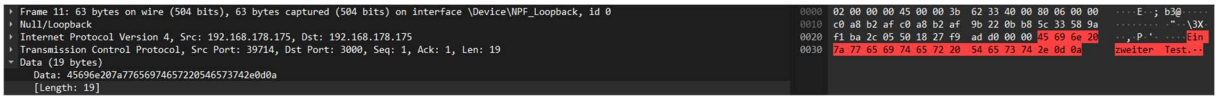
Aufgabe 1:

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:

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



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:

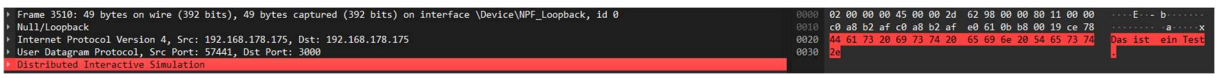
148	45.067249	192.168.178.175	192.168.178.175	TCP	56 39868 → 3000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
149	45.067249	127.0.0.1	127.0.0.1	TCP	56 39869 → 39867 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
150	45.067302	127.0.0.1	127.0.0.1	TCP	56 39867 → 39869 [SYN, ACK] Seq=0 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 39868 → 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:

Folgende Datenpakete habe ich mit WireShark aufgefangen:

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

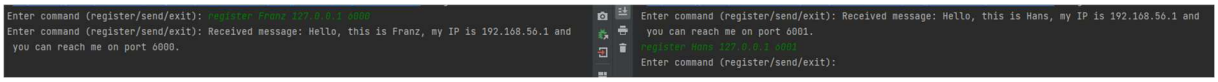
Hier wird DIS als Protokoll genutzt, welches auf UDP aufbaut. Wird der erste Eintrag näher betrachtet, so sind die beteiligten Ports sowie die Nachrichten erkennbar:



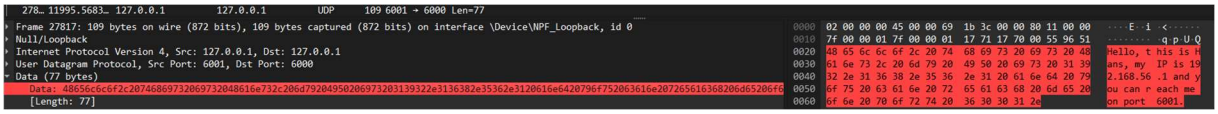
Der Hauptunterschied zwischen UDP und TCP ist, dass TCP den 3-way-handshake durchführt, um eine Verbindung aufzubauen. UDP benötigt dies nicht, stellt aber auch nicht sicher, dass die Pakete ankommen. In den gezeigten Beispielen benötigt TCP mehr Datenmengen um die Kommunikation durchzuführen, UDP ist 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:



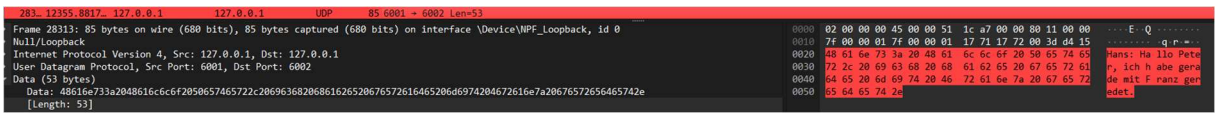
Das in WireShark zugehörige Protokoll bei der Registrierung:



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):

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:

Das dazugehörige WireShark Protokoll der letzten Nachricht:



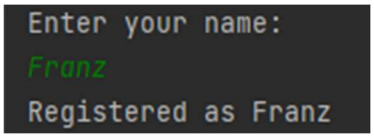
Aufgabe 3:

Zu Beginn wird zunächst ein Server gestartet, dieser erhält ausschließlich das Argument *-l* <*port*>. Daraufhin können beliebig viele Clienten 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:

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

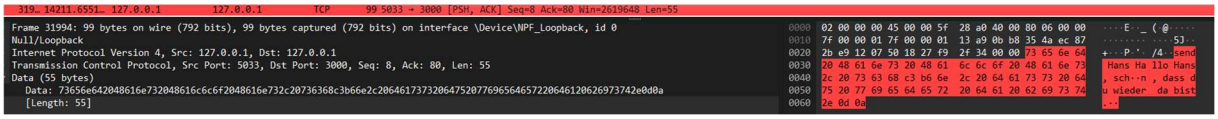
31080	13592.3465…	127.0.0.1	127.0.0.1	TCP	56 5033 → 3000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
31081	13592.3465…	127.0.0.1	127.0.0.1	TCP	56 3000 → 5033 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
31082	13592.3465…	127.0.0.1	127.0.0.1	TCP	56 5034 → 5032 [SYN] Seq=0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

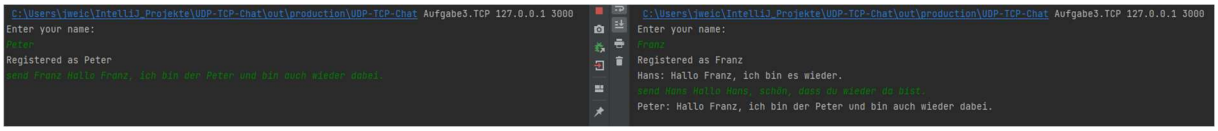


Es können beliebig viele weitere Clienten gestartet werden, der Vorteil gegenüber der UDP-Variante ist, dass hier keine Registrierung in jedem Clienten erfolgen muss, da die Kommunikation über den Server läuft. Gibt es einen Cienten namens Hans, so kann direkt in beide Richtungen gechattet werden:

In WireShark sieht die letzte Nachricht wie folgt aus:



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



Aufgabe 4: