# DataPlane Broker: Open WAN control for multi-site service orchestration

Steven Simpson, Arsham Farshad, Paul McCherry, Abubakr Magzoub, William Fantom,
Charalampos Rotsos, Nicholas Race, David Hutchison
School of Computing and Communications, Lancaster University

***Abstract*—NFV-MANO has become the de-facto standard for network service orchestration in future programmable network infrastructures. Specifically, relevant standards define an architecture and a data model that allows an orchestration entity to deploy, dynamically configure and monitor virtual and physical network function across virtualized datacenters. Although the model offers extensive details for network functions management and host-level network configuration, end-to-end connectivity management beyond the datacenter remains limited, modeled by the WAN Infrastructure Manager (WIM) abstraction. Although many open-source service management frameworks offer NFV-MANO support, the community still lacks a WIM reference implementation. In this paper, we present the DataPlane Broker (DPB), the first open source WIM implementation for Ethernet networks. The system develops an extensive data model to seamlessly translate NFV-MANO connectivity requirement to network configuration for SDN-enabled networks and supports point-to-point and point-to-multipoint virtual links with strong QoS guarantees. DPB currently offers complete integration with the current stable OSM version (v5) and we demonstrate that it provides good scaling properties under high service provision requirements.**

## I. Introduction

Network Function Virtualization (NFV) is a recent networking paradigm promoting the adoption of cloud technologies in the deployment and management of network middleboxes. Specifically, the paradigm advocates the replacement of traditional hardware-accelerated black-box middlebox devices by software systems, running as virtual machines on virtualized hosts. In parallel, existing vendor-locked remote control protocols are replaced by a unified and open management interface. NFV technologies offer infrastructure managers the ability to compress operational costs, while offering greater flexibility, diversity and packet processing elasticity.

NFV management standardization is predominantly driven by the ETSI NFV SIG [8]. The group develops a wide portfolio of standards and reference implementations, including the NFV-MANO architecture and data models. The proposed architecture consists of three management layers: The Virtual Infrastructure Manager (VIM) controlling infrastructure physical resources; the NFV manager (NFVM) controlling and monitoring running NFV instances; and the NFV Orchestrator (NFVO) responsible for fulfilling network service requests from external users.

Although NFV-MANO was initially designed to support NFV management within a single datacenter, the development of new technological paradigms like the Mobile Edge Cloud [6] and cloud-RAN [3] have forced standards to consider requirements for cross-site service deployment. A key requirement to meet these requirements is introducing support for network forwarding and resource management in standards. This functionality was briefly discussed in the original NFV-MANO standards and abstracted by the WAN Infrastructure Manager (WIM) [11] entity. A WIM is a special type of VIM, managing exclusively network resources and offering connectivity between NFV datacenters. Specifically, a WIM is a shim layer mediating MANO virtual link requests to network controllers, in an effort to provide point-to-point and multi-point connectivity across NFV datacenters.

In order to increase standard adoption, the ETSI NFV SIG is developing the OSM project, a reference NFV-MANO implementation. Unfortunately, although, OSM offers extensive support for VNF management, through tight integration with the OpenStack and JuJu toolstacks, the development of the networking subsystem is limited. Specifically, WIM support was introduced in the fifth OSM release, and at the time of writing no open-source WIM implementation is available.

In this paper we present the Data Plane Broker (DPB), the first open-source WIM implementation, built for the MANO standards. Our work exhibits the following key contributions:

- An extensible data model for the deployment of connectivity services over SDN systems.
- A resource allocation and path computation algorithm with support for best-effort deployment of point-to-point and multi-point virtual links with strong bandwidth guarantees.
- An open-source WIM implementation and an OSM driver compatible with the latest stable version of OSM (release 5).

## II. Multi-site Service Deployment

NFV-MANO is currently the de-facto standard for service orchestration. The goal of the standards is to develop a set of data models to control and manage resources in a virtualized datacenter and facilitate the deployment, configuration, management and monitoring of service chains consisting of virtual and physical network functions. Initial standards aimed to support single VIM deployment scenarios, while the interface towards local programmable networks remained undefined. In parallel, a special type of VIM was defined, dubbed as the WIM, to control WAN or access networks using an SDN controller and to provide cross-site connectivity. An example WIM architecture is depicted in Figure 1, where a service is deployed across two datacenters, VIM1 and VIM2. A WIM controller manages the network infrastructure connecting the

two sites and can provision connectivity between the two endpoints. The resulting link will offer layer-2 connectivity to the two sites and enable a single broadcast domain between the two virtual links (VLD1).

In 2018, the ETSI NFV SIG released a white paper [12] discussing a series of use cases for multi-site connectivity and developing a discussion of required standards extensions. In parallel, a work group within the ETSI NFV SIG is currently developing a detailed information model for a WIM interfaces, inspired by the IETF ACTN standards [2].

OSM is the reference implementation of the NFV-MANO standards, offering an automation and coordination framework for the the provision and control of network services and functions. In order to provision connectivity between datacenters, OSM release 5 has introduced a non-standardized WIM plugin model. WIM plugins allow an OSM instance to embed in the service deployment the ability to control SDN switches and establish the necessary point to point and/or multipoint connections whilst providing the necessary bandwidths. The interface allows OSM to create, monitor and destroy point-to-point links.

Based on the goals of the WIM abstraction, Bravalheri *et al*. [1] have developed an integration of the ONF Transport API with the WIM model in OSM and have demonstrated the ability to control optical links two provide multi-site connectivity. Similarly, the 5GUK Exchange infrastructure [9] has presented a series of demos of a multi-site NFV testbed offering support for dynamic connectivity provision. Nonetheless, currently no VIM implementation are currently available open-source.

The specification of network orchestration interfaces is a goal for many SDOs. In addition to the aforementioned effort of the IETF ACTN WG, the Open Grid Form (OGF) has developed a Network Services Framework [10] (NSF) for slicing virtual networks from a physical topology, defining a Network Services Interface (NSI) for interaction between agents providing and requesting network services. This interface allows the definition of point-to-point uni- or bi-directional services, as well as anycast and multicast, including a single bandwidth parameter. OpenNSA, an implementation of NSI, is used in GÉANT Testbeds Service[1] (GTS), whose dynamically deployed network topologies (akin to NSDs) consist of network entities with only two endpoints. Finally, the GEANT's DynPaC framework [7] develops a dynamic interfaces for Bandwidth on demand services over the GÉANT infrastructure. The orchestration layer is implemented as part of the OpenDayLight SDN controller framework and uses stateful PCE [5] to control resource allocation in the underlying network infrastructure.

## III. Data Plane Broker

WAN and access networks vary significantly with respect to employed network technologies (*e.g*. optical, microwave, MPLS etc.) and topologies. DPB design aims to provide an extensible framework which abstracts the underlying infrastructure diversity. The architecture of a DPB orchestrator is presented in Figure 1. DPB follows a multi-layer design approach and consists of three key layers.

[1]https://gts.geant.net/

At the top of the system is the *aggregator* layer, which is responsible for service persistence, allocation of trunk labels, and end-to-end path computation (§III-C). The aggregator layer exposes a single northbound interface (NBI) to the service orchestrator for the management of logical networks across datacenters. In addition, the NBI of an aggregator can be used by a higher layer aggregator, enabling control delegation.

The *logical-switch* layer (§III-B) constitutes the middle layer of the DPB architecture and is responsible for persisting configuration and forwarding state for network devices, and abstracting technology-specific configuration details from the aggregator layer. The latter is enabled through the technology *adaptation* layer, which translates forwarding policy into technology-specific configuration. The communication between the different layers of DPB is facilitated by a hierarchical data model (§III-A).

### A. Data Model

In order to support the technology and topology diversity, DPB develops a hierarchical model to abstract network connectivity support between the different network management layers. The DPB NBI follows a "one big switch" abstraction; the underlying topology remains hidden from the orchestrator, which can request connectivity for any arbitrary group of network terminals, *i.e*. VIMs. The orchestrator can request from DPB to manage a *(logical) network*; a set of *network services*, *i.e*. layer-2 broadcast domains, between network terminals and with specific bandwidth guarantees. A network service is modeled as a set of ⟨circuit-id, ingress-bandwidth, egress-bandwidth⟩ tuples. A circuit effectively is a unique identifier describing the delivery of different network service to the same specific terminal. Figure 2 provides an example logical network with six terminals, and one service connecting three circuits (`site1-opst:435`, `site2-ofx:961`, `site3-ofx:2010`) and another service connecting two (`site1-opst:91`, `site1-ofx:961`).

The DPB aggregator layer models network services as a collection of terminals, networks and trunks, connecting different circuits. Similar to a logical network, aggregation layer networks have also terminals, which reflect either network-wide network endpoints or intermediate devices ports connecting to an internal physical link. DPB defines two network types. A *logical switch network* is the simplest network type, modeling a single switching fabric, *e.g*. an OpenFlow datapath. An *aggregator (network)* is a complex network type abstracting the control and connectivity of different *inferior* networks, which can be either switch networks or smaller aggregator networks. Fig. 3 shows an aggregator with two terminals (mapping to two inferior networks' terminals) and one trunk (connecting the two inferiors' other terminals). A service across the aggregator between circuits `site1-opst:91` and `site2-opst:961` is realized as a service on `site1` between `opst:91` and `site2:73`, a service on `site2` between `site1:73` and `opst:961`, and allocation of label 73 on the trunk.

A trunk models a slice of a physical link connecting two network terminals of inferior networks. A trunk is a set of virtual links with bandwidth allocated in each direction to each terminal, as well as having unallocated bandwidth and labels.

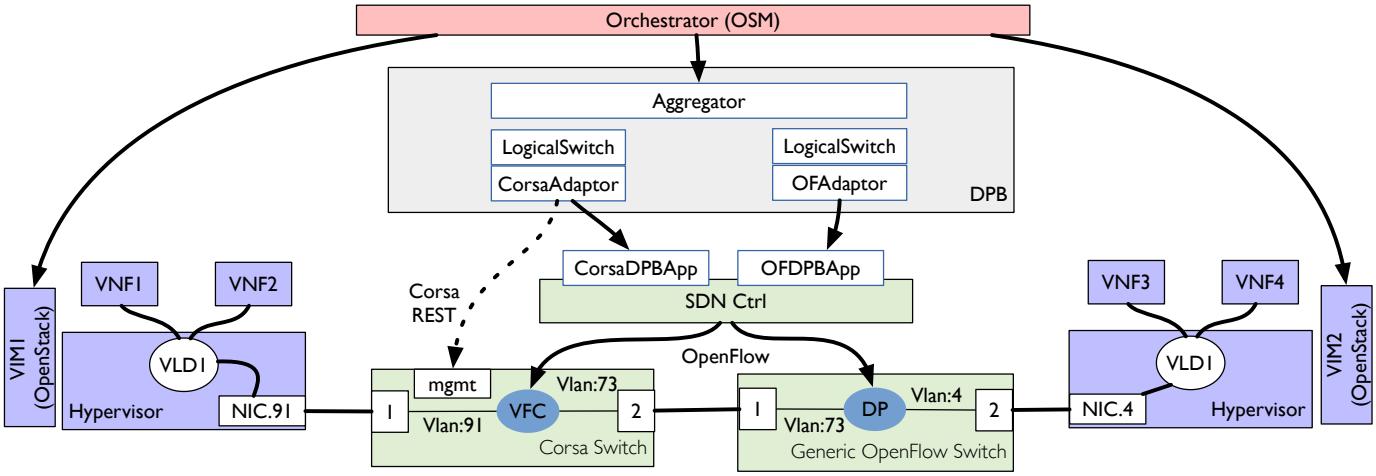



**Fig. 1:** DPB offers an network control NBI to Service Orchestrators. It multilayer architecture offers the ability to abstract control capabilities under TODO INCOMPLETE SENTENCE

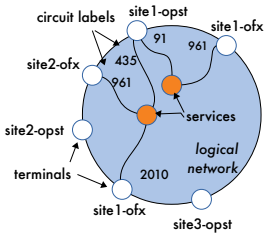



**Fig. 2:** A logical network with six terminals and two services

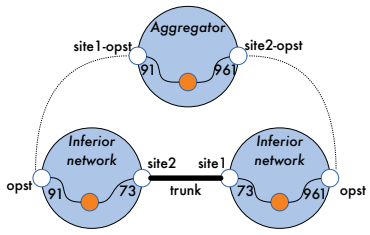



**Fig. 3:** n aggregator operating two inferior networks

Management and control of each trunk is the responsibility of exactly one aggregator.

### *B. Logical Switch and Adaptor Layer*

The main functionalities of the DPB logical switch layer is switch configuration and service mapping persistence and service deployment using an appropriate switch adaptor. The layer offers two primary management interface: `Switch` andNetworkControl. The `Switch` management interface allows runtime control of mappings between terminal and fabric interface names. `NetworkControl` on the other hand can be used to persist services configurations and subsequent invocations in the adaptor layer.

The DPB adaptor layer is responsible to abstract switch technology differences during service configuration and to optimize service deployment. The current DPB release offers for two switch adaptors: generic OpenFlow and Corsa.

*a) Generic OpenFlow Adaptor:* DPB offers a generic 1.3 OpenFlow switch adaptor, that uses OpenFlow metering capabilities, to deliver circuit connectivity. The OpenFlow Adaptor relies on custom Ryu App to translate DPB switch configuration into appropriate OpenFlow commands.

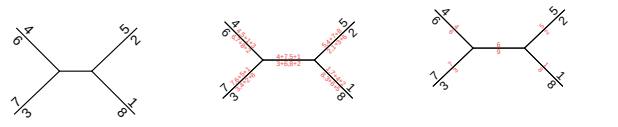
The REST API of the app accepts circuit descriptions which specify an OpenFlow port, an optional VLAN tag, an optional inner VLAN tag, and optional ingress/egress rates. Basic OpenFlow flows are used to forward traffic between circuits, while OpenFlow meters are used to enforce resource control. The controller uses three OpenFlow tables (T0, T1, T2) and the group table to implement E-Line behavior for groups of two circuits, and learning-switch behavior for larger groups. Resource control requires up to two meter entries per circuit, so two meter identifiers are similarly allocated.

*Corsa Adaptor:* The Corsa DP2000 [4] series switches are deeply-programmable SDN switch platforms, built around an FPGA architecture. The vendor offers a wide range of FPGA bitfiles and software drivers, that allow network managers to deploy specialized switching logic optimized for specific network setting (*e.g.* hardware-accelerate Quagga routers).

The DPB codebase offers a Corsa switch adaptor, compatible with the OpenFlow switch firmware. The adaptor exploits the virtualization capabilities of the platform, to a improve QoS and tunnel management. Specifically, the adaptor uses the management interface of the switch to create virtual ports and switches for each service, while network resource policies are enforced using switch and port configuration mechanisms.

The Corsa adaptor uses the management REST API to allocate a dedicates VFC, which is configured to connect to a custom Corsa Ryu Application. When a new service is created, its circuits' terminals are mapped to physical ports and its circuits' labels identify specific VLANs on those ports. Spare virtual ports of the VFC are allocated to the service, and attached to the identified tagged ports. In parallel, a shaping parameter is set on each virtual port based on the corresponding circuit's egress rate, while ingress rate sets the attachment's metering. VLAN tags associated with a virtual port, as well as metering and shaping configurations remain invisible to the VFC's OpenFlow rules and controllers.

The applied packet forwarding logic, depend on service port cardinality. If only two ports are configured for a service, then the adaptor uses static OpenFlow rules to forward traffic between the two ports. If more than two ports are allocated for a service, then a learning switch control application is used on a per-service basis.



**(a)** Sums of terminal ingress and egress bandwidths

**(b)** The tree with contributions summed from each direction

**(c)** Computed bandwidth requirements in each direction of each edge

**Fig. 4:** A tree connecting four vertices with distinct ingress and egress requirements

### C. Aggregator Layer

The Aggregator layer is primarily responsible to interface with external control entities (*e.g.* orchestrator), as well as logical switches and define the required configurations to deliver connectivity services. This layer exposes two interfaces: an `Aggregator` and a `NetworkControl` interface. The Aggregator interface is a configuration interface and allowing network managers to define network topology and terminal name to network mappings. The `NetworkControl` interface is the same interface as in the logical layer and allows an aggregator to present itself as a logical switch to another aggregator and seamlessly enable the ability to form control hierarchies.

Path computation in the Aggregator uses a *Bandwidth-aware Spanning-Tree Algorithm*. The algorithm is built around a spanning tree algorithm and uses available and required bandwidth values as a constraint in tree computation. It can compute point-to-point as well multi-point connectivity requirements with strong bandwidth guarantees.

The algorithm assumes that the aggregator, through network manager configuration and by monitoring the inferior aggregator and logical switch networks, has complete knowledge of the available bandwidth to connect terminal pairs in an inferior network, as well as connectivity characteristics between terminals of different inferior networks (*i.e.* trunk links). The algorithm generates a set of tuples describing trunk link allocations and terminal connections in an inferior network.

The algorithm initially generates a weighted connectivity graph, with each edge derived either from a trunk (using its available capacity as the weight) or a terminal connection in an inferior network (with a zero weight). A partial spanning tree is generated from the connectivity graph by selecting a path between two random terminals. The rest of the terminals are connected by adding the path to the nearest vertex of the tree. If no path is found for at least a terminal, the service request is rejected.

The bandwidth requirements of each trunk edge of the tree are derived in each direction by summing the ingress bandwidths of the leaves reachable in one direction, and summing the egress bandwidths of the leaves reachable in the opposite direction, and choosing the minimum. A bandwidth requirement is thus yielded in each direction for that edge. Figure 4a shows a tree connecting for terminal vertices, and the sums of the ingress and egress bandwidth requirements for circuits at each terminal. Figure 4b shows a pair of sums applied in each direction on each edge. The minimum of each pair is chosen as the bandwidth requirement of the edge in the pair's direction (Figure 4c).

Each trunk edge's bandwidth requirements are checked against the trunk's available capacity. If a trunk cannot meet the computed requirements, the tree is rejected; one trunk edge is eliminated, and the algorithm attempts to build and validate a new tree.

If all trunks meet bandwidth requirements, then the tree is used to generate the algorithm results. A spare label is allocated for each trunk whose edge contributes to the tree, and this label is combined with the trunk's requirements to produce the resultant trunk allocations. The terminals of each selected trunk have the trunk's allocated label attached to identify a pair of circuits. Each circuit is annotated with an ingress rate, the trunk's bandwidth requirement flowing to that circuit's terminal, and with an egress rate, the trunk's bandwidth flowing from it. Each input terminal is augmented with its corresponding label to identify a circuit, which is then similarly annotated with ingress and egress rates of the input terminal. The annotated circuits derived from selected trunks and input terminals are grouped by subnetwork, and each group forms a segment descriptor for that subnetwork.

### D. Implementation

The DPB code is currently released under a 3-clause BSD license [2]. It consists of ~36k lines of Java, and ~2.5k of Python (Ryu controller applications for DPB adaptors). State is persisted in SQLite databases.

DPB exposes a north-bound interface supporting the creation and management of point-to-point and multi-point connectivity services with bandwidth control, and is accessible via a REST API, as well as via SSH. In parallel, users can experiment with API using a simple command-line client. The DPB Adaptor layer provides extensive modularity and developers can readily implement custom adaptors for different technologies. Finally, the DPB codebase offers a fully functional WIM driver, compatible with the WIM driver model of OSM release 5.

## IV. Evaluation

In this section we evaluate the scalability and performance of DPB. Our evaluation focuses on two aspects: the scalability of the path computation algorithm in the aggregator (§ IV-A) and the impact of DPB on the performance of a service orchestrator (§ IV-B). In our evaluation, we use an experimental testbed with three OpenStack clusters representing three distinct network sites. OpenStack clusters are connected via a CORSA switch using virtual switches, while the switch metering capabilities are used to emulate network link properties. Table I presents the technical characteristics of our testbed.

### A. Scalability Analysis

To test the scalability of the spanning-tree algorithm, we randomly generated four scale-free undirected graphs of size 54, 72, 96 and 128, using the Barabasi-Albert model. For each graph vertex, we create an emulated logical switch and we

[2] code:http://scc-forge.lancaster.ac.uk/svn-repos/initiate/dataplanebroker/

**TABLE I:** Experiment Set-ups

| System | Specifications |
|---|---|
| Computing Infra. | Dell Servers equipped with Intel Xeon E5-2630 (32 cores) CPU and 128 GB Memory |
| Networking Infra. | CORSA DP2100 equipped with 10 Gbit/s SFPs |
| Service Orchestrator | ETSI OSM R5.0 with DPB plugins |
| VIM | OpenStack Queens/Rocky releases |
| Network Controller | Ryu OF controller |

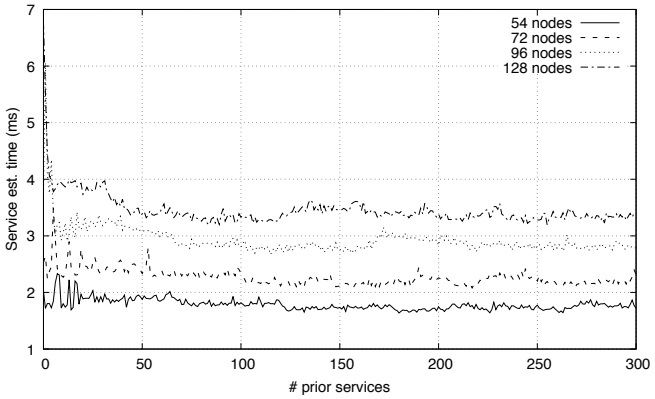



**Fig. 5:** Establishment latency given prior services

attached a `host` terminal to it. Logical switches are configured to connect to a single emulated aggregator. For each edge, we create a terminal node on each switch and connect them using an emulated trunk. Each trunk link has a capacity of $n$Gbit/s, where $n$ is the minimum of the two connecting nodes' degrees. For each experimental run, we generated service requests of 10 Mbit/s circuits between a randomly selected set of 2 to 4 nodes and monitor the computational time, until network resources are exhausted and the aggregator cannot fulfill new requests.

Figure 5 and Figure 6 reports the mean time to process a service request (100 runs), when a number of service and circuits respectively are already deployed, (existing service number varies between 1 and 300). From our results, we highlight that the path computation algorithm used by the aggregator has a bound execution time, which depends to the size of the network. Furthermore, the major latency component is the linear polling of individual switches by the aggregator, in order to obtain accurate state information of inferior networks. While this latency could contribute significantly in practice, various strategies can be adopted to minimize its impact. Specifically, network models can be cached (*i.e.* they do not need to be recomputed), logical switches can be co-located with the aggregator to minimize propagation latencies, while aggregator hierarchies can amortize propagation latencies in high bandwidth/delay networks. In addition, we note in the obtained results that the number of deployed services has an impact on path computation complexity but only for lightly used network infrastructures, where the computational latency is doubled, but these latencies become significantly lower for highly utilized networks. As trunk resources are allocated to new service, trunk capacities are reduced and the algorithm excludes trunks with insufficient capacity, thus operating on a smaller graph.

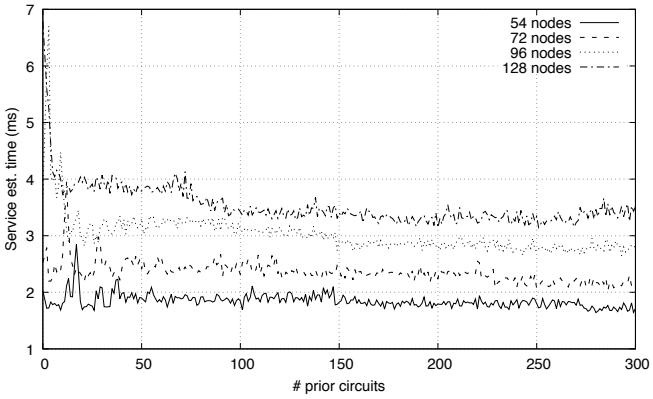



**Fig. 6:** Establishment latency given prior circuits

## B. Performance Analysis

To evaluate the impact of DPB on multi-site service orchestration, we used the WIM driver for OSM to orchestrate a multi-site CDN network service. The CDN service is depicted in Figure 7 and consists of a firewall (FW) VNF (iptables), an L4-Load Balancer (LB) VNF (OpenVSwitch) and multiple content cache VNFs (Apache). All VNFs are build using an Ubuntu 14.04 cloud image and rely on distribution packages to deliver their functionality. OSM deployment is achieved through a set of yaml files containing the Network Service Descriptor (NSD) and the VNF Descriptors (VNFDs) following the ETSI MANO data model. In addition, we used the MON and POL modules of OSM to deliver run-time scale-in policy based on observer CPU utilization.

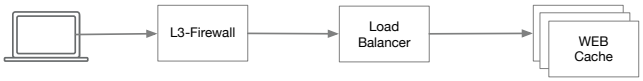



**Fig. 7:** A CDN service comprises of a service chain of a firewall, load balancer and cache

The CDN service is deployed across three sites and the deployment topology is depicted in Figure 8. The firewall was deployed on an edge cloud adjacent to the client and was configured to filter out non-cache traffic. The Load-Balancer was statically deployed on VIM Site-1 and distributed web requests on a dynamic set of VNF caches. The first VNF cache was distributed on the same VIM as the Load-Balancer, while subsequent replicas were deployed on Site-2 VIM.

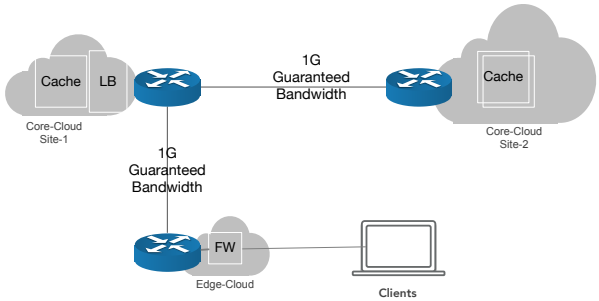



**Fig. 8:** CDN service orchestration across three sites. By increasing the load more caches are deployed to the core-cloud2.

We measured the impact of DPB on the initial deployment process. Specifically, we deployed the CDN VNFD on our platform multiple times and using information from the Resource Orchestrator (RO) logs, we measured the deployment time of different components of the services. For this experiment we define two latency metrics: *Virtual Infrastructure set-up*, which describes the time required to boot and configure all service VNFs, and the *WAN connectivity set-up*, which describes the time required to deploy the connectivity services using DPB. Table II reports the mean and standard deviation of the aforementioned metrics for 10 experiment runs. We highlight that OSM requires 10 seconds to deploy the required connectivity service using the DPB driver. From an overall service deployment perspective, this latency is minimal and increases the overall deployment time by only 8%, since the service requires on average 128 seconds to be deployed in our local testbed. It is worth to highlight, that in a real world scenario the propagation delay would be significantly higher than in our experiment and the estimated WAN set-up time would be slightly increased, but we do not expect to have an increase more than 10% of the overall time, since this latency inflation would be similar for all service management components.

**TABLE II:** Latency to deploy and configure the compute and network resources in our multi-site CDN service. DPB incurs less than 10% increase in the overall deployment time.

| Metrics (sec) | Mean | Stddev |
|---|---|---|
| Virtual Infra. set-up | 118 | 7.52E-5 |
| WAN set-up | 10 | 7.40E-5 |
| Total Service Bootstrap | 128 | 1.33E-4 |

## V. Conclusion and Future Work

Support for multi-site service deployment is currently a major functional requirement for NFV standardization, which is hinder by the lack of support for network resource orchestration. Relevant standards currently offset the responsibility for inter-site connectivity to a special VIM type capable to control WAN networks, called WIM. Nonetheless, there is no standard yet released regarding a WIM data model or interface. Nonetheless, OSM, the reference NFV-MANO implementation, has released a *de-facto* standard for a WIM lifecycle.

This paper presented DPB, the first open-source WIM implementation for SDN networks. DPB enable hierarchical control of programmable WANs, defines a detailed WAN control data model and provides out-of-the-box support for multi-site OSM service orchestration. Furthermore, using a series of emulation we demonstrated that DPB can support a large networks and increasing numbers of service request with small computational overhead, while establish inter-site connectivity in OSM using DPB increases service delivery latency by 8%.

We believe that the release of the DPB framework will offer the opportunity for further development of multi-site support in service orchestration standardization. Firstly, during our experimentation we identified a number of shortcoming in existing data models with respect to multi-site support. For example, IP address allocation in OSM cannot function for multi-site service delivery since the model forces the same IP ranges to be used in all VLDs. Secondly, our implementation uses a variation of the spanning tree algorithm to compute service paths. Nonetheless, the multi-layer design of the data model in DPB offer the opportunity to explore multiple different optimization algorithms for path estimation and effectively a highly realistic framework to evaluate different function placement strategies.

## References


[1] A. Bravalheri, A. S. Muqaddas, N. Uniyal, R. Casellas, R. Nejabati, and D. Simeonidou. Vnf chaining across multi-pops in osm using transport api. In *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3, March 2019.

[2] D. Ceccarelli and Y. Lee. Framework for abstraction and control of te networks (actn). RFC 8453, RFC Editor, August 2018.

[3] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann. Cloud ran for mobile networksâĂŤa technology overview. *IEEE Communications Surveys Tutorials*, 17(1):405–426, Firstquarter 2015.

[4] Product Overview. https://www.corsa.com/products/.

[5] E. Crabbe, I. Minei, J. Medved, and R. Varga. Path computation element communication protocol (pcep) extensions for stateful pce. RFC 8231, RFC Editor, September 2017.

[6] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang. Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Systems Journal*, 12(3):2495–2508, Sep. 2018.

[7] A. Mendiola, J. Astorga, J. Ortiz, J. Vuleta-RadoiÄ■iÄĞ, A. Juszczyk, K. Stamos, E. Jacob, and M. Higuero. Towards an sdn-based bandwidth on demand service for the european research community. In *2017 International Conference on Networked Systems (NetSys)*, pages 1–6, March 2017.

[8] ETSI NFV. ETSI NFV SIG. https://www.etsi.org/technologies/nfv.

[9] B. Nogales, I. Vidal, D. R. Lopez, J. Rodriguez, J. Garcia-Reinoso, and A. Azcorra. Design and deployment of an open management and orchestration platform for multi-site nfv experimentation. *IEEE Communications Magazine*, 57(1):20–27, January 2019.

[10] Guy Roberts, Tomohiro Kudoh, Inder Monga, Jerry Sobieski, Chin Guok, and John MacAuley. Network Services Framework v2.0. Technical report, Open Grid Forum, 2014.

[11] ETSI SGI. Network Functions Virtualisation (NFV); Infrastructure; Network Domain . Technical report, 2014.

[12] ETSI SGI. Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on Management and Connectivity for Multi-Site Services . Technical report, 2018.