This is the post peer-review accepted manuscript of:
W. Cerroni *et al.*, "Intent-based management and orchestration of heterogeneous openflow/IoT SDN domains," *2017 IEEE Conference on Network Softwarization (NetSoft)*, Bologna, 2017, pp. 1-9.

The published version is available online at:
https://doi.org/10.1109/NETSOFT.2017.8004109

Intent-Based Management and Orchestration of Heterogeneous OpenFlow/IoT SDN Domains

Walter Cerroni, Chiara Buratti, Simone Cerboni, Gianluca Davoli,
Chiara Contoli, Francesco Foresta, Franco Callegati, Roberto Verdone
DEI - University of Bologna, Italy
Email: {walter.cerroni, c.buratti, gianluca.davoli, chiara.contoli, francesco.foresta, franco.callegati, roberto.verdone}@unibo.it

***Abstract*—One of the main challenges in delivering end-to-end service chains across multiple Software Defined Networking (SDN) and Network Function Virtualization (NFV) domains is to achieve unified management and orchestration functions. A very critical aspect is the definition of an open, vendor-agnostic, and interoperable northbound interface (NBI) that should be as abstracted as possible from domain-specific data and control plane technologies. In this paper we propose a reference architecture and an intent-based NBI for end-to-end service orchestration across multiple technological domains. In particular, we consider the use case of an Internet of Things (IoT) infrastructure deployment and the corresponding cloud-based data collection, processing, and publishing services with quality differentiation. We also report the experimental validation of the proposed architecture over a heterogeneous OpenFlow/IoT SDN test bed.**

I. Introduction

Following the recent innovations brought about by cloud computing and resource virtualization, current advances in communication infrastructures show an unprecedented central role of software-based solutions [1], [2]. In particular, the Network Function Virtualization (NFV) paradigm fosters flexible and cost-effective service provisioning by deploying network functions as pieces of software running on vendor-independent hardware platforms, bringing the benefits of cloud computing to network infrastructure management [3]. At the same time, Software Defined Networking (SDN) decouples software-based network control and management planes from the hardware-based forwarding plane, turning traditional vendor locked-in infrastructures into communication platforms that are fully programmable via a standardized, open, southbound interface (SBI) [4]. The joint adoption of SDN and NFV provides enhanced flexibility to service deployment: the so-called *service chain*, i.e., the sequence of network functions to be applied to data flows exchanged by a given customer (or set of customers), can be dynamically controlled and modified over a relatively small time scale, significantly reducing the management burden compared to traditional networks [5].

End-to-end services provided to customers are typically delivered across different network administrative and/or technological domains. Therefore, guaranteeing certain levels of service has always been a challenging task in multi-domain environments. This is even more complex in case of services spanning multiple SDN/NFV domains, because of the more advanced control features provided by these new paradigms [6]–[8]. A very critical aspect to achieve unified management and orchestration of end-to-end services across multiple domains is the definition of an open, vendor-agnostic, and interoperable northbound interface (NBI), through which applications are allowed to control the underlying heterogeneous NFV and SDN infrastructures and take advantage of dynamic service chaining. Although a standard NBI definition is not available yet, a commonly accepted approach is to adopt a so-called *intent-based* interface that allows to declare high-level service policies rather than specify detailed networking mechanisms [9].

In this paper we present a reference architecture and define a related intent-based NBI for end-to-end service management and orchestration across multiple technological domains. In particular, we consider the use case of an Internet of Things (IoT) infrastructure deployment and the corresponding cloud-based data collection, processing, and publishing services with quality of service (QoS) differentiation.

In line with the multiple software-defined infrastructure scenarios foreseen by the 5G initiative, the IoT domain considered here is inspired by existing work aimed at extending the SDN concepts to wireless sensor networks (WSNs) [10]–[12]. The above papers represent important contributions to the SDN literature as they provide convincing motivations for the extension of the SDN paradigm to IoT domains. [13] presents the idea of exploiting the OpenFlow technology to address the reliability in WSNs, while [14] reports experimental results showing advantages in using SDN approaches in WSNs.

In this paper, we extend the work in [12] and [15], where a solution for separating the data and the control plane of an IoT network is proposed, in order to virtualize the IoT domain, allowing the IoT controller to program the network with the aim of guarantee a specific QoS requested by the consumer. Moreover, the framework has been integrated with an OpenFlow-based SDN infrastructure. In particular, a heterogeneous test bed consisting of IoT, OpenFlow and cloud domains has been set up and experimental results are reported in this paper to validate the proposed architecture.

The remainder of the paper is organized as follows. We propose our reference architecture and define the intent-based NBI in Sections II and III, respectively. Then we provide specific examples and technical details related to IoT and OpenFlow/cloud domains in Sections IV and V. We report the experimental validation in Section VI, and we finally conclude

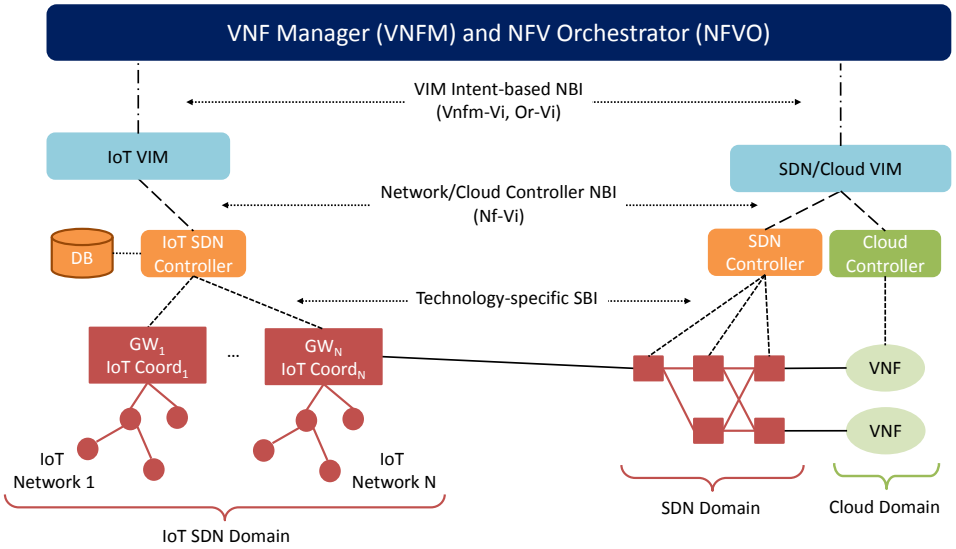

Fig. 1. Reference multi-domain SDN/NFV architecture, specialized for the use case of IoT data collection and related cloud-based consumption.

the paper in Section VII.

II. Reference Network Architecture

The reference multi-domain SDN/NFV architecture considered in this paper is shown in Fig. 1. Although our approach to intent-based orchestration could be generalized to any SDN/NFV technology domain, the reference architecture is specialized for the use case considered here, where data collected from sensor and actuator devices of a software-defined IoT domain are dispatched across a wired SDN infrastructure to reach a set of suitable consumers, implemented by means of virtual network functions (VNFs) and deployed within a cloud computing domain.

Considering the purpose of our study and the nature of the orchestration features we are interested in, our reference architecture is inspired by the ETSI NFV specifications, with particular reference to the Management and Orchestration (MANO) framework [16], although our approach considers an end-to-end service perspective. The rationale behind this choice is that, on one hand, the proposed architecture has the advantage to be consistent with the most relevant NFV standard initiative to date; on the other hand, the architecture itself can be seamlessly extended to include any further SDN/NFV domain and technology as part of the underlying virtualized infrastructure.

Each SDN/NFV domain in Fig. 1 consists of a technology-specific infrastructure, including:

- data plane components, such as IoT nodes and gateways, SDN switches, virtual machines running in cloud computing nodes, physical and virtual interconnecting links; these components provide the network, compute, and storage resources to be orchestrated;
- control plane components, such as SDN and cloud controllers with related data stores and interfaces; these components are responsible for proper VNF deployment and traffic steering across VNFs and domains;
- management plane components, such as Virtualized Infrastructure Managers (VIMs) specialized for managing resources in the IoT-based SDN infrastructure, the wired SDN infrastructure, and the cloud infrastructure; based on the available implementations, some of these components could be in charge of multiple domains [17], as in the case of the SDN/cloud VIM in Fig. 1.

The overarching VNF Manager (VNFM) and NFV Orchestrator (NFVO) components are responsible for programming the underlying VIMs and infrastructure controllers in order to implement and maintain the required service chains in a consistent and effective way, for both intra- and inter-domain scenarios. While technology- and domain-specific northbound (NBI) and southbound interfaces (SBI) are used inside each domain to efficiently control and manage the relevant components, the design of the overarching VNFM and NFVO should be as technology-agnostic as possible, so that a service chain to be deployed can be specified by a customer using a high-level, intent-based description of the service itself. This would also allow the proposed architecture to be more general and capable of being extended to different SDN technologies and domains.

In order to achieve such generality in the high-level management and orchestration components, we argue that *the act of decoupling service abstractions from the underlying technology-specific resources should be performed mainly by the VIMs*. Therefore, we extend the concept of interactions based on intents to the NBI offered by the VIMs, which should be defined as an open and abstracted interface, independent of the specific technology used in the underlying domains. This approach could also allow different administrative domains to expose only service abstractions without disclosing sensitive details related to the underlying infrastructures.

III. VIM Northbound Interface

In general, the definition of an open, vendor-agnostic, and interoperable interface will foster improved and standardized procedures for customer service specification to the underlying multi-domain NFV and SDN platforms. In particular, the powerful abstraction level offered by an intent-based NBI allows to specify policies rather than mechanisms, by taking advantage of formalisms that are close to the customer's natural language [9]. Therefore, in our architecture we assume that some kind of intent-based interface is offered to the customer by the overarching VNFM and NFVO components.

When a given service request is received, the high-level management and orchestration functions must convert that request into a set of suitable service chains and pass them to the relevant VIMs in charge of the underlying infrastructures and domains involved in the service composition. Then each VIM must coordinate the respective controllers in order to:

- verify availability and location in the cloud infrastructure of the VNFs required to compose the specified service, instantiating new ones if needed;
- program traffic steering rules in the network infrastructure to deploy a suitable network forwarding path.

The NBI exposed by the VIMs should allow an abstracted yet flexible definition of the service chain, without knowledge of the technology-specific details such as devices, ports, addresses, etc. This means that a request sent to the VIMs should specify not only the sequence, but also the nature of the different VNFs to be traversed, which is strictly related to the service component they implement, as well as other peculiar characteristics of the service itself, such as quality of service (QoS) metrics and thresholds. In particular, the NBI should allow an abstracted representation of the QoS features for the requested service and the topological characteristics of each VNF to be applied in the service chain.

A possible definition of the VIM NBI is presented here, considering the following service and function abstractions.

- A *QoS feature* is defined in qualitative terms relevant to the specified service, e.g. guaranteed bit rate or limited delay.
- A *QoS threshold* can be specified for the metric of interest, e.g. a minimum bit rate or a maximum delay value.
- A VNF can be *terminating* or *forwarding* a given traffic flow. For instance, a deep packet inspection (DPI) function usually terminates a mirrored copy of a given flow, whereas a network address translator (NAT) forwards incoming flows.
- A forwarding VNF can be *port-symmetric* or *port-asymmetric*, depending on whether or not it can be traversed by a given traffic flow regardless of which port is used as input or output. For instance, a NAT is port-asymmetric, because it must receive inbound and outbound traffic from a port connected to a public and private network, respectively. A basic IP routing function can be considered port-symmetric, as it forwards packets based on the destination address.
- A VNF can be *path-symmetric* or *path-asymmetric*, depending on whether or not it must be traversed by a given flow in both upstream and downstream directions. For instance, an intrusion detection system (IDS) is typically path-symmetric, because it needs to analyze packets in both directions of a given flow. A traffic shaper can be considered path-asymmetric if it must limit only outbound traffic.

In order to implement the aforementioned abstractions, we define a service function chaining template adopting the well-known JSON format. This template should be coupled with other deployment templates defined by the ETSI MANO specifications in order to complete service provisioning. However, in this work we focus only on the service function chaining aspects of the NBI. A service chain is therefore defined as follows:

```
{
  "src": "node_value",
  "dst": "node_value",
  "qos": "qos_type",
  "qos-thr": "qos_value",
  "vnfList": [vnf],
  "dupList": [dup]
}
```

where: `src` and `dst` represent the endpoint nodes of the service chain, either global or limited to a given VIM domain; `node_value` is a text string that contains a high-level unique identifier of a node known to both orchestrator and VIMs, e.g. by means of some form of mapping mechanism as defined in [9]; `qos` represents the QoS feature to be provided with the service chain; `qos_type` is a text string that contains a high-level unique identifier of a QoS metric known to both orchestrator and VIM; `qos-thr` represents the QoS threshold to be applied to the specified metric; `qos_value` is the actual value assigned to the threshold; `vnfList` is the ordered list of VNFs to be traversed according to the specified service; `dupList` is the list of VNFs towards which the traffic flow must be duplicated.

Each VNF is described in terms of its topological abstractions with the following template:

```
vnf ::= {
  "name": "node_value",
  "terminal": "bool_value",
  "port_sym": "bool_value",
  "path_sym": "bool_value"
} | ϵ
```

where `bool_value` is a text string representing either a Boolean or a null value, and the ϵ symbol indicates the possibility that `vnf` is an empty element. Considering that some network functions (e.g., DPI, IDS) require traffic flows to be mirrored, the (possibly empty) list of VNFs towards which the traffic flow must be duplicated is specified with the following template:

```
dup ::= {"name": "node_value"} | ϵ
```

The NBI offered by VIMs can be implemented through the mechanisms of a REST API, and should provide the following methods:

- define a new service chain;
- update an existing service chain;
- delete an existing service chain.

These actions are basically in line with the operations foreseen by the ETSI MANO specifications with reference to the interface between NFVO and VIM. It is worth highlighting that the NBI description given above is indeed based on the concept of intent. QoS metric, VNFs and service chains are specified in a high-level, policy-oriented format without any knowledge of the technology-specific details. A non-intent-based description of a service chain, e.g. using the OpenFlow expressiveness to steer traffic flows and compose the network forwarding path, would require the customer to specify multiple flow rules in each forwarding device for each traffic direction, involving technology-dependent details such as IP and MAC addresses, device identifiers and port numbers.

The NBI defined above is used in the next sections to specify an IoT data gathering service crossing two different SDN domains and an NFV chain, as per the architecture in Fig. 1. For the use case considered here, the high-level QoS features offered by the SDN/NFV platform include "low latency" and "high reliability" services, with the possibility to specify a threshold for the relevant metric. Although the above intent-based NBI definition is common to all VIMs considered in our use case, the orchestrator must specify different content for each VIM depending on the specific resources to be programmed and the specific segment of the service chain to be deployed in each domain.

IV. IoT SDN Domain

The IoT SDN domain included in the architecture of Fig. 1 is composed of: i) a VIM able to manage components and resources in the IoT domain; ii) an *IoT SDN controller* (IoTC), implementing the software-defined control plane of the IoT domain; iii) a set of IoT networks, where different devices send the measured data via multi-hop paths to a coordinator node that forwards them to the final consumer. Since the different IoT networks will possibly use different technologies (e.g., Zigbee, LoraWAN, 6LowPAN, etc.), each IoT coordinator will be connected to a specific gateway (GW) in charge of forwarding data outside the IoT domain.

When a service request is received from the high-level management and orchestration functions, the IoT VIM gets access to the IoTC. As shown in the figure, one of the components of the controller is a database, which stores information about devices of the different networks, such as how to reach them (i.e., the IP address of the corresponding GW), the service provided, and the related QoS feature that could be guaranteed. The VIM tries to map the incoming request with the resource knowledge available in the database, in order to select the proper IoT device(s) to forward the request to. According to the decision taken, the IoTC will: i) program the selected IoT network(s) to make sure that the requested QoS would be guaranteed; and ii) forward the request to the identified GW(s). More details about the different components are provided in the rest of this section.

A. The IoT VIM and database

The VIM is capable of handling requests containing either the particular IoT device to be queried, or a high-level description of the service requested by the customer, together with some other possible specification related to the QoS (in terms of reliability or maximum latency). Let us consider the case of a customer that wishes to periodically collect temperature values in a given room and monitor them by means of a processing/publishing service called `ServP` running as a virtual function in the cloud domain. Assume that the customer is interested in having a complete record of the measured temperature request, thus requiring a high-reliable service. Then the intent-based request sent to the IoT VIM, expressed according to the JSON format specified in Section III, could be as follows:

```
{
  "src": "ServP",
  "dst": "Temperature Room X",
  "qos": "SR",
  "qos-thr": "90%",
  "vnfList": "null",
  "dupList": "null"
}
```

In the IoT domain, following the typical IoT device query approach, `src` represents the source of the query, that is the final consumer of the data to be collected. In our example, this is the processing/publishing service in the cloud. `dst` represents the final endpoint of the query, that could be one or multiple IoT devices. This text string may contain i) a unique identifier of a specific IoT device, or ii) a high-level intent-based description of the requested service. The second option is used in our example above. `qos` represents the requested QoS feature either in terms of latency, expressed as data plane round-trip time (see below), or reliability, that is the probability of successfully receiving the data from that device. In our IoT VIM implementation `qos` may assume the following values: real time (`RT`), non real time (`NRT`), strictly reliable (`SR`), and loosely reliable (`LR`). If needed, the user may also provide `qos-thr`, representing either the maximum tolerable latency or the minimum requested throughput/reliability. In the example above, `SR` with a 90% threshold is requested. Finally, `vnfList` and `dupList` are not specified in the example because we assume that the orchestrator opted for VNFs located in the cloud domain.

At this point the VIM checks in the database if the destination the user is looking for is present or not and if the requested QoS (if any) could be satisfied. In particular, the database contains an entry per IoT device and each entry includes:

- the unique MAC address (e.g., the IEEE 802.15.4 64-bit address);
- the corresponding network address (i.e., the short address used in IEEE 802.15.4 at 16-bit);

TABLE I
EXAMPLE OF QOS VALUES STORED INTO THE IOT DATABASE.

Node ID	Service	$\mathbf{RTT}_1$	$\mathbf{R}_1$	$\mathbf{RTT}_2$	$\mathbf{R}_2$	$\mathbf{RTT}_3$	$\mathbf{R}_3$
1	Smoke Detector Room X	12 ms	90%	24 ms	95%	36 ms	99%
2	Smoke Detector Room X	null	null	23 ms	96%	38 ms	98%
3	Light Room X	null	null	null	null	34 ms	98%

- the ID of the IoT network the device belongs to;
- the service provided by the device (e.g., temperature sensor, light sensor, etc.);
- the value and timestamp of the last measurement gathered from the device;
- the corresponding QoS in terms of latency and reliability: these values are computed by averaging among different measurements taken over time.

When the IoTC receives a new measurement from a device, the data is stored in the database, together with the instant in which it was received. Once a new request for the same device arrives the VIM checks the timestamp and decides if the data should be updated or not (if not the value is immediately returned). With reference to the QoS, it is important to underline that in case the same device could reach the IoT coordinator via different paths (e.g., having different number of hops), the corresponding QoS values are stored in the database. A simplified example is reported in Table I, where we are considering a room having two carbon monoxide sensors detecting the presence of smoke (devices 1 and 2) and a light sensor (device 3). Device 1 can reach the coordinator via three different paths, characterized by 1, 2 or 3 hops, and different resulting QoS values, namely round-trip time RTT_i and reliability R_i for i hops. Device 2 has two possible paths, whereas device 3 has only one path. If a user asks for the level of CO in room X and wants the data in real time, with a maximum latency (`qos-thr`) of 15 ms, the VIM will select device 1 and will notify to the IoTC the topology to be used to trigger such node in order to guarantee the requested QoS feature. Once the IoTC receives the request from the VIM, it will program the IoT network according to the selected topology.

B. The IoT controller and network

The main functionalities of the IoT controller are: gathering information from devices, maintaining a representation of the network and establishing routing paths.

In order to achieve the decoupling of the control plane from the data plane, it is fundamental that each device can discover a path toward the coordinator. This is done during the network initialization, as follows. When the coordinator turns on, it sends a *Hello* packet, containing the number of hops from the coordinator (zero in this case). When a device A receives this packet it performs the following operations: i) add the source of the *Hello* and the received signal strength indicator (RSSI) in the list of nodes (neighbors table) that are one hop distant from A; ii) analyze the distance contained in the *Hello* and the RSSI of the received message; iii) compare these values to the corresponding stored values: if the number of hops is lower and the RSSI is higher, the source of the *Hello* is elected as the best next hop toward the coordinator, and the values stored in A are updated. *Hello* packets are sent in broadcast by all devices in order to update neighbors tables. Moreover, the latter tables are periodically sent to the coordinator using the best next hop selected in the initial phase, and then forwarded to the controller to update the database, with the current map of connected devices. Based on this network representation, the controller may define paths. In our implementation topologies are created by assigning to each link a cost having an inverse proportionality w.r.t. the RSSI and by running a modified Dijkstra algorithm, where we impose a maximum number of hops equal to H, and we run the algorithm for different values of H in order to obtain different topologies, corresponding to possibly various performance levels in terms of latency and reliability, as shown in Table I.

Requests coming from the VIM are forwarded by the IoTC to the proper IoT coordinator, together with the information about the selected path connecting the coordinator and the intended device to be setup to guarantee the requested QoS. This path is then forwarded by the coordinator to all devices belonging to the route itself (through the transmission of a packet called *Path*), in order to update the flow tables at devices. In case a device receives a packet for handling which it has no information, a *RuleRequest* packet is sent through the route defined in the initial phase to the controller, that after elaborating it, will reply sending a *RuleResponse*.

V. OPENFLOW AND CLOUD DOMAINS

In this section we consider both the wired SDN domain and the cloud computing domain depicted in Fig. 1, assuming that they are managed by a single SDN/cloud VIM. The data plane topology assumed for the use case considered in this paper is shown in Fig. 2. An OpenFlow-based SDN infrastructure is assumed to be in charge also of the connectivity within the cloud domain, thus providing programmable traffic steering functionality to VNF chains. All the switches included in the topology (`s1`, `s2`, ..., `s7`) are OpenFlow-enabled devices and are governed by an SDN controller (e.g., ONOS [18]), whereas the computing infrastructure is managed through a cloud platform (e.g., OpenStack [19]).

Switch `s6` is an edge device connecting the IoT gateways in the IoT SDN domain to the cloud network. Router `vrl` is the (virtual) edge router of the (virtual) tenant network responsible for the connectivity within the cloud domain of the requested IoT data collection service. Switches `s1` to `s5` are either physical or virtual switches used by the tenant network

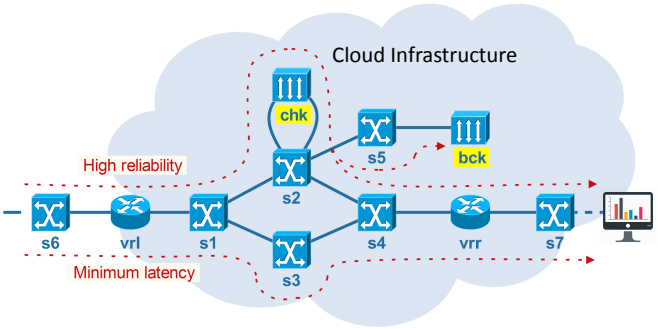

Fig. 2. Data plane topology of the OpenFlow and cloud domains considered for the use case.

for VNF connectivity. Two VNFs are deployed in the cloud: `chk` performs integrity and sanity check on the collected data for improved reliability, whereas `bck` is used to store backup copies of the collected data. Router `vrr` is the (virtual) edge router of the (possibly different) tenant responsible for the IoT data collection, processing, and publishing services. Switch `s7` is a (virtual) switch in the latter tenant's network, providing layer-2 connectivity to the server `ServP` where collected data are processed and published.

According to the QoS features of the use case considered here, the connectivity service offers two different paths in the OpenFlow domain. One path is characterized by minimum latency, where switches are configured with small buffers being continuously monitored by the SDN controller for possible congestion, and such that no VNF processing is performed, which could introduce additional delays. The other path is dedicated to highly reliable traffic flows, where switches have large buffers to reduce losses, and data are processed by `chk` and duplicated at switch `s2` in order to be stored in `bck`.

Therefore, depending on the QoS feature requested by the customer, the high level management and orchestration functions can specify two different service chains. Assuming that, based on the interaction between the orchestrator and the IoT VIM, incoming data will be collected from IoT network k and then forwarded to `ServP`, according to the JSON format specified in Section III the intent-based request to the SDN/Cloud VIM NBI could be

```
{
  "src": "IoT-GW[k]",
  "dst": "ServP",
  "qos": "Max delay",
  "qos-thr": "10 ms",
  "vnfList": "null",
  "dupList": "null"
}
```

for the low latency QoS feature, or

```
{
  "src": "IoT-GW[k]",
  "dst": "ServP",
  "qos": "Reliability",
  "qos-thr": "99%",
  "vnfList": [chk, bck]
  "dupList": [bck]
}

chk ::= {
  "name": "chk",
  "terminal": "false",
  "port_sym": "true",
  "path_sym": "false"
}

bck ::= {
  "name": "bck",
  "terminal": "true",
  "port_sym": "null",
  "path_sym": "false"
}
```

for the high reliability QoS feature. The SDN controller must implement a data plane monitoring service to make sure that, in the former case, the minimum latency path guarantees the requested maximum delay of 10 ms, whereas in the latter case the VNFs inserted in the service chain and the high reliability path ensure the required 99% accuracy.

We developed the VIM for the SDN/cloud domains as an application running on top of the ONOS platform. It is worth to note that ONOS already provides a built-in, intent-based NBI that can be used to program the SDN domain and deploy the required network forwarding paths. However, in order to specify the ONOS intents, some knowledge is required of the specific data-plane technical details, while in our approach we prefer to expose only high-level abstractions to the orchestrator. Therefore, one of the main functions of our VIM is to implement new, more general and abstracted intents that can be expressed according to the NBI specification given above. Then the VIM takes advantage of the network topology features offered by the SDN/cloud controllers to discover VNF location in the cloud and relevant connectivity details, and eventually it is able to compose native ONOS intents and build more complex network forwarding paths.

The VIM can be instantiated as an ONOS service called *ChainService*, which provides the capability of dynamically handling the VNF chains through the abstracted NBI defined in Section III. To achieve extensibility and modularity, the implementation of ChainService is delegated to a module called *ChainManager*, which is in charge of executing all the required steps to translate the high-level service specifications into ONOS-native intents. The input to ChainManager can be given through either the ONOS command line interface (CLI) or a REST API. The latter is preferable because it allows remote applications to use standard protocols (e.g., HTTP) to access resources and configure services. In our implementation, the REST API provides the following service endpoints:

```
POST /chaining/{action}/{direction}
DELETE /chaining/flush
```

In the former endpoint, the `action` variable indicates the operation that the orchestrator intends to perform on a specified service chain (`add`, `update`, or `delete`), whereas

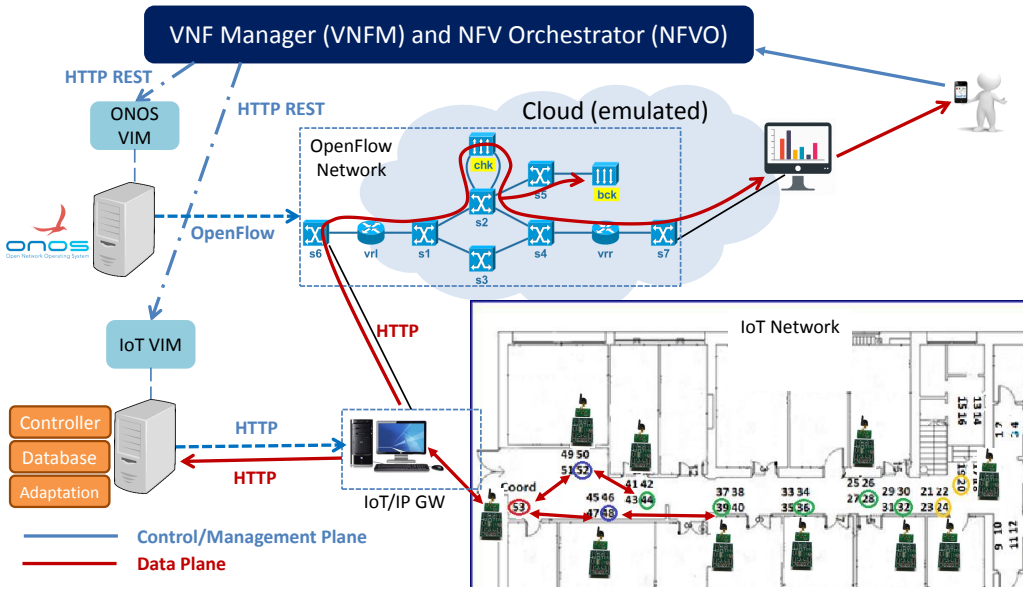

Fig. 3. The NFV/SDN test bed setup developed to demonstrate multi-domain SDN/NFV management and orchestration. Information flow in the management and control planes is displayed in blue. Information flow in the data plane is displayed in red.

in case of an update the `direction` variable (`forth`, `back`, or `both`) defines whether the modified chain specification refers to the existing forwarding path from `src` to `dst`, the opposite way, or both directions. So the basic operations of this endpoint are as follows:

- If the `add` action is given, this will result in defining a new service chain, based on the JSON specification included in the message body. This means that a forwarding path will be created for traffic flowing from `src` to `dst` and another one in the opposite direction. Note that the two paths are not necessarily symmetric, based on the topological abstractions defined by the NBI.
- If the `update` action is given, then the direction is taken into account and the forward path, backward path, or both paths of the specified existing service chain are changed. In fact, a user may be interested in changing only a segment of the forwarding path and only in one direction, to reduce the control plane latency and limiting the impact that a path change can have on the existing traffic flows.
- If the `delete` action is given, then both forwarding paths of the specified existing service chain are removed.

ChainService provides also the `flush` operation through another endpoint, thus offering the possibility of deleting in a single step the forwarding paths of all the service chains previously created.

VI. Experimental Validation

A. Test bed setup

As a demonstration of the feasibility of the multi-domain SDN/NFV management and orchestration solution proposed here, we developed a test bed to implement the reference architecture of the cloud-based IoT data collection service with quality differentiation illustrated in Fig. 1. The complete test bed setup, including the components discussed in Sections IV and V, is shown in Fig. 3. The customer on the top-right corner requests the service to the high-level management and orchestration functions, specifying the desired QoS feature. The orchestrator then forwards the request to the VIM REST NBIs of the relevant domains using the JSON format described in the previous sections. Each VIM performs the operations required in the respective domain and programs the underlying controllers according to the requested service and QoS feature. Data generated by the IoT devices are sent by the relevant gateway via HTTP POST to the collecting/processing/publishing server in the cloud, where the customer can retrieve it (the case of high reliability QoS feature is shown in the figure).

In the test bed, the OpenFlow SDN domain and the cloud domain were emulated using Mininet running in a virtual machine [20]. The data plane topology in Fig. 2 was built with a customized Mininet script specifying the required OpenFlow switches, as well as routers and VNFs as separated network namespaces. Additional virtual machines were instantiated to deploy the data collection/processing server and the ONOS platform components. In order to provide the two paths with different latency, `chk` was configured to introduce an additional random delay uniformly distributed between 25 and 35 ms, with 25% correlation between consecutive samples.

As far as the IoT domain is concerned, in our implementation we setup an IoT network using the "European Laboratory of Wireless Communications for the Future Internet" (EuWIn) platform and, in particular, the flexible topology test bed (Flextop) facility [21], [22]. The lab was composed of 53 TI CC2530 devices, compliant with IEEE 802.15.4 [22] on top of which our SDN protocol stack was running. Nodes were located into boxes on the walls of a corridor at the University

TABLE II
AVERAGE RTT AT THE IOT DATA PLANE (DP), CONTROL PLANE (CP) AND VIM, FOR IOT DEVICES CONNECTED TO THE COORDINATOR ACROSS 1, 2 OR 3 HOPS.

No. of hops	RTT at DP	RTT at CP	RTT at VIM
1	12.6 ms	516.7 ms	522.2 ms
2	26.2 ms	530.2 ms	536.4 ms
3	40.4 ms	545.7 ms	550.5 ms

TABLE III
AVERAGE AND STANDARD DEVIATION OF DATA PLANE (DP) ONE-WAY LATENCY COMPUTED AT THE EMULATED CLOUD NETWORK. CORRESPONDING RTT MEASURED AT THE IOT CONTROL PLANE.

QoS feature	Average lat.	Stdev	RTT at IoTC
Min Latency	0.3 ms	0.28 ms	1.8 s
High Reliability	31.7 ms	2.41 ms	1.8 s

of Bologna. Thirteen boxes were deployed in the corridor, and four nodes per box were deployed at fixed positions. The map with the corresponding identifiers of nodes is shown in the bottom-right part of Fig. 3, where nodes marked with circles are those selected for the experiments. In particular, in the figure we show an example of topology, when setting $H = 3$ (maximum three hops to reach the coordinator): nodes with blue circles were connected via one hop, nodes in green via two hops, and nodes in yellow via three hops. IoT data gathered by the gateway were then duplicated and sent to: 1) the IoTC and then to the IoT VIM; 2) to the Mininet virtual machine and then to the cloud, from which the user could read the measured data.

Because of the technological heterogeneity of our test bed, we were able to directly measure the performance of each domain separately. However, the results obtained allow us to reasonably infer the assessment on the end-to-end service.

B. IoT domain performance

We first considered an application where the user asks for the data measured by each IoT device (that could be at 1, 2 or 3 hops from the coordinator), one by one, and waits for the reply. In the IoT network, both the request and the reply data frames have a payload of 10 bytes, and queries were generated by the user every second. For each query we measured: i) the RTT at the IoT data plane, that is the interval of time between the reception of the query coming from the IoTC at the application layer of the IoT coordinator, and the reception of the reply from the target node, again at the application layer of the coordinator; ii) the RTT at the control plane, that is the interval of time between the reception of the query coming from the VIM at the IoTC, and the reception of the reply coming from the intended GW, again at the IoTC; iii) the RTT measured taking the time stamp at the IoT VIM.

Performance were evaluated by averaging over 10,000 queries generated by the user toward nodes with distance 1 hop, 2 hops and 3 hops. Results are shown in Table II. The average RTT at the IoT data plane is mainly influenced by the number of hops, giving latency values in the order of tens of milliseconds that can be considered acceptable depending on the required QoS. As for the RTT measured at the control plane and VIM, results demonstrate that the most significant contribution depends on the IoTC response time, which is around half a second, a relatively small value.

It is important to underline that paths in the IoT network were refreshed periodically and not at every query received. In particular, in our implementation we sent one *Path* packet per device to be queried (all the nine nodes switched on in this case) every 250 s. As a result, the control plane RTT is not constant, but presents some peaks when a new *Path* packet is generated. We measured the standard deviation of such control plane RTT that was equal to 280 ms (considering all measurements taken).

C. OpenFlow domain performance

We measured the performance within the emulated cloud network when the customer requested the service specifying two traffic classes, according to the QoS features offered by the OpenFlow SDN domain: minimum latency and high reliability. In this case, one-way latency in the emulated cloud network was measured by comparing timestamps of each packet captured at switches `s6` and `s7`. The capture was performed in the server hosting the Mininet virtual machine, so the same reference clock was used for the sake of accuracy. The measurements were made by averaging over 10,000 requests.

Results are reported in Table III in terms of average and standard deviation of the data plane one-way latency. The numbers show the correct behavior of the OpenFlow domain with respect to the requested QoS feature: very limited delays were measured in the minimum latency case, whereas in the high reliability case no packet was lost and `bck` successfully stored a copy of the entire data set transmitted by the IoT GW.

In the table we also report the RTT measured at the IoT SDN controller when setting up the complete end-to-end path from an IoT device to the collecting server in the cloud. With respect to the values reported in Table II, the significant increase in the IoTC response time is due to the controller that must wait for an HTTP OK message from the collecting server in the cloud before considering the data gathering service as successfully established. However, such an increased response time is caused mainly by the IoT GW, a node with limited processing power that must setup an HTTP session with the collecting server.

Finally, we measured the NBI response time at the VIM implemented in ONOS, i.e. the time required by the VIM to process a JSON service chain specification. To assess the scalability of the NBI, we generated an increasing number of requests (from 5 to 200) sent in a batch to the VIM. Each measured response time was obtained as an average over 20 runs with the same number of requests. Figure 4 shows the average NBI response time with 95% confidence intervals. The numbers show that the VIM is very responsive, in the order of tens of milliseconds. The setup of high-reliable service chains takes slightly longer than the minimum latency ones because of the relatively more complex service chain to be processed.

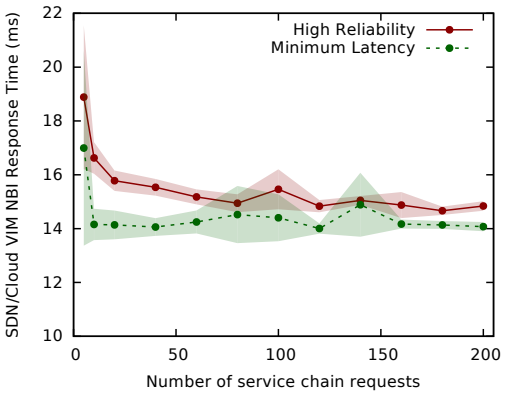

Fig. 4. Average NBI response time and 95% confidence interval at the SDN/cloud VIM with increasing number of service chain requests.

VII. Conclusion

In this paper we proposed a reference architecture, inspired by the ETSI MANO framework, and an intent-based NBI for end-to-end service management and orchestration across multiple technological domains. In particular, we considered the use case of a software-defined IoT infrastructure connected, by means of an OpenFlow-based SDN domain, to the related cloud-based services. We validated the proposed architecture over a heterogeneous OpenFlow/IoT SDN test bed, demonstrating the feasibility of the approach and the potentials of the NBI. The latency values measured at both data and control/management planes allowed us to get a first insight to the performance levels of the overall system, resulting in reasonable response times for service setup and QoS requirement satisfaction. Scalability tests on the ONOS-based VIM also gave promising results. The use case reported here represents a working example of a more general approach to properly define high-level interfaces and develop the related control and management components to unify orchestration capabilities across multiple SDN/NFV domains.

Acknowledgment

This work was partially funded by EIT Digital, Action Line on Future Networking Solutions, Activity no. 16406: "CC4BA - Certification Centre for Business Acceleration of SDN and NFV," by the COST Action CA15104-IRACON, and by the Horizon 2020 grant no. 644090: "EuroCPS - European Network of competencies and platforms for Enabling SME from any sector building Innovative CPS products to sustain demand for European manufacturing."

Authors would also like to thank Slavica Tomovic from University of Montenegro, for her contribution to the implementation of the IoT controller.

References

[1] J. Soares, C. Goncalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, February 2015.

[2] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, October 2016.

[3] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, First quarter 2016.

[4] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, Fourth quarter 2014.

[5] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, and A. Manzalini, "SDN for dynamic NFV deployment," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 89–95, October 2016.

[6] R. V. Rosa, M. A. S. Santos, and C. E. Rothenberg, "MD2-NFV: The case for multi-domain distributed network functions virtualization," in *2015 International Conference and Workshops on Networked Systems (NetSys)*, March 2015, pp. 1–5.

[7] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–4.

[8] B. Sonkoly, J. Czentye, R. Szabo, D. Jocha, J. Elek, S. Sahhaf, W. Tavernier, and F. Risso, "Multi-domain service orchestration over networks and clouds: A unified approach," in *2015 ACM SIGCOMM Conference*, August 2015, pp. 377–378.

[9] *Intent NBI - Definition and Principles*, The Open Networking Foundation (ONF) Tech. Rec. TR-523, October 2016. [Online]. Available: https://www.opennetworking.org/sdn-resources/technical-library

[10] T. Luo, H. P. Tan, and T. Q. S. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, November 2012.

[11] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of software-defined sensor networks," in *Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth International Conference on*, Dec 2013, pp. 410–413.

[12] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 513–521.

[13] A. Mahmud and R. Rahmani, "Exploitation of OpenFlow in wireless sensor networks," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, vol. 1, Dec 2011.

[14] T. Miyazaki, S. Yamaguchi, K. Kobayashi, J. Kitamichi, S. Guo, T. Tsukahara, and T. Hayashi, "A software defined wireless sensor network," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*, Feb 2014.

[15] C. Buratti, A. Stajkic, G. Gardasevic, S. Milardo, M. D. Abrignani, S. Mijovic, G. Morabito, and R. Verdone, "Testing protocols for the Internet of Things on the EuWIn platform," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 124–133, Feb 2016.

[16] *Network Functions Virtualisation (NFV); Management and Orchestration*, The European Telecommunications Standards Institute (ETSI) Std. GS NFV-MAN 001, Rev. 1.1.1, December 2014. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[17] FROG: an SDN/NFV/cloud service orchestrator. [Online]. Available: https://github.com/netgroup-polito/frog4

[18] ONOS: Open Network Operating System. [Online]. Available: http://onosproject.org

[19] OpenStack: Open Source Cloud Computing Software. [Online]. Available: https://www.openstack.org

[20] Mininet: An Instant Virtual Network on your Laptop. [Online]. Available: http://mininet.org

[21] M. D. Abrignani, C. Buratti, D. Dardari, N. E. Rachkidy, A. Guitton, F. Martelli, A. Stajkic, and R. Verdone, "The EuWIn testbed for 802.15.4/Zigbee networks: From the simulation to the real world," in *The Tenth International Symposium on Wireless Communication Systems (ISWCS)*, August 2013.

[22] "D22.1 - Definition of EuWIn@CNIT/bologna testbed interfaces and preliminary plan of activities." NoE-Newcom♯, April 2013.