University of Crete
Computer Science Department
Lecturer: Prof. Dr. X. Dimitropoulos
TAs: Dimitrios Gkounis, George Nomikos
M. Lakiotakis, G. Vardakis

Assignment 2

Contents

1. What this exercise is about

In this exercise you will use the Pyretic framework [1, 3, 6] in order to virtualize a single *physical* OpenFlow switch (emulated by mininet [2]). The use case is the following. We have two IP subnets connected over an OpenFlow switch: an internal network with some end-hosts (PCs) and one HTTP server, and the "Internet". This switch should *virtually* act as the following set of components:

- L2-learning switch for each of the two networks, respectively
- IP gateway for connecting the two networks, since they are different subnets
- stateful firewall for protecting the internal network from the outside. The firewall's state is based for simplicity only on IP address pairs (srcip, dstip) and allows only established connections that are active within some time duration.
- traffic checker and black-hole redirector. The switch should check flows that stem from the Internet and target the server's service, and in case the packet per second (pps) rate of the flows directed to the server exceeds a threshold rate, we consider this as an attempt to DoS the server and we redirect the flow to a black-hole host, which just absorbs the excess traffic. The concept is that this host can be used later on (out of the scope of this exercise) to also examine the incoming flows that the redirector forwards to it.

Your tasks are to:

1. implement the virtual fabric of the switch,
2. implement the black-hole checker and redirector component,
3. implement the whitelist for the firewall (but not the firewall itself), and
4. compose the network policies of the components in order to yield a fully working setup

All other modules (firewall, gateway forwarder, L2-learning switch) are provided to you.
The main assignment is described on page 7. Do not worry about the details at this point, since more information about the setup will be gradually explained as you go trough the exercise. **Attention: The assignment is accompanied by a zip file containing the code you will need to complete the exercise.**

2. Brief Introduction to Pyretic

Pyretic is a programmer-friendly, domain-specific language embedded in Python. Pyretic provides a runtime system that enables programmers to specify network policies at a high level of abstraction, compose them together in a variety of ways, and execute them on abstract network topologies. This time, you will not have to deal with low-level flow rule installation details like in the first exercise, but you will operate on a higher level of abstraction: you will have to think in terms of modular programming and network policy expression. As a learning curve, you will be taken through the steps of writing and running network applications i.e., hub and layer-2 MAC learning on Pyretic and testing them using Mininet. You will observe that you can easily write such simple applications and that complex network functionality (as described in the assignment on page 7) can be decomposed into simple modules to ease development and reasoning about this functionality. Currently, new runtime systems are being developed, which provide richer abstractions and tools to express your network applications — Pyretic is one such example offering a high-level northbound SDN API to the programmer/network operator; an implementation of the specification abstraction that you learned about in the lecture.

3. Initial Steps - Prerequisites

Before proceeding with the exercise, you are required to go through the following stages:

1. Visit and check out the official pyretic website:
   `http://frenetic-lang.org/pyretic/`. **ATTENTION: The exact names of the Pyretic policies and expressions that we use in the exercise are described later in section 4d. Please use the local versions for the final programming, since you may find inconsistencies with the official website due to version changes.**

2. Fire up your mininet VM and log-in using "mininet" as username and password. **At this point, you should already have the VM up and running from exercise 1. If not, please repeat the introductory steps of exercise 1.** The VM should already have two network interfaces (1 NAT eth1 and 1 host-only eth0 interface, besides the loopback one), as the OpenFlow tutorial VM. Make sure that these two interfaces are up, with appropriate IPs assigned to them. If they are down or no IPs are assigned, use the *dhclient* tool with sudo rights in order to fix this issue. The NAT interface allows you to connect to the Internet and download the required packages for the exercise, while the host-only one allows you to access the VM from your host system via SSH. Now leave the native console running, and open one SSH session with the VM's host-facing IP. Please enable X11 forwarding while doing so (-X flag in linux, and parameterization in *putty* in Windows, where an X11 server like XMing should also be running). In general, the process is very similar to what you did for the VM setup in exercise 1.

3. After establishing the SSH, X11-enabled connection with the pyretic VM, by default you should be now in the home folder. **Now run the script "ex2_install_pyretic.sh" that is included with the exercise assignment, after placing it in your home folder:**

   ```
   bash -x -e ex2_install_pyretic.sh
   ```

   In case something fails while running the script, please check the instructions on:
   `https://github.com/frenetic-lang/pyretic/blob/deprecated/INSTALL.md`. After the installation, please logout of your current session(s) and login again in order to make the new environment variables effective.

4. Since this exercise will require some server functionality (apache2 listening at port 80 for HTTP) to be present at the emulated virtual hosts, please carefully follow the necessary next steps:

   - `upload the source.list file from .zip (provided with the assignment) in your home mininet folder.`
   - `execute in your mininet command line:`
     `sudo mv sources.list /etc/apt/sources.list`
   - `Make sure that the script is the fresh one and that it is under root rights: ls -lt /etc/apt/sources.list`
     `You will see sth like:`
     `-rw-r--r-- 1 root root 3379 Oct 22 08:29 /etc/apt/sources.list`
     `(make sure that the date is fresh)`
   - `execute: sudo apt-get update`
   - `execute: sudo apt-get install -y apache2`

5. At this point, the environment required for the exercise should be ready. As explained before, by now you should have already logged out of all your sessions with the VM and

re-established them so that the environment variables are updated. **ATTENTION: You will not need to go through these steps again when you restart or reboot the VM**. At the next sections, you will learn step-by-step how to use Pyretic for building modular SDN applications in order to prepare for delving into the main assignment. You are advised to go through each step carefully in order to build up your knowledge on Pyretic and modular SDN programming.

4. Playing with Pyretic

(a) General

Pyretic is a new language and system that enables modular programming by:

- Defining composition operators and a library of policies for forwarding and querying traffic. Pyretic's parallel composition operator allows multiple policies to operate on the same set of packets, while its sequential composition operator allows one policy to process packets after another.
- Enabling each policy to operate on an abstract topology that implicitly constrains what the module can see and do.
- Providing a rich abstract packet model that allows programmers to extend packets with virtual fields that may be used to associate packets with high-level meta-data.

For more details on Pyretic, please see:
`http://www.frenetic-lang.org/pyretic/`.
We will be using the Pyretic runtime system, so please make sure that the default controller or the POX controller are not running in the background. Also, make sure that the port 6633 used to communicate with OpenFlow switches by the runtime is not bounded, by killing any corresponding controller process running on this port:

```
sudo fuser -k 6633/tcp
```

This will kill any existing process, using this port. You should also run "sudo mn -c" and restart Mininet to make sure that everything is clean and that you are using the faster kernel switch. From your Mininet SSH session window:

```
mininet> exit
sudo mn -c
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

The Pyretic runtime should already be installed and configured after the initial steps (the installation script has already been provided).

(b) Verify Hub behavior with tcpdump

Run the basic hub example:

```
pyretic.py -v high pyretic.modules.hub
```

This tells Pyretic to enable verbose logging and to start the hub component.
TIP: start pyretic first for quicker connection with mininet: You start the controller, and then start Mininet to immediately connect. Wait until the application indicates that the OpenFlow switch has connected.
Now verify that hosts can ping each other, and that all hosts see the exact same traffic - the behavior of a hub. To do this, we'll create xterms for each host and view the traffic in each. *In case you have any problems with xterms, you can always sniff the interfaces with wireshark. You can run wireshark as follows:*

```
sudo wireshark &
```

In the Mininet SSH session window, start up three xterms:

```
mininet> xterm h1 h2 h3
```

Beforehand, make sure that X11 forwarding is enabled in your client. Arrange each xterm so that they're all on the screen at once. Adjust the size of the terminals if needed. In the xterms for h2 and h3, run tcpdump, a utility to print the packets seen by a host:

```
tcpdump -n -i h2-eth0
```

and respectively:

```
tcpdump -n -i h3-eth0
```

In the xterm for h1, send a ping:

```
ping -c1 10.0.0.2
```

The ping packets are now going up to the controller, which then floods them out all interfaces except the sending one. You should see identical ARP and ICMP packets corresponding to the ping in both xterms running tcpdump. This is how a hub works; it sends all packets to every port on the network. Now, see what happens when a non-existent host doesn't reply. From h1 xterm:

```
ping -c1 10.0.0.5
```

You should see three unanswered ARP requests in the tcpdump xterms. When programming/testing, you can interpret three consecutive unanswered ARP requests as an indicator that you might be accidentally dropping packets. You can close the xterms now.
If you look at the hub code (in "~/pyretic/pyretic/modules/hub.py"), you can see the reduction in the amount of code needed to implement hub in Pyretic as compared to POX.

(c) Verify Learning Switch behaviour with tcpdump

This time, lets verify that hosts can ping each other when the controller is behaving like a layer-2 learning switch. Kill the Pyretic runtime by pressing Ctrl-C in the window running the program. Now, run the switch example:

```
pyretic.py -v high pyretic.modules.mac_learner
```

Like before, we'll create xterms for each host and view the traffic in each (*or use Wireshark as an alternative choice*). In the Mininet SSH session window, start up three xterms:

```
mininet> xterm h1 h2 h3
```

Arrange each xterm so that they're all on the screen at once. Adjust the size of the terminals if needed. In the xterms for h2 and h3, run tcpdump, a utility to print the packets seen by a host:

```
tcpdump -n -i h2-eth0
```

and respectively:

```
tcpdump -n -i h3-eth0
```

In the xterm for h1, send a ping:

```
ping -c1 10.0.0.2
```

Here, the switch examines each packet and learns the source-port mapping. Thereafter, the source MAC address will be associated with the port. If the destination of the packet is already associated with some port, the packet will be sent to the given port, else it will be flooded on all ports of the switch. You can close the xterms now. Have a look at the pyretic learning switch code (in "~/pyretic/pyretic/modules/mac_learner.py"), which is pretty self-explanatory. Also check the directory "~/pyretic/pyretic/examples/*" for further experimentation.

(d) Useful Pyretic policies and expressions

Here are some useful pyretic policies and expressions (**this is the main reference for Pyretic policies and expressions that are used for this exercise**):

```
match(f=v): filters only those packets whose header field f's value matches v
~A:         negates a predicate
A & B:      logical intersection of predicates A and B
A | B:      logical union of predicates A and B
fwd(a):     forward packet out port a
flood():    send all packets to all ports on a network minimum spanning tree,
            except for the input port
drop:       produces the empty set (no packet is output)
passthrough: produces the input packet (identity function)
A >> B:     A's output becomes B's input (sequential composition)
A + B:      A's output and B's output are combined in parallel (e.g., run a
            monitoring module in parallel with a routing module, without the need
            for sequential composition)
if_(p,A,B): if packet filtered by p, then use A, otherwise use B
```

ATTENTION: Use the:

'+'

operator for parallel composition of policies, and the:

'|'

operator for the union of matches.

(e) General information sources for Pyretic

These are the most useful information sources about Pyretic:

- Official webpage [1]
- Paper [6]
- Github repository [4] (reminder: we will work on the "deprecated" branch for this exercise)
- Wiki documentation [3]

Please also check the corresponding lecture material on Pyretic. **Keep in mind that the policies and expressions presented in section 4d of the exercise apply to your programming tasks.**

5. Main assignment: transform a physical OpenFlow switch into a working L2-learning switch, virtual gateway, blackhole redirector and stateful firewall using Pyretic

(a) Overview of the Setup

In all the following schematics, the full interconnection details are included (including ports, e.g., P_1, P_2, and so on). **Please pay attention to the interconnection between the components to avoid running into bugs when coding**.

i. Physical setup

The physical setup of this exercise is depicted in Fig. 1.

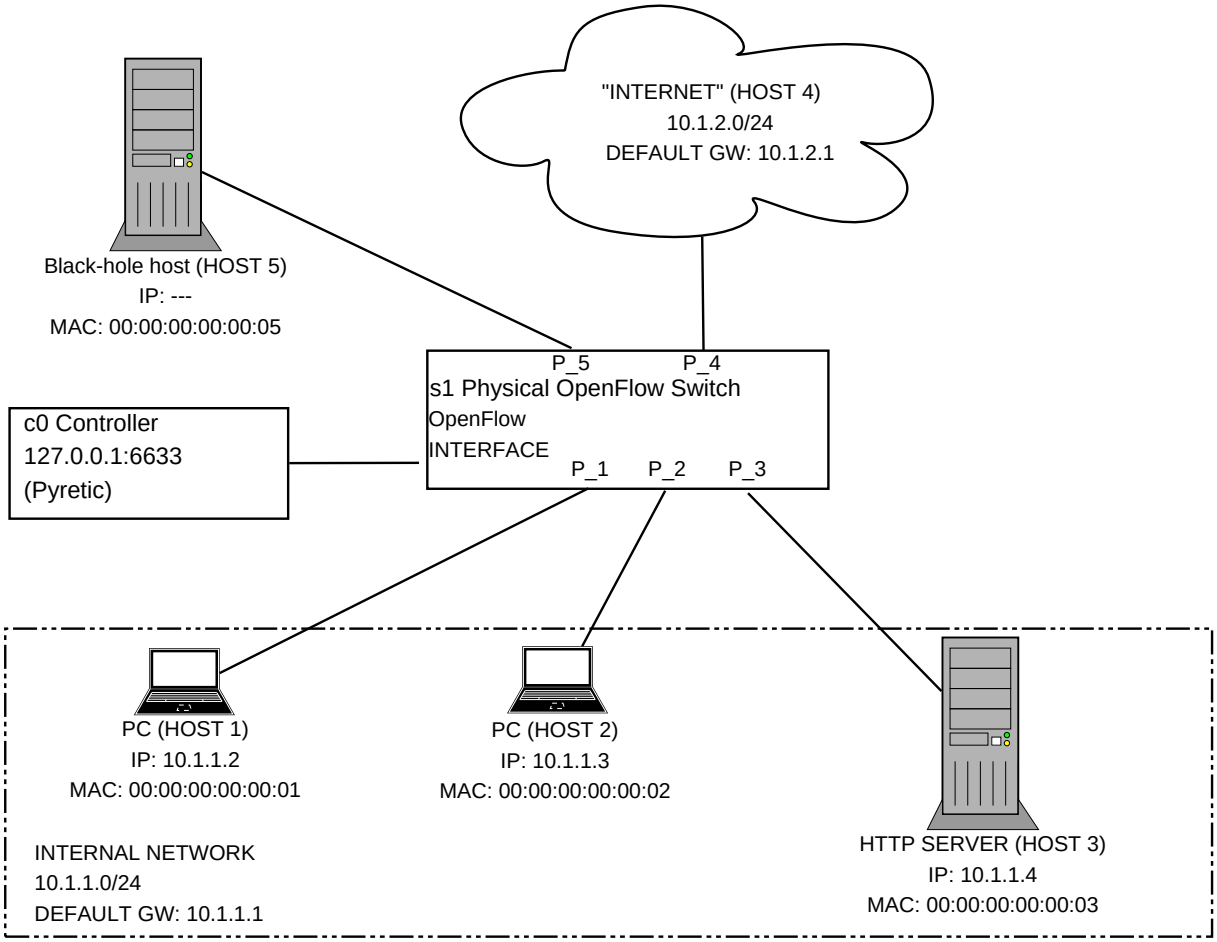

Figure 1: Physical setup of assignment 2

We have two networks:

- "10.1.1.0/24" is the internal network, which includes 2 end-hosts (PCs) and 1 HTTP server (listening http-port 80). These are all emulated as mininet hosts.
- "10.1.2.0/24" is the "Internet", emulated by one mininet host (using interface aliases). **For testing please use the IP of the primary interface which is "10.1.2.2".**

The IP of the default gateway for each network "10.1.X.0/24" is "10.1.X.1", where X=1,2. These two networks are connected via a physical OpenFlow switch, controlled by the Pyretic controller framework. The switch is also connected to a black-hole host, where it redirects flows from the Internet to the server (i.e. HTTP), in case these flows are suspicious of potential DoS

against the server services. Therefore, we want the OpenFlow switch to have the following roles simultaneously:

- L2-learning switch for each of the two networks
- IP gateway for connecting the two networks (since they are different subnets)
- stateful firewall for protecting the internal network from the outside. The firewall's state is based for simplicity only on IP address pairs (srcip, dstip). It allows the internal hosts and server to communicate with the Internet freely, but it allows connections from the Internet to the internal network only if they have been established from the inside, and for a short period of time (e.g., 3 seconds). As an exception, the Internet can communicate directly with the server, even without established connections from the inside. To elaborate:
  - internal hosts can ping the Internet. The Internet can ping them back only as long the connection has been initiated by the internal host and is *ongoing* (i.e., not idle for a specific period of time).
  - server can ping the Internet and the Internet can ping the server without restriction.
- DoS checker and black-hole redirector. The switch should check flows that stem from the Internet and target the HTTP server's services (port 80), and in case the rate of a 3-tuple (srcip, dstip, dsrport) *TCP* flow directed to the server exceeds a threshold rate, we consider this as an attempt to DoS the server and we redirect the flow to a black-hole host, which just absorbs the excess traffic and can be used to further examine the suspicious flows. The rate is measured in packets-per-second (pps).

ii. Virtual setup

Previously we described the required behavior of the physical OpenFlow switch. You can see that this behavior is relatively complex to reason about, if we consider the switch as a single component with multiple roles simultaneously. Now, let's break this physical component into multiple virtual components in the context of the modular approach of Pyretic. The result is a little more complex virtual setup, composed of much simpler components (instead of a physical setup composed of one very complex component). Each virtual device is tagged with a virtual identifier (1000, 1001,etc.). You can use Pyretic to "lift" traffic from the ingress physical network to a virtual switch fabric, apply the fabric policy you want (i.e., process the traffic) and then send it on its way by "lowering" it again to the egress physical network. The full setup of this exercise (combining the physical and virtual picture) is depicted in Fig. 2. You will notice that the virtual splitting of a physical switch into multiple sub-componentsis is the inverse example of the "Big Switch". There you abstracted away multiple physical switches into one single virtual switch, here you virtualize a single physical switch with multiple virtual components. The underlying virtualization principles that are used are similar.

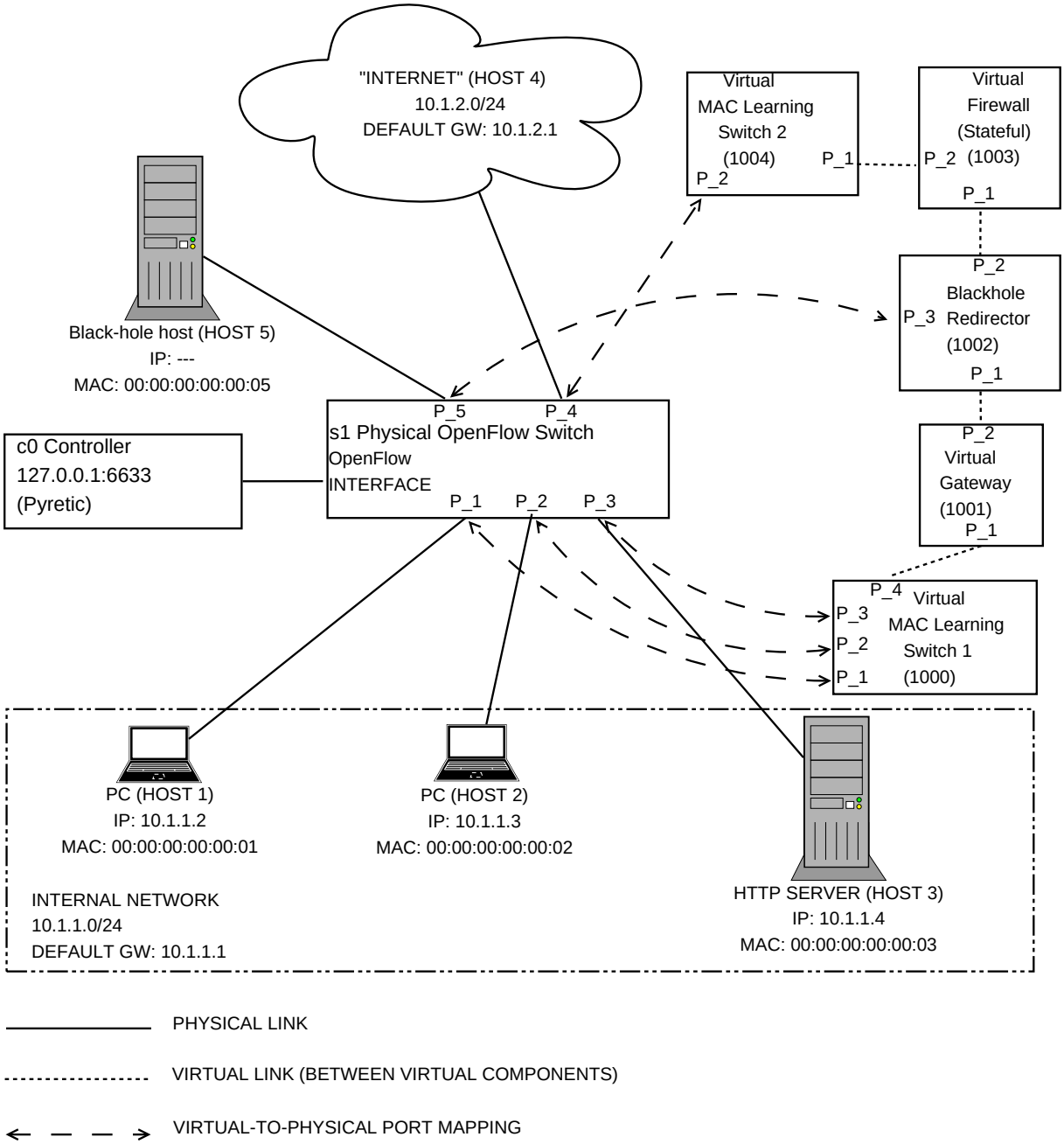

Figure 2: Full setup of assignment 2 (physical and virtual)

(b) Going through the required functionality step-by-step

Please perform the following steps:

1. Place the "ex2_mininet_physical_network.py" script, provided with the assignment, in your home folder. Study the script carefully. **No changes are required in this script.** Run it with:

   ```
   sudo python ex2_mininet_physical_network.py
   ```

   What is the purpose of this script?

   Hint: What did you do in exercise 1 in order to emulate physical networks?

2. Place the "ex2_split_gateway.py" script, provided with the assignment, under the "~/pyretic/pyretic/vdef" folder. Study the script carefully, but do not run anything yet. **Your first task is to fill in the missing parts of this script. See the commented parts marked with "ATTENTION"**. What is the purpose of the script?

   Hint: check the "split_gateway.py" script under the same folder, which comes pre-bundled with pyretic. This "split_gateway.py" script was created in order to support the use-case presented in the paper.

3. Place the following scripts, which are provided with the assignment, under the "~/pyretic/pyretic/examples" folder:

   - "ex2_dumb_forwarding.py"
   - "ex2_firewall.py"
   - "ex2_blackhole_check_red.py"
   - "ex2_pyretic_main.py"

4. Now study the scripts:

   - "ex2_dumb_forwarding.py"
   - "ex2_firewall.py"

   What is their functionality? **No changes are required in these scripts.**

5. Now study the scripts:

   - "ex2_pyretic_main.py"
   - "ex2_blackhole_check_red.py"

   As you see, these scripts are not complete and they require you to fill in parts of the code. **See the commented parts marked with "ATTENTION". Changes are required in both of these scripts. See later point 7 for clarifications**.

6. The "ex2_pyretic_main.py" script is the main script that bundles everything together and transforms the physical switch ito a multi-module entity that performs all the required functionality. This script is the one you should run as:

   ```
   pyretic.py pyretic.examples.ex2_pyretic_main
   ```

   Your second task is to fill in the missing parts of this script and verify its proper functionality. See the commented parts marked with "ATTENTION". Actually, you need to define the policies that need to be followed when traffic reaches each virtual component and combine these policies appropriately to

yield a working setup. Before testing the outcome of this script, please have the physical mininet network running, as shown previously (script "ex2_mininet_physical_network.py").

Hint: In the beginning, use just the dumb_forwarding policy for the blackhole redirector and/or the firewall. At this time, you can also verify that the virtual wiring you did with your ex2_split_gateway.py script is working as expected. At a second step, when you want to test the redirection and the firewall, change the policy appropriately, one module at a time.

7. The "ex2_blackhole_check_red.py" script implements the virtual module that redirects suspicious flows from the Internet to the black-hole host, according to the initial description of the setup. This module monitors the rate of new *TCP* flows coming from the Internet with format (srcIP, dstIP, dstPort) and if they exceed a certain threshold (in packets per sec - pps), it sends the traffic to a black-hole host for inspection instead of the normal path (to the gateway). All other traffic that does not satisfy the requirements for this redirection will pass directly though the virtual gateway.
Your third task is to fill in the missing parts of this script and verify its proper functionality. See the commented parts marked with "ATTENTION". Actually, the basic skeleton of the task is there, but you need to implement the missing functionality.

   Hint: Wireshark is a useful tool to verify which packets reach which interface, if they reach it at all. Telnet and ping are useful for creating traffic towards a host. Also, feel free to play with the threshold of detection: ***the one used in the current exercise is 5 pps.*** *Try smaller or larger thresholds and verify the results.*

General hints: Study all the provided scripts and the modules that are imported in the lines 8-14 of "ex2_pyretic_main.py" carefully before starting to play with the code. Scripts that are pre-bundled with pyretic placed under the "~/pyretic/pyretic/examples" folder will also help you get an idea of what the main script and the redirector should do. Use tools like wireshark, ping and telnet in order to test and debug your code. Comments on the provided code and debug messages will also help you understand better what you should do and how you should do it.

(c) Summary of Tasks

You tasks are the following:

1. Complete the code in "ex2_split_gateway.py" (switch virtualization script)
2. Complete the code in "ex2_pyretic_main.py" (main pyretic script).
3. Complete the code in "ex2_blackhole_check_red.py" (blackhole checker and redirector module script).

(d) How to run and test your code

You can verify that everything runs ok as follows.
First emulate the physical network (using the SSH session with mininet VM):

```
sudo python ex2_mininet_physical_network.py
```

Then run your pyretic code (using a different SSH session with mininet VM, while the previous one is alive):

```
pyretic.py pyretic.examples.ex2_pyretic_main
```

Afterwards, the described functionality should be up and running, if you have done everything correctly (please test before submitting). In case you want to ping/send traffic to the "Internet", please use the IP 10.1.2.2 as explained beforehand. Generally, use IP addresses as traffic destinations and not host aliases (due to the mininet default behavior, which can produce wrong host to IP mappings, even after you manually change the IP address of a host interface).
Example tests are the following:

```
h1 ping -c1 10.1.1.3 #if successful, it means that
                     #the learning switch for the
                     #internal LAN works fine
                     #you can also try with 10.1.1.4

h1 ping -c1 10.1.2.2 #if successful, it means that
                     #the internal LAN hosts can ping
                     #the Internet. Besides, it opens
                     #a 3-sec hole so that the Internet (h4)
                     #can ping them back, since it is considered
                     #an established connection from the inside

h4 ping -c1 10.1.1.2 #Internet to internal hosts: this should only
                     #work for established connections! (see previous comment)

h4 ping -c1 10.1.1.4 #Internet to HTTP server (ping): this should always work

h4 telnet 10.1.1.4 80 #Internet to HTTP server (service flow): this should only work
                      #as long the pps rate of the flow does not exceed
                      #a threshold (try rapidly typing characters and see what
                      #happens)
```

Good luck!

Acknowledgments

This assignment is based on material from a similar course at ETH Zurich [5]. The initial instructions for having Pyretic up and running and testing it have been adapted by the "Software Defined Networking" course archive on `www.coursera.org`.

References

[1] About Pyretic.
`http://frenetic-lang.org/pyretic/`.

[2] Mininet official website. `http://mininet.org/`.

[3] Pyretic documentation wiki.
`https://github.com/frenetic-lang/pyretic/wiki/documentation`.

[4] Pyretic github repository.
`https://github.com/frenetic-lang/pyretic/tree/deprecated`.

[5] ETH Zurich. Advanced Topics in Communications Networks HS 2014: Software-Defined Networking. `http://www.csg.ethz.ch/education/lectures/ATCN/hs2014`.

[6] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing Software-Defined Networks. In *USENIX NSDI*, 2013.