**CELL iteration 4 Advanced Concepts: 10/14/16: Dottie Kessler**

1. Start VM, login (mininet/mininet) and obtain ip (**sudo dhclient eth1**)
2. Run Xming: then xterm from within mininet
3. Putty to VM; run wireshark then filter of ; capture->interfaces->loopback (**sudo wireshare &**)
4. Putty to VM run ODL (**cd ~/opendaylight; sudo ./run.sh -virt ovsdb**)
5. Browser to host open IP of VM (admin/admin) [**http://192.168.56.101:8080**](http://192.168.56.101:8080)
6. Start ONOS, run Virtual box, select ONOS VM (tutorial1/tutorial1)
7. Virtual Box is Version 5.1.2 r108956; Ubuntu 14.04

**Task 4.1 Understanding REST Architecture**

4.1.1 Description: What is REST http://www.restapitutorial.com/lessons/whatisrest.html# (30 mins)

4.1.2 Another stab at understanding REST (because this is important)- http://www.looah.com/source/view/2284 (10 mins).

4.1.3 OSCP Rest Reference- https://wiki.opendaylight.org/view/OpenDaylight\_SDN\_Controller\_Platform\_(OSCP):Rest\_Reference (15 mins)

4.1.4 ONOS, CLI, GUI and REST API: https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614 (Optional - 5 mins)

4.1.1 **What is REST**

1. **Re**presentational **S**tate **T**ransfer
2. Resourced Based, versus action based (things not actions, nouns not verbs)
3. Nouns not verbs, like a person resource, a user resource, rather than method to call an api
4. Resources identified by URI (eye not el), multiple URIs can refer to same resource
5. Example: **Resource** : person; **Service** (GET); **Representation**: name, address, (JSON or xml)
6. The representation we get back, represents nouns in json format, like name, address
7. Represents state that transfers between client and server, most commonly json
8. Six Constraints
   1. Uniform interface (interface between client/server)
      1. HTTP verbs (GET PUT POST DELETE)
      2. URI (resource name)
      3. HTTP response (status, body) usually in json
   2. Stateless
      1. (host contains no client state, each message is self-descriptive)
      2. If there is state, it is kept on the client side
   3. Client-server
      1. Disconnected system, separation of concerns
      2. HTTP stack is the link between client and server
      3. REST architectures are going to be client/server
   4. Cacheable
      1. Server responses (representations) are cacheable
   5. Layered system
      1. Clients cannot assume direct connect to server, could be intermediaries
      2. Multiple layers of software in the mix, client only knows about the api
      3. Good for scalability
      4. From server side back things can be cached
   6. Code on Demand
      1. Examples: Java applets, executable JavaScript
      2. This is the only optional constraint

4.1.2 **Another stab at understanding REST (the guy talking to his wife article)**

1. HTTP protocol
2. Web is built on architectural style REST
3. A web page is a representation of a resource
4. Web services, a machine uses the web similar to a person
5. Polymorphism, different nouns can have the same verb applied to them
6. Universal verbs GET PUT DELETE
7. When you go to a web page, the browser does a GET
8. A POST is to add something to another system, a PUT updates something

4.1.3 **OSCP Rest Reference**

1. REST api for the OpenDaylight SDN controller
2. Firewall setting on the controller need to allow access to the REST API if it is running remote
3. Use port 80, 8000 is deprecated
4. You can query items using HTTP GET (controller-node, alias, interface etc)
5. You can do complex queries using optional query params
6. Parameter format is semicolon-separated list of <name>=<value> pairs
7. New items are created with PUT, not conventional but allows create multiple items in 1 call
8. Items are updated using PUT
9. Item are deleted using DELETE (should only delete flow-entry data type)
10. Do not delete switch, port, host, link out from under the controller

4.1.4 **ONOS, CLI, GUI and REST API (Configuring Authentication)**

1. Is an operating system for networks, distributed architecture, Intent Framework
2. GUI REST API Apache Karaf client (karaf/karaf) and ONOS credentials (onos/rocks)
3. Key based authentication, no cookies, HTTPS not enabled
4. ONOS provides the control plane for an SDN network
5. Manages network components, switches, links
6. User facing software, CLI and GUI
7. ONOS manages the entire network, ONOS platform acts as a distributed SDN controller
8. ONOS is a platform, has a kernel and core services as well as ONOS applications
9. ONOS is written in java, it runs the JVM on the Kataf OSGi container
10. ONOS can run on several underlying OS platforms, Ubuntu for example
11. Ubuntu Server 14.04 LTS 64 bit is recommended (2016); java 8
12. ONOS is open source

**Task 4.2 Open Network Operating System (ONOS)**

4.2.1 To understand ONOS

4.2.2 Read this article to gain an understanding of the Intent concept used by ONOS.

4.2.3 Read this article to obtain a more in-depth view of Intent-based networking.

4.2.4 This article tells you why the network industry uses Intent-based networking.

Purpose. To gain an understanding of ONOS.

4.2.1 **To understand ONOS, ONOS Overview White Paper. Also, please view ONOS Overview video**

1. Use minitnet to create a forwarding plane to generate traffic
2. Start up ONOS, the switches will concept over the flow channel
3. The controller will discover the topology information
4. VM has mininet and ONOS controller
5. In mininet, --mac so posts do not have random macs in mininet
6. Without the controller, the pings will all fail
7. Open the controller ONOS Cardinal Release: ok clean; leaves you in ONOS CLI console window
8. **list** (command in the cli, all commands are launched as application bundles)
9. **devices, list, links** (unidirectional), **hosts, flows, list | grep fwd** (the application running)
10. by default ONOS puts 5 rules in every switch
11. eth type 800 is IPv4 traffic; 806 is ARP traffic
12. After controller running, go back to mininet, and the pings now work, new rules added
13. Reactive flow
14. Start a browser to see the graphical interface, localhost:8181/onus/ui
15. The default ONOS GUI login/password = karaf
16. Hit H in the ONOS GUI, it toggles visibility of host that have been discovered by the controller
17. Forward or backward slash brings up quick help in the ONOS GUI
18. In ONOS GUI hover over the switch and you can see the flow
19. **stop onos-app-fwd** to stop, you will see the pings stop, the flows expire and get dropped
20. Once you stop the onos-app-fwd, then all traffic goes to the controller (application bundle)
21. You do not need to stop ONOS to deal with the application bundles
22. **start onos-app-fwd**
23. then you can stop mininet, and clean it up; you can **logout** ONOS

4.2.2 Read this article (10 mins) to gain an understanding of the Intent concept used by ONOS.

1. ONOS operating system has a subsystem Intent Framework
2. Intent Framework makes needs known using policy-based management to the controller
3. Intent Framework: don’t me what to do, tell me what you want
4. Tell the network what you need and let the controller figure it out
5. The controller takes the request and makes it happen
6. Needs for example more bandwidth
7. The configuration of the network becomes automated

4.2.3 Read this article (10 mins) to obtain a more in-depth view of Intent-based networking.

1. Create a network effect by adoption of IBN (Intend-Based Networking)
2. Not describing what to do with protocols, ports, channels address etc.
3. IBN gaining wide support
4. ODL Helium contains IBN demos
5. IBN approaches being discussed in the open source communities

4.2.4 This article tells you why the network industry uses Intent-based networking.

1. Advantages of IBN
2. Manage a network simply by identifying your intent, what you want to happen
3. Software figures out the how
4. The next step in network automation (scalability)
5. Eliminates need for deep vendor knowledge
6. The process becomes the same regardless of hardware or virtual
7. Improves access control, is stable and secure
8. Rely on software to automate
9. Allows self service

**Task 4.3 Perform BASIC ONOS Tutorial**

4.3.1 Go to the ONOS Download page. Scroll down to the Tutorials section.

4.3.2 Under the **Cardinal** release, in the Tutorial VM column, click on onos-tutorial-1.2.1r2-ovf.zip.

This downloads the ONOS image onto your machine. **UseWinzip 7 to unzip the file.**

4.3.3 Open the Basic ONOS Tutorial. Step through the tutorial to create the ONOS VM.

4.3.4 Once the ONOS VM is created, continue to go through the Tutorial to start Mininet and ONOS, as well as using the Reset capability.

4.3.5 Once ONOS is up and running, work through the Tutorial exercises.

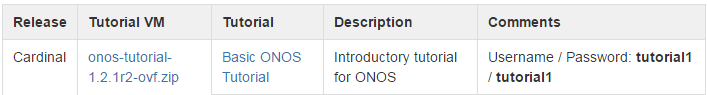
**4.3.1 Go to the ONOS Download page. Scroll down to the Tutorials section.**

<https://wiki.onosproject.org/display/ONOS/Downloads>

**4.3.2 Under the Cardinal release, in the Tutorial VM column, click on onos-tutorial-1.2.1r2-ovf.zip.**

**This downloads the ONOS image onto your machine. Use Winzip 7 to unzip the file.**

1. TUTORIALS: Cardinal Release, onos-tutorial-1.2.1r2-ovf.zip



1. Need to also get winzip 7 on my personal laptop

**iteration4.3.A.unzip.png**

**4.3.3 Open the Basic ONOS Tutorial. Step through the tutorial to create the ONOS VM.**

1. <https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial>
2. Tutorial is a preconfigured virtual machine with the needed software, Ubuntu 14.04
3. I have the required software installed, Virtual box, and the tutorial VM unzipped
4. **Create the virtual machine** locate the OVF file

I made a mistake here; I was clicking on the ovf file inside the zip which creates a temporary file in the temp directory. I needed to make sure I was clicking on the ovf file that was actually unzipped

**right**

C:\Users\dak\Documents\CELL\downloads\onos-tutorial-x86\_64.ovf

**wrong**

C:\Users\dak\Documents\CELL\downloads\onos-tutorial-1.2.1r2-ovf.zip\onos-tutorial-x86\_64.ovf

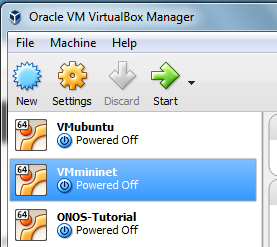
Failed to import appliance **C:/Users/dak/AppData/Local/Temp/Temp1\_onos-tutorial-1.2.1r2-ovf.zip/onos-tutorial-x86\_64.ovf**.Error opening **'C:/Users/dak/AppData/Local/Temp/Temp1\_onos-tutorial-1.2.1r2-ovf.zip\onos-tutorial-x86\_64.vmdk'** for reading (VERR\_FILE\_NOT\_FOUND).

1. Open the ovf file (this should open VirtualBox) gives you a dialog

**iteration4.3.B.OpenOVFfile.png**

1. Import the VM

**iteration4.3.C.importOVF.png**

****

**4.3.4 Once the ONOS VM is created, continue to go through the Tutorial to start Mininet and ONOS**

**as well as using the Reset capability.**

1. Start the VM and login using tutorial1/tutorial1

**iteration4.3.D.VMstartup.png**

**iteration4.3.E.ONOSlogin**

**iteration4.3.F.ONOSdesktop.png**

**4.3.5 Once ONOS is up and running, work through the Tutorial exercises.**

1. Stat mininet: 6 switches, 4 outer each with 6 hosts (24 hosts)

**iteration4.3.G.ONOSmininet.png**

1. Stat ONOS (first time via Reset icon to ensure environment clean)

**iteration4.3.H.ONOSstartONOS.png**

1. Reactive Forwarding, open an ONOS console

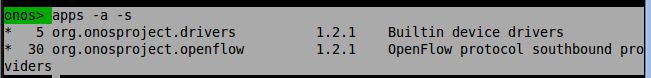
**iteration4.3.I.ONOSconsole.png**

1. Reactive Forwarding (installs flow for every packet that arrives at the controller)

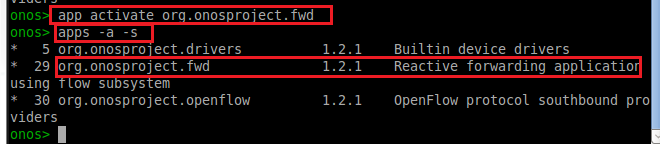
The pings fail at this point because we did not load the reactive forwarding app

**iteration4.3.J.ONOSpingfails.png**

1. List the loaded apps on the ONOS console, there is no reactive forwarding app



1. Load the reactive forwarding app (dynamically load apps)



1. Pings now work

**iteration4.3.K.ONOSpingworks.png**

1. Interrupt applications (deactivate) while they are running

Stop the reactive forwarding application

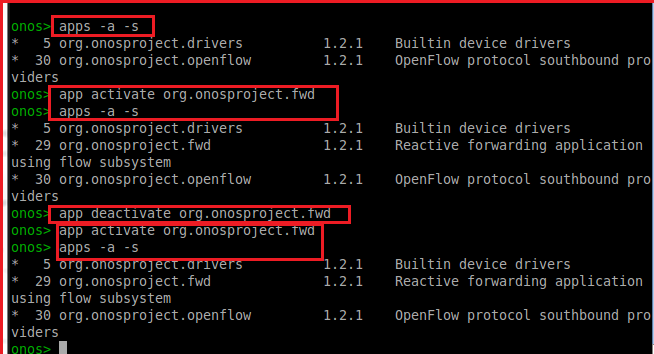
Noticed the pings stopped and now again fail, because the pushed rules have been removed

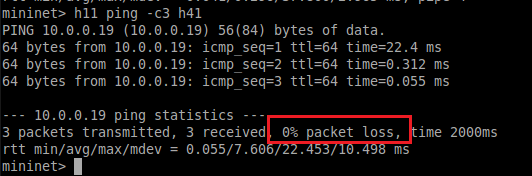
**iteration4.3.L.ONOSpingfailsagain.png**

1. Restart the reactive application in ONOS; then the pings work again

**onos> app activate org.onosproject.fwd**

you see it was not there, then we activated it, then we deactivated it, then again reactivated





1. ONOS CLI commands

**help onos**

**iteration4.3.M.ONOScli.help.png**

**devices** (list devices currently known to the system)

**iteration4.3.N.ONOScli.devices.png**

**links**

**iteration4.3.O.ONOScli.links.png**

**hosts**

**iteration4.3.P.ONOScli.hosts.png**

1. **flows** commands, flows can be in several states

PENDING\_ADD, ADDED, PENDING\_REMOVE, REMOVED

Use ping to generate traffic, and then run the **flows** command

**iteration4.3.Q.ONOScli.flows.png**

I had more flow rules here

1. **paths** command

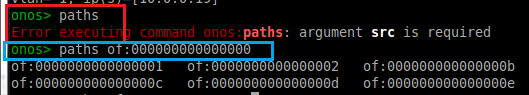
ONOS computes the shortest paths

**iteration4.3.R.ONOScli.paths.png**

cli auto completes by hitting <TAB>

On the first line, I just typed paths

On the second line, I typed paths followed by the TAB



1. **intent** command: **intents -i** shows sub-intents

You can see what intents are stored in the system

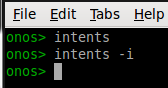
Intents can be in several states:

SUBMITTED, COMPILING, INSTALLING, INSTALLED

RECOMPILING, WITHDRAWING, WITHDRAWN, FAILED

<https://wiki.onosproject.org/display/ONOS/Intent+Framework>

You do not see intents until some have been added



1. **Intent Reactive Forwarding (activate)**

Rather than pushing flows for each packet like the reactive forwarding app, the intent reactive forwarding app provisions the intent, a hosts to host intent:

Remove the reactive forwarding application and load the intent reactive forwarding app

**iteration4.3.S.ONOScli.activateintent.png**

1. **Intentionally React (cause traffic)**

Use **ping** to generate traffic, then you can use **intents -i** to see the intent was installed

The end result of the ping is the same, but how it got there is very different.

**iteration4.3.T.ONOScli.activateintenwithdata.png**

The intent gets pushed by the intentionally reactive application

You can tell that, the appid is org.onlab.onos.ifwd

The intent is host to host intent

The paths have been shown to you

<https://wiki.onosproject.org/display/ONOS/Intent+Framework>

1. **Clear the intents**

The intents command will return nothing when no intents left

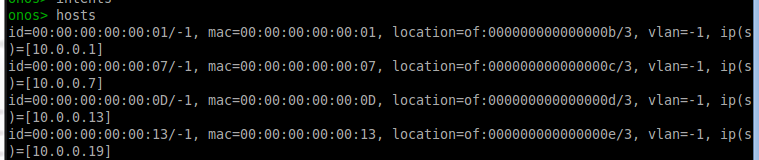
**iteration4.3.U.ONOScli.removeintents.png**

1. **State your intentions**

Advantage: it reconfigures itself (eg if a link goes down)

It tracks your intent; it can also fail (no path available)

At this point in the tutorial, we have 4 hosts



Install a host to host intent between 2 of the hosts (11 and 41)

10.0.0.1 (h11) and 10.0.0.19 (h41)

**iteration4.3.V.ONOScli.addhostintent.png**

1. **check out the flows**

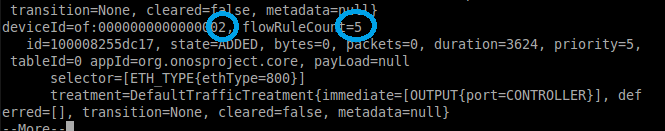
I had many more entries than the tutorial

**iteration4.3.W.ONOScli.flows.png** (shows s1 with 7)

My flow table showed device 1 with 7 flow entries for device 1 (different than tutorial)

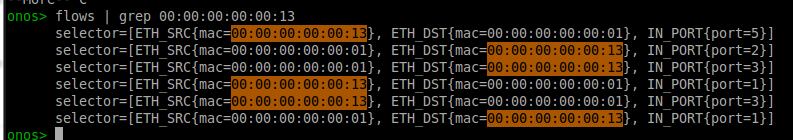


My flow table also showed device 2 with 5 flow entries (different than tutorial)



Even though I had many more flow entries, the traffic was flowing between s1 and s11, same as the tutorial. I had 3 flows between

10.0.0.1 h11, 1 in hex 00:00:00:00:00:01 and 10.0.0.19 h41 13 in base 16 (16+3)



It was hard to take a screen capture of the flows, I couldn’t capture enough information in the terminal window, the selectors between 1 and 13 (base 16) were under device 1, and device 000000000000000b (device 11, b is 11 in hex)

1. **tear down the link between s1 and s11 (this is done in mininet, not ONOS)**



1. **Examine the flows again, s1 loses 2 entries, s2 picks up 2 entries**

When we tore down the link, ONOS detected it and informed the devices that the link went down. The intent logic then realizes one of its intents is impacted, so it recompiles the intent, and this causes the intent to be installed on a different path.

Now, I can see I only have 5 flows **on switch 1**, rather than the 7 I started with **iteration4.3.X.ONOScli.flowsafterswitchdown.png** (shows switch 1 now with 5 flows)



And now, **switch 2** has 7 flows, rather than the 5 it started with

**iteration4.3.Y.ONOScli.flowswitch2picksup.png** (shows switch 2 now with 7 flows)



And now, you can see the details also under the switch 2 flow



1. **Bring links back up (mininet command)**



**ONOS GRAPHICAL USER INTERFACE**

1. **Let’s reload the reactive forwarding application**

onos> **app deactivate org.onosproject.ifwd**

onos> **app activate org.onosproject.fwd**

1. **Click on the ONOS GUI icon (ONOS – Chromium)**

When it loads, it will show you the networks topology

**iteration4.3.Z1.ONOScli.GUI.png**

1. **Click on the ONOS GUI icon (ONOS – Chromium)**

Cheat sheet hit / for toggle commands on the GUI

1. **pingall (in mininet), hosts now appear in GUI**

**iteration4.3.Z2.ONOScli.GUI.png**

1. **GUI features**

Cheat sheet, type /

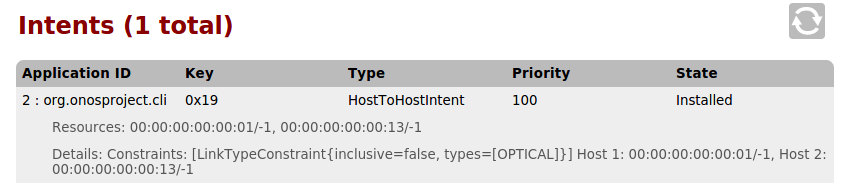
ONOS Summary pane, (can be toggled off)

Switch details, click on a switch

Press shift click to unselect the device on the GUI

**iteration4.3.Z3.ONOScli.GUI.png**

1. **Install an intent using the UI**
   1. **Before I do this, I had one intent existing**



* 1. **Select Nodes 14 and 9**

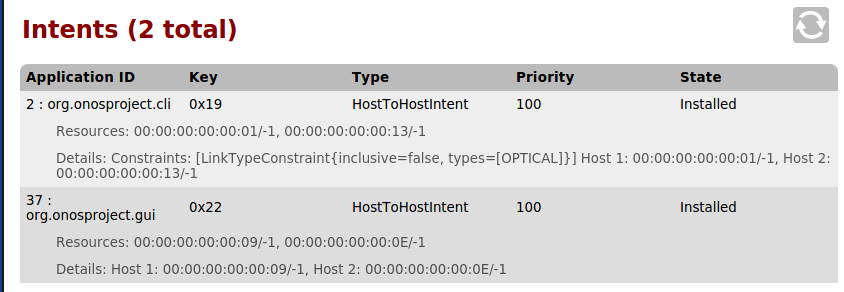
**iteration4.3.Z4.ONOScli.GUI.png**

* 1. **Create Host-to-host FLOW**

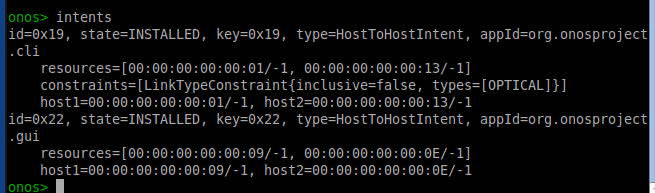
**iteration4.3.Z5.ONOScli.GUI.png**

* 1. **Check that the intent was installed; now you see 2 intents, this is from the GUI**

You now see the second flow, between host 9 and host 14 (0E in hex)



* 1. **You can also check the intent from the ONOS CLI (intent )**

****

* 1. **Now send some traffic on that intent (ping in mininet)**

**iteration4.3.Z6.ONOScli.GUI.png**

**iteration4.3.Z7.ONOScli.GUI.png**





Node 14 goes to 10.0.0.20 which is h42



Node 9 goes to 10.0.0.9 which is h23

One is h42 and one is h23



**Task 4.5 Learn about Advanced and Future SDN Developments CORD**

4.5.1 Take a look at Network Function Virtualization in Wikipedia.

4.5.2 Watch the Network Function Virtualization video (*NFV and SDN; 6 mins*).

4.5.3 Peruse the Nettle site for information on Nettle, a high level language for controlling OpenFlow switches

4.5.4 View the Frenetic site for information on Frenetic, a domain-specific language for programming OpenFlow networks, embedded in Python.

4.5.5 Watch Frenetic Video (*Frenetic: A Programming Language for Software Defined Networks; 38 mins*).

4.5.6 Watch Luxoft Twister Video (*Twister Demo Video; 9 mins*).

4.5.7 Watch Hadoop Acceleration Using OpenFlow Video (*Demo of Hadoop Acceleration using OpenFlow 1.2 based LINC Switch; 6 mins*).

4.5.8 Read about **Google's SDN Network.**

4.5.9 To gain an understanding of the Central Office Re-Architected as a Data Center (CORD) ONOS-AT&T effort, please view the presentation

please view the CORD video at the Open Network Summit.

4.5.1 **Take a look at Network Function Virtualization in Wikipedia**.

1. NFV, no custom hardware, VMS running different software to achieve different tasks
2. No reliance on proprietary or specific hardware
3. Efforts are part of ETSI (Standards organization)
4. Requires management and orchestration layer (ECOMP)
5. Service Channeling – multiple VNFs used in sequence to deliver the service
6. Distributed NFV, locate the NFV in best location, does not need to be in data center
7. Scale up and scale out
8. SDN separates control/data planes (abstraction, control is central, forwarding is distributed)
9. Northbound, communicates with higher lvel applications API
10. Southbound the controller programs the forwarding behavior of the data plane (routers, switches)
11. VFV is NOT dependent on SDN, the work together well
12. Example: the controller controls the distributed data plane and that itself could be visualized
13. Reduces cap-ex and op-ex (save money versus make money)
14. vSwitch provides connectivity VM to VM; and VM to outside world

4.5.2 **Watch the Network Function Virtualization video (NFV and SDN**).

1. <http://www.youtube.com/watch?v=vPqZ0ZQZQ7o> (interview with Dan Pitt)
2. ONF = Open Networking Foundation
3. Saves costs, computing power and electricity
4. Requires cooperation from data plane, this is where SDN comes in with control OpenFlow
5. OpenFlow to control the paths, sends to either real or virtual ports

4.5.3 **Peruse Nettle site for information, a high level language for controlling OpenFlow switches**

1. <http://haskell.cs.yale.edu/other-projects/nettle>
2. Nettles is a domain specific language embedded in Haskell (open source)
3. Haskell is a declarative and expressive language to use to control OpenFlow switches
4. Haskell Programming is a functional reactive language, not like C, Java which are imperative
5. Focus is on what not how (similar to sql or an Excel spreadsheet)
6. Nettles is used for programming network routers, OpenFlow networks in a declarative style
7. Nettles is designed precisely for network applications, “Don’t configure the network, program it”
8. Abandons conventional commercial routers in favor of OpenFlow switches
9. You can download a fully packaged Virtual Machine to use Nettles
10. Nettles layers, OpenFlow, Haskell, HOpenFLow, Functional Reactive Programming, Routing

4.5.4 **View the Frenetic site for information on Frenetic**

**a domain-specific language for programming OpenFlow networks, embedded in Python**.

1. <http://www.frentic-lang.org>
2. Network programming language
3. Domain specific sub language for dataplane processing
4. Frenetic-Ocaml (embedded in OCaml)
5. Pyretic (embedded in python)

Domain specificity means that adaptations evolve to solve problems in particular domains, and therefore are less well suited to solve problems in other domains

4.5.5 **Watch Frenetic Video (Frenetic: A Programming Language for Software Defined Networks)**

1. <https://www.youtube.com/watch?v=mZMy8elvCIo>
2. Professor Jennifer Rexford, Princeton University in conjunction with Cornell
3. Raises the level of abstraction for programming OpenFlow networks
4. Automatically generate switch-level rules that enforce all policies
5. Developing the language Frenetic
6. SDN: logically centralized controller smart slow
7. SDN: controller to API to data plane OpenFlow is the API, to dumb but extremely fast switches
8. OpenFlow drop modify forward, OpenFlow switches can act like firewalls, layer 2, layer 3 etc
9. Controller has the brains that control the OpenFlow networks
10. Simple data plan abstraction
11. Dynamic unfolding as new MAC addresses appear, dynamically adds rules
12. Frenetic provides a query language, similar to sql, a simple level of abstraction, intuitive
13. Minimizes controller overhead
14. OpenFlow API is not modular, things need to happen at the same time, but Frenetic is modular
15. Must think about tasks happening at the same time
16. Major bug – assuming new flow rules hit all the switches at the same time
17. Need to worry about packets “in flight”, hitting switches before the new rules
18. Frenetic has policy updates to handle the in-flight update issues, packet tagging, 2 phase commit
19. FRP = Functional Reactive Programming

4.5.6 **Watch Luxoft Twister Video (Luxoft Corporation) DEMO**

1. [Luxoft Twister Video](http://www.youtube.com/watch?v=QGiDCBPgSeM)
2. Demo video, introducing Twister, an opensource test automation framework
3. Create, edit and run test suites, configure as you wish and create reports
4. Create a project and configure Twister,, looks like an eclipse GUI
5. Configuration option to change db options, define your own schema
6. Define and load customized plugins, and activate them
7. User creates test suite files, similar to jUnit; runs the tests
8. Predefined reports

4.5.7 **Watch Hadoop Acceleration Using OpenFlow Video**

**(Demo of Hadoop Acceleration using OpenFlow 1.2 based LINC Switch)**

1. [Hadoop Acceleration Using OpenFlow V](http://www.youtube.com/watch?v=n4Y1mtX0YPw)ideo
2. No Audio
3. Enable Hadoop perform better with OpenFlow based network switching

4.5.8 **Read about Google's SDN Network.**

1. [**Google's SDN Network**](http://www.networkcomputing.com/networking/inside-googles-software-defined-network/a/d-id/1234201) **(link)**
2. **article 2013, I thought I heard John Donovan say google is no longer player?**
3. **Google calls its SDN network B4, its own network, 10 years in the making**
4. At the start, Google built its own switches (silicon)
5. Google will use OpenFlow for flow instantiation
6. Hybrid implementation strategy (coexist with legacy)
7. Clos topology, collection of small cheap switches are grouped into a much larger switch
8. Internally written centralized software control stack, centralized traffic engineering
9. Custom protocols tailored to its data centers

4.5.9 **To gain an understanding of the Central Office Re-Architected as a Data Center (CORD) ONOS-AT&T effort, please view the** [**presentation**](http://onrc.stanford.edu/protected%20files/PDF/ONRC-CORD-Larry.pdf)**. In addition, please view the** [**CORD video**](https://vimeo.com/131936192) **at the Open Network Summit. CORD= Central Office Re-Architected as a Data Center**

**CORD Video: (Open Network Summit DEMO 2015)**

1. **Control plane is controlled by ONOS**
2. **XOS is orchestration**
3. **RADIUS hold information on subscriber information**
4. **3 types of users, Subscriber, Service Provider, third party content provider**
5. **Subscriber: view home network, can login to subscriber portal, see users in home network**
6. **Subscriber: can purchase to give control his home network, block his kid’s facebook etc**
7. **Content Provider: cache content closer to customer**
8. **vCPE = set of VM connected to private network**
9. **runs on top of ONOS operating system (for networks)**

**CORD Presentation**

1. [presentation](http://onrc.stanford.edu/protected%20files/PDF/ONRC-CORD-Larry.pdf) (link)
2. Dealing with thousands of central offices
3. CORD is SDN, NFV and Cloud
4. Reduce CAPEX and OPEX
5. Elastic scaling
6. Virtualizing existing hardware
7. SDN control App running on ONOS, host control apps and switching
8. Everything as a service
9. Access as a service ACCaaS = SDN
10. Subscriber as a Service SUBaaS = NFV
11. Internet as a Service INTaaS = SDN
12. Content Distribution Network (CDN) = Cloud
13. **XOS management of Services XaaS VNF manager**