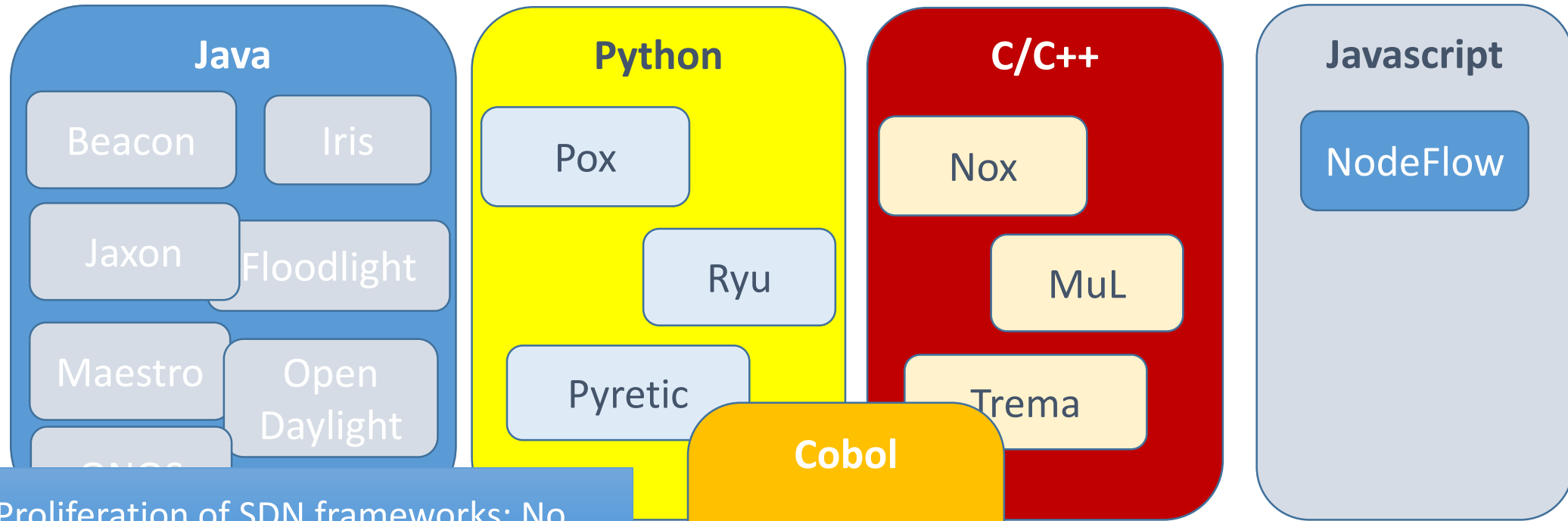




Supporting composed SDN applications and controller independence with NetIDE

Alec Leckey
Intel Labs

SDN Application Development



Proliferation of SDN frameworks: No "one solution fits all"

Cobol
?

Several IDE components are available, "automation" chain is still missing

Openflow

OVSD

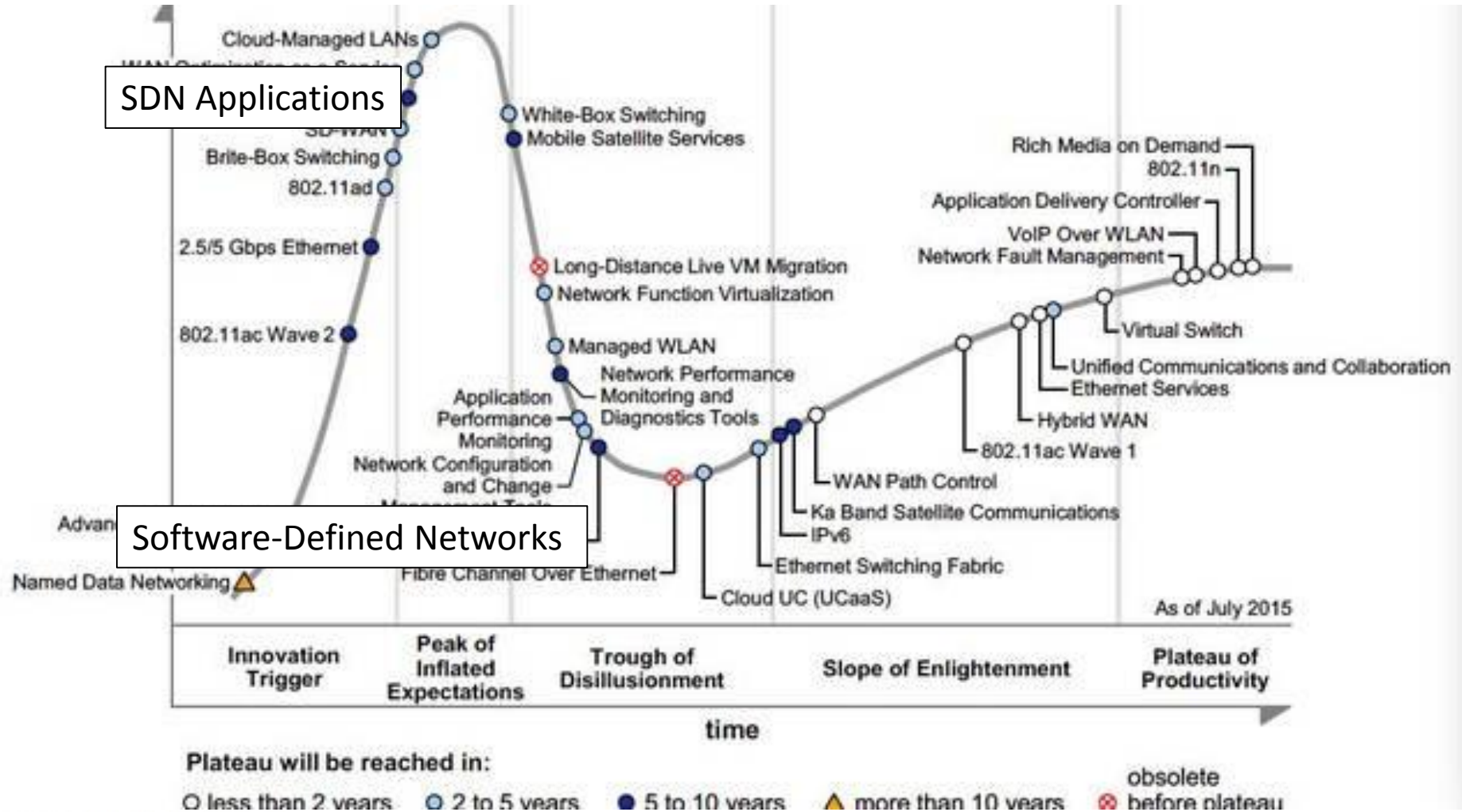
PCEP

I2RS

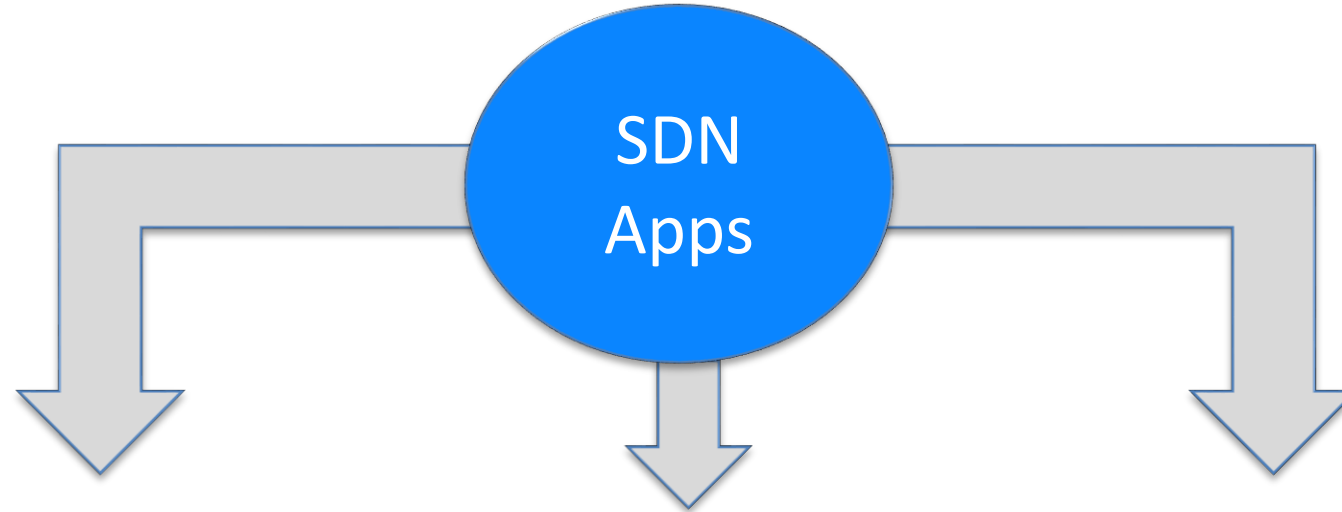


Key network services to support integration between development and testing are missing too

Opportunities for current SDN landscape



Challenges for current SDN landscape



Can't easily port them:
You implement for Controller X, you can't make same code run on Controller Y

Can't easily combine them:
Eg, You can't run an LB app together with an FW app on top of the same network

Can't easily debug them:
Only few SW development tools are available for SDN, in most cases controller-specific

The NetIDE Framework

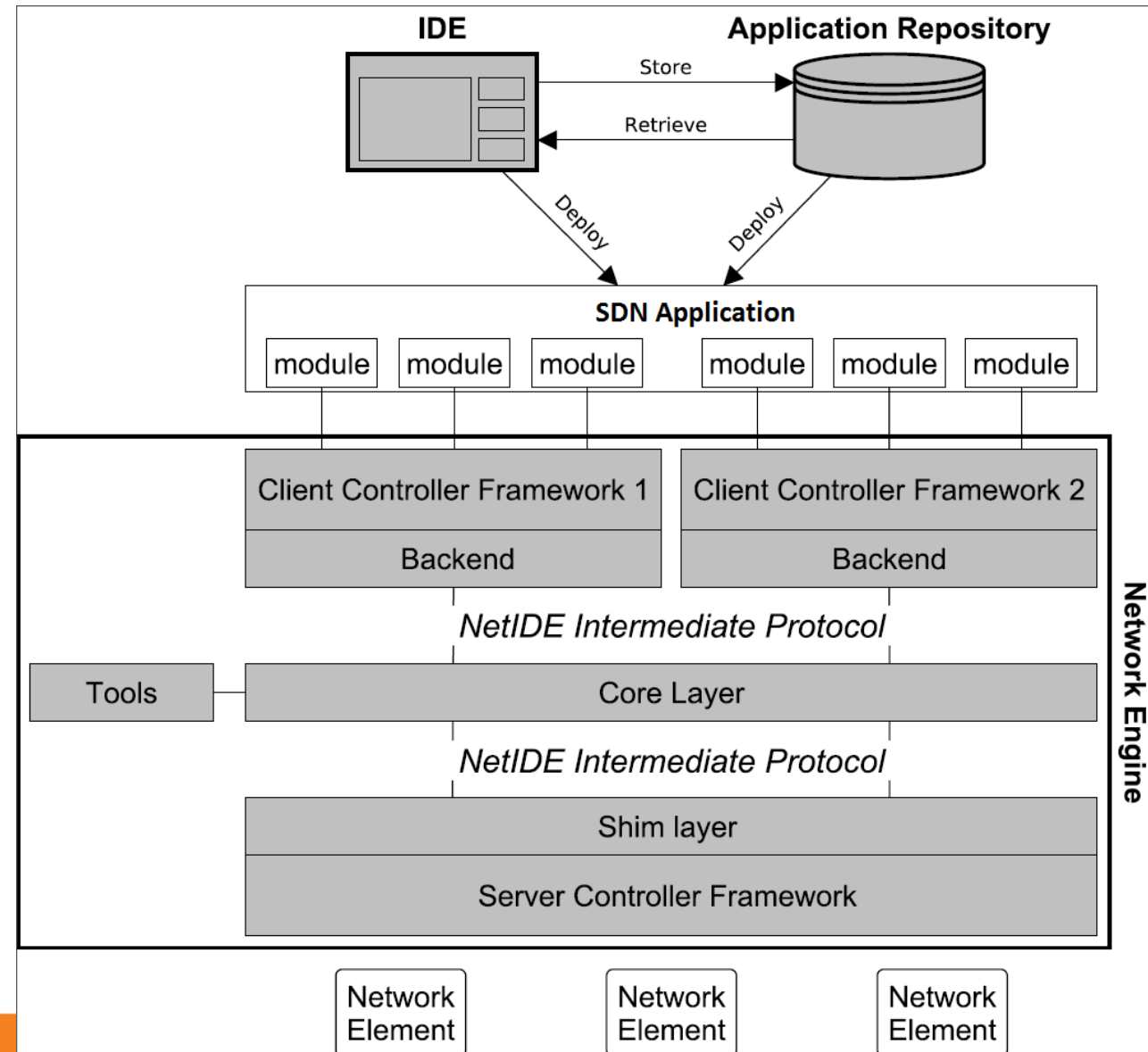
NetIDE aims at supporting the whole development lifecycle of SDN applications in a platform-independent fashion:

- Integrated SDN development environment
- Covering the full lifetime of SDN applications
- It brings all the elements of Software Design/Development to Networking:
 - Platform independence
 - Code re-usability
 - Developer tools (debugger, profiler, logger, etc.)

The NetIDE Architecture

Client/Server SDN controller paradigm of ONF:

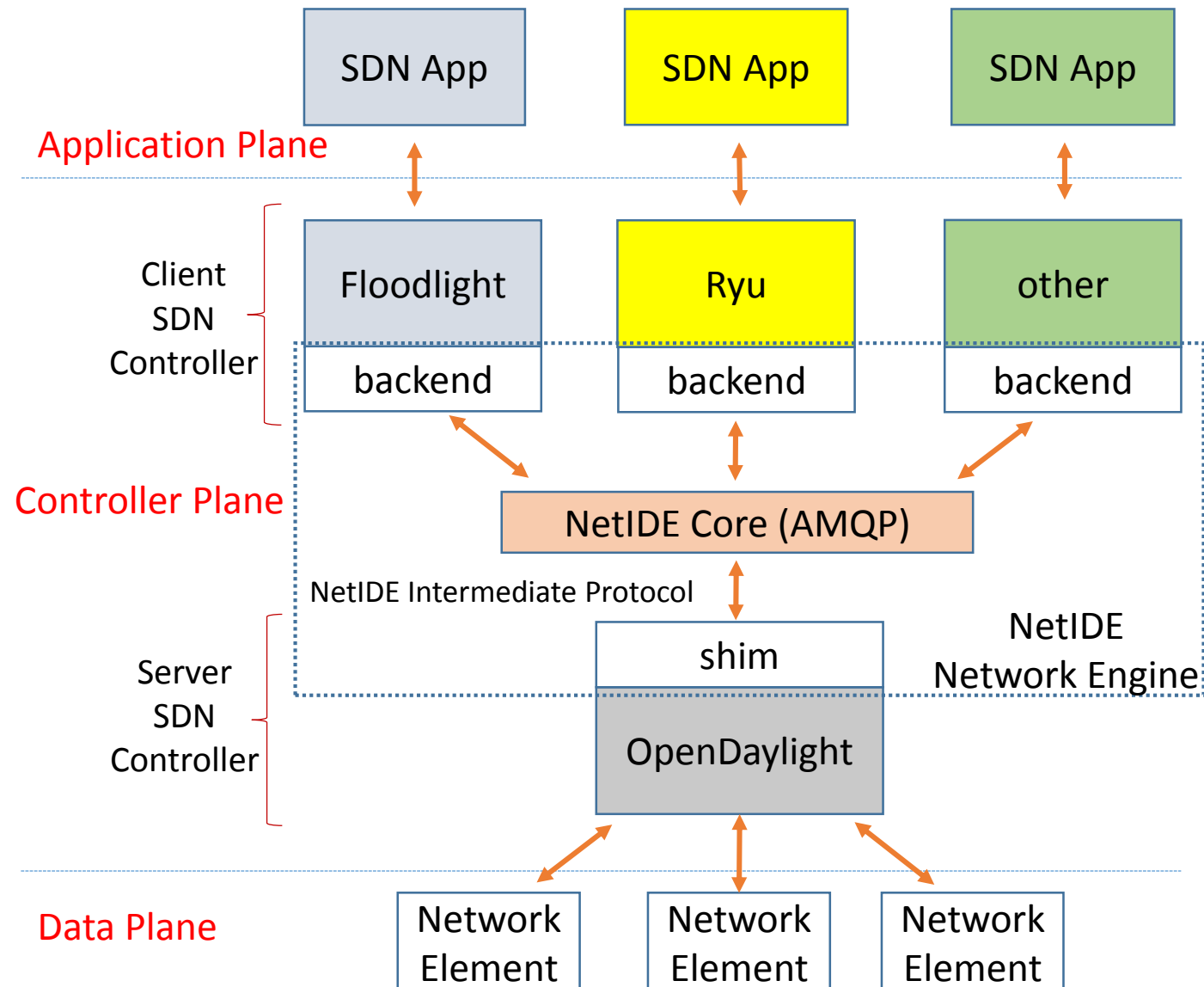
- **SDN Application's** modules are given the runtime environment they expect in the client controller
- **Multi-controller** support (ONOS, OpenDaylight, Ryu, Floodlight, etc)
- **Backend:** Captures control messages translating them to controller neutral format (ie, *Intermediate Protocol*)
- **Core Layer:** provides a controller-independent means to compose applications, resolve conflicts, provide an interface to tools
- **Shim Layer:** northbound plugin to pass control messages south and incoming events north



Network Engine Implementation with ODL

Migration of SDN Apps from legacy controllers to ODL easier

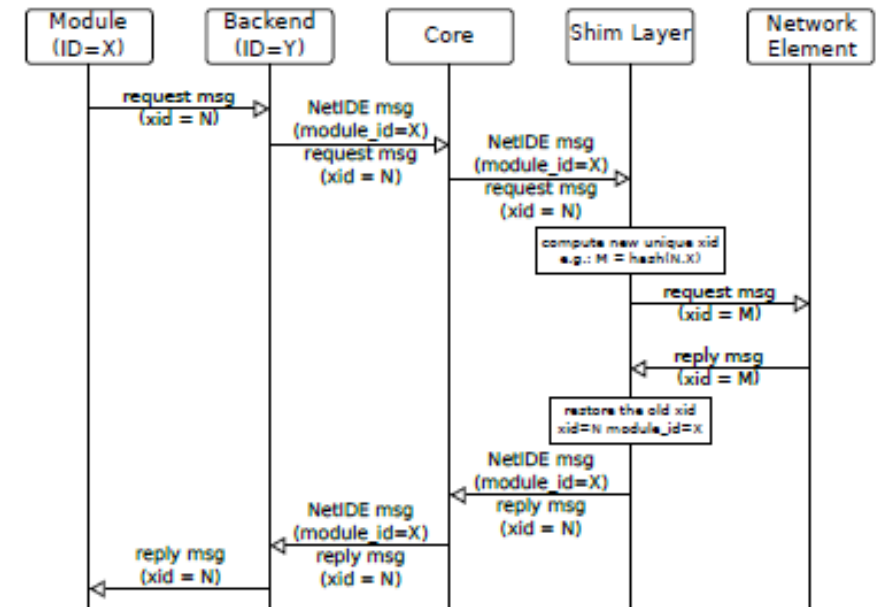
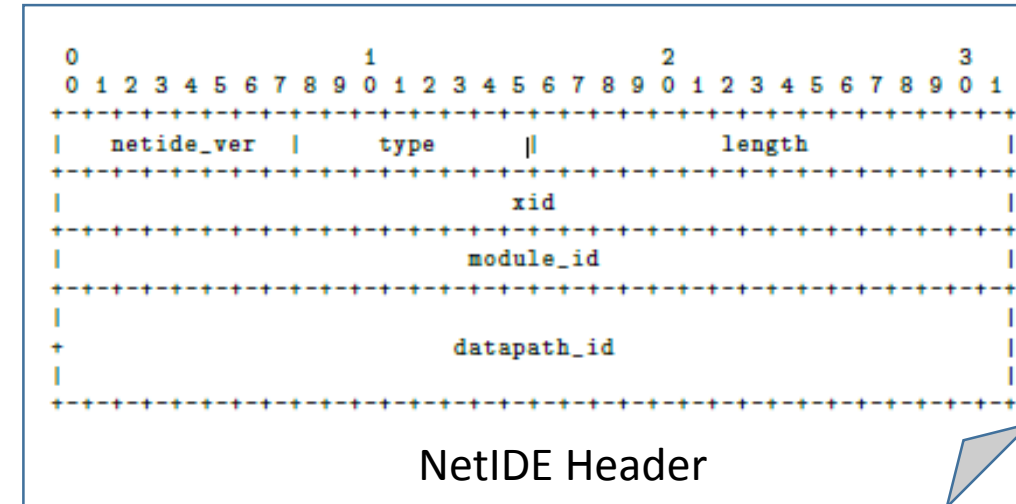
- Client/Server model of SDN controllers
- Client Controllers *communicate* with switches through an interop layer
- **SDN Application's** consume expected API
- **Backend:** provide switch implementations
- **Core:** provides a controller-independent messaging layer
- **Shim:** passes messages to relevant API



Intermediate Protocol

Implements:

- Transport Management Messages between Layers
- Transport Event/Action messages
- Encapsulate protocol Messages (OF, NetConf)
- NetIDE Header:
 - **Module Identifier:**
unique value that allows the Core to orchestrate individual modules of each client controller
 - **Type**
Hello, Error, Mgmt, Openflow, Netconf, HeartBeat, ModuleAnnouncement, ModuleAcknowledgement, Handshake

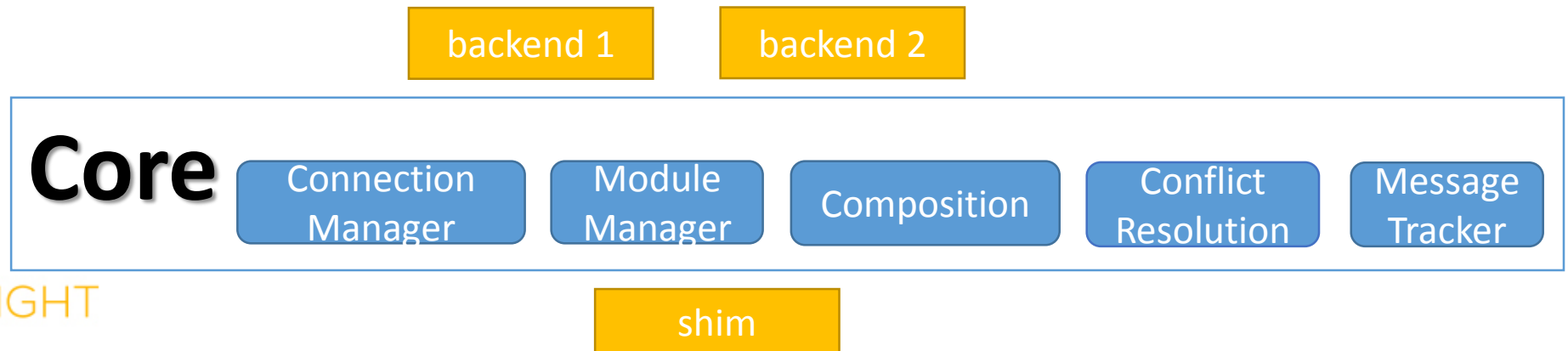


Request/Reply message handling

Core Layer

Implements 3 main functions:

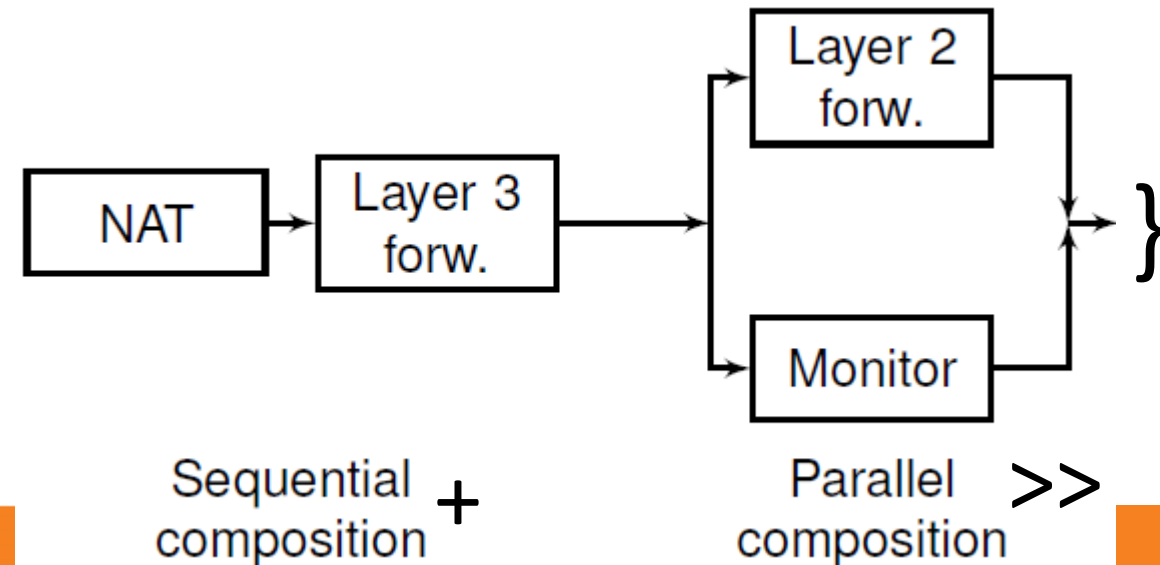
- Interface between Client Backends and Server Shim: manages lifecycle of controllers including modules
- Orchestrates execution of individual modules/complete applications spread across multiple controllers
- Manages symmetric messages (request/reply) exchanged between modules and network elements



Application Composition

- The *Network Engine* allows SDN Application's modules written for different controller platforms to cooperate on controlling the same network infrastructure
- The *Core Layer* in the Engine implements conflict resolution and composition mechanisms that are independent from the applications
- Modules are composed into a single NetIDE SDN application at deployment time

- Composition Semantics: + >> () }
 - (NAT + L3) + (L2 >> Mon)



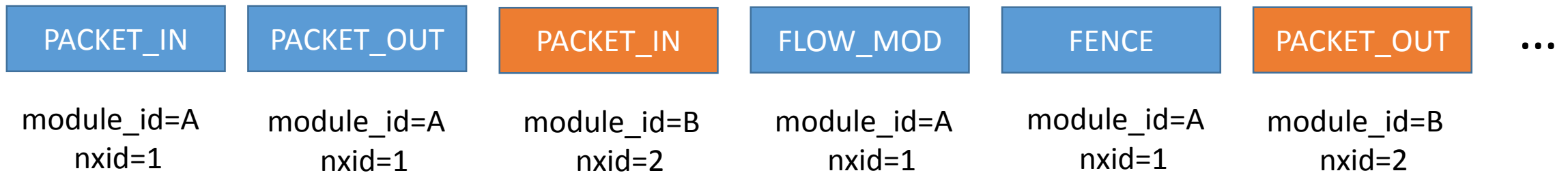
Composition Execution

- Module list
- Execution Semantics
 - Parallel composition
 - Sequential
- Composition Specification
 - Execution flow
- Conflict Resolution
 - Define conflict (match, action)
 - Scope (local, application, global)
 - Resolution Policy (ignore, discard, priority, auto)

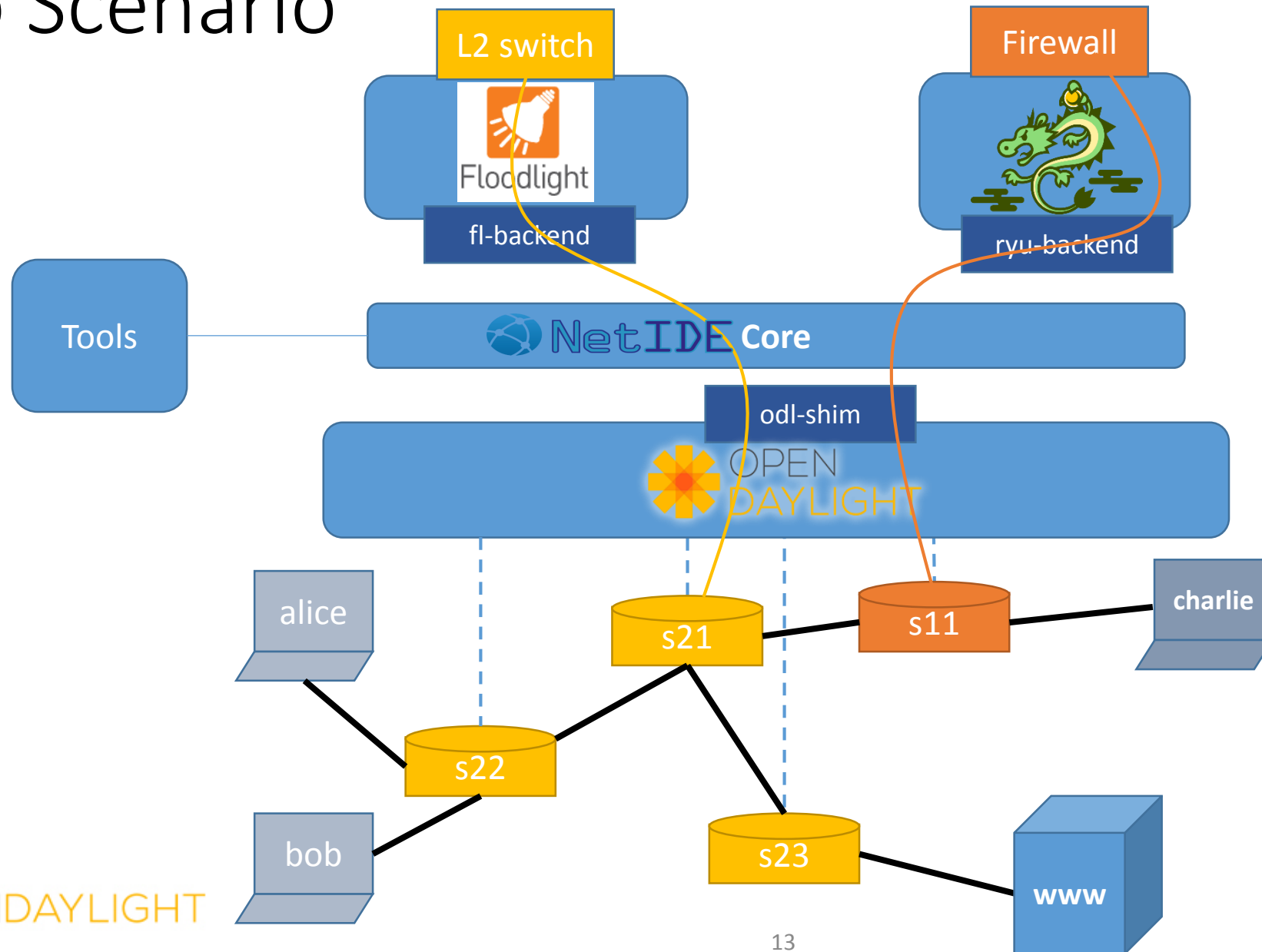
```
<ExecutionManifest>
  <Modules>
    <Module id="NAT">
      <Identier>eu.netide.nat</Identier>
    </Module>
    <Module id="L3-fwd">
      <Identier>eu.netide.l3_fwd</Identier>
      <Filter>event=packet_in,source=10.1.0.*</Filter>
    </Module>
    <Module id="L2-fwd">
      <Identier>eu.netide.l2_fwd</Identier>
    </Module>
    <Module id="Monitor">
      <Identier>eu.netide.monitor</Identier>
    </Module>
  </Modules>
  <ExecutionPolicy>
    <ModuleCall id="NAT"/>
    <ModuleCall id="L3-fwd" emptyResultAction="drop"/>
    <ParallelCall mergePolicy="priority">
      <ModuleCall id="L2-fwd" priority="2"/>
      <ModuleCall id="monitor" priority="1"/>
    </ParallelCall >
  </ExecutionPolicy>
</ExecutionManifest>
```

Message Fencing

- Application Composition is dependent on knowing when a module has completed processing an event
 - May send message prematurely
 - Wait indefinitely
- “Fence” – end of execution marker
 - SDN Controller signal processing complete, eg, *IControllerCompletionListener*{} (Floodlight)

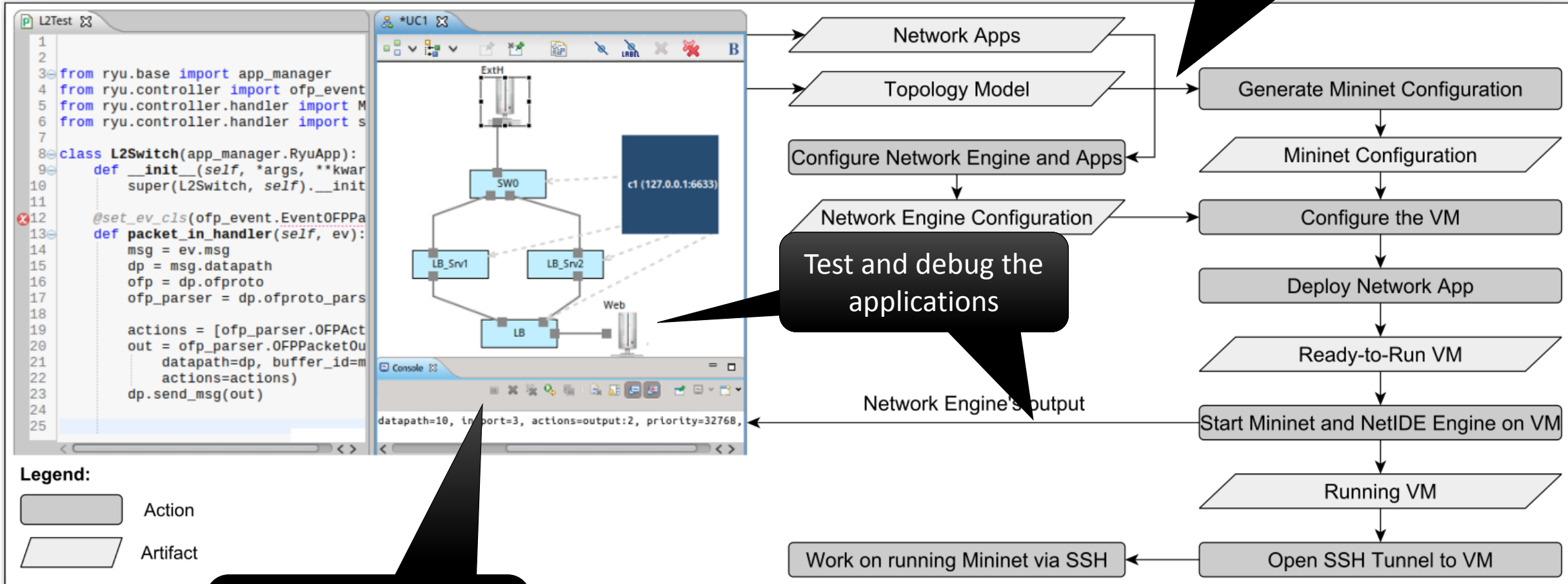


Demo Scenario



Develop - Deploy - Test

Automatically deploy the SDN applications



Develop the code and configure the topology



Integrated Environment

Code Editors

Graphical
Topology Editor

Code Editors (PyDev, Java)

Topology editor

Interface with the
Network Engine and tools

Access to the Mininet CLI

Mininet CLI

```
1  
2  
3 from ryu.base import app_manager  
4 from ryu.controller import ofp_event  
5 from ryu.controller.handler import MAIN_DISPATCHER  
6 from ryu.controller.handler import set_ev_cls  
7  
8 class L2Switch(app_manager.RyuApp):  
9     def __init__(self, *args, **kwargs):  
10         super(L2Switch, self).__init__(*args, **kwargs)  
11  
12     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)  
13     def packet_in_handler(self, ev):  
14         msg = ev.msg  
15         dp = msg.datapath  
16         ofp = dp.ofproto  
17         ofp_parser = dp.ofproto_parser  
18  
19         actions = [ofp_parser.OFPACTIONOutput(ofp.OFPP_FLOOD)]  
20         out = ofp_parser.OFPPacketOut(  
21             datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,  
22             actions=actions)  
23         dp.send_msg(out)  
24  
25
```

```
DemoTopology [NetIDE Controller Deployment] /usr/bin/vagrant  
*** Adding links:  
(N1_h1, N1_s1) (N1_h2, N1_s2) (N1_s1, N1_s2)  
*** Configuring hosts  
N1_h1 N1_h2  
*** Starting controller  
c1  
*** Starting 2 switches  
N1_s1 N1_s2 ...  
*** Starting CLI:  
pingall  
*** Ping: testing ping reachability  
N1_h1 -> N1_h2  
N1_h2 -> N1_h1  
*** Results: 0% dropped (2/2 received)  
  
DemoTopology [NetIDE Controller Deployment] /usr/bin/vagrant  
packetOut: {'outport': 1, 'srcmac': '36:e9:d4:32:79:6d', 'dstmac': '02:00:00:00:00:01'}  
Send to OF client  
New message from the client: packet  
Backend. Datapath: 1 packet_in: version: 0x1 msg_type 0xa xid 00:00:00:00:00:00  
Datapath 1 current state 1: main message type: 10  
handlers: []  
Ryu flowMod: version: 0x1 msg_type 0xe xid 0x7 OFPFlowMod(actions=[OFPACTIONOutput(max_len=65509, port=2)])  
send_install pred: {'switch': 1, 'dstmac': '36:e9:d4:32:79:6d'}  
Action List: [{'outport': 2}]  
Send to OF client  
packetOut: {'outport': 2, 'dstip': '10.0.0.2', 'protocol': 1, 'srcip': '10.0.0.1'}  
Send to OF client  
New message from the client: packet  
Backend. Datapath: 2 packet_in: version: 0x1 msg_type 0xa xid 00:00:00:00:00:00
```

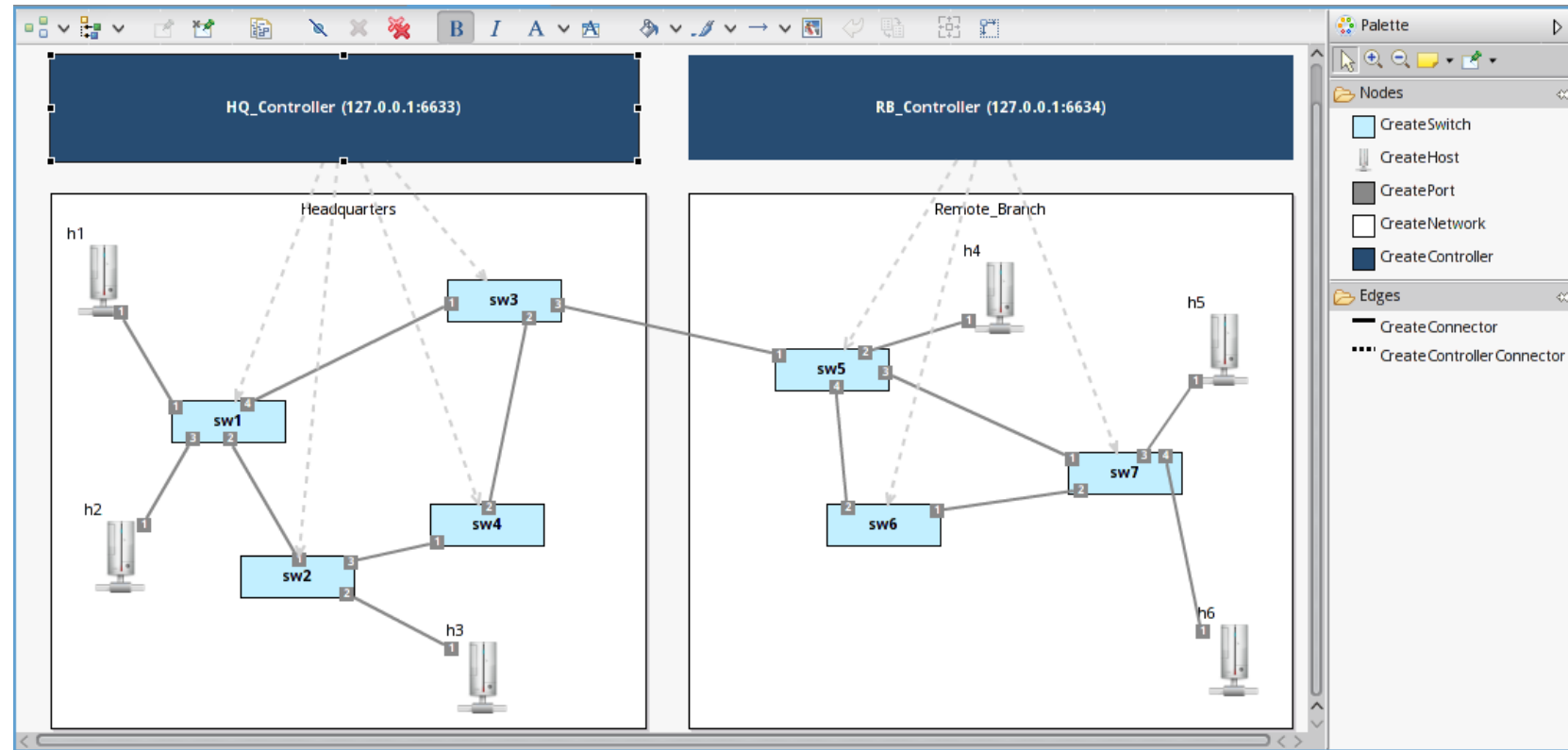
Tools for debugging and
inspecting of the control
channel



Topology Editor

Graphical editor:

- create and edit network topologies
- import underlying topology
- A topology generator outputs a configuration file for Mininet for testing



Runtime Management GUI

Management Tasks:

- Start/stop the VM
- Configure/start/stop the Network Engine
- Start/stop SDN applications

The screenshot displays the OpenDaylight Runtime Management GUI. At the top, there are tabs for 'SSH' and 'Vagrant'. Below the 'Vagrant' tab, the status is 'offline', and there are buttons for 'Vagrant Up' and 'Vagrant Halt'. The middle section shows the 'OpenDaylight' status as 'Offline', with a dropdown menu set to 'OpenDaylight' and buttons for 'Start Server Controller' and 'Stop Server Controller'. To the right, there are buttons for 'Mininet On' and 'Mininet Off'. Below this is a table of SDN applications:

App Name	Aktiv	Platform	Client	Port
simple_switch.py	offline	Network Engine	Ryu	

At the bottom of the table, there are buttons for 'Start', 'Stop', 'Reload', 'Reattach', and 'Provision'. At the very bottom of the GUI, there are buttons for 'Add Test', 'Remove Test', and 'Edit Test'.

Developer Tools

Enabling the developer to systematically test, profile, and tune their SDN App

Logger: tracing capabilities to judge the performance of the deployed SDN Application

Garbage Collector: Cleans the switches' memory from unused flow rules

Model Checker: systematically exercises app behaviour and flag actions that lead to violations of the desired safety properties

Profiler: judging the impact of network failures on the Network App behaviour

Debugger: supports debug of packet processing (OFReplay, packet inspection and flow table checking)

Project Presence



OpenDaylight.org
Get Software
Documentation
User Stories
Community
Blog
Q&A Form
Wiki

Tools
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Page information

Page Discussion Read Edit View history Search

NetIDE:Main

The NetIDE Network Engine enables portability and cooperation inside a single network by using a client/server multi-controller architecture. Separate "Client SDN Controllers" host the various SDN Applications with their access to the actual physical network abstracted and coordinated through a single "Server SDN Controller", in this instance OpenDaylight. This allows applications written for Ryu/Floodlight/Pyretic to execute on OpenDaylight managed infrastructure.

The project also includes an IDE to allow application developers develop and test their applications, including a graphical editor to specify network topologies, a UI for deployment configurations, editors to specify network environments for simulation, as well as a supporting toolsuite (debuggers, profilers, model checkers)

The "Network Engine" is modular by design:

- An OpenDaylight plugin ("shim" in architecture diagram) sends/receives messages to/from subscribed SDN Client Controllers. This consumes the ODL Openflow Plugin
- An initial suite of SDN Client Controller "Backends": Floodlight, Ryu, Pyretic. Further controllers may be added over time as the engine is extensible.

NetIDE Facts

Project Creation Date: June 28th, 2015
Lifecycle State: Incubation
Primary Contact: Alec Leckey <alexander.j.leckey at intel .com>
Project Lead: Alec Leckey
Committers:

- Alec Leckey [alexander.j.leckey at intel com (aleckey)]
- Elisa Rojas [elisa.rojas at telcaria com (erojas)]
- Roberto Doriguzzi [roberto.doriguzzi at create-net org (doriguzzi)]
- Christian Stritzke [christian.stritzke at ip.fraunhofer de (cstritzke)]
- Pedro Aranda Gutierrez [pedroa.aranda at telefonica com (paaguti)]

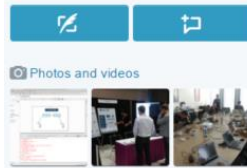
IRC: freenode.net #opendaylight
Mailing List: netide-dev@lists.opendaylight.org

NetIDE



Netide
@ProjectNetIDE · FOLLOWS YOU
NetIDE is a FP7 research project that aims at developing a one-stop solution for the development of SDN applications

Trento
netide.eu
Joined October 2013



Who to follow · Refresh · View all

- James @JMMeakor · Followed by David O'Coimin · Follow
- Michael McIntyre @McL... · Follow
- Soccer Republic @Soc... · Followed by KDUL and others · Follow

Find friends

Home · **NetIDE** · Lists · Metrics · Errors · External Install Button

ment of SDN-Apps. Featuring network topology modeling and testing apps environments.

TWEETS 97 · FOLLOWING 101 · FOLLOWERS 122 · LIKES 21

Tweets · Tweets & replies · Photos & videos

Netide @ProjectNetIDE · Feb 9
Excited to announce "Develop, Deploy and Deliver with NetIDE - Pedro A. Aranda Gutiérrez" at #ONS2016. Come with me? sched.co/67o0

Netide Retweeted
Future Networks @future_networks · Feb 2
at the @ProjectNetIDE meeting at @IMDEA_Networks premises in Madrid...



Netide @ProjectNetIDE · 2 Dec 2015
In the brave new world of software containers, you might not notice SDN, but that doesn't mean it isn't there. #SDN
searchsdn.techtarget.com/tip/Do-softwar...

work

simulation

AILS

ars (4.5), Luna (4.4)
Windows, Mac, Linux/GTK
:: **NetIDE Project**
2015-07-08 11:22
is: Alpha

2015-12-22 19:58
tian Stritzke

Find the documentation at <https://github.com/fp7-netide/IDE>.
tion of NetIDE at <https://youtu.be/sjD5rwdojIA>
: NetIDE can be found at <http://www.netide.eu>.

grant (>v1.6), and an SSH client.

work

simulation

AILS

ars (4.5), Luna (4.4)
Windows, Mac, Linux/GTK
:: **NetIDE Project**
2015-07-08 11:22
is: Alpha

2015-12-22 19:58
tian Stritzke



Share · Email · G+ · Like · 0



FP7 NetIDE

Delivering a single environment to support the whole development lifecycle of SDN programs in a vendor/controller-independent fashion

<http://www.netide.eu>

Repositories · People 18 · Teams 5 · Settings

Filters · Find a repository...

+ New repository

IDE

Provides editor support for various network programming and other specifications languages used in the development lifecycle.

Updated 5 days ago

Engine

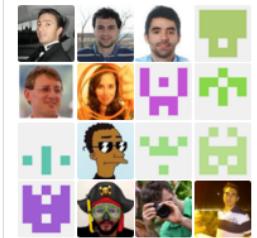
App Engine to enable Network App programs to be executed, systematically tested, and refined on a variety of concrete SDN platforms

Updated 5 days ago

Tools

Python ★ 1 0

People



Invite someone



Search

Upload



NetIDE Project

Official YouTube channel of the European project NetIDE

Subscribe 9

Uploads



Engine Demo v0.3 (2016)
31 views · 2 months ago



Memory Management System (MMS) Demo
20 views · 2 months ago



NetIDE Developer Toolkit
172 views · 8 months ago



EngineDemo v0.2 (2015)
70 views · 8 months ago



Questions

