Virtual Network (Slicing)

高宮 安仁 @yasuhito
Hello World

Switch
Router

Topology Discovery

Routing Switch

Virtual Network
Sliceable Switch

Functions
Network Slice
Schematic of Slices
Sliceable Switch

Routing switch with slice management functions

- Extend a routing switch to enable it to manage network slices
Sliceable Switch

A review of functions of a routing switch
Building Blocks (Classes) of a Routing Switch
When a packet is sent…

PacketIn

FlowMod

PacketOut

Destination Host
Create shortest paths

RoutingSwitch → PathManager → Path

PacketIn → PacketOut

Topology Information

Path.create

FlowMod (From downstream switches)
When a link is torn down…

FlowModDel → PortDown

Destination Host
Delete all invalid paths

PortDown

RoutingSwitch

Topology

delete_link イベント

PathManager

Path

FlowMod (下流から)

(Path.destroy)
Forwarding Messages

- Messages related to topologies
  - RoutingSwitch
  - PathManager
  - Topology

OpenFlow messages

PacketIn messages

Path
Sliceable Switch

Class Structure
Add Classes to Manage Slices

Classes to manage slices

RoutingSwitch

Topology

PathManager

SliceManager

Slice

Path
Shortest Path

On the same slice?

PacketIn

FlowMod

PacketOut

Destination Host
Inheritance and Overriding

- **Topology handler**
  Create a graph for the topology
- **PacketIn handler**
  Create the shortest path
- **Topology handler(Inheritance)**
- **packet_in(Overriding)**
  Check whether target entities are on the same slice before creating the shortest path
def packet_in(_dpid, packet_in):
    slice = Slice.find do |each|
        each.member?(packet_in.slice_source) &&
        each.member?(packet_in.slice_destination(@graph))
    end
    ports = if slice
        path = maybe_create_shortest_path_in_slice(slice.name, packet_in)
        path ? [path.out_port] : []
    else
        external_ports(packet_in)
    end
    packet_out(packet_in.raw_data, ports)
end

On the same slice?
Look up a slice where both src and dst belong

• If such slice exists, forward the packet along the shortest path.
• Otherwise, forward it to all external ports
def self.find(&block)
  all.find(&block)
end

def self.create(name)
  if find_by(name: name)
    fail SliceAlreadyExistsError, "Slice #{name} already exists"
  end
  new(name).tap { |slice| all << slice }
end

Find slices that meet the condition given by block

Create a slice
Encapsulation
- Add/Delete slices
- Add/Delete hosts
Sliceable Switch

Use it as a Component of IaaS
For instance, in the case of Mini IaaS:

Web Interface

REST API
Sliceable Switch

REST API
Docker
<table>
<thead>
<tr>
<th>Description</th>
<th>Method</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a slice</td>
<td>POST</td>
<td>/slices</td>
</tr>
<tr>
<td>Delete a slice</td>
<td>DELETE</td>
<td>/slices</td>
</tr>
<tr>
<td>List slices</td>
<td>GET</td>
<td>/slices</td>
</tr>
<tr>
<td>Shows a slice</td>
<td>GET</td>
<td>/slices/:slice_id</td>
</tr>
<tr>
<td>Add a port to a slice</td>
<td>POST</td>
<td>/slices/:slice_id/ports</td>
</tr>
<tr>
<td>Delete a port from a slice</td>
<td>DELETE</td>
<td>/slices/:slice_id/ports</td>
</tr>
<tr>
<td>List ports</td>
<td>GET</td>
<td>/slices/:slice_id/ports</td>
</tr>
<tr>
<td>Shows a port</td>
<td>GET</td>
<td>/slices/:slice_id/ports/:port_id</td>
</tr>
<tr>
<td>Adds a host to a slice</td>
<td>POST</td>
<td>/slices/:slice_id/ports/:port_id/mac_addresses</td>
</tr>
<tr>
<td>Deletes a host from a slice</td>
<td>DELETE</td>
<td>/slices/:slice_id/ports/:port_id/mac_addresses</td>
</tr>
<tr>
<td>List MAC addresses</td>
<td>GET</td>
<td>/slices/:slice_id/ports/:port_id/mac_addresses</td>
</tr>
<tr>
<td>Shows a MAC address</td>
<td>GET</td>
<td>/slices/:slice_id/ports/:port_id/mac_addresses/:mac_address</td>
</tr>
</tbody>
</table>
Call Method of Remote Classes

Implementing REST API

```ruby
get 'slices/:slice_id' do
  rest_api { Slice.find_by!(name: params[:slice_id]) }
end
```

Call methods of Slice class

Slice Class Proxy

```ruby
class Slice
  def self.find_by!(query)
    # ...
    remote_klass = Trema.trema_process('RoutingSwitch', socket_dir).controller.slice
    remote_klass.find_by!(query)
  end
end
```
Sliceable Switch

Conclusion and Assignment
Conclusion

How to realize virtual networks with OpenFlow

• Extend the routing switch appropriately
  • Add classes to manage slices
  • Use inheritance

• REST API
  • Use the sliceable switch as a component of IaaS
Assignment: Extending a slice function

Add functions to divide a slice and to integrate slices.

Add a function to display slices on a web browser.

Add REST API for the functions of slice division and slice integration.

Run your virtual network on the physical OpenFlow switch
Command
for slice_division

Example:
$ slice split slice_a
    —into slice_b:host1,host3 slice_c:host2,host4