Computernetzwerke
Hochschule für Technik und Wirtschaft – HTW Berlin
Sebastian Bauer
Wintersemester 2022/2023
2. Laboraufgabe vom 26. Oktober 2022
Abgabe bis zum Donnerstag, dem 25. November 2022 via GitLab und HTW-Cloud

---

Aufgrund der aktuellen Corona-Eindämmungsmaßnahmen sind an der HTW keine Präsenzveranstaltungen möglich. Dies bedeutet insbesondere für die Pflichtlabore der Lehrveranstaltung Computernetze ein Problem, da diese bisher auf dedizierter Hardware stattfanden.

In diesem Semester finden die Labore deshalb zu Hause statt. 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. In diesem zweiten Aufgabenblatt beschäftigen wir uns dann direkt mit Mininet.

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-Labor2-<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 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. **Git-Repo** anlegen. Ab diesen Laborübungsblatt benötigen wir ein Git-Repository. Alternativ zur Web-Oberfläche lässt sich ein privates Repository auf GitLab auch via Kommandozeile anlegen, was im Folgenden zu tun ist.

   (a) Lege irgendwo via Kommandozeile ein Order (`mkdir`) mit Namen `cnw-wise-2022-<vornamen>-<nachnahmen>`, wobei `<vornamen>` durch die eigenen Vornamen und `<nachnamen>` die eigenen Nachnamen sind, am Besten alles in Kleinbuchstaben.

   (b) Wechsel in diesen neuen Ordner mit `cd`.

   (c) Initialisiere diesen Ordner als Git-Repository

   ```
   h$ git init
   ```

   Das Kommando legt das Verzeichnis `.git` an, in dem alles wichtige für `git` zu finden ist und das Verzeichnis als Git-Repository identifiziert.

   (d) Füge ein `remote` hinzu und zwar die Adresse Deines GitLab-Namensraum zusammen mit dem Namen des anzulegenden Git-Repository:

   ```
   h$ git remote add origin \
     git@gitlab.rz.htw-berlin.de:<user>/cnw-wise-2022-<vornamen>-<nachnahmen>.git
   ```

   (alles in einer Zeile ohne \ und die Platzhalter ersetzen)

   (e) Füge eine Datei Namens `ReadMe.md` zum Verzeichnis hinzu mit sinnvollen Inhalt

   (f) Füge die neue Datei auch ins Git-Repo hinzu und committe den Zustand

   ```
   h$ git add ReadMe.md
   h$ git commit -m "Bring project to life and add initial ReadMe."
   ```

   (g) Pushe das Repository

   ```
   h$ git push origin main
   ```

   (h) Das neue Repo sollte jetzt auf der Webseite erscheinen. Den sichtbaren Namen des Projekts kannst Du anpassen. Lade außerdem den Dozent als *Maintainer* ein (das funktioniert auch über die Kommandozeile, ist aber aufwendiger).

2. **Kleine Programmierübung in C**. In der Vorlesung haben wir die Base64-Kodierung kennengelernt, die in RFC 4648 definiert ist. Mittels Tool `base64` können wir beliebige Binärdaten in dieses Format konvertieren. Als kleine Nebenübung wollen wir ein ähnliches Programm in der Programmiersprache C schreiben, das `bb64` heißen soll (für *better base64*). Die Kodierung soll allerdings nicht dieselbe sein, sondern wie folgt:

   - Werte 0 bis einschließlich 9 sollen über die ASCII-Ziffern 0 bis 9 kodiert werden
   - Werte 10 bis einschließlich 35 sollen über die ASCII-Großbuchstaben A bis Z kodiert werden
   - Werte 36 bis einschließlich 61 sollen über die ASCII-Kleinbuchstaben a bis z kodiert werden
   - Werte 62 soll `-` sein und 63 der Unterstrich `_`
   - Als Füllzeichen soll `=` genutzt werden. Füllzeichen werden benötigt, wenn die Länge der zu kodierenden Daten nicht durch 3 teilbar ist.

   Beim Starten kann dem Programm ein Argument übergeben werden, das dann den Namen der Datei repräsentiert, die kodiert werden soll. Wird kein Argument übergeben, so liest das Programm die Datei via Standardeingabestrom (`stdin`) ein. Die Ausgabe erfolgt immer im Standardausgabestrom (`stdout`). Die Länger einer Zeile darf 68 Zeichen nicht übersteigen. Fehlermeldungen, z. B. bei nicht existierender Eingabedatei, sind via Standardfehlerstrom auszugeben (`stderr`).

   Neben dem eigentlichen Programm soll auch ein Test geschrieben werden. Daher bietet es sich an, die zu implementierte C-Funktion nicht im Hauptprogramm zu definieren.

   Als Build-System kannst `make` oder `cmake` genutzt werden.

(a) Studiere die RFC 4648, die Vorlesungsunterlagen und andere Quellen über das Prinzip der Kodierung.

(b) Lege die Projektstruktur im Checkout des Git-Repositorys an. Neben dem Verzeichnis, das `bb64` heißen soll, sollen dort auch die drei Quelltextdateien (für das Hauptprogramm, das Modul mit der zu entwickelnden Funktion und den Test) sowie die Header-Datei für die zu entwickelnde Funktion erstellt werden. Sowohl Hauptprogramm als auch Testprogramm können zunächst nur aus einer leeren `main()`-Funktion bestehen und nichts zu.

(c) Sorge dafür, dass das Programm und der Test via `make` bzw. `cmake` gebaut werden können. Der Test soll auch beim Bauen aufgerufen werden.

(d) Vergiss nicht die `.gitignore`-Datei.

(e) Committe alles in Dein Repo.

(f) Schreibe zunächst den Test für die zu entwickelnde Funktion und zwar anhand eines sehr einfachen Beispiels. Entwirf den Test zunächst für eine Eingabe, deren Länge durch drei teilbar ist. Das Vorgehen, zuerst den Test und dann die Funktion zu schreiben, nennt man Test-Driven-Development. Ein solches Vorgehen ist sehr gut und deshalb sollte man Implementierungen immer so durchführen.

(g) Schreibe jetzt an der Funktion, bis der Test funktioniert.

(h) Committe den Zustand.

(i) Erweitere den Test um eine andere Eingabe, deren Länge nicht durch drei teilbar ist. Der Test sollte wieder fehlschlagen.

(j) Vervollständige die Implementation bis der Test nicht mehr fehlschlägt.

(k) Implementiere das Hauptprogramm.

(l) Committe alles.

(m) Zeige im Beleg anhand eines kurzen Beispiels, dass es funktioniert.

3. **Python**. Wie im letzten Laborübungsblatt herausgearbeitet wurde, lässt sich Mininet über Python programmieren. Dies ist eine hervorragende Gelegenheit sich wieder mit dieser Programmiersprache auseinanderzusetzen.[2]

   (a) Falls noch nicht bekannt, mache Dich kurz mit dem Syntax von Python-Quelltexten vertraut. Als Lernumgebung empfiehlt sich z. B. Thonny, das Debian-User einfach per `apt install thonny` installieren können.

   (b) Zähle einige markante Unterschiede zwischen Python und den Programmiersprachen C und C++ auf.

   (c) Starte Mininet wie im ersten Laboraufgabenblatt, d. h.,

      1. Starte die Mininet-VM, falls noch nicht getan.
      2. Starte via `ssh` Mininet. Tipp: Wenn die Session nur für Mininet genutzt werden soll, kann dies auch in einem Rutsch geschehen, da `ssh` die Angabe von weiteren Kommandos erlaubt, z. B. so:

         ```
         h$ ssh -Y mininet@<guest-ipaddr> -t sudo mn
         ```

         Die IP-Adresse verweist auf die der Mininet-VM, diese kann natürlich bei Dir anders aussehen oder einen Namen besitzen, falls ein Eintrag in `/etc/hosts` erfolgte. Die Option `-Y` oder alternativ `-X` wird ebenfalls wichtig sein.
         Wichtig ist dabei die Angabe von `-t`. Was bewirkt die Option? (siehe `man ssh`).

[2] Der Dozent nutzt Python gerne für kleine Hilfsprogramme und Datenanalysen. Andere nutzen es auch für komplexere Programme oder Webseiten. Für Software im Embedded-Bereich ist es nach Meinung des Dozenten nicht die erste Wahl, obwohl es mit MicroPython auch hier Lösungen gibt.

(d) Gib `Hello, World` auf den Bildschirm via Python und Mininet aus. Mininet stellt hierfür das Kommando `py` zur Verfügung, das die Argumente als Python-Ausdruck interpretiert und das Ergebnis ausgibt:

```
mininet> py "Hello, world"
```

(e) Wie geschrieben handelt es sich um einen Pythonausdruck, den man mitgeben kann. Mit der Funktion `bin(x)` lässt sich z. B. die binäre Darstellung des Argument `x` herausfinden. Wie lautet die Ausgabe von:

```
mininet> py bin(88)
```

(f) Wie lautet die IP-Adresse des Netzwerk-Interfaces, über den die Kommunikation mit dem Hostrechner erfolgt?

(g) Wie lautet die Bitrepräsentation der IP-Adresse aus der letzten Teilaufgabe?

(h) Was tut folgender Einzeiler?

```
mininet> py [x for x in range(2,200) if all([x%i for i in range(2,x)])]
```

Mithilfe dieser sogenannten List-Comprehensions lassen kurze und präzise Listen erstellen, ähnlich wie in bei der Mengenschreibweise in der Mathematik.

(i) Wie lautet ein Einzeiler für die Erzeugung eines Vektors mit allen Quadratzahlen zwischen 1 und 100?

(j) Wie lautet ein Einzeiler für die Bestimmung der Summe der Quadratzahlen zwischen 1 und 100 und wie lautet das Ergebnis?

(k) Wie lautet ein Einzeiler für die Erzeugung einer Liste von IP-Adressen als Zeichenkette zwischen 192.168.1.1 und 192.168.1.122? Nutze hierfür die f-Strings (siehe https://peps.python.org/pep-0498/).

(l) Durch die Python-Funktion `locals()` hat man Zugriff auf die Symboltabelle, die im lokalen Scope gültig ist. Protokolliere und interpretiere die Ausgabe von

```
mininet> py locals()
```

(m) Mit der Python-Funktion `dir()` lassen sich die Methoden von Objekten auflisten. Protokolliere die Ausgabe von `dir` aufgerufen auf das Objekt `h1`.

(n) Mit `help()` kann die oft eingebaute Hilfe von Python-Elementen angezeigt werden:

```
mininet> py help(h1.IP)
```

Was tut die Funktion `IP()`?

(o) Was gibt `IP()` für das Objekt `h2` zurück?

4. **Erster Router**. In dieser Aufgabe werden wir den Rechner `h1` zu einem Router umfunktionieren und daran einen weiteren Host `h3` anschließen. Am Ende soll dieser mit `h2` kommunizieren können.

(a) Verschaffe Dir einen Überblick über die Mininet-API auf http://mininet.org/api/annotated.html.

(b) Füge dem Netzwerk nun einen weiteren Host hinzu:

```
mininet> py net.addHost('h3')
```

(c) Protokolliere die aktuelle Netzwerkkonfiguration auf allen Hosts.

```
mininet> h1 ip a
mininet> h2 ip a
mininet> h3 ip a
```

(d) Was fehlt bei `h1` und `h3`, um die Aufgabenstellung zu lösen?

(e) Verbinde jetzt `h1` mit `h3`

```
mininet> py net.addLink(h1, h3)
```

(f) Protokolliere erneut die Netzwerkkonfiguration von Hosts `h1` und `h3`. Halte insbesondere die Namen der neuen Netzwerkschnittstellen fest. Wenn bei der Ausgabe von `ip` ein `@`-Symbol zu sehen ist, dann steht der gesuchte Name davor. Über `netstat -i` und das veraltete `ifconfig -a` können die Namen auch in Erfahrung gebracht werden.

(g) Wähle Netzwerkanteil und Subnetzmaske des neuen Netzes bestehend aus `h1` und `h3` aus und notiere sie. Beachte, dass es sich um ein privates Netz handeln muss. Welchen Einschränkungen unterliegen die IP-Adressen?

(h) Wähle nun zwei passende IP-Adressen für die neuen Netzwerkinterfaces fest. Erinnere Dich, dass `h1` der Router sein soll, welche Adresse ist dann für entsprechende Interface von `h1` sinnvoll?

(i) Konfiguriere die beiden neuen Netzwerkschnittstellen auf `h1` und `h3` mit den eben festgelegten Netzwerkadressen und Subnetmasken. Dies kann wie im ersten Aufgabeblatt via `ip addr add` geschehen. Im Mininet-CLI muss der Namen des Hosts vorangestellt werden, um das Kommando auf diesem Host auszuführen. Notiere die Kommandos und verifiziere, dass es geklappt hat.

(j) Wie lautet das Kommando um von `h1` nach `h3` zu pingen, wobei für `h3` zunächst die IP-Adresse genutzt werden soll? Protokolliere das Ergebnis.

(k) Wie lautet das Kommando um von `h3` nach `h1` zu pingen, wobei für `h1` zunächst die IP-Adresse des neuen Netzwerkinterfaces genutzt werden soll? Protokolliere das Ergebnis.

(l) Welche Ausgaben erscheinen bei

```
mininet> h3 ping h1
mininet> h3 ping h2
```

und

```
mininet> h1 ping h3
```

Was bedeuten die Meldungen und wie erklärst Du Dir den Unterschied?

(m) Mit `ip route` kannst Du Dir die Weiterleitungstabelle des Kernels anzeigen lassen. Führe das Kommando auf `h3` aus und protokolliere das Ergebnis.

(n) Wir möchten jetzt `h3` eine Default-Route bzw. Default-Gateway hinzufügen. Was ist eine Default-Route? Wir können eine Default-Route ebenfalls mit `ip route` festlegen:

```
mininet> h3 ip route add default via <h1 ip> dev <h3 netzwerkinterfacename>
```

Damit sagen wir, dass alle Pakete, zu denen sonst kein Eintrag in der Tabelle zu finden ist, zu der `<h1 ip>` geleitet werden sollen, wobei das Netzwerkinterface `<h3 netzwerkinterfacename>` genutzt wird.

(o) Wiederhole die Versuche aus Teilaufgabe (l). Was hat sich geändert? Protokolliere und interpretiere die Ergebnisse.

(p) Laut Plan des Dozenten sollte ein Ping von `h3` und `h2` noch immer nicht gehen, die anderen aber schon. Wir versuchen dem Problem genauer auf die Schliche zu kommen und nehmen dafür Wireshark, um auf den Link zwischen `h3` und `h1` zu sniffen. Wir überprüfen also zunächst, ob das IP-Paket, das das ICMP-Paket einkapselt sichtbar ist, wobei wir uns den Endpunkt von `h1` herauspicken. Hierzu brauchen wir Zugriff auf das korrekte Netzwerk-Interface aufseiten von `h1`. Entweder erinnerst Du Dich an den Namen oder Du ermittelst ihn via:

```
mininet> links
```

Notiere den Namen des Netzwerkinterfaces.

(q) Startet man jetzt Wireshark so wie im letzten Aufgabenblatt, dann haben wir keinen direkten Zugriff auf die Netzwerkinterfaces von `h1`. Der Grund ist, dass der normale Nutzer bzw. der Rootnutzer zunächst nur Zugriff auf den Vorgabenetzerknamensraum hat, die Netzschnittstellen der Hosts damit nicht sichtbar sind. Wir müssten den Namensraum des Prozesses jetzt

auf dem vom `h1` umstellen, was allerdings mehrere Schritte bedarf.[3] Über die Kommandozeile von Mininet ist jedoch das Kommando `x` verfügbar, was den Vorgang für uns abnimmt und zudem X11-Forwarding erlaubt, was nötig für Nutzerschnittstellen ist. Starte nun Wireshark auf dem `h1`:

```
mininet> x h1 wireshark
```

(r) Wähle das eben notierte Interface und starte den Capture. Führ parallel dazu folgendens Kommando in der Mininet-Console auf:

```
mininet> h3 ping h2
```

Protokolliere das Ergebnis mithilfe eines Screenshots.

(s) Was siehst Du für Pakete, wenn Du das Interface, das mit dem Switch `s1` verbunden ist, im Wiresharkfenster (ggf. einen neuen Capture starten; per Multiselection kann man übrigens von mehreren Interfaces sniffen; jedoch wird das Netzwerkinterface per Vorgabe nicht dargestellt)?

(t) Welche Schlussfolgerung ziehst Du daraus?

(u) Wir wollen jetzt IP-Forwarding auf `h1` einstellen:

```
mininet> h1 echo 1 > /proc/sys/net/ipv4/ip_forward
```

Alternative geht auch:

```
mininet> h1 sysctl net.ipv4.ip_forward=1
```

Was bewirken beide Kommando (Lieblingssuchmaschine)?

(v) Funktioniert jetzt der Ping von `h3` nach `h2`?

(w) Welche Verbesserung hat das Setup allerdings gebracht, wenn man Pakete auf den Interface zu `h2` anzeigen lässt im Vergleich zu Teilaufgabe (s)?

(x) Wie lautet das Kommando um Wireshark auf `h2` in Mininet zu starten?

(y) Kommt der Ping-Request bei `h2` an? (Beweis via Screenshot von Wireshark gestartet auf `h2`)

(z) Wie behebst Du das Problem? Schreibe das Kommando auf[4] und beweise via Output von

```
mininet> h3 ping h2
```

und einen Screenshot von Wireshark.

Leider hat das Alphabet nur 26 Buchstaben, die aller wichtigsten Aspekte der IP-Weiterleitung konnten wir damit allerdings schon abdecken. Glückwunsch :)

5. **Skripte und Mininet**. In der letzten Aufgabe haben wir einen einfachen Router interaktiv aufgebaut und ein Problem der unzureichenden konfigurierten Weiterleitungstablle diagnostiziert und behoben. Das interaktive Vorgehen ist für einfaches Setup ganz gut, für ein größeres Netz allerdings ineffizient und sehr schwer verteilbar. Ziel dieser Aufgabe wird sein, ein der Mininet-Distribution beigelegtes Beispiel, das die Mininet-API nutzt, auszuführen und die Umgebung für ein eigenes Netz zu schaffen.

   (a) Wir werden in dieser und in der nächsten Aufgabe offensichtlich etwas mehr mit Python zu tun haben. Es bietet sich hierfür an, einen Editor auf dem Host zu nehmen, wobei wir das Verzeichnis der Mininet-VM dank `sshfs` einfach in dem Host anhängen können. Installiere auf dem Host `sshfs`, falls noch nicht verfügbar.

   ```
   h# apt install sshfs
   ```

   (b) Lege auf dem Host dann ein neues Verzeichnis an, z. B.

   ```
   h$ mkdir ~/mininet-vm
   ```

[3] Denn leider funktioniert `ip netns` nicht.

[4] Hinweis: Es muss ein Kommando sein, dass auf `h2` ausgeführt wird, denn offensichtlich schickt `h2` keine Antwort zurück

Den Name und den Pfad des Verzeichnisses kannst Du natürlich auch anders wählen, nur müssen dann die folgenden Kommandos und die der nächsten Aufgabe angepasst werden.

(c) Hänge nun das Home-Verzeichnis des Nutzers `mininet` der Mininet-VM via `sshfs` an das soeben erstellte Verzeichnis via

```
h$ sshfs mininet@<guest ip>: ~/mininet-vm
```

und stelle sicher, dass es geklappt hat (vergiss den Doppelpunkt nicht!).[5] Protokolliere die Ausgabe von

```
h$ ls ~/mininet-vm
```

Es sollten darin alle Dateien sichtbar sein, die Du auch in der Shell von Mininet-VM siehst. Wir können jetzt ganz normal auf dem Host entwickeln und insbesondere den Lieblingseditor nutzen.

(d) Öffne nun die Datei `mininet/examples/linuxrouter.py` ausgehend von Home-Verzeichnis der Mininet-VM. Auf dem Host via `vim` zum Beispiel:

```
h$ vi ~/mininet-vm/mininet/examples/linuxrouter.py
```

Es kann natürlich auch ein anderer Editor wie `nano` genutzt werden.

(e) Nun wieder in einer Shell direkt auf der Mininet-VM können wir dieses Setup so starten:

```
m$ sudo python mininet/examples/linuxrouter.py
```

(f) Visualisiere das Netzwerk mit den Kommandos `dump` und `links` und den Webseite http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet wie im letzten Aufgabenblatt und füge das Bild der Lösung dazu.

(g) Verifiziere, dass

```
mininet> h2 ping h1
```

funktioniert und ebenso

```
mininet> pingall
```

und protokolliere die Ausgaben.

(h) Wir schließen die aktuelle Mininet-Instanz, da wir jetzt genug wissen, um mit der nächsten Aufgabe zu beginnen.

```
mininet> exit
```

6. **Netzwerkskript mit eigener Topology**. Ziel wird es sein, ein eigenes Netzwerk bestehend aus vier Netzwerksegmenten N_1, N_2, N_3 und N_4 und zwei Routern R_1 und R_2 zu entwickeln. Netzwerk N_1 und N_2 sind jeweils über einen eigenen Switch an Router R_1 angeschlossen, N_3 und N_4 über jeweils einen weiteren Switch an Router R_2. In den Netzwerken N_1 und N_3 befinden sich jeweils drei Rechner und im Netzwerken N_2 und N_4 jeweils vier.

   Insgesamt soll die Entwicklung des Skripts Host-seitig erfolgen, da hier modernere Tools zur Verfügung stehen.

   (a) Wie viele (virtuelle) Hosts bzw. Rechner gibt es im Netzwerk (ohne Router)?

   (b) Wie viele (virtuelle) Switches benötigen wir? Wir nehmen an, das R_1 mit R_2 direkt verbunden ist.

   (c) Wie viele Netzwerk-Interfaces benötigen wir für beide Router?

   (d) Wähle passende private Netzwerkanteile und Subnetzmaske für die Netzwerke wobei die Netze disjunkt sein sollen (kein Netzwerk soll teil eines anderen sein).

   (e) Zeichne das Netzwerk. Die Rechner im selben Netzwerkes können zusammengefasst werden, die Router allerdings nicht. Trage für die Links auf Routerseite die korrekten IP-Adressen ein. Für die Netzwerke genügt es, nur den Netzwerkanteil mit CIDR-Notation anzugeben.

[5] `umount ~/mininet-vm` macht den Vorgang wieder rückgängig

Beachte, dass es auch einen Verbindung zwischen den Routern existiert, die sich formal im selben Netzwerksegment befinden müssen, sinnvollerweise mit einer langen Netzwerkmaske. Passe ggf. die letzten Teilaufgaben dahin gehend an, falls Du dies noch nicht berücksichtigt hat.

(f) Lege nun an einer Stelle, auf der sowohl die Mininet-VM als auch der Host Zugriff hat das Verzeichnis `02-mininet` an, z. B. von Mininet-Seite

```
m$ mkdir ~/labs
```

oder von Host-Seite (nach obigem Setup von `sshfs`)

```
h$ mkdir ~/mininet-vm/labs
```

(g) Überprüfe Mininet-VM-seitig, ob sich das Verzeichnis `~/labs` tatsächlich existiert.

(h) Wechsel Host-seitig in das Verzeichnis `~/mininet-vm/labs`.

```
h$ cd ~/mininet-vm/labs
```

(i) Ausgehend vom Host und innerhalb des Verzeichnisses `~/mininet-vm/labs` (siehe letzte Teilaufgabe) klone Dein oben angelegtes Git-Repo in dieses Verzeichnis und damit auf den Gast ganz normal via

```
h$ git clone <url> .
```

Der Punkt gibt an, dass kein neues Verzeichnis angelegt werden soll, die Wurzel des Repos sich also im Verzeichnis `labs` befindet.

(j) Kopiere nun das in der letzten Aufgabe benutze Beispiel in das noch anzulegende Verzeichnis `02-mininet` im aktuellen `labs`-Ordner. Es ist sinnvoll, dieses Skript als Grundlage zu nehmen. Versuche spätestens jetzt, es zu verstehen.

(k) Erstelle einen Commit

```
h$ git add .
h$ git commit -m "Add mininet example."
```

und pushe die neuen Commits in das Remote-Repo.

(l) Entwickle jetzt daraus das Skript, um das gewünschte Netzwerk einzurichten. Das neue Skript soll den Namen `ex2.py` tragen und sich im Ordner `02-mininet` befinden. Erstelle aus jeder sinnvollen Änderung einen Commit mit einer noch sinnvolleren Nachricht und pushe diese nach einer erfolgreichen Verifikation[6] in das Remote-Repo. Achte auf eine präzise Dokumentation des Skripts! Das Skript ist fertig, wenn alle Rechner miteinander via `ping` kommunizieren können. Nutze Wireshark, um Fehler einzugrenzen und beachte Box 1 und Box 2.

(m) Protokolliere dann den Aufruf von

```
mininet> pingall
```

(n) Lade den Dozenten zu dem Git-Lab-Projekt ein, falls noch nicht geschehen (`bauers` als Developer) und lade das Protokoll auf die HTW-Cloud hoch und teile es mit dem Dozenten (Modus: *kann bearbeiten*).

[6] Erinnerung: Nutze hierzu `sudo python ex2.py` Mininet-VM-seitig, um das Netz erzeugen zu lassen und zu testen.

Im Beispiel `linuxrouter.py` wird die Methode `addNode()` benutzt, um einen neuen Router anzulegen. Anders als dort praktiziert, empfiehlt der Dozent nicht, eine IP-Adresse via Parameter `ip` festzulegen, da der Name des Netzwerkinterface dann nicht vorhersagbar ist. Anstelle dessen empfiehlt es sich auch für die Adresse des Routers diese mit `addLink()` zu spezifizieren:

```
self.addLink(h, r,
     intfName1='h-eth0', params1={'ip' : "192.168.1.2"},
     intfName2='r-eth0', params2={'ip' : "192.168.1.1"})
```

Hier wird also eine Verbindung zwischen `h` und `r` angelegt, wobei das Interface aufseiten von `h` den Namen `h-eth0` hat und aufseiten von `r` den Namen `r-eth0`.

Box 1: **Hosts hinzufügen**.

In der Hauptfunktion eines Mininet-Programms wird üblicherweise eine Mininet()-Instanz angelegt. Im Beispiel `linuxrouter.py` in der Funktion `run()` heißt diese `net`. Mit dessen Hilfe können u. a. beliebige Kommandos aufgerufen werden, indem die `cmd()` Funktion genutzt wird. Im Beispiel wird mit `info(net['r0'].cmd('route'))` auf Router `r0` das Kommando `route` aufgerufen, was eine weitere Möglichkeit ist, die Weiterleitungstabelle auszugeben. Der umschlossene Aufruf von `info()` sorgt dafür, dass die Ausgabe des Kommandos mithilfe des Python-Logging-Frameworks als Info-Level geloggt wird.[a]

Für die Entwicklung etwas praktischer ist es, anstelle von `cmd()` die Funktion `cmdPrint()` zu nutzen, da diese auch das Kommando noch einmal ausgibt.

[a] Diese Nachrichten werden dann auf dem Bildschirm ausgegeben, da das Skipt den Logger vor `run()` auf Info setzt. Siehe dazu auch https://docs.python.org/3/howto/logging.html.

Box 2: **Logging**.