

3. Hausaufgabe

Aufgabe 1: Wireshark

Machen Sie sich mit der Implementation der UDP- und TCP-Programme vertraut, die in der ersten Vorlesung vorgestellt wurden (die Klassen `nc_udp` und `nc_tcp`, auch zu finden im Repository <https://github.com/syssoft-ds/netcat>). Nutzen Sie Generische KI, um sich Codeabschnitte, die Ihnen unklar sind, erklären zu lassen. Implementieren Sie die beiden Programme und probieren Sie sie aus. Überlegen Sie sich, wie Sie damit Nachrichten schicken und empfangen können.

Nutzen Sie die Capture-Funktion von Wireshark, um Ihren Traffic aufzuzeichnen, während die Programme laufen (auch im Rahmen der nächsten beiden Übungen). Vergleichen Sie die Aufzeichnung des TCP-Programms, mit der Aufzeichnung des UDP-Programms.

Erläutern Sie die Unterschiede und Gemeinsamkeiten der Funktionsweisen der beiden Programme. Nehmen Sie dabei Bezug auf konkrete Pakete in Ihren beiden PCAP-Dateien.

Bei den Implementationen zu UDP und TCP habe ich mich für die **Python-Programme** entschieden, weil Python mir etwas besser liegt, als Java.

Erläuterungen des TCP-Programmes:

- *server*: Funktion enthält Logik für einen TCP-Server und nimmt Port als Argument
- *serveClient*: enthält Socket und IP eines verbundenen Clients und koordiniert Kommunikation mit einem einzelnen Client
- *client*: Funktion enthält Logik für einen TCP-Client und nimmt IP + Port als Argument
- *main*: Hauptfunktion, startet die TCP-Kommunikation mit 2 Argumenten
 - Servermodus: `-l <port>`
 - Clientmodus: `<IP-Adresse> <Port>`

Erläuterungen des UDP-Programmes:

- *receiveLines*: Funktion um UDP-Nachrichten zu empfangen, erhält Port
- *sendLines*: Funktion, um UDP-Nachrichten zu senden, erhält IP-Adresse + Port
- *main*: Hauptfunktion, startet die TCP-Kommunikation mit 2 Argumenten
 - Servermodus: `-l <port>`
 - Clientmodus: `<IP-Adresse> <Port>`

Nachrichten schicken/empfangen:

Nachrichten können durch Initialisierung eines Servermodus und eines Clientmodus jeweils für TCP und UDP gesendet und empfangen werden.

Beispiel TCP (siehe TCP.pcapng):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.178.174	192.168.178.174	TCP	56	50424 → 54321 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000092	192.168.178.174	192.168.178.174	TCP	56	54321 → 50424 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
3	0.000133	192.168.178.174	192.168.178.174	TCP	44	50424 → 54321 [ACK] Seq=1 Ack=1 Win=327424 Len=0
4	2.097434	192.168.178.174	192.168.178.174	TCP	49	50424 → 54321 [PSH, ACK] Seq=1 Ack=1 Win=327424 Len=5
5	2.097471	192.168.178.174	192.168.178.174	TCP	44	54321 → 50424 [ACK] Seq=1 Ack=6 Win=2161152 Len=0
6	4.273502	192.168.178.174	192.168.178.174	TCP	48	50424 → 54321 [PSH, ACK] Seq=6 Ack=1 Win=327424 Len=4
7	4.273539	192.168.178.174	192.168.178.174	TCP	44	54321 → 50424 [ACK] Seq=1 Ack=10 Win=2161152 Len=0
8	16.889484	192.168.178.174	192.168.178.174	TCP	108	50424 → 54321 [PSH, ACK] Seq=10 Ack=1 Win=327424 Len=64
9	16.889523	192.168.178.174	192.168.178.174	TCP	44	54321 → 50424 [ACK] Seq=1 Ack=74 Win=2161152 Len=0
10	29.289562	192.168.178.174	192.168.178.174	TCP	88	50424 → 54321 [PSH, ACK] Seq=74 Ack=1 Win=327424 Len=44
11	29.289592	192.168.178.174	192.168.178.174	TCP	44	54321 → 50424 [ACK] Seq=1 Ack=118 Win=2161152 Len=0

Beispiel UDP (siehe UDP.pcapng):

No.	Time	Source	Destination	Protocol	Length	Info
7	2.085333	192.168.178.174	192.168.178.174	UDP	36	64275 → 54321 Len=4
11	10.077457	192.168.178.174	192.168.178.174	UDP	73	64275 → 54321 Len=41
12	21.733084	192.168.178.174	192.168.178.174	UDP	53	64275 → 54321 Len=21
25	33.365130	192.168.178.174	192.168.178.174	UDP	85	64275 → 54321 Len=53

Gemeinsamkeiten:

- beide Programme lassen sich in einem Server- oder Clientmodus starten, welcher dieselbe Anzahl an Argumenten benötigt
- beide Programme können durch den Befehl „stop“ beendet werden
- beide Programme geben die empfangenen Nachrichten in der Konsole aus

Unterschiede:

- jedes Programm arbeitet mit einem unterschiedlichen Protokoll (**TCP** vs. **UDP**)
- das TCP-Programm wartet auf eine eingehende Verbindung und muss diese akzeptieren, während das UDP-Programm keine Verbindung aufbaut, sondern nur auf eingehende Pakete wartet (**verbindungsorientiert** vs. **verbindungslos**)
- TCP verwendet Threading in *def server(port)* um mehrere Verbindungen zu koordinieren, worauf bei UDP verzichtet wird, hier werden Nachrichten nach und nach abgearbeitet

Bezug zu Paketen:

- beim TCP-Programm sieht man die Anfrage und das Akzeptieren einer Verbindung anhand der ersten drei Pakete in der TCP.pcapng Datei, während beim UDP-Programm von Anfang an nur die gesendeten Nachrichten protokolliert wurden
- beim TCP-Programm wird nach jeder Nachricht eine Empfangsbestätigung versendet, deshalb haben wir auch doppelt so viel Pakete wie in der UDP.pcapng Datei

Aufgabe 2: UDP

Nehmen sie nc_udp als Ausgangspunkt, und bauen Sie es zu einem Chatprogramm um. Jede Instanz des Programms soll einen Namen haben und sich bei einer anderen Instanz registrieren können (also so etwas wie „Hallo, hier ist Marvin, meine IP-Adresse ist die 192.168.0.42 und du kannst mich unter Port-Nummer 31337 erreichen.“). Anschließend sollen die Instanzen, die sich kennen, über einen Befehl „send name message“ sich gegenseitig Nachrichten senden können (name für den Ansprechpartner, message für die Nachricht).

Sicht von User1:

```
Eingabeaufforderung - python
C:\Users\Andi\Desktop\3. HA\A2>python UDP-Chat.py User1 12345
> register 192.168.178.174 23456
Registrierungsanfrage an 192.168.178.174:23456 gesendet.
> 'User2' hat sich registriert: 192.168.178.174:23456
send User2 Hallo
Nachricht an 'User2' gesendet: Hallo
> Nachricht von 'User2': Hallo zurück
peers
Bekannte Peers:
- User2: 192.168.178.174:23456
>
```

Sicht von User2:

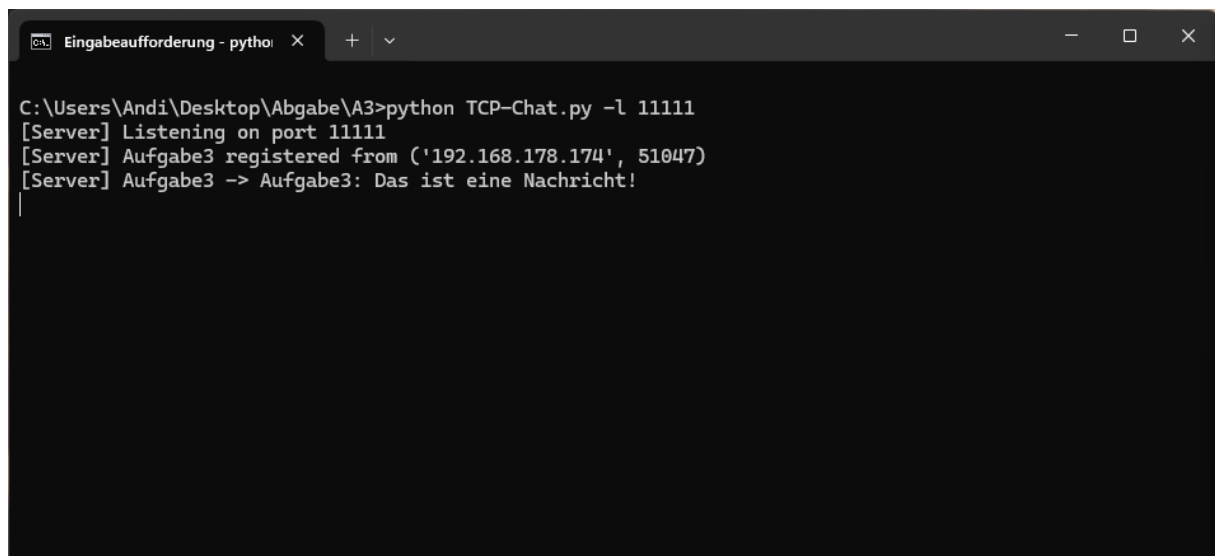
```
Eingabeaufforderung - python
C:\Users\Andi\Desktop\3. HA\A2>python UDP-Chat.py User2 23456
> 'User1' hat sich registriert: 192.168.178.174:12345
Nachricht von 'User1': Hallo
send User1 Hallo zurück
Nachricht an 'User1' gesendet: Hallo zurück
> peers
Bekannte Peers:
- User1: 192.168.178.174:12345
> |
```

Mit dem Befehl „peers“ können zu dem alle aktiven Verbindungen angezeigt werden.

Aufgabe 3: TCP

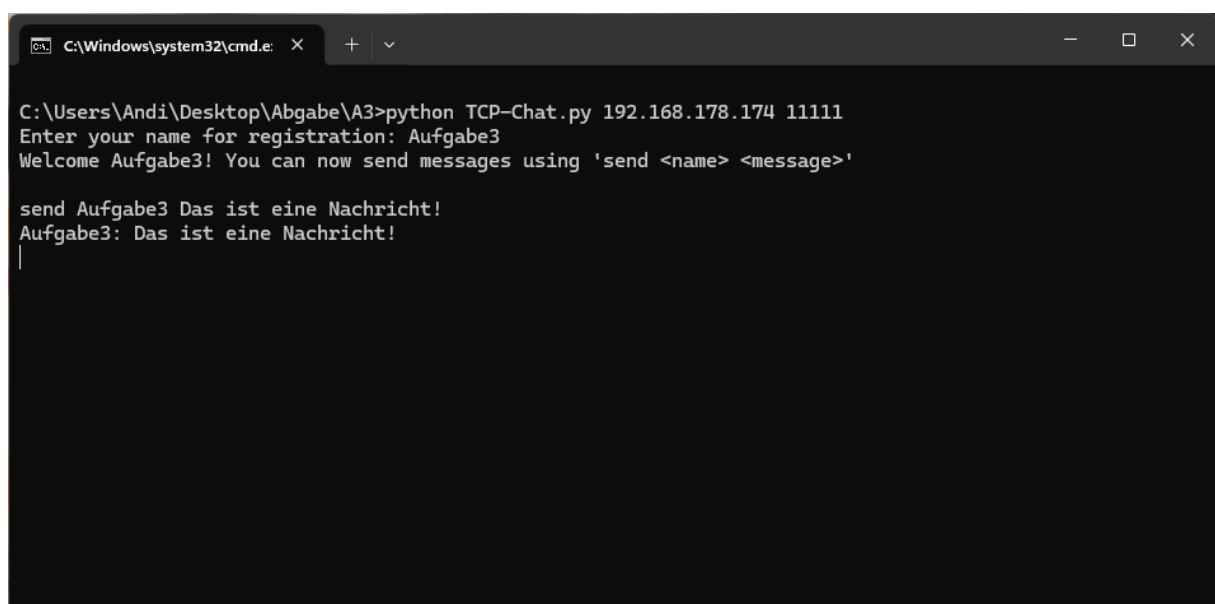
Verändern Sie nc_tcp ebenfalls zu einem Chatprogramm. Allerdings soll die Registrierung der Instanzen hier über den Server ablaufen. Nach der Registrierung soll es den Instanzen jedoch auch hier möglich sein, sich gegenseitig mit „send name message“ Nachrichten zu senden. Beachten Sie hier, dass bei TCP über die gesamte Dauer des Sendens und Empfangens eine Verbindung bestehen muss.

Sicht von Server:



```
Eingabeaufforderung - python X + v
C:\Users\Andi\Desktop\Abgabe\A3>python TCP-Chat.py -l 11111
[Server] Listening on port 11111
[Server] Aufgabe3 registered from ('192.168.178.174', 51047)
[Server] Aufgabe3 -> Aufgabe3: Das ist eine Nachricht!
```

Sicht von Client:

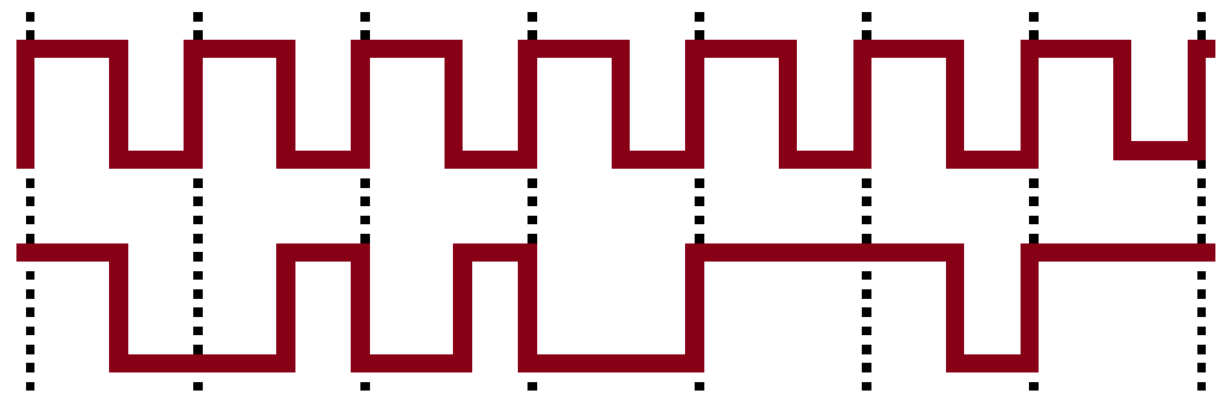


```
C:\Windows\system32\cmd.e X + v
C:\Users\Andi\Desktop\Abgabe\A3>python TCP-Chat.py 192.168.178.174 11111
Enter your name for registration: Aufgabe3
Welcome Aufgabe3! You can now send messages using 'send <name> <message>'

send Aufgabe3 Das ist eine Nachricht!
Aufgabe3: Das ist eine Nachricht!
```

Aufgabe 4: Programmieren

Machen Sie sich mit der Manchester-Codierung vertraut. Zeichnen Sie den daraus resultierenden Manchester-Code. Welche der beiden Definitionen des Codes Sie verwenden, ist Ihnen überlassen.

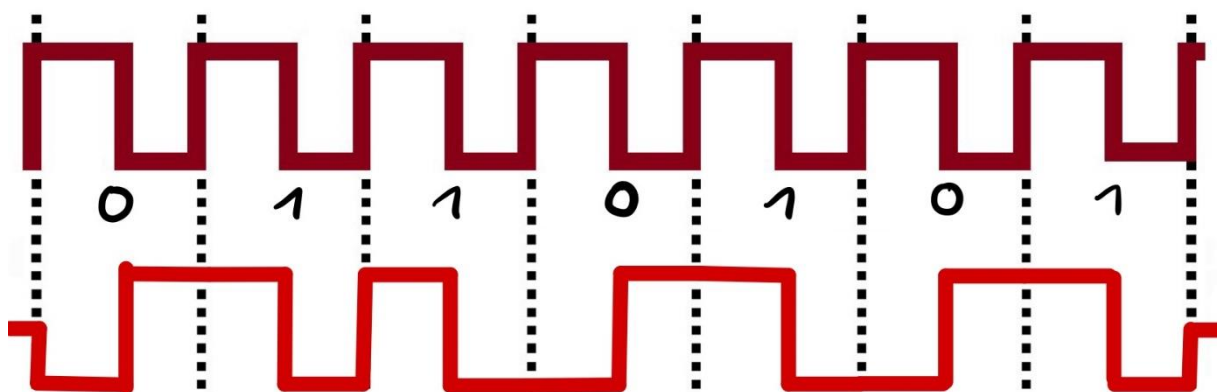


Oben ist die Clock, unten die zu codierende Bitfolge.

Der Code wurde nach der Definition von G.E. Thomas übersetzt, somit steht eine **fallende** Flanke für eine **logische 1** und eine **steigende** Flanke für eine **logische 0** steht.

Für die obige codierte Bitfolge erhalten wir folgende Bitfolge: 0110101

Übersetzt in den Manchestercode erhalten wir dann folgenden Graphen:



Manchester-Code

Da in dieser Aufgabenstellung „**Programmieren**“ steht, wir aber lediglich den Manchester-Code zeichnen sollen, habe ich noch ein Python-Programm geschrieben, welches für eine eingeegebene **Binärzahl** den Manchestercode in der Konsole zeichnet.