NVS HÜ 1

Abgabe von David Pape, Matnr. 01634454

Durchführung

Zum Erzeugen einer 1MB großen Datei wird Python verwendet:

```
# ariez @ Davids-iMac in ~/Documents/Uni/S8/netze/ps1 on git:master x [16:43:21]
[$ echo 'print("a" * (10**6))' | python > 1mb

# ariez @ Davids-iMac in ~/Documents/Uni/S8/netze/ps1 on git:master x [16:43:24]
[$ l
Permissions Size User  Date Modified Name
.rw-r--r--  1.0M ariez  6 Mar 16:43  1mb
.rw-rw-rw-@ 169k ariez  6 Mar 14:50  nvs21.ps.blatt1-nma.pdf
.rw-r--r--   164 ariez  6 Mar 15:12  nvs21_nma_pape.md
```

Figure 1: Dateierzeugung

Bemerkung: es wird davon ausgegangen, dass 1MB $= 10^6$ Bytes. Für eine 1MiB ($= 2^{20}$ Bytes) große Datei müsste man `2**20` in den Befehl einsetzen. macOS zeigt Dateigrößen auf der Festplatte in MB an, während z.B. `scp` Dateigrößen in MiB anzeigt, daher ist im obigen Screenshot eine Größe von `1.0M` angezeigt, während im Output von `scp` (unten) `977KB` steht.

Die Datei wird zunächst mittels `scp` auf den Cosy SSH Server übertragen. Dann wird die Übertragung in die andere Richtung gestartet und überwacht:

```
# ariez @ Davids-iMac in ~/Documents/Uni/S8/netze/ps1 on git:master x [16:28:10]
[$ sudo tcpdump -i en0 src 141.201.2.44 > raw
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C7117 packets captured
10058 packets received by filter
0 packets dropped by kernel
```

Figure 2: Starten von `tcpdump`

Der Output von `tcpdump` wird in die Datei `raw` gespeichert. Zum Kopieren wird wieder `scp` verwendet:

```
# ariez @ Davids-iMac in ~/Documents/Uni/S8/netze/ps1 on git:master x [16:28:07] C:130
$ for _ in {1..10}; do scp -i ~/.ssh/cosy dpape@sshstud.cosy.sbg.ac.at:~/1mb tmp; done
1mb                                                                           100%  977KB   5.0MB/s   00:00
1mb                                                                           100%  977KB   4.9MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
1mb                                                                           100%  977KB   5.2MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
1mb                                                                           100%  977KB   5.2MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
1mb                                                                           100%  977KB   5.1MB/s   00:00
```

Figure 3: Zehnmaliger Aufruf von `scp`

```
1 16:28:17.466166 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50604: Flags [FP.], seq 2225850822:2225850894, ack 277840552, win 501, length 72
2 16:28:18.490382 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50606: Flags [FP.], seq 1360429584:1360429656, ack 4178653402, win 501, length 72
3 16:28:18.654240 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [S.], seq 23980821, ack 3129048169, win 64240, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0
4 16:28:18.677606 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [.], ack 22, win 502, length 0
5 16:28:18.686672 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [P.], seq 1:42, ack 22, win 502, length 41
6 16:28:18.709776 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [P.], seq 42:1122, ack 22, win 502, length 1080
7 16:28:18.753845 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [.], ack 1414, win 501, length 0
8 16:28:18.775162 IP felstaube.cs.sbg.ac.at.ssh > 192.168.0.16.50615: Flags [.], ack 1462, win 501, length 0
```

Figure 4: Roher Output (`raw`)

Da der rohe Output von `tcpdump` etwas unübersichtlich ist, wird die Datei `raw` anschließend mit `awk` etwas verdaulicher gemacht:

```
awk -F '[ ,]' 'BEGIN {print "TIMESTAMP\tSEQ"} \
{print $1 "\t" $10}' raw > cleaned
```

```
1 TIMESTAMP          SEQ
2 16:28:17.466166    2225850822:2225850894
3 16:28:18.490382    1360429584:1360429656
4 16:28:18.654240    23980821
5 16:28:18.677606    22
6 16:28:18.686672    1:42
7 16:28:18.709776    42:1122
8 16:28:18.753845    1414
```

Figure 5: Bereinigte Daten (`cleaned`)

Um schlussendlich die Diagramme zu erstellen, werden Python und matplotlib verwendet. Das Skript, welches die Plots erstellt (`plot.py`), ist im Anhang grob wiedergegeben und kommentiert.

Mit dem Befehl `python plot.py cleaned` wird folgendes Diagramm erzeugt:

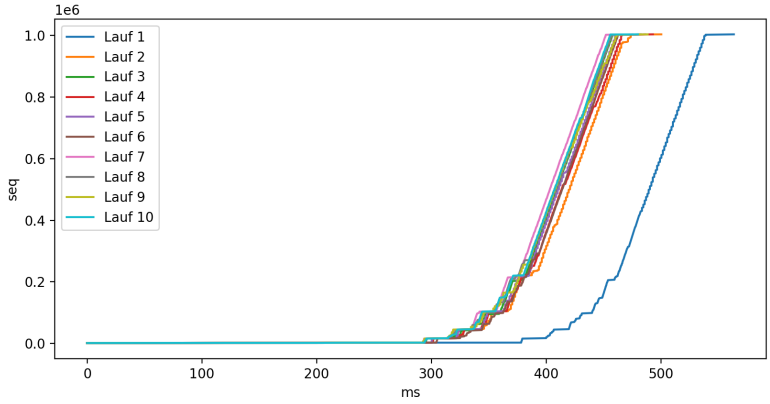

Figure 6: Ethernet-Verbindung

Im Anschluss wird das Experiment auf einem mobilen Rechner im Wi-Fi wiederholt. Als solcher dient ein doch recht gemächlicher Raspberry Pi Zero W, was hoffentlich die deutlich geringere Übertragungsrate erklären kann.

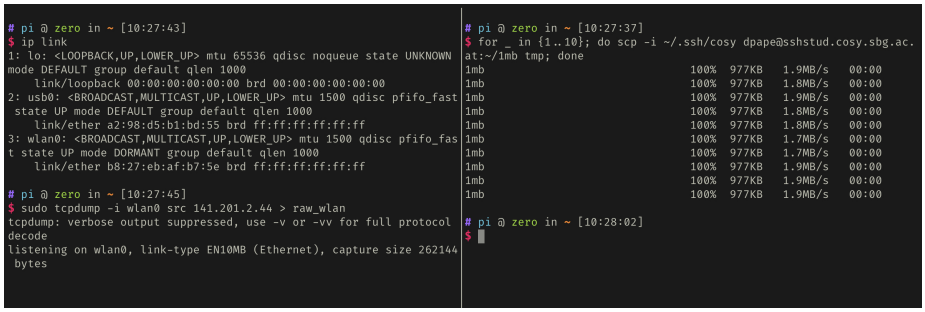

Figure 7: Ausführung auf dem Pi Zero

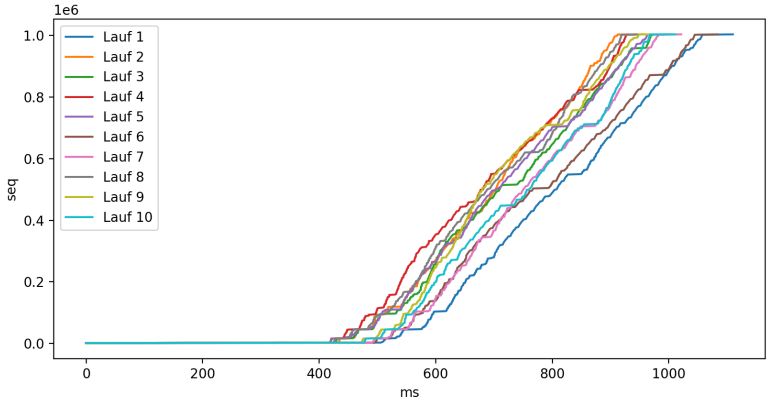

Figure 8: Wi-Fi (stationär)

Um eine schlechte Verbindung zu emulieren, entferne ich mich vom AP und bringe eine Wand zwischen die Geräte. Zusätzlich wird ein `sleep 1` in die `for`-Schleife mit dem `scp`-Aufruf eingebaut, damit mir genügend Zeit dazu bleibt.

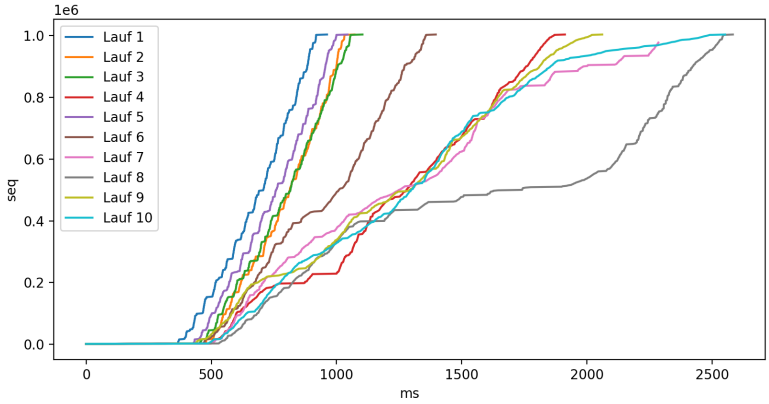

Figure 9: Wi-Fi (schlechte Verbindung)

Diskussion

Die Ergebnisse spiegeln im Allgemeinen die Erwartungen wider. Mit Abstand am konsistentesten ist die Ethernet-Verbindung, wo nur der erste Lauf ca. 80ms länger gebraucht hat als der Rest. Scheinbar war im Anschluss die Verbindung "aufgewärmt", und vielleicht spielt auch die seek time der Festplatte vom `felstaube`-Rechner eine (kleine) Rolle.

Die stationäre Wi-Fi Verbindung hat mit durchschnittlich ca. 1000ms doppelt so lange gebraucht wie die Ethernet-Verbindung und zeigt auch eine deutlich größere Varianz auf. Dies verblasst aber angesichts der simulierten schlechten Verbindung, welche bis zu 2500ms gebraucht hat und ein stark variierendes Wachstum der `seq`-Nummern aufweist.

Beim Diagramm von Letzterem wird eventuell auch ersichtlich, dass der Aufbau von Entfernung zwischen den Geräten nicht der zuverlässigste Weg ist, um eine schwächere Verbindug zu simulieren. Zwar ist der generelle Trend wie erwartet (der Download dauert länger), allerdings gibt es bemerkenswerte Outlier.

Beispielsweise wurde Durchlauf 5 ähnlich schnell wie Durchlauf 1 abgeschlossen, obwohl sowohl der Lauf davor als auch der danach wesentlich langsamer waren. Auch ist der zehnte Durchlauf gleich schnell wie der achte Durchlauf abgeschlossen. Dabei habe ich versucht, die Distanz linear mit der Durchlaufsnummer zu erhöhen, und beim 5. Durchlauf sollte bereits eine Wand zwischen den Geräten gewesen sein.

Anhang

```
timestamps = []
seqs = []

with open(sys.argv[1]) as f:
    prev_seq = 10e20
    for line in f.readlines()[1:]: # ignore header
        split = line.strip().split("\t")

        if len(split) == 2:
            # parse seq and timestamp
            seq = int(split[1].split(":")[0])
            timestamp = parse(split[0]).timestamp()

            # the file contains a series of connections.
            # seq < prev_seq indicates that we finished parsing
            # a connection, so we plot it and prepare for
            # parsing the next
            if seq < prev_seq:
                if len(timestamps) > 500: # ignore noise
                    plt.plot(timestamps, seqs)
                timestamps = []
                seqs = []
                initial_timestamp = timestamp

            prev_seq = seq
            timestamps.append(timestamp - initial_timestamp)
            seqs.append(seq)
```