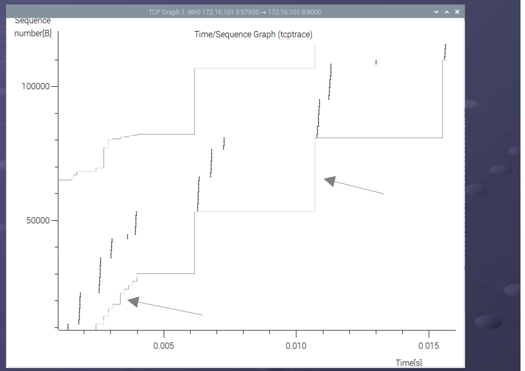
```
pi@pi-2:~ $ nc -l 172.16.101.9 8000
```



- Flow control : a little data (1).
  - Test code: simple python program transferring one or more TCP packets contain random values.

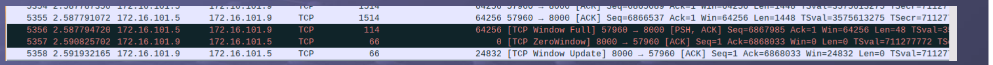
Demo



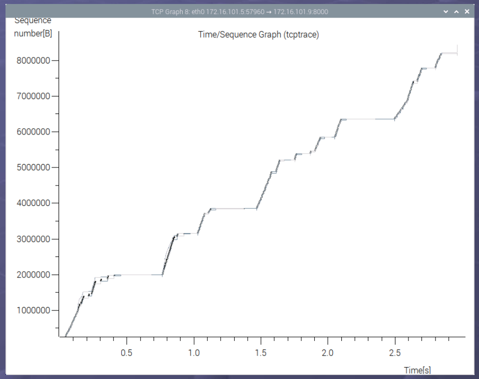


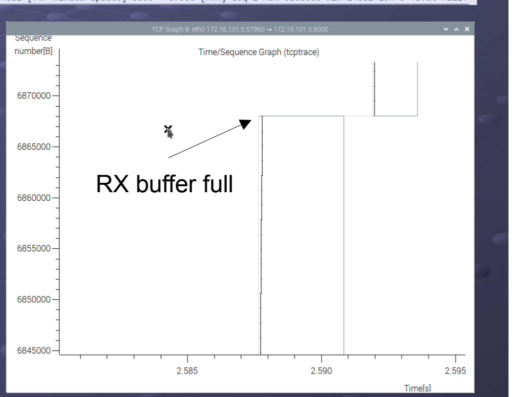
- Flow control : a bit “more” data.
  - More data in flight
  - Move from individual ACKs to cumulative ACK

Demo



```
5355 2.587791072 172.16.101.5      172.16.101.9      TCP        1514             64256 57960 → 8000 [ACK] Seq=6866537 Ack=1 Win=64256 Len=1448 TSval=3575613275 TSecr=71127
5356 2.587794720 172.16.101.5      172.16.101.9      TCP         114             64256 [TCP Window Full] 57960 → 8000 [PSH, ACK] Seq=6867985 Ack=1 Win=64256 Len=48 TSval=3
5357 2.590825702 172.16.101.9      172.16.101.5      TCP          66                 0 [TCP ZeroWindow] 8000 → 57960 [ACK] Seq=1 Ack=6868033 Win=0 Len=0 TSval=711277772 TS
5358 2.591932165 172.16.101.9      172.16.101.5      TCP          66             24832 [TCP Window Update] 8000 → 57960 [ACK] Seq=1 Ack=6868033 Win=24832 Len=0 TSval=71127
```

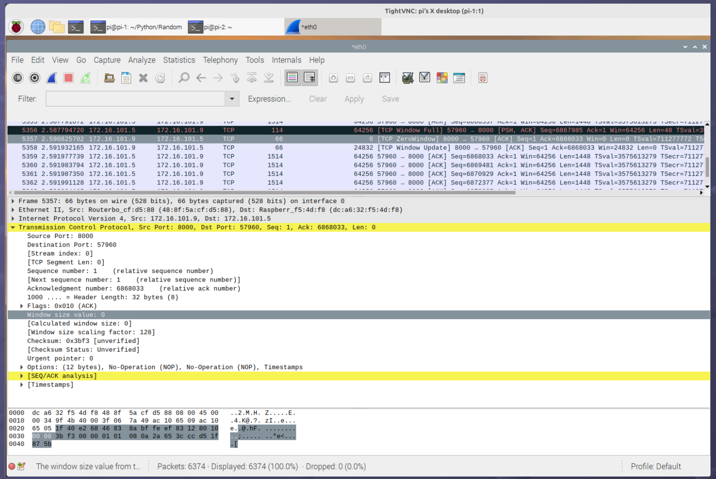




RX buffer full

- Flow control : a lot of data.

Demo



TightVNC: pi's X desktop (pi-1:1)
pi@pi-1: ~/Python/Random
pi@pi-2: ~
*eth0
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save

```
5356 2.587794720 172.16.101.5   172.16.101.9   TCP    114    64256 [TCP Window Full] 57960 → 8000 [PSH, ACK] Seq=6867985 Ack=1 Win=64256 Len=48 TSval=3
5357 2.590825702 172.16.101.9   172.16.101.5   TCP     66        0 [TCP ZeroWindow] 8000 → 57960 [ACK] Seq=1 Ack=6868033 Win=0 Len=0 TSval=711277772 TS
5358 2.591932165 172.16.101.9   172.16.101.5   TCP     66    24832 [TCP Window Update] 8000 → 57960 [ACK] Seq=1 Ack=6868033 Win=24832 Len=0 TSval=71127
5359 2.591977739 172.16.101.5   172.16.101.9   TCP   1514    64256 57960 → 8000 [ACK] Seq=6868033 Ack=1 Win=64256 Len=1448 TSval=3575613279 TSecr=71127
5360 2.591983794 172.16.101.5   172.16.101.9   TCP   1514    64256 57960 → 8000 [ACK] Seq=6869481 Ack=1 Win=64256 Len=1448 TSval=3575613279 TSecr=71127
5361 2.591987350 172.16.101.5   172.16.101.9   TCP   1514    64256 57960 → 8000 [ACK] Seq=6870929 Ack=1 Win=64256 Len=1448 TSval=3575613279 TSecr=71127
5362 2.591991128 172.16.101.5   172.16.101.9   TCP   1514    64256 57960 → 8000 [ACK] Seq=6872377 Ack=1 Win=64256 Len=1448 TSval=3575613279 TSecr=71127
```

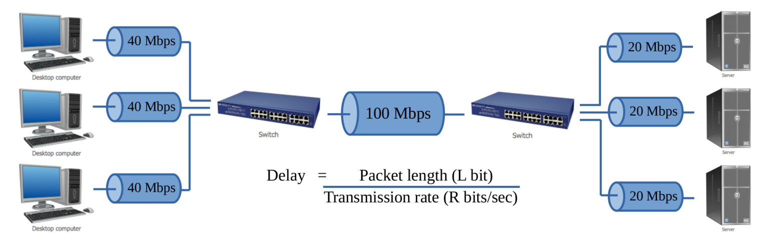
```
▸ Frame 5357: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88), Dst: Raspberr_f5:4d:f8 (dc:a6:32:f5:4d:f8)
▸ Internet Protocol Version 4, Src: 172.16.101.9, Dst: 172.16.101.5
▾ Transmission Control Protocol, Src Port: 8000, Dst Port: 57960, Seq: 1, Ack: 6868033, Len: 0
    Source Port: 8000
    Destination Port: 57960
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 6868033    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x010 (ACK)
    Window size value: 0
    [Calculated window size: 0]
    [Window size scaling factor: 128]
    Checksum: 0x3bf3 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
```

```
0000  dc a6 32 f5 4d f8 48 8f  5a cf d5 88 08 00 45 00   ..2.M.H. Z.....E.
0010  00 34 9f 4b 40 00 3f 06  7a 49 ac 10 65 09 ac 10   .4.K@.?. zI..e...
0020  65 05 1f 40 e2 68 46 83  8a bf fe ef 83 12 80 10   e..@.hF. ........
0030  00 00 3b f3 00 00 01 01  08 0a 2a 65 3c cc d5 1f   ..;..... ..*e<...
0040  87 5b                                              .[
```

The window size value from t... Packets: 6374 · Displayed: 6374 (100.0%) · Dropped: 0 (0.0%) Profile: Default

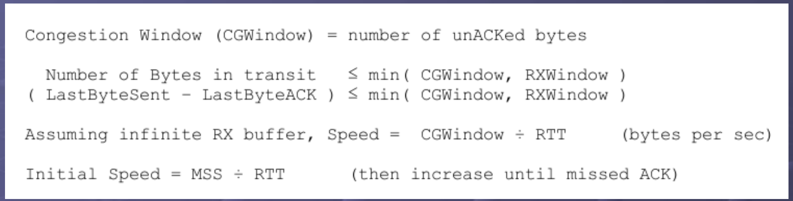
- RX window full (0) TX host must stop sending data.

Pause to consider ...



- We have seen that TCP can adjust TX speed to match the RX speed to avoid buffer overflow.
- However, what should the system do if we just have too much traffic on the network i.e. segments are being lost because of buffer overflows in routers.
  - Remember packet switching uses store and forward.

Congestion control



```
Congestion Window (CGWindow) = number of unACKed bytes

  Number of Bytes in transit   ≤ min( CGWindow, RXWindow )
( LastByteSent - LastByteACK ) ≤ min( CGWindow, RXWindow )

Assuming infinite RX buffer, Speed =  CGWindow ÷ RTT     (bytes per sec)

Initial Speed = MSS ÷ RTT       (then increase until missed ACK)
```

- TCP handles congested networks by using another "window", the Congestion window :). It detects a networks "loading" via the ACK segments
  - ACK received for unACKed segment. All is good increase congestion window size i.e. number of segment in flight.
  - Time-out, or multiple ACKs for same segment RX. Network congested, decrease congestion window size, i.e. packets are being dropped, reduce traffic.

Congestion control

- TCP congestion-control algorithm
  - RFC 5681: https://tools.ietf.org/html/rfc5681
  - Three core elements: slow start, congestion avoidance and fast recovery (implementations do vary)
- Slow start :
  - Initially CGWnd set to 1 MSS i.e. speed= MSS/RTT.
  - On each RX ACK add 1 MSS to CGWnd i.e. doubled, speed increases exponentially, not too slow :).
  - Continues until ACK time-out i.e. host probes the network to find "max" speed. SSThresold set to CGWnd/2 and CGWnd reset to 1 MSS.
  - Process repeated until SSThresold reached, then switch to congestion avoidance mode.

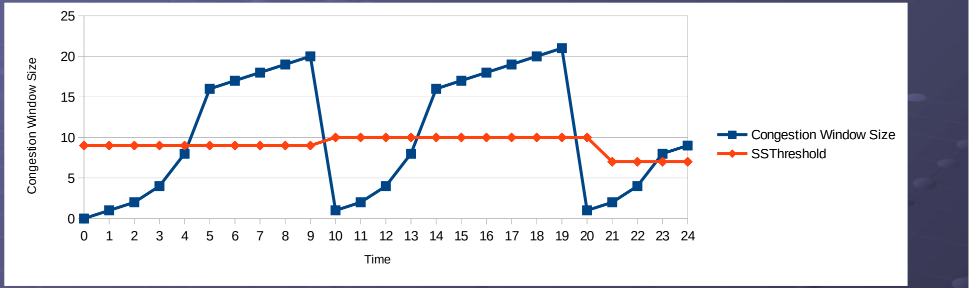
Congestion control

  - If three repeated ACKs for the same SEQ number RX, perform fast retransmit and change mode to fast recovery.
- Congestion avoidance:
  - CGWnd is half the value at which congestion was detected. Now rather than doubling CGWnd size it is incremented by 1 MSS when all CGWnd segments have been ACKed i.e. linear increase rather than exponential.
  - If ACK time-out. SSThresold set to CGWnd/2, CGWnd reset to 1 MSS, switch back to soft start mode.
  - If three repeated ACKs for the same SEQ number RX, perform fast retransmit and change mode to fast recovery.

Congestion control

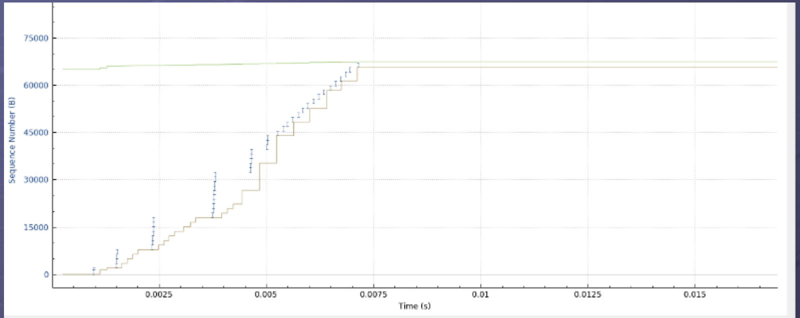
- Fast recovery :
  - CGWnd set to (CGWnd/2)+3MSS. CGWnd increased by MSS for each duplicate ACK RX.
  - If ACK time-out occurs, CGWnd reset to 1 MSS and SSThresold set to CGWnd/2, switch back to soft start mode.
  - If ACK received, CGWnd set to SSThresold, then enter congestion avoidance mode.

Congestion control



- Note, different strategies taken depending on the version of TCP used. Refer to RFC5681 for additional details e.g. initial CGWnd size, strategies used in each mode to adjust CGWnd size etc.

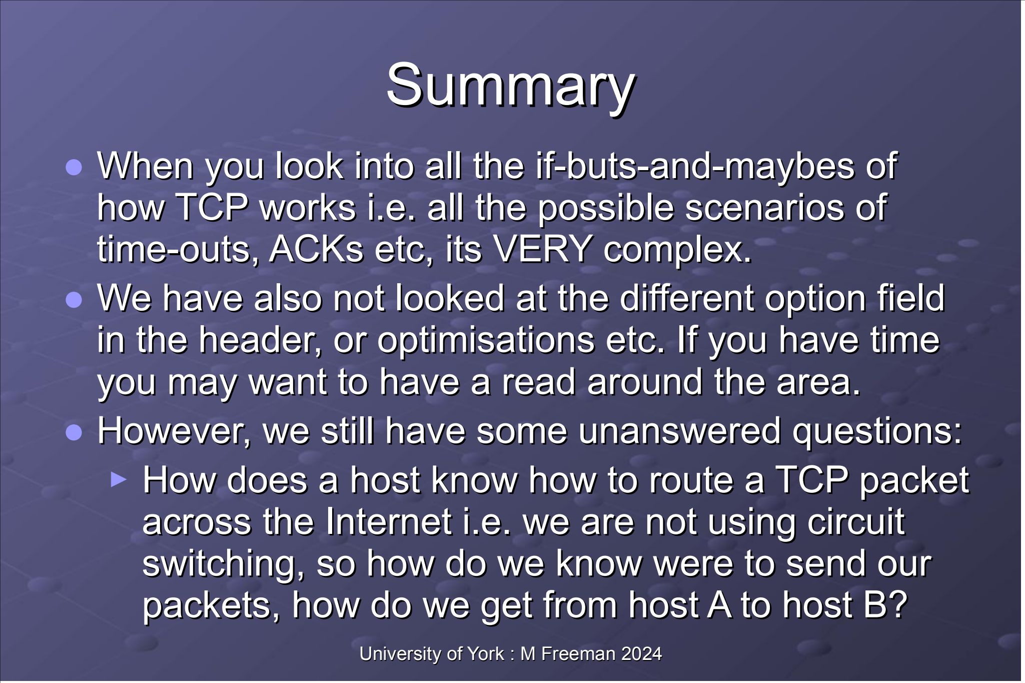
Congestion control



- Remember that flow control i.e. RXWindow size, is also active at this time, therefore, speed will also be throttled based on RX host buffer size.

Summary

- TCP is used to implement:
  - Telnet, SSH, FTP, HTTP, SMTP, POP …
  - The internet's workhorse protocol.
- But, to ensure a reliable connection we need additional state information (memory on host), header fields and handshakes in protocol etc, i.e. reliability has a cost.
  - Sequence and acknowledgement numbers to ensure segments are not lost.
  - Time-outs, different types of segments (flags) i.e. SYN, ACK, PSH etc.
  - Initial three way handshake to ensure server is ready to communicate and agree upon parameters used in this transfer e.g. RXWindow sizes.



Summary

- When you look into all the if-buts-and-maybes of how TCP works i.e. all the possible scenarios of time-outs, ACKs etc, its VERY complex.
- We have also not looked at the different option field in the header, or optimisations etc. If you have time you may want to have a read around the area.
- However, we still have some unanswered questions:
  - How does a host know how to route a TCP packet across the Internet i.e. we are not using circuit switching, so how do we know were to send our packets, how do we get from host A to host B?