Distributing connectivity management in Cloud-Edge infrastructures: challenges and approaches

David ESPINEL

Journées Cloud 2019 - Toulouse

September 12 2019
Presentation

- Context
- Required networking services
- Networking-related challenges
- Distributed inter-site services module
Context

Traditional IaaS architecture

New emerging needs challenging the centralized approach:

- IoT
- NFV
- MEC
Spanning the VIM to the entire infrastructure

Single centralized model
- SPOF
- Bottleneck
- Network partitioning
- Implementation performance
Context

Leveraging a distributed IaaS architecture

Needs to be closer to the user:

- Delay constraints
- Traffic overhead
- Legal requirements
Context

Leveraging a distributed IaaS architecture: Management questions

How to manage and interconnect networking constructions belonging to different VIMs in a distributed natively way?

While providing:
- Scalability
- Resiliency
- Data locality awareness
- Abstraction & automation
**Required services**

*Layer 2 network extension*

VIM 1

API

Module #1 ...

Distributed VIM Connectivity Management

VIM 2

API

Module #1 ...

Distributed VIM Connectivity Management

VIM 3

API

Module #1 ...

Distributed VIM Connectivity Management

VIM 4

API

Module #1 ...

Distributed VIM Connectivity Management

VIM 5

API

Module #1 ...

Distributed VIM Connectivity Management

VIM N

API

Module #1 ...

Distributed VIM Connectivity Management

10/24

network_A

Instance

VM

Instance

network_B

network_C

Instance

VM

network_D

Instance

VM

network_E

network_F

10/24

10/24

10/24
Required services

Routing function

VIM 2

API

Module #1

Distributed VIM Connectivity Management

VIM 3

API

Module #1

Distributed VIM Connectivity Management

VIM 4

API

Module #1

Distributed VIM Connectivity Management

VIM 5

API

Module #1

Distributed VIM Connectivity Management

VIM N

API

Module #1

Distributed VIM Connectivity Management

20.0.2.0/24

30.0.2.0/24

network_E

network_B

network_F

30.0.2.0/24
Required services

Traffic filtering, policies & QoS
Required services  

Service Function Chaining

VIM 1  
API  
Module #1  
Distributed VIM Connectivity Management

VIM 2  
API  
Module #1  
Distributed VIM Connectivity Management

VIM 3  
API  
Module #1  
Distributed VIM Connectivity Management

VIM 4  
API  
Module #1  
Distributed VIM Connectivity Management

VIM 5  
API  
Module #1  
Distributed VIM Connectivity Management

VIM N  
API  
Module #1  
Distributed VIM Connectivity Management

IP/MPLS  

Distributed VIM Connectivity Management

Required services  
Service Function Chaining
Information granularity

- Which information for which service?
- Identifying how information should be shared
- Different sharding strategies

**Challenge #1**

L2 extension among net_y (VIM 2) and net_x (VIM 1)
Information scope

Challenge #2

- Avoid unnecessary management traffic
- Share info only when requested
- Contact only the relevant sites
- Sharing networking information on-demand and in an efficient manner
Information availability

Challenge #3

- Each VIM should be able to deliver network services
- Non-disconnected sites continue to provide inter-site services
- Inter-site traffic forwarding needs to be assured
Automatized interfaces

**Challenge #4**

- Management integration challenge
- Smoothly bridge of the two interfaces
- Abstracted and automated process
Distributed inter-site services module

DISCO & ODL (Federation + NetVirt) inspired

- Type of service (L2 or L3/ QoS and SFC pending)
- List of N resources that will compose the service

OpenStack #N / RegionN

Neutron Server

ML2 Core Plugin
Neutron Inter Service Plugin

User-face API
Module-face API
Inter-site service logic engine
DB

Inter-site Module

Vertical interface
Horizontal interface
Remote module N+1
Remote module N+2
Remote module N+3
Distributed inter-site services module

DISCO & ODL (Federation + NetVirt) inspired

Algorithm 4 Vertical inter-site networking services CREATE request

Require: \( N_i \) : net id of network, in Site\( i \)

procedure

SEND: Vertical API CREATE inter-site \{type : L3/L2, resources : \( \{N_i, N(i+1), N(i+2), ..., N_{n-1}, N_{n}\} \)\} to Site\( i \)

function Vertical API CREATE inter-site(type, resources):

local ← resources.findLocal()
resources_subnets ← \{
resources_subnets ← get(N_i, getSubnet()_{n=1}^n)
subnet_CIDRs ← \{
subnet_CIDRs ← get(resources_subnets, getCIDR()_{n=1}^n)

if type = L3 then
    overlappingAddresses ← \bigcap_{i=1}^n \text{subnet_CIDRs}_i
    if overlappingAddresses! = \{
        error : overlapping addresses
    \}

if type = L2 then
    main_CIDR ← subnet_CIDRs[0]
    available_\text{ips} ← 2^{32 - \text{main_CIDR}.\text{host_bits}} - 3
    total_sites ← \text{len(resources)}
    distributed_ranges ← divide.ip.range(main_CIDR, available_\text{ips}, total_sites)
    local_params ← distributed_ranges[0]

for object in remotes do
    if object! = local then
        interconnections.add(interconnection.create(type, local, object))
    id ← GENERATE.ID()
    inter-site_service ← new.Service(id, type, local, params, resources, interconnections)
    inter-site_service.db.commit()

if type = L2 then
    update.local_subnet(local, params)
    update.vms_ip(local)

for object in remotes do
    if object! = local then
        if type = L2 then
            params ← distributed_ranges[object.index]
        SEND: Horizontal API CREATE inter-site Site\( i \)(object.index)

\{id : id, type : type, remotes : resources, params : params\}

return 201, Created inter-site services

Collecting information from the resources

Per service type sharding strategy

Doing local changes

Send service request to remote modules
Distributed inter-site services module

DISCO & ODL (Federation + NetVirt) inspired

**Algorithm 5** Horizontal inter-site networking services CREATE request

**Require:** \( N_i : net \) id of network in Site

**procedure**

SEND: Horizontal API CREATE inter-site \{global id : id, type : L3, params : params, resources : \( \{N_i, N_i+1, N_i+2, \ldots, N_n-1, N_n\}_{i=1}^{n} \)\} to Site

function HORIZONTAL API CREATE inter-

SITE(id, type, params, resources):

\[
\begin{align*}
local & \leftarrow resources.findLocal() \\
local.params & \leftarrow params \\
for & \text{object in remotes do} \\
if & \text{object}! = \text{local then} \\
\quad & \text{interconnections.add}([\text{interconnection.create(type, local, object)}) \\
if & \text{type} == 'L2' then \\
\quad & \text{update_local_subnet(local.params)} \\
\quad & \text{update_vms_ips}(local) \\
\text{inter-site.service} & \leftarrow \text{new Service(id, type, local.params, resources, interconnections)} \\
\text{inter-site.service.db.commit}() \\
\text{return} & \text{201, Created inter-site service}
\end{align*}
\]

Doing local changes
**DEMO: Layer 2 network extension**

**net_x: (Initial)**
*Allocation-pool:* 10.0.0.2-10.0.0.254  
*VM_A IP:* 10.0.0.8

**net_x: (Final)**
*Allocation-pool:* 10.0.0.3-10.0.0.128  
*VM_A IP:* 10.0.0.8

**net_y: (Initial)**
*Allocation-pool:* 10.0.0.2-10.0.0.254  
*VM_B IP:* 10.0.0.10

**net_y: (Final)**
*Allocation-pool:* 10.0.0.129-10.0.0.254  
*VM_B IP:* 10.0.0.135
Next steps & open questions

- Improve the code for the create and update requests
- Add support for networking disconnections
- Test the performance (Grid’5000)
- (For the L2 strategy) Until which point can we shrink the allocation pool to add new sites to a service?
- (For the L2 strategy) CIDR division or global IP sharing?
Thanks !
References


