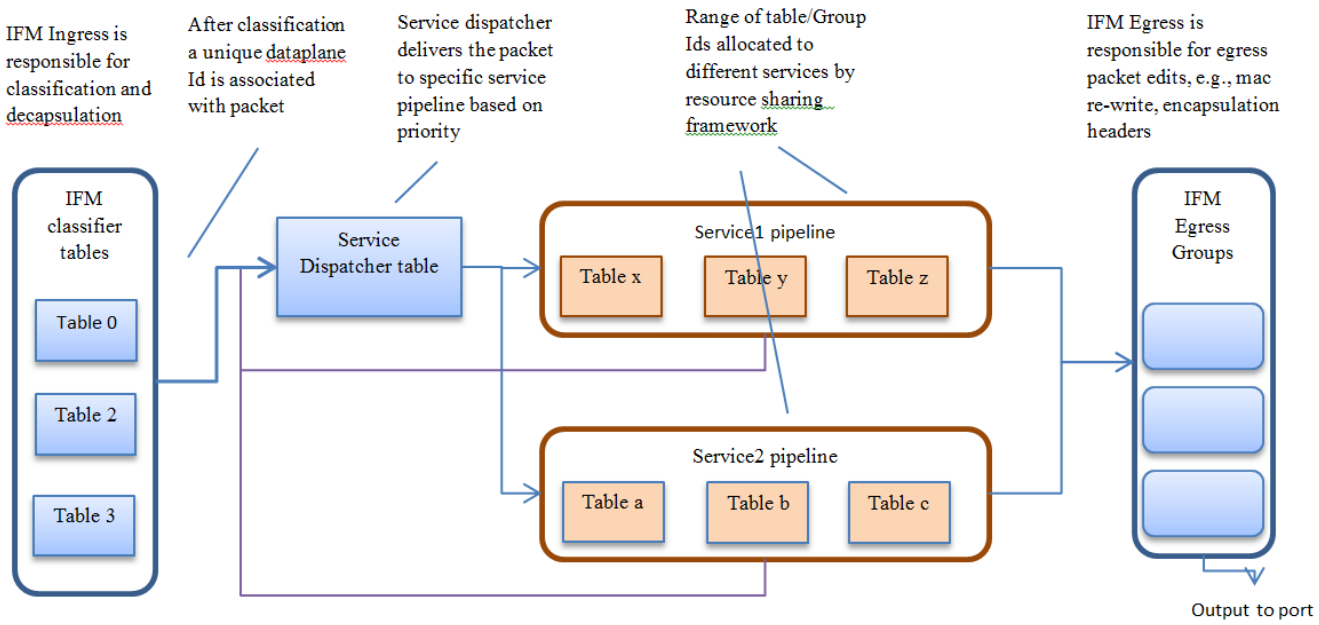
Genius : App co-existence with Genius

Genius provides components and framework for application co-existence. Following diagram gives an overview of how multiple applications/service based on genius components, can work together.



Genius addresses app co-existence issue based on following principles –

- Different applications use isolated set of openflow resources (tables, groups etc.)
    1. Achieved using Resource Sharing/Management framework
- Traffic ingress and egress is handled by a common entity serving multiple applications
    1. Achieved using Interface Manager/ Overlay Tunnel Manager

Contents

Interface Management Service

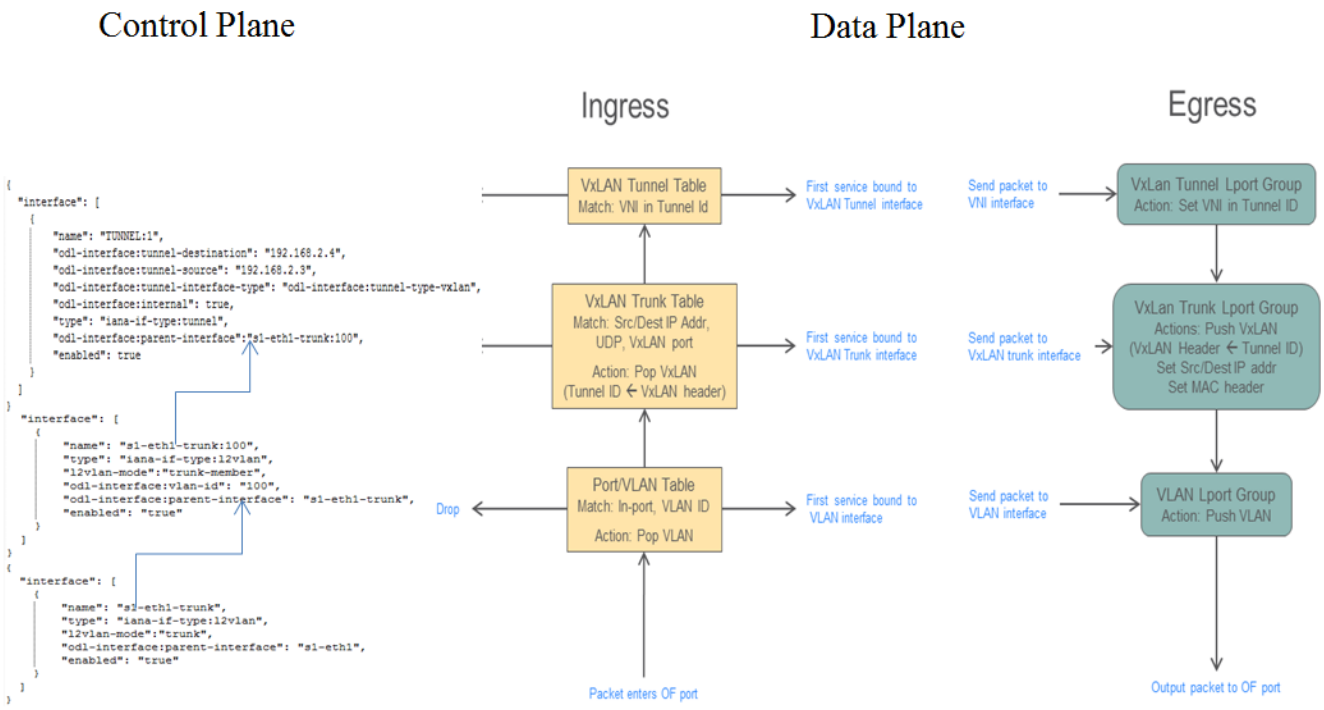
Genius Interface Manager (IFM) provides following main services --

1. Ingress Classification & decapsulation
2. Service Bindings
3. Egress encapsulation

Ingress Classification & decapsulation

Interface Manager allows creation of different granular logical ports/ interfaces for different type of network connections and logical ports, e.g. VLAN ports (classified with in-port+VLAN tag), tunnel (VxLAN & GRE) interfaces classified with ([DPN-id]+[in-port]+[vlan]+local-ip+ remote-ip).

Interface Manager provides Yang based API for creation/configuration of different type of interfaces. Each interface is created with a unique interface-name (e.g. UUID of a neutron port) and its classification parameters. Interface configuration also contains a 'parent-ref' parameter which defines the parent port/interface which feeds to the child interface. This way a hierarchy of interfaces can be created. IFM classification tables in data-plane also form similar hierarchy.



When the parent (openflow) port appears on the south-bound interface, Interface Manager accordingly programs corresponding IFM classifier tables and groups.

When a packet hits table 0, IFM classification rules in table 0 removes the encapsulation/tags and direct packets to appropriate next level of classification table. Once the packet is classified belonging to a particular Interface, corresponding data-plane-id is associated with the packet and the packet is sent to the service dispatcher table.

Service dispatcher table, depending upon the services bind to the interface, apply service specific parameters to the packet and dispatches the packet to appropriate service pipeline. If the packet is not consumed (either dropped or sent out) by the particular service, the service pipeline must send(resubmit) the packet back to the dispatcher table for other services for further processing.

Metadata usage

First 24 bits of metadata is reserved by Interface Manager to carry the 21 bit data-plane-id (if-index) along with 3 bits service-index. Applications/ services must preserve these bits for use of interface manger when the packet is sent back to dispatcher table. If a particular service is sure of consuming the packet and packet is not required to be sent back to dispatcher table, in that case the service is allowed to make use of full 64 bits of the metadata.

Binding Services to Interfaces

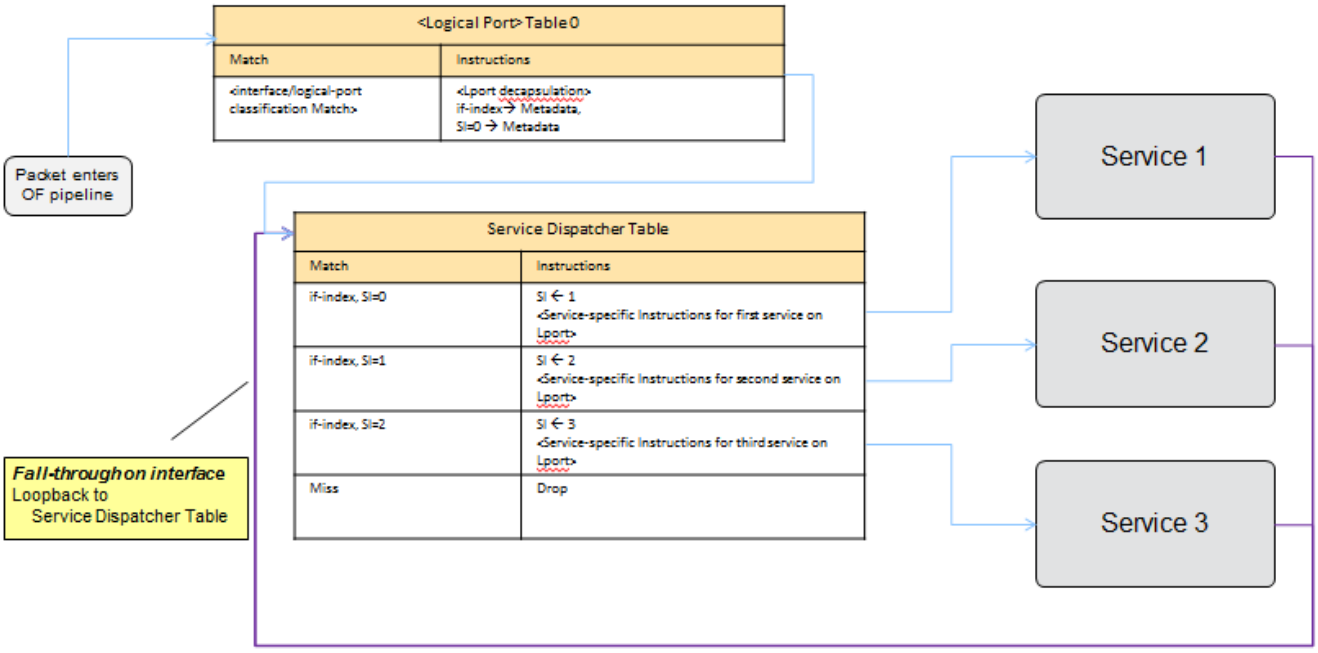
After creation of interfaces, different applications can bind services to it. To bind a service to an Interface, applications will use a Yang based service binding API provided by interface manager. Following are the service binding parameters –

[[File:interface_hierarchy.png]| align:right]

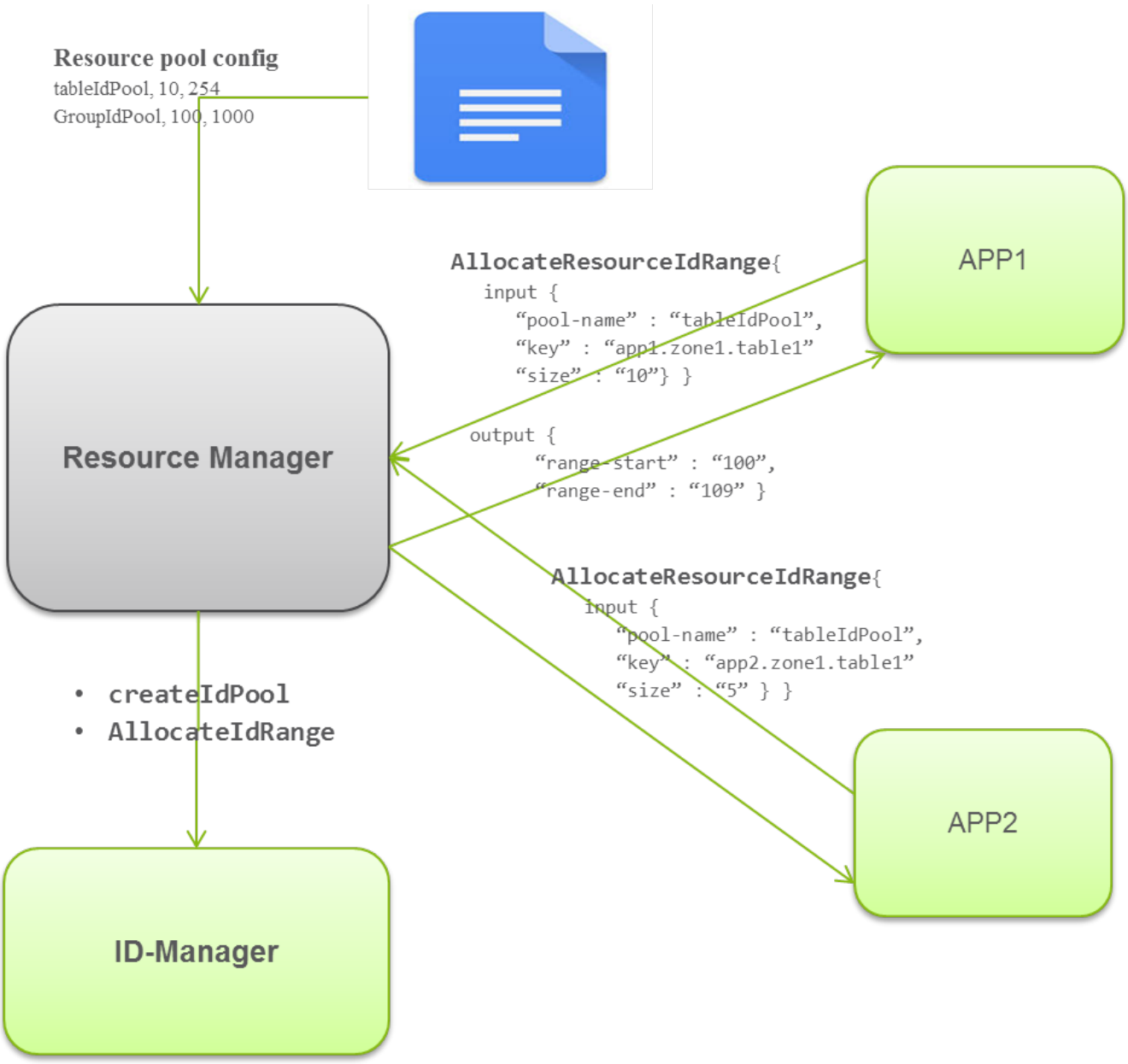
- Service-Priority
- Service-Name
- Service-Info
    1. (for service-type openflow-based)
    2. Flow-priority
    3. Instruction-list

Service priority parameter is used to define the order in which the packet will be delivered to different services bind to the particular interface. Service priorities must be agreed upon by the participating applications to avoid conflict.

When a service is bind to an interface, Interface Manager programs the service dispatcher table with a rule to match on the interface data-plane-id and the service-index (based on priority) and the instruction-set provided by the service/application. Every time when the packet leaves the dispatcher table the service-index (in metadata) is incremented to match the next service rule when the packet is resubmitted back to dispatcher table.



Resource Sharing Framework


Resource pool config
tableIdPool, 10, 254
GroupIdPool, 100, 1000
AllocateResourceIdRange{
input {
"pool-name" : "tableIdPool",
"key" : "app1.zone1.table1"
"size" : "10"} }
APP1
Resource Manager
output {
"range-start" : "100",
"range-end" : "109" }
AllocateResourceIdRange{
input {
"pool-name" : "tableIdPool",
"key" : "app2.zone1.table1"
"size" : "5" } }
APP2
createIdPool
AllocateIdRange
ID-Manager