System Design for Forwarding Orchestrator Application

Abstract Model:

Figure below shows the abstract model of the forwarding orchestrator APP. It communicates with ODL and Openstack via RESTful APIs. There is a shared database to store necessary information about various network elements. The modules will be described in the next section.

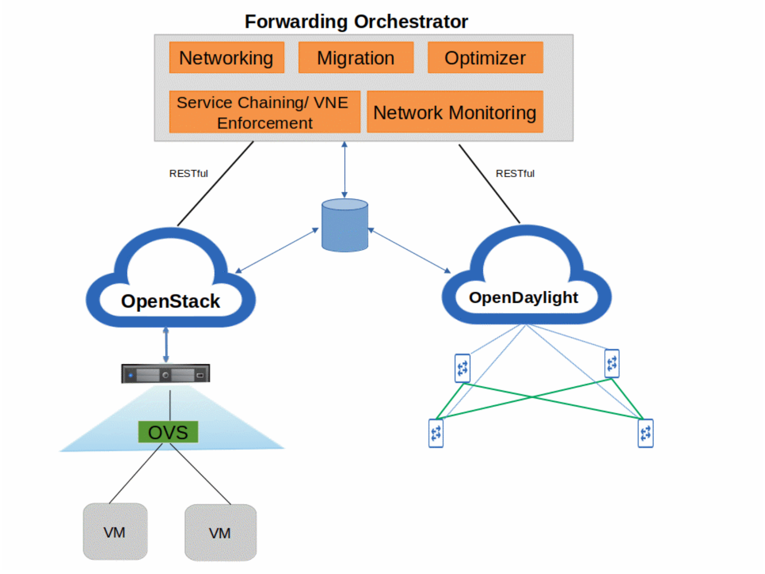
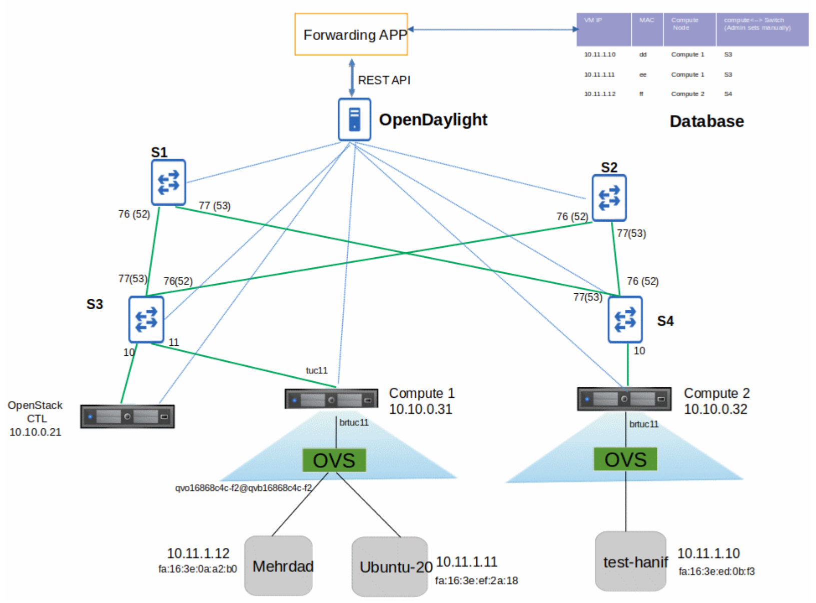


Fig 1. Abstract Model of the Forwarding Orchestrator APP

Actual Network Infrastructure:



VM IP	MAC	Compute Node	compute<--> Switch (Admin sets manually)
10.11.1.10	dd	Compute 1	S3
10.11.1.11	ee	Compute 1	S3
10.11.1.12	ff	Compute 2	S4

Fig 2. Current Network Infrastructure

Forwarding Orchestrator Modules:

The details and task of each module is described in this section.

- ### Network Monitoring
  - Openstack Monitoring
    - Get hosts (VMs) info via RESTful API
      - Where is the VM located? (which compute node)
      - What are the MAC and IP of the VM?
    - Tasks
      - Retrieve information from various services in openstack
        - Connecting neutron for MAC and IP
        - Connecting Nova for getting server information
      - Parsing JSON to retrieve necessary info from it
  - OpenDaylight Monitoring
    - Get network topology information
      - Switch information via LLDP
      - Links between switches (topology API)
      - Get link and port information via Rest API
- ### Networking
  - Forwarding (proactive forwarding)
    - Due to not using micro-service, we have limitations to listen to packet_in packets (reactive forwarding) through RESTful API. Hence, it seems we should develop **proactive forwarding** mechanism.
      - Accordingly, a mechanism is required for signaling in which when a VM is created the corresponding flow entries on related switches being installed proactively.
    - Develop a code to calculate end-2-end forwarding mechanism (python)
    - Layer 2/3 Forwarding
  - Handling ARP mechanisms considering minimum packet flooding (python)
    - Install per-defined permanent rules on switches to allow ARP traffic between different compute nodes.
    - ARP request/reply should forward between compute nodes (end to end) without flooding the whole network
  - Handling LOOP!!??
- ### Policy Enforcement
  - Add/remove flows on corresponding switches (install flow)
  - Enforce and execute migration process after all requirements are fulfilled
  - Execute the VM creation process after all requirements are met

- **Database**

    - Develop a module to insert and retrieve data into database (python)
    - Design database tables
        - Store master key, IP, MAC, Compute node, switch
        - Store network parameters for calculate the path
        - Store VM migration parameters to calculate best place to be migrated
    - Install DB and test insert/fetch new records on the DB via Python

- **Migration**

    - Study how migration should be done!
        - Online or Offline?
        - How it is done? What is the sequence and signaling?
        - What are the requirements?
    - Develop a code to perform migration process via RESTful API (python)
        - Determine migration requirements
            - Calculate the best destination to be migrated
            - Develop a code to send migration parameters to the CPLEX by python
        - Prepare the VM to be migrated (send command to openstack via python)
            - shutdown the VM in offline migration
        - Design and determine migration sequence and process
            - Design and develop the best network path to migrate the VM using migration parameters via CPLEX (Python)
            - Updating necessary changes for IP/MAC of migrated VM
    - Notifying the APP about migration (signaling)

- **Optimizer**

    - Design requirements for optimization (input parameter, algorithm, output)
    - Implement optimization algorithm in python.
    - Apply optimization algorithm to calculate the best end-2-end path in real-time considering network parameters through CPLEX (python API)
    - Apply optimization to find out suitable compute node to host migrated VM via CPLEX

- **Unsolved questions**

    - Can a python app listen to packet-in process or we really needs a micro service inside of the ODL? (should be studied)
    - How our app is gonna speak with other applications?
        - How we will connect to ONAP?

        - How service chaining is performed? What are the connections and signalings of service chaining and our application?
    - How sub modules in the app speaks to each other?
    - Is the proposed model scalable?

- **Project Management**
    - Using GitHub as a common collaborative repository to develop different part of it
    - Using standard coding which can be extendable in the future (Object Oriented Programming: OOP)
    - Task scheduling and assignments