CS 305 Lab Tutorial
Lab 8 TCP Sliding Window & WebSocket

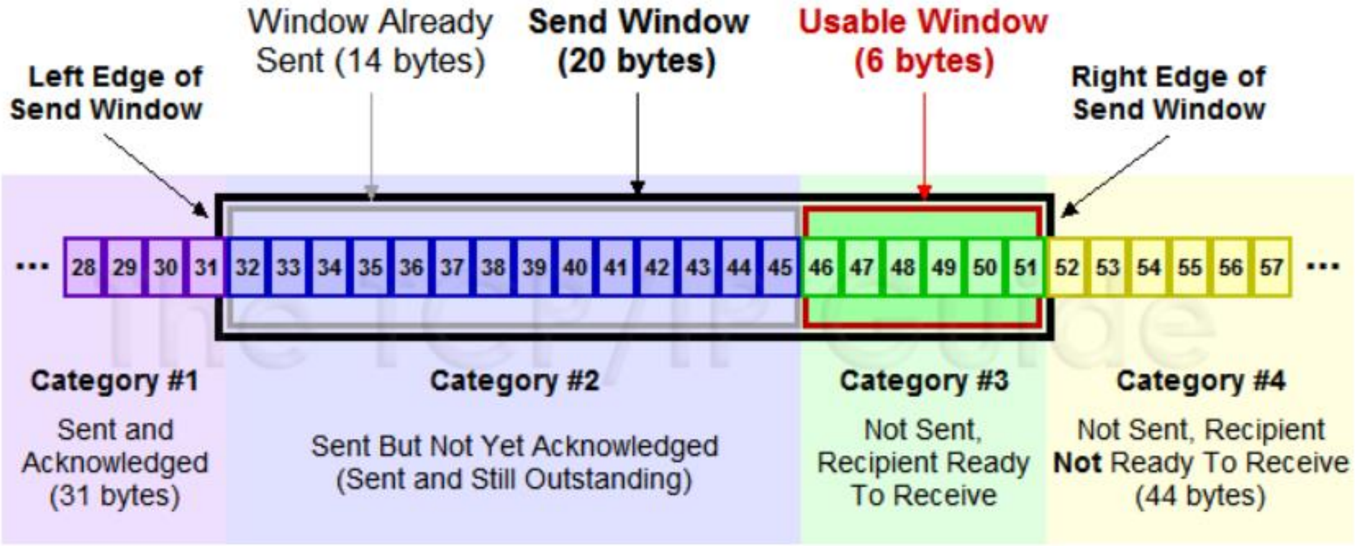
Dept. Computer Science and Engineering

Southern University of Science and Technology

SUSTech
Southern University
of Science and Technology

Thanks to Wei WANG & HHQ ZHANG

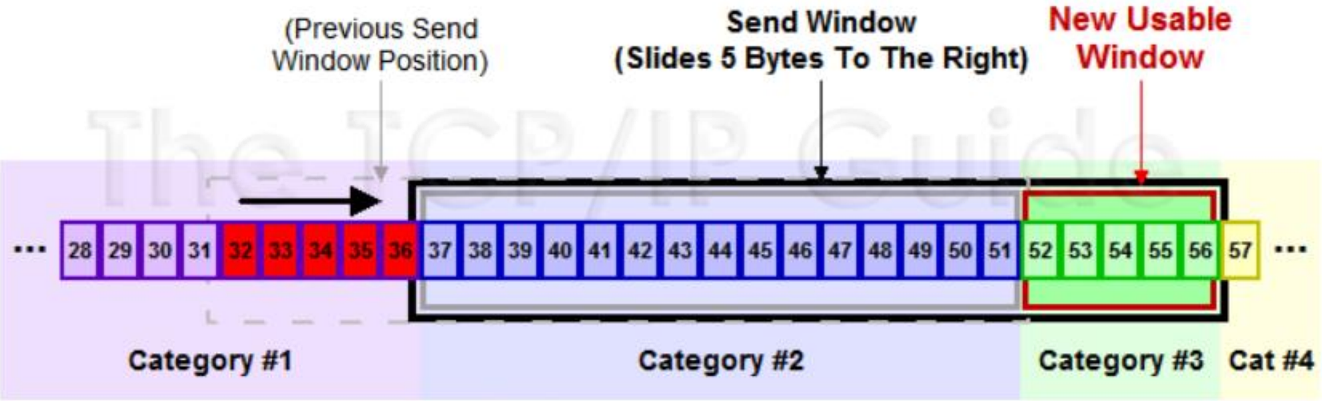
Part A.1 Sliding window system(1)


Left Edge of Send Window
Window Already Sent (14 bytes)
Send Window (20 bytes)
Usable Window (6 bytes)
Right Edge of Send Window
... 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 ...
Category #1
Sent and Acknowledged (31 bytes)
Category #2
Sent But Not Yet Acknowledged (Sent and Still Outstanding)
Category #3
Not Sent, Recipient Ready To Receive
Category #4
Not Sent, Recipient Not Ready To Receive (44 bytes)
The TCP/IP Guide

The ***send window*** is the key to the entire TCP sliding window system: it represents the maximum number of unacknowledged bytes a device is allowed to have outstanding at once.
The **usable window** is the number of bytes that the sender is still allowed to send at any point in time; it is equal to the size of the send window less the number of unacknowledged bytes already transmitted.

http://www.TCPipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-6.htm

Sliding window system(2)



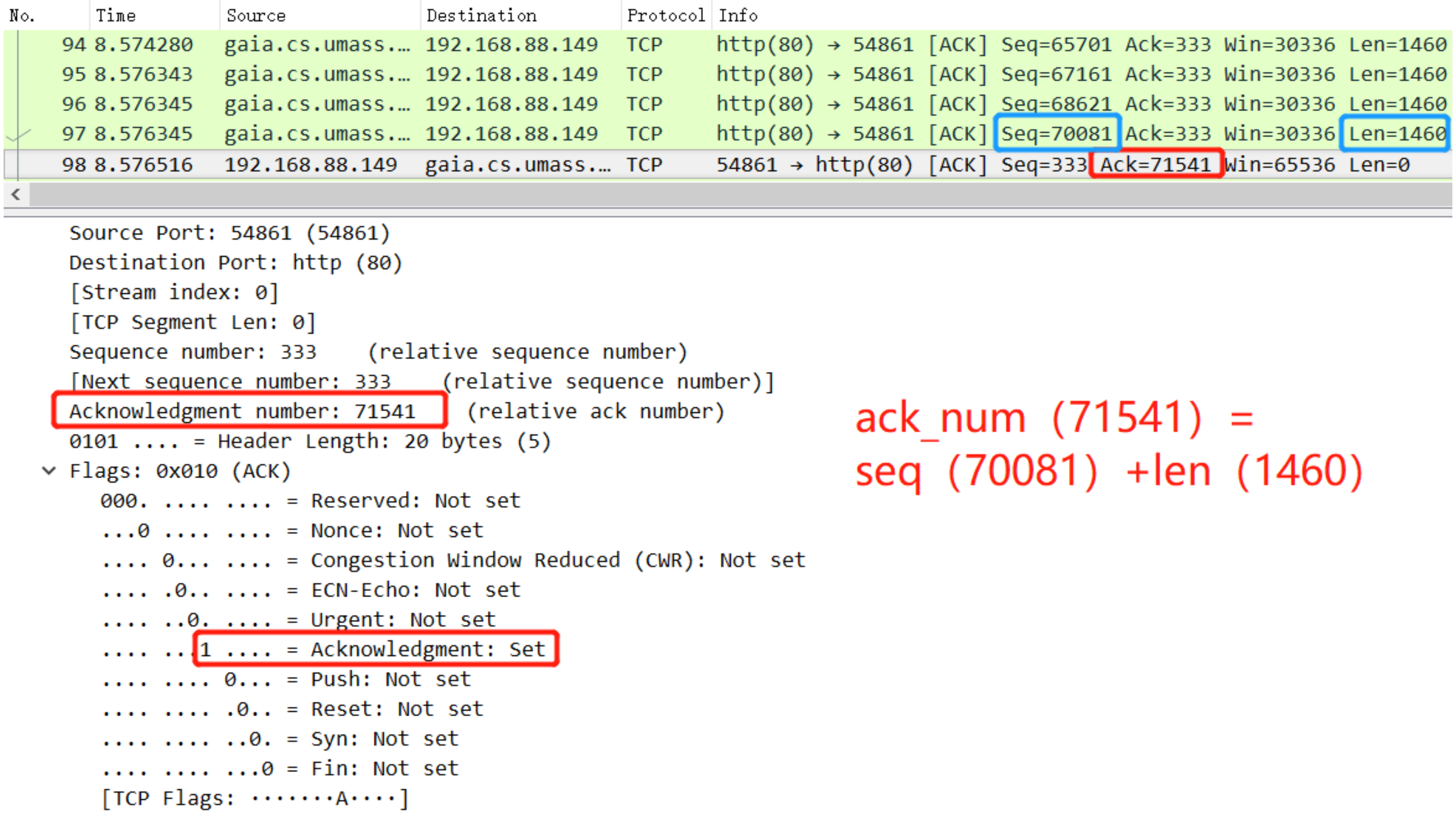
When the sending device **receives new acknowledgment**, it will be able to transfer some of the bytes from Category #2 to Category #1, since they have now been acknowledged. When it does so, something interesting will happen. Since five bytes have been acknowledged, and the window size didn't change, the sender is allowed to send five more bytes. In effect, the window shifts, or ***slides***, over to the right in the timeline.

At the same time five bytes move from Category #2 to Category #1, five bytes move from Category #4 to Category #3, **creating a new usable window for subsequent transmission**.

SUSTech Southern University of Science and Technology

http://www.TCPipguide.com/free/t_TCPSlidingWindowAcknowledgmentSystemForDataTranspo-6.htm

ACK number, Sequence number, len



```
No.   Time       Source           Destination      Protocol Info
   94 8.574280   gaia.cs.umass.…  192.168.88.149   TCP      http(80) → 54861 [ACK] Seq=65701 Ack=333 Win=30336 Len=1460
   95 8.576343   gaia.cs.umass.…  192.168.88.149   TCP      http(80) → 54861 [ACK] Seq=67161 Ack=333 Win=30336 Len=1460
   96 8.576345   gaia.cs.umass.…  192.168.88.149   TCP      http(80) → 54861 [ACK] Seq=68621 Ack=333 Win=30336 Len=1460
   97 8.576345   gaia.cs.umass.…  192.168.88.149   TCP      http(80) → 54861 [ACK] Seq=70081 Ack=333 Win=30336 Len=1460
   98 8.576516   192.168.88.149   gaia.cs.umass.…  TCP      54861 → http(80) [ACK] Seq=333 Ack=71541 Win=65536 Len=0
```

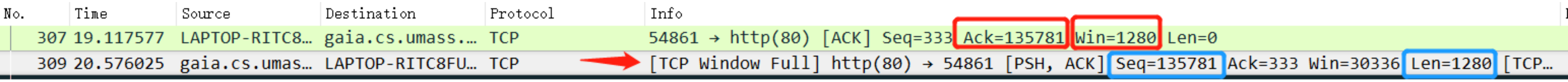
```
    Source Port: 54861 (54861)
    Destination Port: http (80)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 333    (relative sequence number)
    [Next sequence number: 333    (relative sequence number)]
    Acknowledgment number: 71541    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  ⌄ Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······A····]
```

ack_num (71541) = seq (70081) +len (1460)

Changes of window

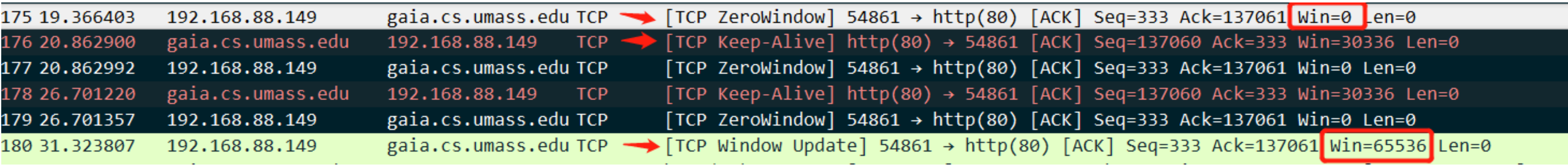
While the size of usable window turn to be 0, it means the sender will not send any segment at this moment. Wireshark mark the segment with “[Tcp Window Full]”

Seq（135781）+ len（1280） – ack（135781） == win（1280）



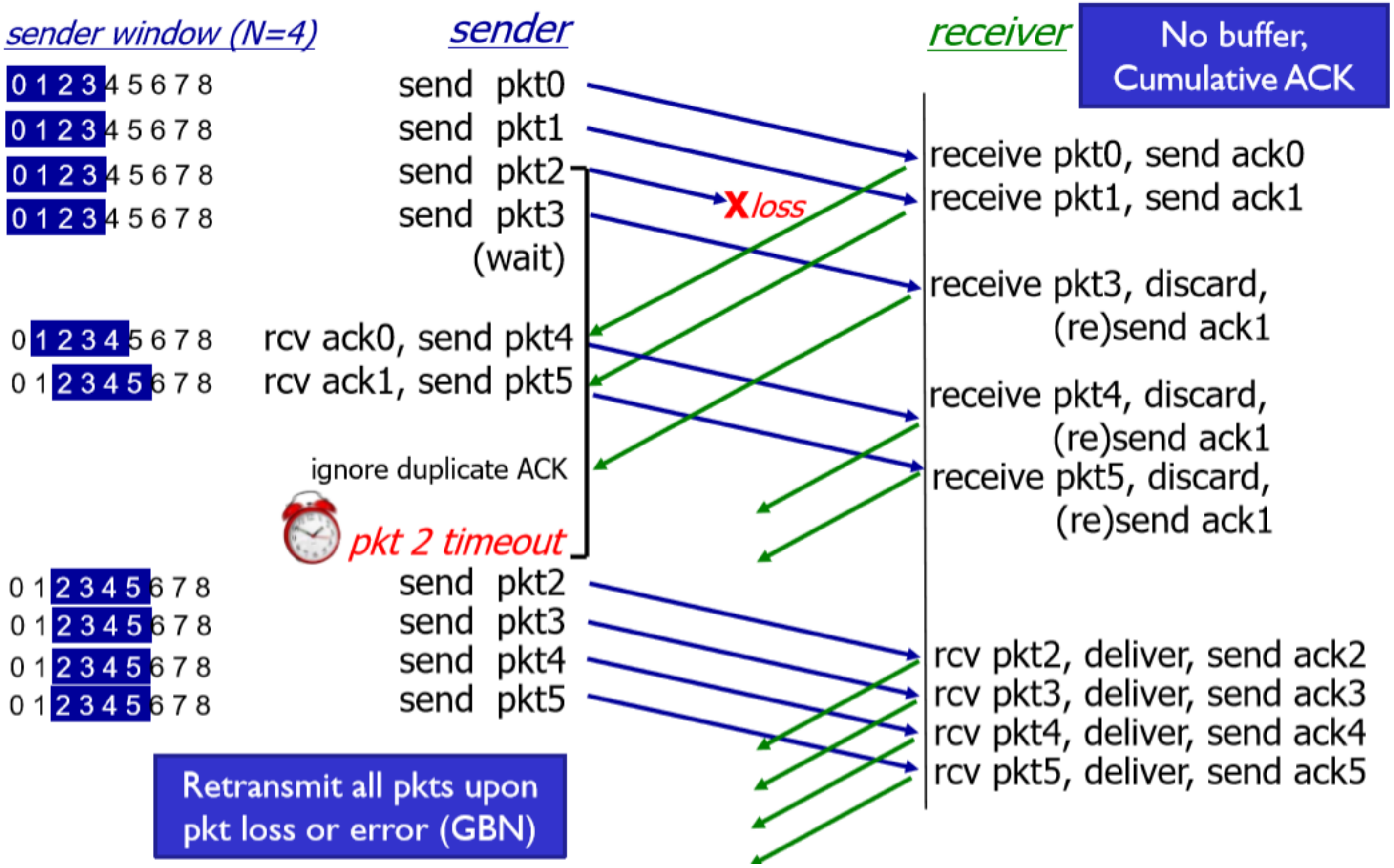
```
No.  Time       Source          Destination       Protocol  Info
307 19.117577  LAPTOP-RITC8…   gaia.cs.umass.…   TCP       54861 → http(80) [ACK] Seq=333 Ack=135781 Win=1280 Len=0
309 20.576025  gaia.cs.umas…   LAPTOP-RITC8FU…   TCP       [TCP Window Full] http(80) → 54861 [PSH, ACK] Seq=135781 Ack=333 Win=30336 Len=1280 [TCP…
```

While the recv window turn to be zero, sender will stop to send packet, it will send “[TCP Keep-Alive]” to keep the TCP connection, waiting for the changing of recv window.

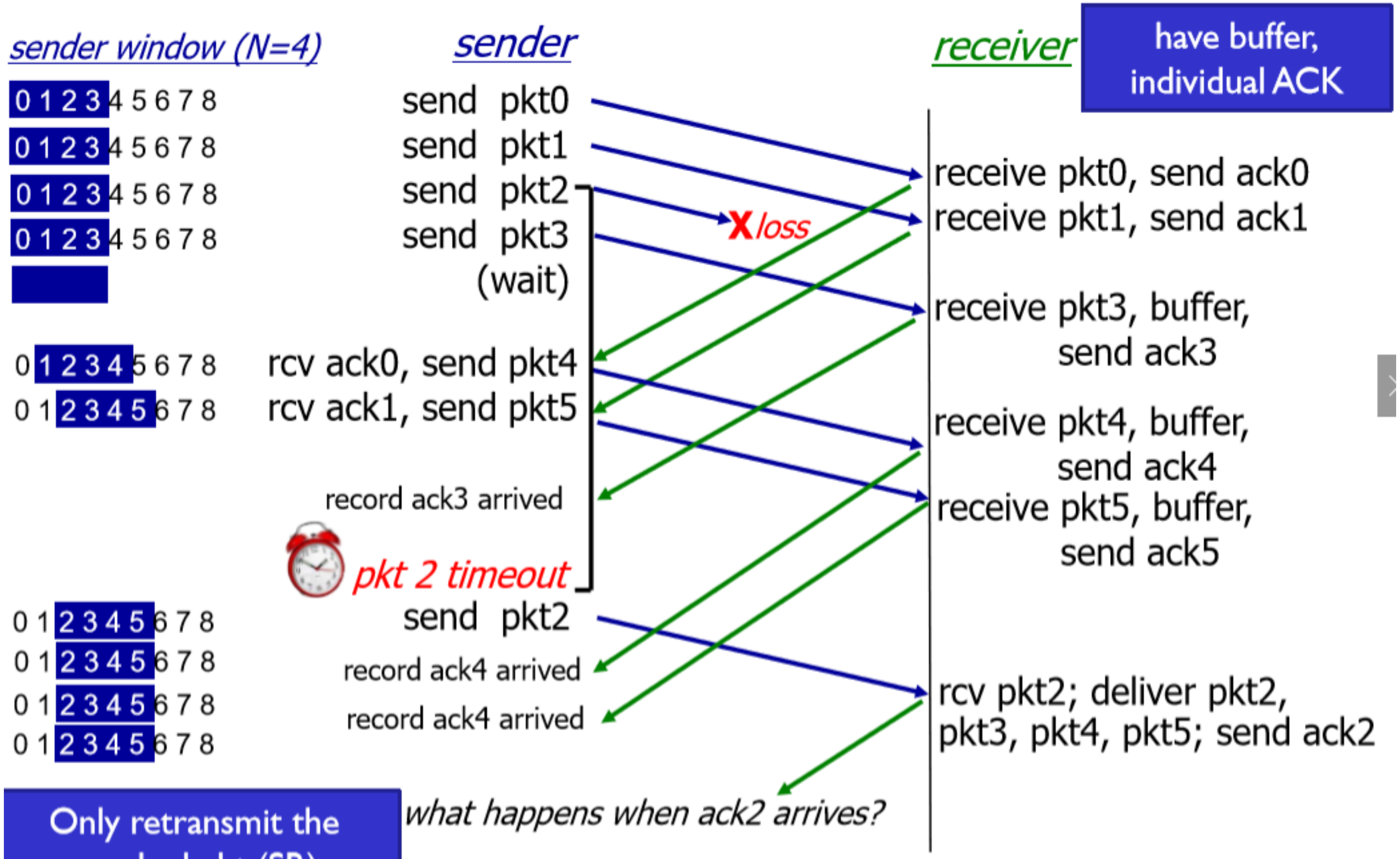


```
175 19.366403  192.168.88.149     gaia.cs.umass.edu TCP  [TCP ZeroWindow] 54861 → http(80) [ACK] Seq=333 Ack=137061 Win=0 Len=0
176 20.862900  gaia.cs.umass.edu  192.168.88.149    TCP  [TCP Keep-Alive] http(80) → 54861 [ACK] Seq=137060 Ack=333 Win=30336 Len=0
177 20.862992  192.168.88.149     gaia.cs.umass.edu TCP  [TCP ZeroWindow] 54861 → http(80) [ACK] Seq=333 Ack=137061 Win=0 Len=0
178 26.701220  gaia.cs.umass.edu  192.168.88.149    TCP  [TCP Keep-Alive] http(80) → 54861 [ACK] Seq=137060 Ack=333 Win=30336 Len=0
179 26.701357  192.168.88.149     gaia.cs.umass.edu TCP  [TCP ZeroWindow] 54861 → http(80) [ACK] Seq=333 Ack=137061 Win=0 Len=0
180 31.323807  192.168.88.149     gaia.cs.umass.edu TCP  [TCP Window Update] 54861 → http(80) [ACK] Seq=333 Ack=137061 Win=65536 Len=0
```

Part A.2 retransimission : GBN



SR



sender window (N=4)

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

X loss

rcv ack0, send pkt4
rcv ack1, send pkt5

record ack3 arrived

pkt 2 timeout
send pkt2
record ack4 arrived
record ack4 arrived

receiver

have buffer, individual ACK

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4
receive pkt5, buffer,
send ack5

rcv pkt2; deliver pkt2,
pkt3, pkt4, pkt5; send ack2

what happens when ack2 arrives?

Only retransmit the unacked pkt (SR)

TCP SACK

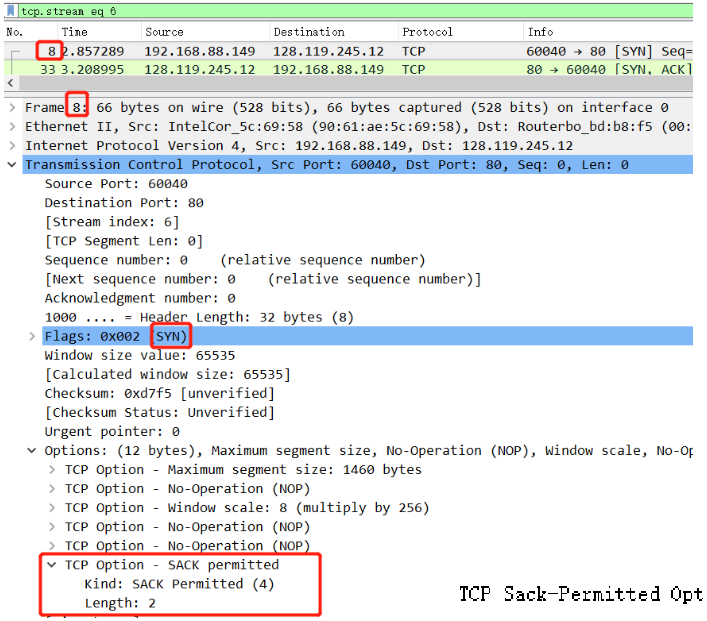
- A Selective Acknowledgment (**SACK**) mechanism, combined with a **selective repeat retransmission policy**, can help the sender retransmit only the missing data segments.
  - The receiving TCP **sends back SACK packets to the sender** informing the sender of data that has been received.
  - With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need **retransmit only the segments that have actually been lost.**

- The selective acknowledgment extension uses two TCP options:
  - **SACK-permitted** option
  - **SACK** option

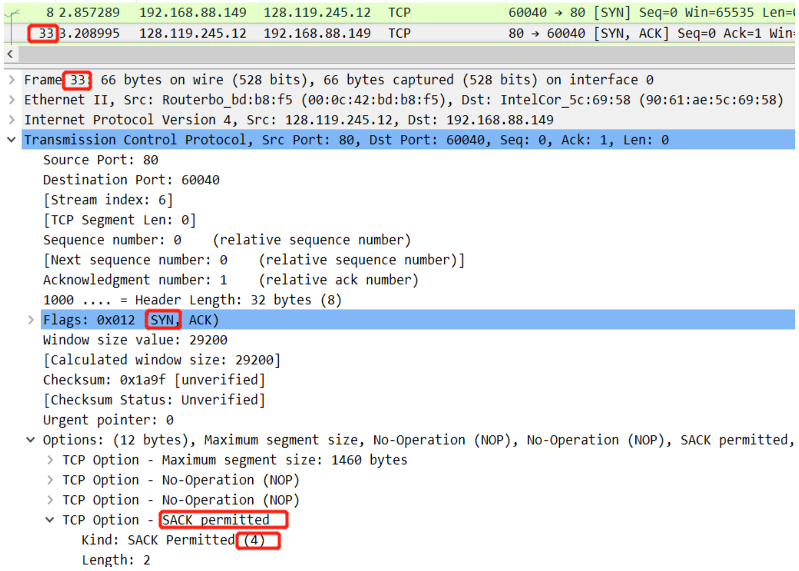
https://tools.ietf.org/html/rfc2018

SACK-permitted option

- "**SACK-permitted**", which may **be sent in a SYN segment** to indicate that the SACK option can be used once the connection is established.



```
tcp.stream eq 6
No.    Time      Source          Destination     Protocol   Info
   8 2.857289   192.168.88.149  128.119.245.12  TCP        60040 → 80 [SYN] Seq=
  33 3.208995   128.119.245.12  192.168.88.149  TCP        80 → 60040 [SYN, ACK]

> Frame 8: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: IntelCor_5c:69:58 (90:61:ae:5c:69:58), Dst: Routerbo_bd:b8:f5 (00:
> Internet Protocol Version 4, Src: 192.168.88.149, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 60040, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 60040
    Destination Port: 80
    [Stream index: 6]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0xd7f5 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Op
    > TCP Option - Maximum segment size: 1460 bytes
    > TCP Option - No-Operation (NOP)
    > TCP Option - Window scale: 8 (multiply by 256)
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    v TCP Option - SACK permitted
        Kind: SACK Permitted (4)
        Length: 2
```



```
   8 2.857289   192.168.88.149  128.119.245.12  TCP        60040 → 80 [SYN] Seq=0 Win=65535 Len=
  33 3.208995   128.119.245.12  192.168.88.149  TCP        80 → 60040 [SYN, ACK] Seq=0 Ack=1 Win=

> Frame 33: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: Routerbo_bd:b8:f5 (00:0c:42:bd:b8:f5), Dst: IntelCor_5c:69:58 (90:61:ae:5c:69:58)
> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.88.149
v Transmission Control Protocol, Src Port: 80, Dst Port: 60040, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 60040
    [Stream index: 6]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0x1a9f [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted,
    > TCP Option - Maximum segment size: 1460 bytes
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    v TCP Option - SACK permitted
        Kind: SACK Permitted (4)
        Length: 2
```

```
TCP Sack-Permitted Option:

Kind: 4

+---------+---------+
| Kind=4  | Length=2|
+---------+---------+
```

Wireshark tips: TCP.option_kind==4

SACK option(1)

The **SACK option** is to be **sent by a data receiver to inform the data sender of non-contiguous blocks of data that have been received and queued**.

The data receiver awaits the receipt of data (perhaps by means of retransmissions) to fill the gaps in sequence space between received blocks.

When missing segments are received, **the data receiver acknowledges the data normally** by advancing the left window edge in the Acknowledgement Number Field of the TCP header.

The SACK option does not change the meaning of the Acknowledgement Number field.

This option contains a list of **some of the blocks of contiguous sequence space occupied by data that has been received and queued within the window.**

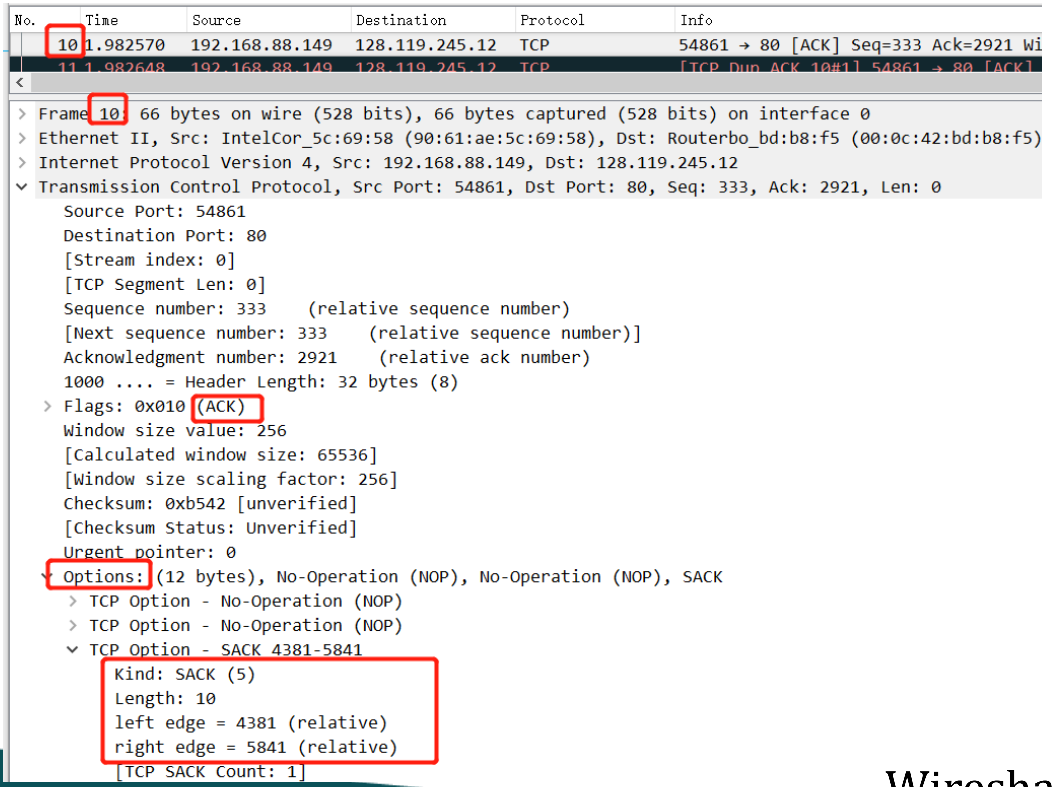
Each contiguous block of data queued at the data receiver is defined in the SACK option by **two 32-bit unsigned integers** in network byte order:

- **Left Edge** of Block This is the first sequence number of this block.
- **Right Edge** of Block This is the sequence number immediately following the last sequence number of this block.

Each block represents received bytes of data that are **contiguous and isolated**; that is, **the bytes just below the block, (Left Edge of Block - 1), and just above the block, (Right Edge of Block), have NOT been received**.

SACK option(2)

- SACK option , which may be sent over an established connection once permission has been given by SACK-permitted.

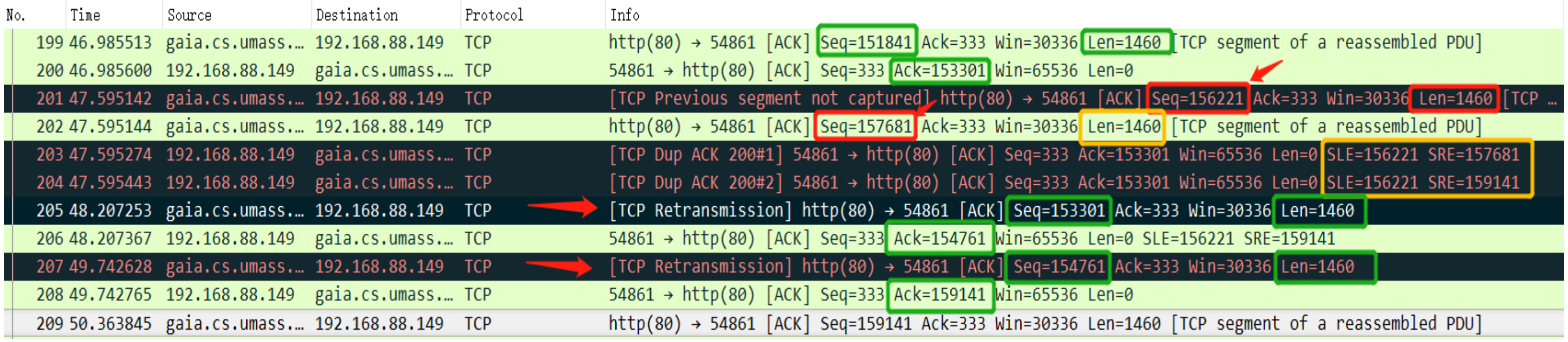


```
No.   Time        Source           Destination      Protocol   Info
   10 1.982570    192.168.88.149   128.119.245.12   TCP        54861 → 80 [ACK] Seq=333 Ack=2921 Wi
   11 1.982648    192.168.88.149   128.119.245.12   TCP        [TCP Dup ACK 10#1] 54861 → 80 [ACK]

> Frame 10: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: IntelCor_5c:69:58 (90:61:ae:5c:69:58), Dst: Routerbo_bd:b8:f5 (00:0c:42:bd:b8:f5)
> Internet Protocol Version 4, Src: 192.168.88.149, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 54861, Dst Port: 80, Seq: 333, Ack: 2921, Len: 0
    Source Port: 54861
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 333    (relative sequence number)
    [Next sequence number: 333    (relative sequence number)]
    Acknowledgment number: 2921    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window size value: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0xb542 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), SACK
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    v TCP Option - SACK 4381-5841
        Kind: SACK (5)
        Length: 10
        left edge = 4381 (relative)
        right edge = 5841 (relative)
        [TCP SACK Count: 1]
```

```
TCP SACK Option:

Kind: 5

Length: Variable

                  +--------+--------+
                  | Kind=5 | Length |
+--------+--------+--------+--------+
|      Left Edge of 1st Block       |
+--------+--------+--------+--------+
|      Right Edge of 1st Block      |
+--------+--------+--------+--------+
|                                   |
/            . . .                  /
|                                   |
+--------+--------+--------+--------+
|      Left Edge of nth Block       |
+--------+--------+--------+--------+
|      Right Edge of nth Block      |
+--------+--------+--------+--------+
```

Wireshark tips: TCP.option_kind==5

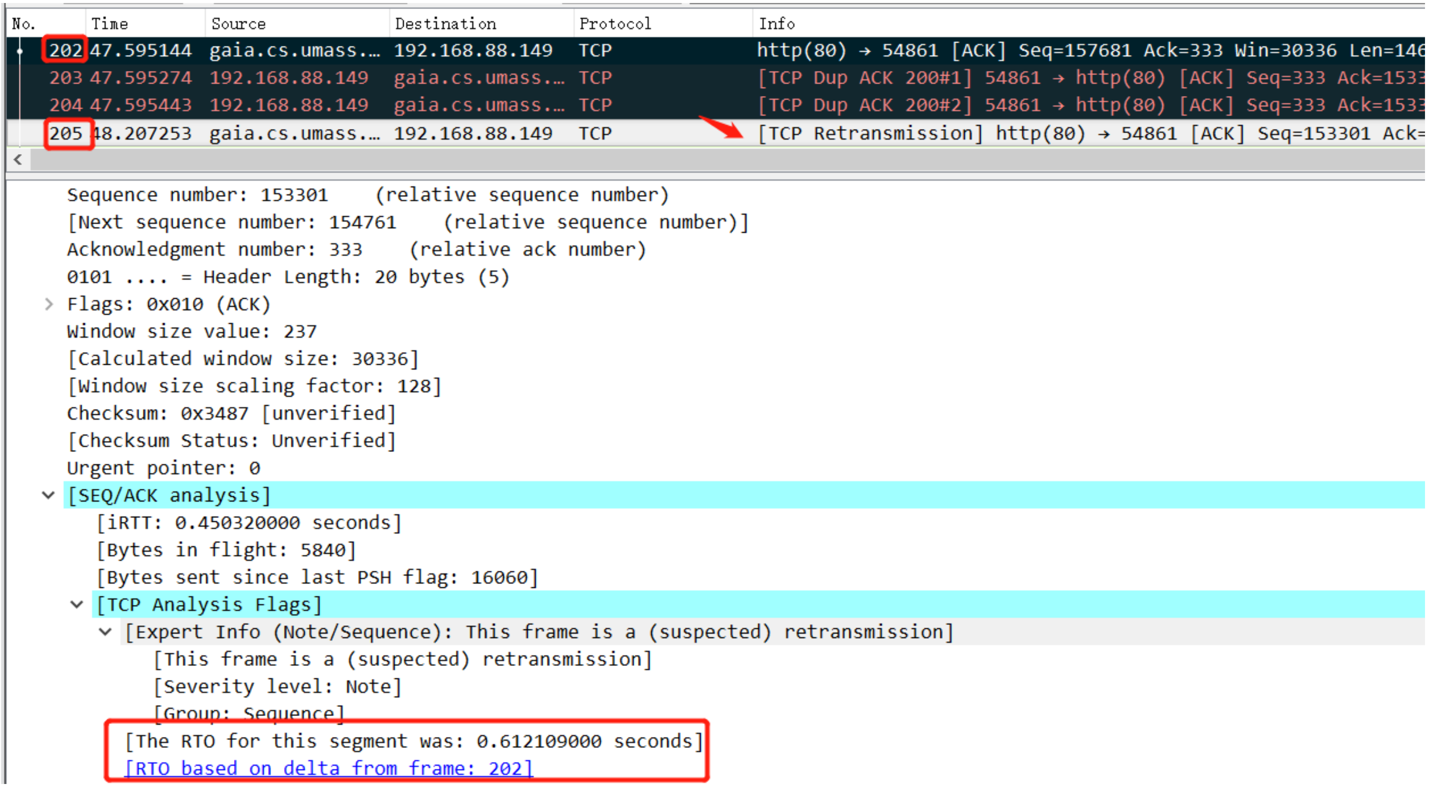
SACK option(3)



```
No.  Time       Source          Destination     Protocol  Info
199 46.985513  gaia.cs.umass…  192.168.88.149  TCP       http(80) → 54861 [ACK] Seq=151841 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
200 46.985600  192.168.88.149  gaia.cs.umass…  TCP       54861 → http(80) [ACK] Seq=333 Ack=153301 Win=65536 Len=0
201 47.595142  gaia.cs.umass…  192.168.88.149  TCP       [TCP Previous segment not captured] http(80) → 54861 [ACK] Seq=156221 Ack=333 Win=30336 Len=1460 [TCP …
202 47.595144  gaia.cs.umass…  192.168.88.149  TCP       http(80) → 54861 [ACK] Seq=157681 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
203 47.595274  192.168.88.149  gaia.cs.umass…  TCP       [TCP Dup ACK 200#1] 54861 → http(80) [ACK] Seq=333 Ack=153301 Win=65536 Len=0 SLE=156221 SRE=157681
204 47.595443  192.168.88.149  gaia.cs.umass…  TCP       [TCP Dup ACK 200#2] 54861 → http(80) [ACK] Seq=333 Ack=153301 Win=65536 Len=0 SLE=156221 SRE=159141
205 48.207253  gaia.cs.umass…  192.168.88.149  TCP       [TCP Retransmission] http(80) → 54861 [ACK] Seq=153301 Ack=333 Win=30336 Len=1460
206 48.207367  192.168.88.149  gaia.cs.umass…  TCP       54861 → http(80) [ACK] Seq=333 Ack=154761 Win=65536 Len=0 SLE=156221 SRE=159141
207 49.742628  gaia.cs.umass…  192.168.88.149  TCP       [TCP Retransmission] http(80) → 54861 [ACK] Seq=154761 Ack=333 Win=30336 Len=1460
208 49.742765  192.168.88.149  gaia.cs.umass…  TCP       54861 → http(80) [ACK] Seq=333 Ack=159141 Win=65536 Len=0
209 50.363845  gaia.cs.umass…  192.168.88.149  TCP       http(80) → 54861 [ACK] Seq=159141 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
```

#203 and #204 are SACK
#203 tells that 156221~157681 are **contiguous and isolated**
#204 tells that 156221~159141 are **contiguous and isolated**
#200 tells that the block before 153301 are acked
So
#205 retransmit 153301 ~ 153301+ 1460 -1 ,#206 ack it with 154761
#207 retransmit 154761 ~ 154761+1460 -1
#208 ack it with 159141 (for 157681~159140 are **contiguous but NOT isolated**)

Retransmission(1)

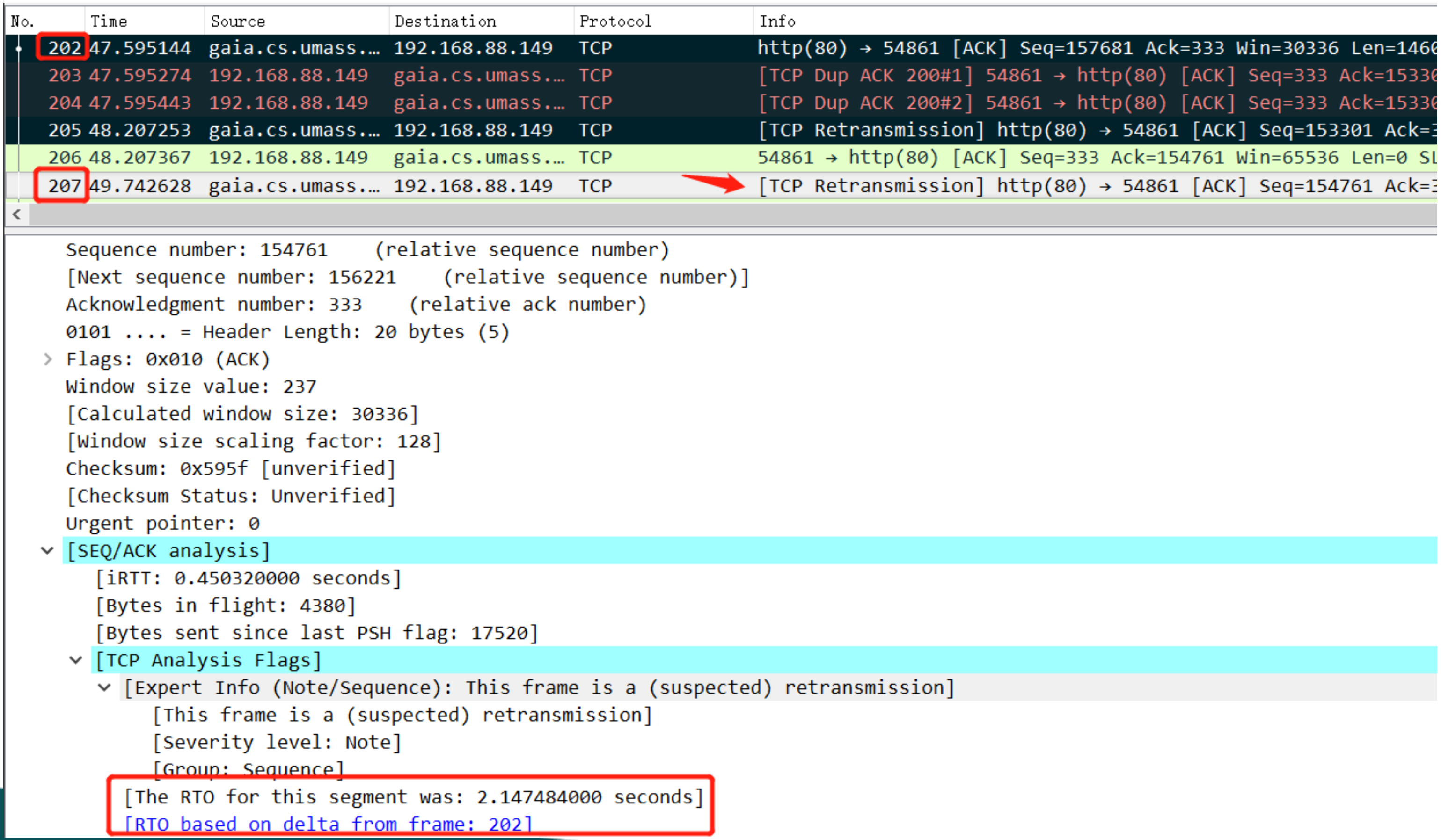


```
No.  Time       Source          Destination     Protocol  Info
202 47.595144 gaia.cs.umass.… 192.168.88.149  TCP       http(80) → 54861 [ACK] Seq=157681 Ack=333 Win=30336 Len=146
203 47.595274 192.168.88.149  gaia.cs.umass.… TCP       [TCP Dup ACK 200#1] 54861 → http(80) [ACK] Seq=333 Ack=1533
204 47.595443 192.168.88.149  gaia.cs.umass.… TCP       [TCP Dup ACK 200#2] 54861 → http(80) [ACK] Seq=333 Ack=1533
205 48.207253 gaia.cs.umass.… 192.168.88.149  TCP       [TCP Retransmission] http(80) → 54861 [ACK] Seq=153301 Ack=
```

```
    Sequence number: 153301    (relative sequence number)
    [Next sequence number: 154761    (relative sequence number)]
    Acknowledgment number: 333    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 30336]
    [Window size scaling factor: 128]
    Checksum: 0x3487 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  v [SEQ/ACK analysis]
      [iRTT: 0.450320000 seconds]
      [Bytes in flight: 5840]
      [Bytes sent since last PSH flag: 16060]
    v [TCP Analysis Flags]
      v [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
          [This frame is a (suspected) retransmission]
          [Severity level: Note]
          [Group: Sequence]
        [The RTO for this segment was: 0.612109000 seconds]
        [RTO based on delta from frame: 202]
```

While RTO timeout , retransmission is triggered

Retransmission(2)

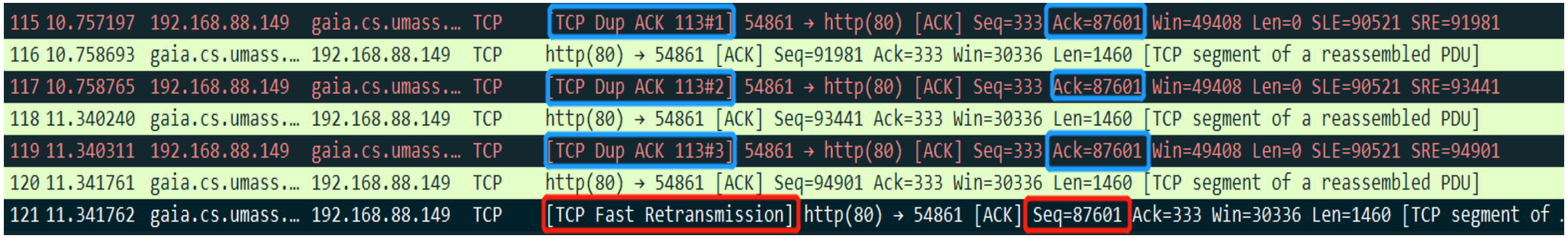


```
No.  Time       Source           Destination      Protocol  Info
202 47.595144  gaia.cs.umass…  192.168.88.149   TCP       http(80) → 54861 [ACK] Seq=157681 Ack=333 Win=30336 Len=146
203 47.595274  192.168.88.149  gaia.cs.umass…   TCP       [TCP Dup ACK 200#1] 54861 → http(80) [ACK] Seq=333 Ack=1533
204 47.595443  192.168.88.149  gaia.cs.umass…   TCP       [TCP Dup ACK 200#2] 54861 → http(80) [ACK] Seq=333 Ack=1533
205 48.207253  gaia.cs.umass…  192.168.88.149   TCP       [TCP Retransmission] http(80) → 54861 [ACK] Seq=153301 Ack=
206 48.207367  192.168.88.149  gaia.cs.umass…   TCP       54861 → http(80) [ACK] Seq=333 Ack=154761 Win=65536 Len=0 SL
207 49.742628  gaia.cs.umass…  192.168.88.149   TCP       [TCP Retransmission] http(80) → 54861 [ACK] Seq=154761 Ack=
```

```
Sequence number: 154761    (relative sequence number)
[Next sequence number: 156221    (relative sequence number)]
Acknowledgment number: 333    (relative ack number)
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
Window size value: 237
[Calculated window size: 30336]
[Window size scaling factor: 128]
Checksum: 0x595f [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]
   [iRTT: 0.450320000 seconds]
   [Bytes in flight: 4380]
   [Bytes sent since last PSH flag: 17520]
   [TCP Analysis Flags]
      [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
         [This frame is a (suspected) retransmission]
         [Severity level: Note]
         [Group: Sequence]
      [The RTO for this segment was: 2.147484000 seconds]
      [RTO based on delta from frame: 202]
```

Fast retransmission

- TCP may generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received. This duplicate ACK should not be delayed. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected.
- Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received.
  - It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK.
  - If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost.
- TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.



```
115 10.757197  192.168.88.149  gaia.cs.umass…  TCP   [TCP Dup ACK 113#1] 54861 → http(80) [ACK] Seq=333 Ack=87601 Win=49408 Len=0 SLE=90521 SRE=91981
116 10.758693  gaia.cs.umass…  192.168.88.149  TCP   http(80) → 54861 [ACK] Seq=91981 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
117 10.758765  192.168.88.149  gaia.cs.umass…  TCP   [TCP Dup ACK 113#2] 54861 → http(80) [ACK] Seq=333 Ack=87601 Win=49408 Len=0 SLE=90521 SRE=93441
118 11.340240  gaia.cs.umass…  192.168.88.149  TCP   http(80) → 54861 [ACK] Seq=93441 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
119 11.340311  192.168.88.149  gaia.cs.umass…  TCP   [TCP Dup ACK 113#3] 54861 → http(80) [ACK] Seq=333 Ack=87601 Win=49408 Len=0 SLE=90521 SRE=94901
120 11.341761  gaia.cs.umass…  192.168.88.149  TCP   http(80) → 54861 [ACK] Seq=94901 Ack=333 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
121 11.341762  gaia.cs.umass…  192.168.88.149  TCP   [TCP Fast Retransmission] http(80) → 54861 [ACK] Seq=87601 Ack=333 Win=30336 Len=1460 [TCP segment of .
```

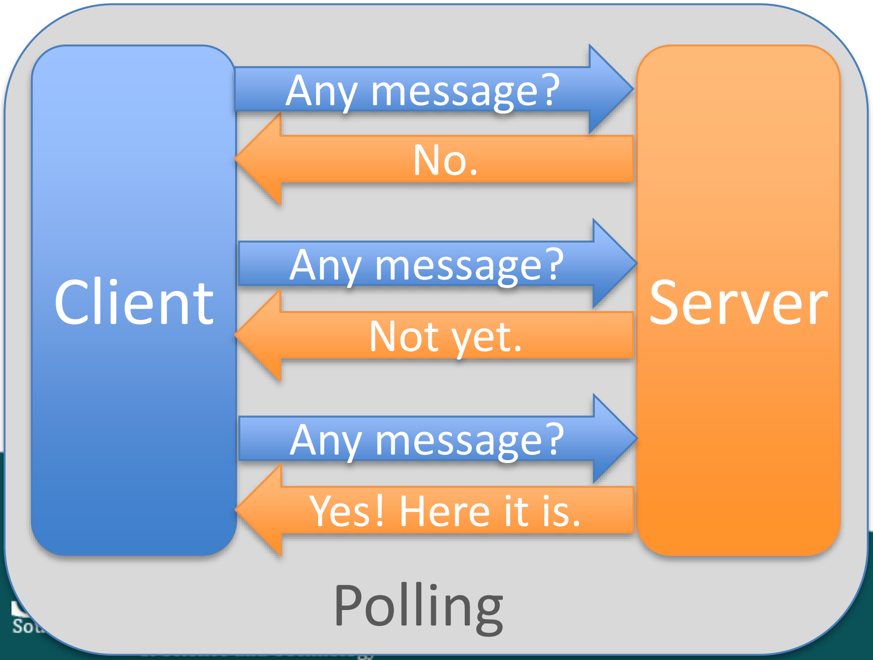
https://tools.ietf.org/html/rfc2001

Part B. WebSocket

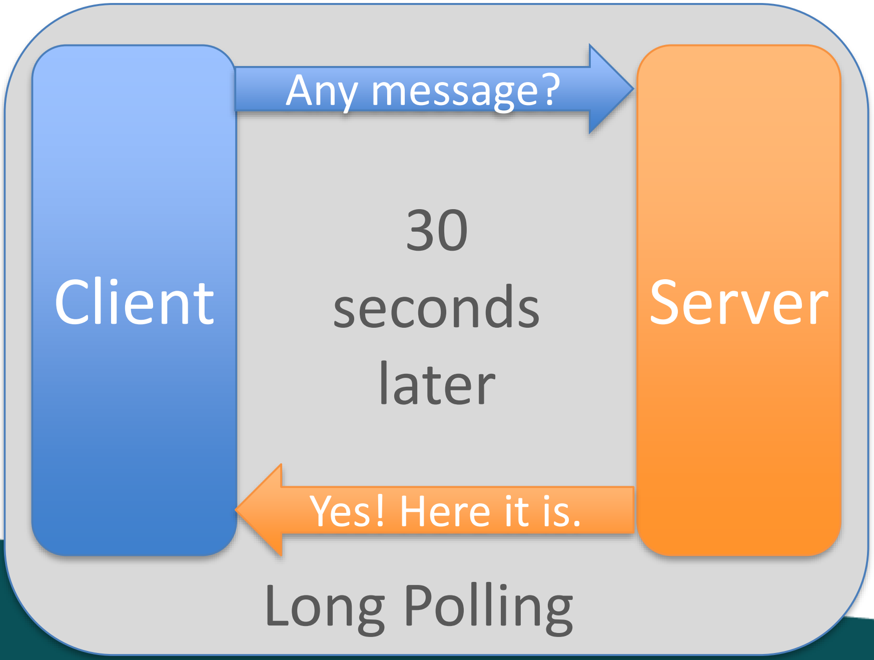
- The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication).
- The WebSocket Protocol attempts to address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure.
  - support HTTP proxies and intermediaries
  - does not limit WebSocket to HTTP, and future implementations could use a simpler handshake over a dedicated port without reinventing the entire protocol.

HTTP polls for new messages

- Historically, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls .



Polling



Long Polling

WebSocket protocol overview

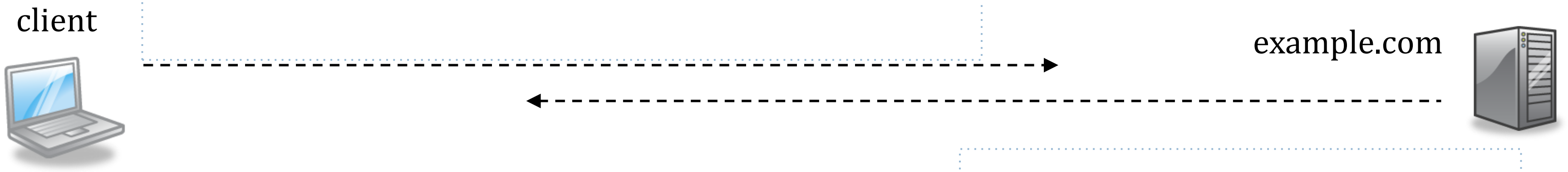
- Two parts
  - Handshakes: Opening Handshake & Closing Handshake
  - Data transfer
- The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.
  - Uses port 80 for regular WebSocket connections
  - Uses port 443 for WebSocket connections tunneled over TLS
  - Can not establish a connection with servers of pre-existing protocols like SMTP and HTTP

URI

- A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource.
- ws-URI = "ws:" "//" host [":" port] path ["?" query]
  - ws://example.com/chat
- wss-URI = "wss:" "//" host [":" port] path ["?" query]

Opening handshake

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```



"dGhlIHNhbXBsZSBub25jZQ=="

Step1. concatenate "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"

Step2. take the SHA-1 hash

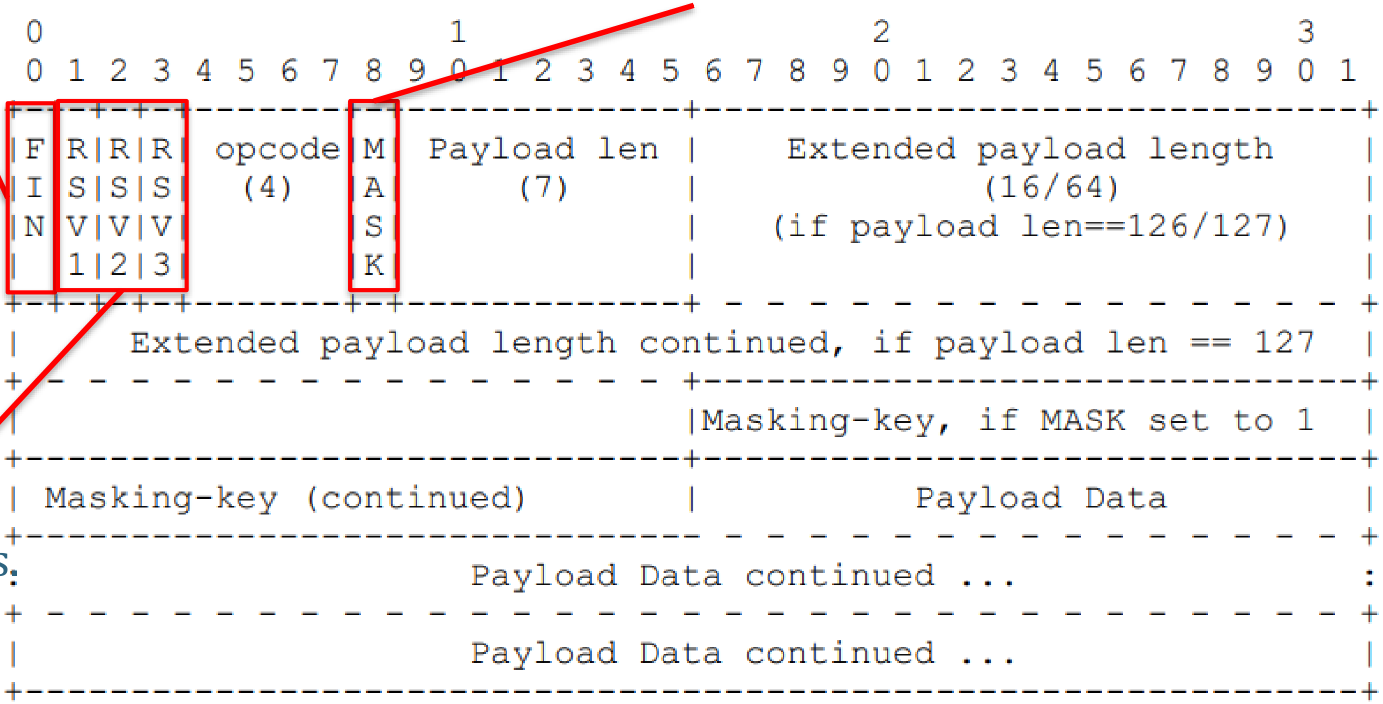
Step3. base64-encoded the hash value

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Data framing

Indicates that this is the final fragment in a message.

Defines whether the "Payload data" is masked.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|R|R|R| opcode|M| Payload len |    Extended payload length    |
|I|S|S|S|  (4)  |A|     (7)     |             (16/64)           |
|N|V|V|V|       |S|             |   (if payload len==126/127)   |
| |1|2|3|       |K|             |                               |
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
|     Extended payload length continued, if payload len == 127  |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               |Masking-key, if MASK set to 1  |
+-------------------------------+-------------------------------+
| Masking-key (continued)       |          Payload Data         |
+-------------------------------- - - - - - - - - - - - - - - - +
:                     Payload Data continued ...                :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                     Payload Data continued ...                |
+---------------------------------------------------------------+
```

MUST be 0 unless an extension is negotiated that defines meanings for non-zero values.

Data framing (continued)

- Opcode (4 bits)
  - %x0(0000) denotes a continuation frame
  - %x1(0001) denotes a text frame
  - %x2(0010) denotes a binary frame
  - %x3-7(0011-0111) are reserved for further non-control frames
  - %x8(1000) denotes a connection close
  - %x9(1001) denotes a ping
  - %xA(1010) denotes a pong
  - %xB-F(1011-1111) are reserved for further control frames

Data framing (continued)

- Payload len
  - 7 bits: if 0~125 bytes
  - 7 + 16 bit: if the 7 bits equals to 126
  - 7 + 64 bits: if the 7 bits equals to 127
  - payload length = the length of the "Extension data" + the length of the "Application data"
- Masking key
  - 0 bits: if the mask bit is set to 1
  - 32 bits: if the mask bit is set to 0
  - The masking key is a 32-bit value chosen at random by the client.

Data framing examples

- A single-frame unmasked text message
  - 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains "Hello")
- A single-frame masked text message
  - 0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains "Hello")
- A fragmented unmasked text message
  - 0x01 0x03 0x48 0x65 0x6c (contains "Hel")
  - 0x80 0x02 0x6c 0x6f (contains "lo")
- Unmasked Ping request and masked Ping response
  - 0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains a body of "Hello", but the contents of the body are arbitrary)
  - 0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains a body of "Hello", matching the body of the ping)
- 256 bytes binary message in a single unmasked frame
  - 0x82 0x7E 0x0100 [256 bytes of binary data]

Closing the connection

- An endpoint MUST send a Close control frame(opcode = 1000)
- Connection Close Code
  - 1000: a normal closure.
  - 1001: an endpoint is "going away".
  - 1002: a protocol error occurs.
  - 1003: an endpoint has received a type of data it cannot accept.
  - 1004, 1005, 1006, 1015: Reserved.
  - 1007: an endpoint has received an inconsistent type of data.
  - 1008: an endpoint has received a message violates its policy.
  - 1009: an endpoint has received a message that is too big to process.
  - 1010: the client has expected the server to negotiate one or more extensions but received no response about that.
  - 1011: the server encountered an unexpected condition.

Example 1: Mimic a WebSocket Server

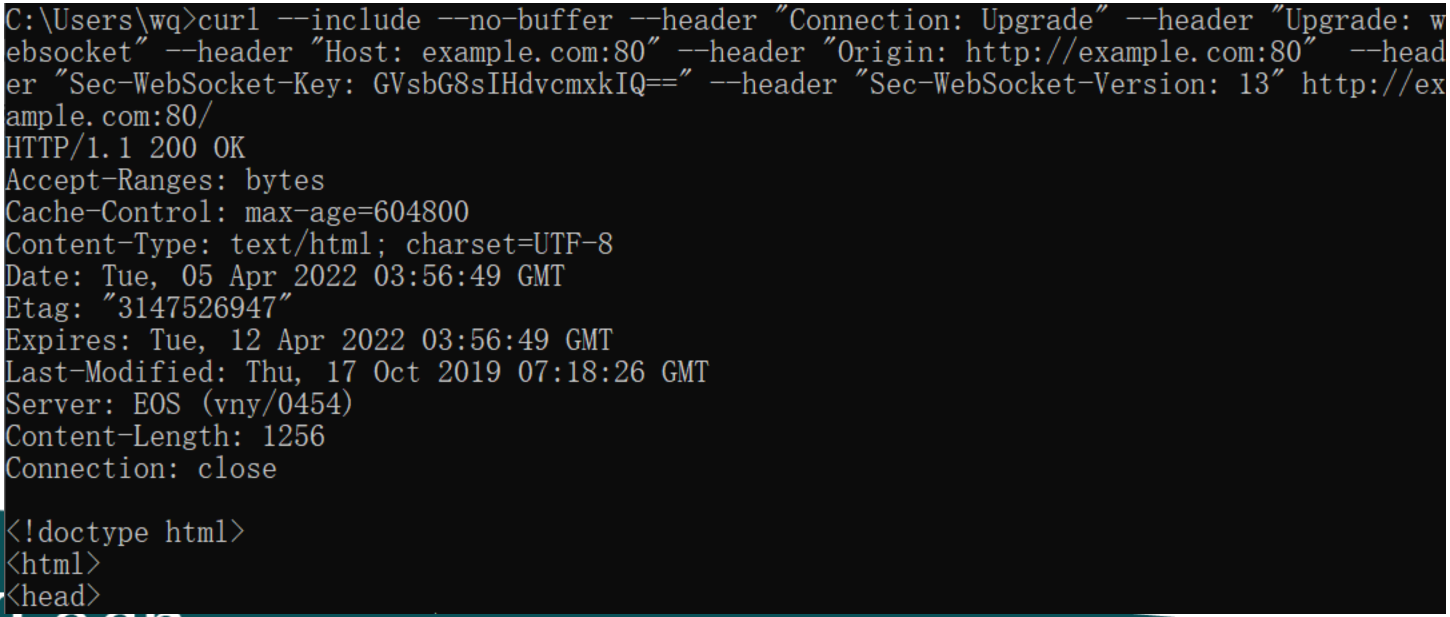
```
import asyncio
import websockets

async def echo(websocket, path):
    async for message in websocket:
        message = "I got your message: {}".format(message)
        await websocket.send(message)

asyncio.get_event_loop().run_until_complete(
    websockets.serve(echo, '127.0.0.1', 8766))
asyncio.get_event_loop().run_forever()
```

Example 2: Mimic a WebSocket Client

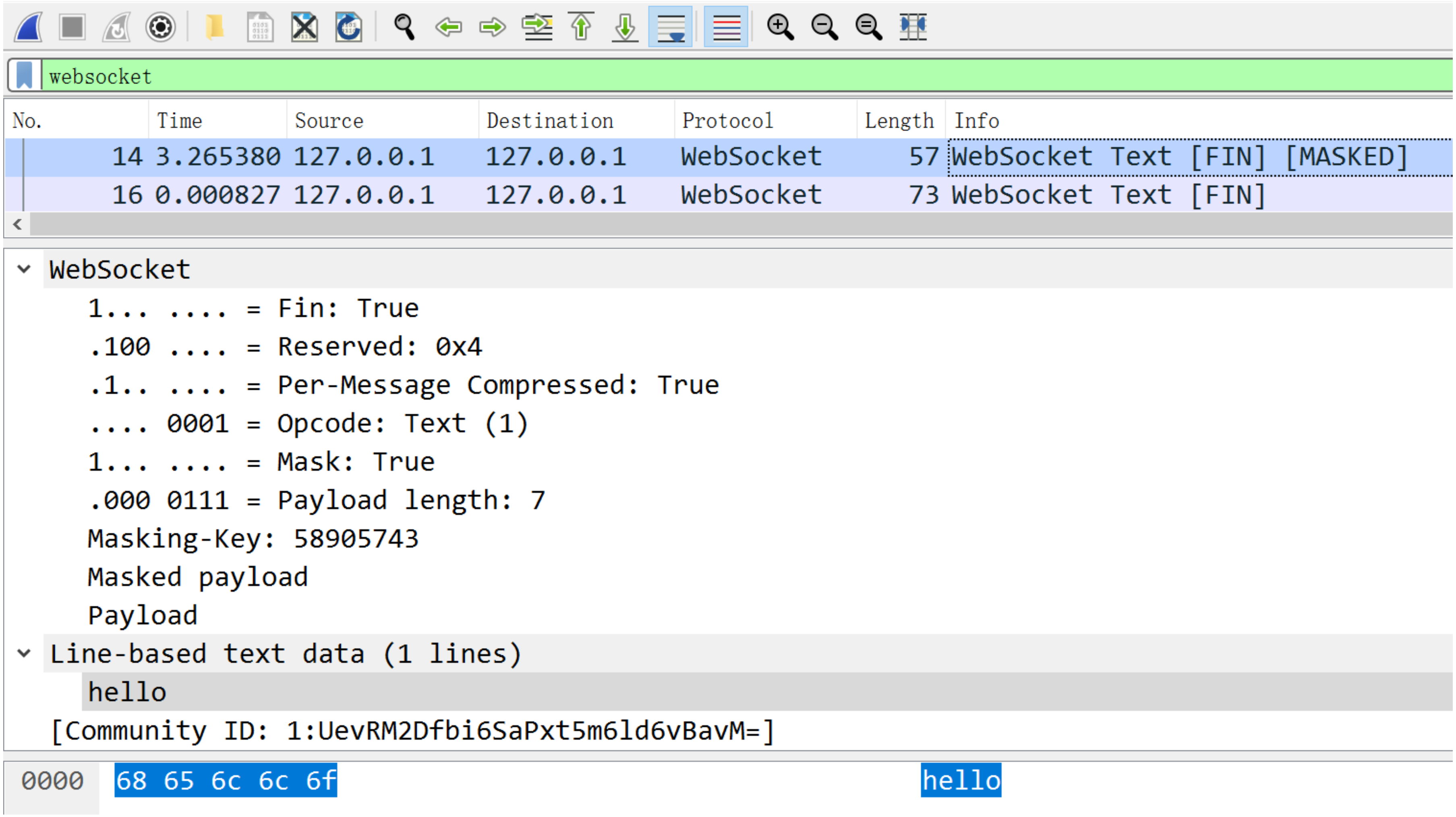
```
import asyncio
import websockets

async def echo(uri):
    async with websockets.connect(uri) as websocket:
        while True:
            message = input("Write down your message:")
            await websocket.send(message)
            print("<", message)
            recv_text = await websocket.recv()
            print("> {}".format(recv_text))

asyncio.get_event_loop().run_until_complete(
    echo('ws://127.0.0.1:8766'))
```

Example 3: Use curl

> curl --include --no-buffer --header "Connection: Upgrade" --header "Upgrade: websocket" --header "Host: example.com:80" --header "Origin: http://example.com:80" --header "Sec-WebSocket-Key: GVsbG8sIHdvcmxkIQ==" --header "Sec-WebSocket-Version: 13" http://example.com:80/



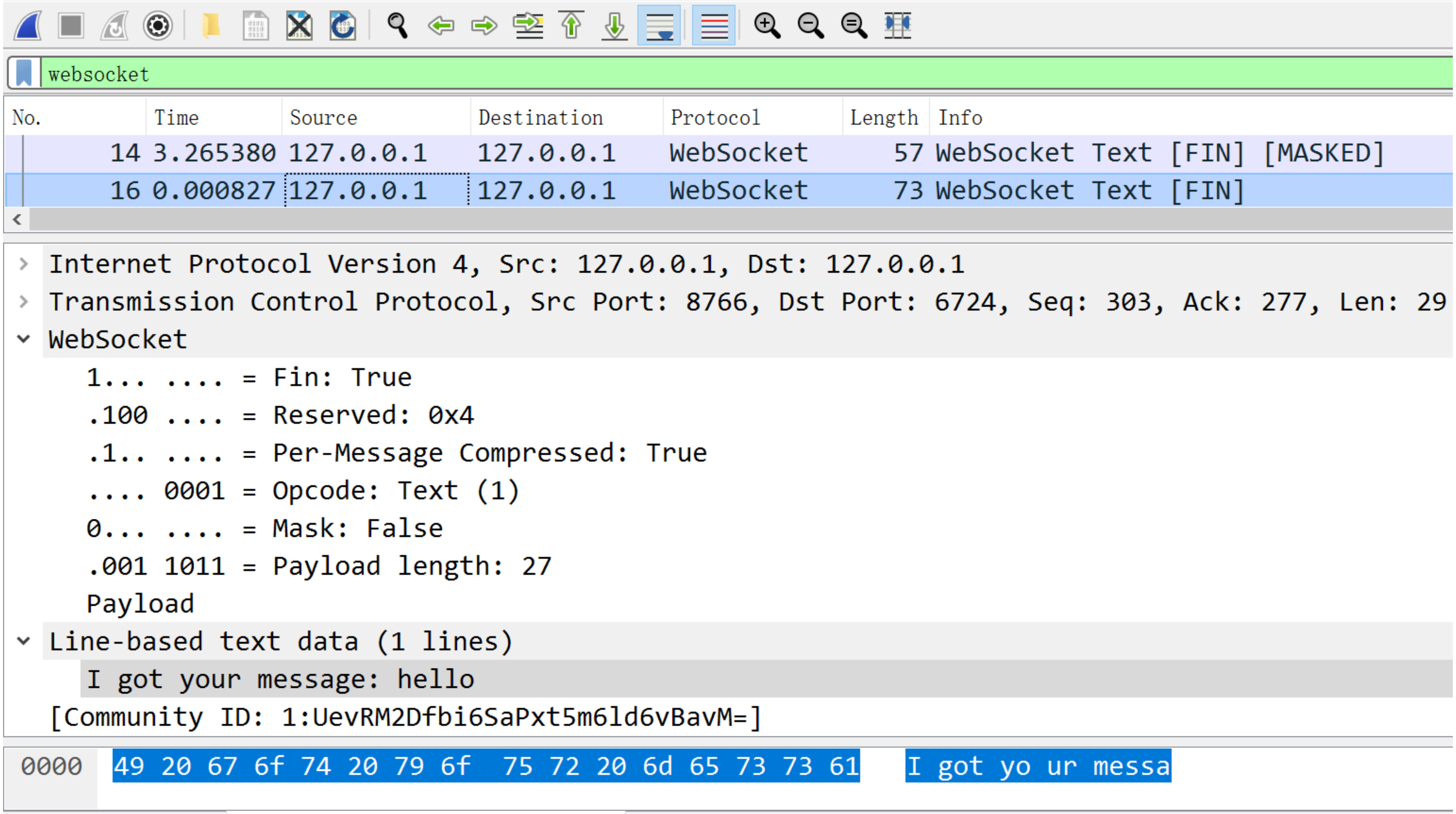
```
C:\Users\wq>curl --include --no-buffer --header "Connection: Upgrade" --header "Upgrade: w
ebsocket" --header "Host: example.com:80" --header "Origin: http://example.com:80"  --head
er "Sec-WebSocket-Key: GVsbG8sIHdvcmxkIQ==" --header "Sec-WebSocket-Version: 13" http://ex
ample.com:80/
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 05 Apr 2022 03:56:49 GMT
Etag: "3147526947"
Expires: Tue, 12 Apr 2022 03:56:49 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: EOS (vny/0454)
Content-Length: 1256
Connection: close

<!doctype html>
<html>
<head>
```



```
No.      Time     Source     Destination  Protocol   Length Info
      14 3.265380 127.0.0.1  127.0.0.1    WebSocket      57 WebSocket Text [FIN] [MASKED]
      16 0.000827 127.0.0.1  127.0.0.1    WebSocket      73 WebSocket Text [FIN]
```

```
⌄ WebSocket
    1... .... = Fin: True
    .100 .... = Reserved: 0x4
    .1.. .... = Per-Message Compressed: True
    .... 0001 = Opcode: Text (1)
    1... .... = Mask: True
    .000 0111 = Payload length: 7
    Masking-Key: 58905743
    Masked payload
    Payload
⌄ Line-based text data (1 lines)
    hello
  [Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]
```

```
0000  68 65 6c 6c 6f                                     hello
```



websocket

No.	Time	Source	Destination	Protocol	Length	Info
14	3.265380	127.0.0.1	127.0.0.1	WebSocket	57	WebSocket Text [FIN] [MASKED]
16	0.000827	127.0.0.1	127.0.0.1	WebSocket	73	WebSocket Text [FIN]

```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 8766, Dst Port: 6724, Seq: 303, Ack: 277, Len: 29
v WebSocket
    1... .... = Fin: True
    .100 .... = Reserved: 0x4
    .1.. .... = Per-Message Compressed: True
    .... 0001 = Opcode: Text (1)
    0... .... = Mask: False
    .001 1011 = Payload length: 27
    Payload
v Line-based text data (1 lines)
    I got your message: hello
  [Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]
```

```
0000  49 20 67 6f 74 20 79 6f  75 72 20 6d 65 73 73 61   I got yo ur messa
```

Practise 8.1

- Using Wireshark to capture and analysis the TCP stream.
- Invoke a HTTP request to get http://gaia.cs.umass.edu/wiresharklabs/alice.txt
- Analysis the TCP stream
  - Any duplicate ack, what's the possible reason?
  - Any TCP segment with sack permit option and sack option
    - select a tcp package which contained a sack option, find the segment ranges which is acked in this sack option
  - Any TCP retransmission? Is it retransmission or fast retransmission？
  - Any window size 0 segment, what does it mean? If the window size is 0, what would happened next on this tcp connection?
  - Any TCP window full segment, what does it mean?
- Tips: you can use some tools to cause your network congestion, such as clumsy-0.2-win64, which has been uploaded in Sakai site.

Tips on wireshark(1)

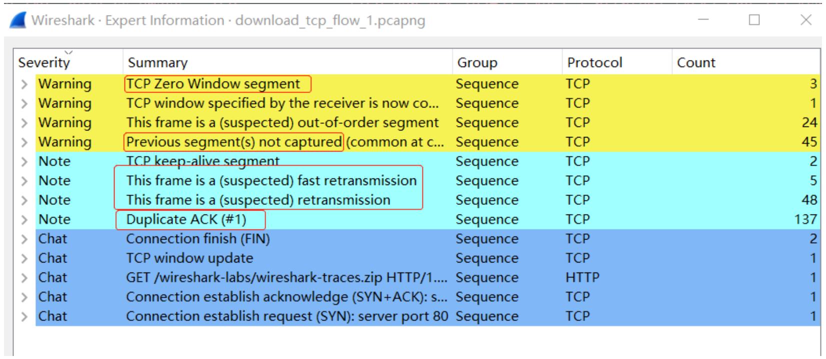
- How to get all the TCP segments of the TCP stream related to a http session in Wireshark:
  - 1st step: Find a HTTP packet in the HTTP session
  - 2nd step: right click the packet which will invoke a shortcut menu, then choose “follow ->TCP” in the shortcut menu
- How to find a special segment in a TCP stream
  - Using view filter in Wireshark
  - 1st step: if a TCP stream is filted, then a description such as “TCP.stream eq xx” (xx here is the id of this TCP stream) could be found in the view filter
  - 2nd step: in the same view filter, make a new filter rule description along with the original one,then press “Enter” key in your keyboard to make the new filter run
    - such as :

    TCP.stream eq 1 is the original one, to find a TCP zero window in this TCP stream,

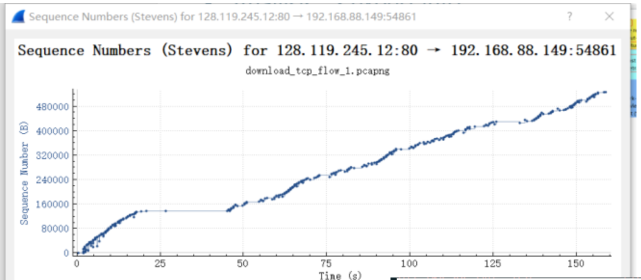
    TCP.stream eq 1 && TCP.window_size_value==0 is the new view filter description

Tips on wireshark(2)

- Analysis -> expert info
- Statistic->TCP stream graphs



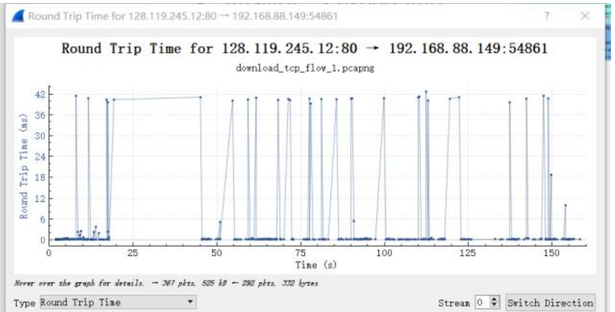
Severity	Summary	Group	Protocol	Count
Warning	TCP Zero Window segment	Sequence	TCP	3
Warning	TCP window specified by the receiver is now co...	Sequence	TCP	1
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	24
Warning	Previous segment(s) not captured (common at c...	Sequence	TCP	45
Note	TCP keep-alive segment	Sequence	TCP	2
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	5
Note	This frame is a (suspected) retransmission	Sequence	TCP	48
Note	Duplicate ACK (#1)	Sequence	TCP	137
Chat	Connection finish (FIN)	Sequence	TCP	2
Chat	TCP window update	Sequence	TCP	1
Chat	GET /wireshark-labs/wireshark-traces.zip HTTP/1....	Sequence	HTTP	1
Chat	Connection establish acknowledge (SYN+ACK): s...	Sequence	TCP	1
Chat	Connection establish request (SYN): server port 80	Sequence	TCP	1



Hover over the graph for details. → 367 pkts, 525 kB ← 292 pkts, 332 bytes

Type Time / Sequence (Stevens)

Mouse ● drags ○ zooms



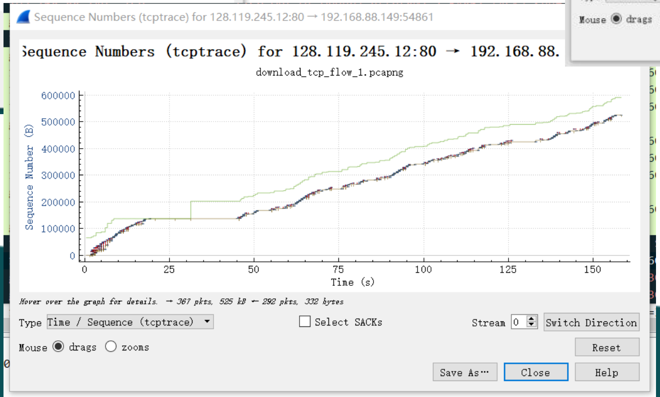
Hover over the graph for details. → 367 pkts, 525 kB ← 292 pkts, 332 bytes

Type Round Trip Time

Stream 0

Switch Direction

Mouse ● drags ○ zooms



Hover over the graph for details. → 367 pkts, 525 kB ← 292 pkts, 332 bytes

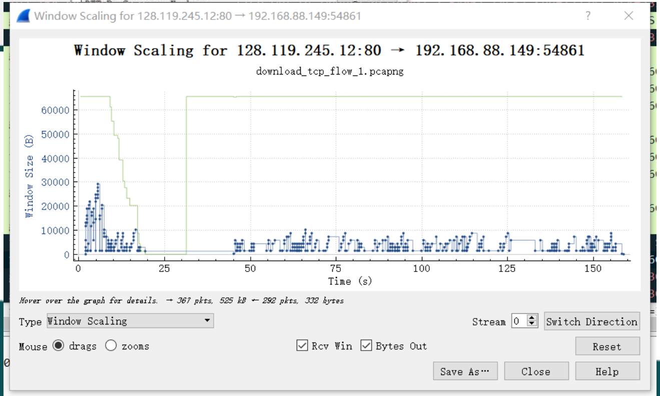
Type Time / Sequence (tcptrace)

□ Select SACKs

Stream 0

Switch Direction

Mouse ● drags ○ zooms



Hover over the graph for details. → 367 pkts, 525 kB ← 292 pkts, 332 bytes

Type Window Scaling

Stream 0

Switch Direction

Mouse ● drags ○ zooms

☑ Rcv Win ☑ Bytes Out

SUSTech
Southern University
of Science and Technology