24Nov’16 Thesis meeting

**Work given in last meeting(17 November 2016)**

* Mininet 1.3 with NetIDE -
* Try mininet in combination with platforms specified in NetIDE(Ryu done, Floodlight-port#7753, ONOS ok clean port#6653, apps -a -s and then mininet)
* Get details about NetIDE, components and if it is suitable for **testbed**
* OpenDayLight Setup - Done

NetIDE Details

1. Cross platform
2. Composing diff SDN app or using these as modules to create a new complex SDN app
3. N/w apps are modules
4. controllers act as client for server controller which drives the underlying equipments
5. CoVisor(Compositional Hypervisor) does the similar work
6. allow a *Network Application* to consist of modules written for different controller frameworks.
7. SDN applications are called *Modules*
8. Network Engine provides to unmodified SDN applications the runtime they expect
9. South-bound Interface (SBI) for the client controllers to the North-bound Interface (NBI) of the server controllers. But as these interfaces do not normally match, adaptation is necessary.
10. we use separate adaptors for the SBI, called *Backend*, and for the NBI, called *Shim*. This separation imposes a protocol between them, the *NetIDE Intermediate Protocol*
11. ***Core***: it hosts all logic and data structures that are independent of the particular controller frameworks and communicates with Shim and Backends using the same NetIDE Intermediate Protocol. The Core makes both Shim and Backend light-weight and easier to implement for new controllers.
12. The **Shim is a platform-specific component** that **translates the NBI of the server controller to the NetIDE API,** thereby exposing it to the other components of the Network Engine.
13. **Backend** - At boot-time, the Backend starts the discovery of the application modules running on top of the client controller (*Module Discovery*) and registers them to the Core (*Announcement Handler*), which, in turn, assigns a spe- cific identifier (module\_id) for each registered module. As part of its initialization process, the Backend also queries the Shim for the physical topology. Backends store the topology information as instances of the network elements
14. **NetIDE Intermediate protocol** implements the follow- ing functions: (i) **to carry management messages between the Network Engine’s layers (Core, Shim and Backend)**; e.g., to exchange information on the supported SBI protocols, to provide unique identifiers for application modules, implement the *fence* mechanism, (ii) **to carry event and action messages between Shim, Core, and Backend**, properly demultiplexing such messages to the right module based on identifiers
15. **ODL and ONOS are carrier-grade controllers** targeted to service providers, enterprises and mainstream deployments. Being **able to scale to a large number of network elements,** we **recognized both ODL and ONOS as the most suitable SDN platforms** for playing the server controller’s role.
16. **ODL and ONOS projects** are **based** on **Apache Karaf** [, so that we **implemented** the **Shims** for such platforms **in form of OSGi bundles**
17. Support OpenFlow versions 1.0 and 1.3
18. The **Shim for Ryu** has been **implemented** as an application based on the **Python northbound API**. It includes **serializer and deserializer methods** **based** on the Python’s ***struct* module**
19. RTT



1. Packet Arrives at switch(No rules installed)
2. It will contact the controller (Server controller) that uses FL shim
3. Flooding application i.e. written in Ryu
4. Server controller has the