Seize control of your network with Ryu

Ewen McNeill
<ewen@naos.co.nz>

Naos Ltd

2014-09-13 — Kiwi PyCon 2014

Outline

1. Introduction
   - Administrivia
   - What is Ryu?
2. Getting Started
   - Installing Ryu
   - Development environment
   - Ryu applications
3. Controlling your network
   - OpenFlow model
   - Example MiniNet network topology
   - Simple worked example
4. Summary

Administrivia

- About the speaker
  - Freelance consultant in Wellington through Naos Ltd
  - Works at intersection of Networking, Sysadmin and Development
  - Used Python for about 18 months (be gentle!)
- Questions Policy
  - If it is about the current slide, raise your hand.
  - Please ask more general questions at the end.
- Slides:

  `http://www.naos.co.nz/talks/seize-control-with-ryu/`

Ryu

Ryu is an OpenFlow Controller written in Python, which can be used to create a Software Defined Network

- `http://osrg.github.io/ryu/`
- Apache 2.0 license
- Originally a project of NTT Communications (Japan)
- “Ryu” (pronounced “ree-yooh”) is Japanese for “flow”

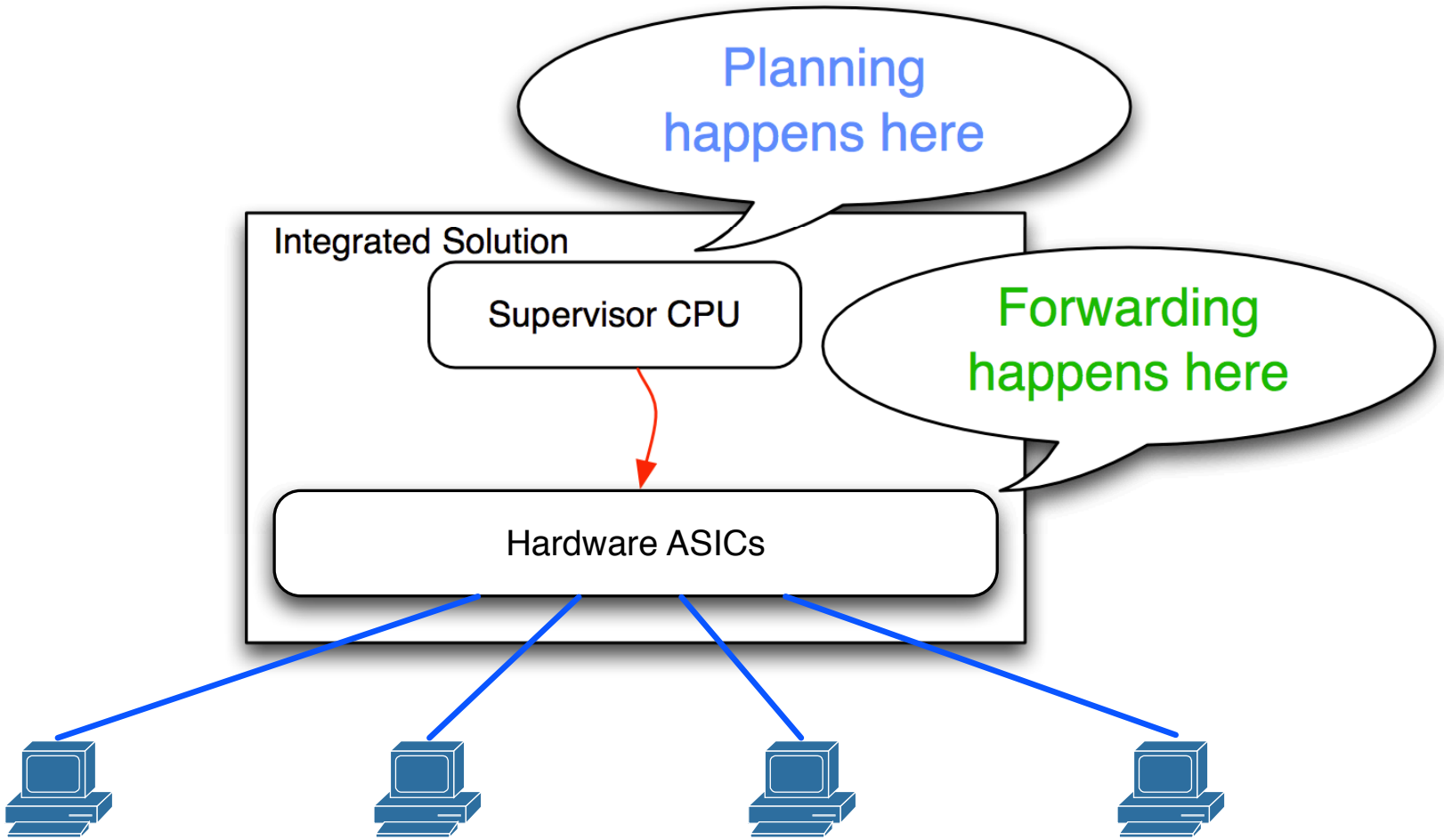
Software Defined Networking – 1/4

Software Defined Networking ("SDN")
. . . is a buzzword

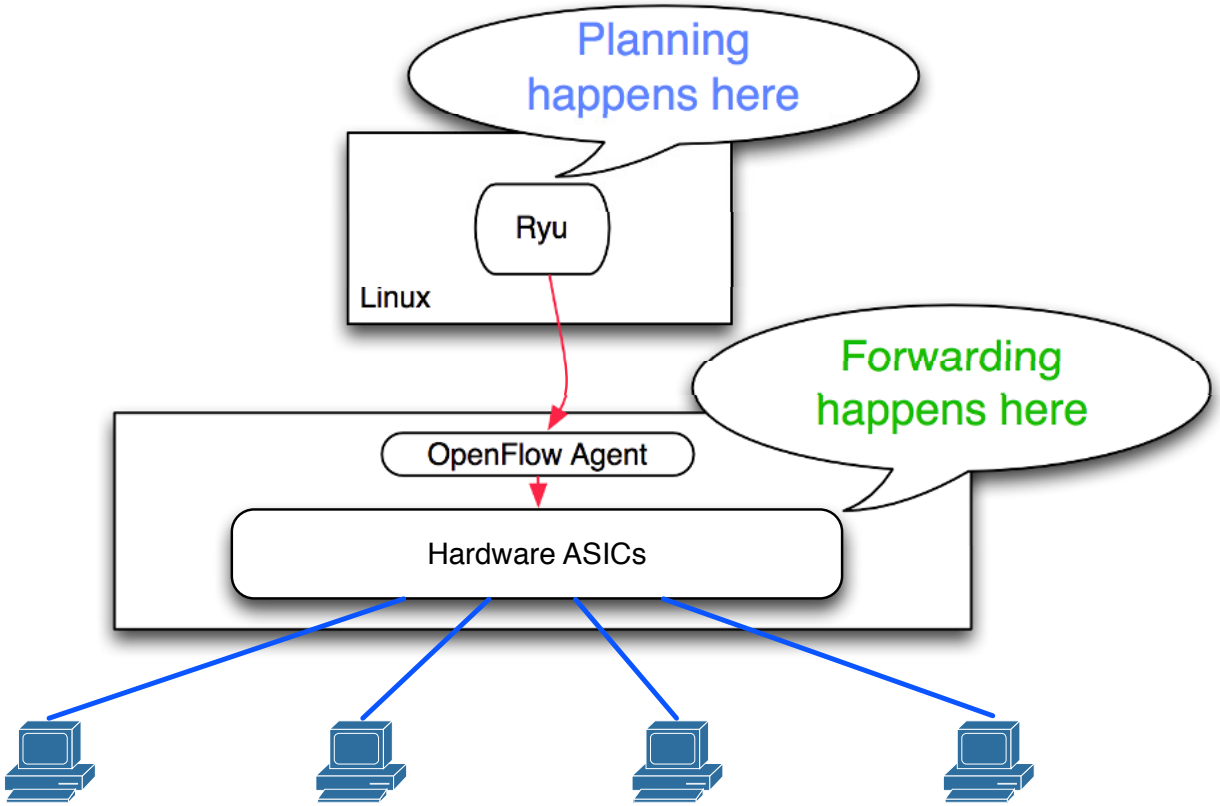
- All modern networking is "Software Defined"
- Contrast with "Hardware" defined network
- ie, external switch or router appliance
- Which traditionally has proprietary network stack

Software Defined Networking – 2/4

Modern hardware switch/router:



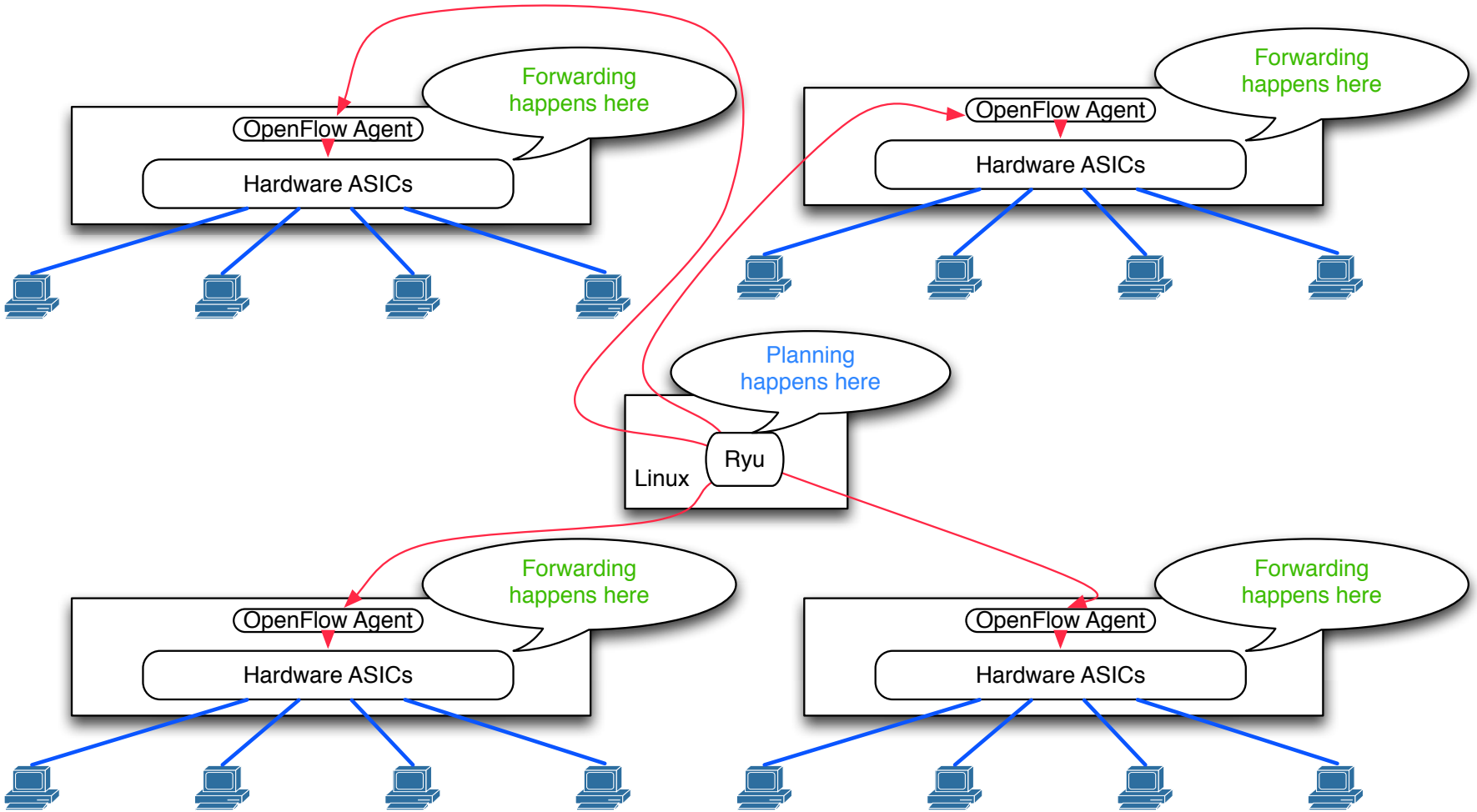
Software Defined Networking – 3/4



Software Defined Networking is the radical concept that the supervisor CPU and the forwarding hardware do not have to be in the same box, or be from the same vendor.

Software Defined Networking – 4/4

1:1 hardware:supervisor CPU is optional too :-)



OpenFlow

OpenFlow is a standarised protocol for communication between a SDN Controller and separate forwarding hardware.

- OpenFlow 1.0: Dec 2009: IPv4 only, limited features
- OpenFlow 1.3: Jun 2012: IPv4 and IPv6, tables, etc
- OpenFlow 1.4: Oct 2013: not widely implemented yet

- TCP/6633 (older convention)
- TCP/6653 (standardised 2013-07-18)
- TLS recommended since OpenFlow 1.3

Ryu revisited

Ryu is an OpenFlow Controller written in Python, which can be used to create a Software Defined Network

- Lets you write software
- To control network forwarding hardware
- Using the full power of Python

Installing Ryu

Ryu is on PyPI:

```
pip install ryu
```

From git source:

```
git clone git://github.com/osrg/ryu.git
cd ryu; python ./setup.py install
```

Dependencies:

- Many modern Python dependencies
- Most tested with Python 2.7, on Linux

From Ubuntu Linux 14.04 packages:
http://ewen.mcneill.gen.nz/blog/entry/2014-08-31-ryu-on-ubuntu-14-04/

Development environment

Development environment needs:

- ryu-manager and your Ryu application
- an OpenFlow compatible switch
- two or more systems to generate traffic
- way to see OpenFlow messages

Modern Linux includes Open vSwitch

- `http://openvswitch.org/`
- replacement for Linux software bridge (brctl, etc)
- supports OpenFlow 1.0/1.3/1.4

Mininet

Mininet is a Python project that provides a realistic virtual network.

- `http://mininet.org/`
- Provides Python classes wrapping Linux networking
- ... and Linux container features
- Wire up useful test network, including OpenFlow switch, and test systems, using Python objects

Installation:

- Install as test VM, from Linux distribution, or from git
- `http://mininet.org/download/`
- Mininet 2.10 packaged in Ubuntu Linux 14.04 LTS

Wireshark

Wireshark:

- `https://www.wireshark.org/`
- v1.12 (released 2014-07-31) has OpenFlow dissector
- Probably need to build from source or use upstream binary
- . . . unless you run bleeding edge distro

Usage:

- tshark -Ttext -d tcp.port==6633,openflow -O openflow_v4 -P -tad
- `http://wiki.wireshark.org/OpenFlow`
- $HOME/.wireshark/preferences:
  - openflow.tcp.port: 6633 (historical convention)
  - openflow.tcp.port: 6653 (standardised)

Ryu applications

A Ryu application:

- is a Python class (subclass of ryu.base.app_manager.RyuApp)
- that is event driven
- ryu-manager can run multiple applications at once
- one *light weight* thread per app
- apps can pass messages to each other, to cooperate

For more detail see:

- Documentation:
  `http://ryu.readthedocs.org/en/latest/`
- Ryu book (free PDF/eBook/HTML, with 10 worked examples):
  `http://osrg.github.io/ryu/resources.html#books`

Minimal Ryu application (kiwipycon1.py)

```
from ryu.base import app_manager

class KiwiPycon(app_manager.RyuApp):
  def __init__(self, *args, **kwargs):
    super(KiwiPycon, self).__init__(*args, **kwargs)
```

Running applications (with default config):

```
ryu-manager ./kiwipycon1.py
```

May have to override default config (eg, avoid default log to /var/log/ryu/ryu.log; see eg, /etc/ryu/ryu.conf):

```
touch ryu.conf
ryu-manager --config-file ./ryu.conf ./kiwipycon1.py
```

(and may have to stop Ryu service if installed from packages)

Minimal Ryu OpenFlow application (kiwipycon2.py)

```
from ryu.base                    import app_manager
from ryu.controller              import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto                 import ofproto_v1_3

class KiwiPycon(app_manager.RyuApp):
  OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION] #OpenFlow 1.3

  def __init__(self, *args, **kwargs):
    super(KiwiPycon, self).__init__(*args, **kwargs)

  @set_ev_cls(ofp_event.EventOFPStateChange,
              MAIN_DISPATCHER)
  def new_connection(self, ev):
    dp = ev.datapath
    self.logger.info("Switch connected (id=%s)" % dp.id)
```

OpenFlow “flows”

- Similar model to firewall ACL
- Designed to be implemented in hardware ASIC
- Stateless (except new flows created by controller)
- Openflow “flows” consist of:
  - A priority (higher priority wins)
  - Timeout options (clock time, since last matched)
  - Cookie (optional tag)
  - Match pattern (with wildcards)
  - Instructions (OpenFlow 1.2+)
- Arranged into “tables” (OpenFlow 1.2+)
- Processed as a pipeline, starting table 0

OpenFlow matches

- Model is a set of wildcarded matches:
  - Layer 1: input port
  - Layer 2: src MAC, dst MAC, Ethernet frame type, ...
  - Layer 3: src IP, dst IP, ...
  - Layer 4: src TCP port, dst TCP port, ICMP type, ...
- Hardware ASIC may have limits on combinations
- Combinations sometimes configurable, sometimes not
- Software implementations (eg, Open vSwitch) usually flexible

OpenFlow Instructions

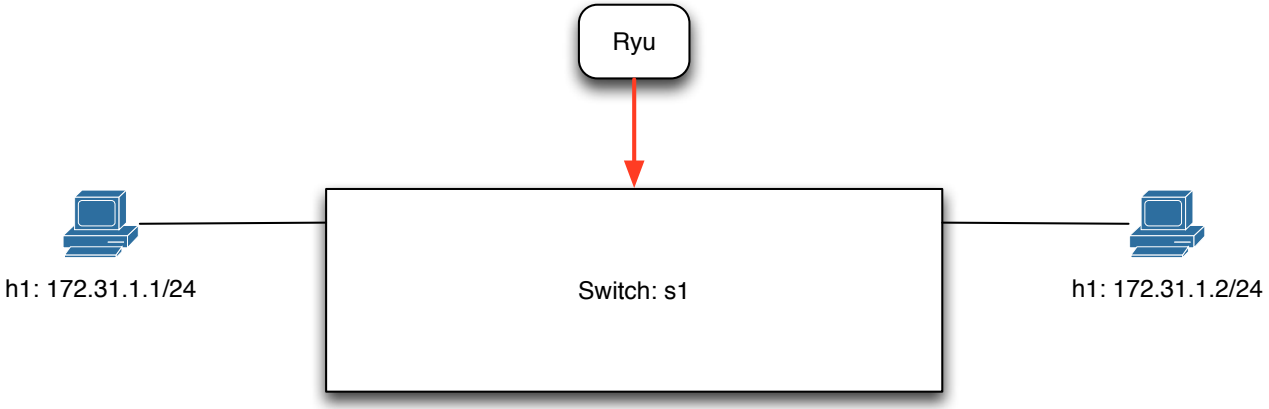
- OpenFlow 1.2+ only
- OpenFlow 1.0 only had actions
- Instructions:
  - Goto table N
  - Write Action
  - Apply Action immediately (optional)
  - Clear Actions
  - Write Metadata (for later matching)
  - Apply meter (rate limiting)

OpenFlow Actions

- Output frame to port(s)
  - Specific physical port
  - ALL ports
  - To controller
  - In port (back out port received on)
  - Normal/Flood
- Push/Pop VLAN tags
- Push/Pop MPLS tags
- Set queue

(Many of these are OpenFlow 1.2+)

Example network



- Two endpoints (h1 and h2)
- Separated by an OpenFlow capable switch
- Controlled by a Ryu application

MiniNet code for example network:
http://www.naos.co.nz/talks/seize-control-with-ryu/kiwipycon-mininet.py

Example network – Mininet 1/3

```
#! /usr/bin/python
# Simple Mininet network: host -- switch -- host

from mininet.net  import Mininet
from mininet.node import OVSSwitch, RemoteController
from mininet.topo import Topo
from mininet.log  import setLogLevel
from mininet.cli  import CLI
from mininet.util import run

setLogLevel('info')
#setLogLevel('debug')     # For diagnostics

# ... (continued next slide) ...
```

Example network – Mininet 2/3

```
# ... (continued from previous slide) ...

# Implement host - switch - host topology
class KiwiPycon2014(Topo):
  def __init__(self):
    super(KiwiPycon2014, self).__init__()
    leftHost  = self.addHost('h1',ip='172.31.1.1/24')
    rightHost = self.addHost('h2',ip='172.31.1.2/24')
    oneSwitch = self.addSwitch(
                    's1', dpid='0000000000000099',
                    listenPort=6634)
    self.addLink(leftHost,  oneSwitch)
    self.addLink(oneSwitch, rightHost)

# ... (continued next slide) ...
```

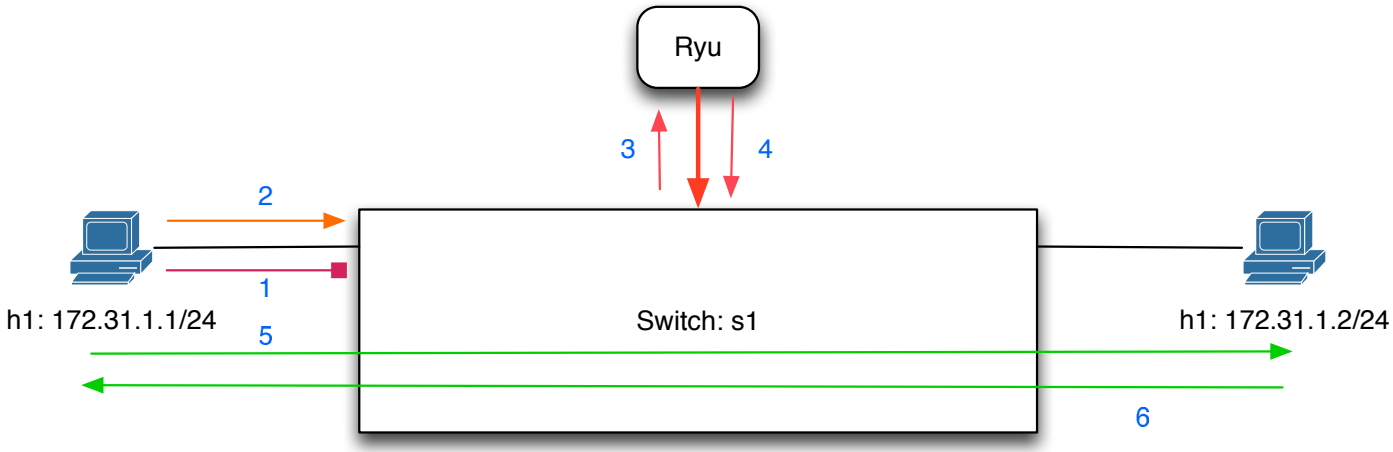
Example network – Mininet 3/3

```
# ... (continued from previous slide) ...

ryu = RemoteController('ryu', ip='127.0.0.1', port=6633)
net = Mininet(topo=KiwiPycon2014(), switch=OVSSwitch,
              build=False)
net.addController(ryu)
net.build()
net.start()

# Explicitly enable OpenFlow 1.3, then run the network
run("ovs-vsctl set bridge s1 protocols=OpenFlow13")
CLI(net)
net.stop()
```

Example behaviour



- h1 wants to communicate with h2
- OpenFlow switch stops h1 talking to h2 (1)
- Until a magic unlock token is seen (2, 3, 4)
- Then h1 is allowed to communicate with h2 (5,6)
- No assistance required from h1 or h2

Ryu/OpenFlow requirements

1. IPv4 traffic from h1 should be blocked by default
2. Need a way to allow traffic (overriding default)
3. Need a way to trigger “allow traffic”:
   - UDP packet
   - Containing “xyzzy”
4. Simplifying assumptions:
   - ARP should be unrestricted
   - h2 only responds, never initiates (stealth!)
   - IPv4 only (IPv6 is exercise for the reader!)
   - Flood traffic (for simplicity)
   - (Mostly) ignore race conditions, errors

0. On-connect policy

```
# ... (imagine Ryu application boilerplate here) ...

class KiwiPycon(app_manager.RyuApp):
  # Internal constants for ports, priority, etc
  MAGIC_COOKIE                    = bytearray(b"xyzzy")
  (PORT_H1, PORT_H2)              = (1,2)
  (PRI_LOW, PRI_MID, PRI_HIGH) = (20, 30, 40)

  @set_ev_cls(ofp_event.EventOFPStateChange,
              MAIN_DISPATCHER)
  def new_connection(self, ev):
    dp = ev.datapath
    self.logger.info("Switch connected (id=%s)" % dp.id)
    self.block_traffic_by_default(dp)
    self.flood_all_arp(dp)
    self.add_notify_on_udp_from_host_1(dp)
```

1. Block traffic from h1 by default

```
def block_traffic_by_default(self, dp):
  ofp    = dp.ofproto
  parser = dp.ofproto_parser

  self.logger.info("Clearing existing flows")
  self.del_flows(dp)

  self.logger.info("Blocking traffic from h1's port")
  match   = parser.OFPMatch(in_port=KiwiPycon.PORT_H1)
  self.add_flow(dp, KiwiPycon.PRI_LOW, match, None)

  self.logger.info("Allowing traffic from h2's port")
  match   = parser.OFPMatch(in_port=KiwiPycon.PORT_H2)
  actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD,
                                    ofp.OFPCML_NO_BUFFER)]
  self.add_flow(dp, KiwiPycon.PRI_LOW, match, actions)
```

2. Allow all ARP

```
from ryu.ofproto import ofproto_v1_3, ether, inet
# ...

  def flood_all_arp(self, dp):
    ofp    = dp.ofproto
    parser = dp.ofproto_parser

    self.logger.info("Permitting ARP, by flooding")
    match = parser.OFPMatch(eth_type=ether.ETH_TYPE_ARP)
    actions = [parser.OFPActionOutput(ofp.OFPP_FLOOD,
                                      ofp.OFPCML_NO_BUFFER)]
    self.add_flow(dp, KiwiPycon.PRI_MID,
                  match, actions)
```

3. Send us UDP so we can look for cookie”

```
def add_notify_on_udp_from_host_1(self, dp):
  ofp    = dp.ofproto
  parser = dp.ofproto_parser

  self.logger.info("Request notify on UDP from h1")
  match = parser.OFPMatch(in_port  = KiwiPycon.PORT_H1,
                          eth_type = ether.ETH_TYPE_IP,
                          ip_proto = inet.IPPROTO_UDP)
  actions = [parser.OFPActionOutput(
                        ofp.OFPP_CONTROLLER,
                        ofp.OFPCML_NO_BUFFER)]

  self.add_flow(dp, KiwiPycon.PRI_MID, match, actions)
```

4. Look at traffic we are sent

```
from ryu.lib.packet import packet, ethernet
#...

  @set_ev_cls(ofp_event.EventOFPPacketIn,
              MAIN_DISPATCHER)
  def handle_packet(self, ev):
    pkt = packet.Packet(ev.msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)

    self.logger.info("UDP received from %s" % eth.src)

    if ev.msg.data.find(KiwiPycon.MAGIC_COOKIE) >= 0:
      self.logger.info("Magic cookie found from %s" \
                       % eth.src)
      self.permit_traffic_from_mac(ev.msg.datapath,
                                   eth.src)
```

5. Permit traffic by MAC (if we found cookie)

```
def permit_traffic_from_mac(self, dp, src_mac):
  ofp    = dp.ofproto
  parser = dp.ofproto_parser

  self.logger.info("Permitting traffic from %s" \
                   % src_mac)
  match   = parser.OFPMatch(eth_src = src_mac)
  actions = [parser.OFPActionOutput(
                        ofp.OFPP_FLOOD,
                        ofp.OFPCML_NO_BUFFER)]
  self.add_flow(dp, KiwiPycon.PRI_HIGH,
                match, actions)
```

Util: add flows helper

```
def add_flow(self, dp, priority, match, actions):
  ofp    = dp.ofproto
  parser = dp.ofproto_parser
  inst   = []
  if actions:
    inst = [parser.OFPInstructionActions(
                              ofp.OFPIT_APPLY_ACTIONS,
                              actions)]

  mod = parser.OFPFlowMod(datapath=dp, table_id=0,
                              priority=priority,
                              match=match,
                              instructions=inst)
  dp.set_xid(mod)             # Preallocate transaction ID
  dp.send_msg(mod)
```

Util: delete all flows helper

```
def del_flows(self, dp):
    ofp    = dp.ofproto
    parser = dp.ofproto_parser

    wildcard_match = parser.OFPMatch()
    instructions   = []

    mod = parser.OFPFlowMod(datapath=dp, table_id=0,
                            command    = ofp.OFPFC_DELETE,
                            out_port   = ofp.OFPP_ANY,
                            out_group  = ofp.OFPP_ANY,
                            match      = wildcard_match,
                            instructions=instructions)

    dp.send_msg(mod)
```

Terminal 1: Running Mininet

```
ewen@mininet:~$ sudo ./kiwipycon-mininet.py
[sudo] password for ewen:
Unable to contact the remote controller at 127.0.0.1:6633
*** Creating network
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Terminal 2: Running Ryu

```
ewen@mininet:~$ ryu-manager \
> --config-file ./ryu.conf kiwipycon3.py
loading app kiwipycon3.py
loading app ryu.controller.ofp_handler
instantiating app kiwipycon3.py of KiwiPycon
instantiating app ryu.controller.ofp_handler of OFPHandler
Switch connected (id=153)
Clearing existing flows
Blocking traffic from h1's port by default
Allowing traffic from h2's port by default
Permitting ARP, by flooding
Request notify on UDP from h1
```

Test ping h1 to h2

```
mininet> h1 ping -c 5 h1
PING 172.31.1.1 (172.31.1.1) 56(84) bytes of data.
64 bytes from 172.31.1.1: icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from 172.31.1.1: icmp_seq=2 ttl=64 time=0.028 ms
64 bytes from 172.31.1.1: icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from 172.31.1.1: icmp_seq=4 ttl=64 time=0.035 ms
64 bytes from 172.31.1.1: icmp_seq=5 ttl=64 time=0.034 ms

--- 172.31.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3998ms
rtt min/avg/max/mdev = 0.013/0.028/0.035/0.007 ms
mininet> h1 ping -c 5 h2
PING 172.31.1.2 (172.31.1.2) 56(84) bytes of data.

--- 172.31.1.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4001ms

mininet>
```

Ready to knock? Turn the key...

In MiniNet:

```
mininet> h1 dig @172.31.1.2 +time=1 +tries=1 +short xyzzy.example.com
;; connection timed out; no servers could be reached
mininet>
```

Ryu application responds:

```
UDP received from 4e:19:42:3f:41:b5
Magic cookie found from 4e:19:42:3f:41:b5
Permitting traffic from 4e:19:42:3f:41:b5
```

It's Play School

```
mininet> h1 ping -c 5 h2
PING 172.31.1.2 (172.31.1.2) 56(84) bytes of data.
64 bytes from 172.31.1.2: icmp_seq=1 ttl=64 time=0.283 ms
64 bytes from 172.31.1.2: icmp_seq=2 ttl=64 time=0.045 ms
64 bytes from 172.31.1.2: icmp_seq=3 ttl=64 time=0.052 ms
64 bytes from 172.31.1.2: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 172.31.1.2: icmp_seq=5 ttl=64 time=0.053 ms

--- 172.31.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.045/0.097/0.283/0.093 ms
mininet>
```

Flows: before we unlocked...

```
ewen@mininet:~$ ovs-ofctl -O OpenFlow13 dump-flows tcp:127.0.0.1:6634
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=2.628s, table=0, n_packets=0, n_bytes=0,
             priority=30,arp actions=FLOOD
 cookie=0x0, duration=2.628s, table=0, n_packets=0, n_bytes=0,
             priority=20,in_port=1 actions=drop
 cookie=0x0, duration=2.628s, table=0, n_packets=0, n_bytes=0,
             priority=20,in_port=2 actions=FLOOD
 cookie=0x0, duration=2.628s, table=0, n_packets=0, n_bytes=0,
             priority=30,udp,in_port=1 actions=CONTROLLER:65535
ewen@mininet:~$
```

Flows: ... and after

```
ewen@mininet:~$ ovs-ofctl -O OpenFlow13 dump-flows tcp:127.0.0.1:6634
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=130.113s, table=0, n_packets=2, n_bytes=84,
             priority=30,arp actions=FLOOD
 cookie=0x0, duration=130.113s, table=0, n_packets=0, n_bytes=0,
             priority=20,in_port=1 actions=drop
 cookie=0x0, duration=130.113s, table=0, n_packets=5, n_bytes=490,
             priority=20,in_port=2 actions=FLOOD
 cookie=0x0, duration=130.113s, table=0, n_packets=1, n_bytes=88,
             priority=30,udp,in_port=1 actions=CONTROLLER:65535
 cookie=0x0, duration=109.593s, table=0, n_packets=7, n_bytes=574,
             priority=40,dl_src=4e:19:42:3f:41:b5 actions=FLOOD
ewen@mininet:~$
```

That's All Folks!

Ryu and OpenFlow:

- Flexibility of Python, speed of hardware
- Mininet lets you make test networks in Python
- Wireshark invaluable for seeing interactions

Questions?

Slides:

`http://www.naos.co.nz/talks/seize-control-with-ryu/`

Examples (in same directory):

- kiwipycon-mininet.py
- kiwipycon3.py