Tunnels as a Connectivity and Segregation Solution for Virtualized Networks

Assaf Muller, Associate Software Engineer, Cloud Networking, Red Hat

assafmuller.wordpress.com, amuller@redhat.com, amuller on Freenode (#openstack)
Networks Topology
Compute Node & VLANs

- **Manually** configure VLANs on physical switches
- VLANs where required – Tedious and rigid
- All VLANs everywhere – Simple but inefficient
- Extend VLANs into the virtual world

**VLAN 100**
**VLAN 200**
Tunnels in the Physical World

- GRE/VXLAN/Tunnel – Like a VPN, but not encrypted
- Connect two sites
- Work from home
- SSH from host (behind NAT) in Tel-Aviv to host in Sydney
- Access site resources
Encapsulation

Brussels
10.0.0.0/8

1.1.1.1

10.0.0.1 -> 172.16.0.1
1.1.1.1 -> 2.2.2.2

Namur
172.16.0.0/16

1.1.1.1

2.2.2.2

Internet

Home
Connectivity

- Tunnels formed between all hypervisors
- VM traffic is encapsulated into traffic between hypervisors
- Hypervisors just need L3 connectivity
- Local connectivity (VMs in same hypervisor) – Same as with VLANs – One shared switch
Segregation

- Tunnel traffic is tagged, much like VLAN traffic
- Each network gets its own tunnel ID
- Incoming traffic can be identified by its tunnel ID
- Local segregation (VMs in the same hypervisor) – Same as with VLANs – Locally significant VLAN tagging
Unicast Traffic

- Reminder – Layer 2 learning switches map incoming port to source MAC
- Virtual switch on hypervisor maps incoming tunnel ID & peer to source MAC
- Learned unicast addresses are persisted, unknown unicast traffic is flooded

<table>
<thead>
<tr>
<th>Port</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C, D, E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Peer &amp; Tunnel ID</th>
<th>Tunnel ID &amp; MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1, 1</td>
<td>1, A</td>
</tr>
<tr>
<td>10.0.0.1, 2</td>
<td>2, B</td>
</tr>
<tr>
<td>10.0.0.2, 3</td>
<td>3, (C, D, E)</td>
</tr>
</tbody>
</table>
Broadcast Traffic

- Unknown unicast, multicast, and broadcast traffic – Historically go out through all tunnels
- Can we do better?
  - Minimize broadcasts – hypervisors answer local ARP requests*
  - Optimize broadcasts - Forward broadcast traffic only to eligible hypervisors**

* ML2 plugin with Linux bridge mechanism driver since Havana. OVS planned for Icehouse
** ML2 plugin since Havana
Open vSwitch

- Open vSwitch bridges operate in one of two modes:
  - Normal mode is a regular layer 2 learning switch
  - Flow mode is entirely custom behavior
- Flows can be configured via:
  - Local ovs-ofctl commands
  - Remote OpenFlow calls
- neutron-openvswitch-agent configures br-tun (Tunneling bridge) via local ovs-ofctl commands, following controller RPC calls
Flows

- Flows have a match and action part:
  - Should the flow process an incoming message?
    - Match against layer 2, 3, 4 headers
  - What to do:
    - Change headers
    - Forward to one or more ports
    - Broadcast
    - Drop
    - Insert new flows
    - Resubmit to another table
Bridges have multiple tables:
- Messages enter table 0
- Messages can be resubmitted to other tables
- Each table's flows are processed by priority, table has implicit drop at the end (Or send message to SDN controller if one is configured)
From a VM on the local node

* Depending if MAC learning mechanism is enabled
From a VM on a remote node

Learn source MAC address, populate table 20

Remove tunnel ID

Add VLAN tag

From a tunnel

Tunnel device X

br-int
More Information

- Official OVS configuration tutorial
- Scott Lowe's (amazing) GRE blog posts
- ovs-vsctl show
- ovs-ofctl dump-flows br-tun
- assafmuller.wordpress.com (Shameless plug!)
Tunnels as a Connectivity and Segregation Solution for Virtualized Networks

Assaf Muller, Associate Software Engineer, Cloud Networking, Red Hat

assafmuller.wordpress.com, amuller@redhat.com, amuller on Freenode (#openstack)