EP2120 Lab 2

Group:
Edgar Welte (welte@kth.se),
Anamitra Bhar (anamitra@kth.se),
Appana Naga Satya Sai Surya Raghava Rathan (nsssrrap@kth.se)

Content

5. Measuring TCP with ttcp and Wireshark 2
6. TCP connection management 3
6.1. Connection establishment and termination 3
6.2. Connecting to a non-existent port 3
6.3. Connecting to a non-existing host 3
7. Fragmentation in TCP 4
8. TCP data transfer 5
8.1. Interactive application - fast link 5
8.2. Bulk transfer - fast link 6
8.3. Interactive application - slow link 9
8.4. Bulk transfer - slow link 10
9. TCP retransmission 12
10. TCP congestion control 13
Attachments 15

5. Measuring TCP with ttcp and Wireshark

5	211.429283	10.0.1.1	10.0.3.1	TCP	74	44420 → 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=537634 TSecr=0 WS=128
6	211.429404	10.0.3.1	10.0.1.1	TCP	54	5001 → 44420 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	216.449579	Raspberr_3f:c8:43	Intel_10:93:04	ARP	42	Who has 10.0.3.2? Tell 10.0.3.1
8	216.449973	Intel_10:93:04	Raspberr_3f:c8:43	ARP	60	10.0.3.2 is at 00:07:e9:10:93:04
9	225.766809	10.0.1.1	10.0.3.1	TCP	74	41342 → 1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=539068 TSecr=0 WS=128
10	225.766961	10.0.3.1	10.0.1.1	TCP	74	1234 → 41342 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=520727 TSec
11	225.767905	10.0.1.1	10.0.3.1	TCP	66	41342 → 1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=539068 TSecr=520727
12	225.768402	10.0.1.1	10.0.3.1	TCP	1066	41342 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1000 TSval=539068 TSecr=520727
13	225.768511	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=1001 Win=31232 Len=0 TSval=520755 TSecr=539068
14	225.768555	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=1001 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
15	225.768667	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=2449 Win=34176 Len=0 TSval=520755 TSecr=539068
16	225.768676	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=2449 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
17	225.768708	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=3897 Win=36992 Len=0 TSval=520755 TSecr=539068
18	225.768807	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=3897 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
19	225.768856	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=5345 Win=39936 Len=0 TSval=520755 TSecr=539068
20	225.768918	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=5345 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
21	225.768975	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=6793 Win=42880 Len=0 TSval=520755 TSecr=539068
22	225.769058	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=6793 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
23	225.769105	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=8241 Win=45696 Len=0 TSval=520755 TSecr=539068
24	225.769230	10.0.1.1	10.0.3.1	TCP	1514	41342 → 1234 [ACK] Seq=8241 Ack=1 Win=29312 Len=1448 TSval=539068 TSecr=520727
25	225.769235	10.0.1.1	10.0.3.1	TCP	378	41342 → 1234 [FIN, PSH, ACK] Seq=9689 Ack=1 Win=29312 Len=312 TSval=539068 TSecr=520727
26	225.769279	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [ACK] Seq=1 Ack=9689 Win=48640 Len=0 TSval=520755 TSecr=539068
27	225.770080	10.0.3.1	10.0.1.1	TCP	66	1234 → 41342 [FIN, ACK] Seq=1 Ack=10002 Win=51584 Len=0 TSval=520756 TSecr=539068
28	225.770902	10.0.1.1	10.0.3.1	TCP	66	41342 → 1234 [ACK] Seq=10002 Ack=2 Win=29312 Len=0 TSval=539068 TSecr=520756

Figure 1: Wireshark output for sending 10000 bytes via TCP from Host A to Host B.

Questions:

(a) How many packets are transmitted in total (count both directions)?

In total 19 packets are transmitted.

(b) What is the range of the sequence numbers used by the sender (Host A)?

Actual sequence number ranges from 383341817 to 383351818
(Relative sequence numbers 0-10001)

(c) How many packets do not carry a data payload?

12 packets carry no data payload.

(d) What is the total number of bytes transmitted in the recorded transfer? From those, calculate the amount of user data that was transmitted?

11336 bytes, User data was 10000 bytes

(e) Compare the total amount of data transmitted in the TCP data transfer to that of a UDP data transfer. Which of the protocols is more efficient in terms of overhead? What is the efficiency in percentage for these two protocols? (Recall the UDP measurements from the previous lab. How many bytes were sent in total using UDP?)

In UDP data transfer, the overhead is just

$$34\ bytes\ (IP\ header\ +\ Ethernet\ header) * number\ of\ fragments\ +$$
$$8\ bytes\ (one\ UDP\ header\ in\ the\ first\ fragment).$$

In comparison to TCP, where the overhead includes a TCP header for each single segment plus additional packets for connection establishment and ACK. Therefore, TCP has more overhead.

In the previous lab ten UDP messages of 1000 bytes were transmitted each with an overhead of 42 bytes. This results in a UDP overhead of 10*42 bytes = 420 bytes.

TCP efficiency is $\frac{10000}{11336} \approx 88\%$ and for UDP it is $\frac{10000}{10420} \approx 96\%$

6. TCP connection management

6.1. Connection establishment and termination

Questions for Connection establishment:

(a) Which packets constitute the three-way handshake? Which flags are set in the headers of these packets?

1	0.000000	10.0.1.1	10.0.3.1	TCP	74 50314 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=581971 TSecr=0 WS=128
2	0.000096	10.0.3.1	10.0.1.1	TCP	74 23 → 50314 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=563659 TSecr=581971 WS=128
3	0.001080	10.0.1.1	10.0.3.1	TCP	66 50314 → 23 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=581971 TSecr=563659

Flags – SYN, SYN-ACK, ACK

(b) What are the initial sequence numbers used by the client and the server, respectively?

Server – 0 ; Client - 1

(c) Which packet contains the first application data?

1	0.000000	10.0.1.1	10.0.3.1	TCP	74 50314 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=581971 TSecr=0 WS=128
2	0.000096	10.0.3.1	10.0.1.1	TCP	74 23 → 50314 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=563659 TSecr=581971 WS=128
3	0.001080	10.0.1.1	10.0.3.1	TCP	66 50314 → 23 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=581971 TSecr=563659
4	0.001463	10.0.1.1	10.0.3.1	TELNET	93 Telnet Data ...
5	0.001500	10.0.3.1	10.0.1.1	TCP	66 23 → 50314 [ACK] Seq=1 Ack=28 Win=29056 Len=0 TSval=563659 TSecr=581971

(d) What are the initial window sizes for the client and for the server?

Client – 29200 ; Server - 28960

(e) How long does it roughly take to open the TCP connection?

1.08 ms

Questions for Connection termination:

(f) Which packets are involved in closing the connection?

83	49.480701	10.0.3.1	10.0.1.1	TCP	66 23 → 50314 [FIN, ACK] Seq=976 Ack=134 Win=29056 Len=0 TSval=568607 TSecr=586917
84	49.481949	10.0.1.1	10.0.3.1	TCP	66 50314 → 23 [FIN, ACK] Seq=134 Ack=977 Win=31360 Len=0 TSval=586919 TSecr=568607
85	49.482124	10.0.3.1	10.0.1.1	TCP	66 23 → 50314 [ACK] Seq=977 Ack=135 Win=29056 Len=0 TSval=568607 TSecr=586919

(g) Which flags are set in these packets?

FIN-ACK, FIN-ACK, ACK

6.2. Connecting to a non-existent port

Questions:

(a) How does the server host (Host B) close the connection?

1	0.000000	0.0.0.0	255.255.255.255	DHCP	382 DHCP Discover - Transaction ID 0x4613d345
2	16.347220	10.0.1.1	10.0.3.1	TCP	74 57068 → 24 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=594871 TSecr=0 WS=128
3	16.347321	10.0.3.1	10.0.1.1	TCP	54 24 → 57068 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

It sends an RST i.e., reset in sent in response from the server.

(b) How long does the process of ending the connection take?

Around 16.3s

6.3. Connecting to a non-existing host

Questions:

(a) How often does the client try to open a connection? Note the time interval between attempts.

22	82.122419	10.0.1.1	10.0.1.42	TCP	74	45490 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=615479 TSecr=0 WS=128
23	82.122809	10.0.1.2	10.0.1.1	ICMP	102	Redirect (Redirect for host)
24	82.122937	Raspberr_69:62:93	Broadcast	ARP	42	Who has 10.0.1.42? Tell 10.0.1.1
25	82.122816	Intel_10:94:02	Broadcast	ARP	60	Who has 10.0.1.42? Tell 10.0.1.2
26	83.116159	Intel_10:94:02	Broadcast	ARP	60	Who has 10.0.1.42? Tell 10.0.1.2
27	83.167383	Raspberr_69:62:93	Broadcast	ARP	42	Who has 10.0.1.42? Tell 10.0.1.1
28	83.167416	10.0.1.1	10.0.1.42	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 45490 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=615584 TSecr=0 WS=128
29	83.167747	10.0.1.2	10.0.1.1	ICMP	102	Redirect (Redirect for host)
30	84.116164	Intel_10:94:02	Broadcast	ARP	60	Who has 10.0.1.42? Tell 10.0.1.2
31	84.207368	Raspberr_69:62:93	Broadcast	ARP	42	Who has 10.0.1.42? Tell 10.0.1.1
32	85.116193	10.0.1.2	10.0.1.1	ICMP	102	Destination unreachable (Host unreachable)
33	85.116253	10.0.1.1	10.0.1.42	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 45490 → 23 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=615778 TSecr=0 WS=128

Around every 1.5s (on an average)

(b) Does the client stop trying to connect at some point? If so, after how many attempts?

Yes, it stops trying after 3 attempts.

7. Fragmentation in TCP

Questions:

(a) How many packets did Host A measure and how many packets did Host B measure? Why?

A – 42 packets, B – 42 packets

In our transmission is no packet loss so every packet that leaves A reaches B.

(b) Is the DF flag set in the datagrams? Why?

DF is set to 1. For performance reasons, TCP can dynamically change its segment size to match the path MTU, so that each segment is carried in one IP packet. In this case fragmentation is not needed and wished.

(c) Do you observe fragmentation? If so, where does it occur?

No, fragmentation does not occur.

(d) Study the ICMP messages recorded at Host A. Which node is the source? What is the type and the code of the messages?

Source node is 10.0.1.2. Message has Type 3 and Code 0.

8	11.002580	10.0.1.2	10.0.1.1	ICMP	120	Destination unreachable (Network unreachable)
9	17.555462	10.0.1.1	10.0.3.1	TCP	74	41356 → 1234 [SYN] Seq=0 Win=29200 Len=0 MSS=146
10	17.556620	10.0.3.1	10.0.1.1	TCP	74	1234 → 41356 [SYN, ACK] Seq=0 Ack=1 Win=28960 Le
11	17.556726	10.0.1.1	10.0.3.1	TCP	66	41356 → 1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 T

```
> Frame 8: 120 bytes on wire (960 bits), 120 bytes captured (960 bits)
> Ethernet II, Src: Intel_10:94:02 (00:07:e9:10:94:02), Dst: Raspberr_69:62:93 (b8:27:eb:69:62:93)
> Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.1.1
v Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 0 (Network unreachable)
    Checksum: 0x185c [correct]
    [Checksum Status: Good]
    Unused: 00000000
```

8. TCP data transfer

8.1. Interactive application - fast link

Questions:

(a) Describe the payload of each packet.

The payload of each IP packet is a TCP header and a TCP payload. In case of just an ACK packet, the TCP payload is empty. In case of TELNET packet, the TCP payload hold just the typed character (one byte).

(b) Explain why you do not see four packets per typed character.

We see only three packets per character, because the ACK for the first TCP from Host A to Host B is included in the echo packet from Host B to Host A.

(c) When the client receives the echo, it waits a certain time before sending the ACK. Why? How long is the delay?

In theory there should be a noticeable delay of the ACK visible, as the client tries to include the ACK in a new data frame of the next typed character. So it waits a little time before sending just a empty TCP packet only holding the ACK. Unfortunately, in our Wireshark trace the time of the ACK is visible just directly after receiving the Echo packet. The time difference is only 0.000101 seconds.

7	4.689293	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
8	4.690834	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
9	4.690935	10.0.1.1	10.0.3.1	TCP	66 50326 → 23 [ACK]

Figure 2: Wireshark trace of one transmitted character via TELNET from HostA to HostB

(d) In the segments that carry characters, what window size is advertised by the telnet client and by the server? Does the window size vary as the connection proceeds?

The server (host B) always advertises a window size of 227 bytes. While the client (host A) advertises most of the time a window size of 245 bytes only during the last few transmissions advertise a window size of 262 bytes.

```
> Internet Protocol Version 4, Src: 10.0.3.1, Dst: 10.0.1.1
v Transmission Control Protocol, Src Port: 23, Dst Port: 50
    Source Port: 23
    Destination Port: 50326
    [Stream index: 0]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 1]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3309074199
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 2    (relative ack number)
    Acknowledgment number (raw): 1076509066
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
    Window: 227
```

```
> Internet Protocol Version 4, Src: 10.0.1.1, Dst: 10.0.3.1
v Transmission Control Protocol, Src Port: 50326, Dst Port:
    Source Port: 50326
    Destination Port: 23
    [Stream index: 0]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 1]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1076509065
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3309074199
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
    Window: 245
```

Figure 3: Advertised windows sizes of client and server during TELNET communication

(e) (Fast typing of character) Do you observe a difference in the transmission of segment payloads and ACKs?

We notice that the ACK from the client is attached more often directly to the next TCP packet which contain the next data. So, the number of empty TCP packets (with no payload) is reduced.

159	0.281807	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
160	0.283105	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
161	0.292312	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
162	0.293609	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
163	0.306385	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
164	0.307600	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
165	0.314030	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
166	0.315228	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
167	0.363172	10.0.1.1	10.0.3.1	TCP	66 50326 → 23 [ACK]
168	0.376594	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
169	0.377838	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
170	0.377977	10.0.1.1	10.0.3.1	TCP	66 50326 → 23 [ACK]
171	0.394762	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
172	0.395946	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
173	0.443227	10.0.1.1	10.0.3.1	TCP	66 50326 → 23 [ACK]
174	0.492412	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
175	0.493725	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
176	0.493792	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
177	0.495079	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
178	0.525430	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
179	0.526544	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
180	0.566348	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
181	0.567576	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
182	0.613176	10.0.1.1	10.0.3.1	TCP	66 50326 → 23 [ACK]
183	0.625955	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data ...
184	0.627229	10.0.3.1	10.0.1.1	TELNET	67 Telnet Data ...
185	0.627274	10.0.1.1	10.0.3.1	TELNET	67 Telnet Data

Figure 4: When typing many characters in short time, the ACK for receiving the echo is included in the next data packet from HostA to HostB, so less empty TCP packets are visible in the Wireshark trace.

8.2. Bulk transfer - fast link

Questions:

(a) How often does the receiver send ACKs? Can you see a rule on how TCP sends ACKs?

The first ACK is sent after receiving 1000 bytes, all other ACK were sent after receiving 1448 additional bytes (1448 bytes is average value for most of the ACK packets. Sometimes there were some jumps where 2*1448=2896 bytes were acknowledged. this is probable due to lost packets.) Interesting to see is, that the ACK packets mostly come in pairs of two packets within a very short time. So, in the tcptrace graph it looks like two segments were acknowledge at the same time. These bundles came with a time distance of in average 0.00025 seconds.

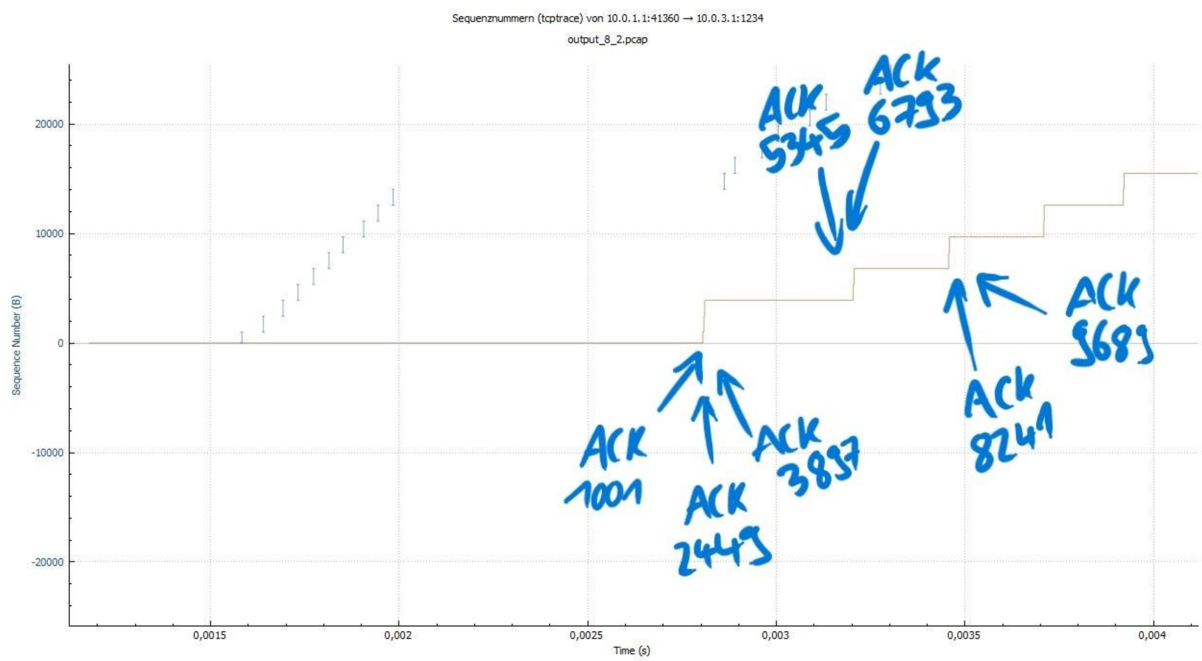

Figure 5: ACK from HostB come mostly groups of two packets at nearly the same time. Each ACK acknowledges 1448 bytes (one segement).

The value 1448 matches the MSS on the Ethernet link with a MTU of 1500 bytes (1500 – 20 bytes IP header – 32 bytes TCP header (20 + 12 bytes options) = 1448 bytes).

(b) How many bytes of data does a receiver acknowledge in a typical ACK?

As said above the typical ACK packet acknowledges 1448 bytes.

(c) How does the window size vary during the session?

The receiving window size increases at the beginning quickly 28960 to about 170000 bytes after some time it grows linearly to 503000 bytes and stays at this value until the end of the transmission.

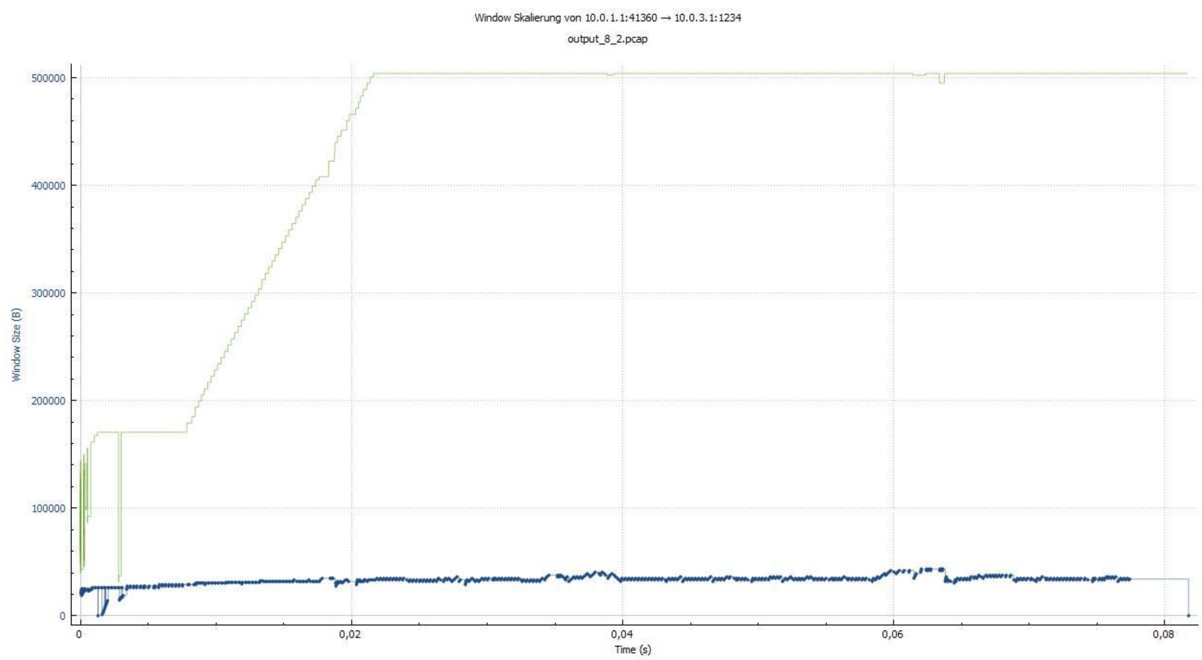

Figure 6: The graph of the receiving window over time (upper curve) and the sent segments (lower curve).

(d) Select any ACK packet in the Wireshark trace and note its acknowledgement number. Find the original segment in the Wireshark output. How long did it take from the transmission until it was ACKed?

It takes about 0.001335 seconds to ACK a sent segment. See in the following picture:

21	*REF*	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=19825 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
22	0.000043	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=21273 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
23	0.000115	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=5345 Win=39936 Len=0 TSval=771485 TSecr=789798
24	0.000187	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=22721 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
25	0.000215	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=24169 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
26	0.000117	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=6793 Win=42880 Len=0 TSval=771485 TSecr=789798
27	0.000260	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=25617 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
28	0.000305	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [PSH, ACK] Seq=27065 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
29	0.000368	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=8241 Win=45696 Len=0 TSval=771485 TSecr=789798
30	0.000370	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=9689 Win=48640 Len=0 TSval=771485 TSecr=789798
31	0.000467	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=28513 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
32	0.000515	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=29961 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
33	0.000551	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=31409 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
34	0.000620	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=11137 Win=51584 Len=0 TSval=771485 TSecr=789798
35	0.000621	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=12585 Win=54400 Len=0 TSval=771485 TSecr=789798
36	0.000657	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=32857 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
37	0.000700	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=34305 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
38	0.000831	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=14033 Win=57344 Len=0 TSval=771485 TSecr=789798
39	0.000833	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=15481 Win=60160 Len=0 TSval=771485 TSecr=789798
40	0.000864	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=35753 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
41	0.000904	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=37201 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
42	0.001086	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=16929 Win=63104 Len=0 TSval=771485 TSecr=789798
43	0.001120	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=38649 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
44	0.001088	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=18377 Win=66048 Len=0 TSval=771485 TSecr=789798
45	0.001188	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=40097 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
46	0.001334	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=19825 Win=68864 Len=0 TSval=771485 TSecr=789798
47	0.001366	10.0.1.1	10.0.3.1	TCP	1514	41360 → 1234 [ACK] Seq=41545 Ack=1 Win=29312 Len=1448 TSval=789798 TSecr=771485
48	0.001335	10.0.3.1	10.0.1.1	TCP	66	1234 → 41360 [ACK] Seq=1 Ack=21273 Win=71808 Len=0 TSval=771485 TSecr=789798

Figure 7: Measuring the time between sending a segment and receiving the ACK for this whole segment.

(e) Does the TCP sender generally transmit the maximum number of bytes as allowed by the receiver?

No, the sender is allowed to send as many bytes as the receiving window suggest (if now congestion occurred). But here the sender just sends ~30000 bytes before receiving the ACK for a segment. This is much less than the receiving window (at the end ~503000 bytes).

8.3. Interactive application - slow link

Questions:

(a) How many packets are transferred for each keystroke? Does the number change when you type faster?

When typing slow, also only one character is transmitted at a time. So, three packets are transferred per keystroke:

- TCP packet with keystroke as payload (HostA -> HostB),
- TCP packet with echo as payload (HostB -> HostA),
- TCP packet with only ACK (HostA -> HostB)

When typing faster multiple characters are combined in one TCP packet. So, there are less than three packets transferred per keystroke.

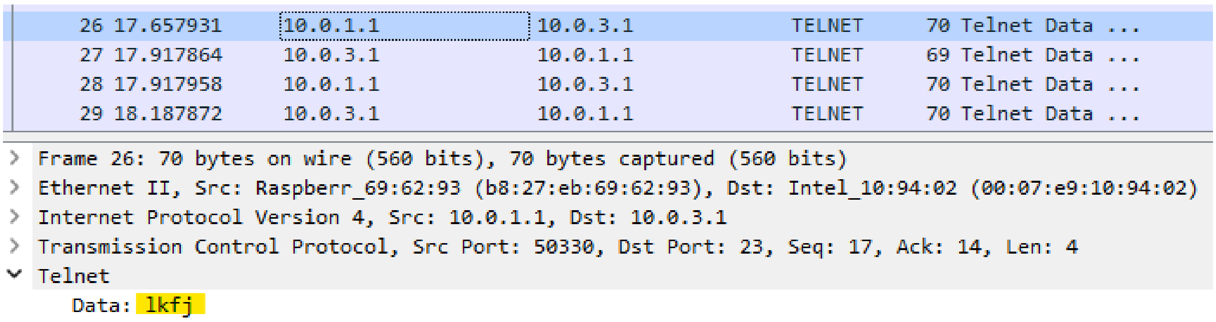

Figure 8: On the slow link many characters are combined in one packet when typing fast.

(b) Do you observe delayed acknowledgements?

Yes, one example is shown in following image. An echo packet was received on the client, but the ACK was only sent after 0.044 seconds. This is a delayed ACK.

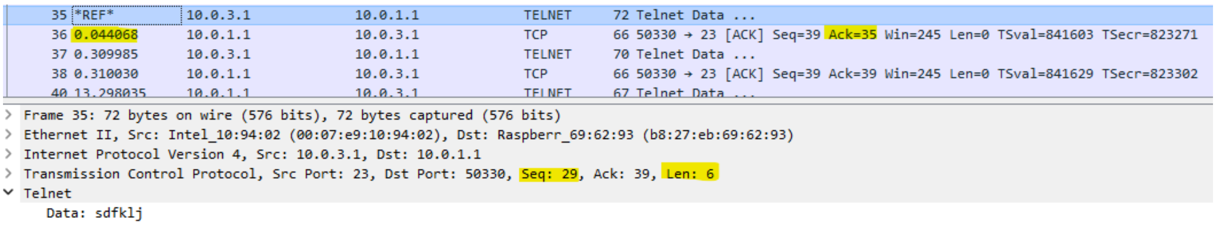

Figure 9: Example of delayed ACK. The bytes 29-34 are ACK only after 0.044 seconds.

(c) Do you observe the effect of Nagle's algorithm? How many characters can you see in a segment?

Yes, the effect of Nagle's algorithm can be observed. Nagle's algorithm says that only one tinygram should be not acknowledged, so the sender needs to wait and collect data, until the previous sent data was acknowledged. This can be seen in the Wireshark trace clearly. At some times 4-6 characters are accumulated in one tinygram before sending.

```
25 17.657857   10.0.3.1   10.0.1.1   TELNET   69 Telnet Data ...
26 17.657931   10.0.1.1   10.0.3.1   TELNET   70 Telnet Data ...
27 17.917864   10.0.3.1   10.0.1.1   TELNET   69 Telnet Data ...
> Frame 26: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
> Ethernet II, Src: Raspberr_69:62:93 (b8:27:eb:69:62:93), Dst: Intel_10:94:02 (00:07:e9:10:94:02)
> Internet Protocol Version 4, Src: 10.0.1.1, Dst: 10.0.3.1
> Transmission Control Protocol, Src Port: 50330, Dst Port: 23, Seq: 17, Ack: 14, Len: 4
v Telnet
    Data: lkfj
```

ACK17

Figure 10: TCP sender accumulated multiply characters until it receives the ACK for the previously sent segment.

8.4. Bulk transfer - slow link

Questions:

(a) Look at the pattern of segments and ACKs. Did the frequency of ACKs change compared to the bulk transfer on the fast link? How?

The frequency of the ACKs is much lower than on the fast link. On slow link the ACKs comes only every 0.8 or 1.25 seconds.

```
27 10.820573   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=1549 Win=32512 Len=0 TSval=984983 TSecr=1002928
29 12.070620   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=2097 Win=33664 Len=0 TSval=985108 TSecr=1002928
31 12.900608   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=2449 Win=34688 Len=0 TSval=985191 TSecr=1002928
34 14.170670   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=2997 Win=35840 Len=0 TSval=985318 TSecr=1002928
36 15.420642   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=3545 Win=36992 Len=0 TSval=985443 TSecr=1002928
39 16.260680   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=3897 Win=38016 Len=0 TSval=985527 TSecr=1002928
41 17.510690   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=4445 Win=39168 Len=0 TSval=985652 TSecr=1002928
44 18.770759   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=4993 Win=40192 Len=0 TSval=985778 TSecr=1002928
46 19.610757   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=5345 Win=41344 Len=0 TSval=985862 TSecr=1003078
49 20.860814   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=5893 Win=42368 Len=0 TSval=985987 TSecr=1003181
51 22.110815   10.0.3.1   10.0.1.1   TCP   66 1234 → 41368 [ACK] Seq=1 Ack=6441 Win=43520 Len=0 TSval=986112 TSecr=1003308
```

Figure 11: ACKs on the slow link only received every 0.8-1.25 seconds.

(b) Are the window sizes advertised by the receiver different from those of the previous exercise?

Yes, the receiver window size will not get as large as on the fast link, but only because less data is transmitted. It increases in a similar way linearly from 28960 to about 136704 bytes.

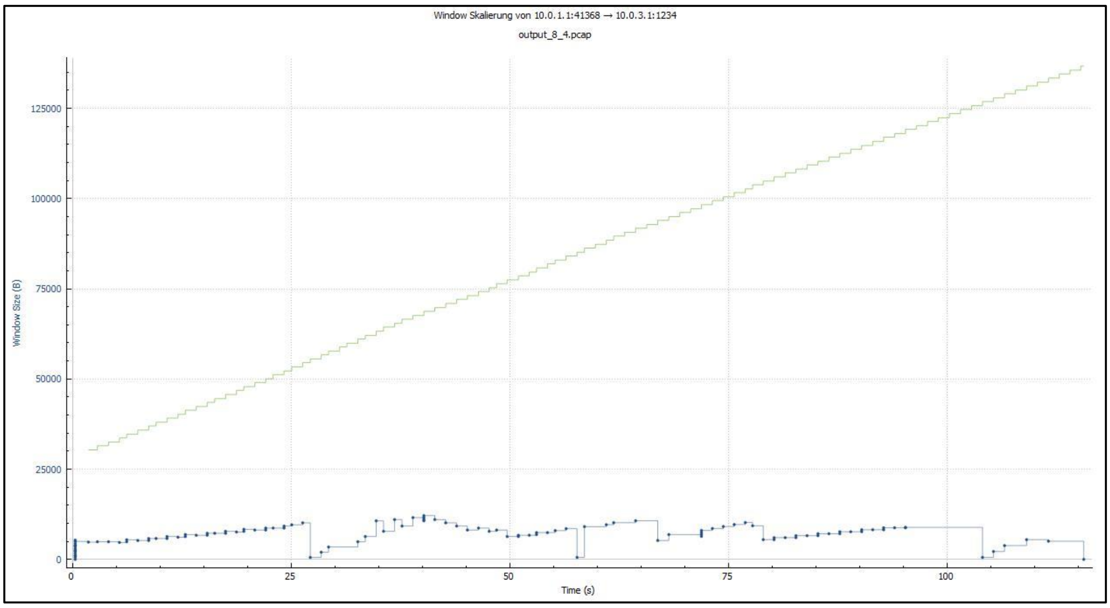

Figure 12: The receiving window size (upper curve) increases linearly during transmission.

(c) Does the TCP sender generally transmit the maximum number of bytes as allowed by the receiver?

No, if we look at the following example, the advertised receiver window size is 36992 bytes, but the sender only transmits 9458 bytes until it receives a ACK for a sent segment.

36	15.420642	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=3545 Win=36992 Len=0 TSval=985443 TSecr=1002928
37	15.420698	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=8241 Ack=1 Win=29312 Len=548 TSval=1003768 TSecr=985443
38	15.420720	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=8789 Ack=1 Win=29312 Len=548 TSval=1003768 TSecr=985443
39	16.260680	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=3897 Win=38016 Len=0 TSval=985527 TSecr=1002928
40	16.260724	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=9337 Ack=1 Win=29312 Len=352 TSval=1003852 TSecr=985527
41	17.510690	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=4445 Win=39168 Len=0 TSval=985652 TSecr=1002928
42	17.510739	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=9689 Ack=1 Win=29312 Len=548 TSval=1003977 TSecr=985652
43	17.510762	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=10237 Ack=1 Win=29312 Len=548 TSval=1003977 TSecr=985652
44	18.770759	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=4993 Win=40192 Len=0 TSval=985778 TSecr=1002928
45	18.770884	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=10785 Ack=1 Win=29312 Len=352 TSval=1004103 TSecr=985778
46	19.610757	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=5345 Win=41344 Len=0 TSval=985862 TSecr=1003078
47	19.610807	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=11137 Ack=1 Win=29312 Len=548 TSval=1004187 TSecr=985862
48	19.610838	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=11685 Ack=1 Win=29312 Len=548 TSval=1004187 TSecr=985862
49	20.860814	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=5893 Win=42368 Len=0 TSval=985987 TSecr=1003181
50	20.860938	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=12233 Ack=1 Win=29312 Len=352 TSval=1004312 TSecr=985987
51	22.110815	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=6441 Win=43520 Len=0 TSval=986112 TSecr=1003308
52	22.110906	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=12585 Ack=1 Win=29312 Len=548 TSval=1004437 TSecr=986112
53	22.110932	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=13133 Ack=1 Win=29312 Len=548 TSval=1004437 TSecr=986112
54	22.970815	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=6793 Win=44544 Len=0 TSval=986197 TSecr=1003433
55	22.970849	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=13681 Ack=1 Win=29312 Len=352 TSval=1004523 TSecr=986197
56	24.210844	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=7341 Win=45696 Len=0 TSval=986322 TSecr=1003516
57	24.210934	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=14033 Ack=1 Win=29312 Len=548 TSval=1004647 TSecr=986322
58	24.210958	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=14581 Ack=1 Win=29312 Len=548 TSval=1004647 TSecr=986322
59	25.460917	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=7889 Win=46848 Len=0 TSval=986447 TSecr=1003516
60	25.461020	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=15129 Ack=1 Win=29312 Len=352 TSval=1004773 TSecr=986447
61	26.300903	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=8241 Win=47872 Len=0 TSval=986531 TSecr=1003643
62	26.300992	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=15481 Ack=1 Win=29312 Len=548 TSval=1004857 TSecr=986531
63	26.301015	10.0.1.1	10.0.3.1	TCP	614	41368 → 1234 [ACK] Seq=16029 Ack=1 Win=29312 Len=548 TSval=1004857 TSecr=986531
64	27.550956	10.0.3.1	10.0.1.1	TCP	66	1234 → 41368 [ACK] Seq=1 Ack=8789 Win=49024 Len=0 TSval=986656 TSecr=1003768
65	27.551084	10.0.1.1	10.0.3.1	TCP	418	41368 → 1234 [ACK] Seq=16577 Ack=1 Win=29312 Len=352 TSval=1004982 TSecr=986656

Figure 13: TCP sender sends 9458 bytes during one RTT. The advertised receiver window size is 26992 bytes.

9. TCP retransmission

Questions:

(a) How many packets are transmitted at retransmission timeout?

42 packets are sent between disconnection time (127s) and reconnection time (260s).

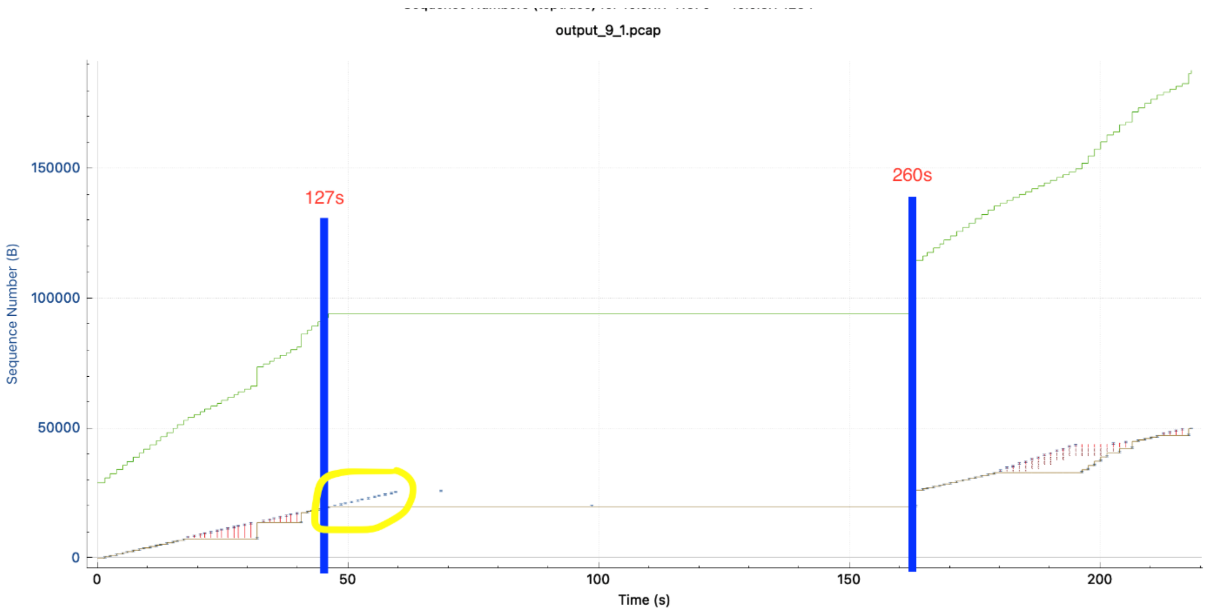

Figure 14: tcptrace graph of the transmission of TCP data, were during transmission the connection was disconnected for some time.

(b) Do the retransmissions end at some point?

Retransmissions were continuous until the end. Last full data packet from host A to host B was a retransmission of a previously lost packet. After receiving the ACK for nearly all data (just 136 bytes left), HostA sends the rest of the data and set the FIN flag to end the connection.

204	312.437251	10.0.1.1	10.0.3.1	TCP	614	41370 → 1234 [ACK] Seq=49317 Ack=1 Win=29312 Len=548 TSval=1135474 TSecr=1117144
205	312.437402	10.0.3.1	10.0.1.1	TCP	78	[TCP Window Update] 1234 → 41370 [ACK] Seq=1 Ack=47125 Win=135552 Len=0 TSval=1118647 TSecr=1134975 SLE=47673 SRE=49865
206	313.697229	10.0.1.1	10.0.3.1	TCP	614	[TCP Retransmission] 41370 → 1234 [ACK] Seq=47125 Ack=1 Win=29312 Len=548 TSval=1136850 TSecr=1118522
207	313.697303	10.0.3.1	10.0.1.1	TCP	66	1234 → 41370 [ACK] Seq=1 Ack=49865 Win=130704 Len=0 TSval=1118773 TSecr=1136850
208	314.227204	10.0.1.1	10.0.3.1	TCP	202	41370 → 1234 [FIN, PSH, ACK] Seq=49865 Ack=1 Win=29312 Len=136 TSval=1137098 TSecr=1118773
209	314.228058	10.0.3.1	10.0.1.1	TCP	66	1234 → 41370 [FIN, ACK] Seq=1 Ack=50002 Win=137728 Len=0 TSval=1118826 TSecr=1137098
210	314.477167	10.0.1.1	10.0.3.1	TCP	66	41370 → 1234 [ACK] Seq=50002 Ack=2 Win=29312 Len=0 TSval=1137151 TSecr=1118826

10. TCP congestion control

Questions:

(a) Try to observe periods when TCP sender is in slow start phase and when the sender switches to congestion control. Verify if the congestion window follows the rule of the slow-start phase.

In our trace there is no really appearance of slow-start. The sender sends at the start already 14692 bytes (~10*MSS), which is not the recommended initial CWND of 1…4*MSS. But after that the Congestion avoidance phase is clearly visible, when looking on the send bytes. After every RTT the amount of sent bytes increases by 1448 (MSS).

2	12.678897	10.0.1.1	10.0.3.1	TCP	74	41372 → 1234 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1189309 TSecr=0 WS=128
3	12.821790	10.0.3.1	10.0.1.1	TCP	74	1234 → 41372 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1170967 TSe
4	12.821926	10.0.1.1	10.0.3.1	TCP	66	41372 → 1234 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1189323 TSecr=1170967
5	12.822316	10.0.1.1	10.0.3.1	TCP	1066	41372 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1000 TSval=1189323 TSecr=1170967
6	12.822380	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=1001 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
7	12.822441	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=2449 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
8	12.822493	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=3897 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
9	12.822565	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=5345 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
10	12.822626	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=6793 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
11	12.822698	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=8241 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
12	12.822754	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=9689 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
13	12.822832	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=11137 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
14	12.822899	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=12585 Ack=1 Win=29312 Len=1448 TSval=1189323 TSecr=1170967
15	12.941800	10.0.3.1	10.0.1.1	TCP	66	1234 → 41372 [ACK] Seq=1 Ack=1001 Win=31232 Len=0 TSval=1171010 TSecr=1189323
16	12.941932	10.0.1.1	10.0.3.1	TCP	1514	41372 → 1234 [ACK] Seq=14033 Ack=1 Win=29312 Len=1448 TSval=1189335 TSecr=1171010
17	13.061852	10.0.3.1	10.0.1.1	TCP	66	1234 → 41372 [ACK] Seq=1 Ack=2449 Win=34176 Len=0 TSval=1171010 TSecr=1189323

(b) Can you find occurrences of fast recovery?

Yes, fast recovery is visible in the trace. When looking on the window size graph we can see that the amount of sent bytes drops after a short phase of duplicate ACKs at time t=12.5 seconds to the ½ of the original CWND size (here ~11000 bytes).

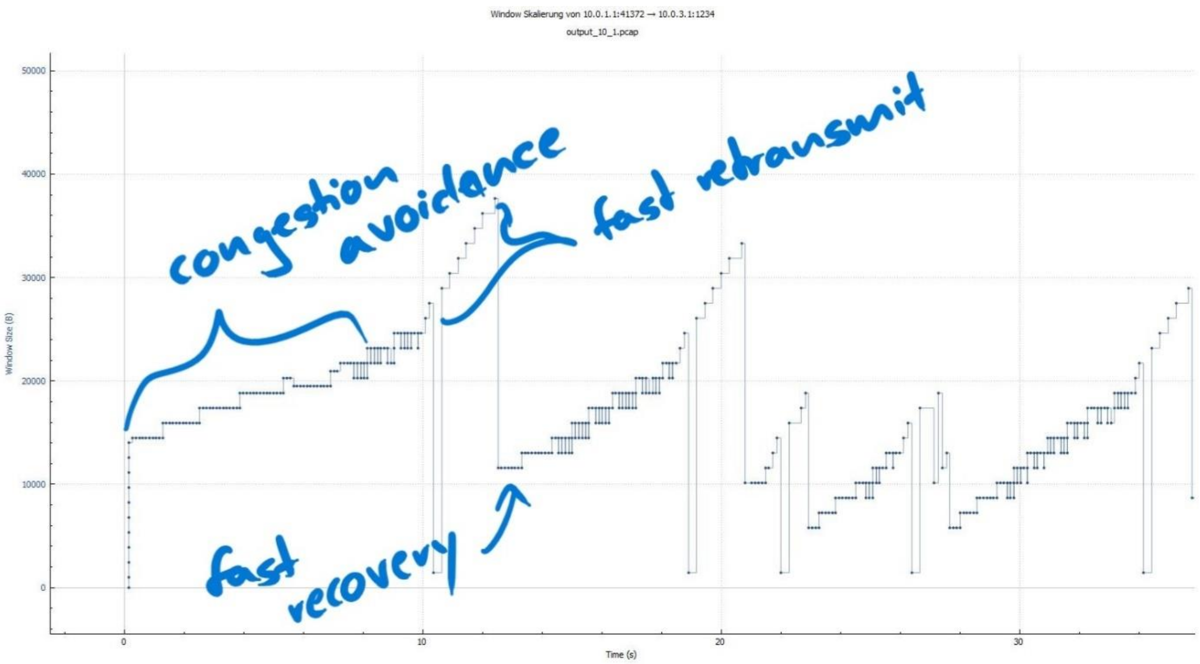

Figure 15: Graph of the amount of sent bytes at HostA (fast link data, slow link ACK). Congestion avoidance and Fast recovery is visible.

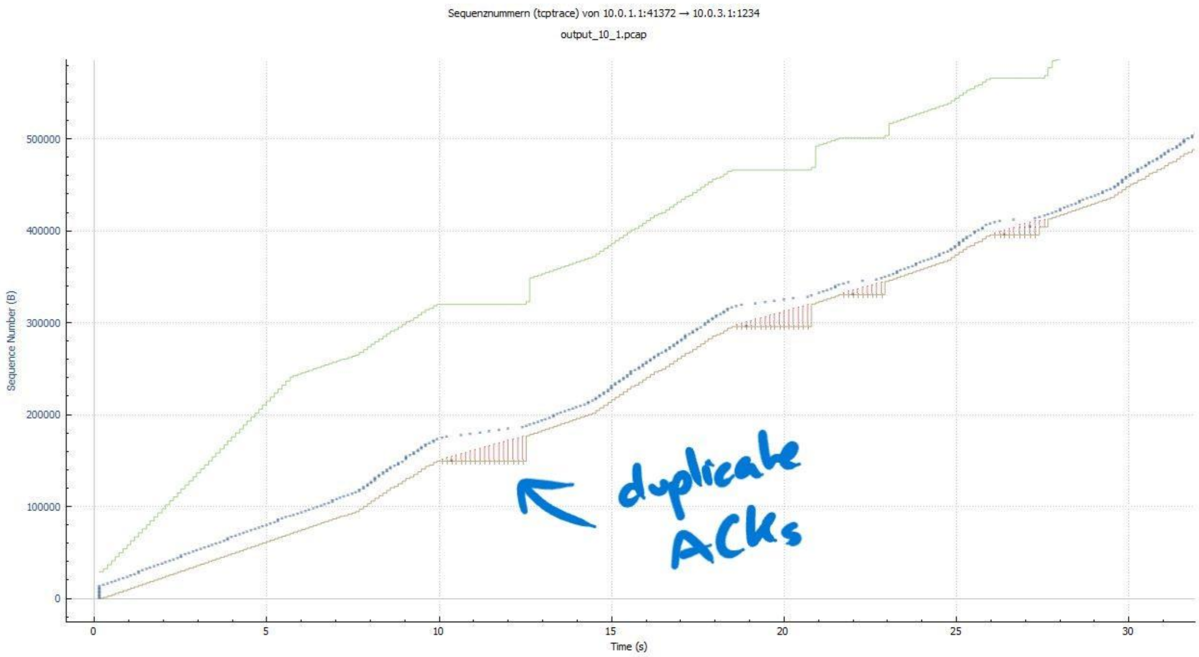

Figure 16: The tcptrace graph where the receiving of the duplicated ACK are visible. This is a trigger fast retransmit and fast recovery.

Comparison to the experiment where both directions are on fast link.

If both directions are on the fast link, the ACK will come earlier, so the transmission of the data is faster. If we just look at the congestion control, we can see that in our measurements a little less moments of Fast Recovery occurred. This is probably due to the better transmission quality of the Ethernet link compared to the serial link. Apart from this fact the two graphs looking very similar.

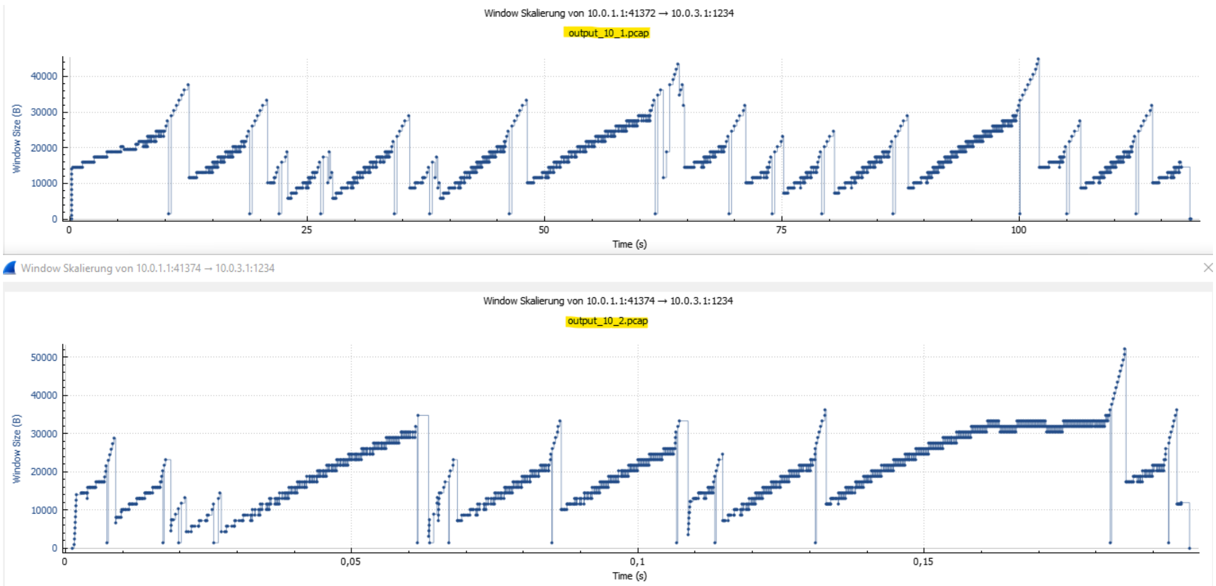

Figure 17: In the top chart the amount of sent bytes in the fast/slow link experiment is visualized. The bottom chart the amount of sent bytes of the fast/fast link experiment are visualized.

Attachments

- output_5.pcap
- output_6_1.pcap
- output_6_2.pcap
- output_6_3.pcap
- output_7_A.pcap
- output_7_B.pcapng
- output_8_1.pcap
- output_8_2.pcap
- output_8_2_tcptrace.jpg
- output_8_2_tcptrace_windowsize.jpg
- output_8_2_tcptrace_zoomin.jpeg
- output_8_2_tcptrace_zoomin_edit.jpg
- output_8_3.pcap
- output_8_4.pcap
- output_8_4_tcptrace.jpg
- output_8_4_tcptrace_windowsize.jpg
- output_9_1.pcap
- output_9_1_tcptrace.png
- output_10_1.pcap
- output_10_1_tcptrace.jpg
- output_10_1_tcptrace_windowsize.jpg
- output_10_1_tcptrace_windowsize_edit.jpg
- output_10_1_tcptrace_zoomin.jpg
- output_10_1_tcptrace_zoomin_edit.jpg
- output_10_2.pcap
- output_10_2_tcptrace.jpg