IK2220 - SDN & NFV - Phase 1

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

Description

The goal of the assignment is to give you hands-on experience in 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 data plane devices using both SDN and NFV techniques,
- Capture the state of any device in the network,
- Capture traffic to inspect the message exchanges,
- Implement advanced network functions (i.e., firewall, load balancer, NAPT, and IDS).

This assignment will be split into two phases.

- Phase I (Mininet+SDN+Testing)
- Phase II (NFV+Testing)

Team Formation

The assignment requires teams of (up to) 4 students to be formed.

Make your team by 25th March.

Phase 1 (SDN only)

In the first phase, you will use SDN tools to implement a small topology shown in Figure 1. The goal of phase one is to start playing with Mininet, a system to simulate a network, and OpenFlow, an SDN protocol that allows the OpenFlow-enabled switches to communicate with an SDN controller, in our case, POX.

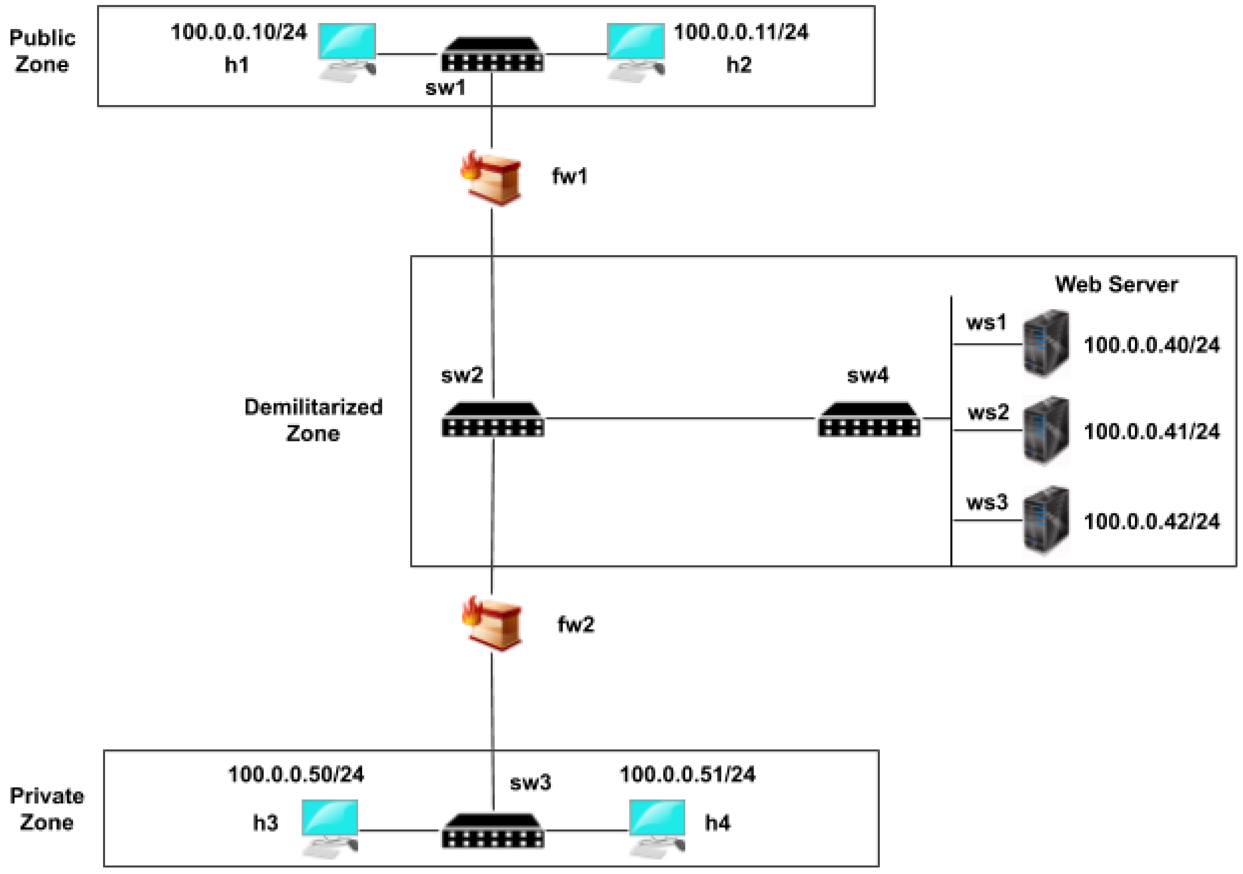


Figure 1: A simple Cloud Topology with three main zones. A public zone(PbZ) that sits close to the Internet, a protected, demilitarized zone (DmZ) that contains servers (Web) and a private zone (PrZ) that contains cloud resources (e.g. VMs). In phase 1, all the network elements are L2 switches and all hosts belong to the same subnet (100.0.0/24).

Topology

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 a cluster of web servers interconnected with one core switch (sw2). The right-hand side servers (ws1, ws2, and ws3) formulate a cluster that provides Hypertext Transfer Protocol (HTTP) services. Switch sw4 is used to connect the load balancer with the Web cluster.
4. A firewall (fw2) that controls the access to the private zone (PrZ).
5. A PrZ with two hosts and one L2 switch (sw3) that interconnects this zone with DmZ (through fw2).

The SDN network elements are controlled by the POX SDN controller via the OpenFlow v1.0 protocol. These elements are switches sw1-sw4 and firewalls fw1-fw2. The remaining nodes (h1-h4 and ws1-ws3) of the topology are Mininet hosts. These nodes are not programmable and they do not belong to the network; they serve only as a mean to perform testing by injecting and sinking traffic.

In this first phase, all nodes belong to the same subnet because the devices that interconnect the nodes are L2 switches. To implement this topology, we use the Mininet network emulator [1] and POX [2] SDN controller.

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. 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 outcome of this phase is to create an L2 network, where all hosts belong to the same subnet. However, since there are 2 firewalls in place (fw1 and fw2), not every host is allowed to communicate with every other host (i.e. pingall should not work for all pairs of users). Specifically, the 2 firewalls must comply with 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 requests. 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. 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 data-plane 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-sw4 are regular OpenFlow v1.0 [3] L2 learning switches. Use the L2 learning module of POX to instruct the appropriate rules.

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

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 the freedom to place firewall rules appropriately justifying your choices. **If you do not follow this programming approach and program each firewall separately but with a common base class, 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 data-plane implementation, you should instrument the Mininet to use OpenVSwitch as a 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 '`python3 -m http.server 80`' to start an instance very easily.

Phase 1 - Testing

To verify that you have properly implemented phase 1, you must implement some tests. These tests should be packaged to a nice script (e.g., you can use Python or embed the tests in a Mininet file) 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 ws1”. Out of these packets, only packets with destination IP 100.0.0.40 and destination port 80 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 ws2”, etc.
All the tests should be reported in a single file named *phase_1_report*.

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

Phase 1 - Deliverable

1) Two of the main SDN principles are control and data plane 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 python 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 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.

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:

- 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 email 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 17, 11:59 PM**.

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. Mininet topology design and implementation (5/40)
    a. If your topology is correct you get 3/40
    b. If your design is correct you get the rest 2/40
2. POX SDN application design and implementation (25/40)
    a. If your SDN application is correct you get (10/40)
        i. Firewall 1 gets 5/40
        ii. Firewall 2 gets 5/40
    b. If your design is correct you get the rest 15/40
3. Tests get 10/100:

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

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 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 log in 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 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:

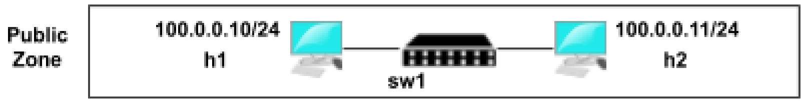
```
scp your_username@192.16.125.240:/opt/ik2220/ik2220.qcow ./
```

How to start

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

Tutorial 1

Start building the topology in Figure 1 incrementally using a Mininet python script. First, create the two PbZ hosts (h1 and h2) and connect them to switch sw1. Use the embedded switch of mininet for now (no POX). Verify h1 and h2 can ping each other.

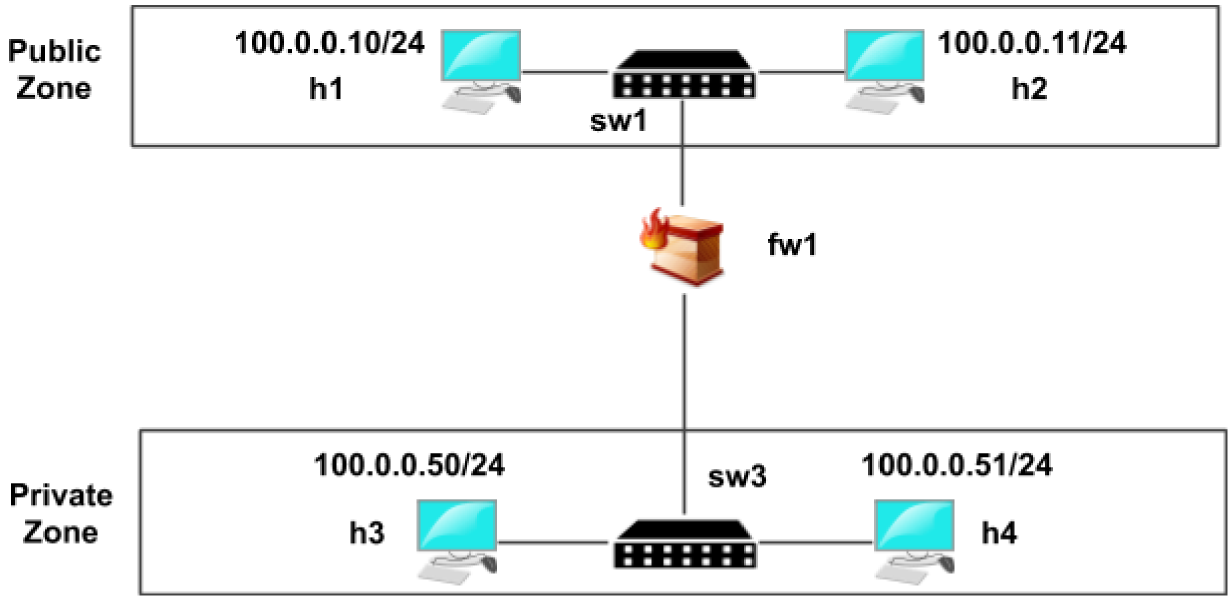


Then, change your mininet source file to create a controller object that will connect to a real POX SDN controller using the `RemoteController` class of mininet. This means that your topology will be orchestrated by your custom SDN application. You have to specify the IP address (localhost if you work on the same machine) and port (6633 is the default) of the remote application in order to establish connections with all the data-plane devices. Then, start the controller and let sw1 become an 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 the 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 the loopback traffic. This tutorial is essentially what is done in the mininet video.

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 the 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.

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.

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.

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/