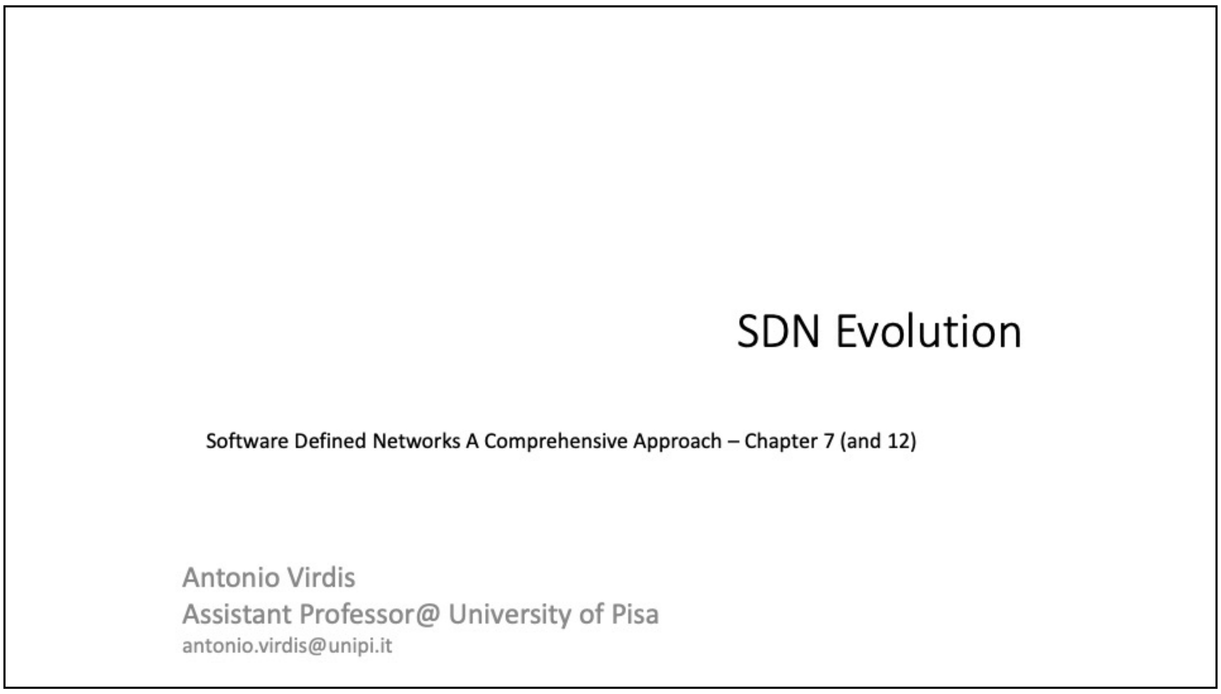


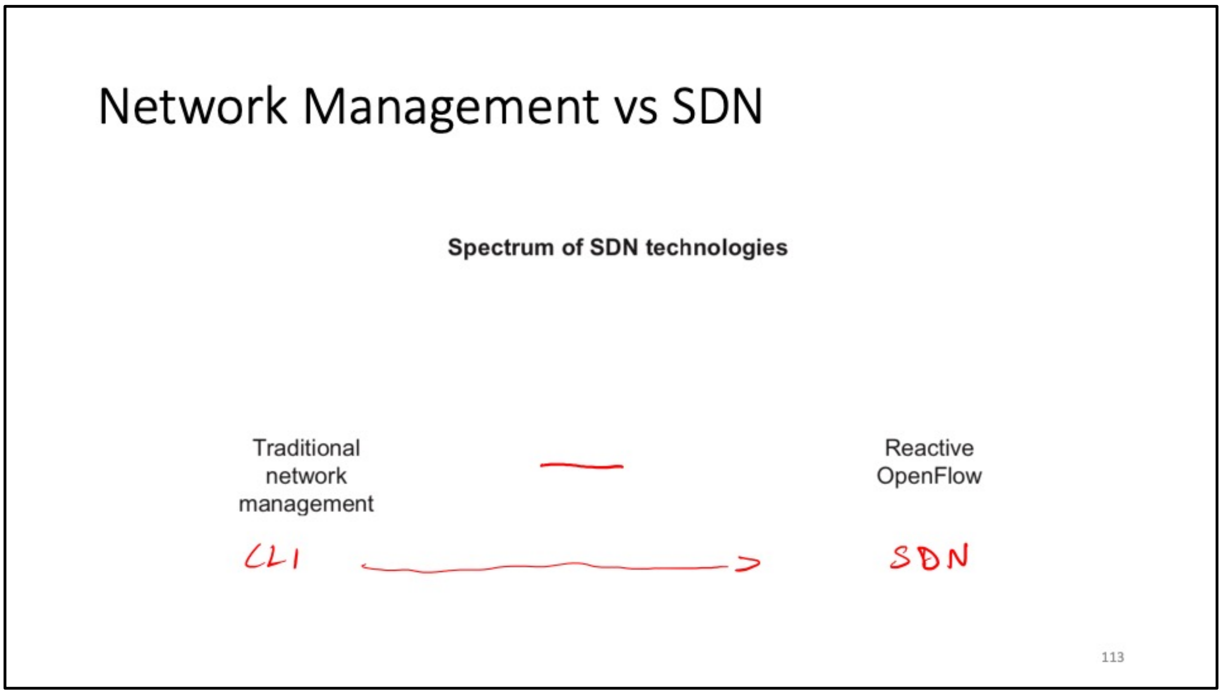
SDN Evolution

Software Defined Networks A Comprehensive Approach – Chapter 7 (and 12)

Antonio Virdis
Assistant Professor@ University of Pisa
antonio.virdis@unipi.it

In the earlier days of SDN, the networking vendors most involved were those more interested in industry disruption.

- Therefore, companies such as Hewlett-Packard, IBM, and NEC, as well as a number of startups, developed solutions oriented around Open SDN and the OpenFlow protocol.
- In the past few years, as major networking incumbents entered the SDN marketplace, we have begun to see changes.
- When SDN began to take root and major vendors such as Cisco and Juniper took notice, this landscape and even the very definition of SDN began to shift.



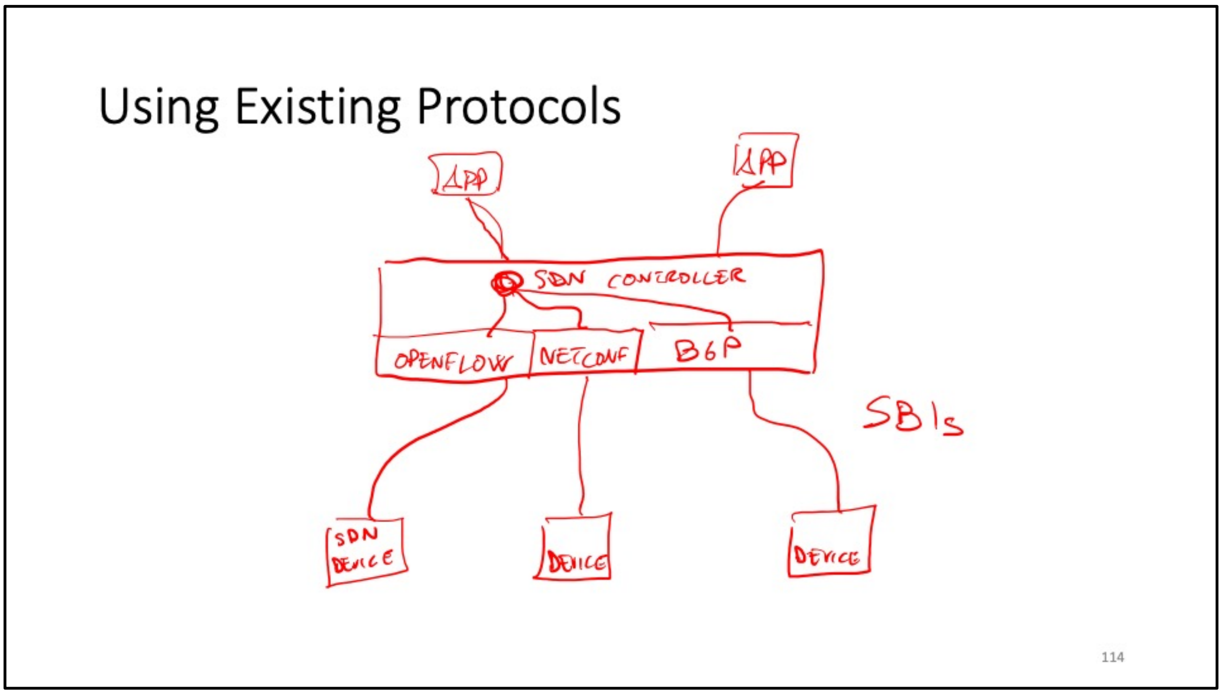
Network Management vs SDN

Spectrum of SDN technologies

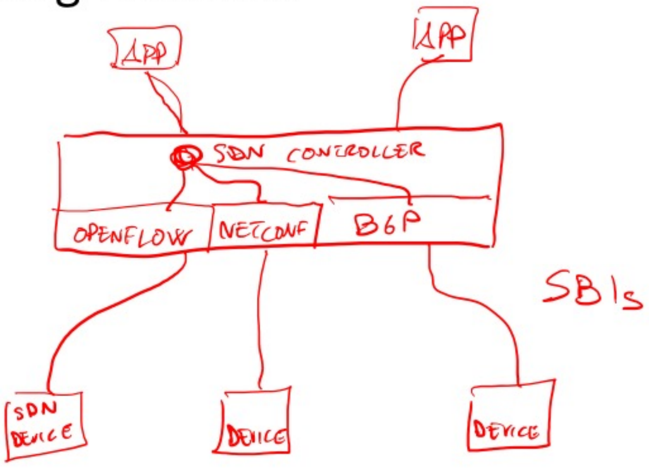
Traditional network management —— Reactive OpenFlow

CLI ⟶ SDN

it is difficult to precisely circumscribe SDN. For the purposes of this discussion, since network management in general shares many of the same attributes as SDN (i.e., centralized control, network-wide views, network-wide policies), we consider such network management-based solutions to also fall under the larger SDN umbrella.

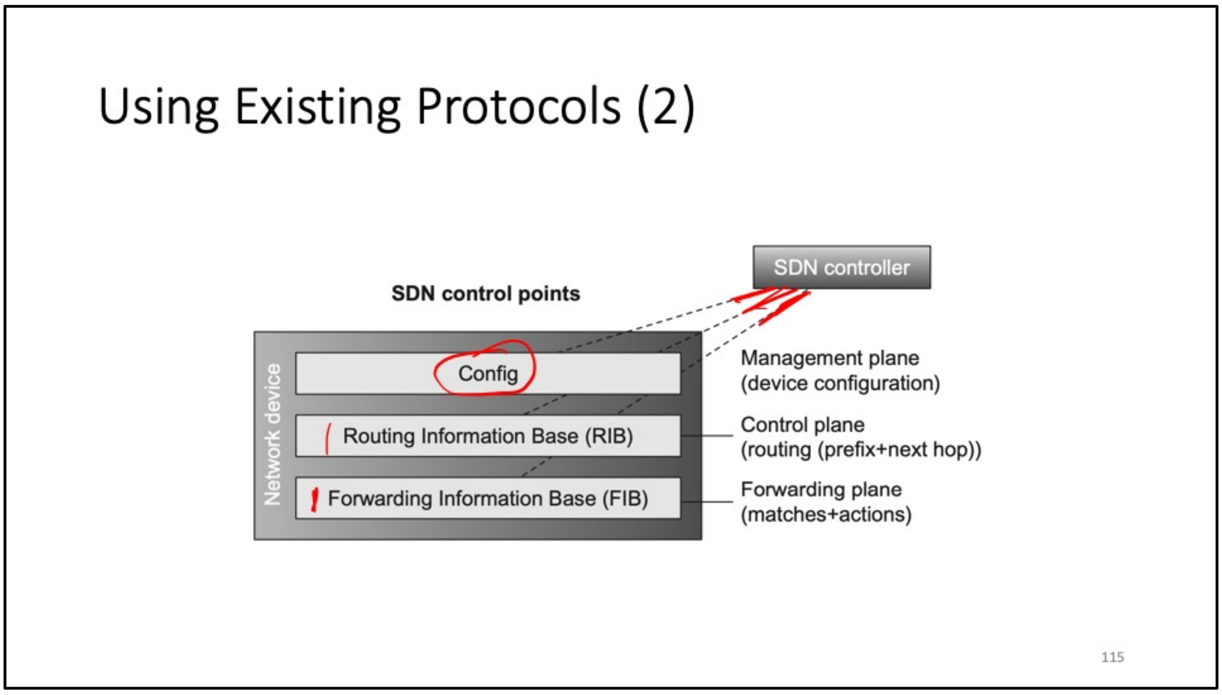


Using Existing Protocols

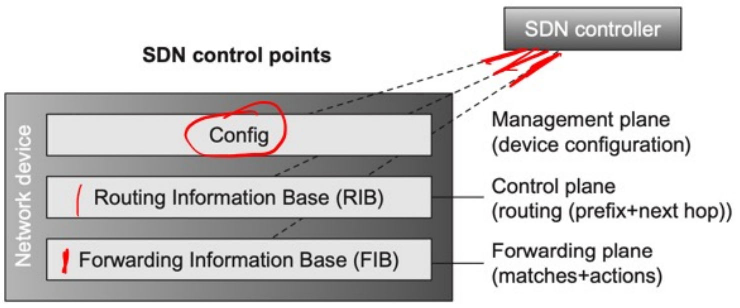


114

While the high-level architecture remains unchanged, the protocols (SBI and NBI) as well as the controller's behaviour change.

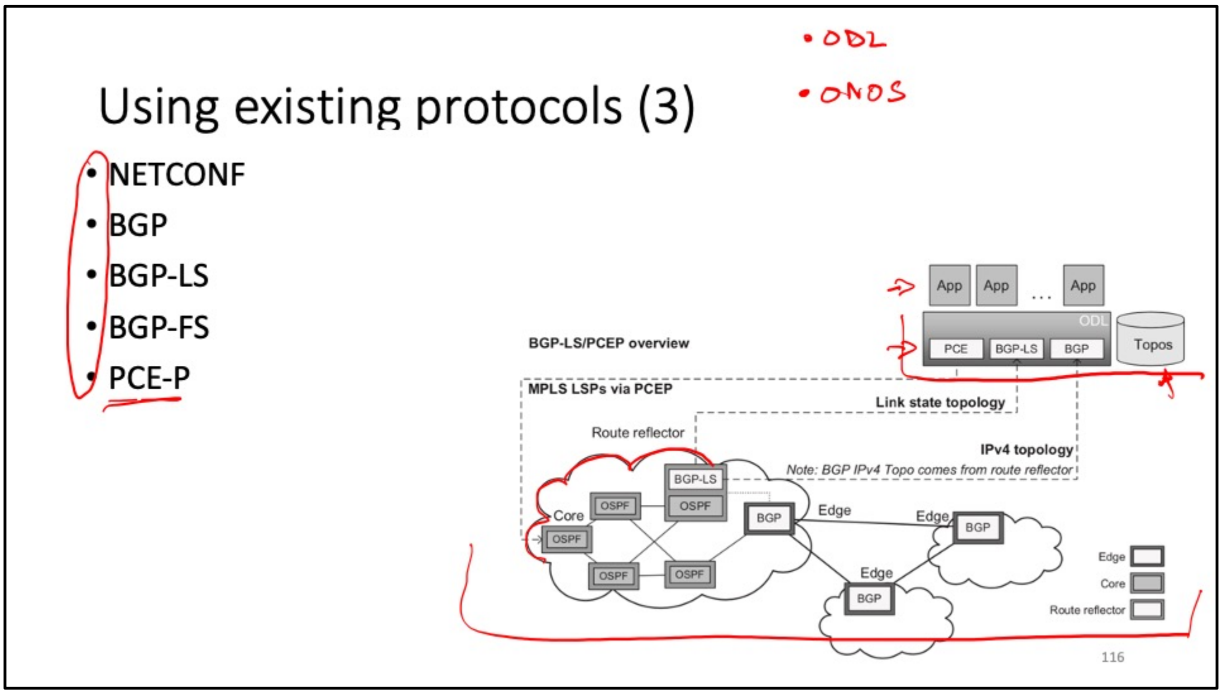


Using Existing Protocols (2)



115

As we consider this use of existing protocols, a helpful perspective may be to look at the different control points shown in Figure that are managed and configured by the SDN application. These control points are *Config*, where general configuration is done, *Routing Information Base* (RIB), where routes (e.g., prefixes and next-hops) are set, and *Forwarding Information Base* (FIB), which is lower level and can be considered *flows*, where packet headers are matched and actions are taken.

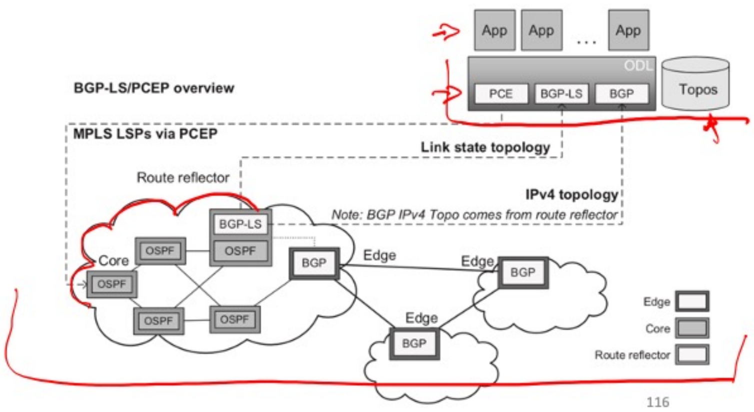


Using existing protocols (3)

• ODL

• ONOS

- NETCONF
- BGP
- BGP-LS
- BGP-FS
- PCE-P



116

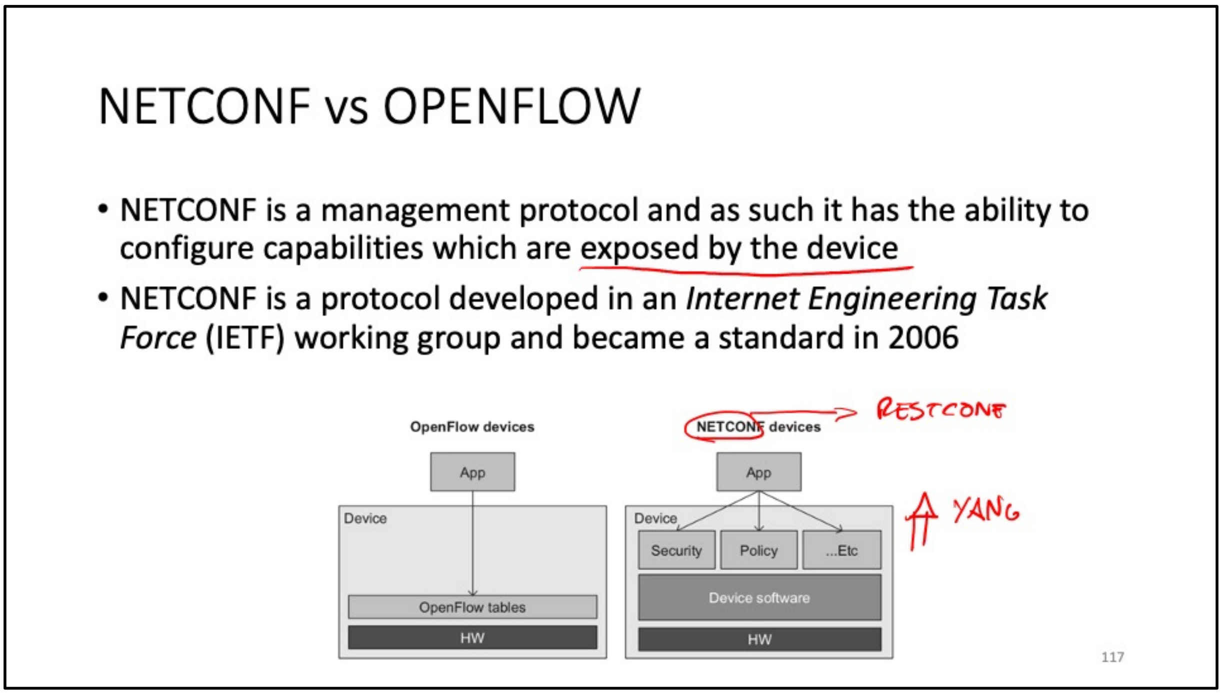
NETCONF's role is for setting configuration parameters.
BGP is involved in setting RIB entries
PCE-P is used for setting MPLS paths through the network.
BGP- LS is used to gather topology information from the RIB.
BGP-FlowSpec (BGP-FS) is employed to set matches and actions, similar to what is done with OpenFlow, using instead the BGP-FS functionality of the router. BGP-FS leverages the BGP *Route Reflection* infrastructure and can use BGP *Route Targets* to define which routers get which routes. Unlike OpenFlow, BGP-FS does not support layer 2 matches but only layer 3 and above.

w.r.t. the figure:

BGP-LS: The BGP-LS protocol is used by ODL to gather link state topology information from the routing protocols running in the clouds in the figure. This topology reflects routers and interconnecting links within the *Open Shortest Path First* (OSPF) or *Intermediate System to Intermediate System* (IS-IS) domains.

BGP: The BGP protocol is used by ODL to gather IP *Exterior Gateway Protocol* (EGP) topology from the BGP routers connecting the clouds (domains) in the picture.

PCE-P: The PCE-P protocol is used by ODL to configure MPLS *Label Switched Paths* (LSPs) for forwarding traffic across those networks.



NETCONF vs OPENFLOW

- NETCONF is a management protocol and as such it has the ability to configure capabilities which are exposed by the device
- NETCONF is a protocol developed in an *Internet Engineering Task Force* (IETF) working group and became a standard in 2006

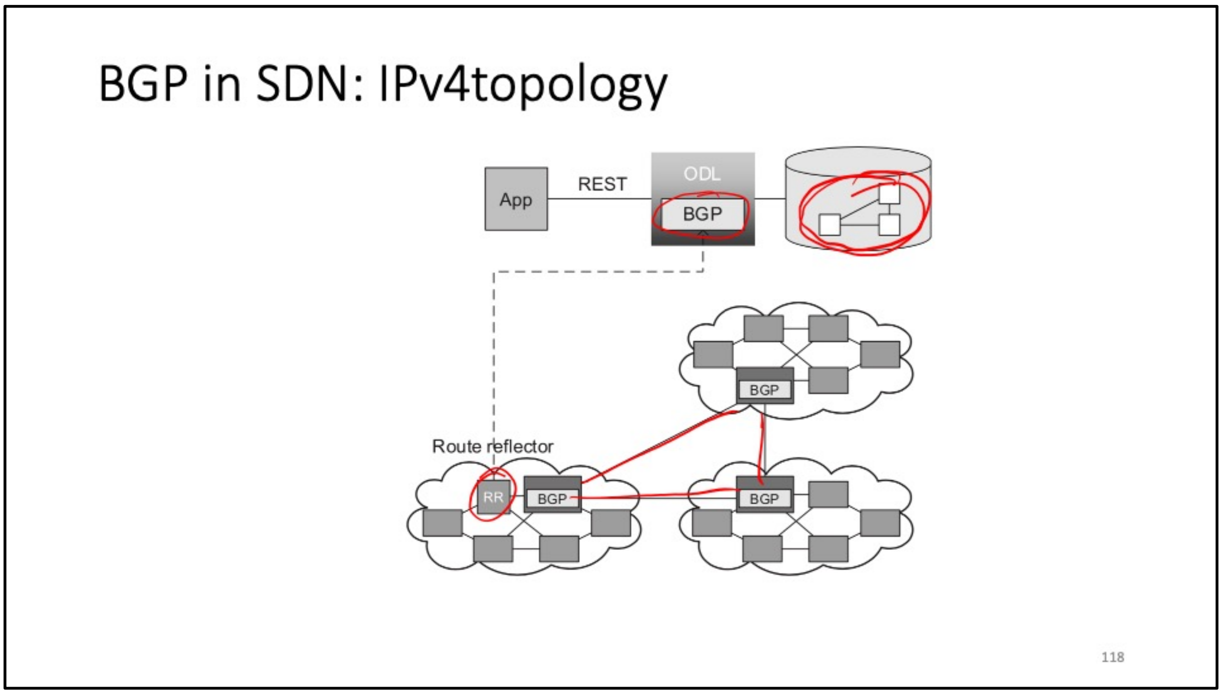
OpenFlow devices
App
Device
OpenFlow tables
HW
NETCONF devices
RESTCONF
App
Device
Security
Policy
...Etc
Device software
HW
YANG
117

OpenFlow configures the lower levels of the networking device, that is, the ASIC containing the TCAM. This entails setting the matches and actions of the FIB.

NETCONF, on the other hand, performs traditional configuration of the device but via the NETCONF protocol rather than via the CLI or SNMP. The SDN application developer is limited by what the device exposes as configurable. If the device exposes NETCONF data models that allow for the configuration of *Access Control Lists* (ACLs), then the application can configure those. Similarly, if the device has data models for configuring *Quality of Service* (QoS) or static routes, then those will be possible as well.

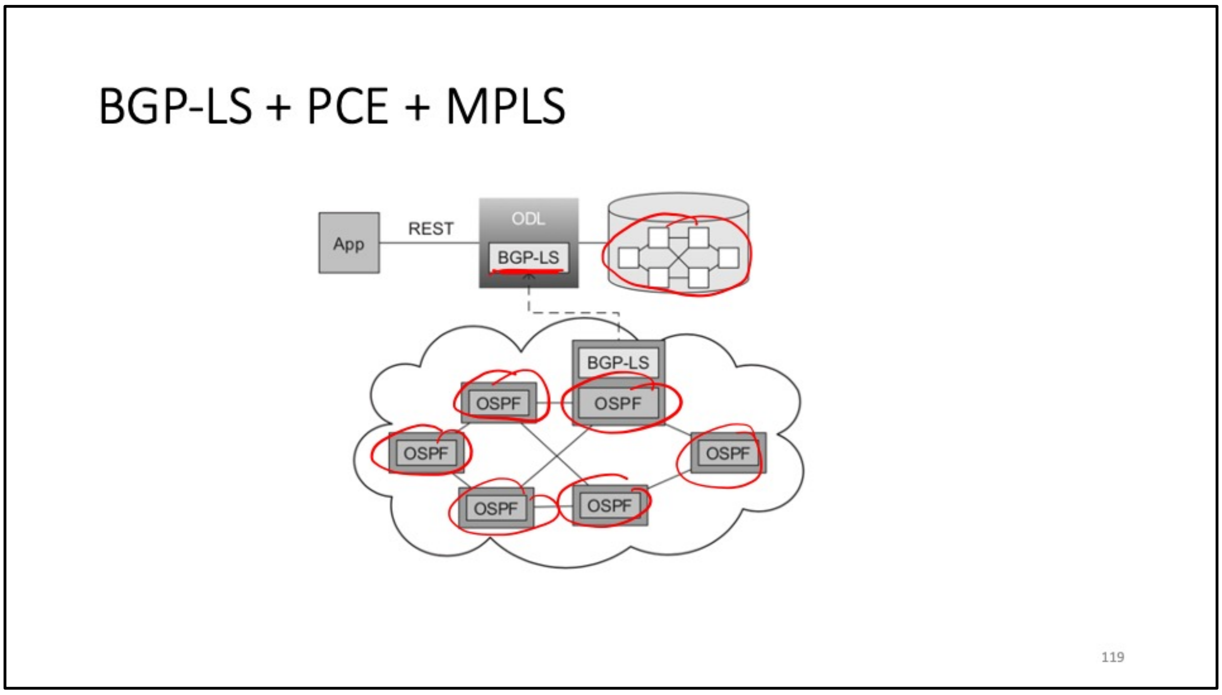
Protocols as NETCONF become useful via the data models which convey information and requests to and from the device. **YANG** provides a standardized way for devices to support and advertise their capabilities. One of the first operations that takes place between a NETCONF client on the controller and a NETCONF server running on the device is for the device to inform the client which data models are supported. This allows the SDN application running on the controller to know which operations are possible on each device.

SDN controllers often support *REST-based NETCONF* (**RESTCONF**) as a mechanism for communicating between the controller and devices. RESTCONF works like a general REST API in that the application will use HTTP methods such as GET, POST, PUT, and DELETE in order to make NETCONF requests to the device.

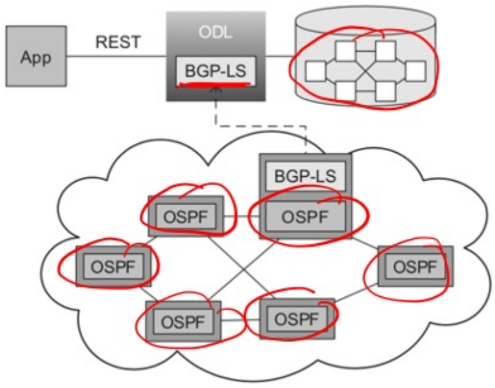

BGP in SDN: IPv4topology

The BGP plugin running inside ODL is implementing an actual BGP node, and as such it has access to topological information via the *Route Reflector* (RR). This information provides the topology between devices implementing the EGP, often referred to as the IPv4 topology.
The figure shows an EGP network with routers supporting BGP, and an RR communicating topology information to the BGP node running inside the ODL controller. This information helps to provide the network-wide views characteristic of SDN solutions, and it can be used to dynamically configure intelligent routing paths throughout the network. Within ODL there are APIs for creating a RIB application which can be used to inject routes into the network. A key aspect of this technique is that the ODL's controller's BGP plugin appears to the network as a normal BGP node. Note that the plugin does not advertise itself as a next hop for any of the routes. It will, however, effect changes on the adjacent nodes that will in turn propagate routing information throughout the network. In this way, an SDN application running on ODL can force RIB changes throughout the network, thereby achieving SDN-like agility and automatic network reconfiguration.

Route reflector: have one or more iBGP speakers act as *concentration* routers, commonly known as *route reflectors*. The introduction of route reflectors then creates a hierarchy among the iBGP speakers by *clustering* a subset of iBGP speakers with each route reflector.

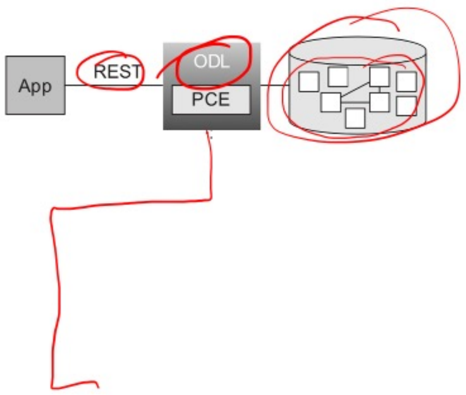


BGP-LS + PCE + MPLS

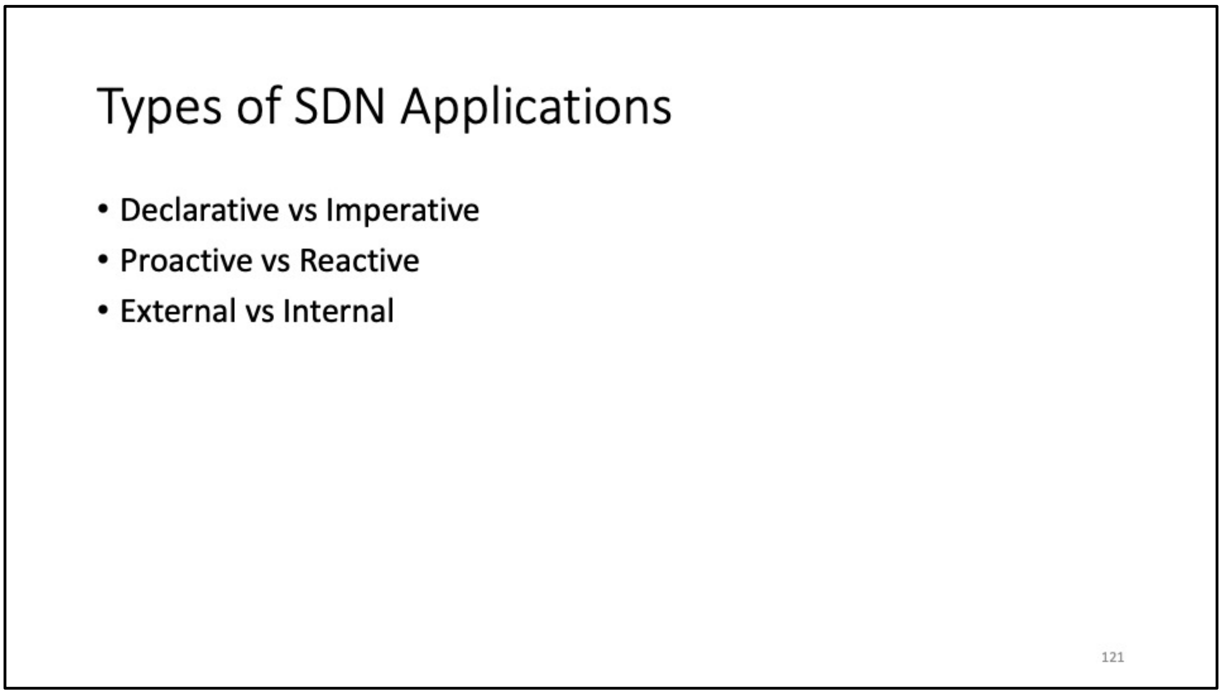


119

BGP-LS + PCE + MPLS (2)



120



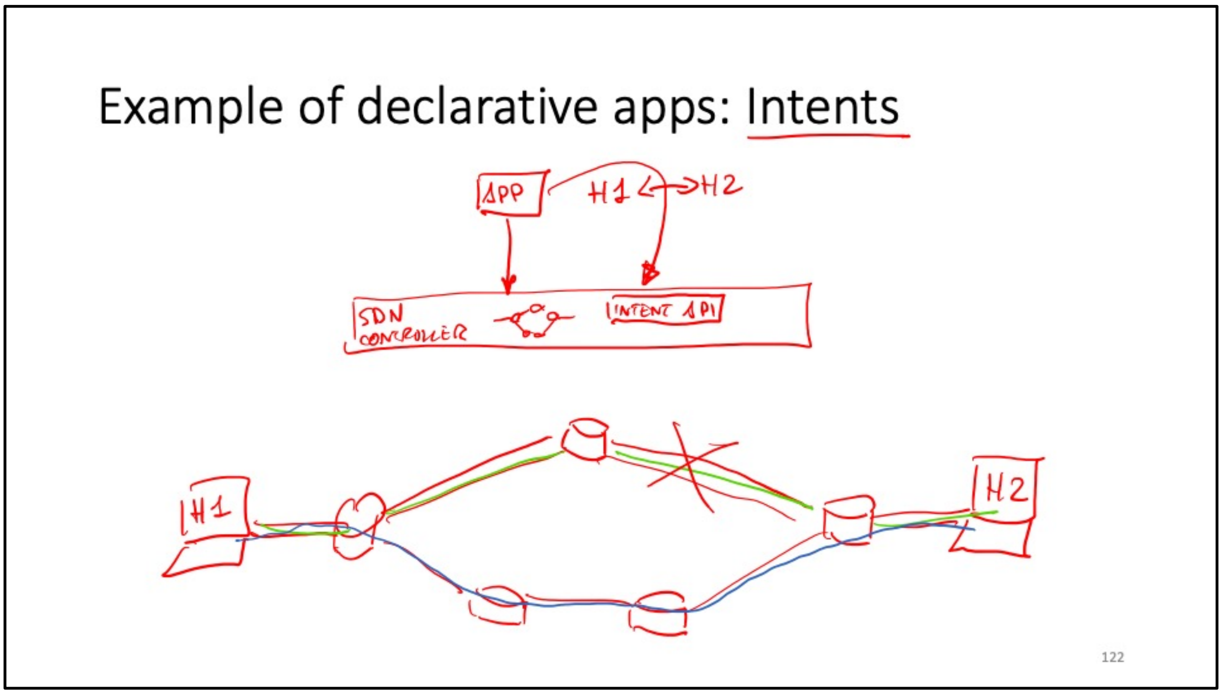
Types of SDN Applications

- Declarative vs Imperative
- Proactive vs Reactive
- External vs Internal

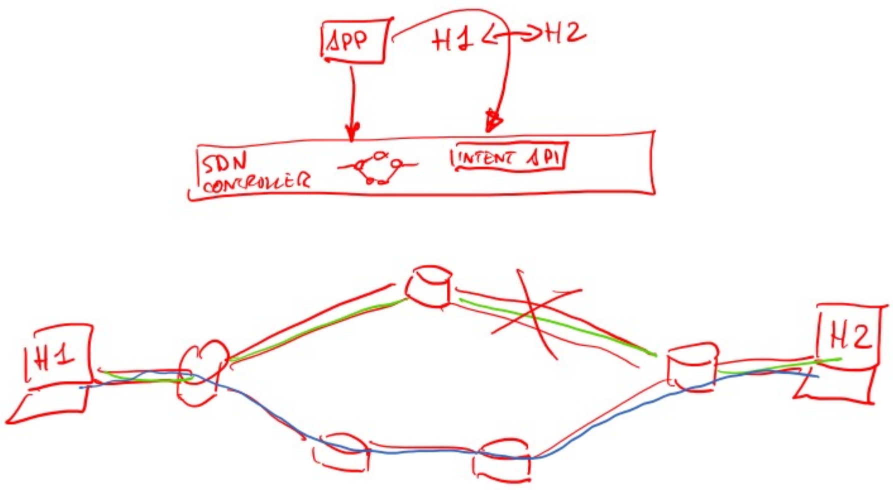
121

An **imperative** system requires that it be told exactly *how* to do something. The net result of all the *hows* is that the system will achieve the *what* that was intended in the first place.

A **declarative** system needs only be told *what* needs to be done; the system itself figures out *how* to accomplish the goal.



Example of declarative apps: Intents

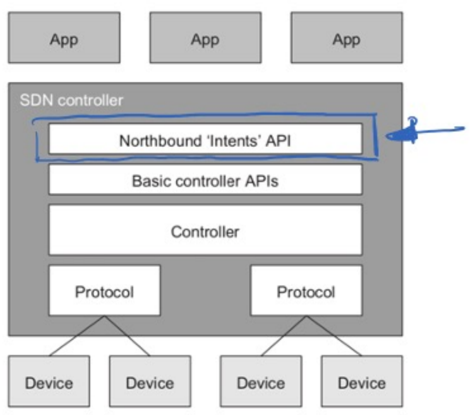


122

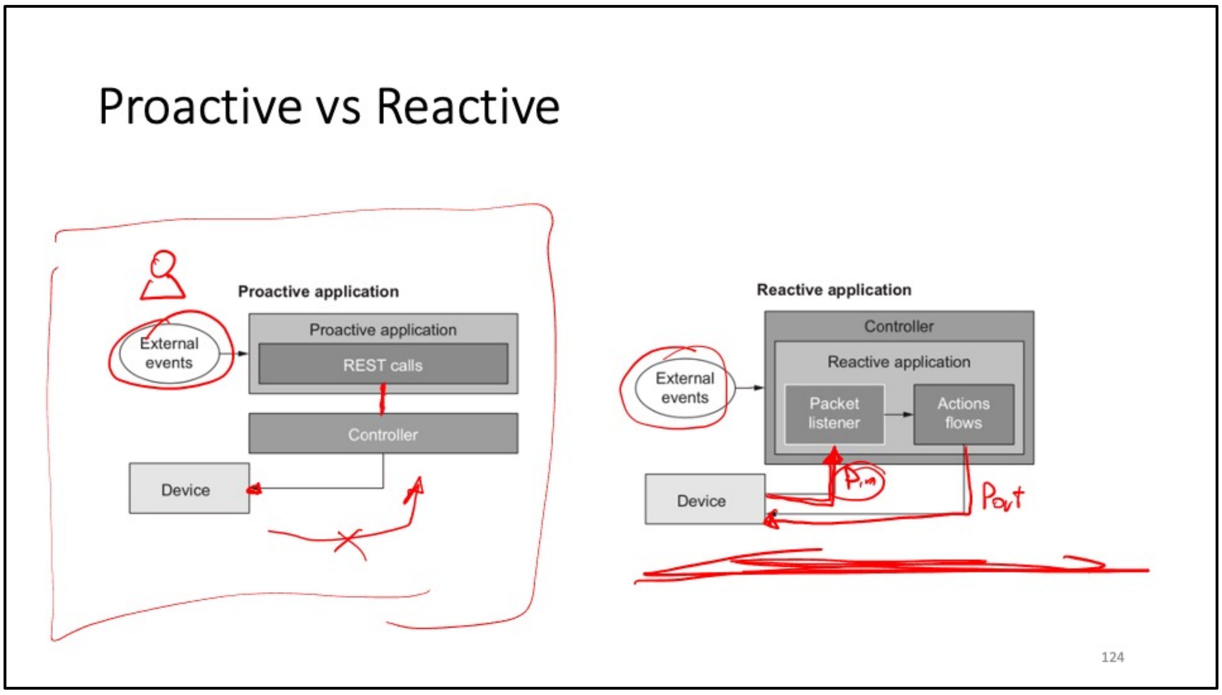
As an example, an application wishing to provide some level of connectivity between two hosts A and B, using existing base controller APIs would program flows on switches all across the topology, from the switch connecting A to the network, to the switch connecting B to the network, including all switches along the path between the hosts. This clearly increases the complexity of the SDN application.

On the other hand, an intents-based API would allow the application to specify something more abstract, such as *the desire to allow host A to communicate with host B*. Invoking that API would result in the SDN controller performing flow programming of the switches on the path from host A to host B, assuming that OpenFlow was the SDN protocol in use. If some other type of protocol was being used, this would not perturb the application, since the controller would be responsible for making that connectivity happen with whatever southbound API was appropriate.

Intent-based operations

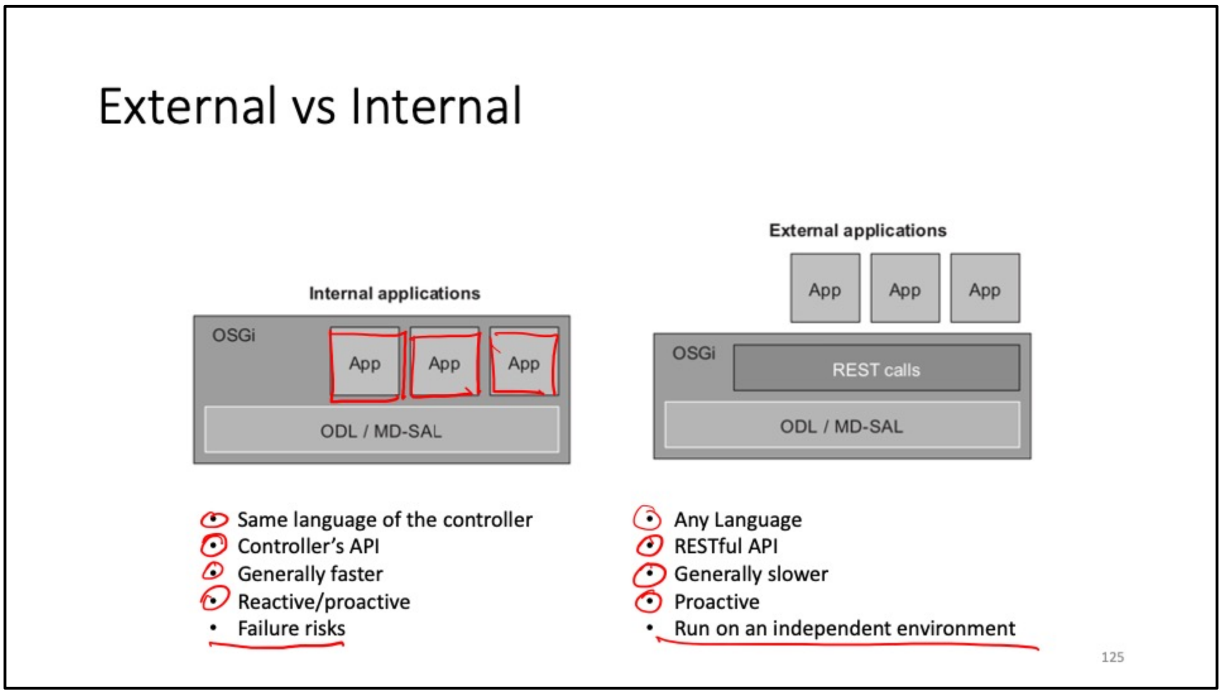


123

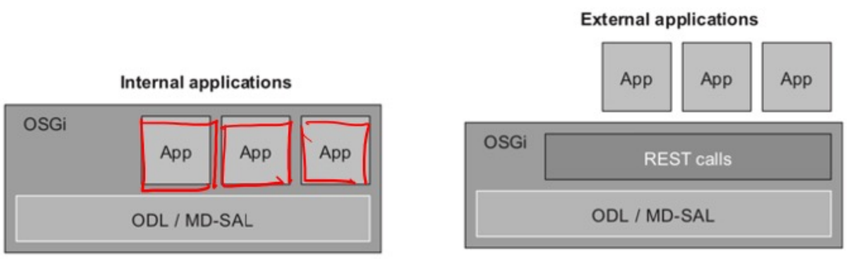


Proactive vs Reactive

By a reactive approach, we mean that when the first packet of a flow arrives at a switch, the controller is contacted to insert a new flow entry in the flow table. That is, the flow entries are inserted on demand and deleted when the flow is over. While this suggests that a flow table would be used efficiently, there is, however, an overhead in terms of the flow set up time. Firstly, if a switch loses its connection to the controller, then a switch is not able to accommodate a new flow. With a proactive approach, the controller computes the flow table entries and they are inserted at the switches ahead of time. This eliminates any set up time. Secondly, a proactive approach, by its nature, sets up flow entries in the flow table based on wildcard rules. Thirdly, if a switch loses the connection to the controller, it can still work independently to allow traffic movement through the network. As you can see, the aggregated routing goes well with the proactive approach.



External vs Internal



- Same language of the controller
- Controller's API
- Generally faster
- Reactive/proactive
- Failure risks

- Any Language
- RESTful API
- Generally slower
- Proactive
- Run on an independent environment

125

Ref [1] – chapter 12

Internal applications must be written in Java (assuming the controller's native language is Java, which is true of almost all controllers); external applications can be written in any language.

Internal applications will use Java APIs on the controller; external applications will use RESTful APIs.

Internal applications must be deployed inside the environment (typically OSGi) of the controller, running locally with the controller; external applications can run anywhere, even in the cloud if so desired.

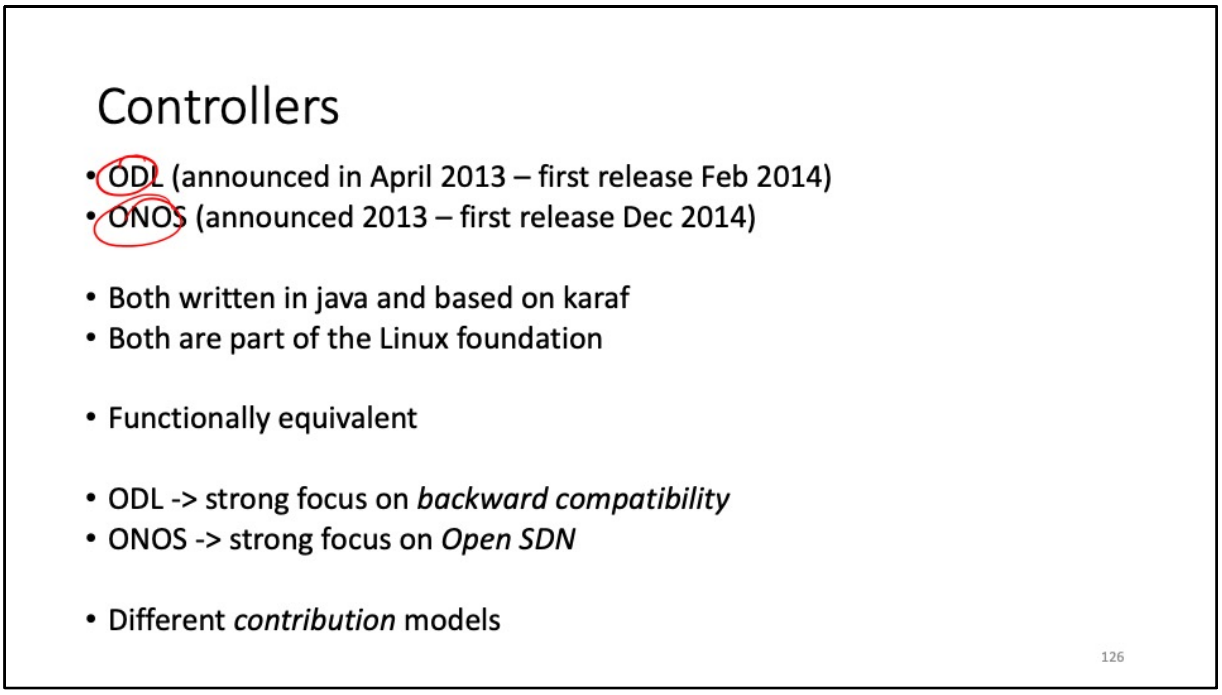
Internal applications are relatively faster; whereas external applications are slower due to: (1) using the REST API, (2) possibly being written in a slower language such as Python, and (3) potentially running remotely from the controller.

Internal applications are capable of running reactive applications (receiving packets forwarded from an OpenFlow switch); external applications are not.

Both internal and external applications are capable of running proactive applications.

Internal applications are running inside the controller, and failures can have negative effects on the

operation of the controller; for external applications, this danger is less severe.



Controllers

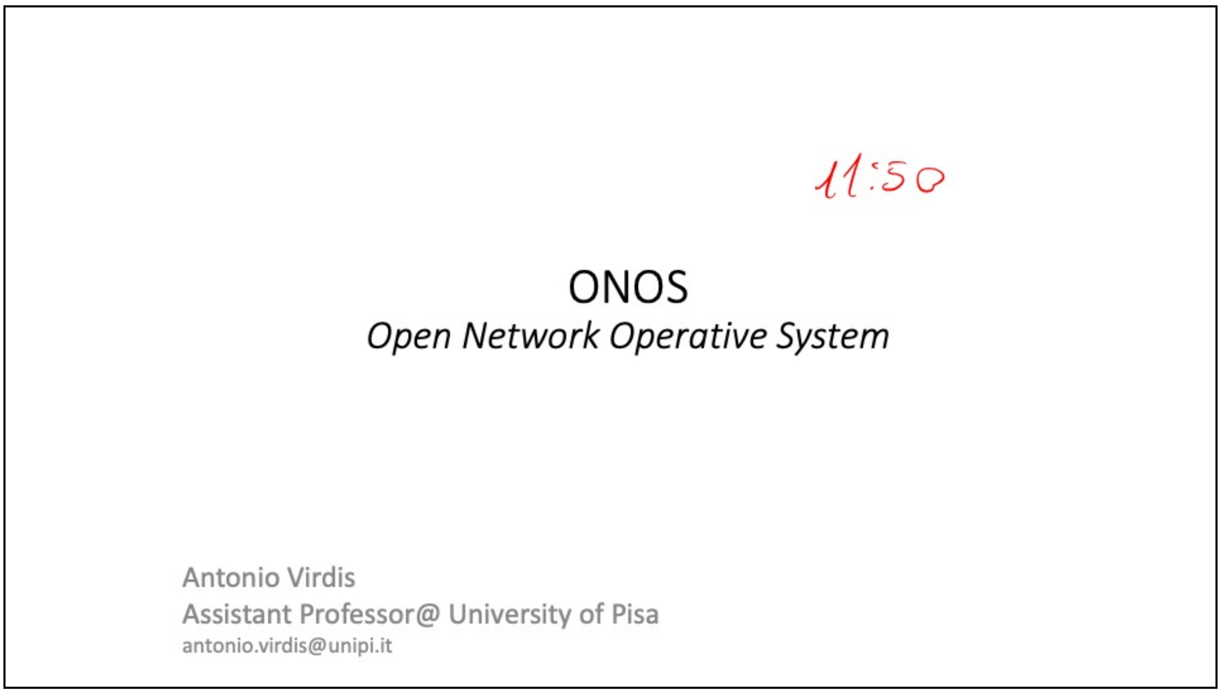
- ODL (announced in April 2013 – first release Feb 2014)
- ONOS (announced 2013 – first release Dec 2014)
- Both written in java and based on karaf
- Both are part of the Linux foundation
- Functionally equivalent
- ODL -> strong focus on *backward compatibility*
- ONOS -> strong focus on *Open SDN*
- Different *contribution* models

126

Both ONOS and ODL are written in Java and designed for modular use with a customizable infrastructure–and it is important to note that every ONOS partner is also an ODL member.

Both are projects under the umbrella of the Linux foundation

ODL is in production in more than 150 organizations around the world–from large telecom carriers collectively supporting 1 Billion subscribers, to web-scale content providers to enterprises, research orga- nizations and universities.

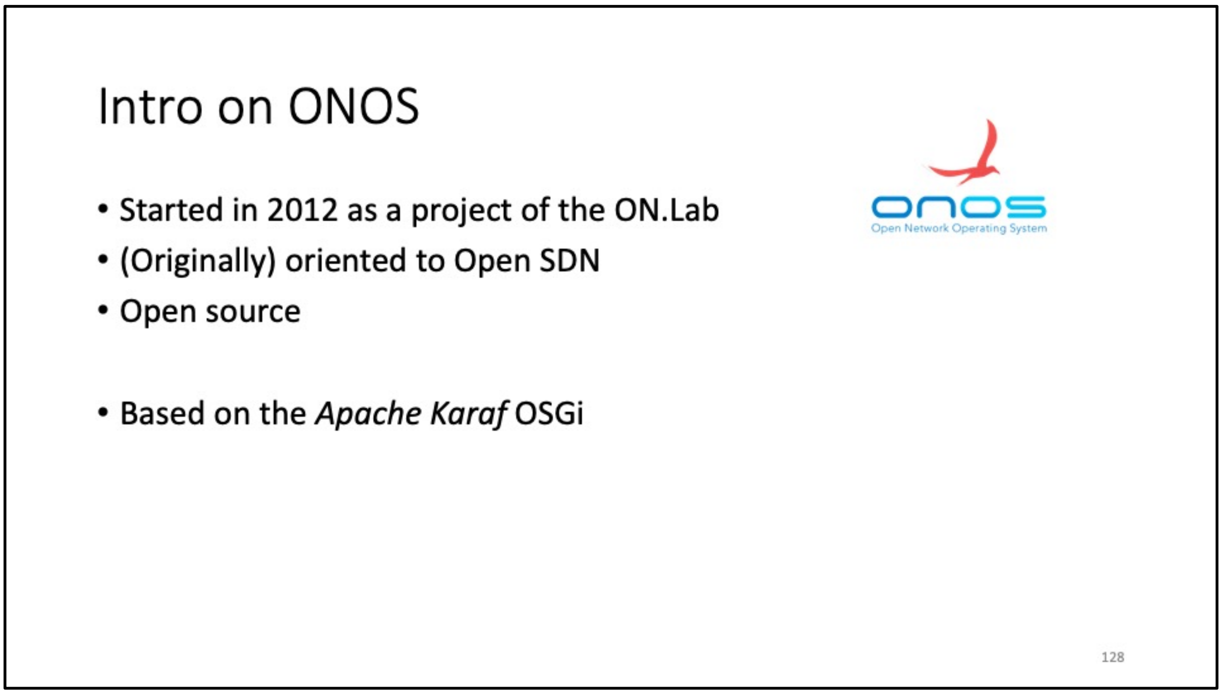


11:50

ONOS

Open Network Operative System

Antonio Virdis
Assistant Professor@ University of Pisa
antonio.virdis@unipi.it



Intro on ONOS

- Started in 2012 as a project of the ON.Lab
- (Originally) oriented to Open SDN
- Open source

- Based on the *Apache Karaf* OSGi

onos
Open Network Operating System
128

The Open Networking Foundation (ONF) is an operator led consortium spearheading disruptive network transformation. the ONF has now merged operations with ON.Lab to create a single organization driving vast transformation across the operator space

[ref1]

11.9.3 ONOS

ONOS is a recent arrival on the SDN scene, but its presence has drastically altered the SDN controller landscape compared to just a few years ago. While many controllers have played their part in the evolution of SDN, the only two significant open source controllers as of this writing are ODL and ONOS. As can be seen in Fig. 11.2, ONOS has attracted more board participation from service providers and NEMs focused on that market than does ODL. ONOS is firmly committed to the principles of Open SDN, which is not strictly true of ODL. Whereas ODL has silver, gold, and bronze levels of participation and corresponding influence, members' influence on ONOS is ostensibly a function of their level of participation rather an amount paid for membership. The ONOS mission is heavily influenced by one of its founding entities, ON.Lab, which, at its core, is a research organization. As we said in Section 11.1.1, Guru Parulkar, the Executive Director of ON.Lab and now ONOS, is strongly opposed to the dilution of SDN from its founding principles. The mission statement *To produce the Open Source Network Operating System that will enable service providers to build real Software Defined Networks* [23], clearly exposes the service provider orientation of this group. ONOS emphasizes the use of OpenFlow and the high reliability features that are required for service provider use. Unlike ODL, it does not prioritize the repurposing of legacy equipment and protocols to achieve SDN goals. Since their controller software was first open-sourced in 2014, the ONOS community has rapidly risen to become an influential player with respect to SDN controllers. We provide more details about ONOS in Section 13.8.3.

12.7.2 ONOS

ONOS is a relative newcomer, having first been released in late 2014, but it has gained traction with service providers such as AT&T [13] and NEMs like Ciena [14] and Fujitsu [15]. As such, ONOS warrants consideration by prospective SDN application developers.

OpenFlow: An SDN network engineer should consider ONOS if his/her interests lie primarily in creating OpenFlow-based applications, as this is the focus of ONOS. ONOS is a project emanating from the research communities at Stanford and other universities, with more of a focus on OpenFlow-based experimentation and development.

Service provider: ONOS is explicitly targeted at service providers, and has gone to lengths to recruit them to join the ONOS project.

Intents: For the developer interested in creating applications which reside above a layer of intents-based abstraction, ONOS has been built with this type of API in mind and provides a rich environment for applications that operate at this level.

ONOS offers both RESTful APIs for creating external applications, as well as Java APIs for internal application development. Both the RESTful APIs and Java APIs follow the general models that have evolved alongside with the Beacon, Floodlight, and HP VAN SDN controllers, and their derivatives. SDN developers familiar with those APIs will find similarities with the ONOS APIs. However, ONOS additionally provides intents-based APIs, which are new to SDN in general, and will require some amount of adjustment to this new, albeit improved, model.

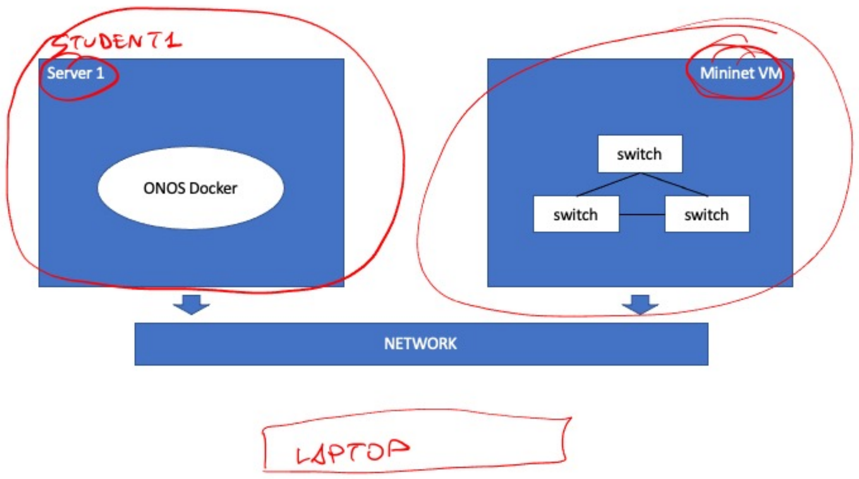
13.8.3 ONOS

As of this writing, the only true challenger to ODL's dominance is ON.Lab's ONOS project. Many of the same major service providers and transport vendors investing in ODL development are also investing in ONOS. Indeed many of these companies were instrumental in the very formation of ONOS as a competitor to ODL. It is thus unsurprising that ONOS has quickly risen to become a major factor in the debate over open source SDN controllers.

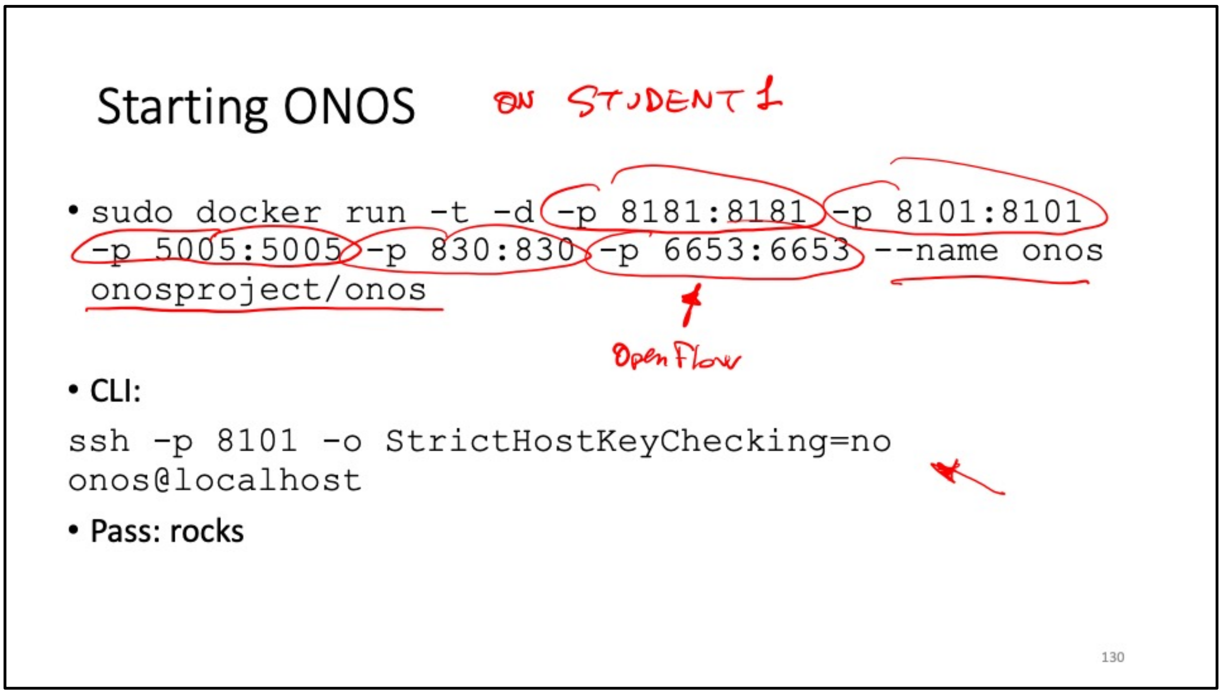
Like ODL, ONOS is supported by a number of major vendors and service providers, including Alcatel-Lucent, AT&T, China Unicom, Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC, NTT Communications, among others. The Linux Foundation has also entered into a strategic partnership with the ONOS project. Hence the foundation supports *both* ODL and ONOS. The two controllers have fundamentally different focus areas. ODL emphasizes legacy protocols while ONOS emphasizes OpenFlow.

One way to compare activity levels of ODL versus ONOS is to look at the amount of code and the number of contributors to their respective open source projects. BlackDuck *Open HUB* [18] tracks open source projects, calculating statistics such as those cited previously. As of the time of this writing, the numbers for both projects *for the preceding 12 months* are shown in Table 13.7. As is apparent from the table, ODL has a significant lead in number of contributors and lines of code, although it is also true that ONOS has been around for less time than ODL.

Lab Configuration



129



Starting ONOS

ON STUDENT 1

- sudo docker run -t -d -p 8181:8181 -p 8101:8101 -p 5005:5005 -p 830:830 -p 6653:6653 --name onos onosproject/onos

OpenFlow

- CLI:

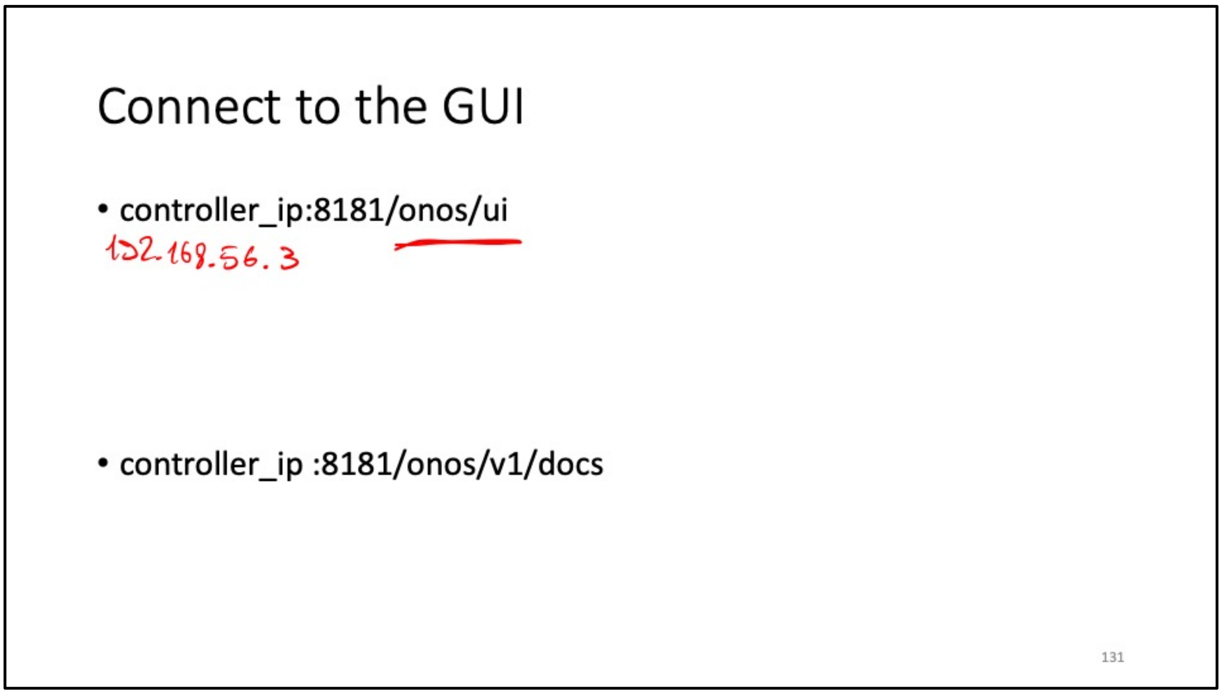
```
ssh -p 8101 -o StrictHostKeyChecking=no
onos@localhost
```

- Pass: rocks

130

```
sudo docker kill onos ; sudo docker rm onos ; ssh-keygen -f "/home/ubuntu/.ssh/known_hosts" -R "[localhost]:8101"

sudo docker run -t -d -p 8181:8181 -p 8101:8101 -p 5005:5005 -p 830:830 -p 6653:6653 --name onos onosproject/onos
ssh -p 8101 -o StrictHostKeyChecking=no onos@localhost
```



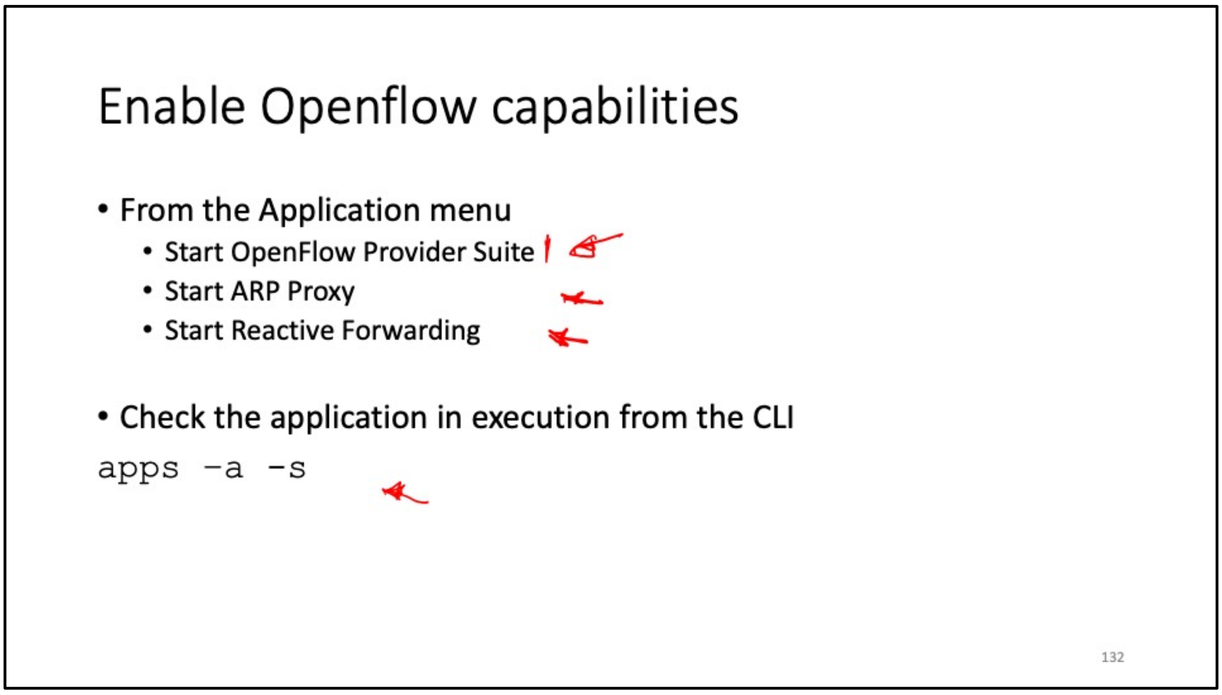
Connect to the GUI

- controller_ip:8181/onos/ui

192.168.56.3

- controller_ip :8181/onos/v1/docs

131



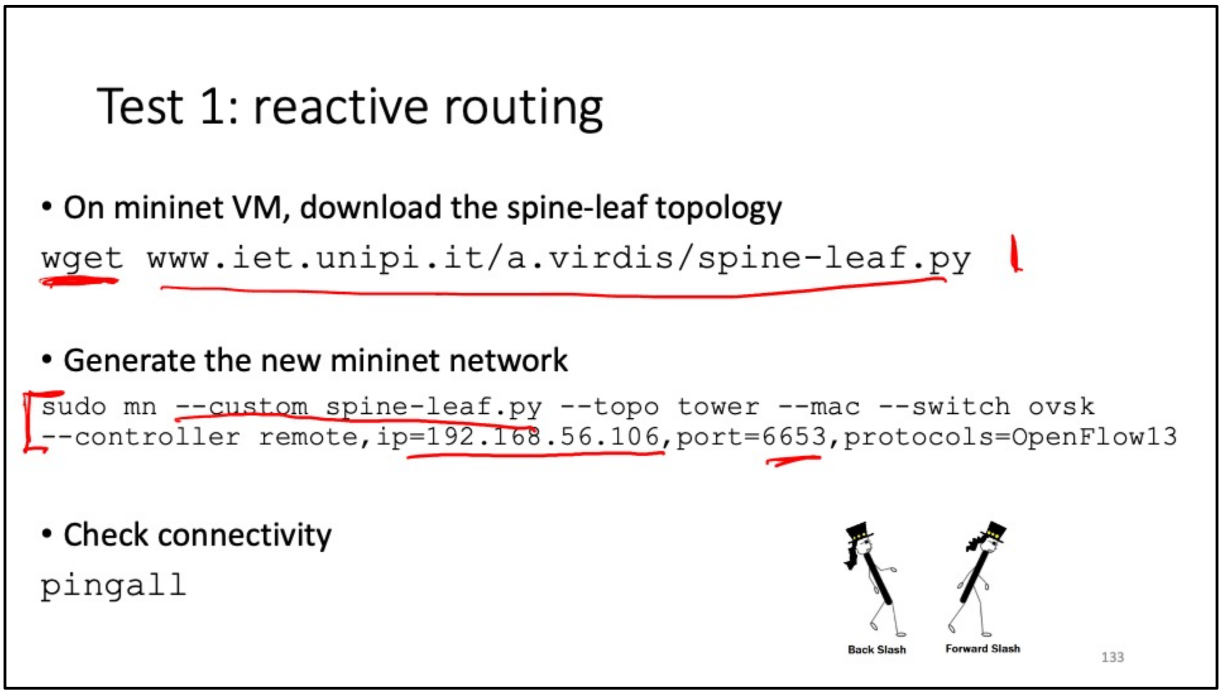
Enable Openflow capabilities

- From the Application menu
  - Start OpenFlow Provider Suite !
  - Start ARP Proxy
  - Start Reactive Forwarding
- Check the application in execution from the CLI

```
apps -a -s
```

132

check with
apps -a -s



Test 1: reactive routing

- On mininet VM, download the spine-leaf topology

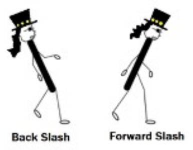
```
wget www.iet.unipi.it/a.virdis/spine-leaf.py
```

- Generate the new mininet network

```
sudo mn --custom spine-leaf.py --topo tower --mac --switch ovsk
--controller remote,ip=192.168.56.106,port=6653,protocols=OpenFlow13
```

- Check connectivity

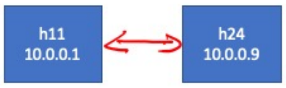
```
pingall
```

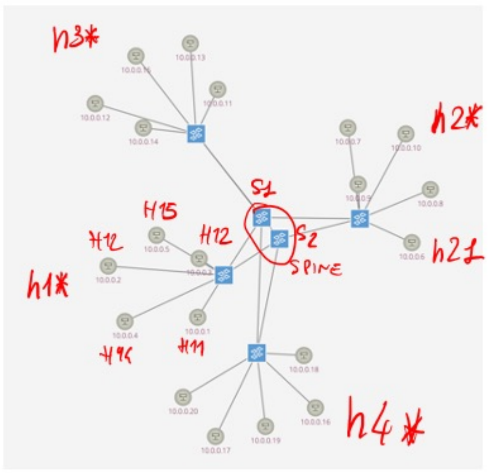


133

Intent based routing

- Create a communication intent between h11 and h24





134

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

Test 2: intent routing

- From the command line write the following command than hit ***tab*** to select the nodes

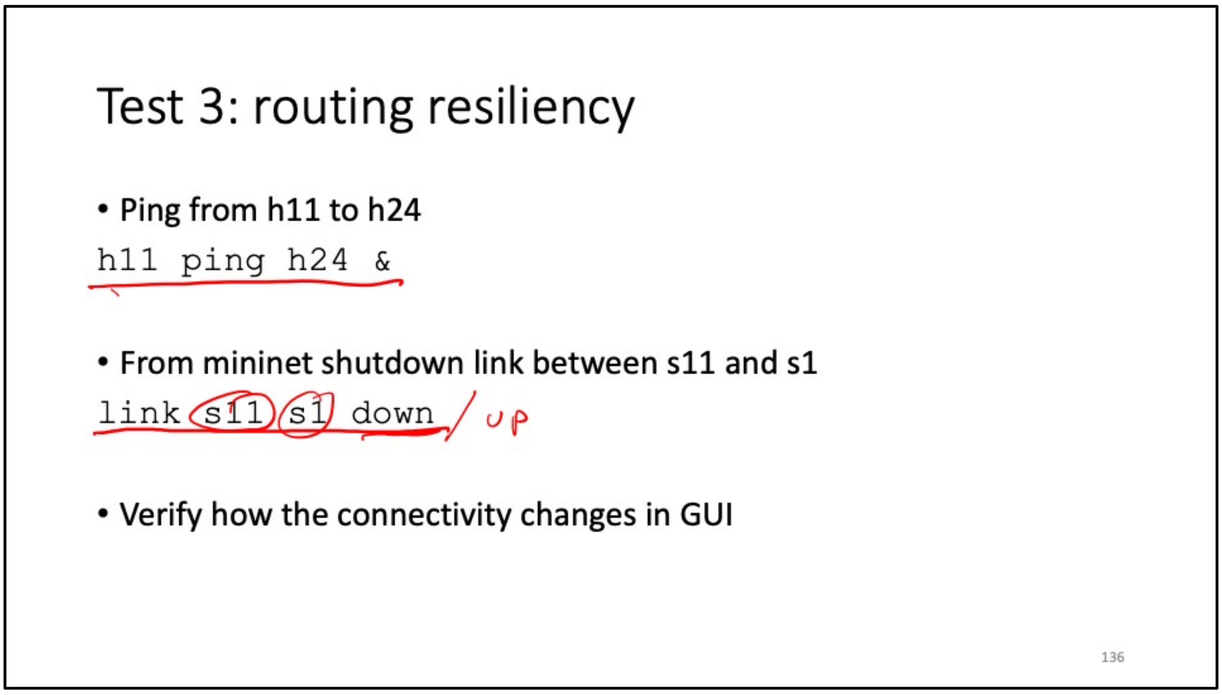
```
add-host-intent
```

- From mininet

```
h11 ping h24
```

- From the GUI, go to the *intent* menu and enable the path visualization

135



Test 3: routing resiliency

- Ping from h11 to h24

```
h11 ping h24 &
```

- From mininet shutdown link between s11 and s1

```
link s11 s1 down
```

/ up

- Verify how the connectivity changes in GUI

136

Test 4: network performance

- you can test connectivity in mininet using iperf

```
h25 iperf -s &> h25.log &

h11 iperf -t 10 -c 10.0.0.10 &> h11.log &
```

- Observe the traffic flow in ONOS

137