Project responsible: Marco Chiesa mchiesa@kth.se
Giacomo Verardo verardo@kth.se

P4 Project: “VPLS Layer 2” virtualization

Getting started

Prerequisites: Basic Python2 knowledge is required to obtain higher grades than D. You can follow this simple tutorial in case you are not familiar with Python:

- https://www.codecademy.com/learn/learn-python
- You should be familiar with the topics covered in:
  - Python syntax.
  - Conditionals and Control flow.
  - Functions.
  - Lists & Dictionaries.
  - Loops.
- If you prefer a textual traditional tutorial, check:
  - Numbers: https://docs.python.org/2/tutorial/introduction.html#numbers
  - Strings: https://docs.python.org/2/tutorial/introduction.html#strings
  - Lists: https://docs.python.org/2/tutorial/introduction.html#lists
  - Control flows: https://docs.python.org/2/tutorial/controlflow.html
  - Dictionaries: https://docs.python.org/2/tutorial/datastructures.html#dictionaries
  - Also useful: some basic “module “knowledge.

Setting up the VM (do it before the first lab)

Homework before the first lecture:

- Install the provided VM using the OVA image (if you haven’t done it already for the STP lab)
  update link
- The VM is based upon the P4 tutorial created at ETH Zurich:
  https://github.com/nsg-ethz/p4-learning

- Shared folder tip (strongly suggested):
  - Set up a shared folder between your host OS and the VM. This will allow you to edit files in the VM directly from your host OS with your favorite editor.
  - To do it, open VirtualBox and click on the p4-learning VM. Then click on settings on the top bar. Now click on “Shared folders” and click on the folder with the “plus” sign on the right. Set the folder path to the folder where you want to see the files shared with the VM. Set the folder name as you wish. Check both “auto-mount” and “make Permanent”. Set the mount point to “/media/shared/”. This means that every file in the /media/shared folder of your VM will appear in the shared folder of your host OS.
  - Copy all the exercises files of the P4 tutorial in the VM shared folder:
    ```
    cp -r ~/p4-tools/p4-learning/exercises/ /media/shared/
    ```
  - You will now see all the exercise files in the shared directory directly on your host. You can edit the files. Every time you save them, the changes are reflected in the VM as well. You will be able to run P4 directly from this shared folder inside the VM (see more details in the tutorial videos).

Homework and tentative schedule

February 2 (9:15am-12am)

- P4 programming introduction, tutorial for exercises from the ETH tutorial 01, 02.
- Assignments for next lab: redo exercises 01 and 02, plus independent work on 03.

February 10 (9:15am-12am)

- Q&A for 03.
- L2 learning and ECMP tutorial (basically 04, 05). Project presentation.
- Start forming groups!
- Assignment for next lab: redo 04, 05. Do 08 and our register-based tutorial.
- Assignment for next lab: prepare a pipeline schema and think about packet headers.

February 17 (9:15am-12am)

- Python controller lab (exercise 08) and registers.
- Final groups deadline. Students should decide who will report on a subpart of the work during the following meetings.
- Q&A for the project.

February 26 (9:15am-12am)

- Q&A for the project.
- Per-group ~20' meeting. Pipeline presentation from the students

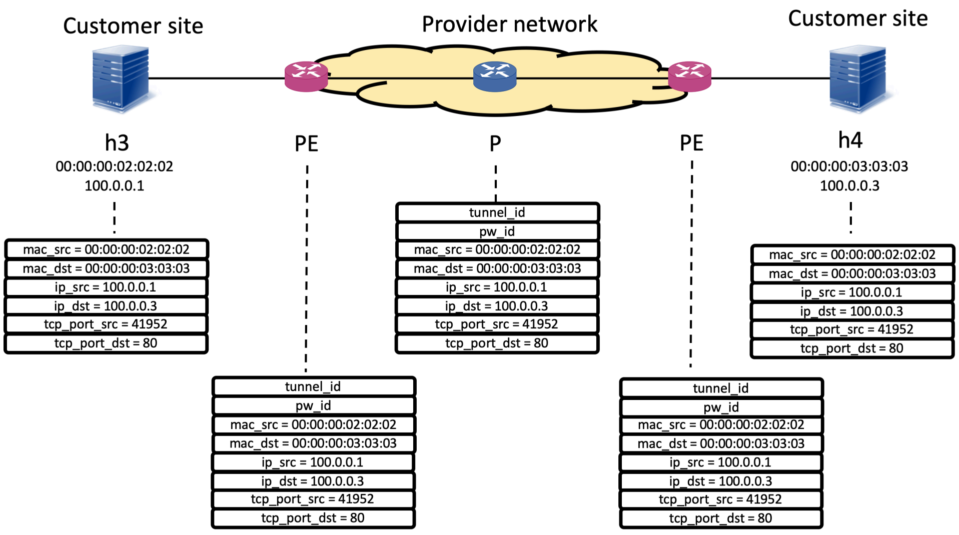
March 4 (1:15pm-3pm)

- General Q&A for the project.
- Per-group ~20' meeting.

Project description: VPLS virtualization

Simplified VPLS overview

Your goal is to implement a VPLS-like provider network to offer Ethernet-based virtualization to your customers. All hosts within a customer network belong to the same IP subnet. The provider network consists of provider edge (PE) nodes, which interconnect customer devices to the provider network, and provider router (P), which are internal to the provider network. When a customer sends a packet to a PE device, the PE device is responsible for forwarding the packet to the intended destinations. A destination may be attached to the same PE device or to a different PE device. Packets between different PE devices must be encapsulated into a tunnel identified with a `tunnel_id`. Packets received from a customer node are labelled with a proper id (we call it `pw_id`).
The picture below shows the transmission of a packet from one customer node to another. In this figure, we assume that the sending host already knows the MAC address of the destination and that the left-most PE device knows which tunnel to use to reach the destination MAC address. Note that you may have to add or remove some header fields in order to implement VPLS in P4 (think how you will parse a packet).



Project goal. Your goal is to build VPLS functionalities in P4 according to the tasks explained after in this document.

Final submission

You will submit your source code as a group and an individual report explaining your work. Do not use more than 10 pages, minimum font 12pt, do not change the margin.
Readability of the report is taken into account for the report.
For each task in the project, it is important that you discuss the following aspects in the report:

- What is the task about?
- What have you implemented and what is missing?
- Is there any limitation in the current implementation?
- How does the data-plane pipeline look like?
- Why did you choose this pipeline?
- How is the pipeline populated (i.e., how does the control-plane work)?
- How did you test that the code satisfies the project requirements? Provide the commands that you ran.
- Explain the contributions of each member in the group.

You can use the above questions as the headings for the report.
We will run and test your code based on your instructions.
The code must work! Copy/pasting code from the ETH tutorial is allowed.

Deadline (strict): March 18, 5pm.

Points

There is a maximum of 40 points obtainable on the P4 programming. 30 points can be obtained through the project and the remaining 10 during the written exam. One has to take at least 10 points on the project and 4 points during the written exam in order to pass the project part.

Points are divided into basic, intermediate, and advanced points in both the project and the written exam. Note that the quality of the code and the short report is taken into account for the final grade. The granularity of the points is 0.5 units.

We will provide six different topologies, of increasing level of difficulty, that should be addressed during the project.

Tasks

There are five tasks that should be addressed throughout the project.
Each task consists of both data-plane and control-plane programming.

You can assume that:

- packets from the customers are at most 1KB
- the location of the customers' host is given as input (see below the topology .conf files). More specifically, you know to which switch of the provider network is a customer host connecting to.
- You can assume that you know the MAC and IP address of the customer host except in the third task of the project. In that task, you need to learn them at run-time.
- customers' hosts connected directly to the PE, *i.e.*, no customer edge

You **cannot** assume that:

- all customers' packets are IP or TCP
- customer's IP and MAC addresses do not overlap

Total points overview

Level	Task #1	Task #2	Task #3	Task #4	Task #5	Points
Basic	8	6			1	15
Intermediate			3	4	3	10
Advanced			2	2	1	5

TASK #1: Unicast forwarding over provider tunnels (8 points)

The goal of this task is to enable basic communication among the customers.
Requirements:

- traffic from one customer should not reach another customer
- the data-plane pipeline can be configured *statically* (i.e., using txt files) or automated with L2 learning --- L2 learning is the third task of this project. To make things clear, if you plan to implement task #4, you can rely on the L3 learning from task #4 and do not need to submit the *static* configuration files. If you do not plan to implement task #3, you should submit the static files for each topology that you will test.
- adding a new customer host should only require modifying the configuration of the switch where the host has been connected. Hint: pay attention to how you defined the labels.

You can assume that:

- Auto-ARP-tables is set to true so there are no ARP requests in the network.
- You know in advance the MAC addresses and IP addresses of all the customer's nodes.

Points evaluation:

- Pipeline correctness: 2 basic points
- Control-plane:
  - Tunnel configuration: 1 basic point
  - L2 forwarding: 1 basic points
- Testing (choose one):
  - 1 basic points if it works on 3 topologies
  - 2 basic points if it works on all topologies
- Report documentation: 1 basic points

How do we test this case?

- We use the test_topology_0n.sh script (n=0,1,2,3,4,5,6). This is only testing TCP packets.
- We will also manually look at the code to check whether all the requirements and assumptions are satisfied.

TASK #2: ECMP multipath forwarding (6 points)

In this task, you are expected to support ECMP forwarding inside the provider network.
Requirements:

- Same requirements as in Task #1, plus the following ones
- All packets with an IP header and TCP/IP header from one provider edge switch to another provider edge switch should be load balanced across all the shortest paths (in terms of hops). All the other type of traffic can be forwarded on a single path.

You can assume that:

- Same as Task #1 plus the following ones
- In case you support Layer 2 learning, Layer 2 flooded packets (e.g., ARP packets, MAC flood) should not be load balanced across multiple shortest paths even if they contain an IP and TCP/IP header. Note that also ARP packets should not be load balanced.

Points evaluation:

- Pipeline correctness: 2 basic points
- Control-plane:
  - ECMP group configuration: 1 basic points
- Testing (choose one):
  - 1 basic points if it works on 3 topologies
  - 2 basic points if it works on all topologies
- Report documentation: 1 basic points

Task #3: Multicast forwarding over provider tunnels (5 points)

In this task, you should make the network capable of handling multicast packets such as ARP requests. An ARP request is sent by a host to learn about the MAC address associated to the IP address of another host. This means the hosts do not know the MAC addresses of the other hosts.
Requirements:

- Flood ARP packets towards all the sites where a customer is located.

You can assume that:

- Same assumptions as in tasks #1 and #2 except for "Auto-ARP-tables is set to true so there are no ARP requests in the network"
- ARP packets do not need to be load balanced across multiple paths.

You **cannot** assume that:

- Auto-ARP tables are set to true. This attribute must be set to false.

Packets with an unknown destination MAC should be flooded towards all the sites where a customer is located.

Points evaluation:

- Pipeline correctness: 1 intermediate points
- Control-plane:
  - Multicast group configuration: 1 advanced points
- Testing (choose one):
  - 1 intermediate points if it works on 3 topologies
  - 1 intermediate + 1 advanced point if it works on all topologies
- Report documentation: 1 intermediate points

Task #4: Layer 2 learning (6 points)

In this task, you will implement an event-driven controller that populates the forwarding pipeline of the provider network based by learning the MAC addresses of the hosts similar with a Layer-2 learning approach.

Requirements:

- When a PE device receives a packet from an adjacent customer's host and it does not know the destination MAC address, it floods the packet to 1) all the hosts of the customer attached to that PE device and 2) all the tunnels that reach a PE device connected to a host of the same customer.
- When a PE device receives a packet from a tunnel and it does not know the destination MAC address, it only floods the packet to all the hosts of the customer attached to the PE device.

You can assume that:

- Same assumptions as in tasks #1 and #2 except for "Auto-ARP-tables is set to true so there are no ARP requests in the network"
- Layer-2 flooding does not need to be load balanced using the ECMP paths.

Points evaluation:

- Pipeline correctness: 1 intermediate point
- Control-plane:
  - Multicast group configuration: 2 advanced points
- Testing (choose one):
  - 1 intermediate points if it works on 3 topologies
  - 2 intermediate points if it works on all topologies with automation
- Report documentation: 1 intermediate point

How to test the code?

- Check the controller receives packets from the switch and generates rules to be installed into the switch
- use h1 arp -n for each host and verify all the tables are empty.

Task #5: infer the Round-Trip-Time (RTT) between customers' hosts (6 points)

In this task, your goal is to infer the Round-Trip-Time (RTT) among all the hosts of the customers. The RTT is defined as the time that it takes for a packet to travel back-and-forth

from one host to another, *i.e.*, the time for a packet to go from a host to another one and back.
Requirements:

- The RTT should be inferred in the provider network without involving the controller for the measurement.
- Once the network infers an RTT, it generates a message to the controller with the following format:
    - Ethernet header should be active with an Ethertype of 0x5678
    - An RTT header should follow the Ethernet header. The format of the RTT header is provided in the skeleton code. You should report the customer ID, the IP addresses of the involved hosts and the inferred RTT value
    - The network should be able to infer the RTT of the all the active connections traversing a switch with probability of at least 99% when the number of active connections traversing a switch is at most 5.

You can assume that:

- Same assumptions as in Task #1. You can however have stricter assumptions if you also implemented the other tasks.

You <u>cannot</u> assume that:

- You can send packets to the controller for performing the measurement. Only the inferred RTT should be sent to the controller.

Points evaluation:

- Pipeline correctness (choose one):
    - 1 basic point if it works when there is <u>at most</u> one single TCP connection traversing a switch, *i.e.*, you do not support the last requirement.
    - 1 basic + 1 advanced point if supports the last requirement
- Testing:
    - 1 intermediate points if it works on 3 topologies
    - 2 intermediate points if it works on all topologies with automation
- Report documentation: 1 intermediate points

Provided code

We provide you with the code describing the different topologies and a general control-plane skeleton.

Topology files

Each topology has two files: a "`X-p4app.json`" and a "`X-vpls.conf`". These files should not be modified. The `p4app` file contains the topology information as well as IP/MAC address configuration of the hosts. The `vpls` file contains information about where the different hosts are connected. This file is loaded by the provided control-plane skeleton (must be passed as the first argument).

Control-plane skeleton code

For some exercises involving automated control-plane interaction, we provide you with a skeleton of the controller code. The code is located in `routing-controller.py`. You are not forced to use it. The controller skeleton consists of three parts:

1. The "main" code, which creates a RoutingController object, invokes the main() function and then instantiate *n* EventBasedController controllers and start them.
2. The EventBasedController is a per-switch controller. It is instantiated using a params object, which is a dictionary containing information that you want to pass to the specific controller. This controller listens for incoming packets from its associated switch on the 'cpu_port' (see exercise 04-Learning).
   When the controller receives a packet, it extracts the Ethernet header and then casts the Ethernet payload into either the CPU or the RTT header. The CPU header can be used for the L2 learning task while the RTT header (do not modify it) should be used to output the measured RTTs. You have to define the format of the CPU header. The packet is then processed using the 'process_packet()' function. You should implement the learning logic here.
3. The RoutingController is a global controller that can interact with all the switches. You should use this controller to install tunnel forwarding entries, ECMP groups, and multicast operations. The main logic should be contained in the 'process_network()' function. See exercise 08-Simple-Routing as an example.

To use the skeleton code you should:

- download the skeleton code from Canvas. You will find a link it in the "Project" module:
  https://kth.instructure.com/courses/21523/modules/items/265881
- unzip the skeleton code and move it to a folder shared with the VM

Debugging the code tips:

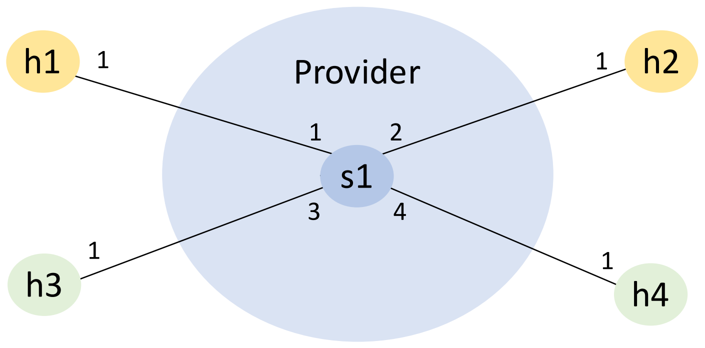
- You can use the "truncate -s 0 s{1,2,3,4,5,6}.log" command to erase your log files so that you can see the "new" logs when you launch your pings. You don't thus have to remember the line of code where it starts (as suggested in the videos).

```
from …
import …

class CpuHeader(Packet):
    …
    ###define your own CPU header

class EventBasedController(threading.Thread):
  …
    def run(self):
        sniff(iface=self.cpu_port_intf, prn=self.recv_msg_cpu)

    def recv_msg_cpu(self, pkt):
      …
      if packet.type == 0x1234:
           …
            self.process_packet([(None)]) # TODO: pass some useful parameters to the function

      elif packet.type == 0x5678:
            rtt_header = RttHeader(packet.payload)
            # the packet information is passed to a function that
            # will print the value on the terminal
            self.process_packet_rtt ([(rtt_header.customer_id,
                                       rtt_header.ip_addr_src,
                                       rtt_header.ip_addr_dst,
                                       rtt_header.rtt)])


    def process_packet(self, packet_data):
        ### TODO: write your learning logic here
        ### use exercise 04-Learning as a reference point

class RoutingController(object):
 …
    def process_network(self):
        ### TODO: logic to be executed at the start-up of the topology
        ### hint: compute ECMP paths here
        ### use exercise 08-Simple Routing as a reference

if __name__ == "__main__":
    …
    for sw_name in controller.topo.get_p4switches().keys():
        …
        thread = EventBasedController(params) ### pass parameters to the EventController here
 …
```

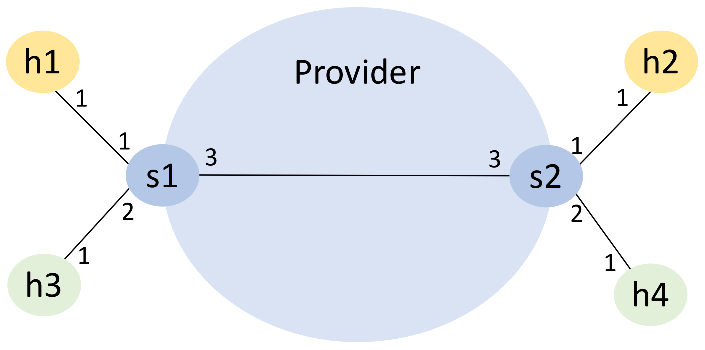
Topologies

Physical port numbering on the links

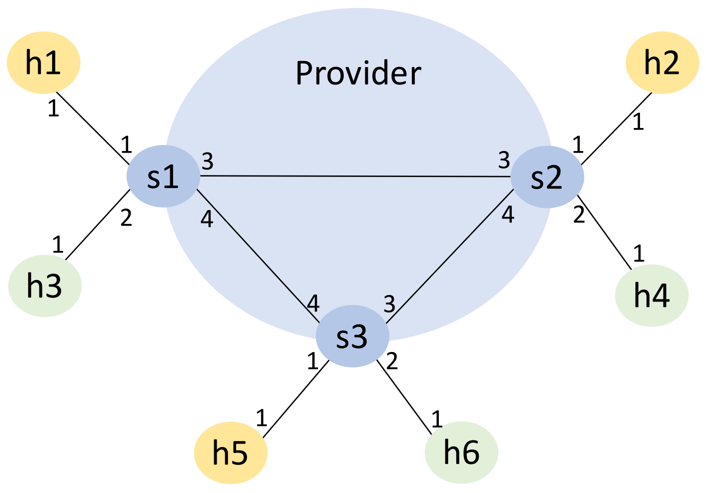
01 – Methane



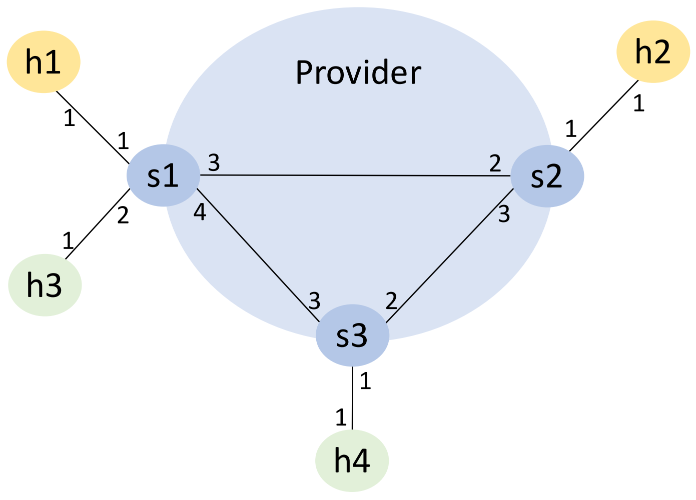
02 – Ethylene



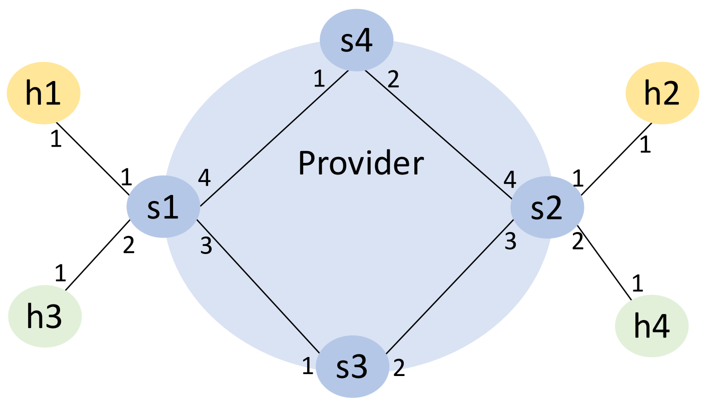
03 – Cyclopropane



04 - Rooster

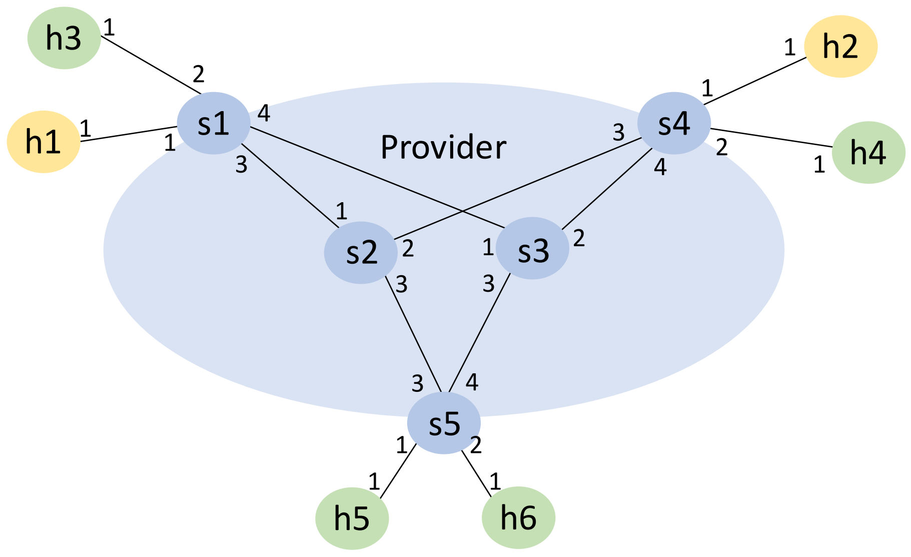


05 – Dioxetane



Project responsible: Marco Chiesa mchiesa@kth.se
Giacomo Verardo verardo@kth.se

06 – Arius


h3
h1
s1
Provider
s4
h2
h4
s2
s3
s5
h5
h6
1
2
3
4