# IK2220 VT18 - SDN & NFV Assignment

The target of the course's assignment is to involve students to modern networking design and implementation using Software Defined Networking (SDN) and Network Functions Virtualization (NFV) principles.

## High-level Network Design

Students are requested to implement the Cloud provider's topology depicted in Figure 1.

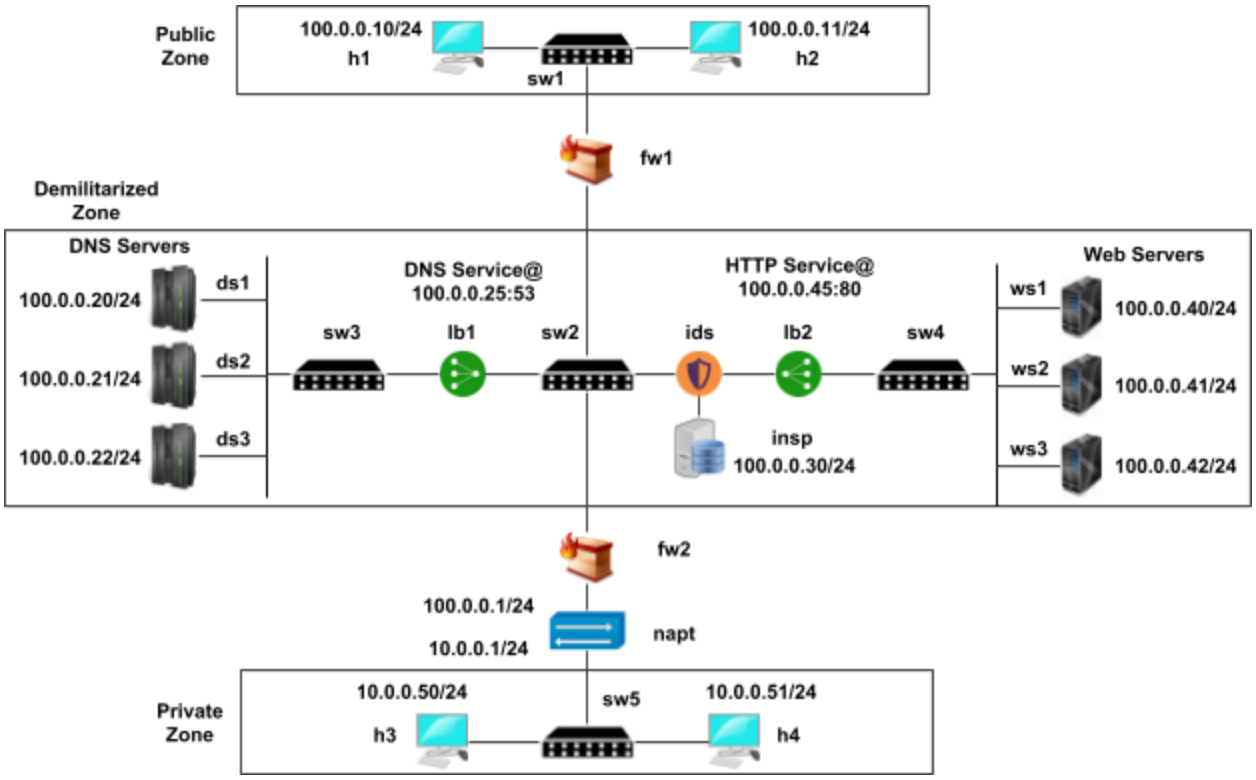


*Figure 1: Cloud Topology with three main zones. A public zone(PbZ) that sits close to the Internet, a protected, demilitarized zone (DmZ) that contains servers (DNS, Web) and a private zone (PrZ) that contains cloud resources (e.g. VMs).*

This topology consists of:

1. A public zone (PbZ) with two hosts and one L2 switch (sw1) that interconnects the entire network to the Internet.
2. A firewall (fw1) that controls the access to the demilitarized zone (DmZ).
3. A DmZ that contains two clusters of servers interconnected with one core switch (sw2). The left hand side servers (ds1, ds2, and ds3) formulate a cluster that provides Domain Name System (DNS) services. The IP of the DNS service is 100.0.0.25 and it is virtual (does not correspond to a physical network element). A load balancer (lb1) sits between the core switch (sw2) and the three DNS servers ensuring that incoming requests, which

target the virtual IP, will be modified accordingly with the destination IP address of a server in a round-robin fashion. Switch sw3 is used to connect the load balancer with the DNS cluster.

The right hand side servers (ws1, ws2, and ws3) formulate a cluster that provides Hypertext Transfer Protocol (HTTP) services. The IP of the Web service is 100.0.0.45 (virtual too). A load balancer (lb2) sits between the core switch (sw2) and the three Web servers ensuring that incoming requests, which target the virtual IP, will be modified accordingly with the destination IP address of a server in a round-robin fashion. Switch sw4 is used to connect the load balancer with the Web cluster. Before lb2, the incoming packets must be inspected by ids, an Intrusion Detection System (IDS) module. This module will search at the incoming packets' payload for certain "suspicious" patterns. If such a pattern is identified, the packet will be redirected to the inspector (insp) server for further processing. Otherwise, legal traffic will pass through lb2 as described afore.

4. A firewall (fw2) that controls the access to the private zone (PrZ).
5. A Network Address and Port Translator (NAPT) that translates the private IP addresses of hosts in PrZ into public IP addresses within the range of DmZ and PbZ.
6. A PrZ with two hosts and one L2 switch (sw5) that interconnects this zone with DmZ (through napt and fw2).

# Description

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

- emulate network infrastructure,
- generate traffic patterns using well-known tools,
- launch and program an SDN controller,
- instruct the dataplane devices using both SDN and NFV techniques,
- capture the state of any device in the network,
- capture traffic to inspect the message exchanges,
- implement more advanced network functions (i.e., firewall, load balancing, NAPT, IDS).

## Phase 1 (SDN only)

In the first phase, you will use SDN tools to implement a simplified version of the topology shown in Figure 1. This simplified topology is visualized in Figure 2. The key difference between the two topologies is that some of the network elements are simplified. Specifically, the load balancers lb1-lb2, the IDS node, and the NAPT should be replaced by simple L2 switches. The reason is because we want to implement the functionality of these network elements at the second phase of the assignment using NFV tools. In this first phase, there is no NFV, hence these elements must be replaced with SDN switches.

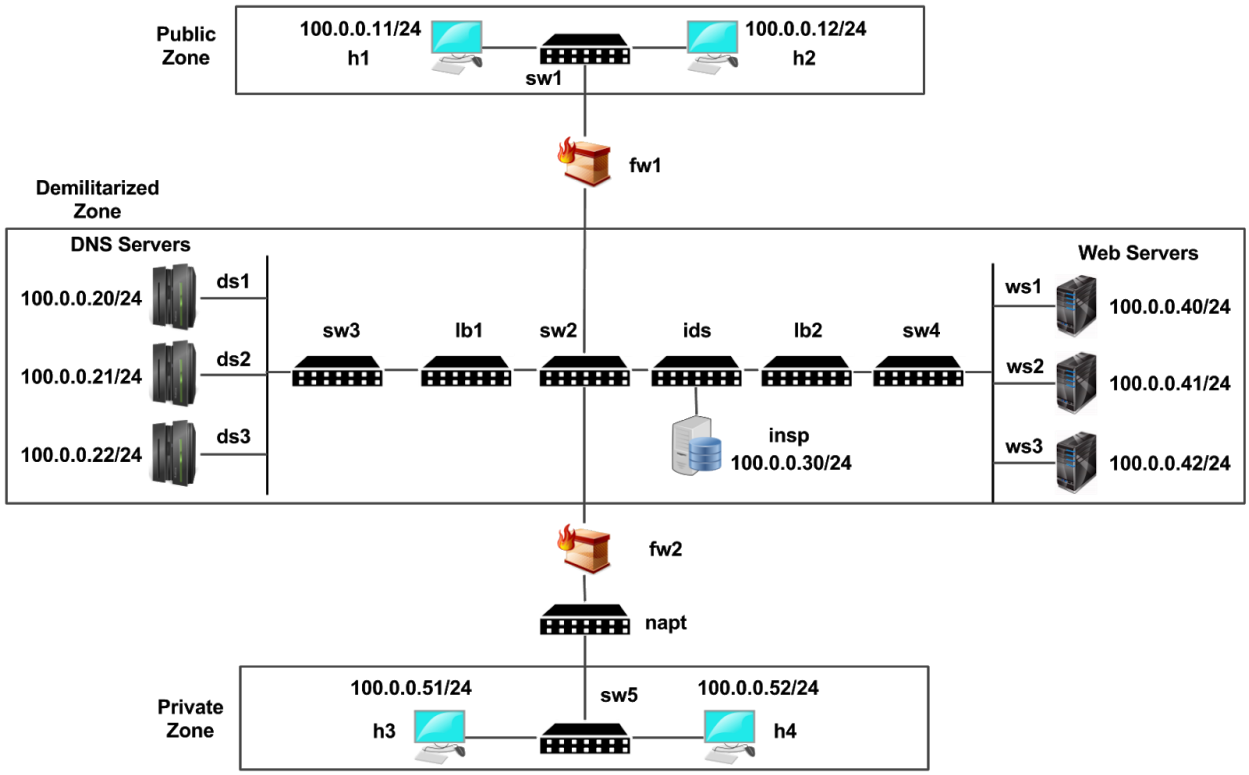


*Figure 2: A simplified version of the topology depicted in Figure 1. All the network elements are L2 switches and all hosts belong to the same subnet (100.0.0/24).*

Figure 3 shows more details about the way we have split the network elements of the target network in three main categories.

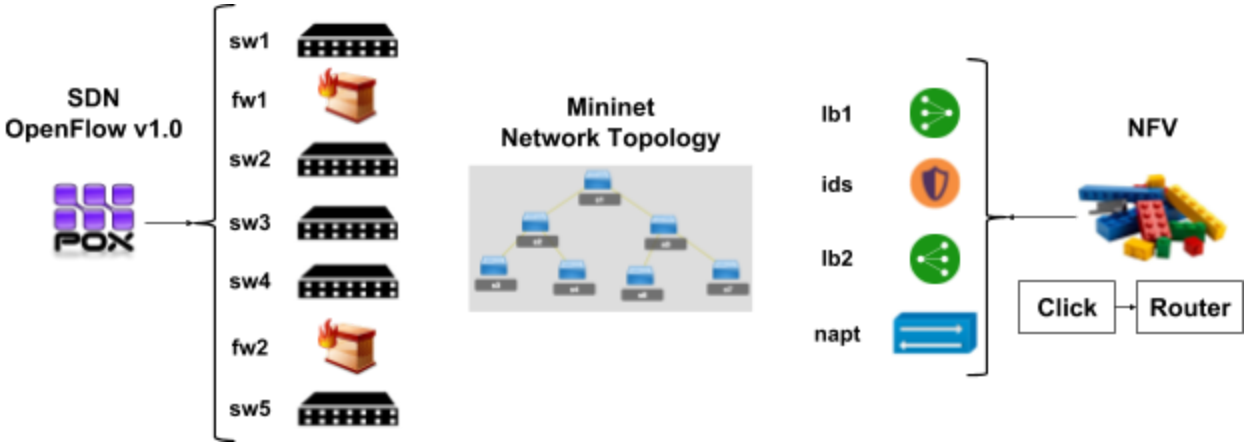


*Figure 3: Mapping of network functions to handlers. POX SDN controller and Click handle different functions in the network.*

First, the SDN network elements that are controlled by the POX SDN controller via the OpenFlow v1.0 protocol. These elements are switches sw1-sw5 and firewalls fw1-fw2. The second category of elements includes more advanced network functions that must be implemented using the Click NFV framework. The elements of this category are the load balancers lb1-lb2, the ids, and the napt. Finally, the remaining nodes of the topology are Mininet hosts. These nodes are not programmable and they do not belong to the network; they serve only as a means to perform testing by injecting and sinking traffic.

In this first phase, the NFV elements of the topology (i.e., lb1, lb2, ids, and napt) will act as SDN switches with a simple MAC learning functionality and L2 forwarding abilities. Additionally, the IP configuration of the nodes in PrZ is modified. All nodes belong to the same subnet because the devices that interconnect the nodes are L2 switches. To implement this topology use Mininet network emulator [1] and POX [2] SDN controller.
In your code, you should strictly follow the IP configuration and naming conventions of Figure 2 as well as the following guidelines. To create the first switch (e.g. sw1) you should call sw1 = addSwitch('s1',..) 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. s1, s2, …, sN) so as to generate switches with human-friendly datapath ids (DPID). Using this convention, s1 (sw1 in the figure) gets dpid=1, s2 (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 outcome of this phase is a L2 network where all hosts belong to the same subnet. However, because there are 2 firewalls in place (fw1 and fw2), not every host is allowed to communicate with every other host (pingall should not work for all pairs of users). Specifically, the 2 firewalls must comply to the following rules:

1. Any outbound traffic is allowed from the private zone (PrZ). Inbound traffic toward PrZ is allowed if and only if it is a response to already initiated outbound request. For example, when h3 pings h1, the ping response should reach h3. However, if h1 initiates a ping to either h3 or h4, it must fail. This rule is set to protect the private zone.
2. Users from PbZ and PrZ must be able to reach the services provided by the servers (by using the correct protocols and ports). For example to reach ws1, one must generate TCP packets toward 100.0.0.40 and destination port 80. Similarly, to reach ds1, one must generate packets with destination IP address 100.0.0.20 and destination port 53. ICMP pings are not allowed to reach the servers.
3. The rest of the traffic from PbZ and PrZ to DmZ must be blocked.

## Phase 1 - Implementation Details

To realize the above requirements, you should instruct the dataplane nodes to forward/drop traffic accordingly. The instruction will be done using SDN techniques. The following list describes each network element's functionality separately:

1. Switches sw1-sw5, lb1, lb2, ids, and napt are regular OpenFlow v1.0 [3] L2 learning switches. Use the L2 learning module of POX to instruct the appropriate rules.

2. Firewalls fw1 and fw2 are OpenFlow v1.0 L2 learning switches extended with stateful access control rules (sent by the SDN controller). These rules realize the functionality of DmZ and PrZ described above. Use the L2 learning module of POX as a basis to extend each firewall.
   You must design a basic firewall class that implements the common functionality of fw1 and fw2. Then, for each firewall, the extra functionality can be implemented in child classes. You have freedom to place firewall rules appropriately justifying your choices.
   **If you do not follow this programming approach and program each firewall separately, you will have a large penalty in the grade**. The intention here is to learn how to reuse software components across similar functions, thus learning how to design robust software.

In this phase it is compulsory to use OpenFlow v1.0 as POX does not support other OpenFlow versions. For the dataplane implementation, you should instrument Mininet to use OpenVSwitch as constructor class for the switches.
The hosts and servers of this topology are not SDN-enabled devices; they only need to have an assigned IP address, mask, and default gateway to reach the network. For the web servers, you also 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 'python -m CGIHTTPServer &' to start an instance very easily (Make sure to set the port to 80 because some servers use other default ports).
The DNS server can be implemented using Python and scapy. Use the example in [9] to customize your own Python script.

## Phase 1 - Testing

To verify that you have properly implemented phase 1, you should implement some tests. These tests should be packaged to a nice script (we suggest Python) with appropriate stdout/stderr messages that separate the tests and inform the tester about i) the success or failure of the test and ii) whether this success or failure complies to the requirements. Your tests should stress all the firewall rules and produce a report with a score. For example, one test can be: "Send different packets to server ds1". Out of these packets, only packets with destination IP 100.0.0.20 and destination port 53 must be allowed. Therefore, your test can generate e.g., 10 different packets, of which, only one meets this requirement. The server will respond to only one of these packets (the one allowed by fw1), hence your script will capture all the responses and calculate a success rate for this test. Then you move to the second test, where you test e.g., "Send different packets to server ds2", etc.
All the tests should be reported in a single file named as phase_1_report.

## Phase 1 - Deliverable

*1) Two of the main SDN principles are control and dataplane separation as well as code-reuse*. You should submit your mininet topology implementation and SDN application as two separate applications in separate sub-folders. Be careful, your VM is not connected to the Internet, hence it might not support all your fancy pythonic packages, just stick to the existing ones.

2) Besides the source code, you should also deliver Makefile(s) that create(s) the topology using rule ‘topo’, start(s) the SDN application using rule ‘app’ and clean(s) using rule ‘clean’. Moreover, you must also have a rule with name ‘test’ that starts the traffic tester above.

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). You can comment in this file if you want to further explain the reported results but it is preferable to enclose these comments in the script.

4) When your are ready to submit, strictly follow the structure below:

- Create a folder for all your files with name ik2220-assign-phase1-team<Number>
    - E.g., ik2220-assign-phase1-team1
- Create a file MEMBERS stating the name and e-mail of each member of the team.
- Create a subdirectory ‘topology’
    - Put your topology file(s) into this directory
    - Include a Makefile that builds the topology by typing ‘make topo’
    - It should also clean the system by typing ‘make clean’
- Create a subdirectory ‘application’
    - Create a subdirectory ‘sdn’
    - Put your SDN sources into directory ‘sdn’
    - Include a Makefile that builds the application by typing ‘make app’
    - It should also clean the system by typing ‘make clean’
- Create a subdirectory ‘results’
    - In this directory you should include the tests that stress your entire application.
        - Include a Makefile that starts the tests and produces the phase_1_report
- Make a tarball of the folder ik2220-assign-phase1-team<Number>(.tar.gz)
- Upload it before:
    - **April 09 12:05 AM**.

## Phase 2 (SDN + NFV)

In the second phase we target to turn the functionality of some of the nodes in Figure 2 into the desired functionality depicted in Figure 1. These nodes are lb1, lb2, ids, and napt which are emulated mininet switches. As shown in Figure 3, you must use the Click NFV framework to realize more advanced functions on these elements. This can happen by instructing Click to read/write packets from/to the interfaces of a target mininet switch (e.g., lb1) using FromDevice and ToDevice Click elements respectively. In between these elements, you should then construct the Click pipeline that is required to realize a given network function (e.g., load balancing). Note also that in this phase, the nodes of PrZ must change IP configuration since the presence of NAPT will provide IP address translation from one subnet (10.0.0/24of PrZ) to another (100.0.0/24 of DmZ/PbZ).

The topology of phase 2 should comply to the following rules:

1. Private zone sits behind a NAPT, hence the IP addresses of h3 and h4 are not visible to the outside world (DmZ, PbZ). The NAPT must apply Source NAPT function to the outbound traffic (from PrZ) and Destination NAPT function to the inbound traffic (towards PrZ).
2. DmZ should provide virtual services to both PrZ and PbZ. Specifically:
    a. DNS/UDP service will be accessible on IP address 100.0.0.25/24 and port 53. 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 100.0.0.25 it must redirect the packet to one of the three DNS servers. This means that destination IP address 100.0.0.25 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 100.0.0.25 (virtual Service's IP) such that the host does not understand the existence of the servers behind this service. This functionality implies also the existence of NAPT, besides load balancing.
    b. HTTP/TCP service will be accessible on IP address 100.0.0.45/24 and port 80. The service setup is exactly the same as the DNS service above and lb2 is responsible for the virtual IP of this service. In addition, the existence of IDS before lb2 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.
    c. Users from PbZ and PrZ must be able to (successfully) ping the virtual IPs (i.e., the load balancers must generate ICMP responses). If a host pings the IP address of a physical server (e.g., 100.0.0.20), an ICMP error must be generated as a response.
    d. The rest of the traffic from PbZ and PrZ to DmZ must be blocked.
3. PbZ nodes should be accessible from anywhere.
4. 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 inspector must simply be always on in order for the IDS to push the "suspicious" packets there.

## Phase 2 - Implementation Details

To realize the above requirements, you should instruct all the dataplane nodes to forward/drop traffic accordingly. The instruction will be done using **both** SDN and NFV techniques. The following list describes each network element's functionality separately:

3. Switches sw1-sw5 are regular OpenFlow v1.0 [3] L2 learning switches. Use the L2 learning module of POX to instruct the appropriate rules. (PHASE 1)
4. Firewalls fw1 and fw2 are OpenFlow v1.0 L2 learning switches extended with stateful access control rules (sent by the SDN controller). These rules realize the functionality of DmZ and PrZ described above. Use L2 learning module of POX as a basis to extend each firewall. (PHASE 1)

5. The load balancers, 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, lb2, ids, and napt respectively. POX should not send any OpenFlow rules to these switches because their functionality will be programmed using the Click language. POX will only get registration events from these mininet nodes (when mininet boots). Once POX gets such an event, it should simply start the Click module responsible for this node. In the following paragraphs we explain the Click functions:
    a. Load Balancers (lb1 and lb2): First, each load balancer must read packets from each 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, but the hosts should be able to generate IP packets after getting back an APR 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 each 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 PbZ and 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. You must design the load balancer Click class such that you can use the same one for both lb1 and lb2 but with different arguments. If you do not follow this instruction, you will have a large penalty in the grade of the LB.

    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.
6. IDS (ids) captures packets from the interfaces of mininet switch ids. No IP address is assigned to this module hence it should be acting as a forwarder of the traffic. Specifically, ARP frames, ICMP ping requests and responses, as well as TCP-signaling must traverse the IDS transparently. A Classifier element must be used to classify all the possible traffic patterns that might pass through the IDS and search for patterns in the HTTP payload. 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. 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 any 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 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 lb2, otherwise the packet is sent to the inspector.

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 hexadecimal values of the following keywords:

a. cat /etc/passwd
b. cat /var/log/
c. INSERT
d. UPDATE
e. DELETE

7. The NAPT will also be implemented as a Click module that captures packets from the interfaces of mininet switch napt. This module must handle ARP properly and apply address and port translation on TCP, UDP, and ICMP packets. For TCP and UDP 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 DmZ interface of the NAPT is 100.0.0.1 and the IP address of its PrZ interface is 10.0.0.1 as shown in Figure 1.

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 a proof to assess the correct behavior of the IDS module.

## Phase 2 - Testing

In this phase you should extend your tests from phase 1 with more tests. For example, in phase 2 we have the concept of virtual IPs so one of your new tests must ping these IPs and parse the responses.

Besides the automatic tests that will stress your entire 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. Number of dropped packets must be included since it is also a traffic class. Use the Counter element in this case.

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 as <function ID>.report. For example, lb2.click should generate a file lb2.report.

```
=================== LB2 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 2 (lb2.report).*

## Phase 2 - Deliverable

1) Use the same directory structure as you had in phase 1. In the application folder, add a subfolder `nfv` where you need to place the Click implementation of the load balancer (remember, one Click file for both lb1 and lb2), the IDS, and the NAPT.

2) Your SDN application should now call the Click scripts above, hence you need some minimal modifications to your SDN controller.

3) When your are ready to submit strictly follow the structure below:

- Create a folder with name ik2220-assign-phase2-team<Number>
    - E.g., ik2220-assign-phase2-team1
- Create a file MEMBERS stating the name and e-mail of each member of the team.
- Use the very same ‘topology’ subdirectory as in phase 1
- Use the very same ‘application’ subdirectory as in phase 1
    - Create a subdirectory ‘nfv’
    - Put your NFV sources into directory ‘nfv’
- In the subdirectory ‘results’
    - include the tests that stress your entire application.
    - include 4 .counters files, each one corresponding to a Click-based network function (see above).
        - E.g., lb1.counters
- Make a tarball of the folder ik2220-assign-phase2-team<Number>(.tar.gz)
- Upload it before:
    - **May 14 12:05 AM**.

4) Your deliverable in this phase must contain the full functionality of the assignment (both SDN and NFV parts).

## 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. For the DNS testing you can write your own Python script that sends some predefined queries to your DNS server.

### 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 loopback interface and choose '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.

### OpenVSwitch

OpenVSwitch [4] is the module that implements a virtual switch in 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 dataplane. 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 dataplane devices. Rules should be sent by your Python application atop POX.

### Click at high-level

Click is a software platform for composing small network processing functions such as 'decrement TTL of IPv4 packet' or 'change destination IPv4 address' into more complex ones such as a router or a firewall. The Click components are known as *elements* while a composition of those elements is called *module*. Click is written in C++ but it provides a high-level domain-specific Click language [7] to compose existing elements together to implement advanced network functions. The elements in the Click repository are almost 400 and you can find them in [8]. In the next section, we provide you some details on how to start and test click with existing elements and modules as well as details on how to implement your own modules.

### Click paths

Location: /opt/ik2220/click/
Elements: /opt/ik2220/click/elements/
Existing configuration files: /opt/ik2220/click/conf/

Click is already installed for you in your VMs under the location above. The kernel module to start click can be launched as follows:

```
$ sudo /usr/local/sbin/click-install <path-to-click-configuration-file>
```

while the user space click can be launched as follows:

```
$ sudo /usr/local/bin/click <path-to-click-configuration-file>
```

### User-space Click

It is mandatory to use the user-space Click program. The command to start Click in userspace is already provided above. All the results of your module (e.g. when you print a packet) are written in the stdout/stderr. Most of click's elements are designed to support both user and kernel drivers so you do not have restrictions on the elements you can use. In this link, you can find all available Click elements.

### Careful Click Usage

If you instruct Click to get packets from your VM's primary interface (i.e., FromDevice(eth0)), it will grab the packets from NIC before being delivered to hosts which means that you must handle the packets properly otherwise you will lose connectivity from the VM.
Whoever does this is responsible for the consequences; in this assignment we will not use real interfaces as inputs/outputs to Click modules. You Click network functions must be interacting with the virtual interfaces of Mininet.

## Useful Tutorials

To incrementally involve yourself in the development of this assignment, you can follow the exercises below. These exercises will help you to build you code gradually and verify your understanding.

### Tutorial 1

Start building the topology in Figure 2 incrementally. First, create the two PbZ hosts (h1 and h2) and connect them to switch sw1. In your mininet source file, you should create a controller object using the RemoteController class of mininet. This means that your topology will be orchestrated by your custom SDN application. You have to specify the IP address (localhost if you work on the same machine) and port (6633 is the default) of the remote application in order to establish connections with all the dataplane devices. Then start the controller and let sw1 become a l2 learning switch. In this small topology, start wireshark in promiscuous mode in the background. Then, type 'h1 ping -c 4 h2' to the Mininet's CLI. Stop wireshark after ping is done. Use OVS command line to inspect sw1's OpenFlow table. Check the contents of the table. Also check the different packets that you captured (i.e. based on protocol type) and their order. To see the OpenFlow messages, do not forget to capture loopback traffic.

### Tutorial 2

Secondly, attach the private zone to the above topology, including one firewall in between. After this, you should have nodes h1 and h2 attached to sw1, nodes h3 and h4 attached to sw3 and one firewall (fw1) between the two switches. Make all switches L2 learning devices and verify that pingall command succeeds for all pings. Then you have to implement the firewall rules to achieve private zone's isolation. With this in place, ping h3 from h1, then h1 from h3 and check the results. Also, start an iperf server to h4 (port 1025) and send data from h2. Then do the other way around (h2 is the server at the same port) and check the results.

## Team Formation

The assignment requires teams of 4 students to be formed.

## Course's Infrastructure

For the needs of this course, students can use our dedicated infrastructure to design, run and test the assignments. Each team will be assigned with one VM that contains all the required tools to perform the assignment. This VM will be available until the end of the course. You will receive an email with guidelines to access the VM. The machine is totally isolated from the Internet, you will run everything locally. For this reason, you have to use our jump server first to access the cluster; from there you can internally jump to your VM.
After you login to the VM, you may find all the useful stuff under /opt/ik2220/. You have to execute the following command to bring the ownership of the folder to you:

```
sudo chown your_username:sudo -R /opt/ik2220/
```

*You are responsible to keep your VM alive and tidy during the whole lifetime of the course so do not do any nasty things! We intentionally give you root access to the VM to have the full control. Back-up your work regularly and be careful!*

*For those who wish to use their own machines, there is an image of the same VM under /opt/ik2220/ folder on the jump server. You can SCP the VM and install it to your VirtualBox. The command to get the VM is:*

```
scp your_username@192.16.125.240:/opt/ik2220/ik2220.qcow ./
```

## Technical Details

In this section we will guide on the two directions of the assignment, namely the topology and SDN/NFV application.

Topology

You need to create exactly one Python file with the class of your topology. This class should extend Mininet's 'Topo' class. In the constructor you can design your topology calling addSwitch, addHost and addLink methods appropriately. Then, you need to create an instance of this class that will be given as argument to the Mininet constructor together with the switch type you want to use (we recommend OVSSwitch), the controller (you should use RemoteController) and other arguments that you consider as important. The Mininet constructor will return an object that represents your network (e.g., net = Mininet(topo=myTopo, switch=OVSSwitch, controller=RemoteController(..),..)).
Using this net object you can configure any node of the topology (net.get('h1').cmd('your command')), start/stop your topology (e.g. net.start()/stop()) or call the Mininet CLI (CLI(net)).
In order to perform the required tests you need to write several Python methods (e.g. one per test) and call them after net.start() and before net.stop().

Make sure that you always clean your mininet environment every time you stop an execution by calling mn -c.

SDN/NFV Application

The application that realizes the functionality of Figure 1 consists of two types of source files. First, you need to implement a Python source file (controller.py) that will implement the SDN controller of your assignment.

One way to make your Python class observable by POX is to:

- Create a new folder (e.g. ik2220) under /opt/ik2220/pox/ext/
- Copy your Python file of the SDN application there
- Compile it (python -m compileall <application path>)
- Start POX: $POX_PATH/pox.py $APP_PATH.<main.py> $APP_ARGS
    - Where POX_PATH=/opt/ik2220/pox
    - APP_PATH=ik2220 (The folder in pox/ext/)
    - APP_ARGS can have any optional arguments you give to main.py
- You can write the above steps in a script that automatically creates (only once) a folder under pox/ext, copies your compiled (.pyc) files there and executes pox with your main file as argument.

The controller.py file should follow the requirements of slide 9 and 10. In this file you can register the class (e.g. Controller) that you want to catch all the OpenFlow events. Class Controller should follow the structure of slide 12 (except for the launch method that is already included). This class should be able to catch switch connections and disconnections (the ConnectionDown handler is not included in the example). When a switch connects, the controller should be able to recognise it (using the appropriate DPID number) and assign to it one object (of a relevant class such as Firewall) that is going to handle all the upcoming events (e.g. PacketIn). For the basic switches of the topology you are allowed to use the existing LearningSwitch class of POX but bare in mind that you may need to add some extra rules on their forwarding tables in order to have your services forwarding properly established. For the two firewalls, as already suggested above, use one parent class with the shared firewall functionality and 2 derived classes that differentiate firewall 1 from firewall 2. All the classes and calls of the SDN application must be placed into the same file named controller.py. Use the slides and POX wiki as reference.

Some of the network elements of the topology must be programmed in Click. This means that your controller should be able to recognize the switches that need to be handled by Click. For example, if lb1 has DPID 5, then a switch connection message with this DPID should cause the controller to start the Click script that handles this node (essentially load balancer). Therefore, together with all your Python scripts (related to the SDN functionality), you need to deliver one Click script per network function (lb.click, napt.click, ids.click). As we explained above, the two load balancers must parameterize lb.click accordingly.

Testing

To prove the correctness of your application you should build automated tests that stress the behavior of the network and show whether your application complies with the rules described in Phase 2 above. The tests must generate the appropriate traffic patterns to prove the correctness of each rule. For example, to show that your PbZ hosts are accessible by any other node, create a test with name pbz-hosts and run the appropriate traffic (e.g., ping) to cover all possible cases.

For each test, print messages that intuitively show the outcome of the test. E.g., “h1 is accessible from all other hosts”, “NAPT translates request with source IP 100.0.0.10 into 10.0.0.1, NAPT translates response with source IP 10.0.0.50 into 100.0.0.1”, “Load Balancer 1 translates request 100.0.0.25:53 into 100.0.0.21:53”, etc. You can implement the tests in mininet but do not mix the topology functions with the testing ones.

Aggregate the description and output of each test to a final report file. One should be able to verify the correctness of your application by simply reading this report.

## Grading

In this section we provide you the way to assess your final grade. 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. [PHASE 1] Mininet topology design and implementation (5/100)
    a. If your topology is correct you get 3/100
    b. If your design is correct you get the rest 2/100
2. [PHASE 1] POX SDN application design and implementation (25/100)
    a. If your SDN application is correct you get (10/100)
        i. Firewall 1 gets 5/100
        ii. Firewall 2 gets 5/100
    b. If your design is correct you get the rest 15/100
3. [PHASE 2] NFV applications’ design and implementation (45/100)
    a. If your NFV applications are correct you get (30/100)
        i. Load balancer (one to be used by lb1 and lb2) gets (10/100)
        ii. NAPT gets 10/100
        iii. IDS gets (10/100)
    b. If your design is correct you get the rest 15/100
4. [PHASES 1+2] Tests get 25/100:
    a. [PHASE 1] 10/100 is the testing script for phase 1.
    b. [PHASE 2] 15/100 is the testing script for phase 2.

## References


1. Mininet network emulator: http://mininet.org/
2. POX Wiki: https://openflow.stanford.edu/display/ONL/POX+Wiki
3. OpenFlow: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
4. OpenVSwitch: https://github.com/openvswitch/ovs

5. Wireshark: https://www.wireshark.org/
6. Click modular router: https://github.com/kohler/click/wiki
7. Click language: https://github.com/kohler/click/wiki/Language
8. Click elements: https://github.com/kohler/click/wiki/Elements
9. DNS using Python scapy: https://thepacketgeek.com/scapy-p-09-scapy-and-dns/