Fully Automated Peer Service Orchestration of Cloud and Network Resources Using ACTN and CSO

Ricard Vilalta, Ramon Casellas, Ricardo Martínez, Raul Muñoz
Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA)
Email: ricard.vilalta@cttc.es

Young Lee, Haomian Zheng, Yi Lin
Huawei

Victor López, Luis Miguel Contreras
Telefónica I+D/Global CTO

Abstract—**Current applications, such as on-line gaming or media content delivery, require dynamic bandwidth allocation and a tight integration between the network resources and computing resources. Service orchestration faces the challenge of combining and controlling the resources of these different stratums and optimizing them.**

This paper proposes the fully automated establishment of a network service using a peer inter-CSO interface in ACTN. The underlying network resources have been abstracted and virtualized in order to provide a network slice. We present the CSO/ACTN architecture and detail the main components. The system is implemented and demonstrated in an experimental testbed, where we characterize the setup delay of a virtual deep packet inspection service across several network domains using the proposed peer interface.

I. Introduction

Cloud computing is able to provide a variety of services (such as on-line gaming, media content delivery, network slicing). These services allow end-users to access large pools of compute and storage resources, enabling various application services (e.g., Video Caching, VM mobility, media content delivery, IoT, etc.). Cloud computing services is one of the faster emerging businesses for Internet Server Providers (ISP).

Data centers (DC) provide the physical and virtual infrastructure in which cloud computing applications are deployed and chained into end-services. The proposed services might be also aligned with Network Function Virtualization (NFV) services. Since the DCs used to provide services may be distributed geographically around a set of interconnected networks, service deployment can affect on the state of the network resources. Conversely the capabilities and current state of the network can have a major impact on the service performance; DCs have been spread geographically to reduce latency to the end user, and that has led into an exponential growth on the inter-datacenter traffic [1]. Consequently, DC interconnection is one of the major problems that service providers have to face, along the need to adapt the actual rigid and fixed transport networks, enabling them with the flexibility provided with the Software Defined Networking (SDN) architecture.

SDN is the solution to improve network programmability, including the dynamic allocation of the network resources. SDN proposes a centralized architecture where the control entity (SDN controller) is responsible for providing an abstraction of network resources through programmable Application Programming Interfaces (APIs). One of the main benefits of this architecture resides on the ability to perform control and management tasks of different network forwarding technologies such as packet/flow switches, circuit switching and optical wavelength switched transport technologies. In the IETF new data models are currently under definition, in order to allow the integration of abstracted traffic enginery (TE) information in the description of network resources [2] and in the allocation of these resources [3].

Moreover, Abstraction and Control of Traffic Engineered networks (ACTN) enables virtual network operations using abstraction and virtualization mechanisms [4]. It allows customers to request a virtual network over operators transport networks, which are often multi-layer and multi-domain TE networks. This virtual network is presented as an abstract topology to customers, in such a way that they can use this abstracted topology to offer applications over its virtual network. Therefore, ACTN enables multi-tenant virtual network services with flexibility and dynamicity. Along ACTN, and in order to deal with the joint allocation of DC and network resources, Cross Stratum Optimization (CSO) [5] involves a cooperation between the Application Stratum and Network Stratum to efficiently utilize cloud and network resources and provide for overall application level quality of service.

In this paper we present a possible CSO architecture involving multiple CSO service orchestrators, which interact in a distributed (peer) model with the objective of provisioning an End-to-End (E2E) service over multiple administrative network and cloud domains. A virtual Deep Packet Inspection (vDPI) service is provided as our use case, and we measure the setup delay in deploying the service using the cloud computing platform of the ADRENALINE testbed [6].

II. State of the Art

In this section we will present the current state of the art of both the ACTN and CSO functional architectures, setting the basis for highlighting their complementarity and how they can interact and be used to fulfill our purpose of dynamic service establishment over peer-CSO service orchestrators using ACTN.

A. Abstraction and Control of Traffic Engineered Networks

ACTN provides an architecture for the virtualization of TE networks. The architecture defines multiple functional entities (controllers) according to their main functions and roles. In this setting, the types of controller defined in the ACTN architecture are shown in Fig. 1 and are as follows:

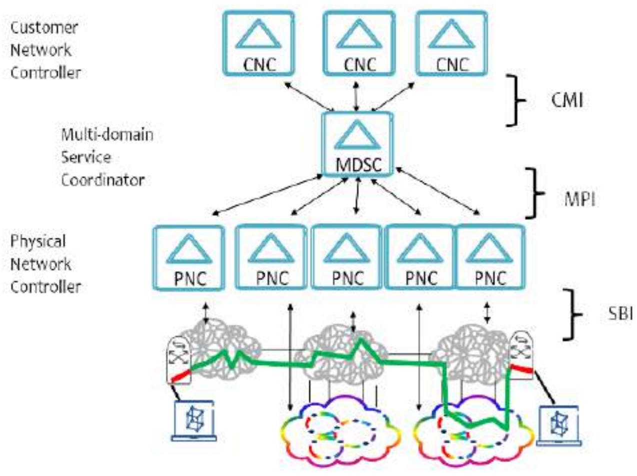

Customer Network Controller
CNC
CNC
CNC
CMI
Multi-domain Service Coordinator
MDSC
MPI
Physical Network Controller
PNC
PNC
PNC
PNC
PNC
SBI

Fig. 1. ACTN architecture

1) Customer Network Controller (CNC): A Virtual Network is requested by the Customer Network Controller via the CNC-MDSC Interface (CMI). As the Customer Network Controller directly interfaces to the applications, it is able to understand multiple application requirements and their needs. It is assumed that the CNC and the MDSC have a common knowledge of the end-point interfaces based on their business negotiations prior to service instantiation.

2) Multi Domain Service Coordinator (MDSC): The Multi Domain Service Coordinator (MDSC) sits between the CNC that issues VN requests and the Physical Network Controllers (PNCs) that manage the physical network resources. The MDSC is the responsible for the following functions: multi-domain coordination, virtualization/abstraction, customer mapping/translation, and virtual service coordination. Multi-domain coordination and virtualization/abstraction are referred to as network control/coordination functions. While customer mapping/translation and virtual service coordination are referred to as service control/coordination functions. The key point of the MDSC is detaching the network and service control from underlying technology to help the customer express the network as desired by business needs. The MDSC provides the deployment and control of the right technology to meet business criteria. In essence it controls and manages the primitives to achieve functionalities as desired by the CNC. A hierarchy of MDSCs can be foreseen for scalability and administrative choices.

3) Physical Network Controller (PNC): The Physical Network Controller (PNC) refers to a standard SDN controller, which allows configuration of the network elements, monitoring the topology (physical or virtual) of the network, and passing information about the topology (either raw or abstracted) to the MDSC.

Moreover, a data model for the control of a data network has also been proposed in [7]. This data model can be applied as the NBI of a MDSC. It provides the basic elements for Create/Read/Update/Delete (CRUD) operations over ACTN VNs. It also describes two main ACTN elements:

- VN member: The VN can be understood as set of end-to-end tunnels from a customer point of view, where each tunnel is known as a VN member. Each VN member might be formed by recursive abstraction of paths in underlying networks.
- Access point (AP): An access point provides confidentiality between the customer and the provider. It is a logical identifier shared between the customer and the provider, used to map the end points of the border node in both the customer and the provider network. A set of APs are used by the customer when requesting VN service to the provider.

B. Cross Stratum Optimization

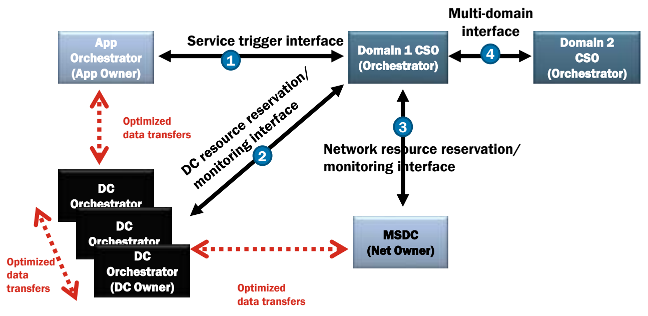

Multi-domain interface
App Orchestrator (App Owner)
Service trigger interface
1
Domain 1 CSO (Orchestrator)
4
Domain 2 CSO (Orchestrator)
Optimized data transfers
DC resource reservation/ monitoring interface
2
3
Network resource reservation/ monitoring interface
DC Orchestrator
DC Orchestrator
DC Orchestrator (DC Owner)
MSDC (Net Owner)
Optimized data transfers
Optimized data transfers

Fig. 2. CSO architecture

Cross Stratum Optimization (CSO) involves a cooperation between the Application Stratum and Network Stratum in order to optimize cloud and network resources and provide for overall application level quality of service.

The *Application stratum* is the functional grouping which considers application resources and the control and management of these resources. These application resources are used along with network services to provide an application service to customers. Application resources are non-network resources critical to achieving the application service functionality. Examples of application resources include: caches, mirrors, application specific servers, content, large data sets, and computing and storage power. Application service is a networked application offered to a variety of clients. Several application service examples are: server backup, VM migration, video cache, virtual network on-demand, 5G network slicing. The entity responsible for application stratum control and management of its resources is referred to as application orchestrator.

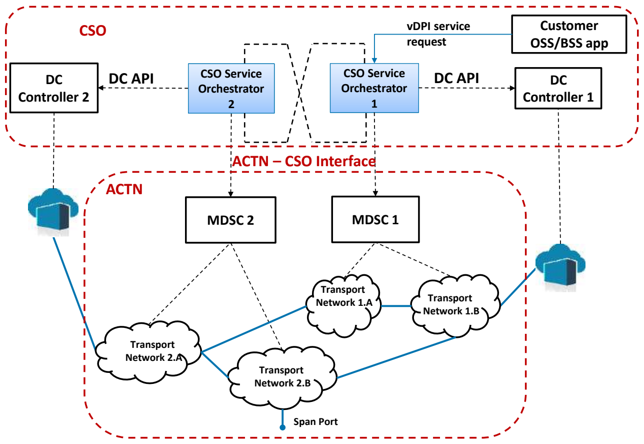


Fig. 3. Proposed ACTN-CSO architecture

The *Network stratum* is the functional grouping which includes network resources and the control and management of these resources providing transport of data between customers and application sources. Network resources are resources such as bandwidth, links, paths, path processing (creation, deletion, and management), network databases, path computation, admission control, and resource reservation. There are different types of network stratum controllers/orchestrators.

Figure 2 summarizes the CSO architecture, where the data center orchestrator provides its data center network resource abstraction pertaining to the applications to the CSO. Since application services may use resources in multiple data centers via data center interconnection, each data center orchestrator involving in the application service should provide its resource abstraction to the CSO so that the CSO would be able to compute the optimal resource sequence and path meeting the service objective.

The Wide Are Network (WAN) SDN controller(s) is(are) also involved for an end-to-end service instantiation and its life cycle operation that may traverse multiple data centers dispersed in multiple domains. WAN SDN controllers may also comprise multiple hierarchical SDN controllers, each of which is responsible for domain control of IP or optical networks in multiple domain networks.

The offered interfaces are the following:

- CSO interface type 1 is referred to as the Service Trigger Interface. This interface shall be able to describe service requirements, taking into consideration DC and network requisites.
- CSO interface type 2 is referred to as the DC Resource Reservation and Monitoring Interface. This interface on a high level abstracts a DC level resource abstraction as well as a host/server level resource abstraction needed per application.
- CSO interface type 3 is referred to as the Network Resource Reservation and Monitoring Interface. This interface should provide several functionalities such as: a) abstraction of the network resource information of the operators transport networks providing DC interconnection; b) service connection reservation request; c) monitoring data and measurement pertaining to the service connection among the key requirements.
- CSO interface type 4 is referred to as multi-domain CSO interface.

III. Proposed ACTN/CSO Joint Architecture

Figure 3 shows the proposed architecture, which is able to jointly orchestrate IT and network resources. It consists of four main building blocks: the customer application, CSO service orchestrator, DC controller and Multi-domain Service Coordinator (MDSC). The CSO service orchestrator offers the NorthBound Interface (NBI) for the dynamic service deployment, considering the necessary joint orchestration of IT and network resources.

Coupling the ACTN with the CSO, an end-to-end automated orchestration of services/applications is made possible with the joint optimization of cloud and network resources the applications consume. The MDCS of the ACTN architecture acts as a network orchestrator of multi-layer and multi-domain networks. A CSO Orchestrator is able to request virtual networks to each MDSC. The CSO is the entity that has the application requirements knowledge and the necessary cloud/DC resources associated with the application. A CSO in one operator domain interacts with another operator domains CSO as the applications are provided across multiple operator domains.

The CSO service orchestrator is responsible for peer coordination with other CSO service orchestrators in other administrative domains. Moreover, each CSO service orchestrator is responsible for its domain network and cloud resources. In the following subsections, CSO service orchestrator is detailed.

The DC controller is responsible for the creation/migration/deletion of VM instances (computing service), disk images storage (image service), and the management of the VM network interfaces (networking service). The computing service (e.g., Nova in OpenStack) is responsible for the management of the VM into the compute hosts (i.e., hypervisors). A compute service agent is running in each host and controls the computing hypervisor (e.g., KVM) responsible of the creation/deletion of the VMs. The image service (e.g., Glance in OpenStack) handles the disk images which are used as templates for VM file systems; it also operates in a centralized manner by maintaining a copy of all the disk images in the DC controller. An image-service agent is running in each host to request the download of images when a new VM instantiation requires it. It also permits to create new images of the currently working instances, a process known as snapshotting, which is used for VM migration. Finally, the connectivity between VMs and virtual switches inside the hosts is managed by the networking service (e.g., Neutron in OpenStack). It creates the virtual interfaces, attaches them into the virtual switches, such as OpenVSwitch, and it offers a DHCP service for the VMs to get the assigned IP address. The CSO service orchestrator controls the DC controller through a RESTful API (e.g., OpenStack API), which is used to trigger the DC controller actions and to get the necessary information about the running VM instances.

The MDSC is introduced in order to support end-to-end connectivity by orchestrating the different network technologies or control domains. In the proposed network architecture, the inter-DC network is controlled using IETF TEAS topology [2] and tunnel [3] data models using RESTconf protocol [8]. We have based the MDSC architecture on the IETF ABNO [9]. The ABNO is the IETF reference architecture for SDN controllers, where it reuses standardized components, such as path computation element (PCE). The Topology Server, included in the ABNO, is the component responsible of gathering the network topology from each control domain and building the whole network topology which it is stored in the Traffic Engineering Databased (TED). The TED includes all the information about network links and nodes, and it is used by the PCE for calculating routes across the network. The Virtual Network Topology Manager (VNTM) is the responsible for managing the multi-layer provisioning. In the proposed architecture, the VNTM will arrange the set-up of an optical connection and offer it as a L2 logical link to satisfy the L2 connectivity demand. The Provisioning Manager implements the different provisioning interfaces to push the forwarding rules and the establishment of segments into the data plane. Flow server stores the connections established in the network into a FlowDB. Finally the Network Orchestration Controller handles all the processes involved inside the MDSC to satisfy the provisioning of end-to-end connectivity.

A. Customer Network Service and Inter-CSO Peering model

Figure 4 shows the customer view offered from CSO service orchestrator 1 to the Customer application. A virtual network is provisioned using ACTN, through requesting several VN members. Each VN member is an e2e tunnel from the customer perspective. It can be observed that the served VN is an abstract construct on top of Transport Network 2.B and 1.B, each from a different domain and controlled by a different peer CSO service orchestrator. The costumer application only has the necessary information regarding the deployed application and the VN interconnecting it. In this case the customer application is a vDPI, where a span port (where the traffic to be analyzed is captured) is interconnected to the vDPI application running in DC1.

B. Proposed CSO service orchestrator architecture

Figure 5 shows the internal architecture for CSO service orchestrator. It can be observed that the main component is the internal orchestrator, which is the responsible for providing

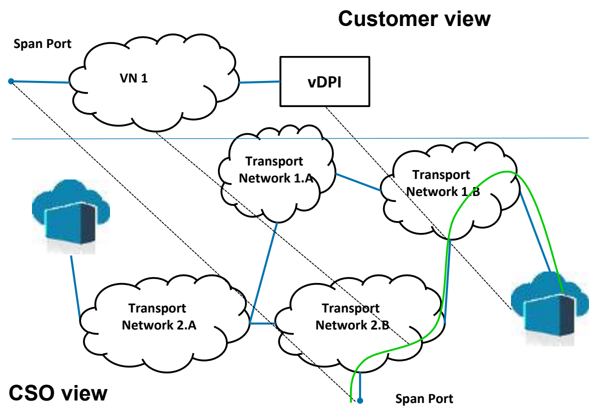


Fig. 4. Customer's view from application perspective

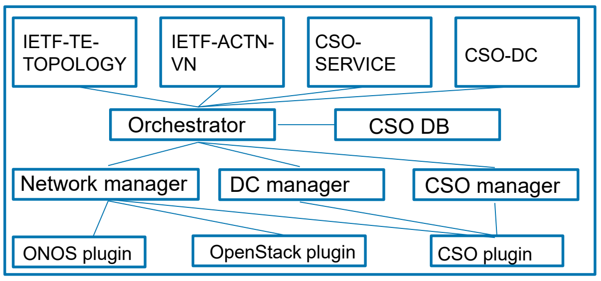


Fig. 5. Proposed CSO service orchestrator architecture

the necessary workflows in order to deploy, modify and tear down services. On top, four NBIs are offered:

1) CSO Service: This interface is offered for the end-user. It provides network intents and compute intents in order to scale in/out the requested service, which might include several applications.

2) CSO-DC: This interface is used to provide computing and storage resources to end-user and to other CSO service orchestrators. It provides an abstraction to OpenStack API, in order to offer a generalized compute and storage service, which might be offered using different underlying controllers.

3) ACTN VN: It provides VN resources to end user and to other CSO service orchestrators. A list of access points is provided, which represent the possible end points of the VN members.

4) TE Topology: A more detailed view of network topology might be obtained using this interface by a peer CSO in order to perform better network resource optimization.

The CSO database (DB) is used to keep synchronized all the information related to peer CSOs, as well as the information regarding the underlying controlled DCs and MDSCs. Finally, the Network, DC, and CSO manager are the responsible for each of the underlying resources, and they use the necessary plugins (to consume the NBI defined for each OSS, such as

ONOS and OpenStack).

C. Resource allocation algorithm which considers Compute and Network resources

A heuristic algorithm for the CSO has been developed in is further explained in [10]. Once the CSO receives a request, it allocates the necessary compute and networking resources by using the previously recovered information. The CSO heuristic algorithm receives the following inputs: Network Service requested (including compute requirements, and network interconnections and span ports). The algorithm outputs are the VM allocation per DC, and the necessary VM interconnections though transport networks.

D. Fully Automated Service workflows

Figure 6 shows the information recovery workflow from the CSO service orchestrator 1 (CSO1) perspective in order to discover all the cloud and network available resources in a peer hierarchical approach as has been presented in [11]. Four steps are included in this process:

1) Discovery of underlying DC resources using OpenStack API [12] towards DC controller.
2) Discovery of underlying Network resources using TE Topology towards MDSC1.
3) Discovery of peer CSO service orchestrator 2 (CSO2) DC resources using CSO DC interface. CSO2 will initiate discovery of underlying DC resources.
4) Discovery of available access points in CSO2 in order to provide network resources.

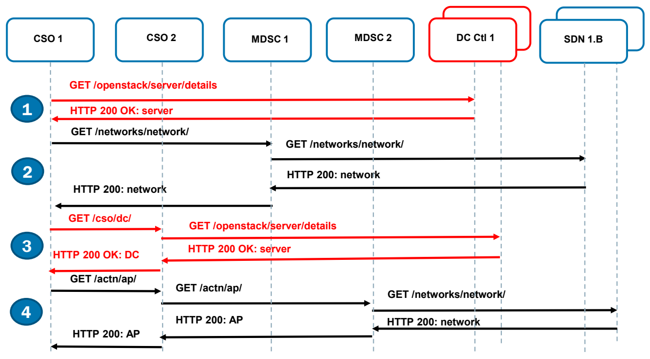


Fig. 6. Message exchange for inititializing CSO1 and CSO2

Figure 7 show the message exchange in order to dynamically deploy a service, such as the depicted in the reference scenario (Fig. 3). It consists of the following steps:

1) A customer requests a service to CSO1.
2) The resource allocation algorithm is triggered and allocates the necessary VM in DC1.
3) It also computes the necessary virtual network members, and it triggers the virtual network creation. CSO1 requests to CSO2 a virtual network, in order to reach the requested span port. CSO2 forwards the request to the MDSC under its domain.
4) A virtual network is directly requested to MDSC1. Each request triggers the creation of a TE tunnel.
5) Finally, the CSO1 offers the resource view to the customer, as seen in Figure 4.

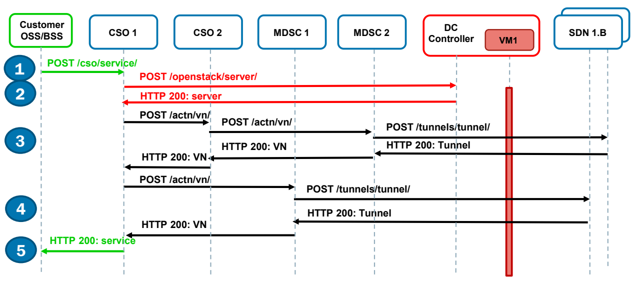


Fig. 7. Message exchange for provisioning service

IV. Experimental Demonstration

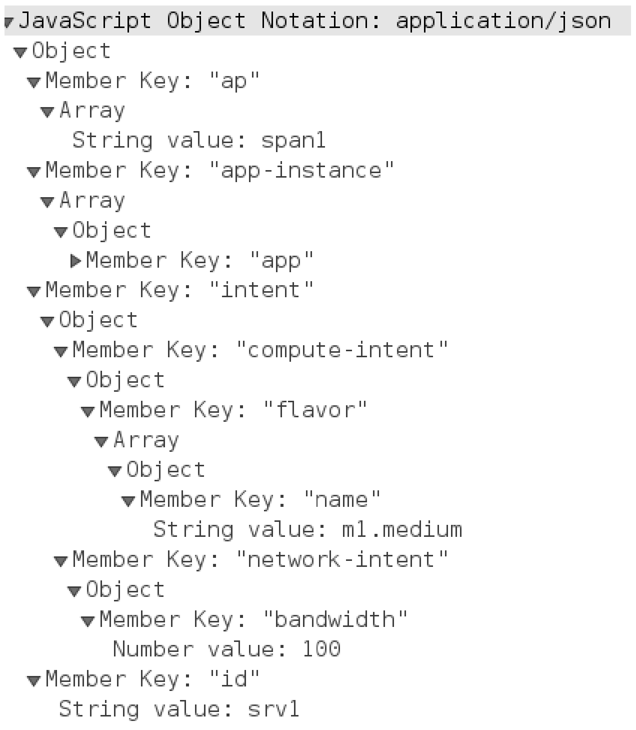
The proposed architecture has been validated in the Cloud Computing platform of the ADRENALINE Testbed. The OpenStack Havanna release has been deployed into five physical servers with 2 x Intel Xeon E5-2420 and 32GB RAM each, one dedicated to the cloud controller and the other four as compute pool (hypervisors) for VM instantiation [13]. The network resources have been emulated using mininet and ONOS, on top of which we have included an agent to provide the necessary TE topology and ACTN VN interfaces. The CSO service orchestrator has been developed using python and its interfaces how been generated using swagger codegen.

In Figure 8, a JSON object of the requested service can be observed. The span port is included in the access point array, as well as the network and compute intents (including bandwidth and flavor, respectively). The application to be deployed (in this case vDPI) is described in app-instance array.

Figure 9 shows the HTTP conversation between CSO1 and CSO2 service orchestrators at their initialization. Firstly, it can be observed how CSO1 loads the information from underlying DC controller 1 (OPS1). Secondly, CSO1 requests to MDSC1 the topological network information. Thirdly, DC status of resources is requested to CSO2. Finally, access points are requested.

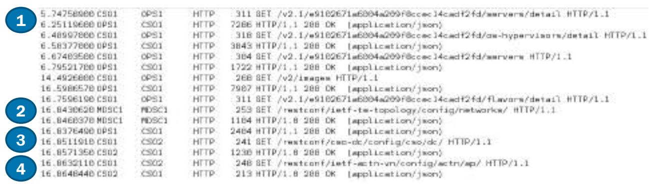
Figure 10 shows the capture HTTP message exchange between CSO1 and CSO2 in order to provision a service requested by client. Firstly, we can observe the requested service. Secondly, A VM is deployed in order to support the requested application. CSO1 checks the status of the VM until the VM is fully available. Thirdly, CSO1 request to MDSC1 the creation of a VN. Fourthly, CSO1 requests CSO2 the creation of a VN, considering the span access point. Finally, the service is provided to the client.

It can be observed in Figure 10 that the setup delay to provision a service is of 14.7 seconds, being the main delay contributor, the span of the VM to deploy the requested application. The necessary networking resources have been allocated in 132 milliseconds.



```
▼JavaScript Object Notation: application/json
 ▼Object
  ▼Member Key: "ap"
   ▼Array
     String value: span1
  ▼Member Key: "app-instance"
   ▼Array
    ▼Object
     ▶Member Key: "app"
  ▼Member Key: "intent"
   ▼Object
    ▼Member Key: "compute-intent"
     ▼Object
      ▼Member Key: "flavor"
       ▼Array
        ▼Object
         ▼Member Key: "name"
           String value: m1.medium
    ▼Member Key: "network-intent"
     ▼Object
      ▼Member Key: "bandwidth"
        Number value: 100
  ▼Member Key: "id"
    String value: srv1
```

Fig. 8. Wireshark capture of requested service JSON object



```
5.74758000 CSO1   OPS1   HTTP   311 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/detail HTTP/1.1
6.25119600 OPS1   CSO1   HTTP  7206 HTTP/1.1 200 OK  (application/json)
6.48997000 CSO1   OPS1   HTTP   318 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/os-hypervisors/detail HTTP/1.1
6.58377000 OPS1   CSO1   HTTP  3843 HTTP/1.1 200 OK  (application/json)
6.67403500 CSO1   OPS1   HTTP   304 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers HTTP/1.1
6.79521700 OPS1   CSO1   HTTP  1722 HTTP/1.1 200 OK  (application/json)
14.4926000 CSO1   OPS1   HTTP   268 GET /v2/images HTTP/1.1
16.5986570 OPS1   CSO1   HTTP  7907 HTTP/1.1 200 OK  (application/json)
16.7596190 CSO1   OPS1   HTTP   311 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/flavors/detail HTTP/1.1
16.8430620 MDSC1  MDSC1  HTTP   253 GET /restconf/ietf-te-topology/config/networks/ HTTP/1.1
16.8460370 MDSC1  MDSC1  HTTP  1104 HTTP/1.0 200 OK  (application/json)
16.8376490 OPS1   CSO1   HTTP  2404 HTTP/1.1 200 OK  (application/json)
16.8511910 CSO1   CSO2   HTTP   241 GET /restconf/cso-dc/config/cso/dc/ HTTP/1.1
16.8571350 CSO2   CSO1   HTTP  1230 HTTP/1.0 200 OK  (application/json)
16.8632110 CSO1   CSO2   HTTP   240 GET /restconf/ietf-actn-vn/config/actn/ap/ HTTP/1.1
16.8648440 CSO2   CSO1   HTTP   213 HTTP/1.0 200 OK  (application/json)
```

Fig. 9. Captured wireshark message exchange for initializing CSO1 and CSO2

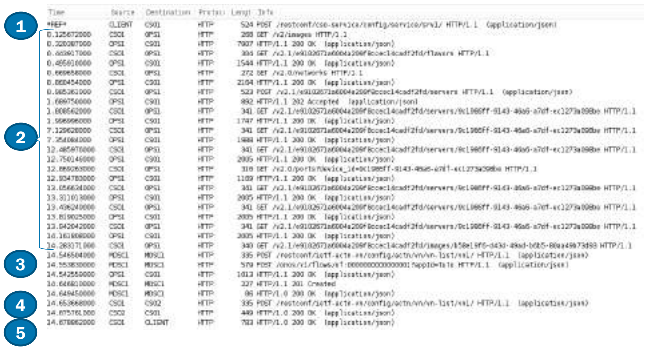
V. Conclusion

Several innovations are presented in this paper. An inter-CSO peer interface has been introduced, by providing descriptions to consider DC/cloud resources and VN resources. Internal CSO service orchestrator has been described. Also, SDN controller support for TEAS Topology and ACTN VN interfaces has been demonstrated. Finally, a proof of concept has been experimentally demonstrated in order to validate the proposed architecture.

As further work, scalability of the proposed architecture should be addressed, in terms of necessary setup requests, as well as scale in/down mechanisms for the requested resources.

Acknowledgments

This research has been partly funded by EC H2020 5GPPP 5GTANGO (H2020-ICT-2016-2 761493) and Spanish MINECO project DESTELLO (TEC2015-69256-R).



```
Time         Source  Destination  Protocol Length Info
*REF*        CLIENT  CSO1         HTTP     524 POST /restconf/cso-service/config/service/srv1/ HTTP/1.1  (application/json)
0.125672000  CSO1    OPS1         HTTP     268 GET /v2/images HTTP/1.1
0.320307000  OPS1    CSO1         HTTP    7907 HTTP/1.1 200 OK  (application/json)
0.443917000  CSO1    OPS1         HTTP     304 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/flavors HTTP/1.1
0.485010000  OPS1    CSO1         HTTP    1544 HTTP/1.1 200 OK  (application/json)
0.669658000  CSO1    OPS1         HTTP     272 GET /v2.0/networks HTTP/1.1
0.880454000  OPS1    CSO1         HTTP    2104 HTTP/1.1 200 OK  (application/json)
0.985361000  CSO1    OPS1         HTTP     523 POST /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers HTTP/1.1  (application/json)
1.689750000  OPS1    CSO1         HTTP     892 HTTP/1.1 202 Accepted  (application/json)
1.808562000  CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
1.998996000  OPS1    CSO1         HTTP    1747 HTTP/1.1 200 OK  (application/json)
7.129628000  CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
7.354084000  OPS1    CSO1         HTTP    1988 HTTP/1.1 200 OK  (application/json)
12.485976000 CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
12.750146000 OPS1    CSO1         HTTP    2005 HTTP/1.1 200 OK  (application/json)
12.869263000 CSO1    OPS1         HTTP     316 GET /v2.0/ports?device_id=9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
12.934783000 OPS1    CSO1         HTTP    1169 HTTP/1.1 200 OK  (application/json)
13.056624000 CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
13.311013000 OPS1    CSO1         HTTP    2005 HTTP/1.1 200 OK  (application/json)
13.436240000 CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
13.819025000 OPS1    CSO1         HTTP    2005 HTTP/1.1 200 OK  (application/json)
13.842042000 CSO1    OPS1         HTTP     341 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/servers/9c1986ff-9143-46a6-a7df-ec1273a898be HTTP/1.1
14.161808000 OPS1    CSO1         HTTP    2005 HTTP/1.1 200 OK  (application/json)
14.283071000 CSO1    OPS1         HTTP     340 GET /v2.1/e9102671a6004a209f8ccec14cadf2fd/images/b58e19f6-d43d-49ad-b6b5-80aa49b73d93 HTTP/1.1
14.546504000 MDSC1   MDSC1        HTTP     335 POST /restconf/ietf-actn-vn/config/actn/vn/vn-list/vn1/ HTTP/1.1  (application/json)
14.553830000 MDSC1   MDSC1        HTTP     579 POST /onos/v1/flows/of:0000000000000001?appId=1s1s HTTP/1.1  (application/json)
14.542550000 OPS1    CSO1         HTTP    1013 HTTP/1.1 200 OK  (application/json)
14.646810000 MDSC1   MDSC1        HTTP     227 HTTP/1.1 201 Created
14.649450000 MDSC1   MDSC1        HTTP      86 HTTP/1.0 200 OK  (application/json)
14.653668000 CSO1    CSO2         HTTP     335 POST /restconf/ietf-actn-vn/config/actn/vn/vn-list/vn1/ HTTP/1.1  (application/json)
14.875761000 CSO2    CSO1         HTTP     449 HTTP/1.0 200 OK  (application/json)
14.878862000 CSO1    CLIENT       HTTP     783 HTTP/1.0 200 OK  (application/json)
```

Fig. 10. Captured wireshark message exchange for provisioning service

References

[1] Cisco Global Cloud Index: Forecast and Methodology, 20152020, White Paper, 2016.
[2] X. Liu, et al. ”YANG Data Model for TE Topologies.”, draft-ietf-teas-yang-te-topo-06, IETF, Oct. 2016.
[3] T. Saad (ed), et al. A YANG Data Model for Traffic Engineering Tunnels and Interfaces, draft-ietf-teas-yang-te-05, IETF, Oct. 2016.
[4] D. Ceccarelli and Y. Lee (ed.), Framework for Abstraction and Control of Traffic Engineered Networks, IETF draft, October 2016.
[5] Mapping Cross Stratum Orchestration (CSO) to the SDN architecture, TR-528, ONF, 2016.
[6] R. Vilalta, A. Mayoral, R. Casellas, R. Martínez, R. Muñoz, Experimental Demonstration of Distributed Multi-tenant Cloud/Fog and Heterogeneous SDN/NFV Orchestration for 5G Services , in Proceedings of European Conference on Networks and Communications (EuCnC), June 27-30 2016, Athens (Greece).
[7] Y. Lee (ed.), A Yang Data Model for ACTN VN Operation, IETF draft, October 2016.
[8] A. Bierman et al., RESTCONF Protocol, IETF draft-ietf-netconf-restconf-18, 2016.
[9] A. Aguado, V. López, J. Marhuenda, O. González de Dios, and J. P. Fernández-Palacios, ABNO: A feasible SDN approach for multivendor IP and optical networks [invited], Journal of Optical Communications and Networking, vol. 7, num. 2, A356-A362, 2015.
[10] A. Mayoral, R. Muñoz, R. Vilalta, R. Casellas, R. Martínez, V. López, Need for a Transport API in 5G for Global Orchestration of Cloud and Networks Through a Virtualized Infrastructure Manager and Planner [Invited], IEEE/OSA Journal of Optical Communications and Networking, vol. 9, num. 1, A55-A62, 2017.
[11] R. Vilalta et al., Peer SDN Orchestration End-to-End Connectivity Service Provisioning Through Multiple Administrative Domains, ECOC16.
[12] OpenStack API. http://developer.openstack.org/api-guide/quick-start/
[13] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, SDN orchestration architectures and their integration with cloud computing applications, Optical Switching and Networking, 2016, Elsevier.