Computernetzwerke
Hochschule für Technik und Wirtschaft – HTW Berlin
Sebastian Bauer
Wintersemester 2022/2023
3. Laboraufgabe vom 13. Dezember 2022
Abgabe bis zum Freitag, dem 13. Januar 2023 via GitLab und HTW-Cloud

---

Wir setzen in diesem virtuellen Labor Mininet (`http://mininet.org`) ein, wobei wir uns im ersten Labor hauptsächlich auf die Einrichtung des Systems konzentriert haben. Im zweiten Labor haben wir eine komplexere Topologie erstellt und uns mit Routern beschäftigt.

Das erfolgreiche Lösen der Aufgabe besteht

- im Durcharbeiten und Verstehen eines jeden einzelnen Punktes,
- bei Unklarheiten Fragen im Forum zu stellen,
- die Antworten zu den Fragen in einer Datei zu protokollieren sowie
- den Inhalt der Datei zur HTW-Cloud hochzuladen und mit dem Dozenten zu teilen und ihm darauf Schreibzugriff für Kommentare zu geben (*Kann bearbeiten*).

Da das Laboraufgabenblatt noch neu ist und die Aufgabenstellung neu entwickelt wurde, können sich hier viele Fehler eingeschlichen haben. Der Dozent bedankt sich für jeden Fehlerbericht. Auch über unklare Textstellen und Verbesserungsvorschläge kann gerne berichtet werden. Das Aufgabenblatt wird deshalb im Laufe der Veranstaltung aktualisiert.

Abgabe: Die Abgabe des Protokolls (Beantwortung der Fragen) erfolgt über die HTW-Cloud (Anleitung unter `https://anleitungen.rz.htw-berlin.de/de/cloud`). Der Name der abzugebenden Datei lautet aus `CNW2022-Labor3-<Vorname(n)>-<Nachname(n)>.pdf` ohne Leerzeichen. Die Datei wird dann im Lese/Schreibmodus (aka. *Kann bearbeiten*) mit dem Dozent geteilt. Feedback gibt der Dozent direkt im PDF.
Hinweis: Dieses Arbeitsblatt setzt das Bestehen des ersten und zweiten Aufgabenblatts voraus.
Weiterer Hinweis: Zum Beantworten der Fragen sind verschiedene Eingaben im Terminal zu machen. Da dies nicht immer im selben Kontext geschieht, werden der Einfachheit halber verschiedenen Kommandoprompts genutzt. Hierbei steht `h$` für Eingaben im Host, `m$` für Eingaben auf der Mininet-VM und `mininet>` für Eingaben im Mininet-CLI.[1]

---

[1] CLI bedeutet ausgeschrieben Command-Line-Interface

1. **Datenrate zwischen zwei Hosts ermitteln**. Es sollen die Datenraten zwischen zwei Hosts ermittelt werden. Genutzt werden soll in dieser Aufgabe eine einfache Netzwerk-Topology, die Mininet von Hause aus bereitstellt.

   (a) Informiere Dich über `iperf`. Wozu sind die Parameter `-s`, `-p`, `-f` und `-i` gut?

   (b) Starte auf dem Gastsystem Mininet mit minimaler Topology, im der zwei Hosts `h1` und `h2` via Switch verbunden sind, zum Beispiel vom Host aus:

   ```
   h$ ssh -Y root@<guest-ipaddr> -t mn --topo=minimal
   ```

   (c) Gib Dir alle relevanten Informationen aus und überprüfe sie auf Plausibilität. Notiere die IP-Adressen der beiden Hosts `h1` und `h2`.

   (d) Starte auf jeden der zwei Hosts ein Terminal, zum Beispiel via:

   ```
   mininet> xterm h1 h2
   ```

   (e) Starte auf `h1` das Tool `iperf` im Server-Mode

   ```
   h1> iperf -s -p 5566 -i 1 -f M
   ```

   (f) Starte auf `h2` das Tool `iperf` im Client-Mode unter Angabe der IP-Adresse des Servers.

   ```
   h2> iperf -c <h1-ip> -p 5566 -t 15
   ```

   (g) Beobachte die beiden Terminals und mache vom Inhalt einen Screenshot.

2. **Ergebnisse visualisieren**. Ausgehend von den Daten, die das Server-Programm von `iperf` ausgibt, sollen die Ergebnisse visualisiert werden. Das Setup ist dasselbe wie in der letzten Teilaufgabe. Auf `h1` und `h2` sind noch keine Prozesse von `iperf` gestartet.[2] Die Visualisierung erfolgt auf dem Host, sodass es wieder sinnvoll ist, auf die Dateien der Mininet-Maschine via `sshfs` zugreifen zu können. Für die Visualisierung kannst Du wie in Rechnerorganisation `R` nutzen. Andere Möglichkeiten sind `maplotlib` für Python oder `gnuplot`.

   (a) Starte `iperf` im Server-Mode wie oben, jedoch lass die Ergebnisse in eine Datei umleiten, in diesem Beispiel ist dies `h1.results`.

   ```
   h1> iperf -s -p 5566 -i 1 -f M >h1.results
   ```

   (b) Starte auf `h2` wie in der letzten Aufgabe den Client von `iperf`.

   (c) Kopiere `h1.results` von der Mininet-Maschine auf den Host und füge Sie zum Repository aus dem letzten Labor hinzu in einem Unterverzeichnis `lab3` hinzu.

   (d) Betrachte den Inhalt von `h1.results` auf dem Host, zum Beispiel via

   ```
   h$ cat h1.results
   ------------------------------------------------------------
   Server listening on TCP port 5566
   TCP window size: 0.08 MByte (default)
   ------------------------------------------------------------
   [ 14] local 10.0.0.1 port 5566 connected with 10.0.0.2 port 47370
   [ ID] Interval       Transfer     Bandwidth
   [ 14]  0.0- 1.0 sec  3459 MBytes  3459 MBytes/sec
   [ 14]  1.0- 2.0 sec  3511 MBytes  3511 MBytes/sec
   ...
   ```

   Der Inhalt der Datei entspricht noch nicht die Form, in der er mit `R` gut verarbeitet werden könnte. In der folgenden Teilaufgabe nutzen wir die Shell, um eine Datei in Tabellenform daraus zu generieren.

   (e) Wir erkennen, dass jede relevante Zeile das Wort `sec` enthält und können via `grep` einen Filter für diese Zeilen erzeugen:

[2] Brich ggf. den Server-Prozess via `CTRL+C` ab

```
h$ cat h1.results | grep sec
[ 14]  0.0- 1.0 sec  3459 MBytes  3459 MBytes/sec
[ 14]  1.0- 2.0 sec  3511 MBytes  3511 MBytes/sec
            ...
```

Mit `tr` lassen sich einzelne Zeichen einer Datei durch andere ersetzen. Im folgenden ersetzen wir das '-'-Zeichen durch ein Leerzeichen, sodass wir im Anschluss Leerzeichen als Spaltentrenner annehmen können.

```
h$ cat h1.results | grep sec | tr '-' ' '
[ 14]  0.0  1.0 sec  3459 MBytes  3459 MBytes/sec
[ 14]  1.0  2.0 sec  3511 MBytes  3511 MBytes/sec
...
```

Mit dem Programm `awk` lassen sich nun recht einfach einzelne Spalten extrahieren. Mit

```
h$ cat h1.results | grep sec | tr '-' ' ' | awk '{print $1,$2}'
```

extrahieren wir zum Beispiel Spalte 1 und 2 und stellen fest, dass wir andere Spalten benötigen.

(f) Modifiziere das eben genutzte Kommando, sodass Spalte 4 und Spalte 8 extrahiert werden. Notiere das Kommando. Die Ausgabe könnte in etwa so aussehen:

```
1.0 3459
2.0 3511
3.0 3472
...
15.0 3563
15.0 3547
```

(g) Erkennbar ist, dass die Angabe für 15.0 zweimal auftaucht. Wie man in der originalen Datei sehen kann, ist die letzte Zeile die durchschnittliche Datenrate über die gesamte Zeit, die wir hier nicht brauchen. Via `head -n -1` lässt sich diese Zeile auch herausfiltern. Wie lautet das gesamte Kommando, wenn die finale Ausgabe ein eine Datei `h1.tsv` umgeleitet werden soll? Führe dieses Kommando aus und füge das Ergebnis zu Deinem Repository hinzu.

(h) Schreibe ein Skript, das aus der Textdatei ein Liniendiagramm mit der Zeit auf der X-Achse und die Datenrate auf der Y-Achse als PDF-Datei erzeugt. Falls Du `R` nutzt, so lauten relevante Kommandos: `read.table()`, pdf() und `plot()`. Alternative Möglichkeiten sind `maplotlib` oder `gnuplot`. Füge das Skript und die PDF-Datei mit ins Git-Repository hinzu und bilde das Ergebnis im Protokoll ab.

3. **Parking-Lot**. In dieser Aufgabe soll ein parameterisiertes Netzwerk wie in Abbildung 1 erstellt werden. In diesem Netzwerk existieren n Hosts, die als Sender fungieren und ein Host, der als Empfänger fungiert. Jeder Senderhost ist mit einem eigenen Router verbunden, die ihrerseits wieder eine Kette bilden, die mit dem Empfänger verbunden ist.

   Wie im Labor 2 soll ein Python-Skript geschrieben werden, dass das Netz instanziert, wobei in dieser Aufgabe der Parameter n per Kommandozeilenparameter `--hosts` mitgegeben werden soll. Es genügt den Definitionsbereich von n auf $1 \leq n \leq 10$ für $n \in \mathbb{N}$ zu beschränken. Das Skript soll den Namen `parkinglot.py` tragen und im GitLab-Repo aus Labor 2 bereitgestellt werden.

   Beispielsweise soll ein Aufruf von

   ```
   m$ sudo python parkinglot.py --hosts=10
   ```

   ein Netz mit 10 Sendinghosts erstellen (der Empfängerhost zählt nicht mit). Der Vorgabewert von des Parameters `--hosts` beträgt 5.

   (a) Plane das Netzwerk, insbesondere die zu nutzenden IP-Adressen. Jeder der Senderhosts und der Empfängerhost sollen sich in paarweise verschiedenen Netzwerken befinden (d.h., jeder

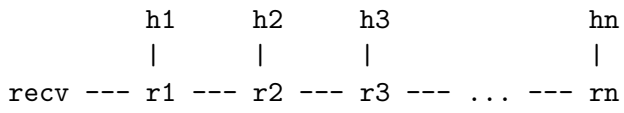

Abbildung 1: **Parking-Lot-Topology**. In einer Parking-Lot-Topologie existiert eine Empfängerrechner (`recv`) und n Senderrechner (`h1` bis `hn`)

Host befindet sich in einem eigenen Netzwerk). Zeichne das Schema mit IP-Adressen auf. Beachte, dass es auch Verbindungen zwischen den Routern gibt.

(b) Informiere Dich über das Python-Modul `argparse`.

(c) Implementiere das Skript. Beachte, dass die Hosts nicht nur nach `recv` senden können, sondern auch an andere Hosts im Netz. Es ist nach Meinung des Dozenten hier wieder besonders wichtig, kleine Funktionalitäten in eigene Funktionen auszulagern, um die Lesbarkeit des Quelltextes zu gewährleisten. Beispielsweise sollte eine Funktion geschrieben werden, die zu einer Hostnummer eine IP-Adresse im selbst gewählten Schema zurückgibt. Das ist übersichtlicher und *idiomatischer* als String-Interpolation zu nutzen, selbst wenn es f-Strings gibt. Ebenso ist es sinnvoll, eine idiomatische Funktion zu haben, die die Aufrufe von `ip route add` wrapt. Trenne auch das Instanziieren der Nodes und Links vom Hinzufügen der Routen. Committe jeden während der Entwicklung durchgeführten Schritt mit einer sinnvollen Commit-Nachricht in Dein Repo.

(d) Verifiziere das Netz für die Größen $n \in \{5, 10\}$ mit dem Kommando `pingall`. Füge für beiden Varianten ein Screenshot zum Beleg.

(e) Kennzeichne nach dem Committen des funktionierenden Zustands diesen Zustand mit dem Tag `version_1_0`, durch das Kommando:

```
h$ git tag v1.0
h$ git push origin v1.0
```

Informiere Dich ggf. über die Funktionalität der Tags, z. B. auf https://www.thomas-krenn.com/de/wiki/Git_Tags.

4. **Datenrate begrenzen**. Mininet besitzt ein großes Repertoire von Funktionen, um Parameter von Netzwerken zu beeinflussen. Ausgehend von der letzten Aufgabe wollen wir nun den Parameter der maximalen Datenrate, die zwischen den Knoten gilt, mit einbauen.

   Mininet bietet hierfür erweiterte Link-Typen an. Diese kann man dem Aufruf von `addLink` via Parameter `cls` mitgeben. Der hier zu nutzende Link-Typ heißt `TCLink`, was für Traffic-Controll-Link steht. Dieser kennt unter anderem den Parameter `bw`, was die maximale Datenrate in MBit/s spezifiziert.[3] Um einen Link von Host `h1` zu Router `r1` anzulegen, über den maximal 10 MBit/s gehen, würden wir also

   ```
   self.addLink(h1, r1, cls=TCLink, bw=10)
   ```

   aufrufen. Die Klasse `TCLink` müssen wir zu Beginn importieren:

   ```
   from mininet.link import TCLink
   ```

   (a) Erweitere das in der letzten Aufgabe geschriebene Skript, um den optionalen Parameter `--maxrate`, der eine Zahl erwartet, die die maximale Datenrate aller Links in MBit/s spezifiziert. Ist der Parameter nicht angegeben, so verhält sich das Programm wie bisher.

   (b) Starte das Netz mit 10 Hosts und einer maximalen Datenrate von 10 MBit/s und überprüfe den Effekt des Parameters mithilfe von `iperf`, indem Du den Server auf `recv` startest und den Client auf `h10`. Als Lösung gilt die Ausgabe von `iperf`.

[3] `bw` steht für engl. *bandwitdh*, was, wie wir wissen, nicht der korrekte Ausdruck ist.

(c) Vergiss nicht, den geänderten Quelltext im Git-Repository zu verewigen.

(d) Kennzeichne den Zustand unter den Tag `v1.1`.