Technology Validation Experiment: IPv6 and Multicast Support on OpenFlow

David R Newman

February 11, 2014

1 Introduction

The purpose of this technology validation experiment is evaluate the state-of-the-art in regards to OpenFlow support for IPv6 and IPv6 multicast. The original specification for this experiment can be found in section 5.7 of Specification of phase 1 business experiments (D6.6.2)[5]. This document set out three environment for evaluating IPv6 and IPv6 multicast:

1. A virtualized environment configured using Mininet[1].
2. A test network created on the OFELIA testbed[2].
3. A test network of OpenFlow switches (both physical and software-b/ased) and virtual machine servers, specifically built for the OFERTIE project.

IPv6 is only supported on OpenFlow switches that run version 1.2 or higher of the OpenFlow firmware. Additionally, IPv6 features were added in OpenFlow version 1.3 [6]. Version 1.4 of the OpenFlow specification was released in October 2013 but this has no new IPv6 features and is yet to have any robust implementations as of November 2013 [7].

Also as of November 2013, switches on the OFELIA testbed islands typically only run OpenFlow 1.0 and no switches run OpenFlow 1.3. Also, the OpenFlow test network for the OFERTIE project has yet to be setup. Therefore, only results for the virtualized environment have been recorded initially.

1.1 IPv6 Features in OpenFlow

Version 1.3 of the OpenFlow specification has a number of minor versions (1.3.0, 1.3.1, 1.3.2 and 1.3.3), the latest of these was published in October 2013, therefore amendments to this may have not yet have be incorporated into the firmware for either physical or software switches purporting to support OpenFlow 1.3, as of November 2013. However, version 1.3.3 of the specification sets out the following IPv6 features [6]:

- From OpenFlow 1.2 there has been support for matching the following basic IPv6 header fields:
  - Source address
  - Destination address
  - Protocol number
  - Traffic class
  - ICMPv6 type
  - ICMPv6 code
  - Neighbour discovery

  It also provides support for matching an IPv6 flow label.
- From OpenFlow 1.3 there has been support for matching the following common IPv6 extension header fields:
  - Hop-by-hop
  - Router
  - Fragmentation

[1] http://mininet.org/
[2] https://alpha.fp7-ofelia.eu/doc/index.php/Main_Page

- Destination options
- Authentication
- Encrypted Security Payload
- No Next Header

There is also support to handle unexpected or out of order IPv6 extension header fields.

2 Virtualized Environments

Ubuntu 12.04.3 LTS was chosen as the operating system on which to setup virtualized environments. This is a well-known version of Linux that has long-term support and it a suitable operating system for installing most if not all software implementations of OpenFlow switches (OpenFlow soft switches), OpenFlow controllers and related applications. In particular the Mininet GitHub repository provides an installer script, which can install many of these components is only designed for Debian/Ubuntu operating systems.

Although there are quite a few software implementations of OpenFlow soft switches, only a few of them support OpenFlow 1.3:

1. Open vSwitch[3]
2. CPqD OpenFlow Switch[4]
3. LINC[5]

The operating system chosen is capable of supporting all these soft switches, however for simplicity the virtualized environment setup will only run one type of switch at any one time.

As well as requiring OpenFlow soft switches that support version 1.3, it is also necessary for the OpenFlow controller to support this version. The OpenFlow wiki lists six different open source OpenFlow controllers but there is fairly limited support for OpenFlow version 1.3 [8]:

Beacon[6] Does not specify which versions of OpenFlow it can support. However, it uses the OpenFlowJ library to work with OpenFlow messages, which is only based on the OpenFlow 1.0 specification. [1]

Floodlight[7] Only supports version 1.0 and states that the timeline for 1.2/1.3 is "currently unknown'. [12]

NOX and POX[8] Officially NOX only supports OpenFlow version 1.0. [3] As POX is just a python-implemented version of NOX it is assumed it also only supports version 1.0. [4] However, CpQD as part of developing their OpenFlow 1.3 soft switch have developed a NOX library for OpenFlow version 1.3. [11]

Trema[9] Only supports OpenFlow version 1.0. [10]

Ryu[10] Supports OpenFlow versions 1.0, 1.2 and 1.3[2]

Open Daylight[11]]Does not specify which versions of OpenFlow it can support. However, it is not expected that a download will be available until Q4 of 2013 and it does not have any apparent code repositories. [9]

This leaves several combinations of OpenFlow 1.3 switch and controller setups. The three most practicable options are:

1. The CpQD OpenFlow switch with CpQD's augmented NOX controller library that supports OpenFlow 1.3.
2. The Open vSwitch switch with the Ryu controller library.
3. The LINC Switch with the Ryu controller library.

[3] http://openvswitch.org/
[4] https://github.com/CPqD/ofsoftswitch13
[5] https://github.com/FlowForwarding/LINC-Switch
[6] http://www.beaconcontroller.net/
[7] http://floodlight.openflowhub.org/
[8] http://noxrepo.org/
[9] http://trema.github.io/trema/
[10] http://osrg.github.io/ryu/
[11] http://www.opendaylight.org/

2.1 Installing Switch and Controller Applications

To install the latest versions of all the applications required in three most practicable options described previously, the following Git repositories need to be cloned:

```
$ git clone git://github.com/mininet/mininet
$ git clone git://openflowswitch.org/openflow.git
$ git clone http://git.openvswitch.org/git/openvswitch
$ git clone https://github.com/FlowForwarding/LINC-Switch.git
$ git clone https://github.com/osrg/ryu.git
```

Once cloned, the applications in these repositories can be installed with the commands that follow. The MiniNet install script will also install the CPqD OpenFlow Switch (of3softswitch), the NOX OpenFlow 1.3 library, the OpenFlow Wireshark dissector and the benchmarking tool *oflops*.

```
$ sudo apt-get install fakeroot build-essential debhelper autoconf automake libssl-dev \
    bzip2 vlan traceroute openssl graphviz python python-all procps python-qt4 \
    python-zopeinterface python-twisted-conch dkms make libc6-dev module-init-tools  \
    uuid-runtime netbase python-argparse python-nose python-pexpect
$ sudo modprobe 8021q
$ mininet/util/install.sh -3fmnwb
$ cd openvswitch
$ dpkg-checkbuilddeps
$ fakeroot debian/rules binary
$ sudo dpkg -i openvswitch-common_2.0.90-1_amd64.deb \
    openvswitch-controller_2.0.90-1_amd64.deb openvswitch-datapath-dkms_2.0.90-1_amd64.deb \
    openvswitch-pki_2.0.90-1_amd64.deb openvswitch-switch_2.0.90-1_amd64.deb
$ sudo update-rc.d -f openvswitch-controller remove
$ sudo update-rc.d -f openvswitch-switch remove
$ sudo service openvswitch-controller stop
$ sudo service openvswitch-switch stop
$ cd ../LINC-Switch
$ wget http://packages.erlang-solutions.com/erlang-solutions_1.0_all.deb
$ sudo dpkg -i erlang-solutions_1.0_all.deb
$ sudo apt-get update
$ sudo apt-get install erlang bridge-utils libpcap0.8 libpcap-dev libcap2-bin uml-utilities
$ make
$ cd ../ryu
$ sudo apt-get install python-pip mz tcpreplay python-eventlet python-routes python-webob \
    python-paramiko python-virtualenv
$ sudo python ./setup.py install
$ cd ~
$ wget http://yuba.stanford.edu/~grg/spanning_tree.101122.tgz
$ cd ~/nox13oflib/src/nox/netapps
$ tar -xzvf ~/spanning_tree.101122.tgz
$ cd ~/nox13oflib/

$ git clone https://github.com/drn05r/ofertie-scripts.git
```

After the above commands have been run, the LINC switch will still need to be configured. This can be done by copying the file in Appendix A and writing over *LINC-Switch/rel/linc/releases/1.0/sys.config*.

The NOX OpenFlow 1.3 Controller library cannot manage topologies with loops by default. To allow it to support loops a spanning tree *net application* is required. Download the following application to the user's home directory:

`http://yuba.stanford.edu/~grg/spanning_tree.101122.tgz`

To install this spanning tree as part of the NOX OpenFlow 1.3 Controller library, next edit *~/nox13oflib/configure.ac.in* and set the *netapps* ACI package to:

```
ACI_PACKAGE([netapps],[misc network apps],
               [discovery
                #add netapps component here
                spanning_tree],
               [TURN_ON_NETAPPS])
```

Then edit *~/nox13oflib/src/etc/nox.json* and set the *Packet_in_ event* block as follows:

```
"Packet_in_event": [
                "trackhost_pktin",
                "hostip",
                "spanning_tree",
                "authenticator",
                "dwh",
                "noop",
                "counter",
                "packetdump",
                "simplerouting",
                "ctlflood",
                "switch",
                "hub",
                "cswitchstats"
                ],
```

Now carry out the following instructions to unpack and build the spanning tree as part of the NOX OpenFlow 1.3 Controller library:

```
$ cd ~/nox13oflib/src/nox/netapps
$ tar -xzvf ~/spanning_tree.101122.tgz
$ cd ~/nox13oflib/
$ ./boot.sh
$ cd build/
$ ../configure -with-python=yes
$ make
```

Now all the applications are installed, various OFERTIE scripts required for this technology validation experiment can be downloaded from GitHub with the following command:

```
$ git clone https://github.com/drn05r/ofertie-scripts.git
```

The scripts relevant to this technology validation experiment can be found in the *tve_ipv6_and_multicast* folder.

2.2 Running Virtualized Environments

After investigating the various virtualized environments, it was decided to focus on a single setup, so effort could be concentrated on testing the various aspects of IPv6 functionality rather than using it trying to configure multiple environments. From the exploratory work with the three environments it was determined that the CpQD OpenFlow switch and modified NOX controller was the most mature and well documented environment. There is some basic detail for how to setup the Open vSwitch and LINC switches with the Ryu controller but because the datapath control script (*dpctl*) differs significantly between the environments it was decided to focus on producing a more comprehensive set of tests for a single environment.

2.2.1 CpQD Switch and CpQD-modified NOX Controller (CqPD-NOX Setup)

To run the CpQD OpenFlow switch and controller do the following:

```
$ cd nox13oflib/build/src/
$ ./nox_core -v -i ptcp:6633 switch
```

Now, create a MiniNet network with a pre-define topology by running the following command substituting X with the appropriate topology number as described in section 5:

```
$ sudo mn --custom ~/ofertie-scripts/tve_ipv6_and_multicast/ofertie-topos.py \
    --topo topoN --mac --switch user --controller remote
```

2.2.2 Open vSwitch and Ryu Controller Library (Open vSwitch-Ryu Setup)

By default, having used the installation instruction described in section 2.1, Open vSwitch with be disabled. This needs to be enabled and the Ryu controller needs to be invoke before MiniNet can be loaded with a topology.

```
$ sudo service openvswitch-switch start
$ cd ~/ryu
$ PYTHONPATH=. ./bin-ryu-manager ryu/app/simple_switch.py
```

Once, Open vSwich and Ryu are running MiniNet can be started with a pre-defined OFERTIE network topology, substituting X for the appropriate topology. Now, create a MiniNet network with a pre-define topology by running the following command substituting N with the appropriate topology number as described in section 5. This command will allow you to interact with topology, define datapath commands using *ovs-dpctl* and run tests using ping, iperf, tcpdump, etc.

```
$ sudo mn --custom ~/ofertie-scripts/tve_ipv6_and_multicast/ofertie-topos.py \
    --topo topoN --mac --switch user --controller remote
```

2.2.3 LINC Switch and Ryu Controller Library (LINC-Ryu Setup

TO BE FINISHED

```
$ ~/LINC-Switch/rel/linc/bin/linc console
```

3 OFELIA Testbed Test Network Setup

At present no OFELIA testbed island supports OpenFlow 1.3. Although islands may be OpenFlow switches that can support OpenFlow 1.3, even if these were upgraded to this version the test islands would still only be able to support OpenFlow 1.0. This is because FlowVisor, which manages the dataflows so multiple virtual test networks can be run on a single physical network only supports OpenFlow 1.0. As FlowVisor behaves like another switch in the network when it comes to negotiating the version of OpenFlow to use, the OpenFlow controller will determine that version 1.0 should be used has the latest version supported by all components of the network.

There are no current plans by the developers of FlowVisor to update it so it can support OpenFlow 1.3. The OFELIA project is now at an end by there are a number of current OpenFlow projects and existent OFELIA testbed island hosts that are interested in a combined effort to either update FlowVisor or to find a replacement for it within the OFELIA setup that will allow OpenFlow 1.3 to be supported on OFELIA.

4 OFERTIE Test Network Setup

The OFERTIE test network has been setup to be a standalone OFELIA island this means at the present time it cannot be used as a test

5 Network Topologies

To test the capabilities of the switches and controllers that support OpenFlow 1.3, a number of increasingly more complex network topologies are required. These are intended to demonstrate that the switches and controllers support both basic and more advanced IPv6 and multicast functionality, as well as highlighting any current issues or bugs with these devices/applications.

5.1 Network Topology 1

Network topology 1 is an extremely basic network with two hosts connected together via two OpenFlow switches, (see Figure 1).

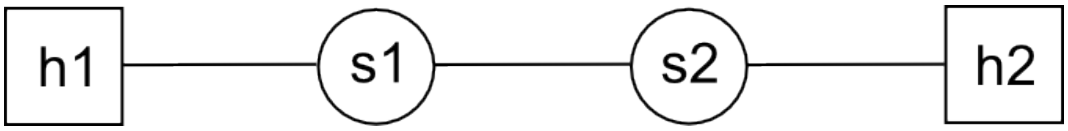

Figure 1: Network Topology 1

The purpose of this topology is to confirm that the OpenFlow switches and controller used can support IPv6 traffic being passed across the network.

5.2 Network Topology 2

Network topology 2 is only slightly more complex than the Network topology 1. The network has two hosts and three switches, (see Figure 2). As the first host is connected to two switches, there are to possible datapaths between the first and second host.

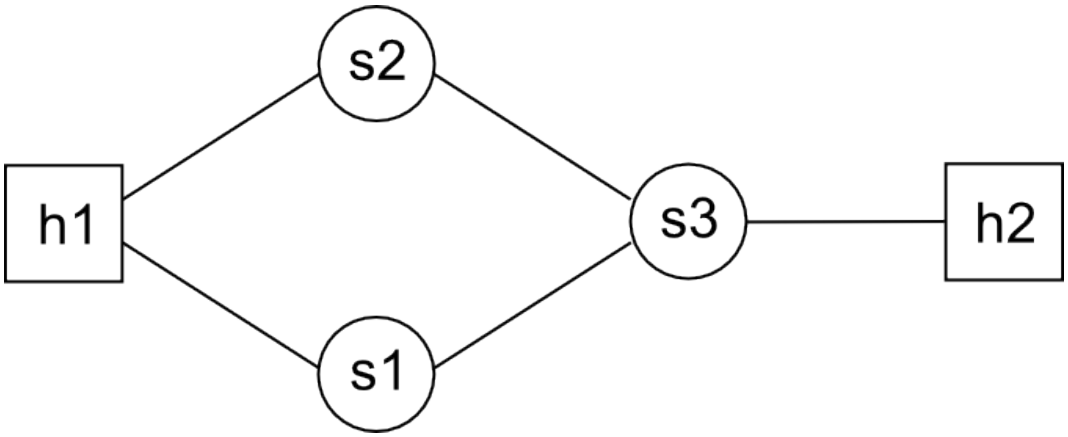

Figure 2: Network Topology 2

The purpose of this topology is to confirm that the OpenFlow controller can properly learn different datapaths. It will also allow for tools such as *dpctl* to be used to dynamically change the datapath being used between the two hosts and demonstrate that IPv6 traffic can still be sent between the client and host.

5.3 Network Topology 3

Network topology 3 is a somewhat more complex topology than topologies 1 and 2. The topology has three hosts and five switches, (see Figure 3). The purpose of this topology, is to allow multiple datapaths between the first client and the server host. The shortest of these datapaths will share hops with the single datapath between the second client host and the server host.

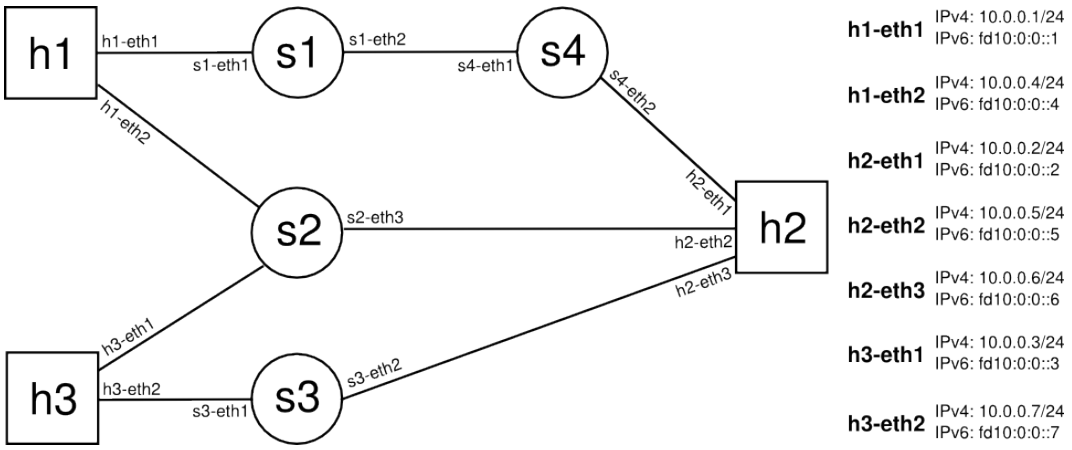

Figure 3: Network Topology 3

The purpose of this topology is to see how altering the levels of traffic between the second client host and the server host will effect the data rate between the first host ad the client. It should also be able to confirm that the datapath between the first client host and the server host can be changed to limit the impact for the other client host's traffic.

6 Results

6.1 Virtualized Environment

TO BE DONE

6.2 OFERTIE Test Network

TO BE DONE

A LINCS config file

```
{linc,
     [
      {of_config, enabled},
      {capable_switch_ports,
       [
        {port, 1, [{interface, "eth0"}]},
        {port, 2, [{interface, "eth0"}, {port_rate, {100, kbps}}]}
       ]},
      {logical_switches,
       [
        {switch, 0,
         [
          {backend, linc_us4},
          {controllers,
           [
            {"Switch0-DefaultController", "localhost", 6633, tcp}
           ]},
          {queues_status, disabled},
          {ports,
           [
            {port, 1, {queues, []}},
            {port, 2, {queues, []}}
           ]}
         ]}
       ]}
     ]}.
```

References

[1] David Erickson. The Beacon OpenFlow Controller. In *HotSDN*. ACM, 2013.

[2] Tomonori Fujita and Kei Ohmura. *Readme for Ryu*. Nippon Telegraph and Telephone Corporation, 2013. Retrieved 2013-11-05.

[3] Murphy McCauley. *About NOX*. NOXRepo.org, 2012. Retrieved 2013-11-05.

[4] Murphy McCauley. *About POX*. NOXRepo.org, 2012. Retrieved 2013-11-05.

[5] OFERTIE Project Team. D.6.6.1 specification of phase 1 business experiments. Technical report, EU Commission: Seventh Framework Programme (FP7), August 2013.

[6] Open Network Foundation. Openflow switch specification (version 1.3.3). Technical report, Open Network Foundation, October 2013.

[7] Open Network Foundation. Openflow switch specification (version 1.4.0). Technical report, Open Network Foundation, October 2013.

[8] Open Networking Foundation. *OpenFlow Wiki*, 2013. Retrieved 2011-11-05.

[9] OpenDaylight. *Open Daylight Download Page*, 2013. Retrieved 2013-11-05.

[10] Yasuhito Takamiya, Yasunobu Chiba, Denis Ovsienko, Nick Karanatsios, and Jari Sundell. *Readme for Trema*. Trema, 2013. Retieved 2013-11-05.

[11] Allan Vidal and Eder LeÃčo Fernandes. *Readme for NOX 1.3 Oflib*. CPqD, January 2013. Retrieved 2013-11-05.

[12] Kuang-Ching Wang, Volkan YazÄścÄś, Jason Parraga, and Bethany Kanui. *FAQ Floodlight OpenFlow Controller*. Project Floodlight, 2013. Retrieved 2013-11-05.