

**Aufgabe 1:**

Zunächst gehe ich auf die Funktionsweise von *nc\_tcp* ein.

Mit *-l 2000* starte ich den Server an Port 3000 und starte daraufhin einen Clienten mit meiner IP-Adresse und dem Port 3000. In der IDE sieht es dann wie folgt aus:

```
C:\Users\jweic\IntelliJ_Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe1.nc_tcp -l 3000
Das ist ein Test.
Ein zweiter Test.
Der letzte Test.
stop
Process finished with exit code 0
```

Mit der Eingabe *stop* wird der Client beendet.

In WireShark habe ich diese Pakete wiedergefunden, hier am Beispiel meiner zweiten Nachricht:

11	2.969614	192.168.178.175	192.168.178.175	TCP	63	39714 → 3000	[PSH, ACK] Seq=1 Ack=1 Win=10233 Len=19
12	2.969645	192.168.178.175	192.168.178.175	TCP	44	3000 → 39714	[ACK] Seq=1 Ack=20 Win=10233 Len=0

  

Frame 11: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface \Device\NPF_{loopback}, id 0		0000	02 00 00 00 45 00 00 3b	62 33 40 00 00 00 00 00	... E ; b3g ...		
Null/Loopback		0010	c0 a8 b2 af c0 a8 b2 af	9b 22 0b b8 5c 33 58 9a	... .. \3X		
Internet Protocol Version 4, Src: 192.168.178.175, Dst: 192.168.178.175		0020	f1 ba 2c 05 50 18 27 f9	ad 00 00 00 45 69 6e 2c	... P . . . Fin		
Transmission Control Protocol, Src Port: 39714, Dst Port: 3000, Seq: 1, Ack: 1, Len: 19		0030	7a 77 65 69 74 65 72 20	54 65 73 74 2e 0d 0a	... 2weiter Test. ...		
Data (19 bytes)							
Data: 45696e207a77656974657220546573742e0d0a							
[Length: 19]							

Bevor jedoch Textpakete gesendet werden können, führt TCP den 3-way-handshake zum Verbindungsaufbau durch. Dabei meldet sich der Client beim Server mittels [SYN], dieser erhält eine positive Antwort mittels [SYN, ACK] und daraufhin wird [ACK] wieder an den Server gesendet und es besteht eine feste Verbindung. In der nachfolgenden Abbildung ist dies bei den rot hinterlegten Einträgen zu erkennen:

148	45.067249	192.168.178.175	192.168.178.175	TCP	56	39868 → 3000	[SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
149	45.067249	127.0.0.1	127.0.0.1	TCP	56	39869 → 39867	[SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
150	45.067302	127.0.0.1	127.0.0.1	TCP	56	39867 → 39869	[SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
151	45.067332	192.168.178.175	192.168.178.175	TCP	56	3000 → 39868	[SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
152	45.067334	127.0.0.1	127.0.0.1	TCP	44	39869 → 39867	[ACK] Seq=1 Ack=1 Win=2619648 Len=0
153	45.067372	192.168.178.175	192.168.178.175	TCP	44	39869 → 3000	[ACK] Seq=1 Ack=1 Win=2619648 Len=0

Im Folgenden wird nun *nc\_udp* vorgestellt.

Hier handelt es sich um die gleichen Befehle wie bei *nc\_tcp*, auch die gleichen Ports sowie nahezu identischen Nachrichten:

```
C:\Users\jweic\IntelliJ_Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe1.nc_udp -l 3000
Das ist ein Test.
Ein zweiter Test.
Ein dritter Test.
stop
Process finished with exit code 0
```

Folgende Datenpakete habe ich mit WireShark aufgefangen:

3510	1749.822947	192.168.178.175	192.168.178.175	DIS	49	PDType: 115	\t Unknown
3515	1755.497145	192.168.178.175	192.168.178.175	DIS	57	PDType: 115	\t Unknown
3524	1758.561547	192.168.178.175	192.168.178.175	DIS	49	PDType: 110	\t Unknown
3527	1760.011067	192.168.178.175	192.168.178.175	UDP	36	57441 → 3000	Len=4

Hier wird DIS als Protokoll genutzt, welches auf UDP aufbaut. Wird der erste Eintrag näher betrachtet, so sind die beteiligten Ports sowie die Nachrichten erkennbar:

Frame 3510: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface \Device\NPF_{...}, id 0		0000	02 00 00 00 45 00 00 2d	62 98 00 00 00 11 00 00	... E ; b ...	
Null/Loopback		0010	c0 a8 b2 af c0 a8 b2 af	e0 61 0b b8 80 19 ce 78	... .. 0 ...X	
Internet Protocol Version 4, Src: 192.168.178.175, Dst: 192.168.178.175		0020	f1 ba 2c 05 50 18 27 f9	ad 00 00 00 45 69 6e 2c	... P . . . Fin	
User Datagram Protocol, Src Port: 57441, Dst Port: 3000		0030	7a 77 65 69 74 65 72 20	54 65 73 74 2e 0d 0a	... 2weiter Test. ...	
Distributed Interactive Simulation						

Der Hauptunterschied zwischen UDP und TCP ist, dass TCP den 3-way-handshake durchführt, um eine Verbindung aufzubauen. UDP benötigt dies nicht, stellt aber auch nicht sicher, dass die Pakete ankommen. In den gezeigten Beispielen benötigt TCP mehr Datenmengen um die Kommunikation durchzuführen, UDP ist für diesen Testzweck somit besser geeignet, da hier verlustfrei Pakete mit weniger Datenlast gesendet werden können.

## Aufgabe 2:

Um das Programm zu starten werden 2 Instanzen von *UDP* gestartet, in diesem Falle mit den Parametern *<name> <port>*, also *Franz 6000* und *Hans 6001*. Sind diese gestartet, so können Hans und Franz sich gegenseitig registrieren:

```
Enter command (register/send/exit): register Franz 127.0.0.1 6000
Enter command (register/send/exit): Received message: Hello, this is Franz, my IP is 192.168.56.1 and
you can reach me on port 6000.

Enter command (register/send/exit): register Hans 127.0.0.1 6001
Enter command (register/send/exit): Received message: Hello, this is Hans, my IP is 192.168.56.1 and
you can reach me on port 6001.
```

Das in WireShark zugehörige Protokoll bei der Registrierung:

```
278.11995.5681- 127.0.0.1 127.0.0.1 UDP 109 6001 → 6000 Len=77
Frame 27817: 109 bytes on wire (872 bits), 109 bytes captured (872 bits) on interface \Device\NPF_{...}
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 6001, Dst Port: 6000
Data (77 bytes)
48616e733a2048616e732c206479204950206977203139322e3136382e35362e3120616e64207965752063616e207265616368206465206465
[Length: 77]
```

Jetzt können zwischen Hans und Franz Nachrichten gesendet werden (wichtig ist, dass die Registrierung in beide Richtungen vorgenommen wird, da sonst nur eine einseitige Kommunikation möglich ist):

```
Enter command (register/send/exit): send Franz Hallo Hans.
Enter command (register/send/exit): Received message: Franz: Hallo ebenso Hans.

Enter command (register/send/exit): send Hans Hallo Franz.
Enter command (register/send/exit): Received message: Hans: Hallo Franz.
```

Ebenso können mehr als 2 Teilnehmer chatten, jedoch nicht in einer Gruppe, sondern im *Privatchat*. Auch hier muss zunächst die Registrierung erfolgen. Während Hans zuvor noch mit Franz geredet hat, kann er jetzt auch mit Peter reden:

```
Enter command (register/send/exit): send Franz Hallo Hans.
Enter command (register/send/exit): Received message: Franz: Hallo ebenso Hans.
register Peter 127.0.0.1 6002
Enter command (register/send/exit): Received message: Hello, this is Peter, my IP is 192.168.56.1 and
you can reach me on port 6002.
send Peter Hallo Peter: Ich habe gerade mit Franz geredet.
Enter command (register/send/exit): Received message: Hans: Hallo Peter, ich habe gerade mit Franz
geredet.
```

Das dazugehörige WireShark Protokoll der letzten Nachricht:

```
283.12355.8817- 127.0.0.1 127.0.0.1 UDP 85 6001 → 6002 Len=53
Frame 28313: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface \Device\NPF_{...}
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 6001, Dst Port: 6002
Data (53 bytes)
48616e733a2048616e732c206479204950206977203139322e3136382e35362e3120616e64207965752063616e207265616368206465206465
[Length: 53]
```

## Aufgabe 3:

Zu Beginn wird zunächst ein Server gestartet, dieser erhält ausschließlich das Argument *-l <port>*. Daraufhin können beliebig viele Clients gestartet werden, diese erhalten als Eingabe die Argumente *<ip> <port>*, wobei der Port dem Port des Servers übereinstimmen muss. In der IDE sieht dies wie folgt aus:

```
C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP -l 3000
Server started on port 3000

C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP 127.0.0.1 3000
Enter your name:
```

Die Registrierung eines Clienten mit dem Server mittels 3-fachem Handshake sieht so aus:

31080	13592.3465...	127.0.0.1	127.0.0.1	TCP	56 5033 → 3000	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
31081	13592.3465...	127.0.0.1	127.0.0.1	TCP	56 3000 → 5033	[SYN, ACK]	Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
31082	13592.3465...	127.0.0.1	127.0.0.1	TCP	56 5034 → 5032	[SYN]	Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
31083	13592.3466...	127.0.0.1	127.0.0.1	TCP	44 5033 → 3000	[ACK]	Seq=1 Ack=1 Win=2619648 Len=0

Nun muss nur noch der Name für den Clienten eingegeben werden, wir wählen wieder Franz:

```
Enter your name:
Franz
Registered as Franz
```

Es können beliebig viele weitere Clienten gestartet werden, der Vorteil gegenüber der UDP-Variante ist, dass hier keine Registrierung in jedem Clienten erfolgen muss, da die Kommunikation über den Server läuft. Gibt es einen Clienten namens Hans, so kann direkt in beide Richtungen gechattet werden:

```

C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP 127.0.0.1 3000
Enter your name:
Hans
Registered as Hans
send Franz Hallo Franz, ich bin es wieder.
Franz: Hallo Hans, schön, dass du wieder da bist.

C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP 127.0.0.1 3000
Enter your name:
Franz
Registered as Franz
Hans: Hallo Franz, ich bin es wieder.
send Hans Hallo Hans, schön, dass du wieder da bist.

```

In WireShark sieht die letzte Nachricht wie folgt aus:

```

319_14211.6551_ 127.0.0.1 127.0.0.1 TCP 99 5033 → 3000 [PSH, ACK] Seq=8 Ack=80 Win=2619648 Len=55
Frame 31994: 99 bytes on wire (792 bits), 99 bytes captured (792 bits) on interface \Device\NPF_{...} id 0
Null/loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 5033, Dst Port: 3000, Seq: 8, Ack: 80, Len: 55
Data (55 bytes)
Data: 73656642048616e732048616e732020736368c3b66e2c206461737320647520776965646572206461206269737420d8a
[Length: 55]

```

Auch hier ist es möglich, mehrere Privatchats parallel laufen zu lassen:

```

C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP 127.0.0.1 3000
Enter your name:
Peter
Registered as Peter
send Franz Hallo Franz, ich bin der Peter und bin auch wieder dabei.

C:\Users\jweich\IntelliJ\Projekte\UDP-TCP-Chat\out\production\UDP-TCP-Chat Aufgabe3.TCP 127.0.0.1 3000
Enter your name:
Franz
Registered as Franz
Hans: Hallo Franz, ich bin es wieder.
send Hans Hallo Hans, schön, dass du wieder da bist.
Peter: Hallo Franz, ich bin der Peter und bin auch wieder dabei.

```

#### Aufgabe 4:

