Languages for Software Defined Networking

COEN 233 – Computer Networks
Santa Clara University

Raja E
Rajashree Kamath
Romil Jain

Table of Contents

Table of Figures

2. Introduction

The objective of this project is to create a high-level abstraction to the existing OpenFlow API's.

OpenFlow is an open standard which provides a standard interface through which researchers can install flexible packet-forwarding rules on physical network. OpenFlow uses a programmable controller which runs separately on a stock machine like NOX. By deploying OpenFlow in the commercial Ethernet switches in the campus or backbone networks, researchers have been able to create a variety of controller applications that introduce new network functionality, like flexible access control, Web server load balancing, energy-efficient networking, and seamless virtual-machine migration.

Although OpenFlow and NOX do make it possible to implement new network services, they do not make it easy. OpenFlow interface is not modular and provides very low level of abstraction. This makes programming OpenFLow quite challenging.

This project aims to improve the way network communications takes place with switches which form an integral part of computer networks. Therefore we feel that this project is well suited for this class.

The current implementation of OpenFlow is not very efficient and poses many challenges to programmers:

1. Networks often perform multiple tasks, like routing, access control, and traffic monitoring. Unfortunately, decoupling these tasks from each other and implementing them independently in separate modules is effectively impossible, since packet-handling rules (un)installed by one module often interfere with overlapping rules (un)installed by other modules.

2. The OpenFlow/NOX interface is defined at a very low level of abstraction

3. Controller programs only receive events for packets the switches do not know how to handle. Code that installs a forwarding rule might prevent another, different event-driven call-back from being triggered. As a result, writing programs for Open-Flow/NOX quickly becomes a difficult exercise in two-tiered programming—programmers must simultaneously reason about the packets that will processed on switches and those that will be processed on the controller.

4. Since network of switches is a distributed system, it is susceptible to various kinds of race conditions. For example, NOX controller's basic task is to handle the first packet of each network flow and install switch-level rules to handle the remaining packets. However, such programs can be susceptible to errors if the second, third, or fourth

packets in a flow arrive before the appropriate switch-level rule is computed and installed on the switches in the network.

To address the above challenges, we want to come up with a tool kit and runtime system that provides two levels of abstraction.

1. A set of source-level operators for constructing and manipulating streams of network traffic.
2. A runtime system that handles the intelligent automation by updating low-level rules on switches.

As writing applications for today's SDN controller platforms is a tedious exercise in low-level distributed programming, there is a need to provide a high level abstraction for the same.

To accomplish this we have decided to come up with a new tool kit to easily code the open flow API's as well as a run time operating system which can intelligently automate the state of the OpenFlow switches.

3. Theoretical Basis and Literature Review

Very little research has been done in the field of language of SDN because this is a relatively new field. OpenFlow is the API used for the controller to talk to the switches below. While SDN makes it possible to program the network, it does not make it easy. Today's OpenFlow controllers offer low-level APIs that mimic the underlying switch hardware. As such a new platform must be created to provide programmers to program with ease and not have to deal with the lower level switches.

There are four problems with the current OpenFlow program. First is that the programs do not compose because of the interaction between concurrent modules. There are rules for each program and when running these programs concurrently the rules will overlap, causing programs to break. Second problem is the low level programming interface that impedes programmers to abstract and create large and complex program for the network. Third problem is the Two-tiered system architecture that forces programmers to specify communication patterns between the controller and each individual switches to avoid tricky concurrency issues. The last challenge is network race conditions that arise directly from the two-tiered system.

One approach to tackle OpenFlow API issues is the language called Frenetic. Frenetic is a new, domain-specific language for programming OpenFlow networks. It is embedded in Python and comprises two integrated sublanguages: (1) a limited, but high-level and declarative network query language, and (2) a general-purpose, functional and reactive network policy management library. The language offers a number of features that make programming more convenient such as a single-tier, "see-every-packet" abstraction, strong compositionality properties, a clear cost model and a simple, race-free semantics.

Frenetic has many advantages including declarative design where the programmer decides what the hardware implements and not how, thus providing high level institutive primitives that all hardware will support. Modular design will provide programmers the ability to not worry about the underlying technology. Single-tier programming that will allow see-every-packet abstractions that will guarantee that every packet is available for analysis. And finally race free semantics will guarantee that race conditions are avoided. The disadvantages are that the programmer has to learn a new language that maybe error prone. Coding tools are missing with the language and there is very little support for this.

Our solution to solve these problems is to create a toolkit that eases the process of defining the OpenFlow API as well as come up with a run time operating system that can intelligently automate the state of the switches. In addition this toolkit will be a SDK that is written in JAVA, that is supported by many different toolkits and other SDKs.

Our solution is different from others because this SDK and runtime system is written in Java SDK. This will provide ease of coding because many people already know Java. Java is not a function language unlike Frenetic so programmers do not need to learn a new type of way of programming.

This solution is better because this will be less error prone since the language is already known to most programmers. It is very easy to code because the SDK will provide high level of abstraction to programmers. This is also a pure object oriented style of coding and avoids forcing programmers to use functional programming. And finally programmers do not need to learn a new language and higher level of application will be able to use this SDK to integrate OpenFlow switches in those applications.

4. Hypothesis

We hope to be able to realize a new toolkit to enhance the programming of OpenFlow switches that allows programmers easy of coding and the ability to integrate into higher level of application that use SDN technology.

5. Methodology

5.1 Network Design

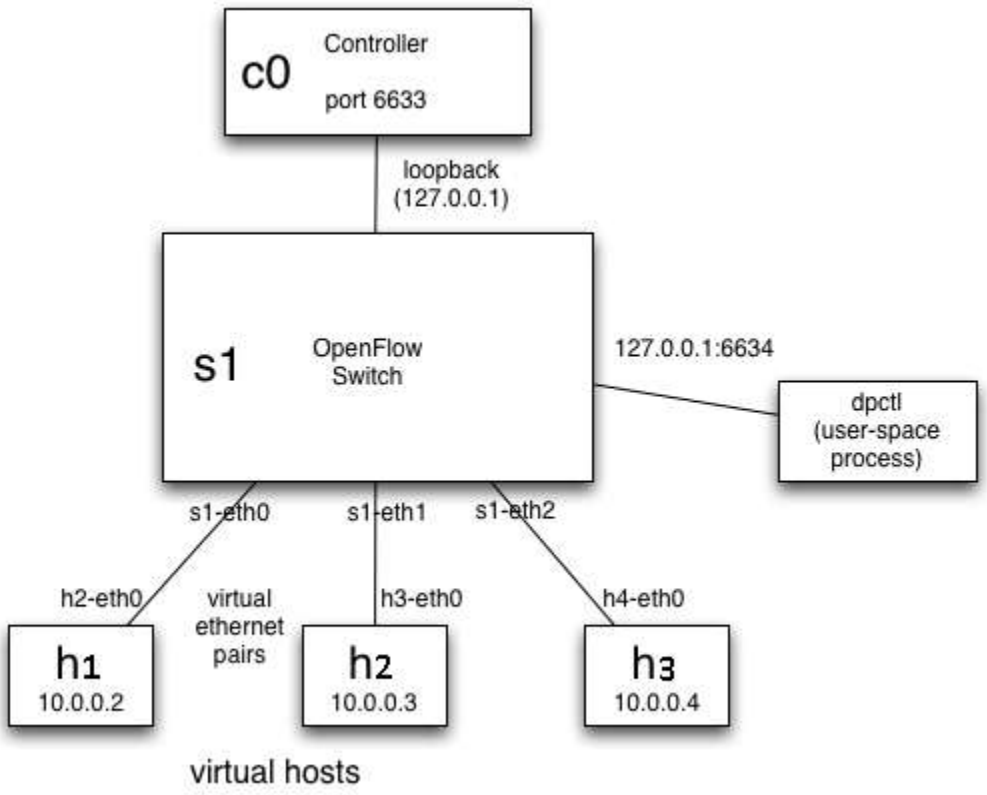

Figure 1 – OpenFlow network design

5.2 Tools

Here is the list of tools that will be used by this project.

1. **Virtualbox console terminal:** connects to OpenFlow environment. This is the one created when you started up the VM.
2. **Virtual Machine:** runs the OpenFlow Controller, OpenFlow switch and hosts
3. **SSH Terminal:** connects to OpenFlow environment. Created by using putty on Windows or SSH on OS X / Linux
4. **Xterm terminal:** connects to a host in the virtual network.

5. **OpenFlow Controller:** sits above the OpenFlow interface. The OpenFlow reference distribution includes a controller that acts as an Ethernet learning switch in combination with an OpenFlow switch.
6. **OpenFlow Switch:** sits below the OpenFlow interface. The OpenFlow reference distribution includes a user-space software switch. Open vSwitch is another software but kernel-based switch, while there is a number of hardware switches available from Broadcom (Stanford Indigo release), HP, NEC, and others.
7. **Hosts:** The end terminals whose traffic flows through the switches.
8. **Mininet:** network emulation platform. Mininet creates a virtual OpenFlow network - controller, switches, hosts, and links - on a single real or virtual machine.
9. **Dpctl:** command-line utility that sends quick OpenFlow messages, useful for viewing switch port and flow stats, plus manually inserting flow entries.
10. **Wireshark:** general (non-OF-specific) graphical utility for viewing packets. The OpenFlow reference distribution includes a Wireshark dissector, which parses OpenFlow messages sent to the OpenFlow default port (6633) in a conveniently readable way.
11. **Iperf:** general command-line utility for testing the speed of a single TCP connection.
12. **Cbench:** utility for testing the flow setup rate of OpenFlow controllers.

OpenFlow will generate the data to be sent from one virtual host to another. This traffic can be captured and viewed in wireshark.

5.3 OpenFlow SDK (oSDK)

OpenFlow SDK will help the developers to write the OpenFlow scripts in java which will in turn be converted into native OpenFlow language. oSDK will also create a runtime system to monitor the traffic coming to the switches and performs intelligent automation to increase the performance and efficiency of the network.

6. Implementation

oSDK provides the network developer a programmatic interface to control the network. oSDK includes all the features of Object Oriented Programming. oSDK provides application interfaces to Controller, Switch and host components.

Flow Control:

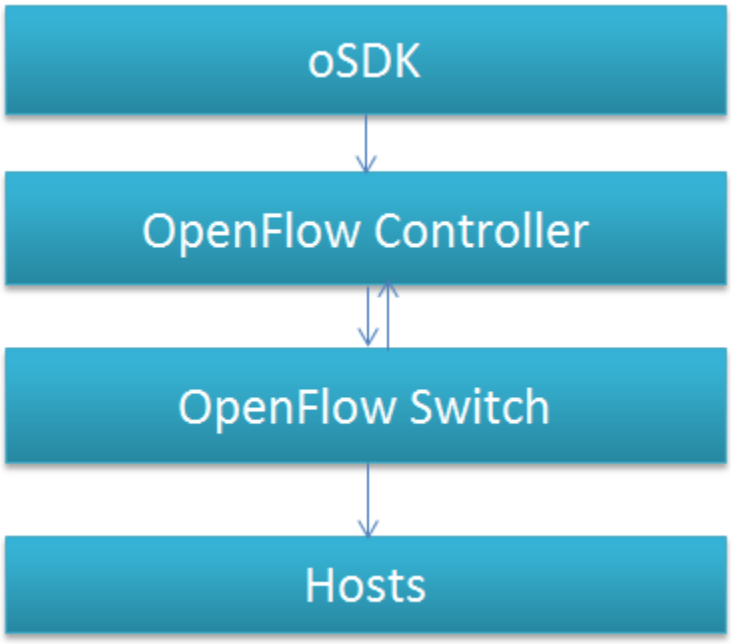

Figure 2 – OpenFlow Flow Control

oSDK includes the following classes:

1. OFController.java:

   This class provides an API to start the controller.

```
public class OFController {

	public OFController(String name){

	}
	public void start(){

	}

}
```

2. OFSwitch.java:

   This class provides an API to get the allocated ports, to add flows and to get all the assigned flows to an OpenFlow switch.

```
public class OFSwitch {

	public OFSwitch(String name) {

	}

	public void addFlow(int fromPort, int toPort) {

	}

	public ports getPorts() {

	}

	public void getFlows() {

	}

}
```

3. OFAdapter.java:

   This class provides an API to implement automations in OpenFlow.

```
public class OFAdapter {

	public void getConnection(String host, String username, String password,
	int port){

	}

	public void getNodes(){

	}

	public int getN() {
		return n;
	}
	public void setN(int n) {
		this.n = n;
	}
	public List<String> getRules() {
		return rules;
	}
	public void setRules(List<String> rules) {
		this.rules = rules;
	}
```

```
public List<Pattern> getPatterns() {
	return patterns;
}
public void setPatterns(List<Pattern> patterns) {
	this.patterns = patterns;
}
public int getPriority() {
	return priority;
}
public void setPriority(int priority) {
	this.priority = priority;
}
public int getTimeout() {
	return timeout;
}
public void setTimeout(int timeout) {
	this.timeout = timeout;
}
public List<String> getActions() {
	return actions;
}
public void setActions(List<String> actions) {
	this.actions = actions;
}

public int getPorts() {
	return ports;
}
public void setPorts(int ports) {
	this.ports = ports;
}

class Pattern{

	String name;

	Pattern(String name){
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Integer getInputPort() {
		return inputPort;
	}

	public void setInputPort(Integer inputPort) {
		this.inputPort = inputPort;
	}
```

```
public Integer getOutputPort() {
	return outputPort;
}

public void setOutputPort(Integer outputPort) {
	this.outputPort = outputPort;
}


public Integer getVlan() {
	return vlan;
}

public void setVlan(Integer vlan) {
	this.vlan = vlan;
}

public String getDataLinkSrc() {
	return dataLinkSrc;
}

public void setDataLinkSrc(String dataLinkSrc) {
	this.dataLinkSrc = dataLinkSrc;
}

public String getDataLinkDst() {
	return dataLinkDst;
}

public void setDataLinkDst(String dataLinkDst) {
	this.dataLinkDst = dataLinkDst;
}



public String getDataLinkType() {
	return dataLinkType;
}

public void setDataLinkType(String dataLinkType) {
	this.dataLinkType = dataLinkType;
}

public String getNetworkSrc() {
	return networkSrc;
}

public void setNetworkSrc(String networkSrc) {
	this.networkSrc = networkSrc;
}

public String getNetworkDst() {
	return networkDst;
}
```

```
	public void setNetworkDst(String networkDst) {
		this.networkDst = networkDst;
	}

	public String getNetworkProtocol() {
		return networkProtocol;
	}

	public void setNetworkProtocol(String networkProtocol) {
		this.networkProtocol = networkProtocol;
	}

	public String getTransportSrc() {
		return transportSrc;
	}

	public void setTransportSrc(String transportSrc) {
		this.transportSrc = transportSrc;
	}

	public String getTransportDst() {
		return transportDst;
	}

	public void setTransportDst(String transportDst) {
		this.transportDst = transportDst;
	}

}
```

4. Host.java

   This class provides all the interfaces required for the host.

```
public class Host {

	public Host(String name){

	}

	public void ping(Host host){

	}

}
```

5. SSHConnection.java

   This class Provides an API to connect to the SSH terminal of OpenFlow Controller and OpenFlow switch.

```
public class SSHConnection {

	public static Session getSession() {

	}
}
```

7. Data Analysis

The output of the various oSDK functions are given below.

OFSwitch.getPorts:

```
features_reply (xid=0xa3510413): ver:0x1, dpid:1
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xfff
 1(s1-eth1): addr:62:63:98:6d:9a:89, config: 0, state:0
     current:    10GB-FD COPPER
 2(s1-eth2): addr:ea:52:bb:35:ac:8f, config: 0, state:0
     current:    10GB-FD COPPER
 3(s1-eth3): addr:d6:45:94:ef:65:79, config: 0, state:0
     current:    10GB-FD COPPER
 LOCAL(s1): addr:2e:8a:a2:25:5a:43, config: 0x1, state:0x1
get_config_reply (xid=0x26aaa1d1): miss_send_len=0
```

OFSwitch.getFlows:

```
stats_reply (xid=0x9e494468): flags=none type=1(flow)
  cookie=0, duration_sec=37s, duration_nsec=242000000s, table_id=0,
  priority=32768, n_packets=5, n_bytes=378,
  idle_timeout=60,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration_sec=36s, duration_nsec=99000000s, table_id=0,
  priority=32768, n_packets=5, n_bytes=378,
  idle_timeout=60,hard_timeout=0,in_port=2,actions=output:1
```

oSDK simplifies the programming effort of a network administrator.

Example: To add the routes to the switch in openflow, the below scripts needs to be executed.

```
$ dpctl add-flow tcp:127.0.0.1:6634
in_port=1,actions=output:2
$ dpctl add-flow tcp:127.0.0.1:6634
in_port=2,actions=output:1
```

In oSDK, the code will be as shown below.

```
OFSwitch ofSwitch = new OFSwitch("S0");
ofSwitch.addFlow(1, 2);
ofSwitch.addFlow(2, 1);
```

8. Conclusions and Recommendations

oSDK greatly simplifies the life of a network administrator by providing a java interface to the openflow. Though oSDK simplifies the programming effort, it does add an overhead over the existing OpenFlow commands, which in turn slows down the network marginally. In order to overcome this defect, we can rewrite all the API's in C language which is slightly faster than Java.

oSDK requires knowledge of Java which all the network administrators may not be aware of.

Recommendations:

1. Rewrite all the APIs in C language which marginally increases the speed of the APIs.

2. Applications can be written on top of these APIs.

3. Enhance the functionality of oSDK.

4. Enhance the functionality of oMap.

9. Bibliography

N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in ACM SIGPLAN International Conference on Functional Programming (ICFP), Sept. 2011.

N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," SIGCOMM CCR, vol. 38, no. 2, pp. 69–74, 2008.

S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking and the past of protocols," Oct. 2011. Invited talk at Open Networking Summit.

Nate Foster, Michael J. Freedman, Arjun Guha, Rob Harrison, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Mark Reitblatt, Jennifer Rexford, Cole Schlesinger, Alec Story, and David Walker "Languages for Software-Defined Networks". See http://www.cs.princeton.edu/~jrex/papers/frenetic12.pdf

The Frenetic language. See http://www.frenetic-lang.org/, Nov 2010.

OpenFlow. See http://www.openflowswitch.org, Nov 2010.

Nate Foster, Rob Harrison, Matthew L. Meola, Michael J. Freedman, Jennifer Rexford, and David Walker. Frenetic: A high-level language for OpenFlow networks. In PRESTO, Nov 2010.

10. Appendices

Source code for oSDK can be found here:

oSDK-Project.zip

JavaDoc for oSDK can be found here:

oSDK-Javadoc.zip

Required Software and download links:

Virtual box: Virtual box can be downloaded from

http://www.virtualbox.org/wiki/Downloads

Virtual Machine image: Virtual Machine can be downloaded from

https://github.com/downloads/mininet/mininet/mininet-2.0.0-113012-amd64-ovf.zip

Xming Server: Xming server can be downloaded from

http://sourceforge.net/project/downloading.php?group_id=156984&filename=Xming-6-9-0-31-setup.exe

Putty: Putty can be downloaded from

http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe