NTNU - Department of Information Security and Communication Technology

TTM4250 - Lab Assignment: Programmable Data Planes with P4

Introduction:

In this lab, you will get hands-on experience with programmable data planes using the P4 programming language. As a prerequisite, you will learn about emulating networks using Mininet and will then proceed to analyze and reason about P4 programs. Finally, you will proceed to define, implement, and deploy your own custom tunneling protocol that will allow encapsulating IP packets and replacing IP-based forwarding decisions with tunnel-based ones.

Preparation

This lab requires you to host a virtual machine locally on your own computer or on a Sahara computer. The LAB VM can be downloaded from `https://s.ntnu.no/ttm4250-p4-lab` and should be compatible with most hypervisor software. We recommend VirtualBox, as it is free, verified to work for the lab, and already available on Sahara computers.

After initializing your VM, download the starter package for this lab at `https://folk.ntnu.no/stanisll/2025/ttm4250/p4-lab.zip`. It contains some helpful documentation (P4 cheat sheet and P4 specification), the code of the program you will analyze in one of the tasks, and starter code for your tunneling project.

Task 3.1 Getting Familiar with Mininet

The P4 learning environment uses Mininet[1] to emulate various network topologies and populate them with hosts, switches, and P4-enabled devices.

Check out the basic usage of Mininet at `http://mininet.org/walkthrough/`. In particular, note how topologies are created and how you can interactively send commands to or open terminal windows on the emulated hosts.

As per the example from the walkthrough, you can define a topology and store it as a python script such as `topo-2sw-2host.py` (available in " /Desktop/Lab-TTM4250-P4/mininet-eg" on VM):

```
"""Custom topology example
Two directly connected switches plus a host for each switch:
   host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        # Explicit IP address assignment if desired, otherwise
        # incremental from 10.0.0.1
        leftHost = self.addHost( 'h1', ip = "10.0.0.5/8" )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Use the following command to launch Mininet with the custom topology.

```
sudo mn --custom topo-2sw-2host.py --topo mytopo
```

Inside Mininet, you can execute commands on the different devices by running `$deviceName $command`, e.g., `h1 ip ad sh`. Aliases for host names are also available, i.e., you can do something like `h1 ping h2`. Furthermore, you can open a terminal window on a given host via commands like `xterm h1`. For this, you will need to be connected to graphical session (X2Go / VNC / ..). You can also run Wireshark / `tcpdump` as root and capture on the emulated ports, e.g., `s3-eth1` in the above example. If you prefer to stay in the terminal, you can also attach a non-graphical session to a mininet host by using the `dump` command to find the process ID (PID) of each session, and connect to it via `sudo mnexec -a $PID bash`, cf. the example in Figure 1.

Finally, you can exit Mininet by typing `exit` or pressing `Ctrl+D`. If Mininet stopped unexpectedly, you can explicitly clean the environment via `sudo mn -c`.

[1] `http://mininet.org/`

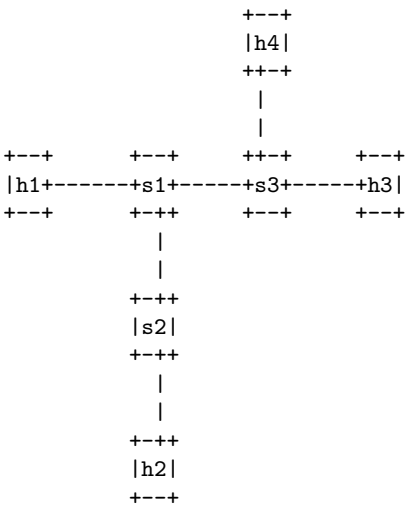


Figure 1: Using `dump` and `mnexec` to connect to a mininet host.

Task 3.2 Reading, Understanding, and Debugging P4 Programs

Extract the folder `program1` from the starter package (`https://folk.ntnu.no/stanisll/2025/ttm4250/p4-lab.zip`) and navigate into it. The network for this example is visualized below. You can find its definition in `network.py` - it uses an extended API of the Mininet library[2], but given your knowledge of Mininet and some browsing in the code, the main steps for the setup should be clear.

Note: `p4app.json` contains the same definitions and results in the same configuration, but uses a JSON config file instead of a Python script.



The line `net.setP4SourceAll('program1.p4')` instructs all P4 switches to use the same P4 program, in this case `program1.p4`, whereas `net.addP4Switch('s3', cli_input='s3-commands.txt')` allows passing specific per-switch configuration commands and table entries to individual switches.

Inside the `program1` folder, you can launch the network via `sudo python3 network.py`.

Task 3.2.1 Program Inspection

Open `program1.p4` and derive the main functionality / behavior that is implemented by the program. Discuss what information is extracted from the packet headers, how that information is used, which parts of the packets

[2] `https://github.com/nsg-ethz/p4-utils/`

are modified, and what actions are applied to the packets. Additionally, take a look at the per-switch configuration files `s*-commands.txt` and explain the resulting behavior. Confirm your hypotheses by gathering evidence using pings and packet captures at appropriate points in the network. Document this in the lab report.

Hint: consult the lecture slides, cheat sheet, and specification if you are unsure what a function / instruction does.

Task 3.2.2 Adjusting Configurations and Debugging Issues

Replace the contents of `s1-commands.txt` with the following. The only change is the last character on the second line.

```
table_set_default ipv4_lpm drop
table_add ipv4_lpm ipv4_forward 10.0.1.1/32 => 00:00:0a:00:01:01 2
table_add ipv4_lpm ipv4_forward 10.0.2.2/32 => 00:01:0a:00:02:02 2
table_add ipv4_lpm ipv4_forward 10.0.3.0/24 => 00:00:00:03:01:00 3
```

First, reason about the effect you expect this change to have on the behavior of the switch.

Then, try sending a *single* ping from `h2` to `h1` (`h2 ping -c1 h1`) and inspect the activity on the related link(s). Do you notice anything unexpected? Given your understanding of the operations that are performed in `program1.p4`, explain the observed behavior.

Finally, fix the bug.

Hint: only packets that pass the check in line 123 (`hdr.ipv4.isValid()`) are handled further, and are dropped otherwise.

Task 3.3 Implementing Your Own Tunneling Protocol Using P4

Disclaimer: this exercise is based on the *basic tunneling* exercise at `https://github.com/p4lang/tutorials`.

The goal of this exercise is to showcase the capabilities of P4 to define and implement support for arbitrary new protocols. Specifically, we will implement a custom tunneling protocol that our programmable switches will use for making forwarding decisions instead of their initial behavior of using destination IP addresses. To this end, the switches will need to parse the new protocol and match the extracted information against a newly created table that maps tunnel IDs to output ports. The new header contains information on the type of the packet being encapsulated (in this case IPv4) as well as the tunnel ID.

To get started, extract the folder `basic_tunnel` from the starter package (`https://folk.ntnu.no/stanisll/2025/ttm4250/p4-lab.zip`) and navigate into it. The network for this example is visualized below. You can find its definition in `network.py`.

Note: before starting the exercise, make the two files `send.py` and `receive.py` executable via `chmod +x send.py receive.py`.

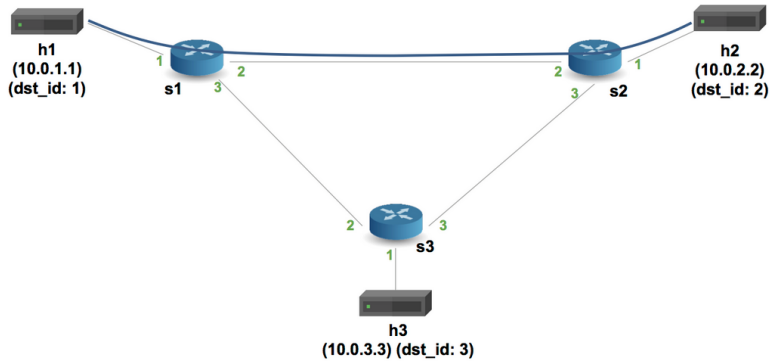


Figure 2: Topology for the basic tunneling scenario.

The program defined in `basic_tunnel.p4` implements an IPv4 forwarding behavior based on longest prefix matching on the destination IP address. The entries defined in the first three lines of each of the `s*-commands.txt` files populate the corresponding tables. Each line follows the pattern `table_add $table_name $action_name $match_parameter => $action_parameter_1 $action_parameter_2 ...` The last three lines in each file will populate the tunnel-related table once we define it, but will not have an effect for now apart from causing a few warnings on launch.

- Run the network via `sudo python3 network.py`.
- In the mininet prompt, launch terminals for h1 and h2 via `xterm h1 h2`.
- Run `./receive.py` on h2 to launch the packet receiver which will listen for incoming packets and print their contents.
- Sequentially run the following commands on h1 and observe the outputs at the sender and receiver.
  - `./send.py 10.0.2.2 "hello p4.a"`
  - `./send.py 10.0.2.2 "hello p4.b" --dst_id 2`
  - `./send.py 10.0.3.3 "hello p4.c"`
  - `./send.py 10.0.3.3 "hello p4.d" --dst_id 2` (the last character is a 2, this is not a typo :))

Given what you know about the program, the current table setup, and your observations, explain the behavior in terms of receiving / not receiving packets at h2 in each of the four cases.

Next, it is time to actually implement the desired new behavior: the switches shall forward regular IPv4 packets as usual, but additionally also support tunnel-based forwarding if incoming packets use the newly defined tunneling protocol to encapsulate IPv4 packets. The code of `basic_tunnel.p4` comes with several `NOTE` and `TODO` markers which contain information on already added functionality and instructions regarding the required additions, respectively.

- NOTE: A new header type has been added called `myTunnel_t` that contains two 16-bit fields: `proto_id` and `dst_id`.
- NOTE: The `myTunnel_t` header has been added to the headers struct.
- TODO: Update the parser to extract either the `myTunnel` header or `ipv4` header based on the `etherType` field in the Ethernet header. The `etherType` corresponding to the `myTunnel` header is `0x1212`. The parser should also extract the `ipv4` header after the `myTunnel` header if `proto_id == TYPE_IPV4`, i.e., `0x0800`.
- TODO: Define a new action called `myTunnel_forward` that simply sets the egress port (i.e., `egress_spec` field of the `standard_metadata` bus) to the port number provided by the control plane.
- TODO: Define a new table called `myTunnel_exact` that performs an exact match on the `dst_id` field of the `myTunnel` header. This table should invoke either the `myTunnel_forward` action if the there is a match in the table and it should invoke the drop action otherwise.
- TODO: Update the apply statement in the `MyIngress` control block to apply your newly defined `myTunnel_exact` table if the `myTunnel` header is valid. Otherwise, invoke the `ipv4_lpm` table if the `ipv4` header is valid.
- TODO: Update the deparser to emit the `ethernet`, then `myTunnel`, then `ipv4` headers. Remember that the deparser will only emit a header if it is valid. A header's implicit validity bit is set by the parser upon extraction. So there is no need to check header validity here.

In case of errors regarding compilation, switch configuration, or other unexpected behavior, you can consult the output of the main program (`sudo python3 network.py`), the automatically generated `log` folder, and packet dumps that are stored in the `pcap` folder.

After completing your implementation, run the four commands between h1 and h2 again and discuss the updated behavior.

Finally, reflect about the advantages and disadvantages of implementing tunneling in P4, and compare that to using established technologies such as MPLS, GRE, or VXLAN. What are some extensions beyond the current functionality of the `basic_tunneling.p4` program that would make it more usable in a practical setting?

Submission details

Please submit an archive (.tar.gz or .zip) which contains all files you want to submit. Please have your group number in the file name and the directory name. Submission channel will be shared during the lectures. A report (one single PDF file, named p4-lab-group(num).pdf) containing the following elements is mandatory:

- Your group number on the first page.
- For all questions: state your assumptions, report what you did, describe what you observed, and explain your conclusions and thought process.
- Include any p4-code edited and used in the lab.
- If you have used Artificial Intelligence, state which AI tools have been used for what in an AI usage section. It is ideal if you include your AI usage directly in your report.

Include scripts that were used to compare the results .csv. If you have used scripting or commands other than the provided ones to encode the videos, include these scripts in your delivery.

We can only grade what we find in your submission and what we understand. Please state your assumptions and observations as clearly as possible.

Due date: November 15, 2026, 11:59 PM