PADERBORN UNIVERSITY

# Advanced Networked Systems SS24

## Lab0: Welcome and Warm-Up

| | |
|---|---|
| **Maximum points:** | 0 |
| **Submission:** | None |
| **Deadline:** | None |
| **Contact:** | lin.wang@upb.de |

The goal of this lab (not graded) is to introduce you to the logistics of the course labs and get you started on the basics of Mininet—a network emulator we will be using for most of the other labs throughout the course. Please make sure you read this document carefully. You should make a pass at least until Section 4 of this document and perform the preparation accordingly before the first exercise session.

## 1 Code of Conduct

We have **zero tolerance** to cheating, fraud, and plagiarism.

You are encouraged to discuss specific issues and share knowledge with peer students (e.g., on the PANDA discussion forum). However, in any circumstances you are expected to write your own code. Do not buy or sell code. Do not copy code from anywhere. Do not give out code to other students. It is unlikely that the labs will result in two students writing almost identical code. Any similarity in the code will lead to further investigation, and any hint of cheating will be reported to the examination board.

## 2 Labs Organization

The course includes in total **five graded labs** (i.e., lab1 through lab5). All of them will involve coding with Python and/or C/C++. If you are not familiar with these programming languages, please find online resources to prepare yourself before you work on the labs. Advanced knowledge of these programming languages is not required, so you can just follow online tutorials (e.g., https://docs.python.org/3/tutorial/ for Python) to learn the basics. After all, this is a course about networking, not about advanced use of programming languages.

All labs will be based on group efforts where you can form **a group of two students** to work on these labs. Having fewer students in a group is of course allowed. You should start looking for your group mates as soon as the course starts. Once you have found a group mate, you can enroll yourself in a lab group on PANDA (module Group Choice: Lab Group). You should have joined a group by the announced deadline.

If your group, unfortunately, falls apart in the middle of the course (e.g., group mate drops the course), you could choose to join another group given that you can find a group willing to accept you and the group is not full yet. You can also continue the rest labs on your own if finding another group to join is not possible. You should inform us with such group changes in advance. Once we approve the change, you will be moved to the new group on PANDA. You are not allowed to switch the group without a legitimate reason and approval.

### 2.1 Lab Assessment

Labs will be assessed with oral examinations in the form of interviews during some of the exercise sessions. You will be asked to upload your code to PANDA as an assignment submission by a deadline before the interview. This submitted version will be used for grading and any changes made after that will not be considered. During the interview, both group members must be present and the grading will be done as follows:

- We will first ask if your group has the lab completed and working.
  - If yes, show us a demo and explain what is happening.
  - If not, demonstrate to us to the best of your abilities where you are.

- Explain various pieces of your code and what design and logic you have considered to implement and why.
- We will also look at the code generality (if you are making assumptions that are reasonable or not), error handling, hardcoding numbers, among others.
- You should also follow a coherent coding style. Ugly, unmanageable code will incur penalties.

Your grade for the lab will be based on: (1) whether the code is working, (2) how well you understand what is implemented (by all group members even if you split the work), (3) how well you answer questions during the interview, and (4) how well you understand the design space and choices present in your implementation. As a result, group members may get different grades for the same lab in some cases. Detailed schedule for the interviews for each lab will be announced on PANDA in due time.

Often enough we hear from students that they have spent so much time and written so many lines of code and why they deserve a higher grade despite non-working code. We would like to point out that that is NOT how programming-based lab assignments work. The lab assignments that are given to you have been done by us carefully and we know how much they should take as a team. This is a six ECTS course, and you need to put in time and effort to get it done. A working code is the best proof of your time, energy, and effort put in this course.

### 2.2 How to Ask for Help

If you are stuck and are looking for help, please follow these steps.

- Read the code given to you carefully and check the online documentations for the tools (e.g., Mininet usage, Ryu APIs, P4 language specifications). For debugging, please look at the output and error logs carefully. Most of the time, your problem will be solved if you spend enough time doing the above.
- Ask Google. Issues with network setups and API usage could be solved with proper Googling most of the time. Make sure you have done sufficient search. Also, consider searching the GitHub issues of relevant projects and posts in online forums like Stack Overflow.
- Ask in the Discussion Forum on PANDA. If the above two steps have not solved your problem, you can post your problem in the Discussion Forum. However, you should never ask for a direct solution to a specific lab. Try to narrow down your problem and ask that specific aspect only. It is possible that other students have already posted the same problem before you, so make sure you have gone through all the problems discussed there and your problem has not been covered before you create a new post. We encourage you to constructively participate in PANDA discussions. Please respect each other and do not give out the solution to the lab directly. We will also attend to the Discussion Forum periodically to answer questions. However, you should be prepared to hear that "debugging such an issue is part of the lab assignment."
- Contact us through email. After exhausting all the above possibilities, if you believe you are really stuck and there is something fundamentally wrong, please reach out to us via the email lin.wang@upb.de. However, do not just paste the screenshot and say that something does not work. Please show us you have put in the effort (e.g., the above steps) to solve the problem. Please include details like the ones below in your email.
  - What is the problem?
  - How to reproduce it? Is it nondeterministic or does it happen every time?
  - What possibilities have you tried to narrow down the error conditions?
  - Have you searched on Google the error message? Did you find anything useful?
  - What do you suspect is happening?

  Inquiry emails without details like these may simply be ignored. Also, timely responses cannot always be guaranteed, especially towards the lab deadlines, so please start early and catch issues as early as possible.

### 2.3 How to Work in a Group

The lab deadlines are spread out based on the perceived complexity and the amount of code that needs to be written. If you show up one day before the deadline saying that your group members decided not to work and dropped the course, or any other reasons that have prevented you from completing the lab, we cannot help you in such a short notice. You should inform us early enough to find a solution.

All of you must be responsible for the group work. Here are some good practices to follow:

- Schedule one or more weekly meetings to discuss the lab. If a group member does not show up after multiple reminders, inform us immediately and notify them that they are not part of your group anymore.
- Meet early and decide in advance who does what and when to synchronize the progress. Use a version control system like `git` to keep track of issues and updates.

We will not be responsible for resolving disputes in your group. If you have trouble in your group, contact us early enough with detailed evidence of what happened.

## 3 Set Up a Virtual Machine for the Labs

The first step is to set up a virtual machine (VM) you will be using for the rest of this course. This will make it easy to install all dependencies for the labs, saving you the tedium of installing individual packages and ensuring your development environment is correct. For the labs, a Linux VM is required, and the specific Linux distribution Ubuntu 20.04 is recommended. Other Linux distributions are also supported (at least for the first three labs), but they may raise library or package dependency issues when setting up the environment for lab4 and lab5.

### 3.1 Install VirtualBox

VirtualBox is a VM provisioner (hypervisor). For Ubuntu/Debian Linux users, you can install VirtualBox simply by running the following command:

```
$ sudo apt install virtualbox
```

This will install the VirtualBox application with GUI on your computer.

For macOS or Windows users, please visit the official VirtualBox page. The download links are under the heading "VirtualBox 7.x.x platform packages". For Windows users, use all the default installation settings, but you can uncheck the "Start Oracle VirtualBox 7.x.x after installation" checkbox.

💣 We have not tested the VM setup on Windows Subsystem for Linux or Macs with Apple Silicon. Using these platforms will be at your own risk. In case of issues, we will unlikely be able to provide help.

### 3.2 Install Vagrant

We recommend using the Vagrant software to manage the VM. Vagrant comes with a large collection of pre-packed VM boxes which we can use directly with just a few simple commands. Also, the management of installed VMs is greatly simplified with a simple, unified interface.

On Ubuntu/Debian Linux you can install Vagrant with the following commands:

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
 ↪  /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
 ↪  https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
 ↪  /etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install vagrant
```

On macOS you can use the package manage `brew` with the following commands:

```
brew tap hashicorp/tap
brew install hashicorp/tap/hashicorp-vagrant
```

For other platforms, please refer to the Vagrant official webpage.

### 3.3 Download the Lab Codebase

The course labs codebase is hosted in this repository on GitHub. Please use the following command to clone the project repository to your VM.

```
$ git clone https://github.com/upb-cn/ans-ss24.git <your local dir>
```

You can also fork the repository to your own GitHub account and collaborate with your group members using the forked repository. If you have never worked with `git` before, please follow some online `git` tutorials like this one and familiarize yourself with the use of the basic `git` commands.

In the code base repository, labs are organized in different folders. Lab0 is already available under folder `lab0`. When we release a new lab, this codebase repository will be updated with a new folder (e.g., `lab1`) for the new lab. You can simply run `git pull` to retrieve the code for the new lab. You should follow the prompt and simply apply a merge with your local repository.

At the top directory of the lab repository, there is also a file called `Vagrantfile`. This file contains the basic configurations for the VM needed for the labs. You can change the resources (e.g., number of CPU cores and memory) allocated to the VM based on your host capability, by modifying the following two lines in this file. (Having 16 cores is probably too luxury for a VM on a laptop, but easy-peasy for a server!)

```
vb.memory = 8192    # 8 GB
vb.cpus = 16        # 16 cores
```

If you have installed Vagrant, you can simply type in command `vagrant up` under the current directory in the terminal to initialize and bring up the VM. Once the initialization process finishes, you can SSH into the VM with a simple command `vagrant ssh`. The username is the default one `vagrant` and the authentication is done through public keys that have been set up by Vagrant during the initialization. Moreover, Vagrant also sets up a shared folder between the host OS and the guest VM. If you navigate to `/vagrant` directory on the VM, you will find out that this directory is mapped to the directory where the VM is initialized, i.e., the top directory of the lab repository.

You can easily shut down the VM with command `vagrant halt` and bring it up again later with command `vagrant up`. For more usage information about Vagrant, you may refer to this page.

### 3.4 Set up X11 Forwarding

You can now access the VM with SSH. However, some of the tasks in the labs require launching a GUI from the VM. To do this remotely, you need an X server installed on your host machine. If you are using Linux as your host OS you already have an X server installed (good job!). If you are using Windows or macOS, before continuing with the next steps, make sure that

- macOS: XQuartz is installed. Follow the XQuartz website for instructions.
- Windows: Xming is installed. Follow the Xming website for instructions. Use the default options and uncheck “Launch Xming” at the end.

Your VM is already configured to use the X server, so running command `sudo xterm` in the VM should pop up a terminal window for the VM on your host OS.

## 4 Mininet 101

☕ *This section will be explained in the exercise session.*

Mininet is a powerful network emulator and we will use it throughout the labs. Mininet is pre-installed in the VM you have just set up. In the following, we will provide the basics of its usage and demonstrate it with an

example. You can find a detailed walkthrough on the Mininet webpage. In some parts of the walkthrough, you may see software-defined networks. We will discuss it later in this course, so you do not need to understand what they mean right now. You just need to know how to run and interact with Mininet.

### 4.1 Basic Commands in Mininet

To get started, we create a simple network by running the following command:

```
sudo mn (--topo=minimal)
```

The above command creates the default topology in Mininet (known as the `minimal` topology) which includes one switch connected to two hosts. A controller is also created but you can ignore it for the moment. Once the above command runs successfully, all four entities are running in the VM and the Mininet Command Line Interface (CLI) comes up. You can type in `help` to check all available commands in the CLI. Some of the commands are explained below.

```
mininet> nodes          # display nodes
mininet> dump           # detailed info about nodes
mininet> links          # display links
mininet> net            # display the topology
```

You can also run simple commands on a node by prepending the node ID. For example, you can run the following command to show the network interfaces on host `h1`.

```
mininet> h1 ifconfig -a
```

You can also run commands on the switch by prepending the switch ID. For example, the following command shows the details of all the interfaces on the switch.

```
mininet> s1 ifconfig -a
```

With that, you can also test the connectivity between hosts with the `ping` command.

```
mininet> h1 ping h2 -c5     # ping 5 times from h1 to h2
mininet> pingall            # check connectivity between all host pairs
```

Sometimes, you may want to run more complex scripts or programs on hosts. You can do that with `xterm`.

```
mininet> xterm h1
```

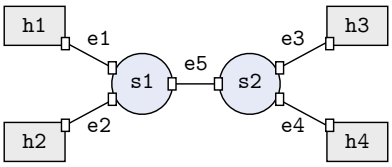
The above command opens up an `xterm` window with the shell prompt for `h1`. Recall the setup for X11 forwarding. If you access the VM via SSH, X11 forwarding is required; otherwise, you will probably only see an error message like “Error: Cannot connect to display” or similar.

Once you are done with the experiment, you can close Mininet by typing in `exit`. It is strongly recommended to clean up after exiting by running command `sudo mn -c`.

### 4.2 Build a Customized Network Topology

Mininet provides handy APIs in Python which allows us to construct customized network topologies. The major class we will use to create a topology is `mininet.topo.Topo`, which you can import to your Python script with

```
from mininet.topo import Topo
```




**Link properties:**
e1-e4: 15 Mbps, 10 ms
e5: 20 Mbps, 45 ms

**Figure 1:** A simple bridge network topology.

This class already provides a collection of methods for specifying the properties of the topology to create. Some examples are shown below:

```
def addHost(self, name, **opts)          # Add a host
def addSwitch(self, name, **opts)        # Add a switch
def addLink(self, node1, node2, *opts)   # Add a link between two nodes
```

A complete list of methods can be found on the Mininet Python API Reference Manual. In this manual, you can find other useful classes, attributes, and methods.

Using the above APIs, let us now build the network topology illustrated in Figure 1. Hosts `h1` – `h4` are represented by squares and switches `s1` and `s2` are represented by circles. The names of the devices you build in Mininet should match the names in the diagram. The hosts are assigned IP addresses `10.0.0.x` where `x` should match the host number (1 – 4), e.g., `10.0.0.2` for `h2`. The switch should have type `OVSBridge` (see here) which is just a simple Ethernet learning switch. The link properties (bandwidth, delay) are as follows: (15Mbps, 10ms) for links `e1` – `e4` and (20Mbps, 45ms) for link `e5`. These link properties can be specified with the options of `bw=<bandwidth>, delay=<delay>` in the `addLink` method. Note that such performance modeling is only possible with the `TCLink` link type (see here), which has to be specified as well.

Once you have completed the topology file, use it to launch a network in Mininet using the following command:

```
$ sudo mn --custom /path/to/topo/file --topo=<topo name>
```

We can also streamline the process of creating the topology and running a network with that topology with a Python script. This helps us automate network experiments. This is encapsulated in the `mininet.net.Mininet` class. The following lists some frequently used methods; more can be found in the Reference Manual.

```
net = Mininet(topo)      # Create a network with the topology topo
net.start()              # Start the network
net.pingAll()            # Perform ping tests
net.stop()               # Stop the network
```

### 4.3 Measure the Performance of the Network

In Mininet, use `xterm` to start terminals for the hosts in the network:

```
mininet> xterm h1 h3
```

Now, please measure the latency and throughput between hosts `h1` and `h3` using `ping` and `iperf`, respectively. `Iperf` is a handy tool for network speed tests and it supports both TCP and UDP connections. You can find more details about its usage here. For `ping` you should make 20 or more attempts and average the results and for `iperf`, you should measure for 20 seconds or longer, with the commands below. Measure with both TCP and UDP connections.

```
h1$ iperf -s (-u)
h3$ iperf -c ip.to.server -t 20 (-u)
```

### 4.4 Effect of Multiplexing

In the above measurement, we only have one connection going at the same time. Now, let us try with simultaneous connections where we let `h1` talk to `h3` and `h2` talk to `h4` at the same time. Try to predict the latency and throughput of the two simultaneous connections. Use `ping` and `iperf` to measure the latency and throughput of the two connections. Check if your predictions match the measurement results and think about why or why not. What if we have two connections all destined to `h4` (i.e., `h1 – h4` and `h2 – h4` )?