



FLOW FORWARDING

VERSION: DRAFT; DATED: FEBRUARY 14, 2014

Quick Start Guide

LINC OpenFlow Switch

Copyright Statements

Copyright © 2014 FlowForwarding.Org - All Rights Reserved.

The information in this document is provided “as is” and is subject to change without notice. FlowForwarding.Org shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

Trademark Statements

Any trademarked names used herein are the properties of their respective owners and are used for identification purposes only.

Company Information

FlowForwarding.org is a community promoting free, open-source, and commercially friendly Apache 2 license implementations based on OpenFlow and Open Network Foundation (ONF) specifications.

More information is available at <http://www.flowforwarding.org/>

Product Information

ONF Standards supported:

- OpenFlow - v 1.2, v 1.3.2

- OF-Config - v 1.1.1

For a list of latest supported hardware information, visit <http://www.flowforwarding.org/hardware>

Document Updated: February 14, 2014

Intended Audience

The Quick Start Guide for the LINC Open Flow Switch is intended for developers, system engineers, and IT senior administrators.

Contents

Introduction	3
LINC OpenFlow Switch Overview	3
LINC Architecture	3
Building a LINC OpenFlow Switch	5
Installation Requirements	6
Hardware Requirements	6
Software Requirements	7
Creating a Bootable USB Device	8
Writing a raw image to a USB device on Linux OS	9
Writing a raw image to a USB device on Mac OS X	10
Deployment Architecture	11
LINC on hardware	11
LINC on Virtual Machine	11
LINC on VMware vSphere 5.x	11
Installing and Configuring the LINC OpenFlow Switch	12
Testing the Installation of LINC OpenFlow Switch	12
About the config_gen script	13
Configuring Switch Capability	13
Erlang Configuration Syntax	15
Enabling OF-Config in LINC	16
OF-Capable Switch Model	16
OF-Logical Switch (OpenFlow Switch)	16
TLS Configuration	18
OF-Configuration Point via NetConf	19
Error Logging Parameters	19
Examples of sys.config File in LINC Switches	20
One Switch with three Controllers	20
Two Switches with different Controllers	22
Two Switches with same Controllers	25
OpenFlow Controllers	28
Ryu OpenFlow Controller	28
Warp OpenFlow Driver	28
Running the LINC OpenFlow Switch	37
Viewing the LINC Switch Information	38
Running Ping Demonstration	38
Setting up the environment	38
Ping Demonstration	41
Running LINC OpenFlow Switch with Mininet	43
About Mininet	43
Installing Mininet	43
Integrating LINC Switch with Mininet	43
Configuring LINC with OF-Config	45
Starting LINC without any Controller	45
Checking that there are no controllers configured	46

Adding a Controller via OF-Config	53
Checking whether a new controller is added	54
Testing and Tools	62
Iperf	62
cURL	62
Wireshark	62
Miscellaneous Tools	64
Case Studies	65
Deploying LINC on Corporate Networks	66
Customer Situation:	66
Solution:	66
Benefits:	66
Results:	66
Software Defined Networking (SDN) for OpenFlow Networks	67
Customer Situation:	67
Solution:	67
Benefits:	67
Results:	67
Deploying Dynamically Programmable Firewall for SDN	68
Customer Situation:	68
Solution:	68
Benefits:	68
Results:	68
Big Data Apache Hadoop Acceleration with OpenFlow	69
Customer Situation:	69
Solution:	69
Benefits:	69
Results:	69
Deploying Scalable and Programmable OpenFlow Switch with OF-Config	70
Customer Situation:	70
Solution:	70
Benefits:	70
Results:	70
Networking Tap/Monitoring with OpenFlow	71
Customer Situation:	71
Solution:	71
Benefits:	71
Results:	71
Feedback	71

Introduction

This guide provides an overview of the LINC OpenFlow Switch and explains how to install and configure the switch. It includes the following sections:

- [LINC OpenFlow Switch Overview](#)
- [Building a LINC OpenFlow Switch](#) on page 5
- [Running the LINC OpenFlow Switch](#) on page 37

LINC OPENFLOW SWITCH OVERVIEW

LINC (Link Is **Not** Closed) is an OpenFlow software switch developed using Erlang, a functional language which is implemented as an Erlang node in the user space of the operating system.

The following are some of the features of LINC OpenFlow Switch:

- Supports OpenFlow Protocol 1.2 and OpenFlow Protocol 1.3.
- Runs multiple logical switches on the OpenFlow capable switch.
- Supports OF-Config 1.1.1 management protocol.
- Modular architecture and easily extensible.

LINC, an open-source switching platform available through FlowForwarding.Org, a community that promotes free open source Apache 2 license implementations based on OpenFlow specifications. LINC is developed to serve as a reference implementation which includes the full feature set of OpenFlow specifications. The primary goals of developing LINC OpenFlow Switch are to allow users to develop LINC quickly and cost effectively on the COTS platform, to evaluate OpenFlow and OF-Config specifications, and to provide feedback to ONF (Open Networking Foundation).

LINC is written in Erlang, a programming language developed by Ericsson. It is a multi-purpose functional language that was designed from the ground up for writing scalable, fault-tolerant, distributed, non-stop, and soft real-time applications. The following features of Erlang has contributed to the rapid design and development of LINC Switch.

- Erlang OTP (Open Telecom Platform) is a large collection of libraries for Erlang that provides solutions for many common problems in networking and telecommunication systems. One example is the supervision tree.
- Erlang provides for extremely convenient bit-manipulation capabilities which makes it well suited for protocol handling and low-level communications.
- Erlang has a built-in support for creating and managing processes with the aim of simplifying concurrent programming.

Erlang is also well suited to exploit the new trend in many-core architectures that provide huge processing capabilities on COTS in a cost-effective way. Since the goal was to develop a software switch that runs on COTS, utilizing multiple cores for achieving high performance is an important secondary goal.

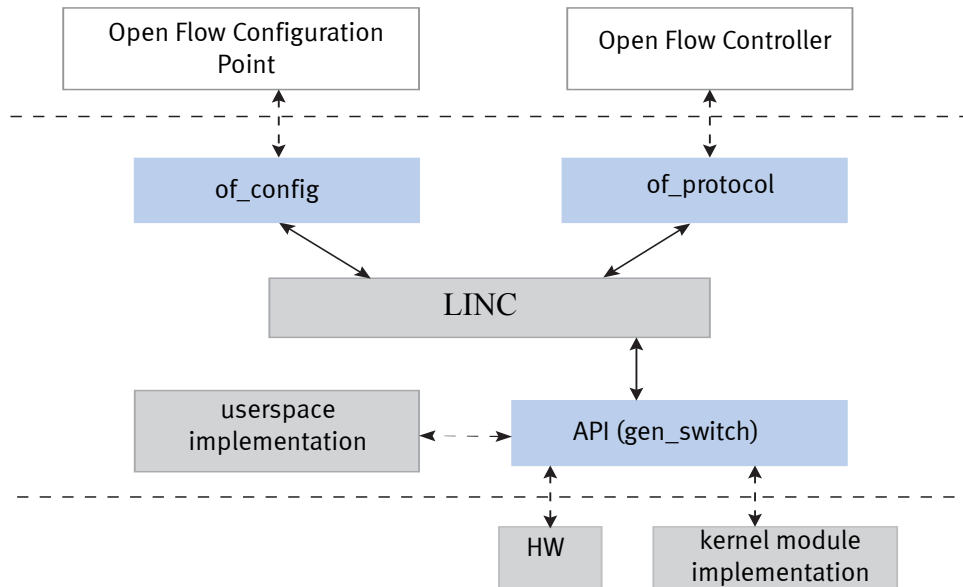
LINC Architecture

LINC is a software switch that fully implements OpenFlow 1.2, OpenFlow 1.3.x, and OF-Config 1.1.1 versions of the OpenFlow specifications that are applicable to pure OpenFlow switches. The block diagram in [Figure 1.1](#) shows its main software blocks - OpenFlow capable switch, OpenFlow protocol module, and OF-Config module. These are designed to follow the Erlang OTP principles and are developed as separate applications in which each application implements a certain well defined functionality, and has a clearly defined interface, dependencies, and supervision tree.

The OpenFlow protocol is implemented in the `of_protocol` library application that defines internal OpenFlow protocol structures, data types and enumerations, offering encode and decode functions for OpenFlow protocol messages and validates their correctness.

The functionality of the OpenFlow Capable switch is implemented in the LINC library. It receives OF-Config commands and executes them in the OpenFlow Operational Context. It handles one or more OpenFlow Logical switches that consist of the channel component, replaceable back-ends and common logical switch. The channel component is a communication layer between the OpenFlow logical switch and the OpenFlow controllers. It handles the TCP/TLS connections to the controllers. It uses the `of_protocol` library for encoding and decoding the OpenFlow Protocol messages. The channel component also passes parsed structures received from the OpenFlow controller at the backend and forwards encoded messages from OpenFlow switch to the controllers.

Figure 1.1 LINC Architecture



The replaceable back-ends implement the actual logic for switching the packets. They manage flow tables, group table, ports, etc. and reply to OpenFlow Protocol messages received from the controller. Due to the use of a common API (`gen_switch`), LINC's logical switch can use any of the available back-ends. Common switch logic does not depend on the back-end. It handles switch configuration, manages the channel component and OpenFlow resources, such as ports and dispatches the messages received from the controller.

The OF-Config protocol handling is implemented by the `of_config` library application that handles parsing, validation and generates an interpretation of the OF-Config messages received from an OpenFlow configuration point. This output is a set of commands to the OpenFlow capable switch application (LINC) to configure the OpenFlow capable switch (i.e. creates an instance of the OpenFlow logical switch, associates an OpenFlow resource with a specific OpenFlow logical switch, etc.)

In addition to these components, LINC has a supervision tree for fault tolerance purposes in accordance with the OTP principles. A supervisor is responsible for starting, stopping and monitoring its child processes. The basic idea of a supervisor is that it should keep its child processes alive by restarting them when necessary.

Building a LINC OpenFlow Switch

This chapter lists the requirements and the steps to install and configure the LINC OpenFlow Switch. Additionally, it includes information about configuring the switch capability and testing the installation. It includes the following sections:

- [*Installation Requirements*](#) on page 6
 - [*Hardware Requirements*](#) on page 6
 - [*Software Requirements*](#) on page 7
- [*Creating a Bootable USB Device*](#) on page 8
 - [*Writing a raw image to a USB device on Linux OS*](#) on page 9
 - [*Writing a raw image to a USB device on Mac OS X*](#) on page 10
- [*Deployment Architecture*](#) on page 11
 - [*LINC on hardware*](#) on page 11
 - [*LINC on VMware vSphere 5.x*](#) on page 11
- [*Installing and Configuring the LINC OpenFlow Switch*](#) on page 12
- [*Testing the Installation of LINC OpenFlow Switch*](#) on page 12
- [*About the config_gen script*](#) on page 13
- [*Configuring Switch Capability*](#) on page 13
 - [*Erlang Configuration Syntax*](#) on page 15
 - [*Enabling OF-Config in LINC*](#) on page 16
 - [*OF-Capable Switch Model*](#) on page 16
 - [*OF-Logical Switch \(OpenFlow Switch\)*](#) on page 16
- [*Examples of sys.config File in LINC Switches*](#) on page 20
 - [*One Switch with three Controllers*](#) on page 20
 - [*Two Switches with different Controllers*](#) on page 22
 - [*Two Switches with same Controllers*](#) on page 25
- [*OpenFlow Controllers*](#) on page 28
 - [*Ryu OpenFlow Controller*](#) on page 28
 - [*Warp OpenFlow Driver*](#) on page 28

INSTALLATION REQUIREMENTS

This section includes the hardware and software requirements for installing the LINC OpenFlow Switch.

Hardware Requirements

- Standard 64 bit, x86 CPU with atleast 4GB RAM and four network ports, capable of running Linux v. 3.1+ kernel.

Figure 2.1 Intel x86 based server



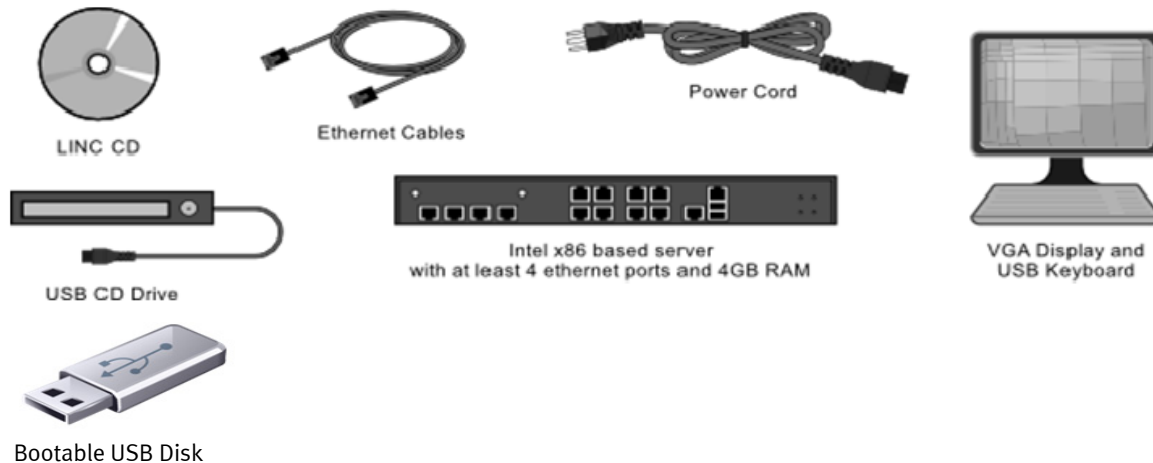
Alternatively, you can purchase off-the-shelf network white boxes from hardware vendors listed in <http://www.flowforwarding.org/hardware>.

Figure 2.2 Network White box



- Additional hardware requirements - USB keyboard, VGA monitor, Power cable, and Ethernet cable.

Figure 2.3 Additional Hardware Requirements





Software Requirements

- LINC runs on any system that can run Erlang OTP. For information about Erlang, refer to <http://www.erlang.org>.
- Bare metal fully bootable and installable USB and CD image files of LINC OpenFlow Switch with all the necessary software (around 460MB). You can download this from http://susestudio.com/a/ENQFFD/fllinc-1_2 as shown in [Figure 2.4](#).

Figure 2.4 Download ISO/USB image

Download



Downloaded 34 times Cloned 4 times

 Media
  Virtual

Preload ISO

A bootable ISO containing the appliance *disk image*. Boots into a simple graphical environment and prompts for confirmation before writing the image onto the hard drive. Useful for performing automated system installs/preloads.



Warning: Overwrites all data on the target hard drive.

 Preload ISO, 2.21 GB, x86_64
 MD5: 5e327f6bffb50d9f23c22c4f8498b734
 

Preload USB stick image

A bootable USB stick containing the appliance *disk image*. Boots into a simple graphical environment and prompts for confirmation before writing the image onto the hard drive. Useful for performing automated system installs/preloads.

Warning: Overwrites all data on the target hard drive.



 Preload USB Image, 719 MB, x86_64
 MD5: 25384467645eb67229b26658f81937fc
 

Additionally, an Open Virtual Format (OVF) file is available for the installation of LINC OpenFlow Switch on popular hypervisors as a virtual switch. You can download the OVF file from http://susestudio.com/a/ENQFFD/fllinc-1_2 as shown in [Figure 2.5](#).

Figure 2.5 Download OVF image



Download

Downloaded 34 times Cloned 4 times

 Media
  Virtual

Open Virtualization Format (OVF)

The **OVF format** is an open standard for packaging and distributing virtual appliances. It is not tied to any particular hypervisor or system architecture.

 OVF Image, 677 MB, x86_64
 MD5: 3306e3743c142ca76cf31afc06c43d65
 

For information about creating a bootable USB device from a USB image file, refer to [Creating a Bootable USB Device](#).

- **Serial Port Drivers.** Most of the network appliances do not have video cards. They allow the configuration via serial port console. Some systems have a serial (DB9) port and some have USB-to-UART Bridge. An example for such a device is a Silicon Labs CP2105. Before connecting the hardware system, the host computer has to install the drivers for CP2105.

To download and install the drivers, complete the following:

1. Visit <http://www.silabs.com/products/interface/usbtouart/Pages/usb-to-uart-bridge.aspx>.
2. Select the **Tools** tab, click **Virtual Com Port (VCP) Download**, and then download the drivers for your operating system.
3. After downloading, install the drivers based on the documentation provided by the vendor.
4. Use the provided USB Mini-B cable to connect the RCC system console to the host computer.
5. Verify whether the host computer can see the serial port.

You have to use a terminal emulator (such as PuTTY, MacOS X built-in Terminal or ZTerm) to connect to the consoles. Following is an example of a setting for such a terminal:

- Speed - 115,200
- Data Bits - 8
- Parity - None
- Stop Bits - 1
- Flow Control - None
- Preferred emulation mode is ANSI

CREATING A BOOTABLE USB DEVICE

You can write the raw image of LINC OpenFlow Switch to a USB device and you can boot from the USB device to install the LINC OpenFlow Switch.

To create a bootable USB device for the LINC OpenFlow Switch on a Linux or Mac OS X system, complete the following:

1. Download the USB install image of OpenFlow capable switch (v1.2/1.3.2) with OF-Config 1.1.1 support from http://susestudio.com/a/ENQFFD/fflinc-1_2.
2. Unzip/Untar the package using the following command:

```
# tar zxvf FFLINC1_2.x86_64-0.0.8.preload.raw.tar.gz
```

 This will yield `FFLINC1_2.x86_64-0.0.8.preload.raw` image file. This file is used to create a bootable USB on Linux systems.
3. Write the raw image of LINC OpenFlow Switch to a USB device. Refer to the following sections for instructions about how to write the raw image to a USB device:
 - [Writing a raw image to a USB device on Linux OS](#) on page 9
 - [Writing a raw image to a USB device on Mac OS X](#) on page 10

Writing a raw image to a USB device on Linux OS

The 'dd' way is used to write the raw image of LINC OpenFlow Switch to a USB device on a Linux system. Log in as user 'root' or 'sudo' and complete the following:

1. Insert a USB device to the system. Open a terminal and type the following command:

```
# df
```

You will see an output similar to the following:

```
/dev/sda2          30969600  15533336  13863100  53% /
udev              1997904      108    1997796   1% /dev
/dev/sda5          92888248  85548000   2621560  98% /home
/dev/sda6          23671572   935276   21533836   5% /var
/dev/sdb1          7816228     1492    7814736   1% /media/disk
```

The last entry should be the USB device which you have plugged in. To verify, you can remove the USB device and run the `df` command again and you will see that the line disappears. The left column in the `df` output is the partition, and the path upto the number is the path to the device. In the above example, `/dev/sdb1` is the partition, and `/dev/sdb` is the path to the device.

Note: It is important to get the device path correct, otherwise it may cause irreparable damage to your system.

2. After you get the device path, run the command `dd` to write your appliance to the USB device.

The command `dd` requires two arguments — the input file (your appliance) and the output file (the path to your USB device). You can run the following command from a terminal window:

```
sudo dd if=/home/suse/FFLINC1_2.x86_64-0.0.8.preload.raw of=/dev/sdb bs=4k
conv=fdatasync
```

In the above command, the input file is `/home/suse/FFLINC1_2.x86_64-0.0.8.preload.raw` and `/dev/sdb` is the path to the device. The argument `bs=4k` is optional, but adding this will write the appliance to a USB device much faster. The `conv=fdatasync` argument ensures that the appliance is fully written to the USB device prior to `dd` completion.

Note: This process will completely overwrite any previously stored data in the USB device. So make sure that you do not have any important data in the USB device.

Please note that writing data to a USB device is generally very slow. When `dd` has finished executing, it displays the statistics about how much data is written to the USB device.

3. If your USB device has a light on it, which blinks when the data is being written, wait until it stops blinking and then remove it.
4. As writing the data to a USB device block wise is a critical process, please compare the `md5sum` of the raw image and the newly created device, using the following command:

```
# md5sum /home/suse/FFLINC1_2.x86_64-0.0.8.preload.raw
# md5sum /dev/sdb
```

The custom software appliance is ready to be booted from your USB device.

Writing a raw image to a USB device on Mac OS X

To create a bootable USB device on a Mac operating system, open a terminal and execute the following commands as user 'root' or 'sudo':

```
# diskutil list
/dev/disk0
#:
```

	TYPE	NAME	SIZE	IDENTIFIER
0:	GUID_partition_scheme		*500.1 GB	disk0
1:	EFI		209.7 MB	disk0s1
2:	Apple_HFS	Macintosh HD	499.2 GB	disk0s2
3:	Apple_Boot	Recovery HD	650.0 MB	disk0s3

```
/dev/disk1
#:
```

	TYPE	NAME	SIZE	IDENTIFIER
0:	Apple_partition_scheme		*845.3 MB	disk1
1:	Apple_partition_map		30.7 KB	disk1s1
2:	Apple_Driver_ATAPI		2.0 KB	disk1s2
3:	Apple_HFS	MadRiver5M993.iPhone...	845.2 MB	disk1s3

```
/dev/disk2
#:
```

	TYPE	NAME	SIZE	IDENTIFIER
0:	FDisk_partition_scheme		*2.0 GB	disk2
1:	DOS_FAT_32	USB DISK	2.0 GB	disk2s1

```
# diskutil unmountDisk /dev/disk2
Unmount of all volumes on disk2 was successful
# sudo dd if=/Users/smysore/shiva/FFLINC1_2.x86_64-0.0.8.preload.raw of=/dev/disk2 bs=4k
210688+0 records in
210688+0 records out
862978048 bytes transferred in 152.044762 secs (5675816 bytes/sec)
# diskutil eject /dev/disk2
```

DEPLOYMENT ARCHITECTURE

The following sections describe the different methods of deploying the LINC OpenFlow Switch:

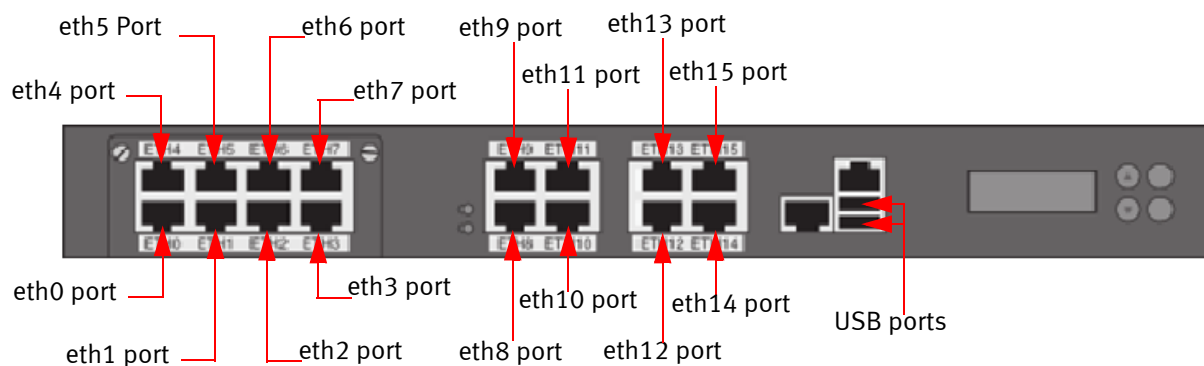
- [LINC on hardware](#)
- [LINC on VMware vSphere 5.x](#)

LINC on hardware

To install the LINC OpenFlow Switch on a hardware, complete the following:

1. Connect an Ethernet cable from a standard switch port to the `eth0` port (management port) of the hardware box so that it can get an IP address from the DHCP server running on the network.
2. Insert the CD or the USB device that contains the LINC OpenFlow Switch software (as appropriate) to the hardware box to install the LINC OpenFlow Switch software. For information about creating a bootable USB device for LINC OpenFlow Switch, refer to [Creating a Bootable USB Device](#) on page 8.
3. Connect the power cable to the hardware box, power it on and boot from the CD or the USB device to install the LINC OpenFlow Switch software.

Figure 2.6 Hardware box



LINC on Virtual Machine

LINC on VMware vSphere 5.x

To install the LINC OpenFlow Switch on VMware vSphere 5.x, complete the following:

1. Download the OVF image file from http://susestudio.com/a/ENQFFD/fflinc-1_2.
2. Deploy the OVF (Open Virtualization format) image file and follow the on-screen directions.
3. Add more NICs to the VM image by editing the VM properties. For information about VMware vSphere documentation, refer to <https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html>.

INSTALLING AND CONFIGURING THE LINC OPENFLOW SWITCH

To install and configure LINC OpenFlow Switch, complete the following:

1. Connect the cables, power it on and boot the hardware box with the USB device and follow the on-screen directions.

Note that the `eth0` port should be connected to a network that serves DHCP.

2. Log in to the LINC hardware box, using the following credentials:

```
username: root
password: linc
```

3. Capture the output of `'ifconfig -a'` command.

Note that if all interfaces are not `eth*`, then the script referred to in the next step has to be modified.

4. Create network interfaces using the following script:

```
/usr/local/src/linc/misc_tools/create_ifcfgeth_files
```

5. After creating the network interfaces, you have to bring each one up – `/sbin/ifup eth1; /sbin/ifup eth2` and so on. Alternatively, you can run the command `service network restart`.

6. To set up the LINC OpenFlow Switch software, run the following command:

```
# cd /usr/local/src/ofswitch/LINC-Switch
```

7. Set up the `sys.config` file using the following command:

```
# scripts/config_gen -s 0 eth1 eth2 eth3 -c tcp:127.0.0.1:6633 -o rel/files/sys.config
```

For information about the `config_gen` script, refer to [About the config_gen script](#) on page 13.

8. Run the command `# make offline` or `make rel`, if you are connected to the internet through the `eth0` port.

9. Test the installation of LINC OpenFlow Switch. For information about running the LINC OpenFlow Switch, refer to [Testing the Installation of LINC OpenFlow Switch](#) on page 12.

10. Set up Maven using the following steps:

// Need information on the role of Apache Maven software management tool in this installation and references for explanation. //

- Execute the following command:

```
# cd /usr/local/src/; ln -s apache-maven-3.1.1 maven;
```

Note that if you are able to run `'mvn -version'` after the next step, then the setup is working.

- You can run Warp using the following command:

```
# java -jar build/binaries/pre-built/warp.jar
```

For information about Warp OpenFlow Driver, refer to [Warp OpenFlow Driver](#) on page 28.

TESTING THE INSTALLATION OF LINC OPENFLOW SWITCH

You can run the following command from the LINC root directory (`/usr/local/src/ofswitch/LINC-Switch` directory) to test the installation of LINC OpenFlow Switch:

```
# rel/linc bin/linc start
```

This will start the LINC OpenFlow Switch in demon mode.

To stop the LINC OpenFlow Switch, execute the following command:

```
# rel/linc/bin/linc stop
```

ABOUT THE CONFIG_GEN SCRIPT

While the usage text only mentions about `eth` devices, you can also specify `tap` devices in the same way. The switch will handle each type of port appropriately.

The `config_gen` script supports configuring multiple logical switches as well as using `eth` interfaces instead of `tap` interfaces. The `-s` option can be specified more than once as follows:

```
scripts/config_gen -s 0 eth0 eth1 -s 1 eth2 -c tcp:127.0.0.1:6633 -o sys.config
```

This command will generate a configuration where logical switch 0 uses `eth0` and `eth1`, while logical switch 1 uses `eth2`. You can also use the `-c` option multiple times in a similar way, but it will not configure different controllers for each logical switch. Instead, each logical switch will use all the specified controllers when you run the command as follows:

```
scripts/config_gen -s 0 eth0 -s 1 eth1 -c tcp:10.48.11.5:6633 -c tls:10.48.11.6:5533 -o
/rel/files/sys.config
```

The following four connections will be established:

- From logical switch 0 to the first controller.
- From logical switch 0 to the second controller.
- From logical switch 1 to the first controller.
- From logical switch 1 to the second controller.

So you can modify the `config_gen` script such that the `-c` option is applied only to the switch specified by the previous `-s` option. Thus, the following command will give a configuration in which each logical switch has its own controller:

```
./config_gen -s 0 eth0 -c tcp:10.48.11.5:6633 -s 1 eth1 -c tls:10.48.11.6:5533 -o
../rel/files/sys.config
```

CONFIGURING SWITCH CAPABILITY

You can configure the switch capability (4-port, 8-port, or n-port) using the `sys.config` file. The switch configuration file is editable and is located in the `$LINC_ROOT/rel/linc/files/sys.config` folder. If you are building the release from the LINC source, the `sys.config` file is copied to the `$LINC_ROOT/rel/linc/releases//` folder, replacing all the changes that you have made in that folder.

The `sys.config` file uses the Erlang configuration syntax. For information, refer to [Erlang Configuration Syntax](#) on page 15. The following sections describe all the parameters that are configured in the `sys.config` file for LINC OpenFlow Switch:

- [Enabling OF-Config in LINC](#)
- [OF-Capable Switch Model](#)
- [OF-Logical Switch \(OpenFlow Switch\)](#)
- [TLS Configuration](#)
- [OF-Configuration Point via NetConf](#)
- [Error Logging Parameters](#)

The structure of the sys.config file for LINC OpenFlow Switch is as follows:

```
[
  {linc,
    [
      {of_config,enabled},
      {capable_switch_ports,
        [
          {port,1,
            [
              {interface,"en0"}
            ]},
          {port,2,
            [
              {interface,"en1"}
            ]},
          {port,3,
            [
              {interface,"en2"}
            ]}
        ]},
      {capable_switch_queues,[]},
      {logical_switches,
        [
          {switch,0,
            [
              {backend,linc_us4},
              {controllers,
                [
                  {"Switch0-Controller","127.0.0.1",6633,tcp}}],
              {controllers_listener,disabled},
              {queues_status,disabled},
              {ports,
                [
                  {port,1,{queues,[]}},
                  {port,2,{queues,[]}},
                  {port,3,{queues,[]}}
                ]}
            ]}
        ]}
      ],
    {enetconf,
      [
        {capabilities,
          [
            {base,{1,0}},
```



```

    {base,{1,1}},
    {startup,{1,0}},
    {'writable-running',{1,0}}},
    {callback_module,linc_ofconfig},
    {sshd_ip,any},
    {sshd_port,1830},
    {sshd_user_passwords,
    [
        {"linc","linc"}
    ]}
    ]},

{lager,
 [{handlers,
  [
    {lager_console_backend,debug},
    {lager_file_backend,
     [
      {"log/error.log",error,10485760,"$D0",5},
      {"log/console.log",info,10485760,"$D0",5}
     ]}
  ]}
 ]},
 {sasl,
  [
    {sasl_error_logger,{file,"log/sasl-error.log"}},
    {errlog_type,error},
    {error_logger_mf_dir,"log/sasl"},
    {error_logger_mf_maxbytes,10485760},
    {error_logger_mf_maxfiles,5}
  ]},
 {sync,
  [{excluded_modules,[procket]}]
 }}

```

Erlang Configuration Syntax

The configuration file of LINC OpenFlow Switch has the syntax in Erlang language. The configuration file should always end with a period. The terms used in the LINC configuration file may be tuples (contains any amount of terms wrapped in curly braces), lists (contains any amount of terms, wrapped in square braces), strings (characters wrapped in double quotes), numbers, and atoms (named lowercase constants without a specific numeric value).

Following are examples of the terms used in the LINC OpenFlow Switch configuration file:

String: "localhost".

Number: 1234, or 1.234 (floating point number).

Atom: this_is_atom.

Tuple: {ofs_port_no, 1}. Tuple contains a pair of atom 'ofs_port' and number 1. Tuples which has just 2 values may sometimes be called as 'pairs'.

List: [1, 2, 3] which has three numbers, or [{"localhost", 1234}] which has one tuple, that has a string and a number.

Enabling OF-Config in LINC

The `of_config` command enables the OF-Config subsystem which consists of three applications: `ssh`, `enetconf`, and `of_config`.

Allowed values: 'enabled' or 'disabled'.

Syntax

```
{of_config, enabled}
```

OF-Capable Switch Model

LINC is an OF-Capable switch. This means that each switch can host multiple logical switches at the same time and each logical switch can have its own controllers, ports, and queues.

Syntax

```
{logical_switches,
 [
  {switch, 0,
   [ ....
   ]},
  {switch, 1,
   [ ....
   ]},
  {switch, 2,
   [ ....
   ]}
 ]}
```

OF-Logical Switch (OpenFlow Switch)

The OF-Logical switch consists of several switch capability configuration parameters. You can configure each logical switch with its own controllers, ports, and queues.

Syntax

```
{switch, 0,
 [
  {backend, linc_us4},
  {controllers, []},
  {ports, []},
  {queues_status, disabled},
  {queues, []}
 ]}
```

Configuring Switch Backend Parameter

You can configure the switch backend implementation parameter used by `ofs_logic` process using the `'backend'` command. By default, the erlang userspace, OpenFlow version 1.3.1 implementation is selected.

Allowed values: `'linc_us3'` for OpenFlow v1.2 and `'linc_us4'` for OpenFlow v1.3.1.

Syntax

```
{backend, linc_us3},
```

Example

For OpenFlow v1.2:

```
{backend, linc_us3},
```

For OpenFlow v1.3.1:

```
{backend, linc_us4},
```

Configuring Controllers

This section contains the list of controllers to be used while switching traffic. Each value is a tuple containing pair of string, which should resolve to an IPv4 address and numeric port, where the controller is listening.

Ideally, this list should be empty and the assignment should be handled by an OF-Config client. The default OFP controller port is 6633.

Syntax

```
{controllers,
 [
   {"Switch0-DefaultController", "localhost", 6633, tcp}
 ]},
```

Configuring Input and Output Ports

This section contains the list of ingress (incoming) and egress (outgoing) ports, which are used for receiving and sending traffic through the switch. There is no configuration option to distinguish between the input and the output ports as they are the same. Each port definition is a property list, containing values of port number, interface name, queues definitions, and the port bandwidth (rate).

You can configure the available ports to the switch while using the userspace at the backend according to your system setup. For Linux systems, all TAP interfaces should be set up in advance as persistent. But for Mac OS X, the TAP interfaces are created during node startup (which requires setting an `ip` parameter).

The key `'port'` defines the logical port number, as it is seen in routing rules and log output. The key `'interface'` defines the system-wide name of the network interface to which the port is attached.

Note: A statement starting with `"%%"` in Erlang is a comment.

Syntax

```
{ports,
 [
   %% {port, 1, [{interface, "eth0"}]},
   %% {port, 2, [{interface, "tap0"}]},
   %% {port, 3, [{interface, "tap1"}, {ip, "10.0.0.1"}]}
 ]},
```

Configuring Queues

The queue configuration is not a part of the OpenFlow specification and should be considered as experimental. This feature is disabled by default.

Allowed value: 'enabled' or 'disabled'.

Syntax

```
{queues_status, disabled},
```

Example

Enable the queue configuration:

```
{queues_status, enabled}
```

Disable the queue configuration:

```
{queues_status, disabled}
```

If queue configuration is enabled, then assign them to ports and set the appropriate rates. The queue configuration is not part of the OpenFlow specification and should be considered as experimental.

Syntax

```
{queues,
 [
  {port, 1, [{port_rate, {100, kbps}},
    {port_queues, [
      {1, [{min_rate, 100}, {max_rate, 100}]},
      {2, [{min_rate, 100}, {max_rate, 100}]}
    ]}}},
  {port, 2, [{port_rate, {100, kbps}},
    {port_queues, [
      {1, [{min_rate, 100}, {max_rate, 100}]},
      {2, [{min_rate, 100}, {max_rate, 100}]}
    ]}}}
 ]}
}
```

TLS Configuration

This section configures the necessary certificates and keys required for setting up a TLS connection between a logical switch and a controller. Here, the switch certificate and the private RSA keys are stored. Values should be base64 encoded and DER encoded strings.

Syntax

```
{certificate, ""},
 {rsa_private_key, ""}
```

Example

```
// Example to be provided by Shivaram //
```

OF-Configuration Point via NetConf

This section configures the port parameters and the authentication credentials necessary for communicating with the OF-Capable switch via NetConf.

// Review Comment: Brief explanation of enetconf capabilities (base, startup, writable-running), what is the default value and possible options, what are they stand for //

For information about the NETCONF protocol refer to, http://www.netconfcentral.org/netconf_docs.

Syntax

```
{enetconf,
 [
  {capabilities, [{base, {1, 1}},
                  {startup, {1, 0}},
                  {'writable-running', {1, 0}}]},
  {callback_module, linc_ofconfig},
  {sshd_ip, any},
  {sshd_port, 1830},
  {sshd_user_passwords,
   [
    {"linc", "linc"}
   ]}
 ]},
```

When you set the `sshd_ip` (default value is `any`), `sshd_port` (default value is 1830), and `sshd_user_passwords` (default username: `linc`, default password: `linc`) to appropriate values, the LINC Switch will start an SSH server on the specified port and it will expect the clients to request the netconf subsystem. This can be done using the command line ssh client, by specifying the `-s` option and giving the subsystem name as the command to execute:

```
ssh -v -p 1830 -l linc -s 127.0.0.1 netconf
```

Error Logging Parameters

Lager is a logging framework for Erlang/OTP. You can configure the parameters for error logging. Note that if you have more debug output on console, then replace 'info' with 'debug' in the following section of the code.

Syntax

// Review Comment: Explanation for all parameters, as given for Sasl logging mechanism //

```
{lager,
 [
  {handlers,
   [
    {lager_console_backend, info},
    {lager_file_backend,
     [
      {"log/error.log", error, 10485760, "$D0", 5},
```

```

        {"log/console.log", info, 10485760, "$D0", 5}
    ]}
}
}},

```

Sasl is another logging mechanism that is built for Erlang.

Syntax

```

{sasl,
 [
  {sasl_error_logger, {file, "log/sasl-error.log"}},
  {errlog_type, error},
  {error_logger_mf_dir, "log/sasl"},           % Log directory
  {error_logger_mf_maxbytes, 10485760},       % 10 MB max file size
  {error_logger_mf_maxfiles, 5}               % 5 files max
 ]}

```

EXAMPLES OF SYS.CONFIG FILE IN LINC SWITCHES

The following sections include example of a `sys.config` file for a single logical switch and for two logical switches:

- [One Switch with three Controllers](#) on page 20
- [Two Switches with different Controllers](#) on page 22
- [Two Switches with same Controllers](#) on page 25

One Switch with three Controllers

You can run the `config_gen` script for one switch with three controllers, as follows:

```

# scripts/config_gen -s 0 eth1 eth2 eth3 -c tcp:127.0.0.1:6633 tcp:10.48.10.10:6644
tls:192.168.10.10:6655 -o rel/files/sys.config

```

The `sys.config` file for one logical switch with three controllers is as follows:

```

[
 {linc,
  [
   {of_config,enabled},
   {capable_switch_ports,
    [
     {port,1,
      [
       {interface,"eth1"}
      ]},
     {port,2,
      [
       {interface,"eth2"}
      ]},
     {port,3,
      [

```

```

        {interface,"eth3"}
    ]]
}},
{capable_switch_queues,[]},
{logical_switches,
 [
  {switch,0,
   [
    {backend,linc_us4},
    {controllers,
     [

```

// Review Comment: Is it ok to have the same name "Switch0-Controller" //

```

        {"Switch0-Controller","127.0.0.1",6633,tcp},
        {"Switch0-Controller","10.48.10.10",6644,tcp},
        {"Switch0-Controller","192.168.10.10",6655,tls}}},
{controllers_listener,disabled},
{queues_status,disabled},
{ports,
 [
  {port,1,{queues,[]}},
  {port,2,{queues,[]}},
  {port,3,{queues,[]}}
 ]}
]}
}},
}},

```

// Review Comment: Under enetconf module -> capabilities sub-module has 2 different bases {1,0} and {1.1}, it would be great to explain why in the Netconf section //

```

{enetconf,
 [
  {capabilities,
   [
    {base,{1,0}},
    {base,{1,1}},
    {startup,{1,0}},
    {'writable-running',{1,0}}}},
  {callback_module,linc_ofconfig},
  {sshd_ip,any},
  {sshd_port,1830},
  {sshd_user_passwords,
   [

```

```

        {"linc", "linc"}
    ]}
}},
{lager,
 [{handlers,
  [
    {lager_console_backend, debug},
    {lager_file_backend,
     [
      {"log/error.log", error, 10485760, "$D0", 5},
      {"log/console.log", info, 10485760, "$D0", 5}
     ]}
  ]}
 ]},
{sasl,
 [
  {sasl_error_logger, {file, "log/sasl-error.log"}},
  {errlog_type, error},
  {error_logger_mf_dir, "log/sasl"},
  {error_logger_mf_maxbytes, 10485760},
  {error_logger_mf_maxfiles, 5}
 ]},
{sync,
 [{excluded_modules, [procket]}]
}}

```

Two Switches with different Controllers

You can run the config_gen script for two switches with different controllers, as follows:

```
# scripts/config_gen -s 0 eth1 eth2 eth3 -c tcp:127.0.0.1:6633 -s 1 eth5 eth6 eth7
tcp:10.48.10.10:6644 tls:192.168.10.10:6655 -o rel/files/sys.config
```

The sys.config file for two switches with different controllers is as follows:

```

[
 {linc,
  [
    {of_config, enabled},
    {capable_switch_ports,
     [
      {port, 1,
       [
        {interface, "eth1"}
       ]},
      {port, 2,
       [
        {interface, "eth2"}
       ]},
     ]},
  ]},

```



```

    {port,3,
      [
        {interface,"eth3"}
      ]},
    {port,4,
      [
        {interface,"eth5"}
      ]},
    {port,5,
      [
        {interface,"eth6"}
      ]},
    {port,6,
      [
        {interface,"eth7"}
      ]},
    {port,7,
      [
        {interface,"tcp:10.48.10.10:6644"}
      ]},
    {port,8,
      [
        {interface,"tls:192.168.10.10:6655"}
      ]}
  ]},
  {capable_switch_queues,[]},
  {logical_switches,
    [
      {switch,1,
        [
          {backend,linc_us4},
          {controllers,
            [
              {"Switch1-Controller","127.0.0.1",6633,tcp}]},
          {controllers_listener,disabled},
          {queues_status,disabled},
          {ports,
            [
              {port,4,{queues,[]}},
              {port,5,{queues,[]}},
              {port,6,{queues,[]}},
              {port,7,{queues,[]}},
              {port,8,{queues,[]}}
            ]}
          ]}
    ]},

```

```

    {switch,0,
      [
        {backend,linc_us4},
        {controllers,
          [
            {"Switch0-Controller","127.0.0.1",6633,tcp}
          ]},
        {controllers_listener,disabled},
        {queues_status,disabled},
        {ports,
          [
            {port,1,{queues,[]}},
            {port,2,{queues,[]}},
            {port,3,{queues,[]}}
          ]}
        ]}
    ]},
    {enetconf,
      [
        {capabilities,
          [
            {base,{1,0}},
            {base,{1,1}},
            {startup,{1,0}},
            {'writable-running',{1,0}}]},
        {callback_module,linc_ofconfig},
        {sshd_ip,any},
        {sshd_port,1830},
        {sshd_user_passwords,
          [
            {"linc","linc"}
          ]}
        ]},
    {lager,
      [{handlers,
        [
          {lager_console_backend,debug},
          {lager_file_backend,
            [
              {"log/error.log",error,10485760,"$D0",5},
              {"log/console.log",info,10485760,"$D0",5}
            ]}
          ]}
        ]}
    ]},

```

```
{sasl,
 [
  {sasl_error_logger, {file, "log/sasl-error.log"}},
  {errlog_type, error},
  {error_logger_mf_dir, "log/sasl"},
  {error_logger_mf_maxbytes, 10485760},
  {error_logger_mf_maxfiles, 5}
 ]},
{sync,
 [{excluded_modules, [procket]}]
}}
```

Two Switches with same Controllers

You can run the `config_gen` script for two switches with the same controllers, as follows:

```
# scripts/config_gen -s 0 eth1 eth2 eth3 -s1 eth5 eth6 eth7 -c tcp:127.0.0.1:6633
tcp:10.48.10.10:6644 tls:192.168.10.10:6655 -o rel/files/sys.config
```

The `sys.config` file for two switches with the same controllers is as follows:

```
[
 {linc,
  [
   {of_config, enabled},
   {capable_switch_ports,
    [
     {port, 1,
      [
       {interface, "eth1"}
      ]},
     {port, 2,
      [
       {interface, "eth2"}
      ]},
     {port, 3,
      [
       {interface, "eth3"}
      ]},
     {port, 4,
      [
       {interface, "-s1"}
      ]},
     {port, 5,
      [
       {interface, "eth5"}
      ]},
     {port, 6,
      [
```

```

        {interface,"eth6"}
    }},
    {port,7,
    [
        {interface,"eth7"}
    ]}
    ]},
    {capable_switch_queues,[]},
    {logical_switches,
    [
        {switch,0,
        [
            {backend,linc_us4},
            {controllers,
            [
                {"Switch0-Controller","127.0.0.1",6633,tcp},
                {"Switch0-Controller","10.48.10.10",6644,tcp},
                {"Switch0-Controller","192.168.10.10",6655,tls}}}},
            {controllers_listener,disabled},
            {queues_status,disabled},
            {ports,
            [
                {port,1,{queues,[]}},
                {port,2,{queues,[]}},
                {port,3,{queues,[]}},
                {port,4,{queues,[]}},
                {port,5,{queues,[]}},
                {port,6,{queues,[]}},
                {port,7,{queues,[]}}
            ]}
        ]}
    ]},
    {enetconf,
    [
        {capabilities,
        [
            {base,{1,0}},
            {base,{1,1}},
            {startup,{1,0}},
            {'writable-running',{1,0}}}},
        {callback_module,linc_ofconfig},
        {sshd_ip,any},
        {sshd_port,1830},
        {sshd_user_passwords,

```

```

    [
        {"linc", "linc"}
    ]}
}},
{lager,
 [{handlers,
    [
        {lager_console_backend, debug},
        {lager_file_backend,
            [
                {"log/error.log", error, 10485760, "$D0", 5},
                {"log/console.log", info, 10485760, "$D0", 5}
            ]}
        ]}
    ]},
{sasl,
 [
    {sasl_error_logger, {file, "log/sasl-error.log"}},
    {errlog_type, error},
    {error_logger_mf_dir, "log/sasl"},
    {error_logger_mf_maxbytes, 10485760},
    {error_logger_mf_maxfiles, 5}
  ]},
{sync,
 [{excluded_modules, [procket]}]
}]

```

// Review Comment: "Two switches with same controllers": I can't see that there are two switches, there is only one //

OPENFLOW CONTROLLERS

An OpenFlow Controller is an application that manages flow control in a software-defined networking (SDN) environment. The following are the OpenFlow controllers which you can run with LINC OpenFlow Switch:

- [Ryu OpenFlow Controller](#)
- [Warp OpenFlow Driver](#)

Ryu OpenFlow Controller

Ryu is a component-based software defined networking framework. Ryu provides software components with well defined API that makes it easy to create new network management and control applications. Ryu supports various protocols for managing network devices such as OpenFlow, Netconf, OF-Config, and others. Ryu supports OpenFlow 1.0, 1.2, 1.3, and Nicira Extensions.

For information on Ryu – Architecture, Programming, and Usage, refer to the Ryu e-book available at <http://osrg.github.io/ryu-book/ja/html/>.

Warp OpenFlow Driver

Warp is an OpenFlow driver which consists of the following two main parts:

- OpenFlow Protocol Driver: It parses and encodes OpenFlow messages.
- OpenFlow Controller Driver: It handles connections with OpenFlow datapath elements.

The main requirements for developing the driver are rapid prototyping, implementation, and development for various versions of OpenFlow protocols.

This section includes the following information about Warp OpenFlow Driver:

- [Architecture and Implementation](#)
- [Using Warp](#) on page 29
- [Programming with Warp](#) on page 31
- [References for Warp](#) on page 36

Architecture and Implementation

The OpenFlow protocol driver is built on modified Apache Avro and it provides a clear API to parse OpenFlow messages. For information about Apache Avro, refer to <http://avro.apache.org/docs/current/>. Apache Avro provides a JSON-based language for protocol description. The usage of Avro to describe the protocol has made it easy to define, understand, and modify the protocol. It helps in easier implementation of the future versions of OpenFlow. Avro has the ability to parse protocols and update protocol description files in runtime, without code generation. The work done by the Avro and Hadoop community that has developed and tested Avro code is utilized and is built upon it by updating the Avro syntax to make it suitable for describing OpenFlow protocol.

The OpenFlow Controller driver is based on Akka. Akka is an open-source toolkit and runtime simplifying the construction of concurrent applications on the Java platform. For information about Akka, refer to <http://akka.io/>. Akka implements a simple and powerful message-driven actors model for low-latency multitask applications and is designed for high performance and scalability. Akka is used to implement message-driven multitask driver, supporting multiple connections with different OpenFlow datapath elements. Our driver provides a clear message-driven API.

Currently the drivers are developed in Java for good portability. Additionally, since Avro has python and C++ implementations, it is relatively straight forward to port the OpenFlow protocol driver to these languages. The OpenFlow protocol and the OpenFlow Controller drivers are separate functions, so it is easy to use the OpenFlow protocol driver alone inside other OpenFlow Controllers.

Using Warp

The following sections includes information about building and running a Warp OpenFlow Driver:

- [Building Warp](#)
- [Running Warp](#)

Building Warp

To build Warp OpenFlow Controller, navigate to the directory where you have unpacked the controller or cloned it from GitHub. For example, you can refer to it as `$WARP_ROOT` and then build Avro and Warp OpenFlow Controller using the following commands:

```
$ cd $WARP_ROOT/src/main/java/avro-trunk/lang/java/avro
$ mvn install -DskipTests
$ cd $WARP_ROOT
$ ant
```

The jar file of the Warp OpenFlow Controller is in the `$WARP_ROOT/build/lib_` directory.

Running Warp

You can run a simple REST API service to the controller and a simple learning switch application. They are both using the OF Java API.

1. You can use the following commands to run REST API:

```
$ java -jar build/lib/warp.jar
```

2. Use the following commands to run learning switch application:

```
$ java -cp build/lib/warp.jar org.flowforwarding.of.demo.Launcher
```

In both cases controllers are listening on TCP port 6633.

REST API Commands

You can install flows using the `curl` command. Following are some examples:

```
$ curl -d '{"switch":"00:0C:29:C9:8E:AE:00:00", "name":"flow-mod-1", "priority":"32768",
"ingress-port":"1", "active":"true", "apply-actions":"output=1"}'
http://localhost:8080/ff/of/controller/restapi
```

```
$ curl -d '{"switch":"00:0C:29:C9:8E:AE:00:00", "name":"flow-mod-1", "priority":"32768",
"ingress-port":"2", "active":"true"}' http://localhost:8080/ff/of/controller/restapi
```

```
$ curl -d '{"switch":"00:0C:29:AC:93:43:00:00", "name":"flow-mod-1", "priority":"32768",
"ether-type":"0x0800", "active":"true"}' http://localhost:8080/ff/of/controller/restapi
```

```
$ curl -d '{"switch":"00:0C:29:AC:93:43:00:00", "name":"flow-mod-1", "priority":"32768",
"ether-type":"0x0800", "dst-ip":"10.10.10.10", "active":"true"}'
http://localhost:8080/ff/of/controller/restapi
```

```
$ curl -d '{"switch":"00:90:FB:37:71:6E:00:00", "name":"flow-mod-1", "priority":"10",
"ingress-port":"5", "active":"true", "write-actions":"output=6"}'
http://localhost:8080/ff/of/controller/restapi
```

```
$ curl -d '{"switch":"00:90:FB:37:71:6E:00:00", "name":"flow-mod-1", "priority":"10",
"ingress-port":"6", "active":"true", "apply-actions":"output=5, output=6"}'
http://localhost:8080/ff/of/controller/restapi
```

Actions and Match Criteria

Table 2.1 Actions

Name	Description
apply-actions	
write-actions	
clear-actions	

Table 2.2 Match Criteria

Name	Description	Prerequisites
ingress-port	Ingress port. This may be a physical or switch-dened logical port	
in-phy-port		
metadata		
src-mac	Ethernet source address	
dst-mac	Ethernet destination address	
ether-type	Ethernet type of the OpenFlow packet payload	
vlan-vid		
vlan-priority		
ip-dscp		
ip-ecn		
protocol	IPv4 or IPv6 protocol number	
src-ip	IPv4 source address	
dst-ip	IPv4 destination address	
src-port	TCP source port	
dst-port	TCP destination port	
udp-src	UDP source port	
udp-dst	UDP destination port	
sctp-src		
sctp-dst		
icmpv4-type		
icmpv4-code		
arp-op		
arp-spa		
arp-tpa		
arp-sha		

Name	Description	Prerequisites
arp-tha		
ipv6-src	IPv6 source address	
ipv6-dst	IPv6 destination address	
ipv6-flabel		
icmpv6-type		
icmpv6-code		
ipv6-nd-sll		
ipv6-nd-tll		
mpls-label		
mpls-tc		
mpls-bos		
pbb-isid		
tunnel-id		
ipv6-exthdr		

To delete the flow from a switch with REST API, you can use the following command:

```
curl -X DELETE
```

Installing Flow

You can run the following command to install the flow:

```
curl -d '{"switch":"00:0C:29:BD:37:38:00:00", "name":"flow-mod-1", "priority":"32768",
"ingress-port":"2", "active":"true", "apply-actions":"output=3"}'
http://localhost:8080/ff/of/controller/restapi
```

Deleting Flow

You can run the following command to delete the flow:

```
curl -X DELETE -d '{"switch":"00:0C:29:BD:37:38:00:00", "name":"flow-mod-1",
"priority":"32768", "ingress-port":"2", "active":"true", "apply-actions":"output=3"}'
http://localhost:8080/ff/of/controller/restapi
```

Programming with Warp

The main idea here is to create an event-driven application. You must first implement your own event handler and launch the controller that is listening on a TCP port. When the controller connects with the switch and gets a version information from the Hello message, it starts an appropriate OpenFlow events handler, supporting a given OpenFlow version which deals with OpenFlow messages and generates events against developer-defined event handler.

OpenFlow protocol Java API

OpenFlow Message and Structure handlers

Use Message and Structure handlers

OpenFlow Message Provider

You may operate with OpenFlow protocol messages and structures via `MessageProvider` class as follows:

```
import org.flowforwarding.of.protocol.ofmessages.IOFMessageProvider;
import org.flowforwarding.of.protocol.ofmessages.IOFMessageProviderFactory;
import org.flowforwarding.of.protocol.ofmessages.OFMessageProviderFactoryAvroProtocol;

IOFMessageProviderFactory factory = new OFMessageProviderFactoryAvroProtocol();
IOFMessageProvider provider = factory.getMessageProvider("1.3");
```

You can use the provider to operate OpenFlow messages. Get messages in binary format ready to put into a channel:

```
byte [] helloMessage = provider.encodeHelloMessage();
/*...*/
byte [] configRequestMessage = provider.encodeSwitchConfigRequest();
/*...*/
byte[] switchFeatureRequestMessage = encodeSwitchFeaturesRequest();
```

To get more complicated messages first you have to build it, and then add fields via Ref classes. You can consider the Flow Modification message:

```
import org.flowforwarding.of.protocol.ofmessages.OFMessageFlowMod.OFMessageFlowModeRef;

OFMessageFlowModRef fmRef = provider.buildFlowMod();
fmRef.addTableId("1");
fmRef.addPriority("256");
/*...*/
fmRef.addMatchInPort("128");
/*...*/
OFStructureInstructionRef instrRef = provider.buildInstructionApplyActions();
instrRef.addActionOutput("12");
/*...*/
fmRef.addInstuction(instrRef);
byte [] fmBuffer = provider.encodeFlowMod(fmRef);
```

OpenFlow Controller Java API

The controller infrastructure is started in another way:

```
import org.flowforwarding.of.controller.Controller;
/*.....*/
Controller.launch (SessionHandler.class); // This launches a controller listening on tcp
port 6633
Controller.launch (SessionHandler.class, configuration); // This launches a controller
listening on the given tcp port
```

Configuration

The following shows a POJO that contains configuration information such as TCP port number:

```
import org.flowforwarding.of.controller.Configuration
/*.....*/
Configuration config1 = new Configuration();      // tcp port 6633 by default
Configuration config2 = new Configuration(6633);
```

Controller - Switch interaction

During the controller sessions, some events are generated. An application handles it via OF Events handler.

Switch Handler

The SwitchHandler class refers to a switch connected to the controller. It is unique and you have to use it to communicate with a switch, send commands or get a switch state.

Syntax:

```
SwitchRef switchRef = SwitchHandler.create();
Long SwitchRef.getDpid();
void SwitchRef.setDpid(Long dpid);
```

Code Example:

```
import org.flowforwarding.of.ofswitch.SwitchState.SwitchHandler;

/*.....*/
public class SimpleHandler extends OFSessionHandler {
    @Override
    protected void handshaked(SwitchRef switchRef) {
        super.handshaked(switchRef);
        Long dpid = switchRef.getDpid();
        /*.....*/
    }
    /*.....*/
}
```

Session handlers

You have to implement your session handler to interact with the controller. You may manage the incoming messages handling or send messages. Session handlers extend the class **OFSessionHandler**.

Syntax:

```
import org.flowforwarding.of.controller.SwitchState.SwitchRef;
import org.flowforwarding.of.controller.session.OFSessionHandler;
import
org.flowforwarding.of.controller.protocol.ofmessages.OFMessagePacketIn.OFMessagePacket
InRef;

public class SimpleHandler extends OFSessionHandler {
```

```

    /*
     * User-defined Switch event handlers
     */
    @Override
    protected void handshaked(SwitchRef switchRef) {
        super.handshaked(switchRef);
        /*You have to implement your own logic here*/
    }

    /*.....*/
    @Override
    protected void connected(SwitchRef switchRef) {
        super.connected(switchRef);
        /*You have to implement your own logic here*/
    }

    /*.....*/
    @Override
    protected void packetIn(SwitchRef switchRef, OFMessagePacketInRef packetIn) {
        super.packetIn(switchRef);
        /*You have to implement your own logic here*/
    }
}

```

Simple Learning Switch application

You can find a source code in the package `org.flowforwarding.of.demo`.

`org.flowforwarding.of.demo.Launcher.java`

```

package org.flowforwarding.of.demo;

import org.flowforwarding.of.controller.Controller;
import org.flowforwarding.of.controller.Controller.ControllerRef;

public class Launcher {
    public static void main(String[] args) {
        ControllerRef cRef = Controller.launch(SimpleHandler.class);
    }
}

```

`org.flowforwarding.of.demo`

```

package org.flowforwarding.of.demo;

import org.flowforwarding.of.controller.session.OFSessionHandler;
import org.flowforwarding.of.ofswitch.SwitchState.SwitchRef;
import org.flowforwarding.of.protocol.ofmessages.IOFMessageProvider;
import org.flowforwarding.of.protocol.ofmessages.OFMessageFlowMod.OFMessageFlowModRef;

```

```

import
org.flowforwarding.of.protocol.ofmessages.OFMessagePacketIn.OFMessagePacketInRef;
import
org.flowforwarding.of.protocol.ofmessages.OFMessageSwitchConfig.OFMessageSwitchConfigRef;
import
org.flowforwarding.of.protocol.ofstructures.OFStructureInstruction.OFStructureInstructionRef;

public class SimpleHandler extends OFSessionHandler {

    @Override
    protected void switchConfig(SwitchRef switchRef, OFMessageSwitchConfigRef configRef) {
        super.switchConfig(switchRef, configRef);

        System.out.print("[OF-INFO] DPID: " + Long.toHexString(switchRef.getDpid()) + "
Configuration: ");

        if (configRef.isFragDrop()) {
            System.out.println("Drop fragments");
        }
        if (configRef.isFragMask()) {
            System.out.println("Mask");
        }

        if (configRef.isFragNormal()) {
            System.out.println("Normal");
        }

        if (configRef.isFragReasm()) {
            System.out.println("Reassemble");
        }

    }

    @Override
    protected void handshaked(SwitchRef switchRef) {
        super.handshaked(switchRef);
        System.out.println("[OF-INFO] HANDSHAKED " +
Long.toHexString(switchRef.getDpid()));

        sendSwitchConfigRequest(switchRef);
    }

    @Override
    protected void packetIn(SwitchRef switchRef, OFMessagePacketInRef packetInRef) {
        super.packetIn(switchRef, packetIn);
    }

```

```

        IOFMessageProvider provider = switchRef.getProvider();

        OFMessageFlowModRef flowModRef = provider.buildFlowModMsg();

        if (packetInRef.existMatchInPort()) {
            flowModRef.addMatchInPort(packetInRef.getMatchInPort().getMatch());
        } else if (packetInRef.existMatchEthDst()) {
            flowModRef.addMatchEthDst(packetInRef.getMatchEthDst().getMatch());
        } else if (packetInRef.existMatchEthSrc()) {
            flowModRef.addMatchEthSrc(packetInRef.getMatchEthSrc().getMatch());
        }

        OFStructureInstructionRef instructionRef = provider.buildInstructionApplyActions();
        instructionRef.addActionOutput("2");
        flowModRef.addInstruction("apply_actions", instructionRef);

        sendFlowModMessage(switchRef, flowModRef);
    }
}

```

References for Warp

For latest information on Warp, refer to <https://github.com/FlowForwarding/warp>.

Running the LINC OpenFlow Switch

This chapter explains how to run the LINC OpenFlow Switch. It includes the following sections:

- [Viewing the LINC Switch Information](#) on page 38
- [Running Ping Demonstration](#) on page 38
 - [Setting up the environment](#) on page 38
 - [Ping Demonstration](#) on page 41
- [Running LINC OpenFlow Switch with Mininet](#) on page 43
 - [About Mininet](#) on page 43
 - [Installing Mininet](#) on page 43
 - [Integrating LINC Switch with Mininet](#) on page 43
 - [Configuring LINC with OF-Config](#) on page 45
- [Configuring LINC with OF-Config](#) on page 45
 - [Starting LINC without any Controller](#) on page 45
 - [Checking that there are no controllers configured](#) on page 46
 - [Adding a Controller via OF-Config](#) on page 53
 - [Checking whether a new controller is added](#) on page 54
- [Testing and Tools](#) on page 62
 - [Iperf](#) on page 62
 - [cURL](#) on page 62
 - [Wireshark](#) on page 62
- [Miscellaneous Tools](#) on page 64

VIEWING THE LINC SWITCH INFORMATION

You can run the following Erlang commands in the LINC Switch console and view the corresponding information:

Erlang Command	Description
<code>linc:lookup(0, flow_table_0).</code>	Look at switch 0 and table 0 to see what it contains (Switch ID, Table #)
<code>linc_logic:get_backend_capabilities(0).</code>	Return the switch capabilities (Switch ID)
<code>linc_logic:get_backend_flow_tables(0).</code>	Get a list of all backend end flow tables (Switch ID)
<code>linc_us4_flow:get_flow_table(0,0).</code>	Get all entries in switch 0 and one flow table (Switch ID, Table#)
<code>ets:tab2list(table_id).</code>	Convert an Erlang table to a list.
<code>linc_logic:get_datapath_id(0).</code>	View the data path ID for a switch.
<code>ets:tab2list(linc:lookup(0, group_table)).</code>	Check the group table, ETS table on switch 0.

RUNNING PING DEMONSTRATION

The following sections describes the ping demonstration:

- [Setting up the environment](#)
- [Ping Demonstration](#)

Setting up the environment

To run a ping (ICMP Echo Request) demonstration, you can setup the environment as follows:

1. To ensure that the project is compiled, run the following command:

```
% make
```

Prepare the components to be used in the ping demonstration. All the commands should be run from the root directory of the repository (\$HOME/openflow on Amazon).

2. Controller Mock: You have to first start a simple mock of the OpenFlow Controller. It will not identify the messages which it receives, but you can connect to the OpenFlow Controller from the switch and use it to send messages to modify the flow table entries.

- You can start the controller, using the following command:

```
% erl -pa apps/*/ebin deps/*/ebin
```

- Compile the controller, using the following command:

```
(controller)1> c("scripts/of_controller_v4.erl").
```

- You can load the records required to send OpenFlow protocol messages to the switch, using the following command:

```
(controller)2> rr(of_protocol), rr(ofp_v4).
```

- Start the controller on port 6633, using the following command:

```
(controller)3> {ok, CtrlPid} = of_controller_v4:start(6633).
```


3. Implementing erlang OpenFlow Switch: When the controller is running, start the implementation of erlang OpenFlow Switch.
 - You can edit the `rel/files/sys.config` file, which contains the switch configuration and then check if the controller (`localhost:6633`) and the two ports (`tap0` and `tap1`) are specified and uncommented.

Syntax

```
{linc,
 [
  {of_config, enabled},
  {capable_switch_ports,
   [
    {port, 1, [{interface, "tap0"}]},
    {port, 2, [{interface, "tap1"}]}
   ]},
  {capable_switch_queues, []},
  {logical_switches,
   [
    {switch, 0,
     [
      {backend, linc_us4},
      {controllers,
       [
        {"Switch0-DefaultController", "localhost", 6633, tcp}
       ]},
      {ports,
       [
        {ports,
         [
          {port, 1, {queues, []}},
          {port, 2, {queues, []}}
         ]}
        ]},
      {queues_status, disabled}
     ]}
    ]}
  ]}
...

```

- Build and run the switch using the following commands (The `sudo` command is required to create the ports):

```
% make rel
% sudo rel/linc/bin/linc console
```

- The switch automatically connects to the controller and the controller accepts the connection. The output will be as follows:

```
=INFO REPORT==== xx-Apr-2012::xx:xx:xx ===
Accepted connection from #Port<0.xxxx> {{127,0,0,1},xxxxx}
```

- After the connection is established, the controller sends test messages to the switch and its flow table is populated. To see these entries, execute the following command in the switch console:

```
(switch)1> ets:tab2list(linc:lookup(0, flow_table_0)).
```

- Additionally, the tap0 and tap1 interfaces are created in the system.

```
% ifconfig {tap0|tap1}
```

- The interface connections must be up, using the following commands:

```
% sudo ifconfig tap0 up
```

```
% sudo ifconfig tap1 up
```

- The erlang representation of these ports are stored in the linc_ports ets table.

```
(switch)2> ets:tab2list(linc:lookup(0, linc_ports)).
```

- As the controller populates the flow table of the switch with the flows, you have to clear them before the ping demonstration. To clear the flow table, create an appropriate flow_mod message as follows:

```
(controller)4> ClearFlowTable = #ofp_message{
    version = 4,
    xid = 100,
    body = #ofp_flow_mod{
        cookie = <<0:64>>,
        cookie_mask = <<0:64>>,
        table_id = 0,
        command = delete,
        idle_timeout = 30000,
        hard_timeout = 60000,
        priority = 1,
        buffer_id = 1,
        out_port = any,
        out_group = any,
        flags = [],
        match = #ofp_match{fields = []},
        instructions = []}}.
```

- Query the switch connection and send a message to the switch using the following command:

```
(controller)5> {ok, [Conn|_]} = of_controller_v4:get_connections(CtrlPid).
```

```
(controller)6> of_controller_v4:send(CtrlPid, Conn, ClearFlowTable).
```

- To see if the above commands are executed, you can check the flow table of the switch again using the following command:

```
(switch)3> ets:tab2list(linc:lookup(0, flow_table_0)).
```

The above invocation should return an empty list.

4. Receiving Port - On the receiving port (tap1), run tcpdump command to watch for pong (ICMP Echo Reply), as follows:

```
% sudo tcpdump -v -i tap1
```

Ping Demonstration

To run a ping demonstration, complete the following:

1. Dropping the packet: When there are no inputs from the controller, all the packets received by the switch are dropped as the first table has no flow entries to match. When you send a single ping on `tap0` using `tcpdump` as follows, you will not receive any output on the receiving (`tap1`) port.

```
% sudo tcpdump -i tap0 pcap.data/ping.pcap
```

2. Adding a flow and forwarding the packet: Complete the following to add a flow and forward the packet:
 - Create a `flow_mod` message containing a match field (match on packets received from port 1) and an action (send matched packets to port 2), as follows:

```
(controller)7> FlowMod = #ofp_message{
    version = 4,
    xid = 100,
    body = #ofp_flow_mod{
        cookie = <<0:64>>,
        cookie_mask = <<0:64>>,
        table_id = 0,
        command = add,
        idle_timeout = 30000,
        hard_timeout = 60000,
        priority = 1,
        buffer_id = 1,
        out_port = 2,
        out_group = 5,
        flags = [],
        match = #ofp_match{
            fields = [#ofp_field{
                class = openflow_basic,
                name = in_port,
                has_mask = false,
                value = <<1:32>>}}],
            instructions = [#ofp_instruction_write_actions{
                actions = [#ofp_action_output{
                    port = 2,
                    max_len = 64}}]]}}.
```

- Query the connection of the switch, using the following command:

```
(controller)8> {ok, [Conn|_]} = of_controller_v4:get_connections(CtrlPid).
```

- Send it to the switch using the following command:

```
(controller)9> of_controller_v4:send(CtrlPid, Conn, FlowMod).
```

- You can check the flow table 0, to see if it contains one flow entry received from the controller, using the following command:

```
(switch)4> ets:tab2list(linc:lookup(0, flow_table_0)).
[{flow_entry,1,
    {ofp_match,
```

```

        [{ofp_field, openflow_basic, in_port, false,
          <<0,0,0,1>>,
          undefined}]],
      [{ofp_instruction_write_actions, [{ofp_action_output, 14, 2, 64}]]}]
    ]]

```

- You can send the same ping packet using the `tcpreplay` command as follows:

```
% sudo tcpreplay -i tap0 pcap.data/ping.pcap
```

It will match with the added flow entry and is forwarded to port 2. The output of `tcpdump` will be as follows:

```

xx:xx:xx:xxxxxx IP (tos 0x0, ttl 64, id 39324, offset 0, flags [none], proto ICMP
(1), length 84, bad cksum 0 (->72be!))

ip-10-152-0-118.ec2.internal > sg1.any.onet.pl: ICMP echo request, id 12814, seq 0,
length 64

```

3. Removing all flows and dropping the packet: Complete the following to remove all flows and dropping the packets:

- Create another `flow_mod` message to remove all the flow entries from the flow table 0, as follows:

```

(controller)10> RemoveFlows = #ofp_message{
    version = 4,
    xid = 200,
    body = #ofp_flow_mod{
        cookie = <<0:64>>,
        cookie_mask = <<0:64>>,
        table_id = 0,
        command = delete,
        idle_timeout = 30000,
        hard_timeout = 60000,
        priority = 1,
        buffer_id = 1,
        out_port = 2,
        out_group = 5,
        flags = [],
        match = #ofp_match{fields = []},
        instructions = []}.

```

- Send it to the switch using the following command:

```
(controller)11> of_controller_v4:send(CtrlPid, Conn, RemoveFlows).
```

- You can check whether the flow entry which you added earlier is removed from the flow table 0, using the following command:

```
(switch)5> ets:tab2list(linc:lookup(0, flow_table_0)).
```

- You can send the same ping packet using `tcpreplay` command as follows:

```
% tcpreplay -i tap0 pcap.data/ping.pcap
```

You will not receive any output for `tcpdump` on port 2.

RUNNING LINC OPENFLOW SWITCH WITH MININET

LINC Switch is shipped with Mininet and it can be run as a part of its virtual network. The aim of this integration is to provide an easy setup environment for testing different scenarios with LINC Switch in the lead role. In particular, you can use Mininet to create topologies through the python API. It is convenient to use python scripts to exchange topologies and scenarios. Additionally, Mininet has a power CLI that allows you to quickly run some simple test cases. This section gives you an overview of Mininet and describes how you can install Mininet, integrate LINC Switch with Mininet and configure LINC Switch with OF-Config. It includes the following sections:

- [About Mininet](#)
- [Installing Mininet](#)
- [Integrating LINC Switch with Mininet](#) on page 43
- [Configuring LINC with OF-Config](#) on page 45

About Mininet

Mininet is a tool that facilitates the creation of realistic virtual networks. Mininet emulates a complete network of hosts, links, and switches on a single machine. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves like a real machine; you can `ssh` into it (if you start up `sshd` and bridge the network to your host) and run arbitrary programs. The programs you run can send packets through, like a real ethernet interface with a given link speed and delay. Packets get processed like a real ethernet switch, router, or middle box, with a given amount of queueing. When two programs, such as `iperf` client and server, communicate through Mininet, the measured performance will match that of two (slower) native machines.

Installing Mininet

To install Mininet with LINC Switch, complete the following:

1. Clone the repository from <https://github.com/FlowForwarding/mininet>.
2. Enter mininet directory, and run the following command:

```
util/install.sh -3nfxL
```

3. Optionally, you can provide a revision to checkout:

```
LINC_SWITCH_REV="issue129" util/install.sh -3nfxL
```

This will install Mininet core, NOX 1.3 Controller, OpenFlow 1.3 Software Switch and the following dependencies:

- `tunctl` (from `uml-utilities` package)
- `brctl` (from `bridge-utils` package)
- `erlang`
- `git-core`

Integrating LINC Switch with Mininet

The following sections describe the process of integrating the LINC Switch with Mininet:

- [Ping](#)
- [Connecting to a Remote Controller](#)
- [Creating Advanced Topologies](#)

Ping

To warm up with the Mininet, run a ping example with the LINC Switch connected to a Mininet network that is governed by a controller.

Follow the steps given below:

1. Start the Mininet with a LINC Switch, two hosts and a remote controller, using the following command:

```
sudo bin/mn --controller=remote --switch=linc
```

2. In another console, attach the LINC Switch console to see whether it works, using the following command:

```
sudo linc attach
```

3. In another console, run the controller using the following commands:

```
cd LINC-Switch/scripts
```

```
sudo ./of_controller_v4.sh -p 6633 -d -s table_miss
```

The controller will connect to the switch and it sends a flow modification message to the switch, which makes the switch send all unmatched packets to the controller.

4. You can use the Mininet CLI to send a ping from one host to the other. Run the following command:

```
h1 ping -c 3 h2
```

Optionally, you can install Wireshark with OpenFlow 1.3 dissector and observe the OpenFlow protocol messages. To install Wireshark with OpenFlow 1.3 dissector, refer to <https://github.com/CPqD/ofdissector>.

Connecting to a Remote Controller

You can use other controllers to experiment with LINC Switch on Mininet.

1. To connect the LINC Switch to a remote controller use the following syntax:

```
sudo bin/mn --controller=remote,ip=<CONTROLLER IP>,&port=<CONTROLLER PORT> --switch=linc
```

For example, you can utilize NOX 1.3 Controller that is shipped with Mininet.

2. You can run NOX with the switch at the backend. To achieve this setup, run NOX Controller using the following command. Note that NOX is installed in the same directory as Mininet, by default.

```
cd nox13oflib/build/src && sudo ./nox_core -i ptcp:6655 switch
```

3. You can start Mininet with LINC Switch using the following command. In the following command, it is assumed that the NOX Controller is running on a 10.0.0.23 machine:

```
sudo bin/mn --controller=remote,ip=10.0.0.23,port=6655 --switch=remote
```

4. Repeat step 1 and step 2 from the [Ping](#) section and then try to ping between the two hosts.

Note that by default a switch started by Mininet tries to connect to a controller that listens on 127.0.0.1:6633.

Creating Advanced Topologies

Mininet allows you to create more complex topologies through the python API. You can setup a topology with two directly connected LINC Switches and a host for each switch, `host --- switch --- switch --- host`.

To get started, complete the following:

1. Start the Mininet with LINC Switch and the custom topology, using the following command:

```
sudo bin/mn --controller=remote --switch=linc --custom custom/topo-3sw-2host.py --topo mytopo
```

2. In another console, run the controller with a prepared scenario, using the following commands:

```
cd LINC-Switch/scripts
```

```
sudo ./of_controller_v4.sh -p 6633 -d -s mininet_mytopo
```

3. You can ping the hosts from the Mininet CLI, using the following command:

```
pingall
```

Optionally, you can execute the following commands in separate terminals to attach to the LINC Switch consoles:

```
sudo linc s3 attach
```

```
sudo linc s4 attach
```

CONFIGURING LINC WITH OF-CONFIG

You can configure the LINC Switch with OF-Config. The following sections describe the OF-Config demonstration:

- [Starting LINC without any Controller](#)
- [Checking that there are no controllers configured](#) on page 46
- [Adding a Controller via OF-Config](#) on page 53
- [Checking whether a new controller is added](#) on page 54

Note that while the default port for NETCONF over SSH is 830, in this example we use 1830 to get around the need for root privileges to use a port number below 1024.

Starting LINC without any Controller

1. There are no controllers configured in the default sys.config file for the LINC Switch. To start the LINC Switch without any controllers, complete the following:

- Run the following commands:

```
$ cd $LINC_ROOT
$ make
$ ./rel/linc/bin/linc console
```

However, if you are using the LINC environment, you have to modify the configuration file after running `ping_example setup`. Open the `/home/vagrant/development/linc/rel/linc/releases/1.0/sys.config` file and change the following line:

```
{controllers, [{"Switch0-Controller", "localhost", 6633, tcp}]},
to
```

```
{controllers, []},
```

- Start the LINC Switch with the following command:

```
ping_example switch
```

2. Start the built-in erlang controller of LINC Switch. You can start the controller either before or after adding it to the LINC configuration. It is recommended that you start the controller before adding it to the LINC configuration using the following commands:

```
$ cd $LINC_ROOT/scripts
$ ./of_controller_v4.sh [Note: use this for running it against OF 1.3.1 Switch]
Erlang R15B02 (erts-5.9.2) [source] [64-bit] [smp:4:4] [async-threads:0]
[kernel-poll:false]
```

```
Eshell V5.9.2 (abort with ^G)
1>
```

3. To use the Ryu Controller, run the following commands:

```
$ $RYU_ROOT/bin/ryu-manager --verbose $LINC_ROOT/scripts/ryu/l2_switch_v1_3.py
```

4. Connect with the `enetconf_client` using the following command:

```
$ cd $LINC_ROOT/deps/enetconf
$ make run_client

1> {ok, C} = enetconf_client:connect("localhost", [{port, 1830}, {user, "linc"},
{password, "linc"}]).

{ok, <0.XXX.0>}
```

5. Connect with Yuma. After building and installing Yuma, a few different applications will be available through the command line. One of the applications is yangcli, which is an interactive client for Netconf. Create a bash script file that will run in yangcli. Run this script to connect to the enetconf or other Netconf server, exchange packets and wait for interactive input. For more information about yangcli commands and syntax, refer to <http://doc.yumaworks.com/manuals/v2/html/yangcli/yuma-yangcli-manual3.xhtml>. Run the following command:

```
#!/bin/bash
yangcli --server=localhost --ncport=1830 --user=guest --password=guest
```

// **Review Comment:** YUMA project went proprietary, and I suggest to remove broken references, and include different projects like OpenYUMA or YUMAPro: <https://github.com/OpenClovis/OpenYuma>
<https://www.yumaworks.com/overview/yumapro/> //

// Shall I remove this section and add reference to OpenYuma/YumaPro //

Checking that there are no controllers configured

You can check that there are no controllers configured with OF-Config using the following methods:

- [Using enetconf_client](#)
- [Using Yuma](#)
- [Directly on LINC](#)

Using enetconf_client

You can run the following commands to check if there are no controllers configured:

```
2> {ok, X} = enetconf_client:get_config(C, running).
{ok, <<"<?xml ... ">>}}
3> io:format("~p~n", [X]).
<<"<?xml ... <rpc-reply ... <controllers/> ... ">>
```

//Review comment: It is not clear how to run the comments //

You will receive an output similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="0">
  <data>
    <capable-switch xmlns="urn:onf:of111:config:yang">
      <id>CapableSwitch0</id>
      <resources>
        <port>
          <resource-id>LogicalSwitch0-Port7</resource-id>
          <configuration>
            <admin-state>up</admin-state>
            <no-receive>false</no-receive>
            <no-forward>false</no-forward>
            <no-packet-in>false</no-packet-in>
          </configuration>
          <features>
            <advertised>
```



```

        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
    </advertised>
</features>
</port>
<port>
    <resource-id>LogicalSwitch0-Port6</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch0-Port4</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch0-Port1</resource-id>
    <configuration>
        <admin-state>up</admin-state>

```

```

        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch0-Port2</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch0-Port3</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>

```

```

    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch0-Port5</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>false</no-receive>
      <no-forward>false</no-forward>
      <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch1-Port7</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>false</no-receive>
      <no-forward>false</no-forward>
      <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch1-Port6</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>false</no-receive>
      <no-forward>false</no-forward>
      <no-packet-in>false</no-packet-in>
    </configuration>
    <features>

```

```

        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port4</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port1</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port8</resource-id>
    <configuration>

```

```

        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port2</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port3</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>

```

```

        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port5</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
</resources>
<logical-switches>
    <switch>
        <id>LogicalSwitch0</id>
        <datapath-id>7A:AC:DF:F0:3A:F2:00:00</datapath-id>
        <controllers>
            <controller>
                <id>Switch0-DefaultController</id>
                <ip-address>10.48.11.5</ip-address>
                <port>6633</port>
                <protocol>tcp</protocol>
            </controller>
        </controllers>
    </switch>
    <switch>
        <id>LogicalSwitch1</id>
        <datapath-id>7A:AC:DF:F0:3A:F2:00:01</datapath-id>
        <controllers />
    </switch>
</logical-switches>
</capable-switch>
</data>
</rpc-reply>

```

Using Yuma

You can connect to yangcli and check that there are no controllers configured with OF-Config by running the following command in yangcli:

```
get-config source=running
```

// Shall I remove this section and add reference to OpenYuma/YumaPro //

Directly on LINC

You can check directly on the LINC Switch that there are no controllers configured with OF-Config, using the following commands:

```
1> rr(of_config).
[capabilities, capable_switch ...]
2> supervisor:which_children(linc:lookup(0, channel_sup)).
[]
3> linc_ofconfig:get_switch_state(0).
[[], [#logical_switch{... controllers = [], ...
```

//Review comment: It is not clear how to run the comments //

Adding a Controller via OF-Config

We can add a new controller to the LINC configuration using the `edit-config` operation. You can add a controller using the following methods:

- [Using `enetconf_client`](#)
- [Using Yuma](#)

Using `enetconf_client`

You can run the following commands to add a controller using `enetconf_client`:

```
4> Controller = {controller, [{id, ["Controller0"]},
                             {'ip-address', ["127.0.0.1"]},
                             {port, ["6633"]}, {protocol, ["tcp"]}]],
  Config = {'capable-switch', [{xmlns, "urn:onf:of111:config:yang"}],
            [{id, ["CapableSwitch0"]}, {'logical-switches',
            [{'switch', [{id, ["LogicalSwitch0"]}, {'datapath-id',
            ["11:11:11:11:11:11:11:11"]},
            {enabled, ["true"]}, {controllers, [Controller]}]}]}]}.
{'capable-switch', ...
5> {ok, Y} = enetconf_client:edit_config(C, running, {xml, Config}).
{ok, <<"<?xml ... <rpc-reply ...
```

//Review comment: It is not clear how to run the comments //

Using Yuma

Yuma is capable of complex commands such as calling `get/edit-config` and operations on XML subtrees. The simplest operation without manually selecting the subtrees and keys is `edit-config`. You can replace the whole config with the file contents.

Prepare a configuration in XML file, say `config1.xml` and run the following commands:

```
edit-config target=running config=@config1.xml
```

or a complex example as follows:

```
edit-config target=candidate default-operation=merge test-option=test
error-option=stop-on-error\
    config=@config1.xml
```

// Shall I remove this section and add reference to OpenYuma/YumaPro //

Example of config.xml configuration file

Checking whether a new controller is added

You can check on the **controller side**, whether the LINC Switch is connected to the controller as follows:

```
XX:XX:XX.XXX [info] Accepted connection from #Port<0.XXXX> {{127,0,0,1}, XXXXX}
```

// Review comment: line that is presented as a command is not a command, but just a info line, and it seems that it's a LINC's [info] message, and not a controller side message //

// Should I change Controller side to LINC side? But in

<https://github.com/FlowForwarding/LINC-Switch/wiki/OF-Config-Demo>, it is mentioned as controller side //

You can use the `get-config` operation and check whether the controller is added in the following ways:

- [Using `enetconf_client`](#) on page 54
- [Using Yuma](#) on page 61
- [Directly on LINC](#) on page 61

Using `enetconf_client`

You can run the following command:

```
6> {ok, Z} = enetconf_client:get_config(C, running).
{ok, <<"<?xml ... ">>}}
7> io:format("~p~n", [Z]).
<<"<?xml ... <rpc-reply ... <controllers><controller ... </controllers> ... ">>
```

The formatted output will be as follows. Note that a controller is added to the logical switch1 but the data path ID & the IP address of the Controller in the following example is different from the above used data path id of 11:11:....11 and the controller IP of 127.0.0.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
```



```

<data>
  <capable-switch xmlns="urn:onf:of111:config:yang">
    <id>CapableSwitch0</id>
    <resources>
      <port>
        <resource-id>LogicalSwitch0-Port7</resource-id>
        <configuration>
          <admin-state>up</admin-state>
          <no-receive>false</no-receive>
          <no-forward>false</no-forward>
          <no-packet-in>false</no-packet-in>
        </configuration>
        <features>
          <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
          </advertised>
        </features>
      </port>
      <port>
        <resource-id>LogicalSwitch0-Port6</resource-id>
        <configuration>
          <admin-state>up</admin-state>
          <no-receive>false</no-receive>
          <no-forward>false</no-forward>
          <no-packet-in>false</no-packet-in>
        </configuration>
        <features>
          <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
          </advertised>
        </features>
      </port>
      <port>
        <resource-id>LogicalSwitch0-Port4</resource-id>
        <configuration>
          <admin-state>up</admin-state>
          <no-receive>false</no-receive>
          <no-forward>false</no-forward>
          <no-packet-in>false</no-packet-in>

```

```

    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch0-Port1</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>false</no-receive>
      <no-forward>false</no-forward>
      <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch0-Port2</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>false</no-receive>
      <no-forward>false</no-forward>
      <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
</port>

```

```

    <resource-id>LogicalSwitch0-Port3</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>>false</no-receive>
      <no-forward>>false</no-forward>
      <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch0-Port5</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>>false</no-receive>
      <no-forward>>false</no-forward>
      <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>>true</auto-negotiate>
        <medium>copper</medium>
        <pause>unsupported</pause>
      </advertised>
    </features>
  </port>
  <port>
    <resource-id>LogicalSwitch1-Port7</resource-id>
    <configuration>
      <admin-state>up</admin-state>
      <no-receive>>false</no-receive>
      <no-forward>>false</no-forward>
      <no-packet-in>>false</no-packet-in>
    </configuration>
    <features>
      <advertised>
        <rate>invalid</rate>
        <auto-negotiate>>true</auto-negotiate>

```

```

        <medium>copper</medium>
        <pause>unsupported</pause>
    </advertised>
</features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port6</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port4</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port1</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>

```

```

        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port8</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>
<port>
    <resource-id>LogicalSwitch1-Port2</resource-id>
    <configuration>
        <admin-state>up</admin-state>
        <no-receive>false</no-receive>
        <no-forward>false</no-forward>
        <no-packet-in>false</no-packet-in>
    </configuration>
    <features>
        <advertised>
            <rate>invalid</rate>
            <auto-negotiate>true</auto-negotiate>
            <medium>copper</medium>
            <pause>unsupported</pause>
        </advertised>
    </features>
</port>

```

```

    <port>
      <resource-id>LogicalSwitch1-Port3</resource-id>
      <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
      </configuration>
      <features>
        <advertised>
          <rate>invalid</rate>
          <auto-negotiate>>true</auto-negotiate>
          <medium>copper</medium>
          <pause>unsupported</pause>
        </advertised>
      </features>
    </port>
    <port>
      <resource-id>LogicalSwitch1-Port5</resource-id>
      <configuration>
        <admin-state>up</admin-state>
        <no-receive>>false</no-receive>
        <no-forward>>false</no-forward>
        <no-packet-in>>false</no-packet-in>
      </configuration>
      <features>
        <advertised>
          <rate>invalid</rate>
          <auto-negotiate>>true</auto-negotiate>
          <medium>copper</medium>
          <pause>unsupported</pause>
        </advertised>
      </features>
    </port>
  </resources>
  <logical-switches>
    <switch>
      <id>LogicalSwitch0</id>
      <datapath-id>7A:AC:DF:F0:3A:F2:00:00</datapath-id>
      <controllers>
        <controller>
          <id>Switch0-DefaultController</id>
          <ip-address>10.48.11.5</ip-address>
          <port>6633</port>
          <protocol>tcp</protocol>

```

```

        </controller>
    </controllers>
</switch>
<switch>
    <id>LogicalSwitch1</id>
    <datapath-id>7A:AC:DF:F0:3A:F2:00:01</datapath-id>
    <controllers>
        <controller>
            <id>Controller0</id>
            <ip-address>10.48.11.5</ip-address>
            <port>6633</port>
            <protocol>tcp</protocol>
        </controller>
    </controllers>
</switch>
</logical-switches>
</capable-switch>
</data>
</rpc-reply>

```

Using Yuma

Run the following command in yangcli console:

```
get-config source=running
```

// Shall I remove this section and add reference to OpenYuma/YumaPro //

Directly on LINC

You can check directly on the LINC Switch whether it is connected to the controller, using the following commands:

```

4> supervisor:which_children(linc:lookup(0, channel_sup)).
[{'127.0.0.1_6633', <0.XXX.0>, worker, [ofs_receiver]]]
5> linc_ofconfig:get_switch_state(0).
{[], [#logical_switch{... controllers = [#controller{ ... }], ...

```

TESTING AND TOOLS

This section describes how to test LINC Switch and the tools you can use for testing.

- Nightly test result report: The test runs automatically whenever there is a check in. The LINC Switch GitHub repository is integrated with Travis.ci, a service that executes all the built-in tests. The report is available at <https://travis-ci.org/FlowForwarding/LINC-Switch>.
- Traffic Generation: You can measure traffic generation using IPerf. For information about IPerf, refer to [Iperf](#) on page 62.
- cURL, a command line tool is used to transfer the data with URL syntax. For information about cURL, refer to [cURL](#) on page 62.
- OpenFlow Wireshark Dissector. For information about Wireshark, refer to [Wireshark](#) on page 62.

Iperf

IPerf is a TCP and UDP bandwidth performance measurement tool. IPerf was originally developed by NLANR/DAST as a tool for measuring maximum TCP and UDP bandwidth performance. IPerf allows the tuning of various parameters and UDP characteristics. IPerf reports bandwidth, delay jitter, and datagram loss.

IPerf3 is the new implementation from scratch, with the goal of a smaller, simpler code base, and a library version of the functionality that can be used in other programs. IPerf3 has a number of new features that are found in other tools such as nuttcp and netperf.

The following are some of the new features in IPerf3:

- Reports the number of TCP packets that are retransmitted.
- Reports the average CPU utilization of the client and server (-V flag).
- Supports the zero copy TCP (-Z flag).
- JSON output format (-J flag).

For information on jPerf, refer to <https://code.google.com/p/xjperf/>.

Usage Example

You can refer to <http://iperf.fr/>.

cURL

cURL is a command line tool used for transferring data with URL syntax. It supports DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet, and TFTP. cURL supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user and password authentication (Basic, Digest, NTLM, Negotiate, kerberos, etc.), file transfer resume, proxy tunneling and a busload of other useful tricks.

Usage Example

TBD.

Wireshark

The OpenFlow Wireshark Dissector is a Wireshark plugin that dissects the OpenFlow protocol.

Installing Wireshark

You can download the latest stable release of Wireshark from <http://www.wireshark.org/download.html>.

OpenFlow Dissector

Wireshark v1.12 is the first stable version to support the OpenFlow protocol, but it is only expected to be released in June 2014. Currently, you can use v1.11.2+ development release to test or use it.

To check whether Wireshark version supports dissection of OpenFlow packets, you can check the output of the following command:

```
$ tshark -G protocols | grep -i openflow
OpenFlow          openflow          openflow
OpenFlow 1.0      openflow_v1      openflow_v1
OpenFlow 1.3      openflow_v4      openflow_v4
OpenFlow 1.4      openflow_v5      openflow_v5
```

The OpenFlow dissector is available with 0 as the default port. To allow the OpenFlow dissector to dissect the OpenFlow packets, you have to change the default port to 6633 or 6653, as described in [Preference Settings](#). However, you can force `tshark` to decode OpenFlow packets without changing the default port in Wireshark preferences, using the following command:

```
tshark -d tcp.port==6633,openflow -r file.pcap
```

Protocol Dependencies

TCP: OpenFlow uses TCP as its transport protocol. Well known TCP ports for OpenFlow traffic are 6633 and 6653 (the official IANA port since 2013-07-18).

Preference Settings

You can change the OpenFlow TCP port from the default port 0 to 6633 or 6653 in the user's preferences file (`~/.wireshark/preferences`) using the following command:

```
# openflow TCP port if other than the default
# A decimal number
openflow.tcp.port: 6633
```

Display Filter

A complete list of OpenFlow display filter fields can be found at <http://www.wireshark.org/docs/dfref/o/openflow.html> (Note: This page may not exist. It will be created on the release date of Wireshark 1.12) or listed with the following command:

```
tshark -G fields | grep -i openflow
```

To show only the OpenFlow based traffic, use the following command:

```
openflow
```

To show only the OpenFlow 1.3 based traffic, use the following command:

```
openflow_v4
```

Capturing OpenFlow Traffic

You cannot directly filter the OpenFlow protocols while capturing the OpenFlow traffic. However, if you know the TCP port used, you can filter the OpenFlow traffic on that port.

To capture the OpenFlow traffic over the default port (6633 or 6653), run the following command:

```
tcp port 6633
```

Additional Wireshark Documentation

You can refer to the following URLs for latest and up-to-date information on Wireshark:

- OpenFlow Support: <http://wiki.wireshark.org/OpenFlow>
- Documentation, Videos, Tutorials, and Presentations: <http://www.wireshark.org/docs/>

MISCELLANEOUS TOOLS

1. EPER: It is a loose collection of Erlang performance related tools. For information, refer to <https://github.com/massemanet/eper>.
2. Erlang Easy Profiling (EEP) - This application provides a way to analyze application performance and call hierarchy. For information, refer to <https://github.com/virtan/eeep>.
3. KCachegrind - It is a profile data visualization tool that is used to determine the most time consuming execution parts of a program. For information about KCachegrind, refer to <http://kcachegrind.sourceforge.net/html/Home.html>.

Case Studies

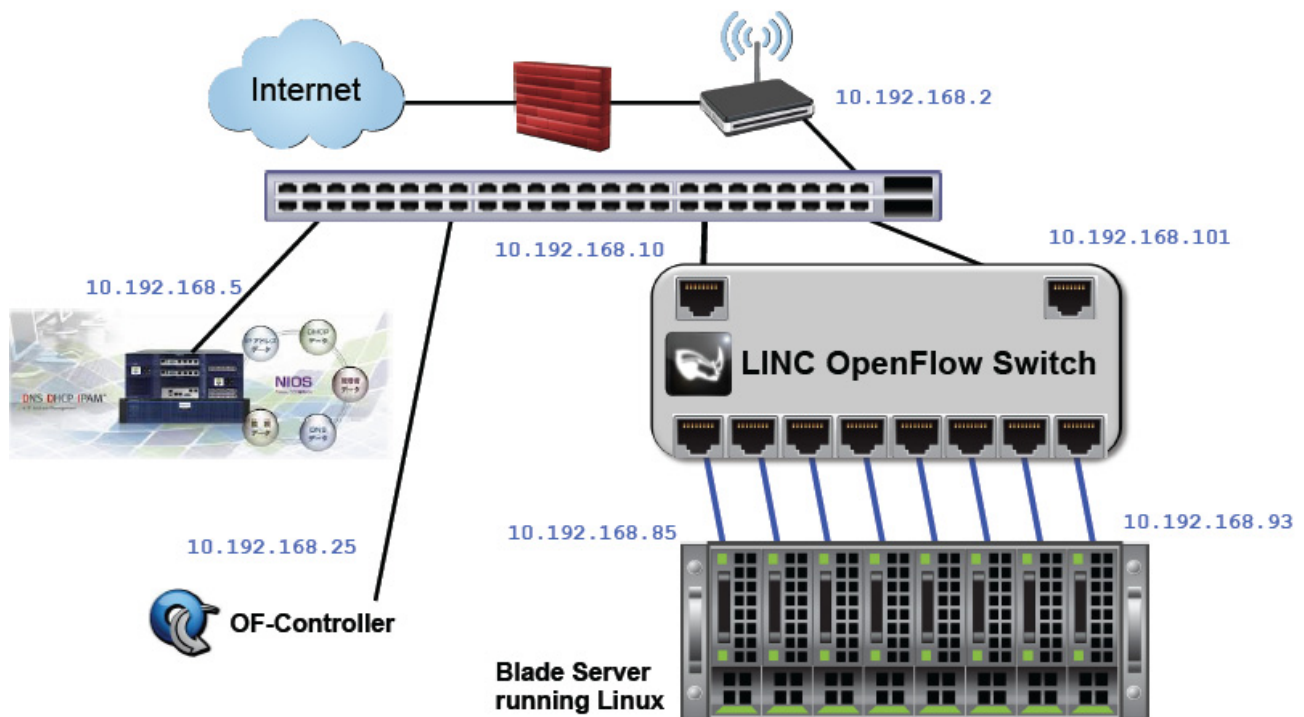
This chapter provides information about the case studies for LINC OpenFlow Switch. It includes the following sections:

- [*Deploying LINC on Corporate Networks*](#) on page 66
- [*Software Defined Networking \(SDN\) for OpenFlow Networks*](#) on page 67
- [*Deploying Dynamically Programmable Firewall for SDN*](#) on page 68
- [*Big Data Apache Hadoop Acceleration with OpenFlow*](#) on page 69
- [*Deploying Scalable and Programmable OpenFlow Switch with OF-Config*](#) on page 70
- [*Networking Tap/Monitoring with OpenFlow*](#) on page 71

DEPLOYING LINC ON CORPORATE NETWORKS

Customer Situation:

Solution:



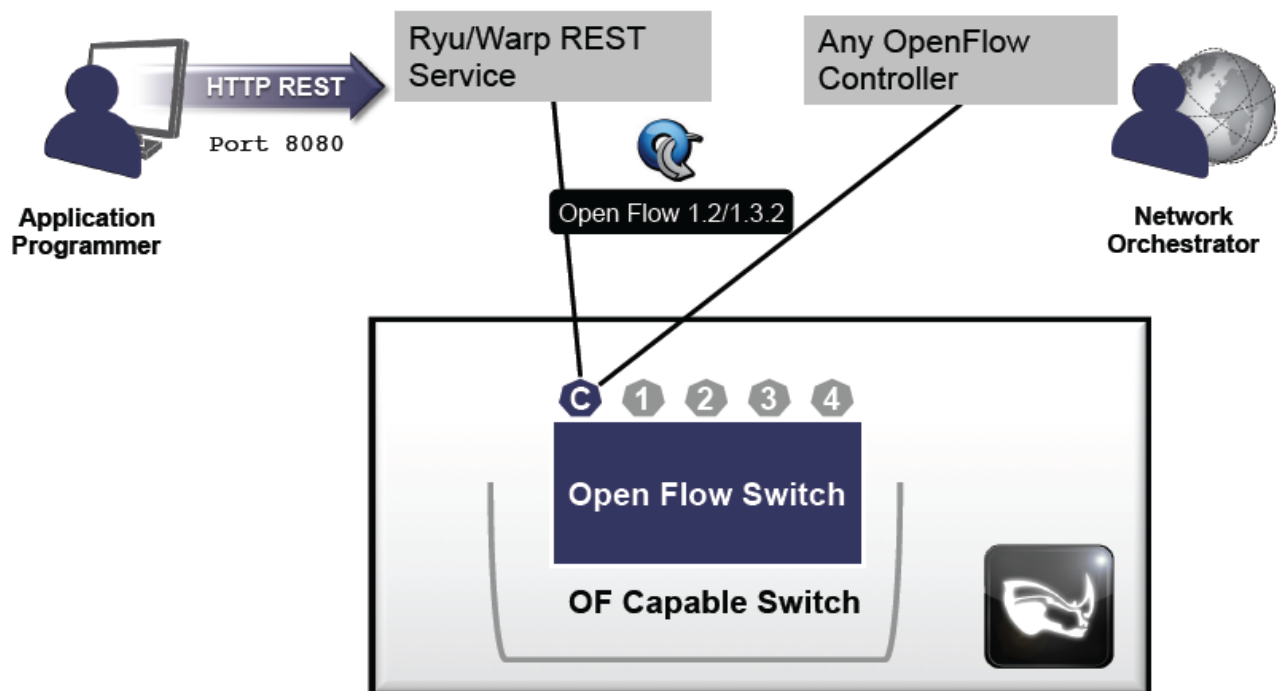
Benefits:

Results:

SOFTWARE DEFINED NETWORKING (SDN) FOR OPENFLOW NETWORKS

Customer Situation:

Solution:



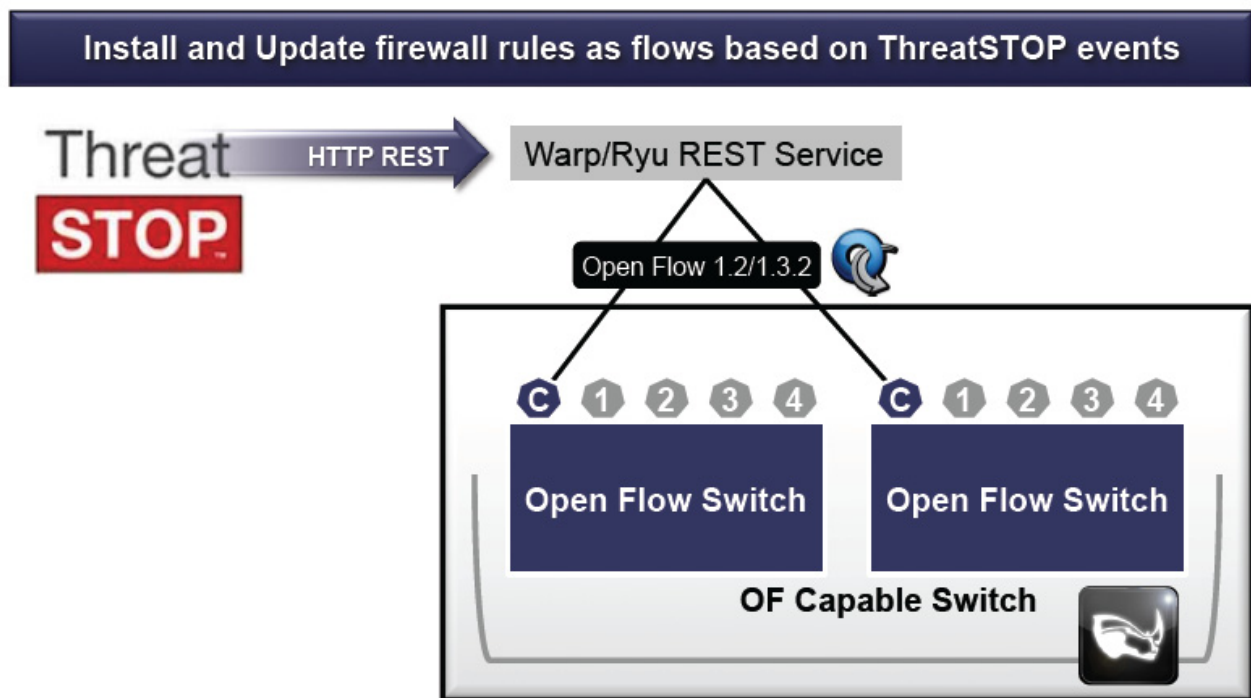
Benefits:

Results:

DEPLOYING DYNAMICALLY PROGRAMMABLE FIREWALL FOR SDN

Customer Situation:

Solution:



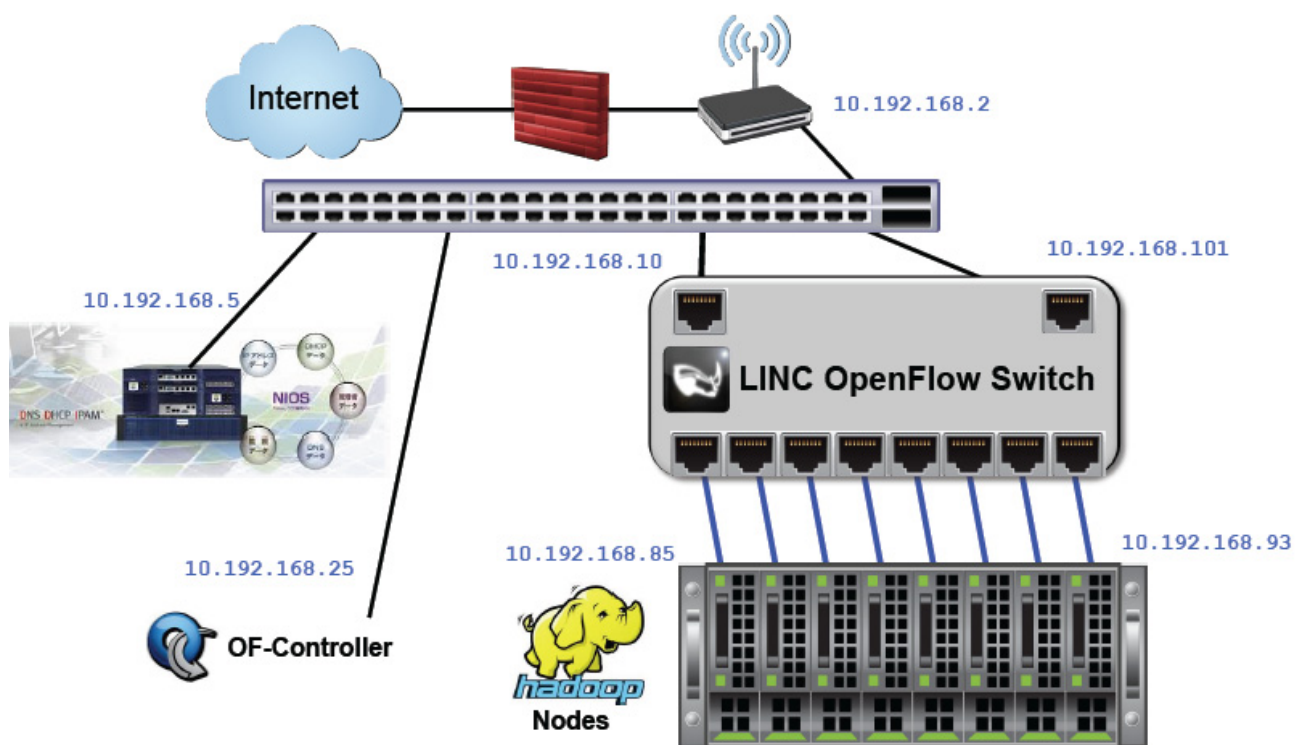
Benefits:

Results:

BIG DATA APACHE HADOOP ACCELERATION WITH OPENFLOW

Customer Situation:

Solution:



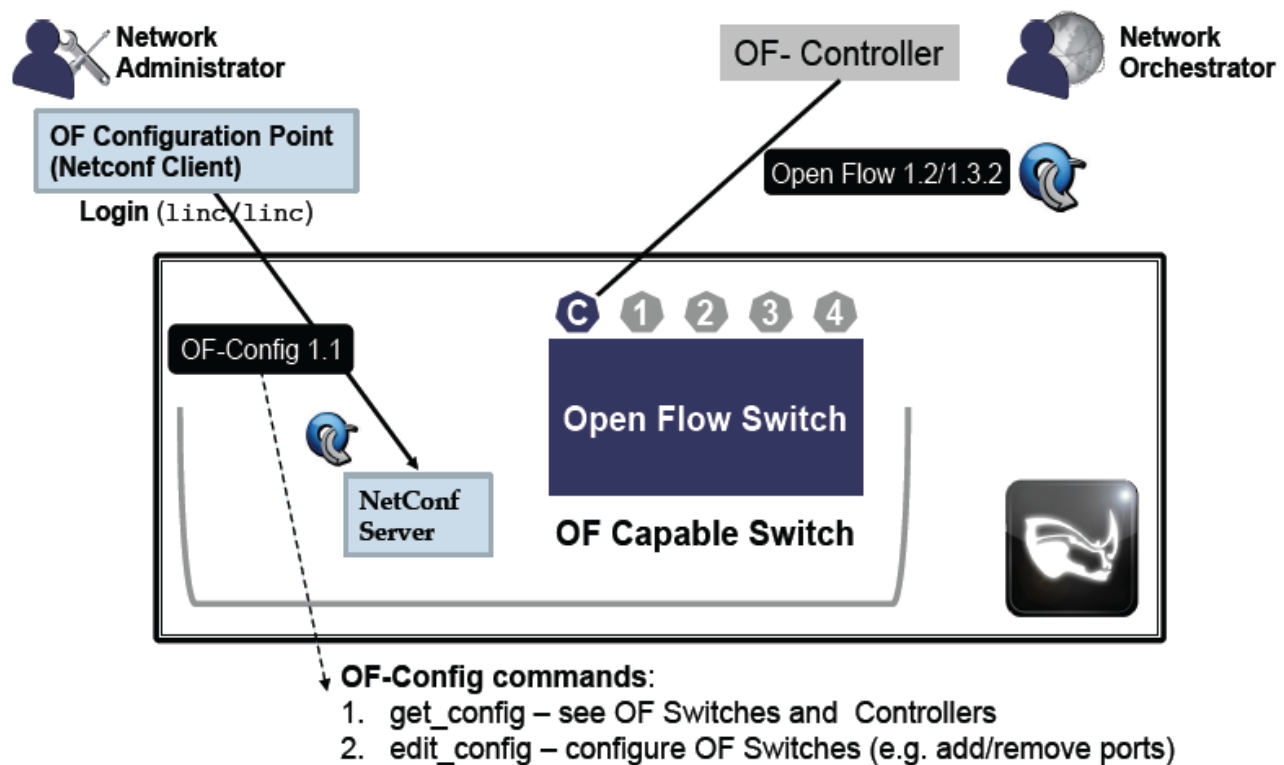
Benefits:

Results:

DEPLOYING SCALABLE AND PROGRAMMABLE OPENFLOW SWITCH WITH OF-CONFIG

Customer Situation:

Solution:



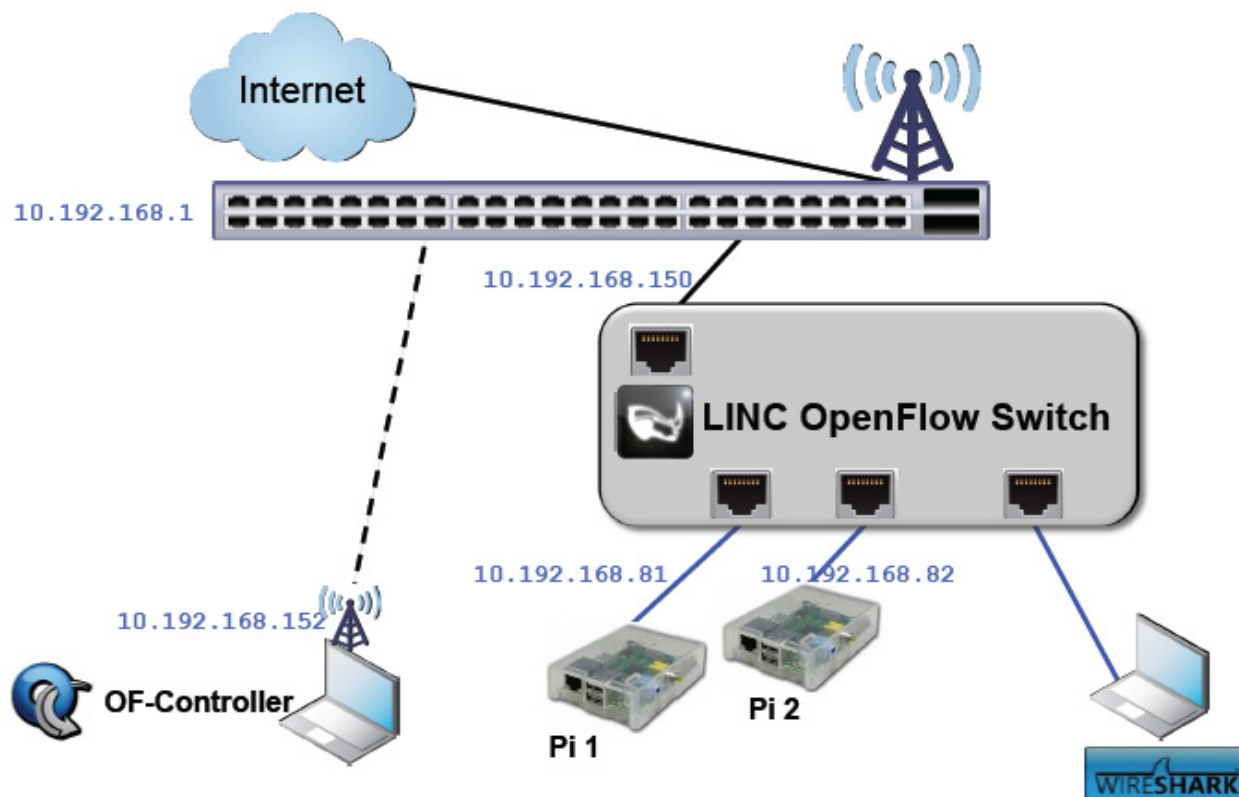
Benefits:

Results:

NETWORKING TAP/MONITORING WITH OPENFLOW

Customer Situation:

Solution:



Benefits:

Results:

Feedback

Please send your comments to linc-dev@flowforwarding.org, including things that you like or dislike, mistakes, corrections, and others.