NetIDE

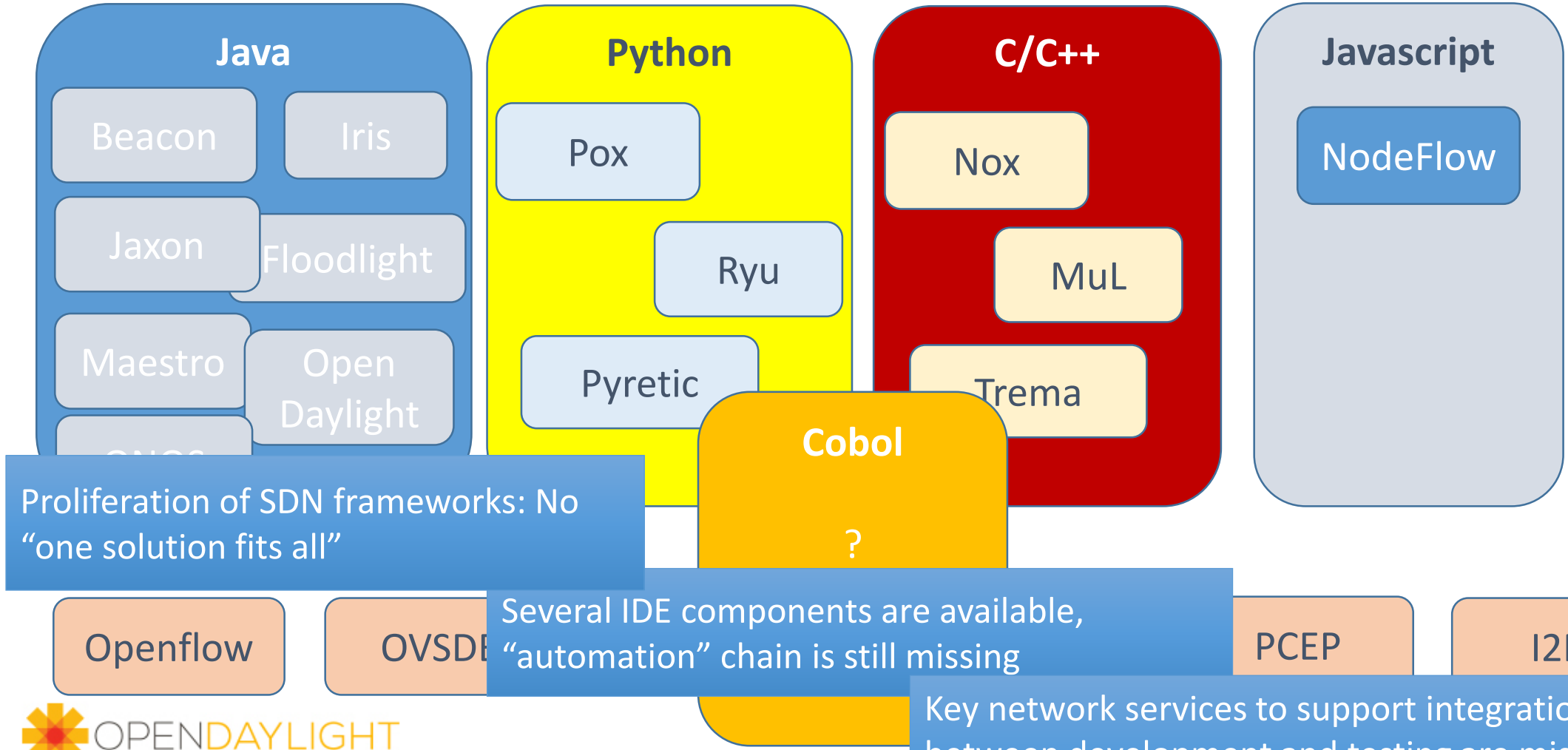
Supporting composed SDN applications and controller independence with NetIDE

Alec Leckey

Intel Labs

LINUX FOUNDATION
COLLABORATIVE PROJECTS

SDN Application Development

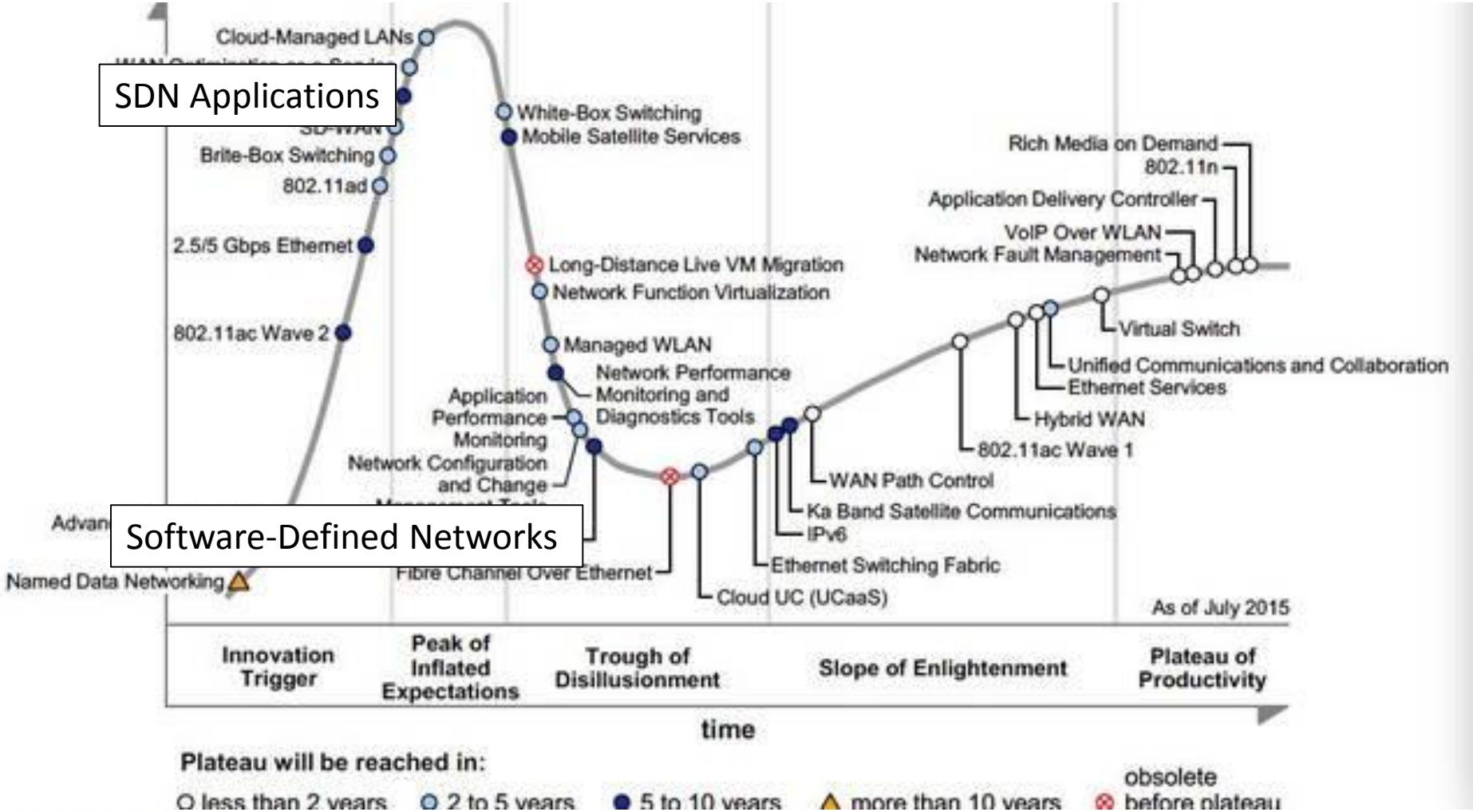


Proliferation of SDN frameworks: No "one solution fits all"

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

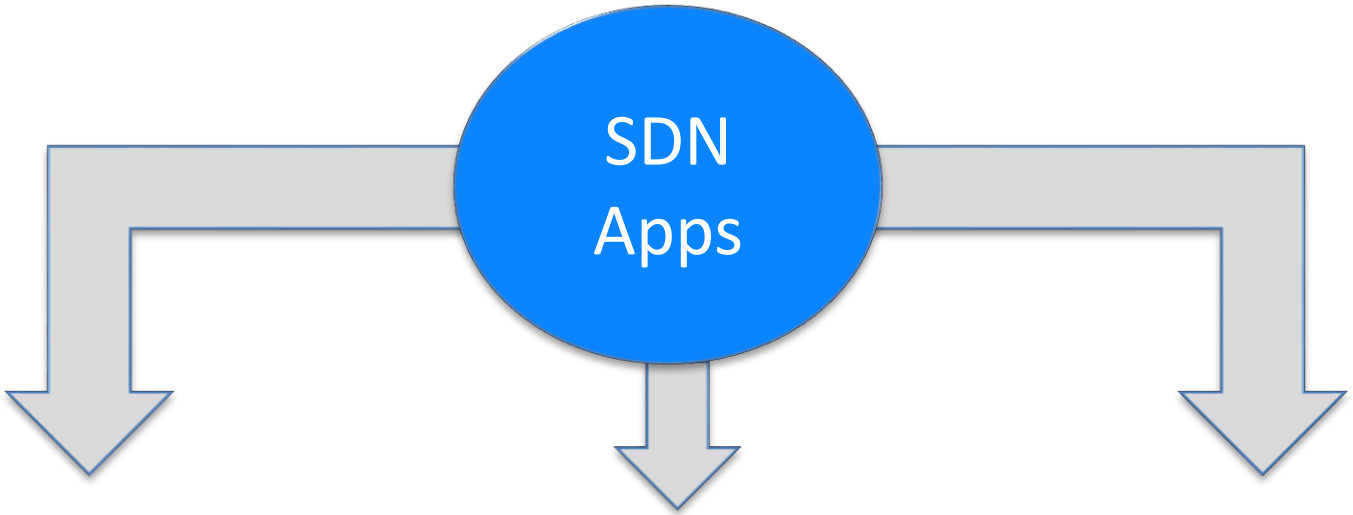
Key network services to support integration between development and testing are missing too

Opportunities for current SDN landscape



OPENDAYLIGHT

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

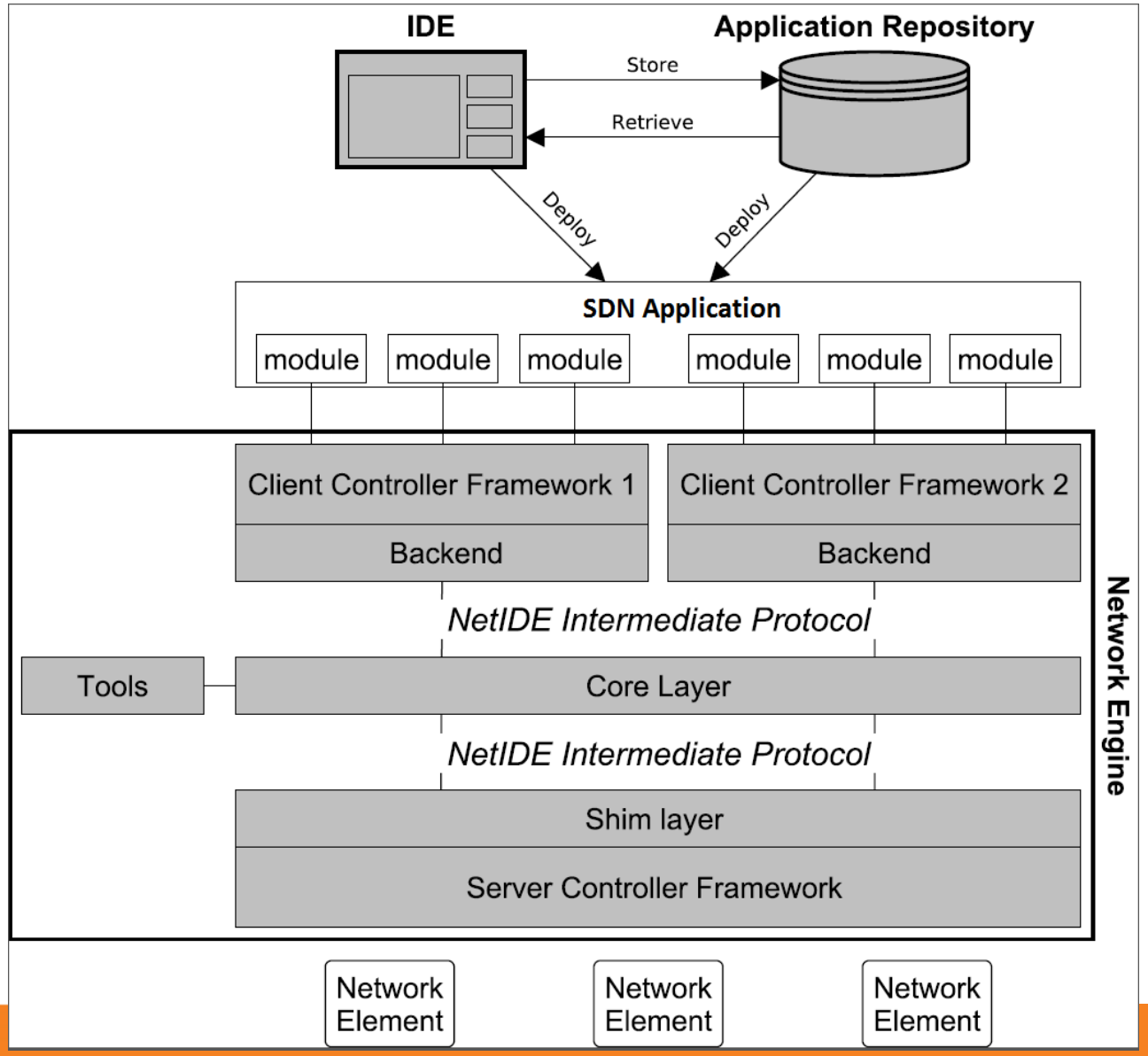
NetIDE aims at supporting the whole development lifecycle of SDN applications in a platform-independent fashion:

- 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



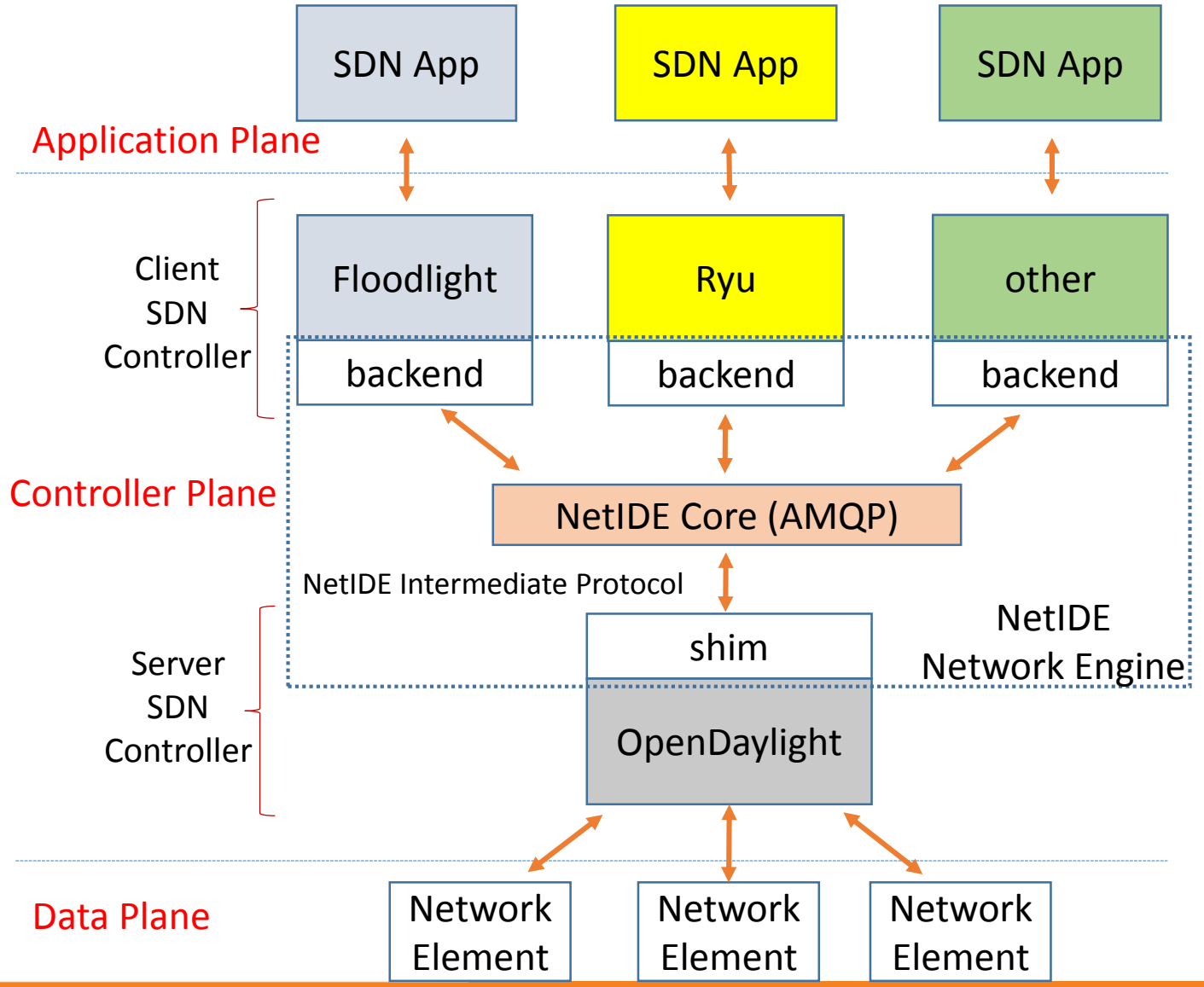
OPENDAYLIGHT

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

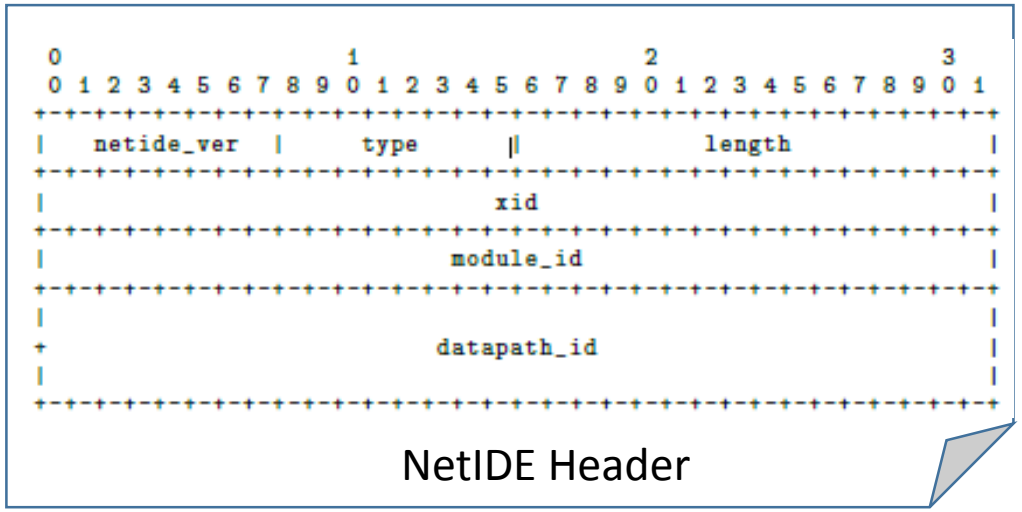


OPENDAYLIGHT

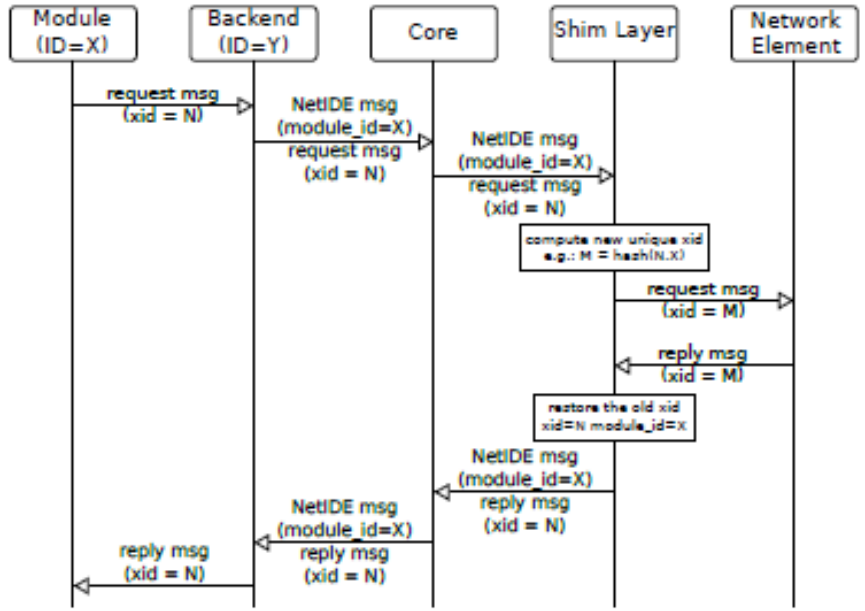
Intermediate Protocol

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



NetIDE Header



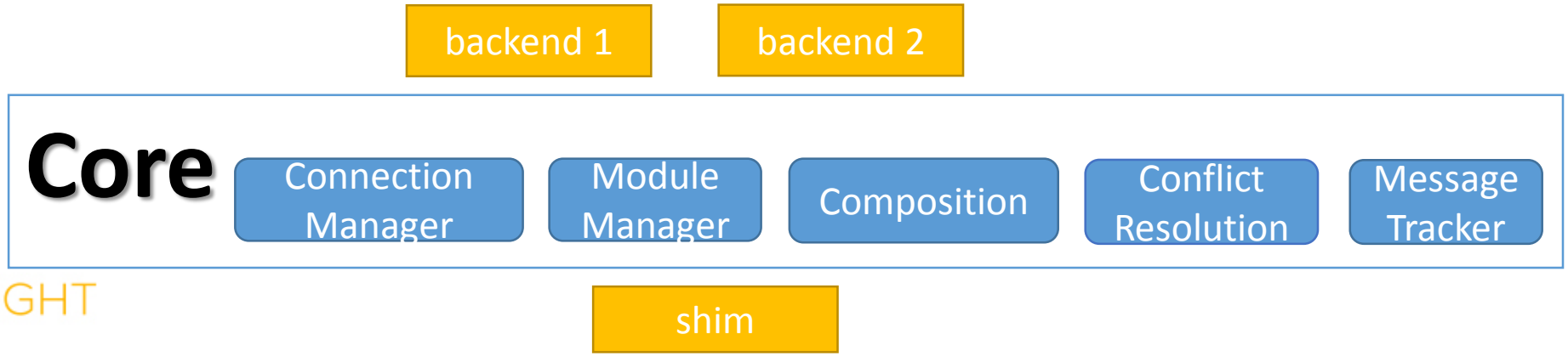
Request/Reply message handling

OPENDAYLIGHT

Core Layer

Implements 3 main functions:

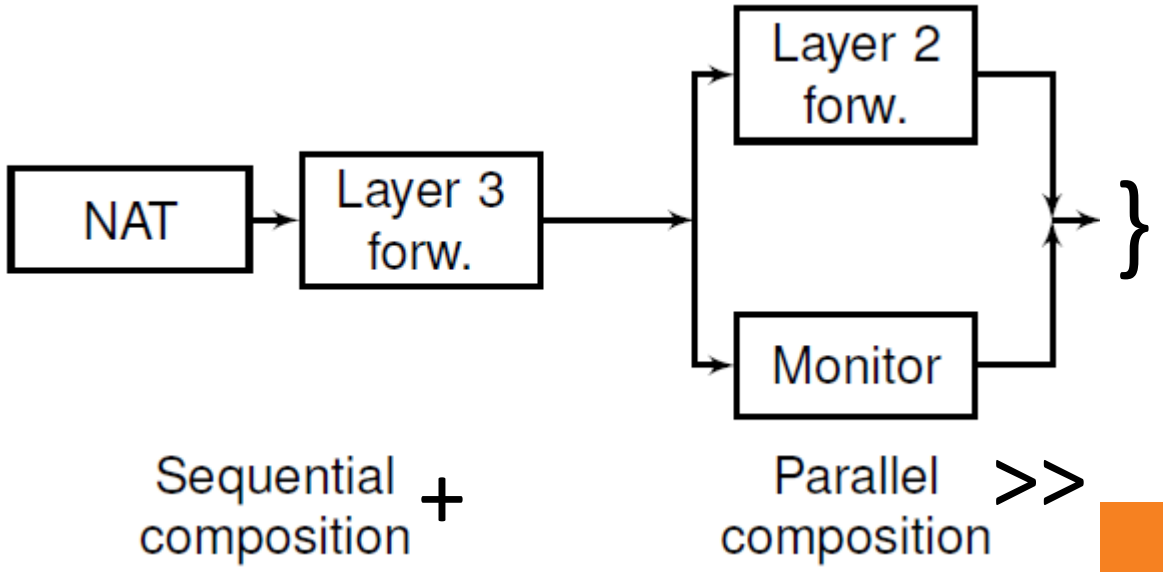
- Interface between Client Backends and Server Shim: manages lifecycle of controllers including modules
- Orchestrates execution of individual modules/complete applications spread across multiple controllers
- Manages symmetric messages (request/reply) exchanged between modules and network elements



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)



OPENDAYLIGHT

Composition Execution

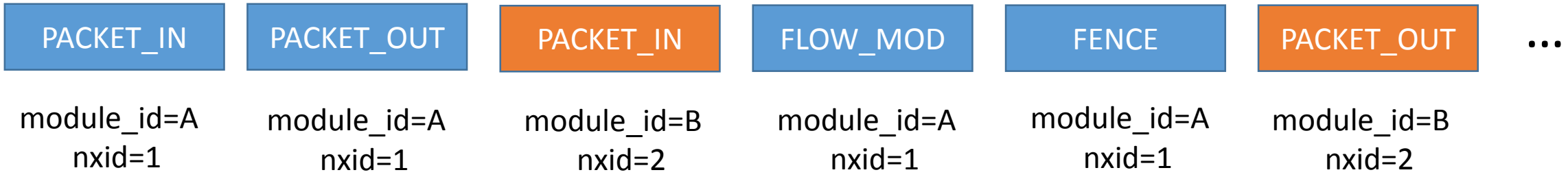
- Module list
- Execution Semantics
  - Parallel composition
  - Sequential
- Composition Specification
  - Execution flow
- Conflict Resolution
  - Define conflict (match, action)
  - Scope (local, application, global)
  - Resolution Policy (ignore, discard, priority, auto)

```
<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>
```

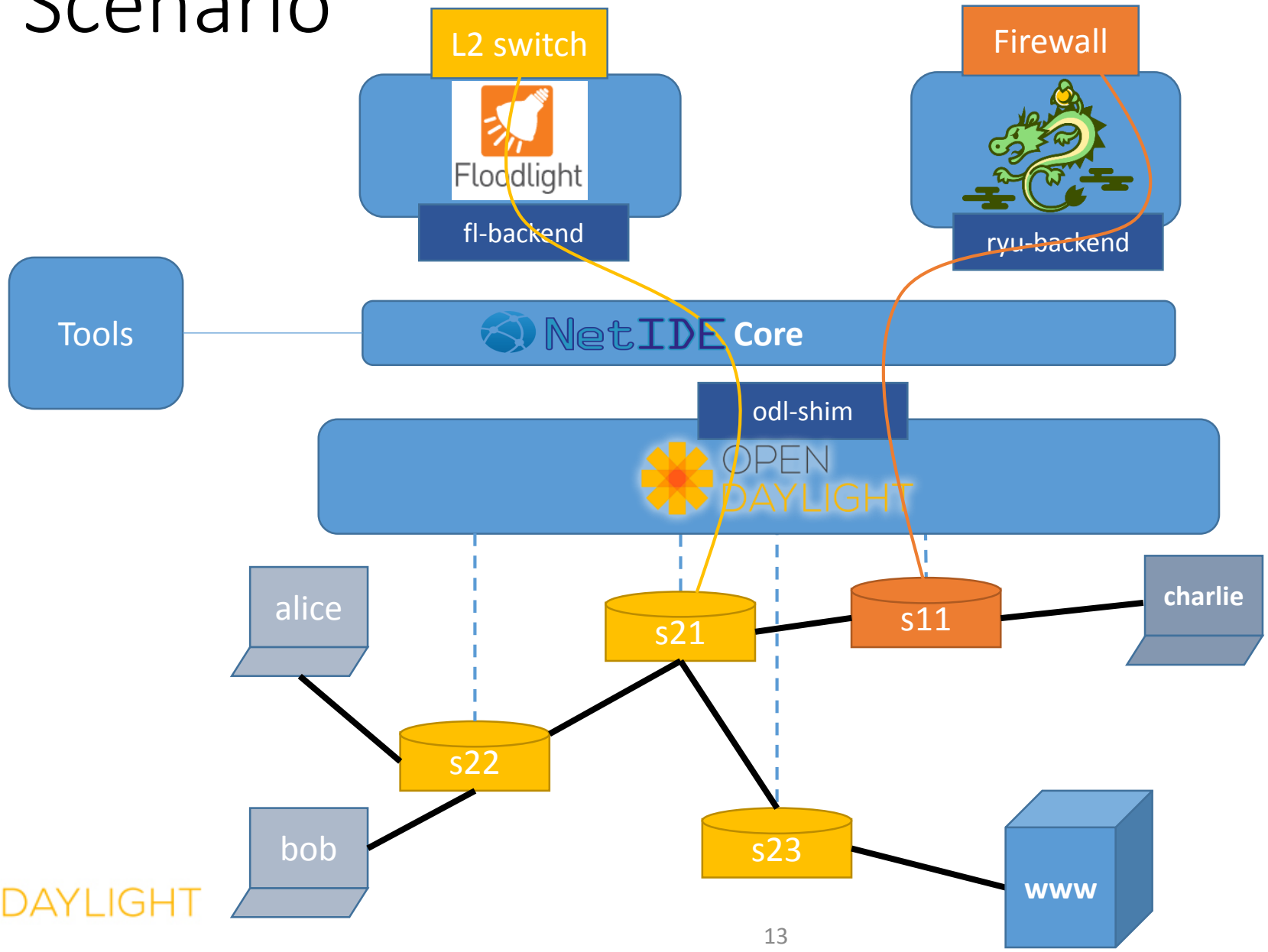
OPENDAYLIGHT

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)

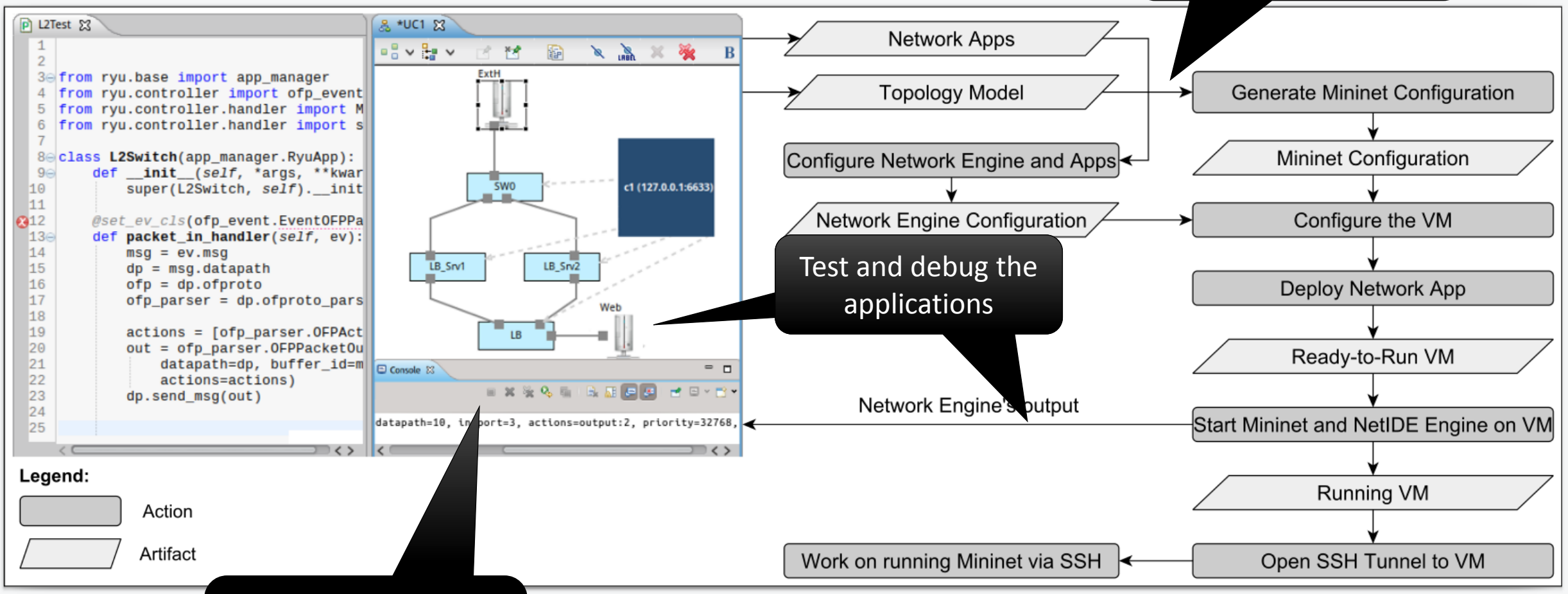


Demo Scenario



OPENDAYLIGHT

Develop - Deploy - Test



Automatically deploy the SDN applications

Test and debug the applications

Develop the code and configure the topology

Integrated Environment

Code Editors

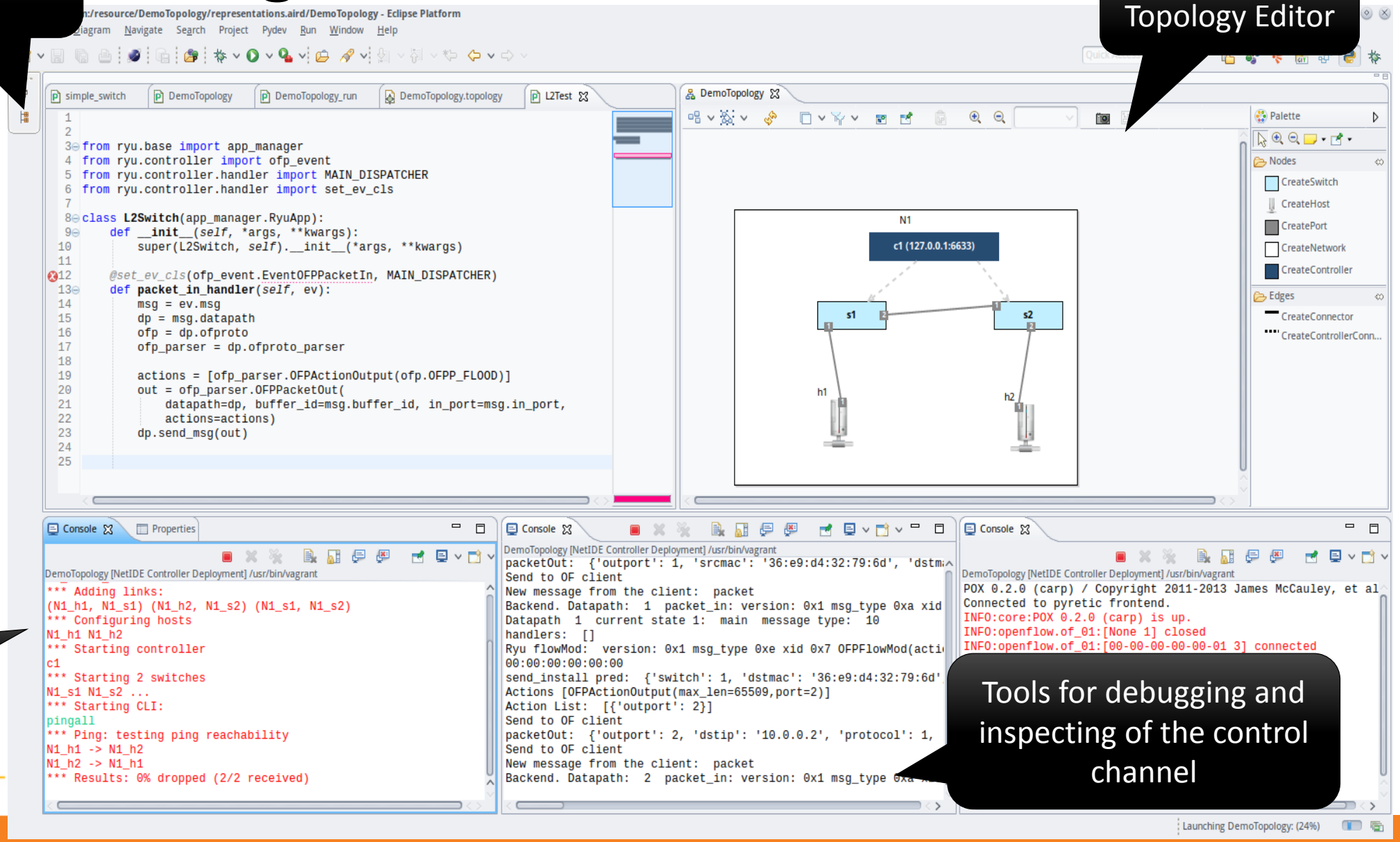
Graphical Topology Editor

Code Editors (PyDev, Java)

Topology editor

Interface with the Network Engine and tools

Access to the Mininet CLI



Mininet CLI

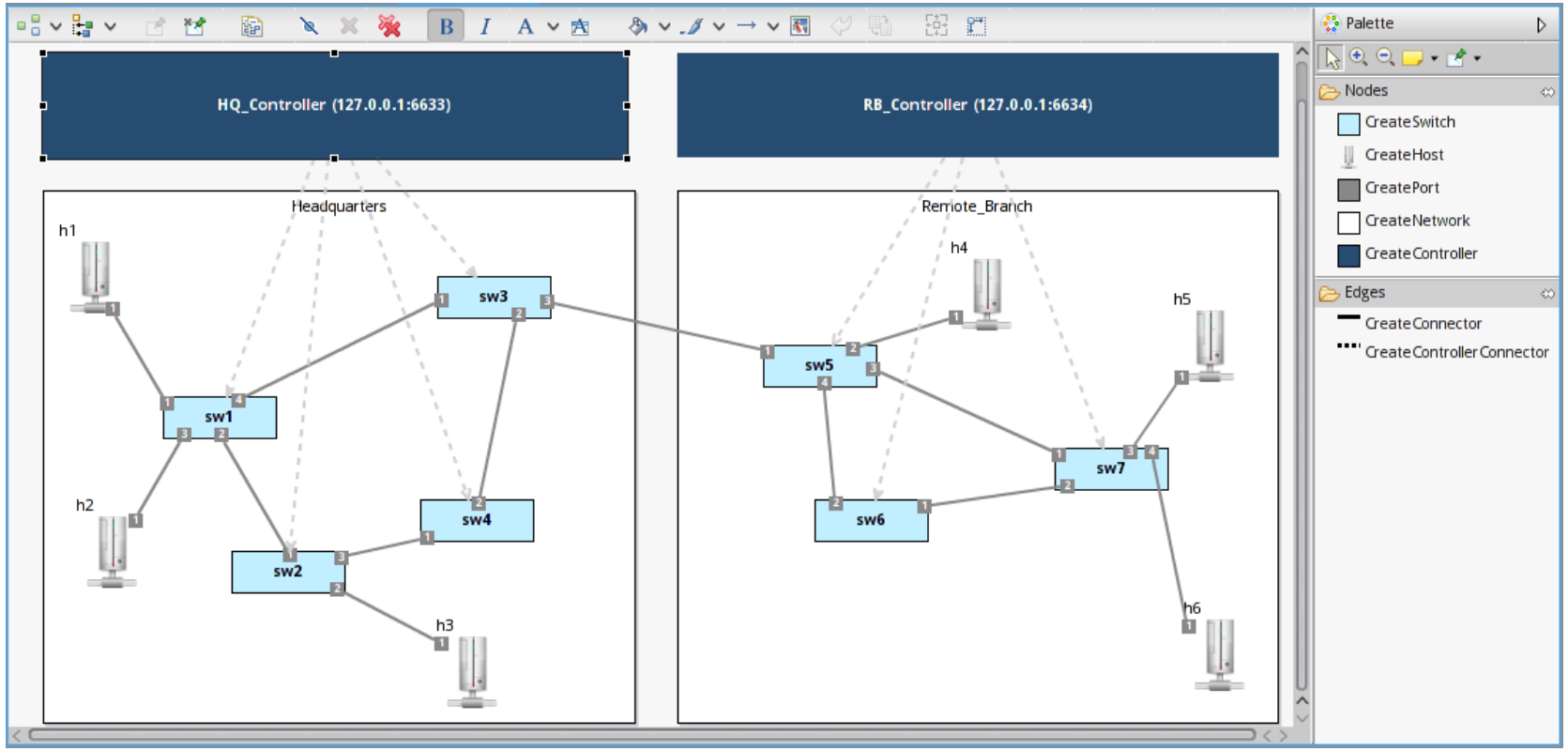
Tools for debugging and inspecting of the control channel

OPENDAYL

Topology Editor

Graphical editor:

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

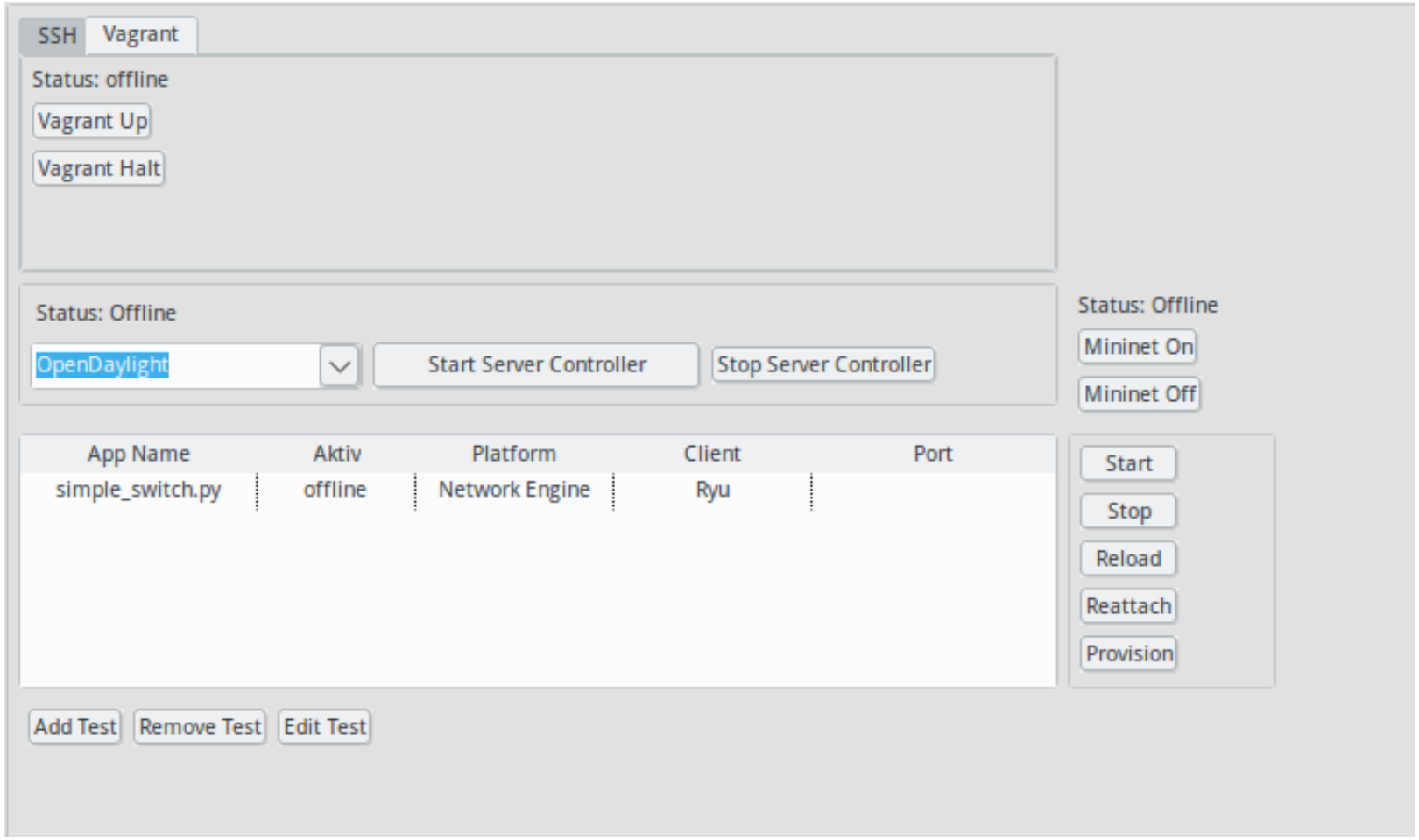


OPENDAYLIGHT

Runtime Management GUI

Management Tasks:

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



OPENDAYLIGHT

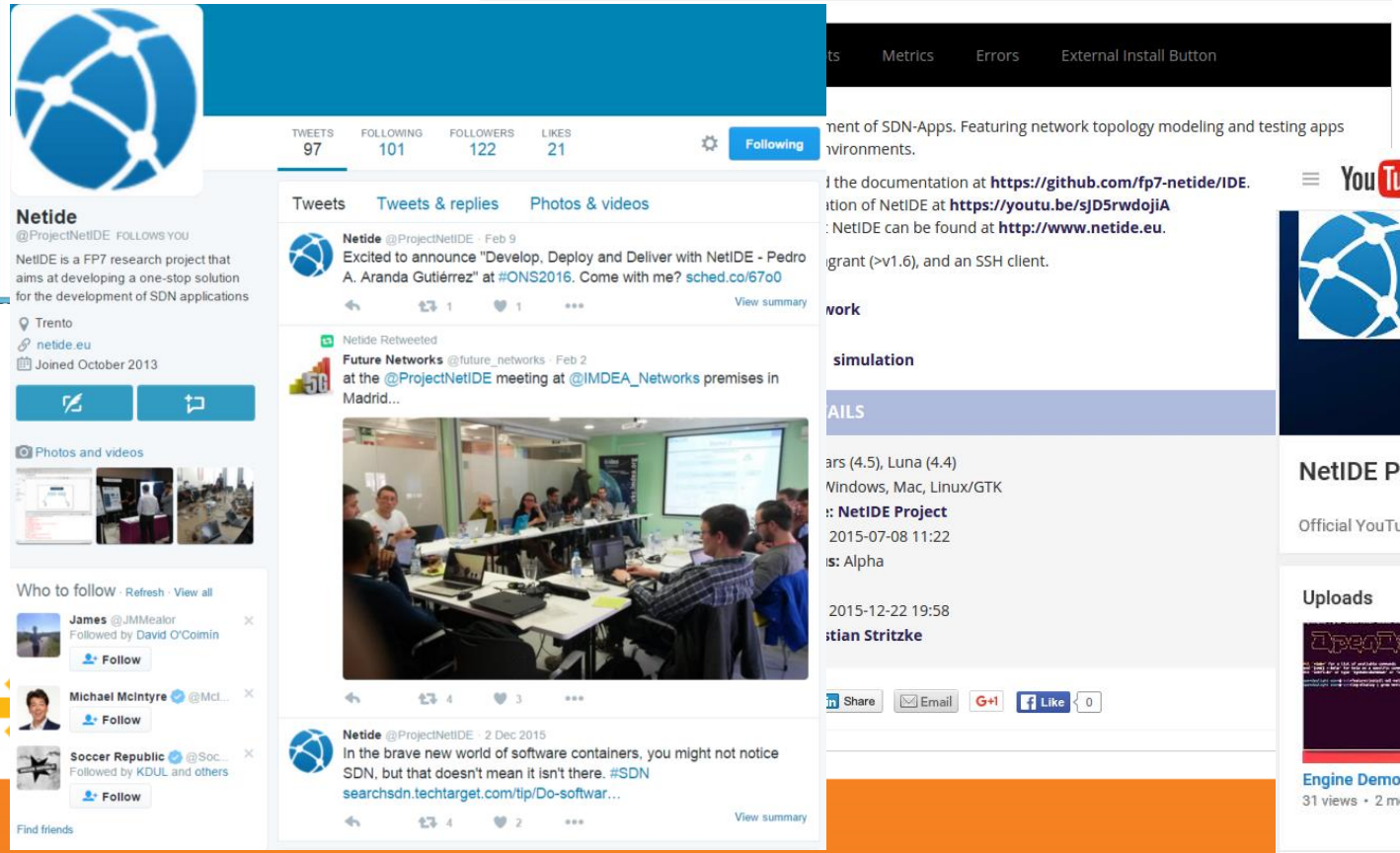
Developer Tools

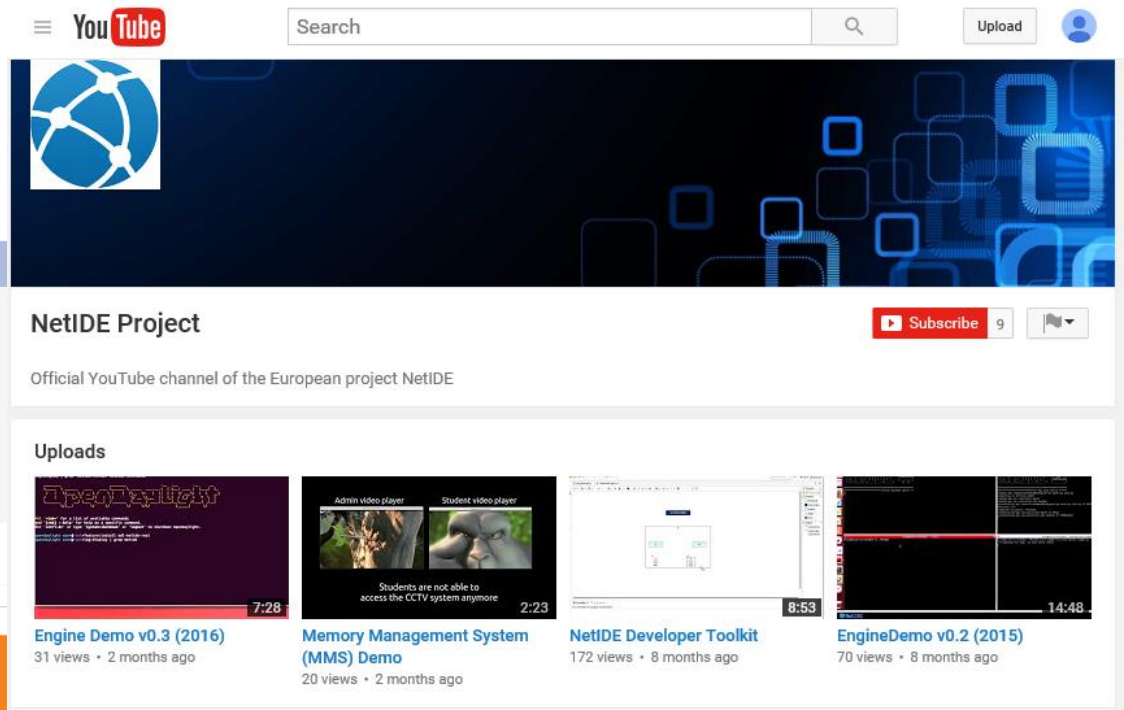
Enabling the developer to systematically test, profile, and tune their SDN App

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

NetIDE





Questions