Software Defined Networking

Lab Work 4

Jeremias Blendin, Leonhard Nobach, Christian Koch, Julius Rückert, Matthias Wichtlhuber

PS - Peer-to-Peer Systems Engineering Lab
Dept. of Electrical Engineering and Information Technology
Technische Universität Darmstadt
Rundeturmstr. 12, D-64283 Darmstadt, Germany
http://www.ps.tu-darmstadt.de/

24. October 2014

Lab 3 Task 1 Example Solution

- Differences between simple_switch.py and simple_switch_13.py
  - Table miss
    - OpenFlow 1.0
      - Single table
      - Implicit rule to send packets to controller
    - OpenFlow 1.3
      - Multiple tables
      - Default behavior drops packet
      - For every new switch
        - Install rule with priority 0 that sends table misses to controller
  - Actions vs. Instructions
    - OpenFlow 1.0
      - Actions only
    - OpenFlow 1.3
      - Instructions and Actions
      - Use OFPIT_APPLY_ACTIONS to get the same behavior as in OpenFlow 1.0
  - Code improves handling of buffering for „packet_in"
    - OpenFlow 1.3 does differentiate between buffered packets and unbuffered packets

Lab 3 Task 2 Example Solution (1)

3. Create an OpenFlow 1.3 version of simple_switch_filter.py.
   - Example solution by Daniel Hermann
   - Use first table for Filtering incoming packets

```
# Add rule for this mac at this port to go to table 1
match = parser.OFPMatch(in_port=in_port, eth_src=src)
instructions = [parser.OFPInstructionGotoTable(1)]
self.add_flow(datapath, 200, match, instructions, None)

# Drop all other packets
instructions = []
match_other = parser.OFPMatch(in_port=in_port)
self.add_flow(datapath, 100, match_other, instructions, None)
```

   - Use second table to forward packets

```
actions = [parser.OFPActionOutput(out_port)]
instructions =
[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                                      actions)]
self.add_flow(datapath, 1, match, instructions, None, 1)
```

Lab 3 Task 2 Example Solution (2)

- Differences between simple_switch.py and simple_switch_13.py
  - Example solution by Daniel Hermann:
    "Using OF 1.0, we cannot use two different tables to achieve the filtering, we thus need one flow entry per (src,dst) pair. In the worst case, all devices are connected to one switch, thus there is no inter-switch links which simplify the rules."

$$UB_{OF1.0} = n * (n-1)\ Flow\ Entries$$

$$UB_{OF1.3} = 3 * n\ Flow\ Entries$$

  - Note that depending on your implementation the value for OpenFlow 1.3 might be slightly different
  - Take-away message:
    - OpenFlow 1.0 with one flow table is not scalable for complex rulesets
    - OpenFlow > 1.1 is required for that, enables linear growth of number of rules in many cases

Lab Work 4

TECHNISCHE UNIVERSITÄT DARMSTADT

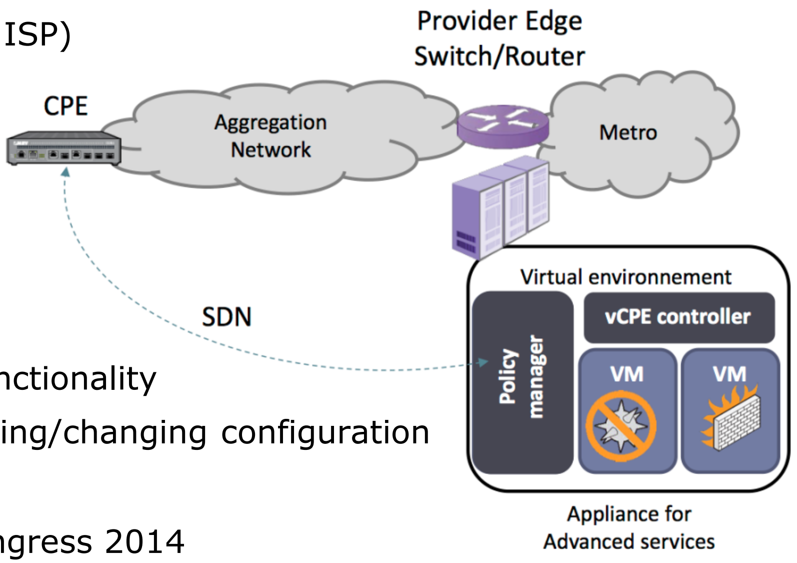
Building a Virtual CPE

Submission Deadline: January 13, 2015, 16:00CET

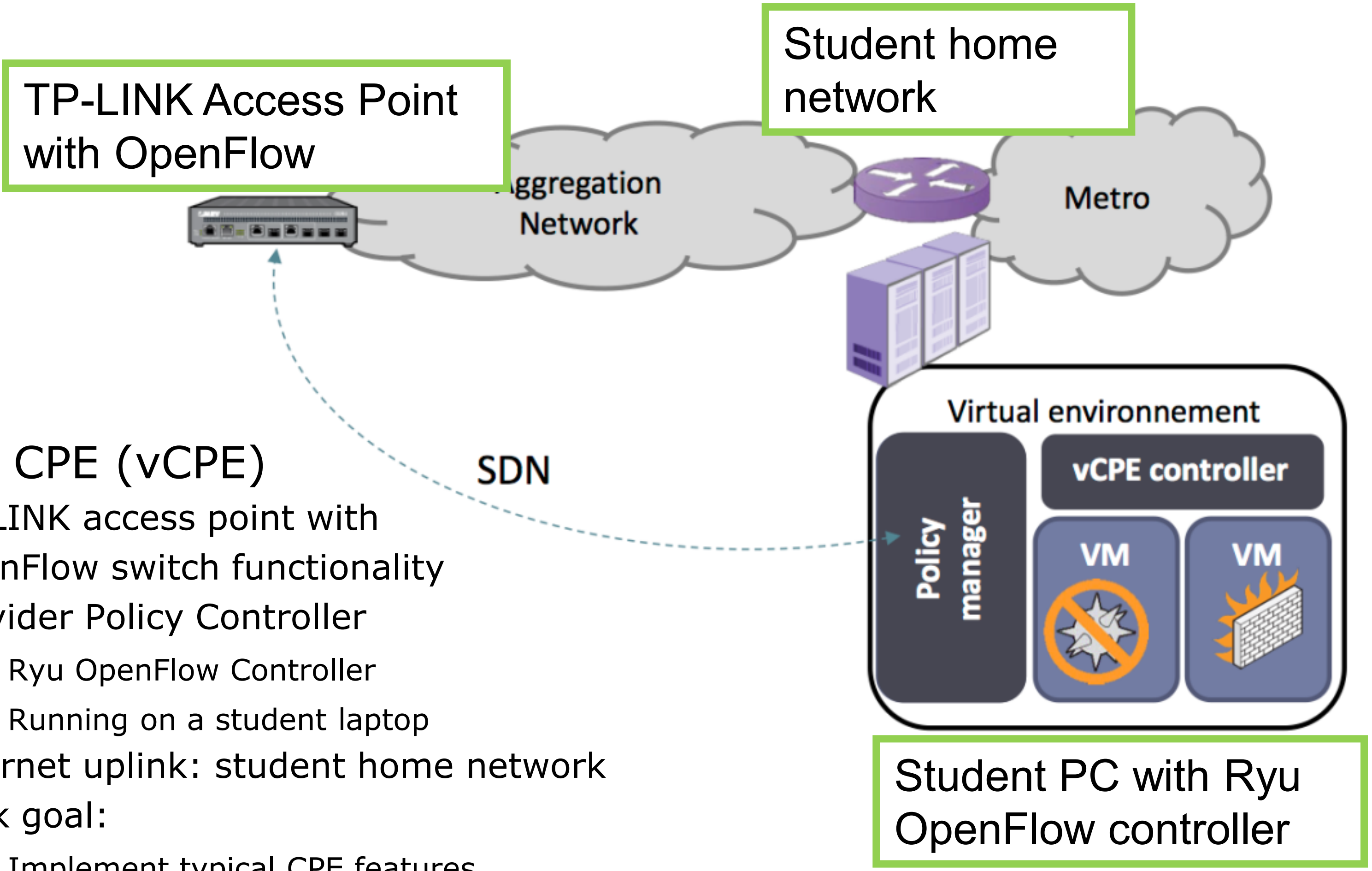
Virtual CPE

- Customer Premises Equipment (CPE)
  - The customer’s telecoms/network equipment from the viewpoint of an ISP
  - Todays devices (e.g. Fritz!Box)
    - Small router/access point
    - Static configuration (from the viewpoint of the ISP)
- Virtual CPE (vCPE)
  - Adaptable CPE
    - Operate required functionality only
    - Install new functionality on demand
  - Why?
    - Flexibility: No new devices required for new functionality
    - Costs: no need to sent a technician for debugging/changing configuration
- Currently a hot topic in the industry
  - Several talks on the topic during the SDN World Congress 2014
  - Work on progress e.g. in the US, there the CPE is often owned by the ISP
    - This is not the case in Germany
  - However, not everyone likes the idea ☺

    http://www.theregister.co.uk/2014/07/22/the_global_conspiracy_to_lobotomise_your_router/



Task Scenario



- Virtual CPE (vCPE)
  - TP-LINK access point with OpenFlow switch functionality
  - Provider Policy Controller
    - Ryu OpenFlow Controller
    - Running on a student laptop
  - Internet uplink: student home network
  - Task goal:
    - Implement typical CPE features

Image source: Wilkinson: “The Critical Role of the CPE”, SDN World Congress 2014

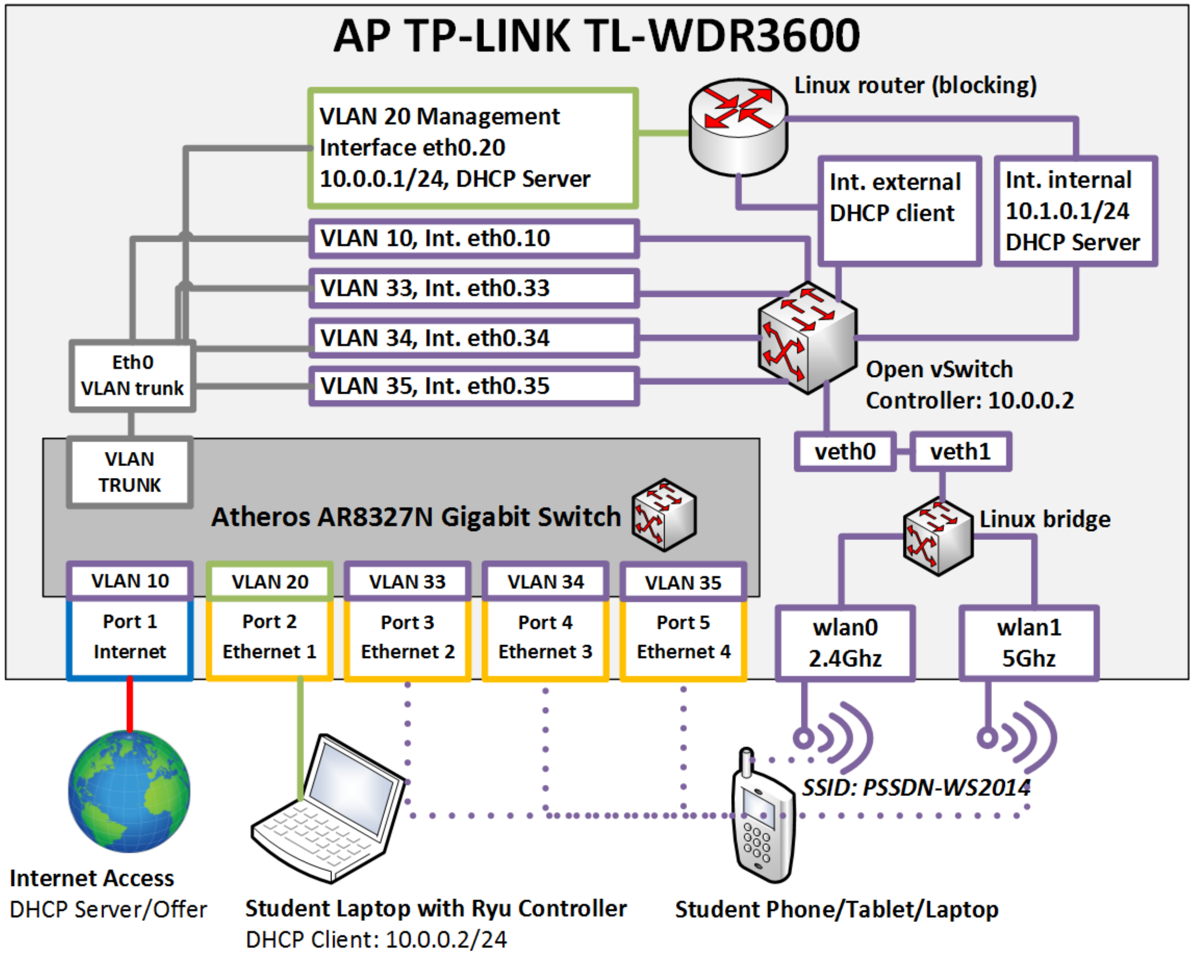
Access Point TP-LINK WDR3600 (1)

- Hardware
  - System: AR9344 MIPS 74Kc SOC: CPU 560Mhz, 128Mb Memory
  - Networking: 1x GbE, Switch ASIC Atheros AR8327N, 2.4 & 5 Ghz Wifi
  - Details
    - Device: http://wiki.openwrt.org/toh/tp-link/tl-wdr3600
    - Switch ASIC: http://wiki.mikrotik.com/wiki/Manual:Switch_Chip_Features

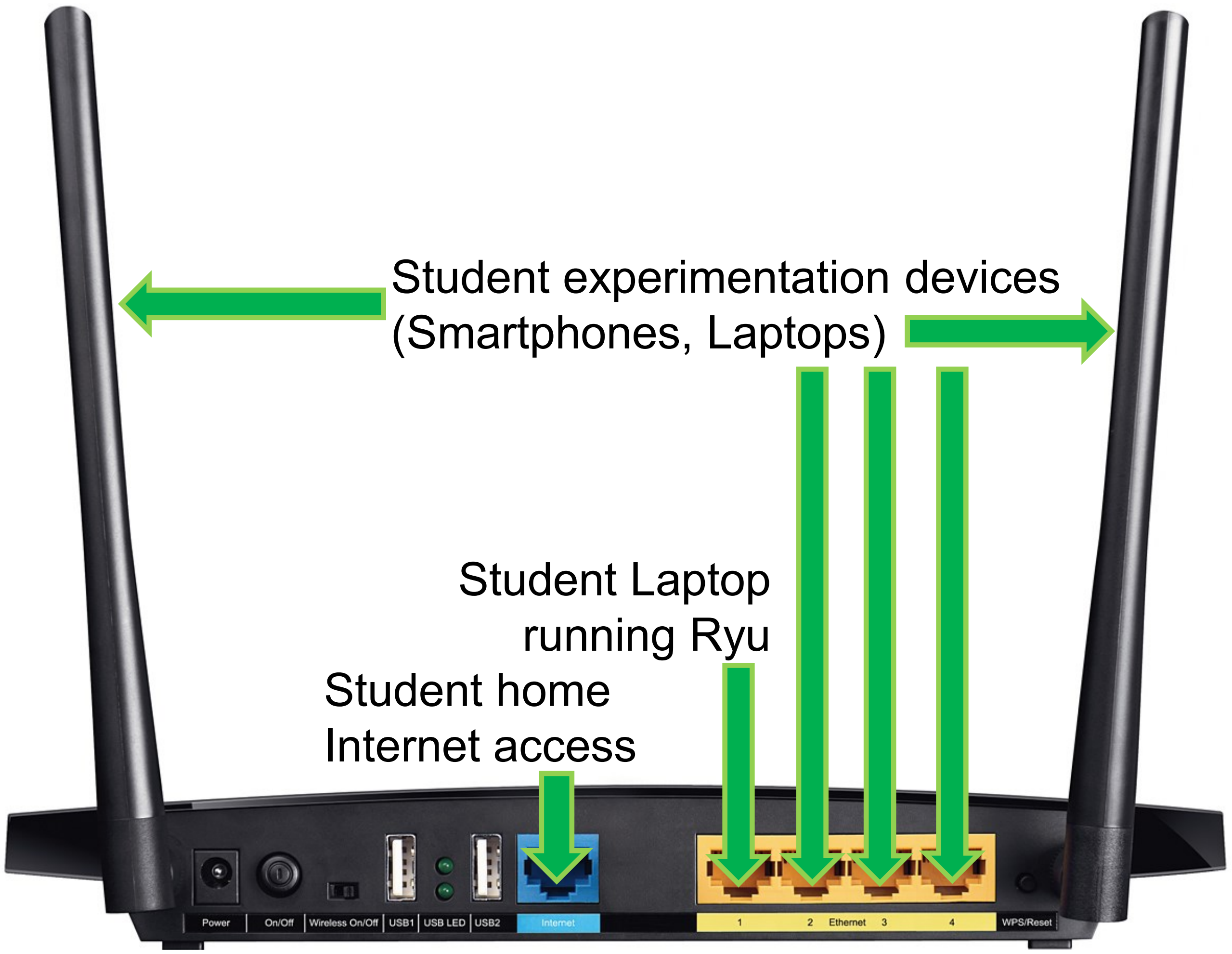
- Software
  - Default firmware is not used
  - OpenWRT OS
    - URL: https://openwrt.org/
    - An embedded Linux distribution with a large software repository
  - Custom Image
    - OpenFlow Switch: Open vSwitch v2.3
    - Wifi:
      - SSID: PSSDN-WS2014
      - Key/password: 3RwGmr2EXX
    - Detailed Network configuration: next slide

Access Point TP-LINK WDR3600 (2)

- Management
  - Port 1 (yellow)
  - Attach laptop here
  - OpenFlow switch connects to 10.0.0.2
- OpenFlow Switch
  - Physical ports
    - Internet port (Blue)
    - Ports 2,3,4 (Yellow)
    - Wifi 2.4 & 5 Ghz
  - Virtual ports
    - External: DHCP client
    - Internal: DHCP Server

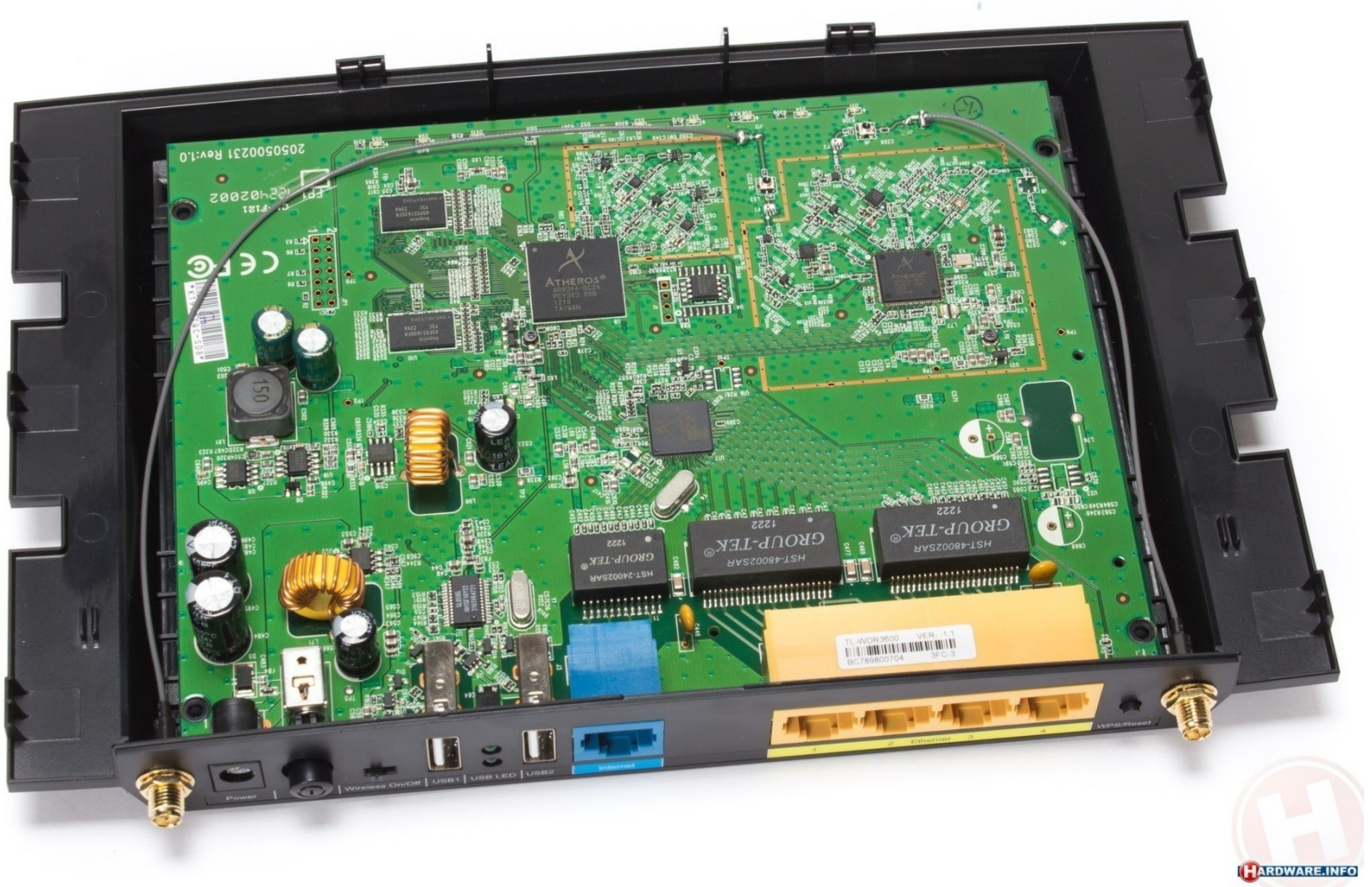


Access Point TP-LINK WDR3600 (3)



Source: http://www.tp-link.com.de/products/details/?model=TL-WDR3600

Access Point TP-LINK WDR3600 (4)



Source: http://uk.hardware.info/reviews/3423/2/tp-link-tl-wdr3600-and-wdr4300-review-two-shades-of-black-almost-identical

Lab Setup

- Open vSwitch OpenFlow Features
  - OpenFlow 1.3
  - Detailed documentation:
    - https://github.com/openvswitch/ovs/blob/branch-2.3/OPENFLOW-1.1%2B
    - http://osrg.github.io/ryu-certification/switch/ovs (slightly different version)
- Ryu Program
  - Starting point for you: simple_switch_13_homerouter.py
  - Maps the OpenFlow switch ports to their usage in our setup
  - Emulates two switches:
    - Switch „External"
      - Physical „Internet" port (blue)
      - Virtual port „external" running the DHCP client
      - Connect to student home network -> „external" port receives IP address through DHCP
    - Switch „Internal"
      - Physical ports „2,3,4", Wifi 2.4,5Ghz
      - Virtual port „internal" running the DHCP server
      - Connect student experimentation devices -> device received IP address through DHCP

Run simple_switch_13_homerouter.py

- Start ryu application:

```
~$  ryu-manager ./simple_switch_13_homerouter.py
loading app ./simple_switch_13_homerouter.py
loading app ryu.app.ofctl.service
loading app ryu.controller.ofp_handler
instantiating app ryu.app.ofctl.service of OfctlService
instantiating app ./simple_switch_13_homerouter.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
OFPGetConfigReply received: flags= miss_send_len=2000
Found port semantic=local_3 number=5
Found port semantic=local_2 number=4
Found port semantic=local_4 number=6
Found port semantic=external number=1
Found port semantic=local_ext number=7
Found port semantic=local_int number=2
Found port semantic=wlan number=3
Topology: external=[7, 1] internal=[5, 4, 2, 6, 3]
ext_swc packet in 256040841356879 22:b8:d0:41:0d:96 f:f:f:f:f:f 7 (local_ext)
```

Task 1: Addressing and Discovery

1. Extend simple_switch_13_homerouter.py
   1. Discover the external network, store the following information
      1. IP and MAC address of the local virtual port „external"
      2. IP and MAC address of the default gateway for your device
   2. Discover the internal network, store the following information
      1. IP and MAC address of the local virtual port „internal"
      2. IP and MAC address of all connected student experimentation devices
   3. Reply to pings send to the virtual ports „external" and „internal"
      1. ICMP requests should be intercepted and replied to by the OpenFlow controller

- Approach
  - Ryu includes and extensive library for packet parsing
    - Documentation: http://ryu.readthedocs.org/en/latest/library_packet_ref.html
  - Watch and extract information from the DHCP packets for subtask 1.1 and 1.2
  - Intercept and extract the ICMP request, craft reply packet for subtask 1.3

Task 2: Source NAT

2. Extend simple_switch_13_homerouter.py
    1. Implement Source NAT (Network Address Translation) for traffic from devices connected to the internal switch which is addressed to the virtual port "internal"
        1. This network function is implemented on virtually every home router
            1. Should be a well known concept from KN1/KN2 (if not, Wikipedia helps ☺)
        2. Process
            1. If packet is send from a student experimentation device
            2. If packet is addressed to MAC address of virtual port „internal" but the IP address destination is another IP
            3. Rewrite source IP and MAC address and UDP/TCP port number before sending it to the default gateway of the virtual port „external"
            4. Set source IP and MAC address to the ones of the virtual port „external"
            5. Select a unique port number of each connection
        3. Requirements
            1. Install a new rule for every flow and set a timeout for the flows (why?)
            2. A correct implementation enables full Internet access for student experimentation devices

- Approach
    - Use OpenFlow features for matching and rewriting addresses and port numbers

Task 3: Guest WIFI

3. Extend simple_switch_13_homerouter.py
   1. Implement "Guest WIFI" functionality
      1. All devices connect to the Wifi are considered „Guests“
      2. Guests should be able to access the Internet, however must not be able to access other devices connected to the physical ports 2, 3, or 4
      3. Guests should only be able to use the HTTP or HTTP protocols
         1. Identify the protocols by their well known port numbers (no further inspection is required)

Task 4: Be Creative

4. This task is not mandatory
    1. The first three tasks are too simple and boring?
    2. Come up and implement an interesting and useful feature on the Access Point
        1. We will present the best ideas in the lab
        2. Surprise us!

Programming with Ryu

- Resources
  - http://osrg.github.io/ryu/
  - Documentation http://ryu.readthedocs.org/en/latest/index.html
  - Book with examples http://osrg.github.io/ryu-book/en/html/

- Things to be aware of
  - Ryu relies on a concurrent networking library called Eventlet (http://eventlet.net)
  - Eventlet is a concurrent networking library that uses coroutines for parallelism
    - Approach is similar to cooperative multitasking
    - No preemption of processes
  - What does that mean?
    - Make tasks that can take a long time to process preemptible
    - How?
      - Allow the scheduler to select another task by giving back the control from time to time
      - Normal function call:

        `self.fancy_method(A,B,C)`
      - Function call that allows the scheduler to switch tasks:

        `hub.spawn(self.fancy_method,A,B,C)`

Running Ryu

- Use Linux
  - Use either the existing VM, which already has Ryu installed
  - Install it on a new OS by running `pip install ryu`

- But my Main Computer Runs Windows!
  - Ryu can made run on Windows, but does not work properly
  - Therefore: run Linux inside a VM
    - Connect the Linux VM to the outer network using „Bridged networking"
    - For the duration of the work, disable DHCP on the network interface of your Windows Computer