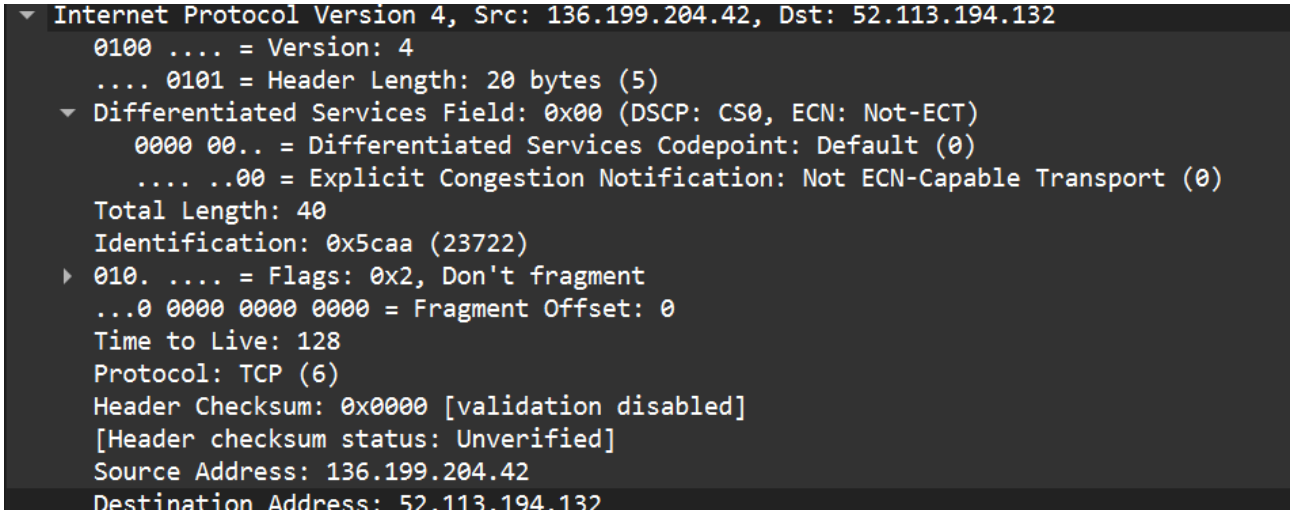
Übungsblatt 4

Aufgabe 1)

Ein Ipv4 Header eines Pakets aus dem Uni-Netz:



Da Wireshark bereits „übersetzt“, kann man einfach die Kategorien des Ipv4 Headers heraus lesen:

Version: 0100 → Version 4: Ein Ipv4 Paket.

Header length: 0101 → 20 bytes

Type of service (Differentiated Services Field): 0x00 → die Default-Option

Total Length: 40 bytes

Identification: 0x5caa

f: 010 → Don´t fragment

Fragment Offset: 0 0000 0000 0000 → kein Offset

Time to Live: 128

Protocol: 06 → TCP

Header Checksum: 0x0000 → keine Überprüfung auf Richtigkeit

Source Address: 136.199.204.42 → Mein Notebook im Uni-Netz

Destination Address: 52.113.194.132 → Microsoft in Redmond, Washington, United States laut ipinfo.io

TCP Header aus dem oberen Paket:

```
Transmission Control Protocol, Src Port: 61908, Dst Port: 443, Seq: 1377, Ack: 31058,
    Source Port: 61908
    Destination Port: 443
    [Stream index: 16]
  ▸ [Conversation completeness: Complete, WITH_DATA (47)]
    [TCP Segment Len: 0]
    Sequence Number: 1377    (relative sequence number)
    Sequence Number (raw): 1377046778
    [Next Sequence Number: 1377    (relative sequence number)]
    Acknowledgment Number: 31058    (relative ack number)
    Acknowledgment number (raw): 1105172180
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x010 (ACK)
    Window: 517
    [Calculated window size: 132352]
    [Window size scaling factor: 256]
    Checksum: 0x4c02 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
```

Source Port: 61908

Destination Port: 443

Sequence Number: 1377046778

Acknowledgement Number: 1105172180

Header Length: 0101 → 20 bytes

Flags: 0x010 → Nur ACK Flag

Window Size: 517

TCP Checksum: 0x4c02

Urgent Pointer: 0

Optionen: Keine

```
Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·····A····]
```

UDP Header (Irgendein anderes Paket):

```
User Datagram Protocol, Src Port: 443, Dst Port: 59731
    Source Port: 443
    Destination Port: 59731
    Length: 43
    Checksum: 0x2fa4 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 7]
  ▸ [Timestamps]
    UDP payload (35 bytes)
```

Source Port: 443

Destination Port: 59731

UDP Length: 43

UDP Checksum: 0x2fa4

Aufgabe 2)

103.161.122.83 → Die Ipv4-Adresse des Gerätes

18 → Anzahl der Eins-Bits der Subnetzmaske

Daraus ergibt sich die Subnetzmaske: 11111111.11111111.11000000.00000000, oder in Dezimaldarstellung: 255.255.192.0

Die Broadcastadresse ergibt sich aus der ODER-Verknüpfung der Ipv4-Adresse und der negierten Subnetzmaske. Im konkreten Fall heißt das:

01100111.10100001.01111010.01010011 ODER
00000000.00000000.00111111.11111111

= 01100111.10100001.01111111.11111111 → 103.161.127.255 (Dezimal)

Die Netzwerkadresse hingegen ergibt sich aus der UND-Verknüpfung der Ipv4-Adresse und der Subnetzmaske:

01100111.10100001.01111010.01010011 UND
11111111.11111111.11000000.00000000

= 01100111.10100001.01000000.00000000 → 103.161.64.0 (Dezimal)

103.161.122.83 liegt im Netz 103.161.64.0

103.161.193.83/18 hat die gleiche Subnetzmaske wie 103.161.122.83/18, daher ergibt sich folgende Berechnung der Netzwerkadresse:

01100111.10100001.11000001.01010011 UND
11111111.11111111.11000000.00000000

= 01100111.10100001.11000000.00000000 → 103.161.192.0 (Dezimal)

103.161.193.83/18 und 103.161.122.83/18 liegen in unterschiedlichen Netzen.

Aufgabe 3)

Beim Test der Chats mit der Musterlösung sowie mit den Implementierungen kam es in nahezu jeder Konstellation zu Problemen. Dabei ist das größte Problem die unterschiedlichen Formatierungen der Nachrichten. Zum Beispiel wird in der Musterlösung für den UDP Chat beim starten des Chats von keiner Seite aus eine

IP-Adresse zum Starten des Programms benötigt, während in meiner Lösung, ähnlich zum netcatUDP Programm, welches als Vorlage gegeben wurde, beim zweiten Client die IP-Adresse des zu erreichenden Clients mit angegeben werden soll. Weiterhin besteht in der Musterlösung der register-“Befehl“ aus 3 Teilstrings, dem „register“, der IP-Adresse des Empfängers und dessen Port in genau dieser Reihenfolge, während in meiner Lösung 4 Teilstrings gebraucht werden und der Name des Empfängers als zweites vor der IP-Adresse und dem Port stehen. Das sorgt in Summe dafür, dass meine Implementierung mit dem Nachrichtenformat der Musterlösung nichts anfangen kann und anders herum genau so. Ähnliche Probleme gibt es auch beim TCP Chat zwischen meiner Lösung und der Musterlösung und der Lösung einiger Kommilitonen. Die Lösung dafür liegt in der Anpassung des Nachrichtenformats zu einer einheitlichen Formatierung in allen Lösungen. Dadurch könnte jedes Programm, die Nachrichten der anderen Programme empfangen und verarbeiten und Nachrichten an andere senden, mit denen auch die anderen Programme arbeiten können. Solange dies der Fall ist, kann theoretisch jedes Programm individuell mit den Nachrichten umgehen, wichtig ist nur, dass alle Nachrichten gemäß der einheitlichen Formatierung gesendet werden und ankommen.