

SOFTWARE DEFINED NETWORKING

Introduction to Pyretic Controller

(PYTHON + FRENETIC = PYRETIC)

Presented By:

Manasa Kulkarni

manasankulkarni@gmail.com

Agenda

1. Introduction
2. Pyretic features
3. Installing Pyretic
4. Running Pyretic
5. Using Pyretic in Mininet
6. Generating TC and UDP Throughput graph
7. Plotting graph using GNUPLOT
8. Assignments
9. Learn More

1. Introduction

Pyretic is one member of the frenetic family of SDN programming languages. One example of a programming language that sits on top of such a north-bound API is **Frenetic**, which is an SQL-like query language. For example, Frenetic would allow a programmer to count the number of bytes, grouped by destination Mac address, and report the updates to these counters every 60 seconds.

Pyretic is an SDN language and run time that implements some of the composition operators. The language is a way of expressing these high level policies, and the run time provides the function of compiling these policies to the OpenFlow rules that eventually are installed on the switches.

Pyretic, a new programmer-friendly domain-specific language embedded in Python. Pyretic is open-source software that offers a BSD style license compatible with the needs of both commercial and research developers. It 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 topologies. As mentioned above, 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.
- Pyretic enables each policy to operate on an abstract topology that implicitly constrains what the module can see and do.
- Pyretic provides 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. Pyretic's core policy language, libraries, and runtime are available on the Pyretic homepage along with documentation, video tutorials. For more details on Pyretic, see <http://www.frenetic-lang.org/pyretic/>

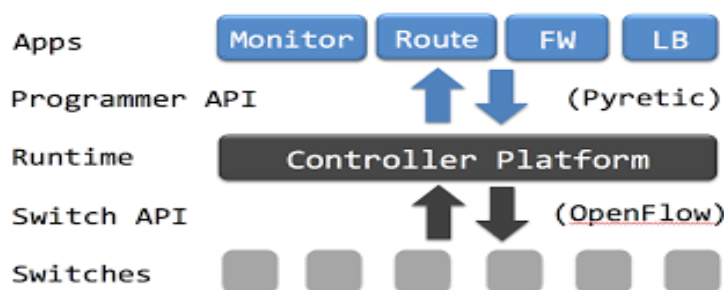


Figure 1 Pyretic language and Platform

2. Pyretic features

Pyretic offers several features.

- The first is the ability to take as input a packet and then return packets at different locations in the network. This effectively allows the implementation of network policy as a function that simply takes packets and returns other packets at different locations.
- The second feature of Pyretic is the notion of Boolean predicates. Unlike OpenFlow rules which do not permit the expression of simple conjunctions such as AND and OR, or negations like NOT, Pyretic allows the expressions of policies in terms of these predicates.
- Pyretic also provides the notion of virtual package header fields. This allows the programmer to refer to packet locations and also to tag packets so that specific functions can be applied at different portions of the program.
- Pyretic, policies are functions that map packets to other packets. Some example functions in Pyretic include the identify function, which returns the original packet; none or drop, which returns the empty set; match, which returns the identity if the field *f* matches the value *v* and returns none or drop otherwise; mod, which returns the same packet with the field *f* set to *v*; forward, which is simply syntactic sugar on mod to say that the output port field in the packet should be modified to the parameter specified; and flood, which returns one packet for each port on the network spanning tree. In OpenFlow, packets either match on a rule or they simply fall through to the next rule. So, OR, NOT, etc, can be tough to reason about.
- In contrast, Pyretic's match function outputs either the packet or nothing, depending on whether the predicate is satisfied. For example, we could apply a match statement that says match destination IP equals 10.0.0.3. and this function would take packets as input and only return packets that satisfy this predicate. In addition to the standard packet header fields, Pyretic offers the notion of virtual packet header fields, which is a unified way of representing packet metadata.
- In Pyretic, the packet is nothing more than a dictionary that maps a field name such as the destination IP address to a value. Now, these field names could correspond to fields in an actual packet header. But they can also be virtual. For example, we could provide a match statement based on a switch, indicating that we only want to return packets that are located at a particular switch or on the input port, indicating that we only want to see packets whose attributes match a particular input port. The match function matches on this packet meta-data and the mod function can modify this meta-data.

3. Installing Pyretic

The Controller Pyretic can be installed in two ways:

One is directly in your Ubuntu operating system and secondly we can download the OVA file directly from the link: <http://sdnhub.org/tutorials/sdn-tutorial-vm/>. Dependencies to Python need to be installed. The installation commands for python dependencies are as follows:

- `sudo apt-get install python-dev python-pip python-netaddr screen hping3 ml-lpt graphviz ruby1.9.1-dev libboost-dev libboost-test-dev libboost-program-options-dev libevent-dev automake libtool flex bison pkg-config g++ libssl-dev python-all python-all-dev python-all-dbg`
- `sudo pip install networkx bitarray netaddr ipaddr pytest ipdb sphinx pyparsing==1.5.7 yappi`

If your Ubuntu doesn't have the Pyretic installed then follow the commands below to install the Pyretic along with new updates.

[1] Patch asynchat Python dependency (addresses some, but not all, of the bugs in this library)

- `wget https://raw.githubusercontent.com/freneticlang/pyretic/master/pyretic/backend/patch/asynchat.py`
- `sudo mv asynchat.py /usr/lib/python2.7/`
- `sudo chown root:root /usr/lib/python2.7/asynchat.py`

[2] Install git subtree

- `git clone https://github.com/git/git.git`
- `pushd git/contrib/subtree/`
- `make`
- `mv git-subtree.sh git-subtree`
- `sudo install -m 755 git-subtree /usr/lib/git-core`
- `popd`
- `rm -rf git`

[3] Clone the Pyretic repository to your home directory

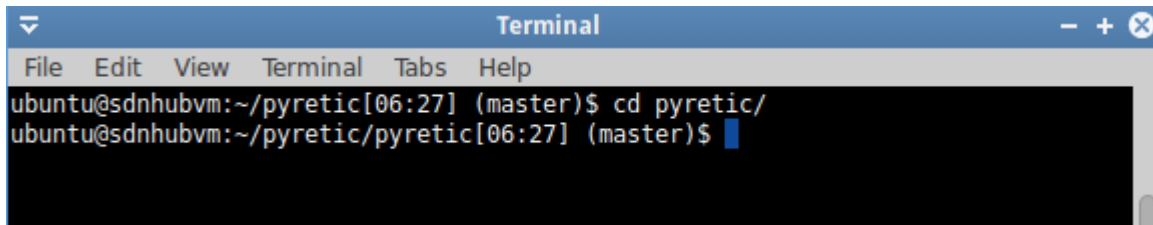
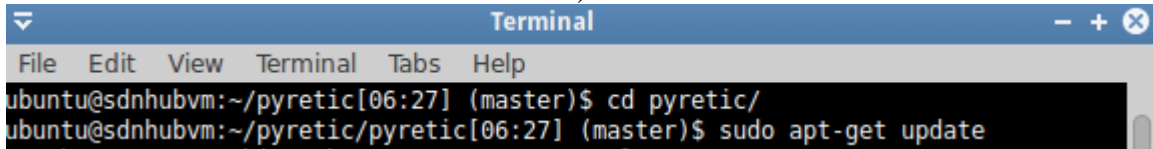
- `cd~`
- `git clone http://github.com/frenetic-lang/pyretic.git`

4. Running Pyretic

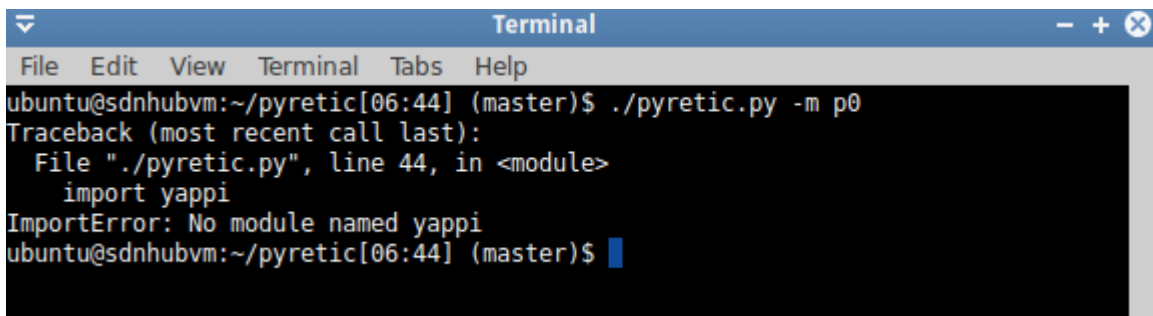
Step 1: Open a new terminal:

Ctrl + Alt + t

(Above command will open a new terminal)

Step 2: Go to Pyretic folder via terminal**cd pyretic/**A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is 'ubuntu@sdnhubvm:~/pyretic[06:27] (master)\$'. The command 'cd pyretic/' has been entered, and the prompt has changed to 'ubuntu@sdnhubvm:~/pyretic/pyretic[06:27] (master)\$'.**Figure 2** change directory path to pyretic**Step 3: Update and install few additional packages****sudo apt-get update**A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is 'ubuntu@sdnhubvm:~/pyretic[06:27] (master)\$'. The command 'cd pyretic/' has been entered, and the prompt has changed to 'ubuntu@sdnhubvm:~/pyretic/pyretic[06:27] (master)\$'. The command 'sudo apt-get update' has been entered.**Figure 3** Update all the packages**Step 4: Try to run the controller using below command****./pyretic.py -m p0**

(When you first run the Pyretic for the first time, you may encounter a problem like the following figure)

A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is 'ubuntu@sdnhubvm:~/pyretic[06:44] (master)\$'. The command './pyretic.py -m p0' has been entered. The output shows a traceback: 'Traceback (most recent call last): File "/.pyretic.py", line 44, in <module> import yappi ImportError: No module named yappi'. The prompt is now 'ubuntu@sdnhubvm:~/pyretic[06:44] (master)\$'.**Figure 4** yappi module needs to be installed**Step 5: Install the following command to solve the module error****sudo pip install yappi**

(Above command will install the yappi packages)

After the installation, we can run again and check the controller run. We get the following information on our screen. The p0 is the **proactive mode** where packets are pushed to switches based on Pyretic policy. Generally, the highest performant mode is made currently available.

```
ubuntu@sdnhubvm:~/pyretic[06:46] (master)$ ./pyretic.py -m p0
Module must be specified

Usage: pyretic.py [options]
(type pyretic.py -h for details)
```

Figure 5 Pyretc run in p0 mode

Step 6: Run the following example to run the controller in a proper manner.

`./pyretic.py -m p0 pyretic.modules.mac_learner`

```
ubuntu@sdnhubvm:~/pyretic[06:46] (master)$ ./pyretic.py -m p0 pyretic.modules.ma
c_learner
Couldn't import dot_parser, loading of dot files will not be possible.
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
```

Figure 6 connected to the pyretic frontend

5. Running Pyretic in Mininet

This section will help you to understand how to configure your Mininet tool to work with Pyretic controller instead of the default controller of Mininet.

Step 1: Open a new terminal.

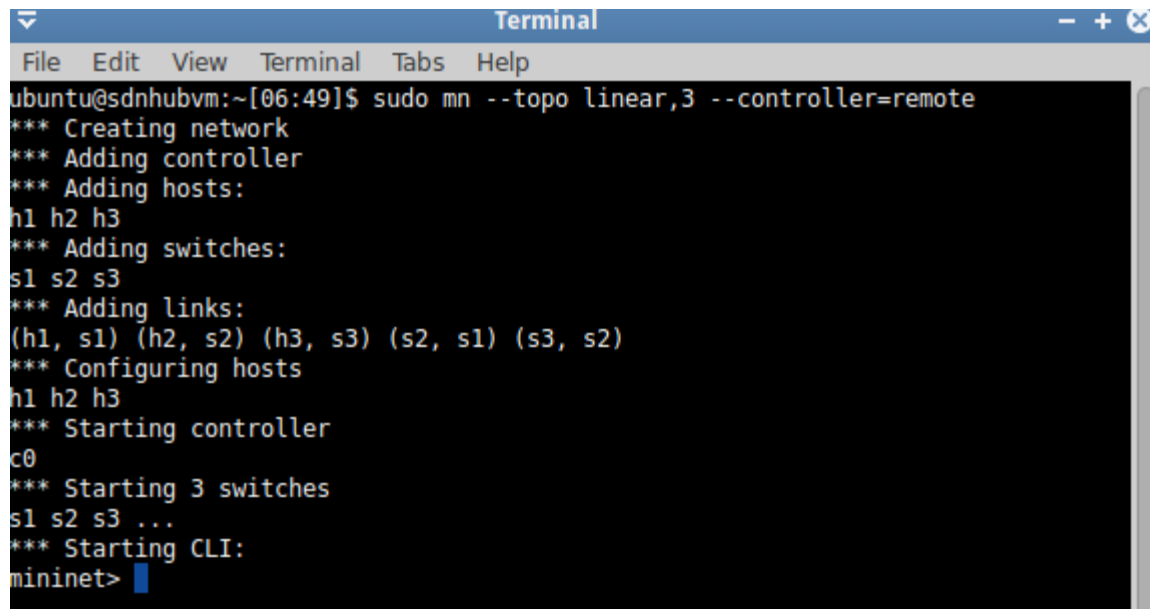
`Ctrl + Alt + t`

(above command will open a new terminal)

Step2: Type the topology command to run the mininet

`Sudo mn -topo linear,3 -controller=remote`

(The above command will create a linear topology remotely)



```

Terminal
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~[06:49]$ sudo mn --topo linear,3 --controller=remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>

```

Please do find out your port and ip address using the command **netstat -a** and further run the command in mininet as per below figure.

```

ubuntu@sdnhubvm:~[19:44]$ sudo mn --topo single,100 --controller=remote, ip=192.168.56.133, port=38916

```

Step 3: Ping the hosts to calculate the packet transmission using the command
h1 ping -c 3 h3

(above command will change directory path to floodlight)

```

mininet> h1 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=515 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.801 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.112 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.112/172.105/515.404/242.749 ms
mininet>

```

6. Generating the TCP and UDP throughput graph in Mininet

Once the pyretic with mininet is up and running, open the xterm terminal using the command **xterm h1 h2**. The two xterm terminals get opened and below are the steps to run the iperf at TCP and UDP protocol.

Step 1: Goto the h2 terminal of the xterm and type the command **iperf3 -s -p 5566 -w 5000 -i 1 > result**. Here -w is the window size which is used only for the TCP, -s is the server end, -p is

the port number of the iperf server, -i is the interval on which the server will measure the traffic and result is the output file where the data will be stored.

Step2: Next, goto the h1 terminal of the xterm and type the command `iperf3 -c 10.0.0.2 -p 5566 -w 5000 -t 150` where -c is the client end.

Step3: Goto the mininet terminal and type `h1 ping h2 > resultant.txt`. Here, this command stores the latency values into the text file later we can filter the data and check its correctness.

Step4: Once the data are being stored into the output file, the next step is to filter the data and plot the graph. The command to filter the values is `cat result | grep sec | head -150 | tr - ' ' | awk '{print $2,$5}' > new_result`. The data in the new_result file is ready for plotting the graph.

Step5: In order to calculate the jitter, bandwidth and data transfer, there is a need to run the iperf server in the UDP mode and store the data into the file. The command used to run at the h2 terminal which is the iperf server is `iperf 3-s -u -p 5566 -w 5000 -i 1 > result`.

Step6: Along with this command need to run another command in the iperf client i.e.; h1 terminal. The command is `iperf3 -c 10.0.0.2 -u -b 10M -p 5566 -w 5000 -t 150`. Here, -u is the UDP traffic, -b sets the maximum bandwidth, -p is port, -w is for setting buffer socket buffer size and -t is for setting the duration of test and in our case, it is 150 seconds.

7. Plotting graph (GNUPLOT) with graph script

Plotting the graph with the help of graph script and analysing the graph based on the Jitter, Throughput and Latency parameters is described.

Once we have the data values of latency, jitter, and throughput. Then plot the graph using GNUPLOT. Open a new command and type gnuplot. It will take us to the gnuplot terminal. There are number of parameters which are needed to be set for the graph like: xrange, yrange, xtics, ytics, xlabel, ylabel, style and much more.

Below is the figure of graph script which includes all the parameters to plot the GNU.


```
set terminal png
set output 'UDP_Resultant_1000_nodes.png'
set title "Topologies run over UDP protocol for 1000 nodes"
set xrange[0:150]
set xtics 10
set yrange[0:250]
set ytics 25
set xlabel "Time (in seconds)"
set ylabel "Throughput (in MBytes)"
set style data line
set grid
set key box
plot "Single_topology" with line, "Linear_topology" with line,
"Reversed_topology" with line, "Mesh_topology" with line,
"Tree_topology" with line, "Star_topology" with line
```

8. Assignments

Demonstrate the following topologies and obtain the topology designs.

1. Find all the shortest paths
2. Pyretic + POX
3. Fat Tree Topology
4. Routing using Dijkstra's algorithm

9. Learn More

<https://www.cs.princeton.edu/~jrex/papers/pyretic13.pdf>

<https://github.com/frenetic-lang/pyretic/wiki>

<https://slideplayer.com/slide/9274331/>