



Software Defined Networking

Dr. Nick Feamster
Professor

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

This Module: Programming SDNs

⦿ Lessons

- Motivation for Programming SDNs
- Programming Languages for SDNs
- **Composing SDN Control**
 - **Pyretic**
- Kinetic: Event-Driven SDN

⦿ Programming Assignment

⦿ Quiz

What is Pyretic?

- ◎ SDN Language and Runtime
 - **Language:** Way of expressing high-level policies
 - **Runtime:** Way of “compiling” those policies to OpenFlow rules
- ◎ Allows programmers to specify policies on “located packets” (packet + location)

Features

- ◎ **Network policy as function:** Take as input a packet, return packets at different locations
- ◎ **Boolean predicates:** In contrast to OpenFlow “exceptions”
- ◎ **Virtual packet header fields:** Can refer to locations, tags on packets, etc.
- ◎ **Parallel and sequential composition:** Compose policies

Network Policies

- ⦿ In OpenFlow, policies are bit patterns (tough to reason about)
- ⦿ In Pyretic, policies are functions that map packets to other packets

Syntax	Summary
<code>identity</code>	returns original packet
<code>none</code>	returns empty set
<code>match(f=v)</code>	<code>identity</code> if field <code>f</code> matches <code>v</code> , <code>none</code> otherwise
<code>mod(f=v)</code>	returns packet with field <code>f</code> set to <code>v</code>
<code>fwd(a)</code>	<code>mod(outport=a)</code>
<code>flood()</code>	returns one packet for each port on the network spanning tree

Boolean Predicates

- ⦿ In OpenFlow, packets either match on a rule, or they “fall through” to the next rule
 - Simple “or”, “not”, etc. is tough to reason about
- ⦿ Pyretic’s match function outputs the packet or nothing depending on the predicate

```
match(dstip=10.0.0.3) | match(dstip=10.0.0.4)
```

```
- -
```

```
-
```

```
-
```

```
-
```

Virtual Packet Header Fields

- ⦿ Unified way of representing packet metadata
- ⦿ Packet is a dictionary that maps a field name to a value
 - `match(inport=a)`
 - `match(switch=T)`
 - `match(dstmac=b)`
- ⦿ The `mod` function can also modify packet metadata

Policy Composition

- ◎ **Sequential composition:** Perform one operation, then the next (e.g., firewall then switch)

```
match(dstip=2.2.2.8) >> fwd(1)
```

- ◎ **Parallel composition:** Perform both operations simultaneously (e.g., counting, forwarding)

```
(match(dstip=2.2.2.8) >> fwd(1)) +  
(match(dstip=2.2.2.9) >> fwd(2))
```


Traffic Monitoring

- ⦿ Can create a query to see packet streams

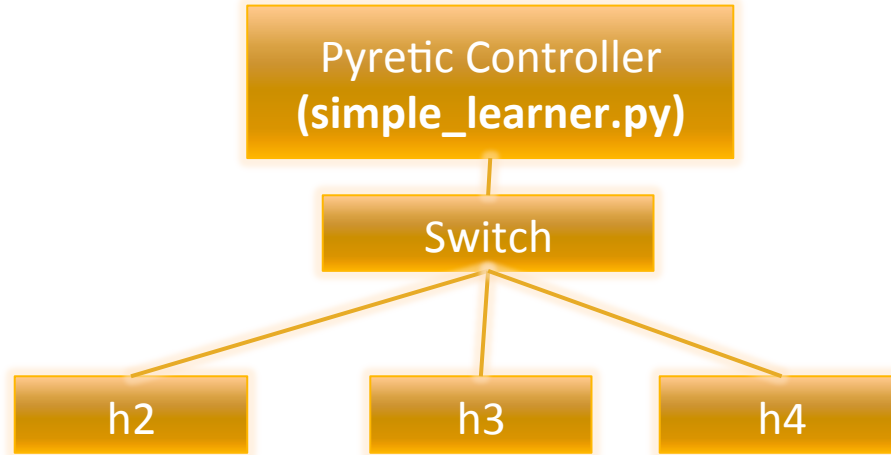
```
self.query = packets(1,['srcmac','switch'])  
self.query.register_callback(learn_new_MAC)
```

- ⦿ Callbacks are invoked to handle each new packet that arrives for the query

Dynamic Policies

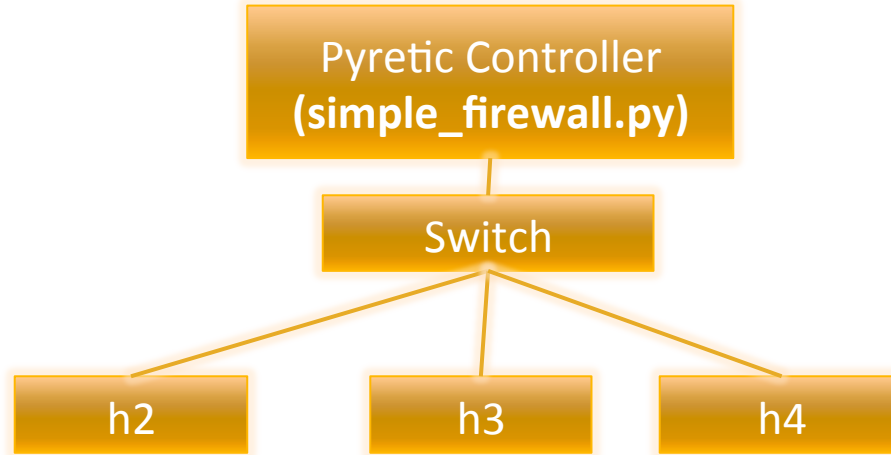
- ⦿ Policies whose forwarding behavior changes
- ⦿ Represented as timeseries of static policies
- ⦿ Current value is `self.policy`
- ⦿ Common idiom
 - Set a default policy
 - Register callback that updates policy
- ⦿ Two examples: switch, firewall

Example: Pyretic Switch



- ⦿ `$ sudo mn --topo single,3 --mac -arp`
- ⦿ Every first packet with new source MAC at the switch is read by a query
- ⦿ Policy is updated with new predicate

Example: Pyretic Firewall



- Create dynamic firewall policy
- Register a callback to check rules
- Sequentially compose with learning switch

Summary

- ◎ Pyretic makes writing complex policies easy
 - Network policy as function
 - Predicates on packets
 - Virtual packet headers
 - Policy composition
- ◎ Composition makes it easy for policies to build on one another
- ◎ Next: Events