IK2221 – Network function virtualization - Phase 1

The target of the course's assignment is to involve students in modern networking design and implementation using Network Functions Virtualization (NFV) principles in an architecture that can potentially support load balancing on inferencing servers for Large Language Models (LLMs).

Goals

The goal of the assignment is to give you hands-on experience in practical NFV implementations. You should learn how to:

- Emulate network infrastructure,
- Generate traffic patterns using well-known tools,
- Launch a centralized POX controller,
- Instruct the data plane devices using NFV techniques,
- Capture the state of any device in the network,
- Capture traffic to inspect the message exchanges,
- Implement advanced network functions (i.e., load balancer, NAPT, and IDS).

NEW 2025: the final project evaluation will be individual, and we will assess each member contribution during the project discussions. In general, every member should be able to identify personal contributions, and be familiar with the whole project, ready to explain any part of it if asked.
Every team member should be present during the project discussions.

Team Formation

The assignment requires teams of 4 students to be formed.

Make your team as soon as possible.

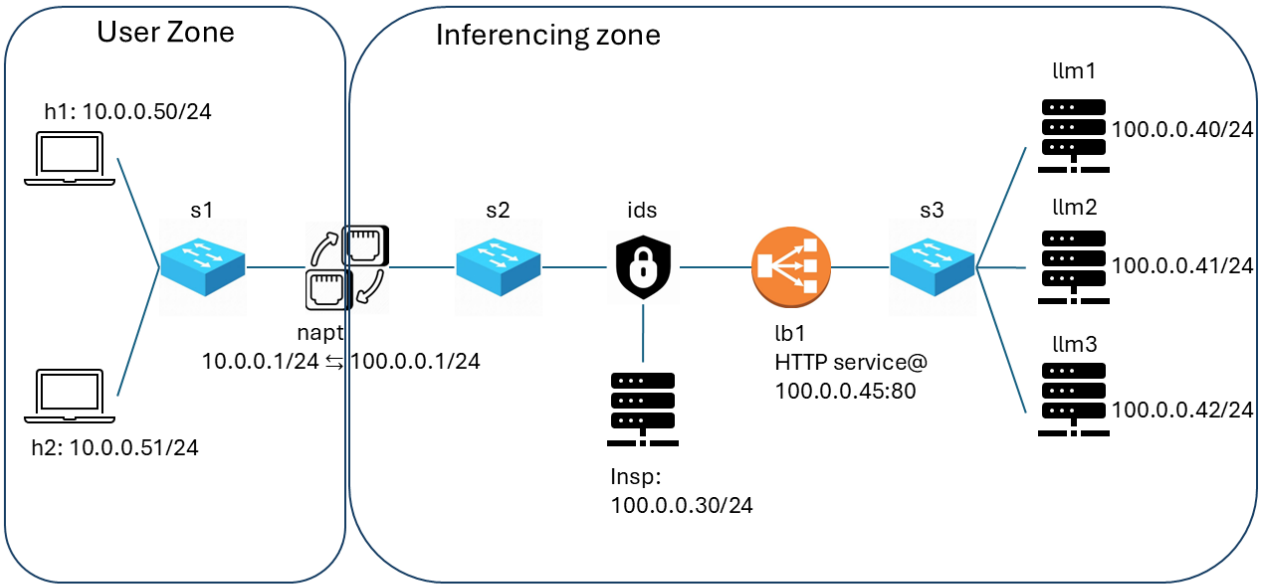

Figure 1: Mininet topology

Description

In this project, you will use Mininet to implement a small topology shown in Figure 1. The goal is to start playing with Mininet, a system to simulate a network.

Network Topology Overview

This topology consists of two primary zones:

- **User Zone (UZ)** – Provides user access to the network and facilitates communication with the inferencing infrastructure.
- **Inferencing Zone (IZ)** – Manages AI inferencing services, security, and load balancing to ensure efficient processing.

User Zone (UZ) Components

- **Hosts (h1, h2):** Two end-user devices that generate requests and interact with the inferencing services.
- **Switch (sw1):** A Layer 2 switch that interconnects the hosts (h1, h2) and serves as a gateway to the network.
- **Network Address and Port Translator (NAPT):** Translates private IP addresses from h1 and h2 into public IPs, enabling external communication.

Inferencing Zone (IZ) Components

- **Core Switch (sw2):** The main interconnect for the inferencing infrastructure, handling traffic from the User Zone (UZ) and internal components.

- **Intrusion Detection System (IDS):** Inspects incoming packets for suspicious patterns. If detected, packets are redirected to the inspector server (insp) for further analysis; otherwise, traffic proceeds normally.
- **Load Balancer (lb1):** Distributes incoming requests across inferencing servers in a round-robin manner to ensure optimal resource usage.
- **Switch (sw3):** Connects the load balancer (lb1) to the inferencing servers.
- **LLM Inferencing Cluster:** Comprising llm1, llm2, and llm3, this cluster processes requests related to large language model (LLM) inference, providing AI-driven responses.

At this point, be sure to have watched the tutorial about Mininet and POX!

In your code, you **should** strictly follow the IP configuration and naming conventions of Figure 1 as well as the following guidelines. You can change the already provided topology, but you must adhere to the following naming convention: To create the first switch (e.g. sw1) you should call `sw1 = addSwitch('sw1',..)` method of Mininet's API. The object to store the outcome of the command (i.e., sw1 in this case) should always have the name assigned in the figure. The first arguments of `addSwitch()` are important to be sequential (i.e. `sw1, sw2, …, swN`) so as to generate switches with human-friendly datapath ids (DPID). Using this convention, `sw1` (`sw1` in the figure) gets `dpid=1`, `sw2` (`sw2` in the figure) gets `dpid=2`, etc. Then it is easy to separate the devices in your code and call the appropriate functions for each one.

The topology should comply with the following rules:

1. The user zone sits behind a NAPT, hence the IP addresses of h1 and h2 are not visible to the outside world. The NAPT must apply the Source NAPT function to the outbound traffic (from user zone) and the Destination NAPT function to the inbound traffic (towards user zone).
2. The inferencing zone should provide these virtual services:
    a. HTTP/TCP service will be accessible on IP address 100.0.0.45/24 and port 80. This IP does not exist in Figure 1 because it is a virtual Service IP. Load balancer lb1 is responsible to handle this virtual IP. When lb1 receives a packet with destination 100.0.0.45 it must redirect the packet to one of the three inference servers. This means that destination IP address 100.0.0.45 should be translated into the IP address of the selected server and then forwarded. In the opposite direction, when lb1 receives a packet from one of the three servers, it must change the source IP address to 100.0.0.45 (virtual Service's IP) such that the host does not understand the existence of the servers behind this service. This functionality also implies the existence of NAPT, besides load balancing. In addition, the existence of IDS before lb1 puts some restrictions on the content of the HTTP requests. Specifically, the IDS module must access the payload of incoming requests and search for patterns that imply (a) code injection and (b) dangerous

use of HTTP methods. If such a pattern is found, the packet will be redirected to the inspector (server insp) for further inspection. *More details are provided below*.

b. Users from the user zone must be able to (successfully) ping the virtual IP of the inference server (i.e., the load balancers must generate ICMP responses).
c. The rest of the traffic must be blocked.

3. The inspector server is a passive node of this topology and does not require any services or communication with other nodes. The link between IDS and the inspector must simply be always on in order for the IDS to push the “suspicious” packets there.

Implementation Details

To realize the above requirements, you should instruct the data-plane nodes to forward traffic accordingly. To facilitate the implementation of the project requirements, we have provided a project skeleton which you should download and use as a base for your project. The skeleton includes a basic implementation of the controller and Mininet configuration. It is important to note that the skeleton should be kept asis and **you should follow the provided structure** when implementing the project requirements.

Switches sw1-sw3 are regular OpenFlow v1.0 [3] L2 learning switches. They are implemented as L2 learning modules of POX with OpenVSwitch as a constructor class.

The Mininet tutorial is showing you how to implement a POX L2 learning switch

The hosts and servers of this topology only need to have an assigned IP address, mask, and default gateway to reach the network. For the interface servers, performing actual LLM inference would be too costly. Instead, you only need to run a lightweight Python-based web server (e.g. CGIHTTPServer, BaseHTTPServer, SimpleHTTPServer) with 3-5 test pages (same for all servers) at the appropriate folder. You can type `‘python3 -m http.server 80’` to start an instance very easily. In the skeleton code, the ip addresses are set so that the topology can work for pings and simple forwarder modules. Remember that you have to change such ip addresses to the ones in Figure 1 in order to satisfy the requirements to pass the first phase of the project.

The load balancer, ids, and napt must be implemented in Click [6]. This means that you should instruct Click to capture packets from the interfaces of the Mininet switches `lb1`, `ids`, and `napt`, respectively. A pre-implemented version of such blocks is in the skeleton code. POX will only get registration events from these Mininet nodes (when Mininet boots). Once POX gets such an event, it simply starts the Click module responsible for this node as coded in the baseController.py script. You need to replace the pre-impemented versions (which simply forward packets on the correct interface) with your logic.

To start Click modules in POX, you can simply use subprocess.Popen() to run the proper click module. Check the skeleton code for some examples

1. In the following paragraphs, we explain the Click functions:

   Load Balancer (lb1): First, the load balancer must read packets from the interface and classify the packets into four basic classes. ARP requests, APR replies, IP packets, other packets. Upon an ARP request (that targets the virtual IP of the corresponding service), an ARP reply must be generated using an `ARPResponder` element per interface. This reply must contain the MAC address of the virtual service. Of course, this MAC address will be virtual as well [1], but the hosts should be able to generate IP packets after getting back an ARP reply from the load balancer. ARP responses must be sent to `ARPQuerier` elements (one per interface). IP packets must be sent to a pipeline of elements that will realize the load balancing procedure explained above. There are two directions for the load balancer, one towards the servers and another towards the clients. Hence two pipelines are required as we explain below. Finally, packets that are neither ARP nor IP must be discarded.

   IP pipeline towards the servers: A classifier/filter must be applied to packets coming from the external interface, which have destination IPs different from the virtual IP. After this classifier, a modification element (i.e., `IPRewriter`) should work in tandem with `RoundRobinIPMapper` in order to write the correct destination addresses (of one server) to the incoming packet.

   IP pipeline towards the clients: The `IPRewriter` element above should also translate the source address of the servers into the virtual addresses that the load balancer possesses. That way, the client cannot notice the existence of this “proxy” node.

   Any other traffic (not ping to LB and web traffic) is not required to work through the Load Balancer. *You can safely ignore the cases where the web server instantiates connections to other external hosts.*

4. IDS (ids) captures packets from the interfaces of the Mininet switch ids. No IP address is assigned to this module hence it should be acting as a forwarder of the traffic (a.k.a. sniffer). Specifically, ARP frames, ICMP ping requests, and responses, as well *as TCPsignaling* must traverse the IDS transparently[2].

   There are two kinds of analysis you must perform, both on HTTP packets. You must classify IP packets using `IPClassifier` to only match HTTP traffic.

   a) The first pattern we want to inspect is the HTTP method of each request. You have to check which HTTP method is used by the user. A Classifier element must be

[1] These MAC are not associated to any NIC created in Mininet, and you can pick any random MAC. Hint: choose something that easily recognizable during debug.

[2] *Don’t over-think it.* Our attack surface is plain HTTP requests (e.g. from curl) with specific payloads and methods. TCP responses from the servers should pass without any filtering.

used to classify all the possible traffic patterns that might pass through the IDS and search for patterns in the HTTP packets. Specifically, HTTP provides the following methods.

- GET and POST are the most common methods used to request a web page. GET passes any parameters via the URL while POST parameters are sent in the HTTP payload. This makes POST safer.
- HEAD method is similar to GET, but the server returns only the headers as a response.
- OPTIONS method asks the server which methods are supported in the web server. This provides a means for an attacker to determine which methods can be used for attacks.
- TRACE method allows the client to see how its request looks when it finally makes it to the server. An attacker can use this information to see if any changes are made to the request by firewalls, proxies, gateways, or other applications.
- PUT method is used to upload resources to the server. This method can be exploited to upload malicious content.
- DELETE method is used to remove resources from the server. Similarly, it can be exploited to delete useful content.
- CONNECT method can be used to create an HTTP tunnel for requests. If the attacker knows the resource, he can use this method to connect through a proxy and gain access to unrestricted resources.

Your IDS module must allow only allow POST and PUT methods. This means that you should identify the exact location of the HTTP method field in the header space and use the Classifier to detect the hexadecimal values of the method asked by the user. Only if the method is POST or PUT the packet is forwarded to lb1; otherwise, the packet is sent to the inspector.

b) The second interesting pattern is Linux and SQL code injection via the HTTP PUT method. The very first bytes of the payload after the HTTP header must be matched against the the following keywords:
   i. cat /etc/passwd
   ii. cat /var/log/
   iii. INSERT iv. UPDATE v. DELETE

5. The `Search` element will be useful to advance the "packet payload" pointer to the first byte of the HTTP payload. Check the HTTP header format to find a way to pass the header and "jump" directly to the payload. Do not use `Search` to look for the keyword themselves, it would be very inefficient because you only want to look at the very first byte of the payload while search will check for the pattern starting at any bytes of the packet. Use Classifier and the hexadecimal values of those patterns instead.
6. The NAPT will also be implemented as a Click module that captures packets from the interfaces of the Mininet switch napt. This module must handle ARP properly and apply address and port translation on TCP and ICMP packets. For TCP, you should use the `IPRewriter` element, while for ICMP an `ICMPPingRewriter` element can be used to

translate only ICMP echo requests and responses (you can safely omit other ICMP packets). Both traffic directions must end up at these translators which will convert 10.0.0/24 addresses into 100.0.0/24 and vice versa. The IP address of the inference zone interface of the NAPT is 100.0.0.1 and the IP address of its user zone interface is 10.0.0.1 as shown in Figure 1. If you get destination unreachable errors, remember that you may need to set the default route with, for instance, “h1 ip route add default via 10.0.0.1”.

7. Finally, the inspector server will be a normal Mininet host that captures packets from its interface (e.g., using `tcpdump`) and dumps the packets to a PCAP file. This file will be used as proof to assess the correct behavior of the IDS module. The PCAP file does not need to be sent to us, but the system to capture packets must be put in place.

Testing

In the provided skeleton you can find a file named “topology_test.py” containing the basic structure for your test. It contains codes for creating your topology in mininet and starting the controller. You are allowed to add as many as functions you need to this file to execute your test scenarios.
Please note that we will test your project with our own test script. So, *it is important that you consider all scenarios in your implementation even though you don’t have a proper test for it*.

Besides the automatic tests that will stress your application, you should also deliver a set of files that assess the functionality of each of your Click-based network functions. To do so, you should use the `AverageCounter` and `Counter` elements in order to measure:

1. The number of packets read and written by each Click module as well as the observed packet rate (throughput). These counters must be placed right after each `FromDevice` and right before each `ToDevice` Click element. Use the `AverageCounter` element.
2. If your Click module classifies traffic, you must count the number of packets observed per traffic class. See the example of lb1 in Table 1. The number of dropped packets must be included since it is also a traffic class. Use the `Counter` element in this case.
3. After your automated tests above have stressed the entire application, you should tear down your POX module as well as all your Click modules. Once a Click module terminates, it can use the `DriverManager` element to print out the collected counters to a designated file. Each Click network function should generate a file named `<function ID>.report`. For example, `lb1.click` should generate a file lb1.report.

The tests must be automated, and should not tell the human what he or she should see, but actually verify it automatically! Python will be helpful for this.

Phase 1 - Deliverable

1) You should submit your Mininet topology implementation and click modules as separate applications in separate sub-folders. Be careful, we should be able to test your project easily,

hence, please avoid using fancy python packages and stick to ones that already exist on the provided virtual machine.

2) Beside the source code, you should also deliver a Makefile in the main folder that creates the topology using rule ‘topo’, start(s) the application using rule ‘app’ and clean(s) using rule ‘clean’. Moreover, you must also have a rule with the name ‘test’ that starts the two other rules and the tester script. Hence, "make test" will launch the topology, the controller, and your series of tests, ultimately generating a summary of the results of the tests.

Strictly follow the submission rules and naming scheme. We use some automatic testing system, if e.g. the Makefiles are in the wrong place, it will fail. We reserve the right to give a zero mark if your code needs too much manual fixing to enable testing. Don’t include “garbage” files in your submission!

3) Before you submit your assignment, make sure that you have a *phase_1_report* file with the redirected output (stdout and stderr) of the make test command (as specified above). Do not hesitate if you want to add lines of comments between tests to further explain the reported results. But you must generate the comments in your scripts, i.e. *phase_1_report* must be entirely automatically generated.

4) When you are ready to submit, strictly follow the structure below (which is the one in the skeleton code):

- Create a folder for all your files with name ik2221-assign-phase1-team<Number>
    - E.g., ik2221-assign-phase1-team1
- Create a file MEMBERS stating the name and email of each member of the team.
- Create a Makefile with the rules mentioned above
- Create a subdirectory ‘topology’
    - Put your topology file(s) into this directory
    - It should also clean the system by typing ‘make clean’
- Create a subdirectory ‘application’
    - Put your controller sources into directory controllers
    - It should also clean the system by typing ‘make clean’
- Create a subdirectory ‘nfv’
    - Put your Click modules here
- Create a subdirectory ‘results’
    - In this directory, you should include the tests that stress your entire application. ● Make a tarball of the folder ik2221-assign-phase1-team<Number>(.tar.gz)
- Upload it before: **Deadline: 2025-04-29 at 17:00. Check the definitive one on Canvas**.

Note that all of the assignments will be checked with a plagiarism tool.
You will face a large penalty if we detect any case!

Grading

Given that you have successfully submitted your assignment in time (before the deadline) and you have followed the restrictions that we give you above, then:

1. NFV applications' design and implementation (40/100)
    a. If your NFV applications are correct you get (30/100)
        i. Load balancer gets (10/100)
        ii. NAPT gets (10/100)
        iii. IDS gets (10/100)
    b. If your design is correct you get the rest (5/100)
        i. This includes the code for http servers and request generation
2. Tests get (5/100).

Phase 1 will account for 40% of the final assignment mark.

Course's Virtual Machine

To guarantee a standard and common environment for all students, you'll use a VM to design, run and test the assignments.
You can download the VM image from the course's Canvas.
In this case you just download the .ova file and double-click on it.

The login to the VM is **ik2221** as username and password. With the .ova installation, you should be able to directly ssh into the VM (once started) by typing in a terminal

```
ssh -p 2222 ik2221@localhost
```

Here's a quick guide to generate an SSH key and store it for ik2221@localhost on port 2222 in order to avoid inserting your password to connect to the VM through ssh:

- Generate the SSH key pair (press Enter to accept the default location): "ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa"
- Copy the public key id_rsa.pub to the server (using port 2222): "ssh-copy-id -p 2222 ik2221@localhost" . If ssh-copy-id is not available, manually append the key in your vm at .ssh/authorized_keys file

Now, you should be able to log in without a password!

To copy file from your machine to virtual machine you can simply scp: "scp -r -P 2222 /path/to/local/folder ik2220@localhost:/path/to/destination/"

How to start

Download the skeleton code and copy it to your home folder of the VM. In one terminal, run “make app” to start the controller. In another terminal run “make topo” to start mininet with the already implemented topology. From mininet, you should be able to ping any node from each other node (e.g., “h1 ping llm1 -c 1”). You can use wireshark to sniff the traffic on the interfaces, or simple tcpdump (e.g., run “tcpdump -i napt-eth2” to sniff on the second interface of the napt).

```
================== LB1 Report ==================
   Input Packet rate (pps): 0.262347351823
  Output Packet rate (pps): 0.89408388857

Total # of   input packets: 395
Total # of  output packets: 258

Total # of   ARP  requests: 13
Total # of   ARP responses: 4

Total # of service packets: 145
Total # of    ICMP packets: 4
Total # of dropped packets: 51
================================================
```

Table 1: Example format of counters reported by load balancer 1 (lb1.report).

Tools

Mininet CLI

To facilitate your testing, when you launch the topology, make sure to enable the Mininet CLI such that it appears right after the nodes are started. This CLI (example) is a Linux-based command line that talks directly to the deployed hosts (not switches). You can use any Linux command that you know since each host is essentially a Linux namespace.

Traffic generation

You can use traffic generation tools such as ping, iperf, netcat, wget, curl, etc. To highlight the functionality of your IDS module, you need to generate HTTP messages with the special patterns provided above. These messages will test whether your IDS captures all the necessary patterns. You can use Python and Scapy to create a test script that sends packets with all these different patterns. The illegal patterns must appear to the PCAP file of the inspector.

SSH with visual effects

To visualize packet capturing process or pop-up shells for different Mininet hosts, you need to supply additional parameters -X or -Y to your SSH command, to be able to use the display. See Xterm and Wireshark below.

Wireshark

Provided that you established an SSH session with the above parameters, you can also start Wireshark [5] by typing 'wireshark &'. This will give you a high-level view of the captured traffic. In order to catch and visualize OpenFlow messages, you need to sniff the loopback interface and choose the 'Decode OFP' option. This tool is very important to accomplish the subtasks below.

Xterm

If you want to split the command line for each host, you can type '<hostname> xterm &'. This command will pop-up a new bash shell window for the specified host.

tmux

You can also use tmux inside your ssh connection to split a single terminal window into multiple sessions where you can run commands simoultaneously.

OpenVSwitch

OpenVSwitch [4] is the module that implements a virtual switch on your computer. You can think about it as an enriched version of a Linux bridge. It supports OpenFlow and provides a broad API to create, update, delete, and monitor your data plane. To check the state of a particular Mininet switch, you can use ovs-vsctl, ovs-ofctl, ovs-dpctl, ovs-controller, etc. commands. For instance, to dump all the OpenFlow rules of switch sw1, type `ovs-ofctl show s1.` Check this for more information. Note that you can use this tool for debugging purposes only. You are **not allowed** to use these commands to send OpenFlow rules to the data-plane devices. Rules should be sent by your Python application written on top of POX.

Click documentation

You can see the documentation for all elements by using the command `man Element`. For instance, `man Search` will give the documentation for the *Search* element. Most element's documentation is available also online https://github.com/kohler/click/wiki

Click's packet stealing

By default, the FromDevice element will take a copy of the packet, meaning that the original packet will continue in the Linux networking stack. This means that you could see duplicate packets, that will lead to a complete mess. You must use the SNIFFER false parameter to destroy the original packet. See https://github.com/kohler/click/wiki/FromDevice.u for more details.

Common Asked Questions about Course Infrastructure

- *"I need to create the topology files .py on the VM preferably using an editor like Gedit. How can I transfer my local topology.py file via SSH to the VM?"*

  Answer:

  There are several ways to edit files remotely and to transfer files. For example, use one of the following three techniques:

  1. Edit files on the VM. VMs have popular terminal-based text editors such as Vim and Nano, and a GUI-based Gedit. You can also install your favorite one.

  2. You can use scp to copy files from your laptop/host to your home folder on the VM--if you have Windows, you can use WinSCP or Microsoft VS Code + SFTP extension. You can also use FileZilla to move files.

  3. However, perhaps the most comfortable way is to create an SSH mount point, so that you can directly acces the VM's file from your system. In that case, you can use any text editors that you are comfortable with and you can copy files via drag and drop.

  Example, run these commands on your laptop in terminal (given that you run Linux or Cygwin) and replace my user name "user" with your own user name and IP address of your VM:

  - sudo mkdir /tmp/ik2220
  - sudo chmod -R 777 /tmp/ik2220
  - sshfs -p 2222 user@localhost:/home/ik2220 /tmp/ik2220

  In short, we are mounting the remote folder to the local filesystem, such that you can locally edit the content directly by pointing your editor to the /tmp/ik2220 folder. You should

replace “-p 2222 user@localhost” with the command you use to connect to the VM, if any different (e.g. when using QEMU).

4. The “modern and upstream” way to solve the problem is to use VS Code with the ssh extension. Simply configure it to connect to your VM and you should be able to edit the files directly.

❖ *"We test our network function on the VM and we cannot use the 'xterm' command on windows, so we have only one terminal to test all the functions. How can we open another new terminal?"*

Answer:

1. You can use Xming to use xterm, see here:

https://laptops.eng.uci.edu/engineering-software/using-linux/how-to-configure-puttyhttps://laptops.eng.uci.edu/engineering-software/using-linux/how-to-configure-putty-xming-on-your-laptopxming-on-your-laptop

2. You can split your terminal using tmux and ssh to the various hosts using logically split windows.

https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/

3. You can also open multiple terminals and ssh to different Mininet hosts, see here:

http://mininet.org/walkthrough/#ssh-daemon-per-host

References

1. Mininet network emulator: http://mininet.org/
2. POX Wiki as PDF: http://intronetworks.cs.luc.edu/auxiliary_files/mininet/poxwiki.pdf
3. POX Wiki online: https://noxrepo.github.io/pox-doc/html/
4. OpenFlow: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spechttps://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdfv1.0.0.pdf
5. OpenVSwitch: https://github.com/openvswitch/ovs
6. Wireshark: https://www.wireshark.org/
7. POX Manual https://noxrepo.github.io/pox-doc/html/
8. https://github.com/tbarbette/fastclick/wiki/Tutorial
9. https://www.bo-yang.net/2015/01/07/click-notes-click-language
10. Some documentation on IPRewriter https://github.com/tbarbette/fastclick/wiki/IPRewriter
11. An example with IPRewriter: https://github.com/tbarbette/fastclick/blob/main/conf/nat/mazu-nat.click
12. Strip TCPHeader:: https://github.com/tbarbette/fastclick/wiki/StripTCPHeader

13. You can use https://hpd.gasmi.net to decode packets as printed by Print(). You may want to set MAXLENGTH -1 to print the whole packet's payload.
14. You can use the elements wiki at https://github.com/tbarbette/fastclick/wiki/Elements for your convenience. Most elements are the same as in the "plain" Click and it is more upto-date w.r.t. the original one. The documentation reachable via man is built directly from the source code of your installation so if something is not working, check the man one.