

RECHNERNETZE LABOR 3

Aufgabe 1.1

Wichtige Funktionen vom Server

Beschreibung	Line	Funktion
TCP socket	95	socket()
Registrierung	113	bind()
Schließt den Socket	116	close(ss)
Buffer	127	listen()
Verbindungsaufbau	150	accept()
Lese Buffer	219	read()
Schreibe an Client	270	write()

Wichtige Funktionen des Clients

Beschreibung	Line	Funktion
TCP socket	112	socket()
Registrierung	149	bind()
Schließt den Socket	155	close(ss)
Schreibe an Server	244	listen()
Lese Buffer	262	read()

1.1.1 Ein Server und ein Client

Versuchsziel:

Ausführen der vorgegebenen Programme auf zwei verschiedenen Rechnern.
Dokumentation des Datentransfers im Netzwerk mithilfe von Wireshark.
Herausfinden woran man erkennt, dass das Programm kein nebenläufiger Server ist.

Versuchsdurchführung:

Man kann in der Konsolenausgabe sehr schön sehen, dass der Server sich immer nur mit einem Client verbindet und auch nur mit einem Client unterhält.

No.	Time	Source	Destination	Protocol	Length	Ir
38	0.087676	134.108.8.37	134.108.8.36	TCP	74	54774 → 9001 [SYN] Seq
39	0.000051	134.108.8.36	134.108.8.37	TCP	74	9001 → 54774 [SYN, ACK
40	0.000181	134.108.8.37	134.108.8.36	TCP	66	54774 → 9001 [ACK] Seq
...						
186	0.000006	134.108.8.36	134.108.8.37	TCP	1514	9001
187	0.000151	134.108.8.37	134.108.8.36	TCP	66	5477
188	0.000028	134.108.8.36	134.108.8.37	TCP	1514	9001
189	0.000006	134.108.8.36	134.108.8.37	TCP	1514	9001
190	0.000182	134.108.8.37	134.108.8.36	TCP	66	5477
191	0.000042	134.108.8.36	134.108.8.37	TCP	1514	9001
192	0.000011	134.108.8.36	134.108.8.37	TCP	1034	9001
193	0.000168	134.108.8.37	134.108.8.36	TCP	66	5477
194	0.000018	134.108.8.37	134.108.8.36	TCP	69	5477
195	0.000007	134.108.8.37	134.108.8.36	TCP	66	5477
196	0.000032	134.108.8.36	134.108.8.37	TCP	66	9001
197	0.039637	134.108.8.36	134.108.8.37	TCP	66	9001
208	0.960503	134.108.8.36	134.108.8.37	TCP	66	9001
209	0.000165	134.108.8.37	134.108.8.36	TCP	66	5477

Aufgabe 1.1.2

Versuchsziel

Es wird versucht mit zwei Clients zu einem nicht nebenläufigen Server eine Verbindung aufzubauen.

Versuchsdurchführung

Client A wird gestartet und sendet unverzüglich Daten zum Server. Anschließend wird in einem weiteren Fenster, ein zweiter Client B gestartet und Daten zum Server abgeschickt. Danach wird die Übertragung von A fortgesetzt und nach der Terminierung von A wird B fortgesetzt und beendet.

Woran ist eindeutig erkennbar, dass der Server sequentiell arbeitet?


Daran, dass Client B erst behandelt wird nachdem Client A die Verbindung beendet hat.

Wo blockiert der Server?

Wird durch die Funktion listen() blockiert, da nur eine Verbindung zugelassen ist.


Three Way Handshake Client A:

No.	Time	Source	Destination	Protocol	Length	Ir
41	0.000000	134.108.8.36	134.108.8.37	TCP	74	48072 → 9001 [SYN] Sec
42	0.000177	134.108.8.37	134.108.8.36	TCP	74	9001 → 48072 [SYN, ACK
43	0.000023	134.108.8.36	134.108.8.37	TCP	66	48072 → 9001 [ACK] Sec




Dynamische Buffer Erweiterung Client A:

43	0.000023	134.108.8.36	134.108.8.37	TCP	66	48072 → 9001 [ACK] Sec
76	4.598437	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [T
77	0.000019	134.108.8.36	134.108.8.37	TCP	1514	[TCP
78	0.000201	134.108.8.37	134.108.8.36	TCP	66	9001 → 48072 [SYN, ACK
79	0.000033	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [ACK] Sec
80	0.000010	134.108.8.36	134.108.8.37	TCP	1514	[TCP
81	0.000188	134.108.8.37	134.108.8.36	TCP	66	9001 → 48072 [SYN, ACK
82	0.000028	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [ACK] Sec
83	0.000010	134.108.8.36	134.108.8.37	TCP	1514	[TCP
84	0.000220	134.108.8.37	134.108.8.36	TCP	66	9001 → 48072 [SYN, ACK
85	0.000028	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [ACK] Sec
86	0.000010	134.108.8.36	134.108.8.37	TCP	1514	[TCP
87	0.000227	134.108.8.37	134.108.8.36	TCP	66	9001 → 48072 [SYN, ACK
88	0.000027	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [ACK] Sec
89	0.000009	134.108.8.36	134.108.8.37	TCP	1514	[TCP
90	0.000171	134.108.8.37	134.108.8.36	TCP	66	9001 → 48072 [SYN, ACK
91	0.000028	134.108.8.36	134.108.8.37	TCP	1514	48072 → 9001 [ACK] Sec
92	0.000009	134.108.8.36	134.108.8.37	TCP	1514	[TCP



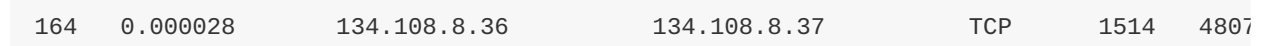
Client B verbindet sich:

151	3.766654	134.108.8.36	134.108.8.37	TCP	74	48074 → 9001 [SYN] Sec
152	0.000191	134.108.8.37	134.108.8.36	TCP	74	9001 → 48074 [SYN, ACK
153	0.000038	134.108.8.36	134.108.8.37	TCP	66	48074 → 9001 [ACK] Sec




Dynamische Buffer Erweiterung Client 2:

156	0.000017	134.108.8.36	134.108.8.37	TCP	1514	[TCP
157	0.000187	134.108.8.37	134.108.8.36	TCP	66	9001 → 48074 [SYN, ACK
158	0.000033	134.108.8.36	134.108.8.37	TCP	1514	48074 → 9001 [ACK] Sec
159	0.000009	134.108.8.36	134.108.8.37	TCP	1514	[TCP
160	0.000224	134.108.8.37	134.108.8.36	TCP	66	9001 → 48074 [SYN, ACK
161	0.000028	134.108.8.36	134.108.8.37	TCP	1514	48074 → 9001 [ACK] Sec
162	0.000010	134.108.8.36	134.108.8.37	TCP	1514	[TCP
163	0.000190	134.108.8.37	134.108.8.36	TCP	66	9001 → 48074 [SYN, ACK
164	0.000028	134.108.8.36	134.108.8.37	TCP	1514	48074 → 9001 [ACK] Sec




165	0.000009	134.108.8.36	134.108.8.37	TCP	1514	[TCF
166	0.000197	134.108.8.37	134.108.8.36	TCP	66	9001
167	0.000026	134.108.8.36	134.108.8.37	TCP	1514	4807




Daten von Client A empfangen und Verbindung beenden:

275	0.000262	134.108.8.37	134.108.8.36	TCP	2962	9001
276	0.000027	134.108.8.36	134.108.8.37	TCP	66	4807
277	0.000180	134.108.8.37	134.108.8.36	TCP	2482	9001
278	0.000026	134.108.8.36	134.108.8.37	TCP	66	4807
279	0.000022	134.108.8.36	134.108.8.37	TCP	69	4807
280	0.000017	134.108.8.36	134.108.8.37	TCP	66	4807
281	0.000146	134.108.8.37	134.108.8.36	TCP	66	9001
282	0.039033	134.108.8.37	134.108.8.36	TCP	66	9001
287	0.961062	134.108.8.37	134.108.8.36	TCP	66	9001
288	0.000038	134.108.8.36	134.108.8.37	TCP	66	4807



Daten von Client B empfangen und Verbindung beenden:

316	0.000215	134.108.8.37	134.108.8.36	TCP	2962	9001
317	0.000063	134.108.8.36	134.108.8.37	TCP	66	4807
318	0.000218	134.108.8.37	134.108.8.36	TCP	2482	9001
319	0.000028	134.108.8.36	134.108.8.37	TCP	66	4807
320	0.000029	134.108.8.36	134.108.8.37	TCP	69	4807
321	0.000013	134.108.8.36	134.108.8.37	TCP	66	4807
322	0.000141	134.108.8.37	134.108.8.36	TCP	66	9001
323	0.039375	134.108.8.37	134.108.8.36	TCP	66	9001
330	0.960711	134.108.8.37	134.108.8.36	TCP	66	9001
331	0.000037	134.108.8.36	134.108.8.37	TCP	66	4807



Aufgabe 1.1.3

Versuchsziel

Ein Server (nicht neben läufig) und ein Client, jedoch wird die Verbindung frühzeitig von der Clientseite beendet.

Versuchsdurchführung

Der Server und der Client werden gestartet. Im Server beantworten wir zunächst die Frage nach Lesen mit „n“ und die anschließende Frage nach Schreiben mit „j“. Im Client beantworten wir nun die Frage nach Schreiben mit „n“ und die anschließende Frage nach Lesen ebenfalls mit „n“.

Warum wird kein PDU mit FIN gesendet?

Der Server schickt zwar Daten, welche auch im Buffer landen aber der Client lehnt

jede reinkommende Nachricht vom Server ab. Wie man auch am Schluss sieht wird sogar die Bestätigung des RST Signals abgelehnt.

Wozu dient die RST-PDU?

Damit der Server mitgeteilt bekommt das der Client nichts annehmen will.

Was passiert mit den Daten des Servers?

Die befinden sich im Buffer werden aber nicht ausgelesen.

No.	Time	Source	Destination	Protocol	Length	Ir
552	55.401718	134.108.8.36	134.108.8.37	TCP	74	48088
553	55.401926	134.108.8.37	134.108.8.36	TCP	74	9001 -
554	55.401970	134.108.8.36	134.108.8.37	TCP	66	48088
739	77.220823	134.108.8.36	134.108.36.102	TCP	66	899 →
757	79.474738	134.108.8.36	134.108.34.11	TCP	112	56612
758	79.479684	134.108.34.11	134.108.8.36	TCP	112	3128 -
759	79.479714	134.108.8.36	134.108.34.11	TCP	66	56612
779	85.167953	134.108.8.36	134.108.8.37	TCP	69	48088
780	85.167983	134.108.8.36	134.108.8.37	TCP	66	48088
781	85.168191	134.108.8.37	134.108.8.36	TCP	66	9001 -
782	85.168218	134.108.8.36	134.108.8.37	TCP	54	48088

Aufgabe 1.2

Versuchsziel:

Verbindung mit einem Rechner in einem anderen Subnetz. Da die MSS nicht ausreicht (zu klein) muss diese angepasst werden.

Versuchsdurchführung:

Ein Rechner aus dem Rechenzentrum verbindet sich auf den Rechner außerhalb des Subnetzes und startet dort den Server. Der Rechner innerhalb des Subnetzes ist unser Client. Dann werden die vorgegebenen Programme ausgeführt.

Wie wurde die Maximum Segment Size berechnet?

Die Headergröße besteht aus 20 Byte IP, 20 Byte TCP und 24 Byte Tunneling) somit bleiben: $1500 \text{ Bytes} - 20 \text{ Bytes} - 20 \text{ Bytes} - 24 \text{ Bytes} = 1436 \text{ Bytes}$ für die MSS.

Was sagt die ICMP Nachricht vom Router?

Wie Wireshark trace zu sehen bei Frame No. 183 schickt der Router eine ICMP Nachricht mit der Info, dass eine Fragmentierung notwendig ist.

Warum soll fragmentiert werden?

Das Datenpaket muss fragmentiert werden, da Ethernet maximal 1500 Bytes pro Paket versenden kann (MTU).

Wie sieht TCP Segmentierung aus? Ist diese Segmentierung sinnvoll?

Die TCP Segmentierung fragmentiert das Datenpaket in zwei Datenpakete, einmal mit 1424 Bytes und einmal mit 24 Bytes Daten, dies führt zu unnötigem Datenverkehr. Dies könnte man durch verringern der MSS um 24 Bytes auf dem Clientserver verbessern.

Ohne Fragmentierung:

No.	Time	Source	Destination	Protocol	Length	Ir
183	19.314823	134.108.11.254	134.108.8.36	ICMP	70	Destination unrea
184	19.314851	134.108.8.36	134.108.190.10	TCP	1490	[TCP Retransmiss
185	19.314861	134.108.8.36	134.108.190.10	TCP	90	[TCP Retransmissic
186	19.314867	134.108.8.36	134.108.190.10	TCP	1490	[TCP Retransmiss
187	19.314872	134.108.8.36	134.108.190.10	TCP	90	[TCP Window Full]
188	19.315528	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
189	19.315558	134.108.8.36	134.108.190.10	TCP	1490	[TCP Window Full]
190	19.315566	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
191	19.315577	134.108.8.36	134.108.190.10	TCP	90	[TCP Window Full]
192	19.315583	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
193	19.315594	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [AC
194	19.315598	134.108.8.36	134.108.190.10	TCP	90	[TCP Window Full]
197	19.316218	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
198	19.316245	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [AC

Mit Fragmentierung:

No.	Time	Source	Destination	Protocol	Length	Ir
232	19.318494	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [PS
237	19.319163	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
238	19.319180	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [AC
239	19.319187	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [PS
244	19.320019	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
245	19.320034	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [AC
246	19.320042	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [PS
251	19.320869	134.108.190.10	134.108.8.36	TCP	66	9001 → 46962 [ACK]
252	19.320884	134.108.8.36	134.108.190.10	TCP	1490	46962 → 9001 [AC

Aufgabe 1.3

Da der Client die Verbindung vorzeitig beendet hat, haben wir einen half-closed status. Der Client sendet [FIN, ACK] und bekommt immernoch Daten, bis der Server ein [ACK] sendet und darauf ein weiteres [FIN, ACK] , [ACK] zum beenden vom Server bekommt.

No.	Time	Source	Destination	Protocol	Length	Ir
-----	------	--------	-------------	----------	--------	----

127	10.534906	134.108.8.37	134.108.8.36	TCP	74	55002 → 9001 [SYN] S
128	10.534950	134.108.8.36	134.108.8.37	TCP	74	9001 → 55002 [SYN, A
129	10.535157	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
215	20.989058	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [FIN, A
216	20.989127	134.108.8.36	134.108.8.37	TCP	66	9001 → 55002 [ACK] S
597	66.007860	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [ACK]
598	66.007902	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [PSH,
599	66.008144	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
600	66.008168	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [ACK]
601	66.008176	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [PSH,
602	66.008360	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
603	66.008374	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [ACK]
604	66.008380	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [PSH,
605	66.008591	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
606	66.008603	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [ACK]
607	66.008608	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [PSH,
608	66.008822	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
609	66.008833	134.108.8.36	134.108.8.37	TCP	1514	9001 → 55002 [ACK]
610	66.008839	134.108.8.36	134.108.8.37	TCP	1034	9001 → 55002 [PSH,
611	66.009049	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S
620	67.008075	134.108.8.36	134.108.8.37	TCP	66	9001 → 55002 [FIN, A
621	67.008275	134.108.8.37	134.108.8.36	TCP	66	55002 → 9001 [ACK] S

Aufabe 1.4

Ohne Fehlerbehandlung

Hier wird gezeigt, wie mit bind eine Portnummer, über die man senden/empfangen kann, an den Socket gebunden. Das bind liefert als Rückgabewert bei erfolgreichem binden eine 0, ansonsten -1.

929	0.000000	134.108.8.37	134.108.8.36	TCP	74	9002
930	0.000054	134.108.8.36	134.108.8.37	TCP	74	9001
931	0.000204	134.108.8.37	134.108.8.36	TCP	66	9002
1186	38.375580	134.108.8.37	134.108.8.36	TCP	74	5500
1187	0.000047	134.108.8.36	134.108.8.37	TCP	74	9001
1188	0.000190	134.108.8.37	134.108.8.36	TCP	66	5500
1440	23.796940	134.108.8.37	134.108.8.36	TCP	69	9002
1441	0.000034	134.108.8.36	134.108.8.37	TCP	66	9001
1442	0.000009	134.108.8.37	134.108.8.36	TCP	66	9002
1444	0.039504	134.108.8.36	134.108.8.37	TCP	66	9001
1518	7.840354	134.108.8.37	134.108.8.36	TCP	69	5500
1519	0.000035	134.108.8.36	134.108.8.37	TCP	66	9001
1520	0.000009	134.108.8.37	134.108.8.36	TCP	66	5500
1521	0.039615	134.108.8.36	134.108.8.37	TCP	66	9001
1775	23.666174	134.108.8.36	134.108.8.37	TCP	66	9001
1776	0.000196	134.108.8.37	134.108.8.36	TCP	66	9002
1865	8.703774	134.108.8.36	134.108.8.37	TCP	66	9001
1866	0.000242	134.108.8.37	134.108.8.36	TCP	66	5500

Mit Fehlerbehandlung

798	0.000000	134.108.8.37	134.108.8.36	TCP	74	9002 → 9001 [SYN] Seq=
799	0.000054	134.108.8.36	134.108.8.37	TCP	74	9001 → 9002 [SYN, ACK]
800	0.000165	134.108.8.37	134.108.8.36	TCP	66	9002 → 9001 [ACK] Seq=



Konsolenausgabe

```
[rn-lab3105@itpc9010 ~]$ CLIENT: Version: 1.3 ; Autor: H.Ws
CLIENT: Server-Port = 9001
CLIENT: addr = 0.0.0.0 ; Gebundener Port = 9002
CLIENT: Fehler bei (bind), Return-Code = -1
CLIENT: Fehler bei bind, fester Port belegt: Address already in use
```

Aufgabe 1.5

Der Nebenläufiger Server kann mehrere Clients zugleich bedienen. Dies kann man im Wireshark trace sehen, da Client B und Client A zugleich bedient werden.

Client A:

```
kabeit00@itpc3105 tcp_v11_ws17]$ ./client_n 134.108.8.37
CLIENT: Version: 1.3 ; Autor: H.Ws
CLIENT: server_port = 9001
CLIENT: Verbindung mit Server 134.108.8.37 auf Port 9001 aufgenommen

CLIENT: Bitte beliebiges Zeichen eingeben, damit zum Server geschrieben wird!a
CLIENT: Anzahl geschriebener Zeichen in write = 3000
CLIENT: Nachricht vom Server: Ihre IP-Adresse lautet '134.108.8.36'
      Ihre Port-Nummer lautet '48376      ...
CLIENT: Anzahl gelesener Zeichen in read = 2000
CLIENT: Bitte beliebiges Zeichen eingeben, damit zum Server geschrieben wird!a
CLIENT: Anzahl geschriebener Zeichen in write = 3000
CLIENT: Nachricht vom Server: Ihre IP-Adresse lautet '134.108.8.36'
      Ihre Port-Nummer lautet '48376      ...
CLIENT: Anzahl gelesener Zeichen in read = 2000
```

Client B:

```
[kabeit00@itpc3105 tcp_v11_ws17]$ ./client_n 134.108.8.37
CLIENT: Version: 1.3 ; Autor: H.Ws
CLIENT: server_port = 9001
CLIENT: Verbindung mit Server 134.108.8.37 auf Port 9001 aufgenommen

CLIENT: Bitte beliebiges Zeichen eingeben, damit zum Server geschrieben wird!k
CLIENT: Anzahl geschriebener Zeichen in write = 3000
```



```

CLIENT: Nachricht vom Server: Ihre IP-Adresse lautet '134.108.8.36'
      Ihre Port-Nummer lautet '48378      ...
CLIENT: Anzahl gelesener Zeichen in read = 2000
CLIENT: Bitte beliebiges Zeichen eingeben, damit zum Server geschrieben wird!a
CLIENT: Anzahl geschriebener Zeichen in write = 3000
CLIENT: Nachricht vom Server: Ihre IP-Adresse lautet '134.108.8.36'
      Ihre Port-Nummer lautet '48378      ...
CLIENT: Anzahl gelesener Zeichen in read = 2000

```

Server:

```

Server:[kabeit00@itpc3105 tcp_v11_ws17]$ ./server_n
SERVER_N: PID = 19209 : Nebenlaeufiger Server, Version: 1.3 ; Autor: H.Ws
SERVER_N 17:42:33.5 > PID = 19209 : server_port = 9001

SERVER_N 17:44:07.7 > PID = 19209 ; Parent-Socket = 3 : Mit Client 127.0.0.1 auf Port

SERVER_N 17:44:07.7 > PID = 19231 : Local socket in child = 4

SERVER_N 17:44:12.1 > PID = 19209 ; Parent-Socket = 3 : Mit Client 127.0.0.1 auf Port

SERVER_N 17:44:12.1 > PID = 19235 : Local socket in child = 4

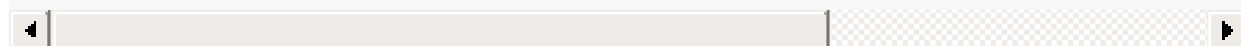
SERVER_N 17:44:14.2 > PID = 19235 : clientport 33038 : Nachricht von Client: Aabcdefg
SERVER_N 17:44:14.2 > PID = 19235 : clientport 33038 : Anzahl gelesener Zeichen in re
SERVER_N 17:44:14.2 > PID = 19235 : clientport 33038 : Anzahl geschriebener Zeichen i

SERVER_N 17:44:16.5 > PID = 19231 : clientport 33036 : Nachricht von Client: Aabcdefg
SERVER_N 17:44:16.5 > PID = 19231 : clientport 33036 : Anzahl gelesener Zeichen in re
SERVER_N 17:44:16.5 > PID = 19231 : clientport 33036 : Anzahl geschriebener Zeichen i

SERVER_N 17:44:17.8 > PID = 19231 : clientport 33036 : Nachricht von Client: Aabcdefg
SERVER_N 17:44:17.8 > PID = 19231 : clientport 33036 : Anzahl gelesener Zeichen in re
SERVER_N 17:44:17.8 > PID = 19231 : clientport 33036 : Anzahl geschriebener Zeichen i
SERVER_N 17:44:18.8 > PID = 19231 : Verbindung mit Client 127.0.0.1 auf Port 33036 be

SERVER_N 17:44:18.5 > PID = 19235 : clientport 33038 : Nachricht von Client: Aabcdefg
SERVER_N 17:44:18.5 > PID = 19235 : clientport 33038 : Anzahl gelesener Zeichen in re
SERVER_N 17:44:18.5 > PID = 19235 : clientport 33038 : Anzahl geschriebener Zeichen i
SERVER_N 17:44:19.5 > PID = 19235 : Verbindung mit Client 127.0.0.1 auf Port 33038 be

```



Wireshark trace:

2244	0.000000	134.108.8.36	134.108.8.37	TCP	74	48
2245	0.000199	134.108.8.37	134.108.8.36	TCP	74	96
2246	0.000043	134.108.8.36	134.108.8.37	TCP	66	48
2287	5.998053	134.108.8.36	134.108.8.37	TCP	74	48
2288	0.000221	134.108.8.37	134.108.8.36	TCP	74	96
2289	0.000032	134.108.8.36	134.108.8.37	TCP	66	48
2526	25.934500	134.108.8.36	134.108.8.37	TCP	2962	48
2527	0.000019	134.108.8.36	134.108.8.37	TCP	170	48
2528	0.000245	134.108.8.37	134.108.8.36	TCP	66	96

2529	0.000213	134.108.8.37	134.108.8.36	TCP	2066	96
2530	0.000045	134.108.8.36	134.108.8.37	TCP	66	48
2592	10.327453	134.108.8.36	134.108.8.37	TCP	2962	48
2593	0.000015	134.108.8.36	134.108.8.37	TCP	170	48
2594	0.000252	134.108.8.37	134.108.8.36	TCP	66	96
2595	0.000236	134.108.8.37	134.108.8.36	TCP	2066	96
2596	0.000062	134.108.8.36	134.108.8.37	TCP	66	48
2717	12.231667	134.108.8.36	134.108.8.37	TCP	2962	48
2718	0.000016	134.108.8.36	134.108.8.37	TCP	170	48
2719	0.000315	134.108.8.37	134.108.8.36	TCP	66	96
2720	0.000112	134.108.8.37	134.108.8.36	TCP	2066	96
2721	0.000064	134.108.8.36	134.108.8.37	TCP	66	48
2722	0.000040	134.108.8.36	134.108.8.37	TCP	69	48
2723	0.000022	134.108.8.36	134.108.8.37	TCP	66	48
2724	0.038853	134.108.8.37	134.108.8.36	TCP	66	96
2734	0.961439	134.108.8.37	134.108.8.36	TCP	66	96
2735	0.000036	134.108.8.36	134.108.8.37	TCP	66	48
2756	1.887127	134.108.8.36	134.108.8.37	TCP	2962	48
2757	0.000021	134.108.8.36	134.108.8.37	TCP	170	48
2758	0.000223	134.108.8.37	134.108.8.36	TCP	66	96
2759	0.000168	134.108.8.37	134.108.8.36	TCP	2066	96
2760	0.000046	134.108.8.36	134.108.8.37	TCP	66	48
2761	0.000038	134.108.8.36	134.108.8.37	TCP	69	48
2762	0.000016	134.108.8.36	134.108.8.37	TCP	66	48
2763	0.038880	134.108.8.37	134.108.8.36	TCP	66	96
2774	0.961381	134.108.8.37	134.108.8.36	TCP	66	96
2775	0.000035	134.108.8.36	134.108.8.37	TCP	66	48

Aufgabe 2.1

Erklären Sie den Ablauf bei UDP:

Bei UDP müssen die Ports mitgesendet werden, da UDP ein verbindungsloses Protokoll ist (kein SYN,ACK/FIN,ACK) wie bei TCP.

Was ist anders zu TCP?

Bei UDP wird eine Checksummenprüfung gemacht, ob das Paket beim Empfänger angekommen ist, wird nicht überprüft. Im folgenden sieht man den Datenaustausch über UDP, wobei im ersten Paket die Portnummern übertragen werden und anschließend die Daten.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
2	0.000009	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
3	0.000012	134.108.8.36	134.108.8.37	UDP	82	9005
4	0.000032	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
5	0.000036	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
6	0.000038	134.108.8.36	134.108.8.37	UDP	82	9005
7	8.320396	134.108.8.37	134.108.8.36	IPv4	1514	Fragment


8	8.320413	134.108.8.37	134.108.8.36	IPv4	1514	Fragment
9	8.320417	134.108.8.37	134.108.8.36	UDP	82	9005



Aufgabe 2.2

Hier ist das Gleiche zu beobachten. Als erstes werden die Port Nummern übertragen und anschließend die Daten in diesem Fall fragmentiert.

No.	Time	Source	Destination	Protocol	Length	Ir
74	12.783421	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
75	12.783433	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
76	12.783436	134.108.8.36	134.108.8.37	UDP	82	9005
77	12.783458	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
78	12.783461	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
79	12.783463	134.108.8.36	134.108.8.37	UDP	82	9005



Time to live = 1 Wenn TTL auf 1 gesetzt wird, so werden die Daten beim Router verworfen und der Router sendet ein Time to live exceeded zurück.

68	3.863807	134.108.8.36	39.135.17.38	ICMP	82	Destination
74	1.659959	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
75	0.000012	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
76	0.000003	134.108.8.36	134.108.8.37	UDP	82	9005
77	0.000022	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
78	0.000003	134.108.8.36	134.108.8.37	IPv4	1514	Fragment
79	0.000002	134.108.8.36	134.108.8.37	UDP	82	9005

