Received: 5 February 2019 | Revised: 26 February 2020 | Accepted: 7 April 2020

DOI: 10.1002/nem.2116

RESEARCH ARTICLE

WILEY

A framework for multi-provider virtual private networks in software-defined federated networks

Habib Mostafaei[1] | **Gabriele Lospoto**[2] | **Roberto Di Lallo**[2] | **Massimo Rimondini**[3] | **Giuseppe Di Battista**[2]

[1]Department of Telecommunication Systems, Technische Universität Berlin, Berlin, Germany

[2]Department of Engineering, Roma Tre University, Rome, Italy

[3]Network Design and Engineering, Unidata S.p.A., Rome, Italy

Correspondence
Habib Mostafaei, part of this work has been conducted while Habib Mostafaei was with Roma Tre University.
Email: habib@inet.tu-berlin.de

Funding information
German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data, Grant/Award Number: ref. 01IS18025A and ref 01IS18037A

Summary

Federated networks represent a remunerable operational way, allowing federated partners to increase their incomes through a sharing resource process. They have been primarily used in the context of cloud computing; nowadays, they are also used to provide connectivity services, like virtual private networks. However, providing such a service by using standard technologies in federated networks requires a nonnegligible effort from different points of view (e.g., configuration effort). In this paper, we propose an software-defined network (SDN)–based framework aiming at overcoming limitations in currently adopted best practices to issue virtual private networks in federated networks. Relying on the SDN architecture, we propose a method allowing federated providers to quickly and easily create federated networks, reducing unneeded costs (e.g., new hardware), and a way for customers to fast-access federated services, without any explicit actions from providers. We evaluate our framework by using SDNetkit. We focus on analyzing the impact of our implementation on both control and data plane, in terms of number of control messages exchanged in the network and size of the forwarding tables, respectively.

1 | INTRODUCTION

Federated networks represent a collaborative operational way for Internet service providers (ISPs) to increase revenues by sharing resources.[1] A federated network can be defined as a network in which federated partners or members (e.g., ISPs) share their resources with any other federated member to satisfy growing demands from customers or possibly issue value-added services (e.g., services that could not be provisioned without the federated network itself).

One of the most relevant benefits of such a network is the increase of providers' incomes. However, there are critical steps that must be carried out to join a federated network, like identifying and defining cost models and agreeing on standard operational tasks (e.g., monitoring). Those examples are just a shortlist of needed steps, but they give the idea that creating a federated network is not so trivial. At the beginning, federated networks were defined to operate in the context of cloud computing. Nevertheless, the promising benefits brought by them encouraged ISPs to provide other services. During the years, several projects arose, like GÉANT[2] and Beacon,[3] with different aims while sharing the same idea of federation.

After discussing with several Italian ISPs, we asked ourselves whether the benefits of federated networks could be exploited to issue other services (e.g., connectivity). One of the most used connectivity services in today's networks is virtual private networks[4] (VPNs). Issuing that service in a network directly managed by a single ISP is not trivial, since many protocols must cooperate to set up a VPN. Also, the provisioning of that service is expensive at least in terms of time. As a consequence, it is even more challenging to span a VPN over two or more networks that are managed by different ISPs by definition of a federated network. Actually, such a service is provisioned in a federated fashion by GÉANT.[5] One of the strong points they specify in describing their VPN service is the ability to *quickly deliver* it: "5 days are needed." Thus, our question is: Is there a way to reduce such a provisioning time?

In this paper, we propose a framework allowing federated ISPs to quickly and easily as follows:

1 create a federated network,
2 set up a VPN service, and,
3 allow customers to join or leave from the service autonomously, namely without any direct actions (e.g., configuration activity) performed by the ISPs.

We define such a service *federated VPN*, namely a VPN allowing customers connected to different federated ISPs' networks, but in the same VPN, to exchange traffic with each other. Our framework relies on SDN and is built on top of di Lallo et al[6] and Mostafaei et al.[7]

The main contributions of this paper can be summarized in the following:

1 drastically reducing the time needed to provision the federated VPNs and
2 relying on the SDN architecture to set up the federated network and to provide the federated services.

We reach those goals by providing a configuration language that allows each federated ISP to easily define federated networks as well as quickly configure and provision federated VPNs. We provide a set of primitives that allow customers to join or leave from federated VPNs *on-demand*, thus reducing both explicit configuration actions of the ISPs and the time required to set up or down the federated VPNs. Also, to allow ISPs to keep unchanged their whole network architecture, we introduce in the network the smallest number of software-defined network (SDN)–enabled devices. Finally, we exploit the SDN architecture to interact with the application level (e.g., the domain name system [DNS] system), allowing each ISP to keep unchanged any IP address plan previously assigned to its customers. The interaction with the DNS is crucial. Indeed, inside a federated network, there are no guarantees about the fact that the IP addresses of each customer participating in a VPN do not overlap. Since this requirement is mandatory, such interaction allows providers to not change the IP address plan of the customers.

Our framework does not have any significant impacts on the ISPs' networks. Indeed, it is completely agnostic with respect to any forwarding strategy adopted by the ISP (e.g., IP or Multiprotocol Label Switching [MPLS]). Also, our framework does not have any impact over any existing configuration: federated VPNs built exploiting our framework coexist with standard VPN set up by using legacy technologies. A customer can be simultaneously served by a federated VPN and a standard one without any limitations. We claim that a strong add-value of our framework is the following: *customers sharing the same IP address plan can be part of the same federated VPNs, without any limitation.* In the evaluation, we focus on two coordinates: control- and data-plane impact. Our experiments show that the number of control plane messages linearly grows to the number of DNS queries in the network, regardless of the amount of traffic injected in the network. We also observe that the impact on the data-plane measured in terms of SDN rules installed at the SDN-enabled devices linearly depends on the number of customers per ISPs that join the federation.

The rest of the paper is organized as follows. In Section 2, we review the state of the art. In Section 3, we discuss today's best practices for federated networks. In Section 4, we show our framework and how it can be used to tackle the most common problems in today's federated networks. In Section 5, we present our configuration language and our primitives, explaining how they allow fast delivery of the service. In Section 6, we show a complete example, detailing the interaction between SDN and the DNS. In Section 7, we summarize the benefits of our framework. In Section 8, we discuss the results of our experiments. Finally, in Section 9, we draw conclusions and future research perspectives.

2 | RELATED WORK

In this section, we review the most relevant literature proposing SDN as the architecture to support the provisioning of federated services in federated networks. Also, we compare with proposals to set up VPNs over different ISP's networks.

Federated networks are widely used for cloud computing.[1,8,9] Over the years, several aspects have been addressed, starting from analyzing architectures for federated networks. In a previous work,[8] the authors present a layered architecture to provide cloud services (IaaS, PaaS, and SaaS). Such services are provisioned exploiting a collaboration among providers that share their resources, aiming at increasing their incomes. The authors in[10] discuss the basis theory of sharing economy among cloud providers to offer cost-efficient services to the customers. The fundamental challenges of offering an scalable and dependable service spanning on multiple cloud providers are discussed in OPTIMIS project.[11]

In a previous study,[1] the authors discuss models to guarantee specific levels of remuneration for federated cloud services. Also, toolkits for modeling and simulating cloud services have been discussed in Calheiros et al.[12] Attempts of using SDN to issue federated cloud services have been made, as reported in Koerner et al,[9] where they propose an architecture in which a software agent handles shared resources used to issue the cloud service. Meantime, several research projects, like GÉANT[2] and Beacon,[3] arose. They build federated networks in which federated services (e.g., VPNs) are issued by sharing resources owned by each federated ISP, investigating the potential of such a model in terms of performance and costs effectiveness.

We argue that other services (e.g., connectivity) beyond cloud computing can take advantage of the federated network model. Indeed, the GÉANT network also offers federated VPN services[5] to its customers by relying on VLANs to cross multiple ISP's networks. As they admit, the provision of a VPN in their network requires a nonnegligible amount of time (order of days). In a previous work,[13] the authors propose a mechanism based on the LISP protocol[14] to span a VPN in a multi-provider network. The main drawback is that each ISP must adopt LISP. The work in Andrade et al[15] analyzes the performance and the benefits of offering a service in a federated network. The study aimed to check the performance variation of network metrics like download speed and latency in such networks. Layers 2 (L2) and 3 (L3) challenges of creating a federated network in a hybrid cloud environment were studied in Moreno-Vozmediano et al.[16]

All the proposals that rely on standard technologies either require a nonnegligible amount of time to make the federated services available to the users or impose assumptions that are not applicable in real networks. In both cases, the provisioning is highly impacted. In contrast, our framework aims at simplifying the provisioning of the federated services, demanding to the SDN-controllers the management of all the aspects related to the federation. Thus, both the configuration effort and the amount of time to set up the federated services are reduced.

We believe that SDN architecture is a key component in dealing with current challenges for federated networks[17] and for providing new services. In di Lallo,[6] we proposed a mechanism based on SDN to support end-to-end connectivity spanning several ISP's networks, and in Mostafaei,[7] we built on top of that paper a mechanism based on the DNS to simplify the communication among end-hosts. In this paper, we extend di Lallo et al[6] and Mostafaei et al[7] by providing a framework to easily subscribe to federated VPN services, avoiding delays introduced by provisioning issues. Unfortunately, federated networks still have to deal with challenges[17] that make the provisioning of federated services difficult and costly, both in terms of money and human resources. Relying on SDN, we address those challenges, proposing a framework that is a first, but a complete step, addressing today's federated network issues.

3 | BEST PRACTICES FOR FEDERATED NETWORKS

In this section, relying on the GÉANT project,[2] we describe the typical architecture and the best practices adopted to create a federated network.

The main idea behind the GÉANT federated network is what they call *federated PoP*.[18] A federated PoP is a physical place in which all ISPs involved in a federation connect. In general, establishing a federated PoP needs many steps, consisting of different activities. For instance, there is the need for establishing connectivity (e.g., by using dark fiber), as well as overcoming technical difficulties (e.g., due to different physical layer technologies). Other steps regard the need of installing and using new hardware (e.g., switches) that will be used by each ISP to connect and all equipment to monitor the services issued by the federation. The network hardware in a federated PoP can be either hardware owned

by the provider itself or shared hardware owned by the federation. It is easy to note that the federated PoP architecture strictly recalls that of any Internet eXchange Point (IXP), where providers are interconnected to allow their customers to exchange traffic.

Federated PoPs allow each ISP to identify a specific way to join a federated network, as well as to isolate the federated network traffic from the standard one; however, a federated PoP introduces costs for both installing and maintaining (also including configuration effort) the devices used in that place. Moreover, a nontrivial agreement process has to be carried out to clearly define operations and responsibilities among federated ISPs in the federated PoP. Once a provider connects to a federated PoP and agrees on the policies, it can start to share resources with other providers and to issue services. Even if this model has been recognized to become the standard architecture for federated networks, it also introduces challenges,[17] like the following:

1 *management*, namely the need of collaboration in standard network operations, like configuration, troubleshooting, and monitoring;
2 *technological differences*, namely the lack of well-defined standards could originate problems due to different ways to realize the forward traffic at physical layer; and
3 *user view*, namely the absence of a common interface clearly describing how to access federated services, hiding the collaboration among providers to the final users.

We argue that the aforementioned challenges involve not only technological aspects. On one hand, coordination activities must be carried out to plan the architecture (e.g., protocols or components needed to issue services). On the other hand, identifying responsibilities inside the federation itself and agreeing on costs (e.g., federated service prices and the level of remuneration for each provider, possibly based on the resources it shares in the federation) is a task that needs to be accomplished by nontechnical staff inside the provider. This observation might have a strong impact especially when a new service is issued for the first time.

4 | SDN-BASED FEDERATED NETWORKS

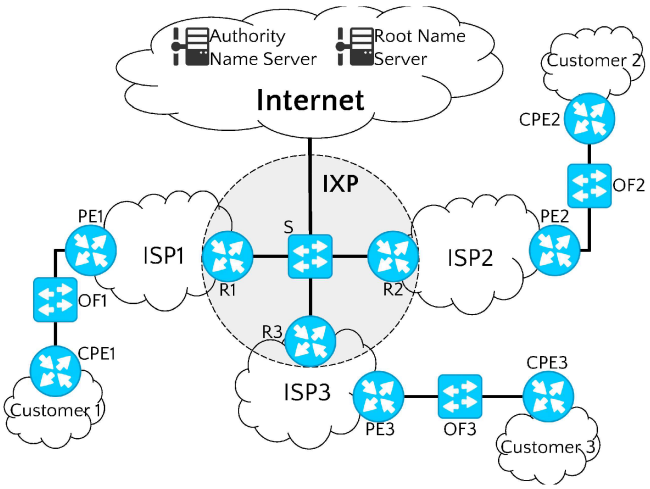
In this section, we present an SDN-based framework dealing with the main problems related to the implementation of federated networks. By relying on the SDN architecture, we argue that our solution can address all challenges reported in Section 3 (and discussed in Fischer et al[17]), namely as follows:

1 management problems,
2 technological differences problems, and
3 absence of a unified user view.

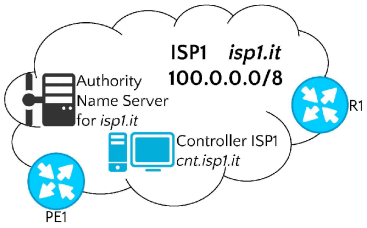
We choose to rely on the SDN architecture since it brings flexibility in issuing services and it makes the provisioning phase easier. Such a choice allows us to overcome the challenges in current federated network architecture. Indeed, we identify one main problem in the architecture described in Section 3, namely the federated PoP. On one hand, such an interconnection point brings several benefits (e.g., clear identification of a place in which providers can federate and clear responsibilities assigned to each federated provider). On the other hand, federated PoPs are *duplicates* of IXPs, requiring further effort for ISPs in terms of expenses and configurations (e.g., buying and managing devices used in the federated PoP). We argue that having a BGP session with an ISP or being connected to an IXP is enough to create a federation and this requirement is easily satisfied by ISPs. Also, each ISP can be connected to several IXPs simultaneously as well as it can have multiple bilateral peerings with no restrictions. Our framework relies on this observation. By doing so, we preserve all the benefits of a federated PoP (e.g., clear identification of responsibilities) and save addictive costs due to new hardware.

4.1 | Reference scenario

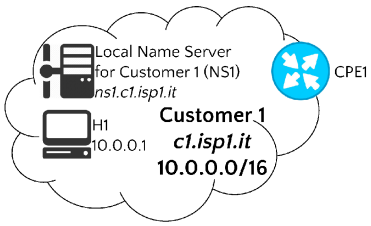
A reference scenario for our framework is depicted in Figure 1. We assume that each partner of the federated network is an ISP offering connectivity to a set of customers. The federated network of Figure 1A has exactly three partners



(A) Overview of a federated network including three ISPs connected through an IXP allowing them to exchange traffic to each other and to access Internet.



(B) A detalied view of the internal architecture of a provider.



(C) A detalied view of the internal architecture of a customer.

FIGURE 1 Reference scenario for software-defined network (SDN)–based federated networks

whose networks are called ISP1, ISP2, and ISP3, respectively. Such networks run IP-based routing protocols inside their backbones (e.g., OSPF for intra-domain routing and BGP for inter-domain routing). Routers R1, R2, and R3 are border routers establishing a BGP peering as in traditional networks not involving federations.

We represent the IXP connecting the federated ISPs through a dashed circle with a light grey background. In the IXP, we assume—without loss of generality—that all ISPs are connected through a legacy (no-SDN) layer 2 switch S. Exploiting the IXP itself, each ISP can also reach the Internet. We assume that somewhere on the Internet there are (at least) two name servers as follows:

1 a root and
2 an authoritative name server for the ISPs' domains.

Routers PE1, PE2. and PE3 are Provider Edges (PEs) and collect the traffic coming from the customers attached to the IPS's network. Each ISP in Figure 1A has one customer (Customer1, Customer2, and Customer3, respectively). Each customer is connected to the ISP's network through an IP-speaking router acting as a Customer Premise Equipment (CPE); those devices are CPE1 for Customer1, CPE2 for Customer2, and CPE3 for Customer3. Each of those routers is, in turn, connected to an SDN-enabled device, more specifically an OpenFlow-enabled switch (OF1 for Customer1, OF2 for Customer2, and OF3for Customer3); placing such devices between the CPEs and the PEs allows us to take the control of all traffic generated by each customer.

Figure 1B,C depicts the internal architecture of an ISP and of a customer, respectively. Referring to Figure 1B, each ISP has a public IP subnet used to allow the communication over the Internet and it has a public domain name (isp1.it for ISP1). The same happens for ISP2 and ISP3, even if we do not report in this paper a specific drawing. Inside each ISP's network, there is an SDN-controller, for example, cnt.isp1.it, having in charge of the management of each SDN-enabled device. We assume that each ISP belonging to the federated network has an SDN-controller in order to provide the service.

With respect to the internal architecture of a customer (Figure 1C), we assume that it has a private IP address subnet used for internal purposes. Also, there is a local name server (NS1 with domain name ns1.c1.isp1.it). The same happens for Customer2 and Customer3, even for the internal IP address subnets: indeed, we allow communication among end-hosts possibly sharing exactly the same IP address. The local name servers might be placed in a publicly accessible portion of the network. If this is the choice, each machine inside the customer must be reconfigured pointing to such an external device. Leaving the local name servers inside the local networks, there is no need for this extra effort. Also, placing those servers behind the SDN-enabled switch allows us to take control of the DNS traffic generated by the customers.

Referring to Figure 1, when H1 (residing in Customer1) wants to exchange traffic with H2 (residing in Customer2), we say that those customers *join* a federated VPN allowing them to send traffic to each other. This operation is steered by the SDN-controllers of each federated ISP, which undertake specific operations in order to set up the federated VPN. Note that by using standard technologies (e.g., layers 3[4] or 2 VPNs[19]), this service cannot be provided, since IP addresses overlap is forbidden. Note that Customer1 can be part of any other VPN provided by ISP1 using standard technologies, as for example, MPLS VPN. Also, each ISP can still provide services that are not SDN-based without any restrictions.

4.2 | Management problems

Management problems happen when common network operations, such as monitoring activities, have to be carried out. Such operations need strong coordination among members of the federated PoPs[17] and agreeing to reach that goal might not be a trivial process. It is easy to see that if the federated network is built exploiting a federated PoP, those activities might involve working teams belonging to different providers. Also, systems used to carry out such activities should include the policies of all federated members, leading to an increase in complexity.

Our framework does not introduce any further connection point, except the IXP which each provider is already connected to. By doing so, each federated ISP carries out performance and monitoring activities independently by each other federated provider, reducing both hardware and human resources costs. Besides, as all customers are forced to send traffic passing through SDN-enabled devices (as shown in Figure 1), tracking the traffic belonging to the services provided by the federation is also easy. This is not the only benefit that our framework brings. Indeed, each provider can autonomously accomplish the task related to the recognition of responsibilities, since there are not shared interconnection points (e.g., federated PoPs) and each federated ISP only manages its network, keeping costs unchanged.

4.3 | Technological difference problems

Technological problems arise when ISPs adopt different protocols to interconnect each other.[17] A preliminary step consists of agreeing on shared choices (e.g., in terms of routing protocols), allowing each ISP to interconnect to each other. Unfortunately, there is no standard interface to access a federated network,[17] resulting in a complication when a service has to be issued.

Our framework does not impose such constraints, since each federated ISP comes with its infrastructure that is completely independent of each other, allowing providers to choose protocols to use in their network, preventing coordination activities needed by the federated PoPs. Our framework requires to have several SDN-enabled switches, autonomously managed by each provider.

4.4 | Unified user view problems

Unified user view problems occur when a customer does not perceive the federated network as a single one. An example of this problem is the following: consider the case in which a customer wants to join a federated VPN. That customer asks its provider to set up the federated VPN allowing it to exchange traffic with another customer connected to the network of a federated partner. If such a request takes a nonnegligible amount of time for being accomplished (e.g., order of hours or days), a customer might perceive that its provider is not able to issue that service autonomously, but it needs to collaborate with other ISPs, making the federated network visible to the users.

A solution to this problem consists in providing a unified way to access federated services that allow each federated ISP to accomplish on-demand customer requests. Also, after receiving such requests, each federated ISP must act as much as possible independently, consequently reducing the collaboration to gain in terms of both the amount of involved human resources (e.g., technical staff for devices configuration) and required time to provision the service. Our framework has a set of primitives that each customer uses to access a federated service.

5 | SUBSCRIBING TO AN SDN FEDERATED VPN SERVICE

In this section, we describe a configuration language for supporting federated networks and VPN services and a set of primitives allowing customers to join or leave such services.

Our configuration language is simple and it just contains information about federated networks and the federated VPN service, without any impact on any existing configuration. The configuration, written with our configuration language, is located at the SDN-controller, and it does not require any additional configuration over standard IP-speaking routers. In particular, our configuration language includes a set of static information (e.g., which are the federated networks the provider belongs to and all information about the SDN-controllers of other federated ISPs) and a set of dynamic ones (e.g., list of federated VPNs) that are updated based on the customer requests performed by using our primitives.

Federated VPN is a collaborative service. Therefore, all SDN-controllers must have the configuration for that service always updated and consistent. To support these requirements, we internally design our SDN-controller as depicted in Figure 2. Incoming and outgoing arrows mean that those components can be publicly accessed by customers and remote controllers, respectively (e.g., they have a public IP address). Note that the SDN-controller is not publicly accessible, avoiding to expose it to the possible security issue. We divide our SDN-controller into three logical components as follows:

1 the Customer Request Handler, which handles the requests coming from the customers to join or leave from a federated VPN,
2 the Remote Controller Handler, which takes care of keeping the information about the federation and the federated VPNs updated, and
3 the SDN controller, which manages the SDN switches.

We highlight that the proposed architecture has the goal of preventing malicious users to take the control of the SDN-controllers, as well as avoiding denial of service attacks against the SDN-controllers themselves. We argue that the components that can be publicly accessible (from the Internet or by a possible malicious customer) are the Remote Controller Handler and the Customer Request Handler, namely those components that do not act the SND-enabled switches, preventing a direct interaction with the SDN controller. We state that such architecture is crucial. Indeed, if an SDN-controller was hacked, an attacker could install in the SDN switches arbitrary forwarding rules, so taking the control of the traffic issued by the customers. However, if one or more SDN-controllers was made unavailable, the

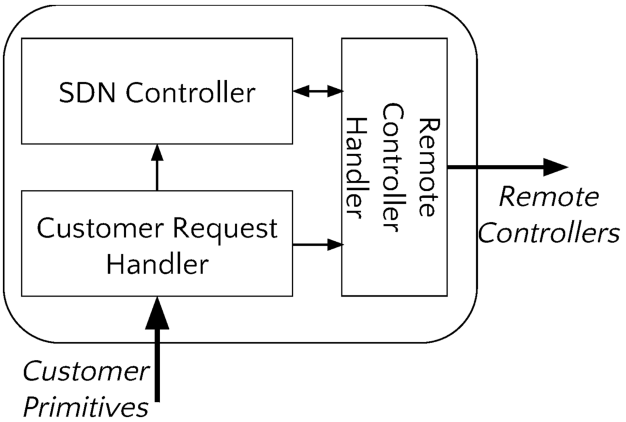


FIGURE 2 Logical architecture of our software-defined network (SDN)–controller, consisting of a set of components each devoted to a specific task

SDN-switches would not be able to properly handle the requests coming from the customers, making the federated service unusable. In this paper, we do not address problems related to robustness in the case of controller failures.

We also define a set of primitives used to keep the configuration consistent; they are used by customers that want to join (or leave) the service at any time. This makes our proposal more flexible and scalable to standard VPN services that require changes in the configuration files of the network devices to support such an *on-demand* feature. We argue that our framework makes the whole provisioning process more *agile*. For sure, the decision of federating with other providers requires a set of agreements that must be carried out (e.g., cost models) and they are not addressed by our framework. Nevertheless, our framework provides several mechanisms to make the creation of federated networks and the provisioning of federated services easier.

We argue that our configuration language and our primitives, together with the SDN architecture, represent our solution for solving problems discussed in Section 4.

5.1 | A configuration language for federated networks and federated VPNs services

Our configuration language is XML-based. The goal is to specify a set of parameters used to easily set up both federated networks and federated VPNs, without modifying any existing configurations. Our choice of relying on XML does not restrict the adoption of any other formats (e.g., JSON), as long as the semantic stays unchanged. The configuration is the input of the SDN-controller, that—based on its content—allows or denies a customer to access the service. We explain the structure of our configuration language relying on the following example in Listing 1.

Listing 1 A configuration example for the language.

```
<federations>
 <federation name="ISP1−ISP2−net">
  <myself>
    <isp id="1" name="isp1.it">
          <controller name="cnt.isp1.it" ip="100.100.100.1"/>
          <nat fake="192.168.0.0/24" public="100.200.0.0/24"/>
    </isp>
  </myself>
  <isps>
   <isp id="2" name="isp2.it">
     <controller name="cnt.isp2.it" ip="200.200.200.1"/>
     <nat public="200.150.0.0/24"/>
   </isp>
  </isps>
  <vpns>
   <vpn id="1" name="C1−C2−vpn">
    <isp id="1" name="isp1.it">
     <customer name="c1.isp1.it">
       <site name="s1.c1.isp1.it" timestamp="0">
        <datapath ip="100.0.0.123" in_port="1" out_port="2"/>
        <subnet private_network="10.0.0.0/16"/>
        <ns domain="ns1.c1.isp1.it" ip_address="10.0.0.3"/>
       </site>
     </customer>
    </isp>
   <isp id="2" name="isp2.it">
    <customer name="c2.isp2.it">
     <site name="s1.c2.isp2.it" timestamp="0">
       <subnet private_network="10.0.0.0/16"/>
       <ns domain="ns2.c2.isp2.it" ip_address="10.0.0.3"/>
      </site>
     </customer>
    </isp>
   </vpn>
  </vpns>
 </federation>
</federations>
```

The root of the XML tree is the element `<federations>`, containing all federated networks the ISP belongs to. Indeed, each partner can participate in more federated networks at the same time and each customer can join multiple federated VPNs belonging to different federated networks. We define the element `<federation>` as a

child of the root element and it contains information about the ISPs in the federated network. Each federated network has a globally unique name.

The <federation> element represents a federated network. In this example, ISP1 and ISP2 belong to a federated network called ISP1-ISP2-net. The <myself> element contains all information about the SDN-controller of the ISP's network in which this configuration is deployed. The most relevant information are the URL and the IP address if the SDN-controller (e.g., cnt.isp1.it and 100.100.100.1, respectively). Additionally, a fake_ip_subnet is used by SDN-controllers in scenarios where hosts sharing the same IP address need to exchange traffic, as reported in Figure 1. Indeed, IP addresses in the fake_ip_subnet are used as a temporary replacement of the actual destination IP address. The public_ip_subnet is used to translate private addresses into public ones, as in standard NAT implementation. Note that, by doing so, any forms of tunneling are avoided, resulting in the full Maximum Transmission Unit [MTU] being kept available. Note that if the ISP belongs to multiple federations, the <myself> element must be declared once for each federation. The <isps> element contains the list of all the other SDN-controllers belonging to the federated network. Even in this case, the relevant information is represented by the URL and the IP address of each controller. However, the element <nat> inside <isps> does not contain the fake_ip_subnet, since it is used from the SDN-controller of the ISP's network in which the end-host that starts the communication resides.

The <vpns> element contains both the list of all federated VPNs defined inside the federated network and the information about the customers belonging to each VPN. In this example, there is only a federated VPN, called C1-C2-vpn.

This configuration allows customers Customer1 and Customer2 to exchange traffic in the federated VPN. Customer1 is connected to the ISP1's network through an SDN-enabled switch whose IP address is 100.0.0.123; its traffic comes from SDN-enabled switch port number 1 (the SDN-enabled port connecting OF1 and CPE1) and goes out from SDN-enabled switch port number 2 (the SDN-enabled port connecting OF1 and PE1). Those information are provided by the <datapath> element. The subnet used by the customer is reported in the <subnet> element, whereas information about which is the local name server for that customer is found in the <ns> element. Customer2 is a remote customer (we recall that this is the piece of configuration deployed at the ISP1's SDN-controller). In this case, information about the SDN-enabled switch is not provided, since the ISP1's SDN-controller does not manage that switch. All the other information are taken as for Customer1.

We highlight that the information inside the configuration of each SDN-controller represents the minimum set of information to be specified to allow two (or more) customers to establish a federated VPN. Also, the configuration is simplified: it is now centralized and no more distributed over multiple devices, making the troubleshooting easier. As a consequence, the provisioning time of the federated services is reduced.

It is important to note the information enclosed inside <myself> and <isps> subtrees are static and manually configured by each ISP after reaching agreements with other ISPs on creating a federated network. While the content of the <vpns> element is dynamically generated. Indeed, federated VPNs' parameters are reported in the configuration exploiting several primitives allowing customers to easily access federated VPN.

Finally, all the information provided in the configuration is commonly known by a provider. Indeed, providers know the port (physical or virtual) at the PE which each customer is connected to, as well as which are the IP addresses used to interconnect each CPE to the PE.

5.2 | Primitives to join federated VPN services

Each ISP belonging to a federated network provides a set of primitives to its customers to join or leave the federated VPN service. Those primitives are received by the Customer Request Handler that performs, cooperating with the other components, checks to allow (or deny) a customer to join the service. The customers do not need to know the presence of the SDN controllers in the ISP's network. Indeed, such primitives can be provided to customers through easy-to-use user interfaces, like a web portal, allowing the usage of the federated VPN intuitive and flexible.

We define three main primitives: *Insert*, *Update*, and *Delete*. The *Insert* primitive is used by a customer to join a federated VPN. Using this primitive, the customer specifies several parameters. With the *Update* primitive, the customer can ask the SDN-controller to modify the parameters previously declared by the *Insert* primitive (e.g., by adding or removing information). Finally, using the *Delete* primitive, a customer exits the federated VPN. By using those primitives, a customer can specify policies to allow or deny other customers to exchange traffic with it. Such policies are verified by all the SDN-controllers in the federated network, and the result of such an operation is sent back to each customer. There are two types of checks. The first one prevents a *malicious* customer to establish federated VPNs with

other customers that are not interested in exchange traffic with it. The second type allows customers to receive traffic from and to send traffic to specific end systems. We deeply illustrate them.

Suppose that there are two customers, C1 and C2, and C1 is a malicious customer trying to set up a VPN with C2. Note that the VPN can be established if and only if both C1 and C2 issue the *Insert*. C1 sends an *Insert* to the Request Customer Handler of its provider. Once that component receives that *Insert*, it transfers suitable information to the SDN-controller so that it can contact the remote SDN-controller. On the other side, C2 should send an *Insert* to the Request Customer Handler of its provider to establish the VPN. Since C2 is not interested in setting up such a VPN, it does not send any *Insert* to that component. Hence, the SDN-controller of the C2's provider replies to the SDN-controller of C1's provider that the federated VPN cannot be set up because it did not receive any *Insert* from C2. This is the first type of check.

The second type is the following. Suppose that two large customers, C3 and C4, want to establish a VPN, but they only allow a small subset of their end systems to exchange traffic. Suppose that ES3 is the set of allowed end systems of C3, whereas ES4 is the set of allowed end systems of C4. After the VPN is established, the two customers start to exchange traffic. Both the SDN-controllers issue forwarding rules to enable the communication among the end systems belonging to ES3 and ES4, so that if one of the customers tries to send traffic to or receives traffic from one end system that does not fall in the declared sets, the communication is forbidden. So, the SDN-controllers are acting as firewalls.

We now describe the semantics of the primitives. *Insert* – By using this primitive, a customer asks its provider to join the federated VPN service. *Insert* takes as input four parameters as follows:

1 Customer that is the name of the customer.
2 Description is a set of parameters describing in detail information about the customer. In this set, a customer specifies its subnet, and (optionally) the IP address of a local name server. The information is translated into the element `<site>` contained in the subtree `<vpn>` of the configuration. Note that the information about the `<datapath>` element can be inferred by the SDN-controller exploiting proper data structures defined by the OpenFlow protocol (e.g., the `in_port` can be inferred by the PacketIn message) and relying on utility functions made available by the underlying SDN framework (e.g., the Ryu framework offers functions to get auxiliary information about the OpenFlow switches, like their IP address).
3 VPN is the ID of the federated VPN which the customer joins. By looking at this parameter, the SDN-controller can properly identify the `<vpn>` subtree to update. After choosing the federated VPN, the customer can also express a set of policies, declaring the set of customers inside the federated VPN which it is available to exchange traffic with.
4 Time contains information about how much time a customer wants to use the federated VPN service. After that time, the customer is no longer part of the service. This information is stored as the value of the parameter |timestamp| of the subtree `<site>`.

Referring to Figure 1, an example of the primitive *Insert* sent by Customer1 to the Customer Request Handler of the ISP1's SDN-controller is the following (we recall that Customer2 has to send a similar primitive to its provider):

```
Primitive: INSERT          Description:
 Customer:                 + Customer IP subnet: 10.0.0.0/16
 + Name: c1.isp1.it      + Local Name Server:
 + Site: s1.c1.isp1.it    + URL: ns.c1.isp1.it
 + IP Address: 10.0.0.3
 VPN: C1-C2-vpn          Time: 0
```

Upon receiving an *Insert*, the Customer Request Handler checks whether a federated VPN having the name reported in the *Insert* primitive corresponding to the key VPN already exists. If it is not the case, then it creates a new `<vpn>` element assigning to it an id automatically generated and the name reported in the primitive, namely C1-C2-vpn. Then, it sends the primitive to the Remote Controller Handler so that the latter forwards the information to the Remote Controller Handler of the remote provider.

Thus, the Customer Request Handler of the two providers can create the element `<isp id=''1'' name=''isp1.it''>` as child of the element `<vpn id=''1'' name=~C1-C2-vpn~>` for the Customer1 and Customer2 configurations, respectively. Based on the information associated with the keys Customer and Time reported in the primitive, new elements are added to the configuration, namely the element

<customer name=∼c1.isp1.it∼> and its child <site name=∼s1.c1.isp1.it∼ timestamp=''0''>. At this point, information about the <datapath> element is added to the configuration. Such information is inferred by inspecting the traffic coming from the SDN-enabled switch, in which the traffic generated by each customer is forced to pass through. Finally, customer's information is included in the configuration, by creating the elements <subnet private_network=∼10.0.0.0/16∼ /> exploiting the key Customer IP subnet of the element Description carried by the primitive and <ns domain=∼ns.c1.isp1.it∼ ip_address=∼10.0.0.3∼ /> exploiting the keys URL and IP Address of the subelement Local Name Server contained in the primitive. We point out that in case the federated VPN already exists, those steps are skipped since the information is already in the configuration.

After performing those checks, the Remote Controller Handler of the ISP1's SDN-controller sends the primitive received by Customer1 to the Remote Controller Handler of the ISP2's SDN-controller, as well as ISP2's does the same with the primitive received by Customer2. Once that message reaches the destination SDN-controller, it undertakes several operations over its configuration based on the content of the receipt message. We highlight that no human resources have been involved in this procedure and the service is provisioned without any delay potentially introduced by the federated nature of the network. By adopting our framework, the collaboration among providers, as well as to provision a service, is completely delegated to the Request Customer Handler, the SDN-controllers and the Remote Controller Handler.*Update* - By using this primitive, a customer can modify what declared in the *Insert.* Indeed, *Update* takes as input the same parameters of *Insert.* Also, this primitive is used to keep information about the customers updated.

Delete - By using this primitive, a customer can leave the federated VPN before the Time parameter declared in the *Insert* primitive expires. *Delete* takes three parameters as input: Customer, Description, and VPN. They have the same semantic described for the *Insert.* This information is propagated among the providers having customers in the declared VPN, so that the information in the federated network is consistent.

5.3 | Security considerations

We rely on the guidelines suggested by the OpenFlow standard[20] for the communication between SDN-controllers and SDN-switches. We also extend the adoption of Secure Socket Layer (SSL) or Transport Layer Security (TLS) technologies to the communication between customers and providers (e.g., exchange of primitives), as well as to the communication among SDN-controllers. Since those phases need to be publicly accessed, we made several design decisions, in such a way to not expose the SDN-controller on the Internet, preventing it to be a possible target.

The components which need to be accessed by the Internet are the Customer Request Handler and the Remote Controller Handler. Indeed, the first one must be reached by the customers that want to access the federated VPN service, whereas the latter one is used for the communication between the SDN-controllers. By doing so, the SDN-controller is not publicly exposed, hence the functionality of the SDN switches is not compromised by possible attacks. To further reduce the risk of cyber attacks, those components might run on different machines, exchanging information through private networks (e.g., standard point-to-point VPNs).

6 | A COMPLETE EXAMPLE

In this section, we provide a complete example of our framework, showing the whole interaction between two end-hosts, sharing *the same* IP address and belonging to two different customers connected to different ISPs' networks. We use a simplified version of the reference scenario in Figure 3 to explain the process. The two ISPs are connected through their border routers, namely, R1 and R2. We suppose that host H1, whose domain name is h1.c1.isp1.it and its IP address is 10.0.0.1, resides in Customer1, and host H2, whose domain name is h2.c2.isp2.it and its IP address is 10.0.0.1, resides in Customer2. Consequently, we call H1 source and H2 destination. We divide the example into three steps as follows:

1 federated VPN access request performed by the customers,
2 domain name resolution undertaken by the source, and
3 IP traffic sent by the source towards the destination.

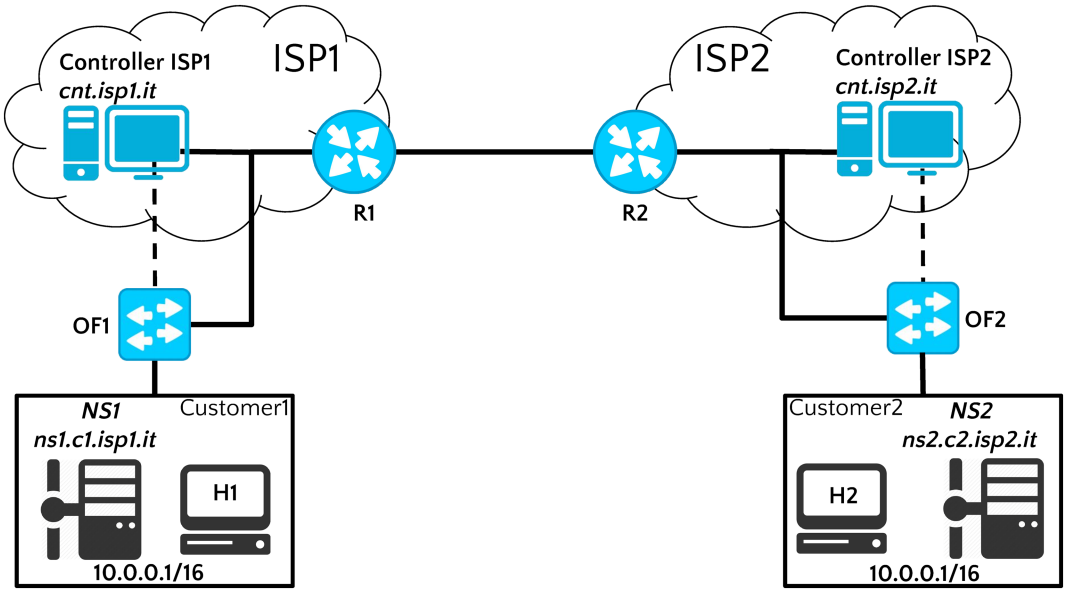


FIGURE 3 The simplified version of our reference scenario to show the interactions among two end-hosts

6.1 | Accessing the federated VPN service

As described in Section 5, the first operation carried out by customers that want to join the federated VPN service is to require the access to the service itself. It is done by sending an *Insert* to the Request Customer Handler. After this step, each SDN-controller owns a configuration, allowing it to provision the service.

6.2 | Name resolution process

Our framework includes an SDN-steered name resolution process that works as follows. When the source wants to exchange traffic with the destination starting a domain name, it sends a recursive DNS request message to its local name server. In our example, H1 starts a recursive DNS lookup by sending a DNS request to NS1, to obtain the H2's IP address (in this, example we concentrate on A resource record, but the interaction is analogous to any other type of resource records). We follow the DNS resolution process. It is easy to see that in our example such interaction cannot take place: if IP address overlap occurs, we cannot exclude that the local name servers NS1 and NS2 have exactly the same IP address. If this is the case, when NS1 tries to send a DNS request message to NS2, that packet will never exit Customer1's network.

We propose a mechanism relying on SDN to allow local name servers with IP addresses in the same subnet (potentially having exactly the same IP address) to exchange DNS traffic. By observing the DNS traffic, the SDN-controllers can manipulate it suitably, which is transparent for the end-users. Our proposal does not require to place any DNS daemon (e.g., BIND[21]) at the SDN-controller. This prevents to introduce any other configuration effort. We only need that the whole traffic generated by customers passes through the SDN-enabled switch, to be (possibly) forwarded to the SDN-controller. Our proposal is based on two main steps as follows:

1 We determine which is the IPS's network hosting the destination and acquire the IP address of the authoritative name server for the domain of the destination.
2 We resolve the destination's domain name.

The second step requires communication among the SDN-controllers (cnt.isp1.it and cnt.isp2.it in our reference example). In the rest of the section, we assume that H1 has domain name h1.c1.isp1.it whereas the domain name associated to H2 is h2.c2.isp2.it.

Determining the IP Address of a Name Server – Since the SDN-controller has to interact with local name servers placed in private networks with private IP addresses, it needs two important information: the first one is the customer's

network in which that name server resides, and the second is the IP address of that name server. To achieve this, we propose two different approaches, and we exploit the configuration described in Section 5. Indeed, the element `<site>` contains the whole needed information. The difference between these two approaches resides in the fact that in the first one, the IP address of the customer's local name server is in the configuration of the SDN-controller, whereas in the second, it is not, as described in Section 5. We call these scenarios *Full* and *Partial configuration*, respectively. In this paper, we focus on the *Partial* scenario, since the *Full* is described in Mostafaei et al.[7]

The *Partial configuration* scenario is more interesting to address, even if the interaction among network devices and machines (e.g., name servers and SDN-controllers) are more complex, as shown in Figure 4). In Section 5, we argued that the specification of the IP address of the local name servers is optional. Such a choice is motivated by two reasons. First, it simplifies the configuration. Second, since the local name server typically has a private IP address, a customer could change it without notifying the ISP, leading to possible misconfiguration problems among the SDN-controllers. Hence, we define a technique to retrieve this information, avoiding such an issue.

At the beginning, H1 sends a recursive DNS request message to NS1, which starts the iterative part of the name resolution process by querying the root name server. Upon receiving the DNS answer message coming from the root name server, OF1 forwards this packet to ISP1's SDN-controller. Since we are assuming that the authoritative name servers of all the customers are private, the SDN-controller inspects the content of the DNS answer, aiming at verifying whether it contains some information on the authoritative name server for the destination's domain. If it is not the case, the SDN-controller sends that packet to OF1 by instructing that device to forward the DNS answer to NS1. This process carries on until a DNS answer containing information about the IP address of NS2 (reported as a *glue record* of an NS DNS record) reaches the SDN-enabled switch, allowing ISP1's SDN-controller to understand which is the IP address of NS2. In this case, the DNS answer message is not forwarded to NS1, preventing it to exchange traffic with a name server potentially having the same IP address, but residing in a different network (we recall that customers in different IPS's networks might share the same IP subnet.)

There are differences between the two approaches we presented. First, in the *Full configuration* approach, there are no other name servers involved in the communication except NS1. Also, the SDN-controller looks at the DNS request messages produced by NS1. Second, in the *Partial configuration* approach, other name servers are involved in the communication and the SDN-controller inspects the DNS answer messages sent by those name servers.

Resolving Domain Names in Presence of IP Addresses Overlap – Once ISP1's SDN-controller acquires the IP address of NS2, it issues a DNS request message Q directed to that name server based on the DNS request message produced by

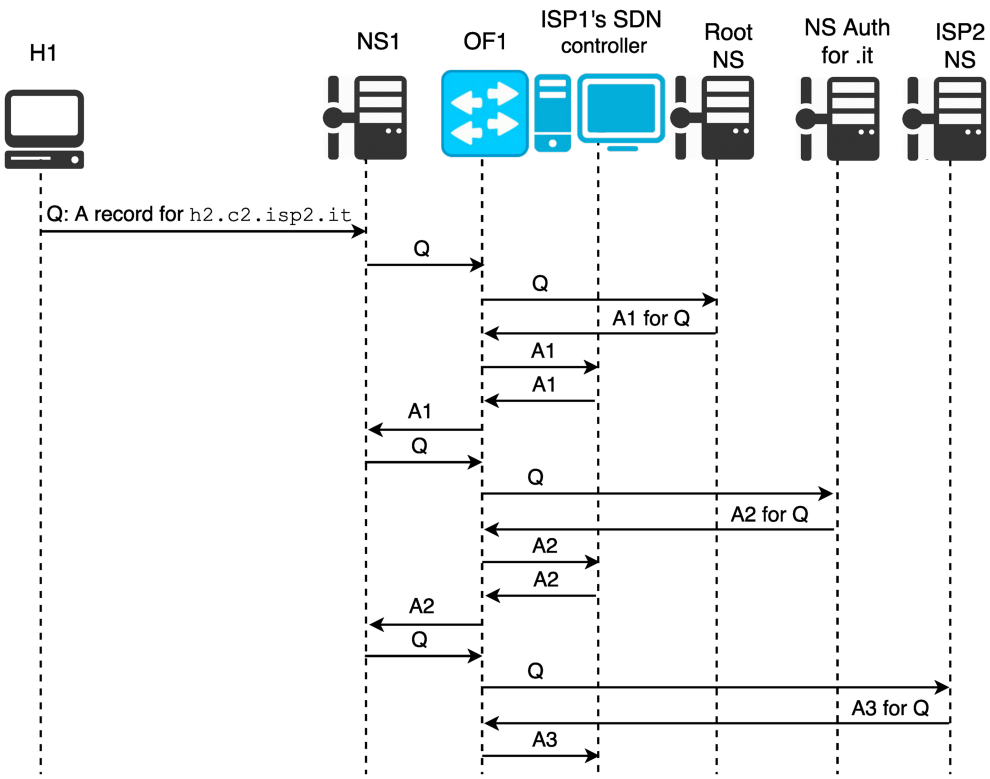


FIGURE 4 Interaction among OpenFlow switch, name servers, and software-defined network (SDN)–controller in the case of *Partial configuration* scenario

the source and it sends that DNS message to ISP2's SDN-controller by using a dedicated communication channel. This is possible because the public IP address of each SDN-controller in the federated network is part of the configuration. Upon receiving Q, ISP2's SDN-controller forwards it to the correct name server (this information is part of the configuration), which replies with H2's IP address.

After receiving the DNS answer message issued by NS2, ISP2's SDN-controller sends that DNS message to ISP1's SDN-controller. Consider that, before forwarding the DNS answer message to NS1, ISP1's SDN-controller must check whether the destination host has an IP address that is in the same subnet of the source. Such a check is mandatory since we allow the communication with a potentially fully overlap of IP addresses and in this case H1 and H2 exactly share the IP address 10.0.0.1. Hence, ISP1's SDN-controller has to change the IP address contained in the DNS answer message, preventing H1 to send traffic inside its network, or to itself. To do that, each controller owns a set of fake IP addresses to use for this purpose, which is declared in the configuration as shown in Section 5. ISP1's SDN-controller picks an IP address from the fake set and replaces the original H2's IP address with the fake one, keeping this association in suitable internal data structures. At the same time, it sends to the SDN-enabled switch a set of rules to forward the traffic according to this IP address replacement action. In this way, H1 is not aware of the fact that it is sending traffic to a destination with an IP address in the same subnet. It is interesting to note that, by using this technique, also NS1 and NS2 can share the same IP address, since the interaction between these two name servers is mediated by the SDN-controllers. At this point, H2's domain name has been resolved and H1 is able to send traffic.

6.3 | Sending IP traffic to the destination

Once the source acquires the IP address of the destination, it starts to send traffic. Since communication is being established between end-hosts with private IP addresses, translation strategies are needed. We now describe the Network Address and Port Translation (NAPT) strategies that we apply to support communications between hosts in different customer sites within a federated network. These strategies are used to alter the IP addresses (and, possibly, the TCP/UDP ports) of exchanged packets in such a way that traffic between hosts with private addresses can be routed on a public IP network. Address translations are performed by SDN-enabled switches according to packet manipulation rules in their SDN flow tables. As soon as the source emits a packet to establish a TCP/UDP connection towards the destination, the SDN-controllers install on OF1 and OF2 suitable translation flow entries to support the communication between the hosts. We describe three strategies as follows:

1 Client Port Preservation (CPP),
2 Client Announces Port Selection (CAPS), and
3 Lazy Address and Port Selection (LAPS).

More detailed discussions regarding these strategies can be found in di Lallo.[6]

7 | TAKEAWAY

By relying on the SDN architecture and providing both a configuration language and a set of primitives, we built a framework that can make the process of creating federated networks and subscribing to a federated VPN service simple and straightforward. Our framework aims at reducing the effort of providing federated VPN services, as well as simplifying the creation of a federated network. Also, the traffic isolation typically provided by standard VPNs is still guaranteed to the extent that a provider adopts MPLS in its backbone. Indeed, our SDN-switches are placed outside the MPLS domain since they interconnect the CPE with the PE. Since the SDN-controllers of our framework instructs the SDN-switches to issue standard IP packets, they can be encapsulated into MPLS packets once they reach the PE routers, as in standard VPN fashion. Recalling the main challenges of Section 3, we now summarize how we solve those issues.

Management Problems – Avoiding additional interconnection points, each federated provider is still able to manage its network as it prefers, without any constraints in terms of collaboration with other federated partners. Our choice to delegate to the SDN-controllers the task of handling the federation and every federated service issued relying on such a network allows us to be independent of any kind of collaboration, having benefits for many operations, for example, monitoring.

Technological Differences Problems – As our framework does not require any ad hoc place to interconnect, except the IXP that has a well-defined interface for exchanging information (e.g. BGP protocol), each provider can use any routing protocol or transmission technology without coordinating with other federated partners. Also, the SDN architecture plays a key role. Indeed, being able to take a centralized decision and sharing them among SDN-controllers exploiting a dedicated communication channel allows us to easily reach interoperability. Also, the choice of using NAT strategies to realize VPNs allows the SDN-enabled switches to issue IP packets, making the traffic forwarding completely independent from specific data plane protocols used in the backbone (e.g., MPLS).

Unified User View – We argue that our configuration language and primitives represent a solid way to provide a standard interface to access the federated VPN service. Furthermore, delegating the coordination activities to the SDN-controllers reduces the amount of time needed for provisioning the service. By relying on our framework, we argue that users perceive the federated VPN service as issued by a single provider, keeping hidden the collaboration among ISPs.

Drawbacks of our framework – We recognize a major drawback in our framework. Although we do not impose any constraints on the protocol adopted by each provider inside the backbone, we assume that each federated ISP has (at least) an SDN controller plus an SDN-enabled switch collecting the traffic of its federated customers. On one hand, such assumptions do not impact the network architecture of the ISP; on the other hand, the SDN equipment might represent a cost for the provider. We argue that a reasonable trade-off for choosing our solution is the size of the federation (in terms of both ISPs and customers) and the flexibility degree that the federated partners want to achieve.

8 | EVALUATION

To validate the effectiveness of our framework, we implemented a prototype SDN-controller by relying on the Ryu framework[22] and OpenFlow 1.3.[20] We focus our evaluation activity on both control and data plane, analyzing how many messages (OpenFlow and DNS) are exchanged in the federated network and which is the impact on each OpenFlow-enabled switch of setting up a federated network. We also measure the required time for the controllers to handle DNS messages. We do not evaluate monitoring and fast setup aspects. Indeed, we do not provide any contributions in terms of how to monitor the network: we just claim that we propose a solution for monitoring issues described in Fischer et al.[17] On the other hand, fast setup is achieved by the SDN-controller, in contrast to Géant European Project.[2]

We run our experiments on SDNetkit,[23] an SDN-enabled enhancement of Netkit,[24] a widely used network emulator. Within SDNetkit, we used BIND[21] to implement name server functionalities and Open vSwitch[25] to implement OpenFlow devices. We run our SDN-controller on topologies reflecting the scenario in Section 4. Our implementation is composed of three main components as follows:

1 *Primitive Handler*, to handle the primitives,
2 *DNS Handler*, to handle DNS messages, and
3 *Routing Handler*, to compute the routing.

In our experiments, we focus on considering three different coordinates as follows:

1 number of ISPs in the federated network,
2 number of customers per ISP, and
3 number of VPNs in the federated network.

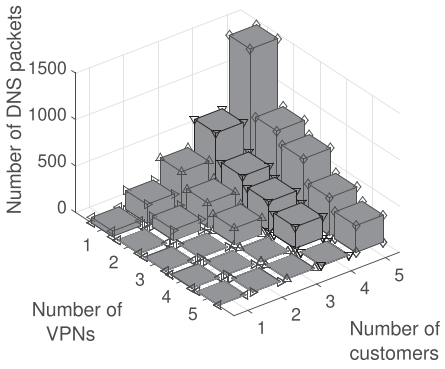
We run several simulations on different topologies, which are built according to the following criteria. First, we build a federated network consisting of two ISPs. In such a federated network, we connect to each ISP a number of customers varying in the range [1,5]. Then, we set up a number of VPNs varying in the range [1,5]. Also, we assume that a customer can be part of a single VPN. Hence, the number of VPNs has to be determined according to the number of customers per ISP (e.g., with a single customer per ISP, we cannot create more than one VPNs, and with two customers, we can set up at most two VPNs). Each customer consists of a host and a local name server, authority for that host. Second, we did the same in a federated network consisting of three ISPs. During each simulation, we perform DNS resolution and standard ping among any pair of hosts belonging to the same VPN to issue the maximum number of DNS

queries. Therefore, each VPN client generates DNS and ICMP traffic to issue the service. We now briefly describe which is the impact of our SDN-controller on both control and data plane.

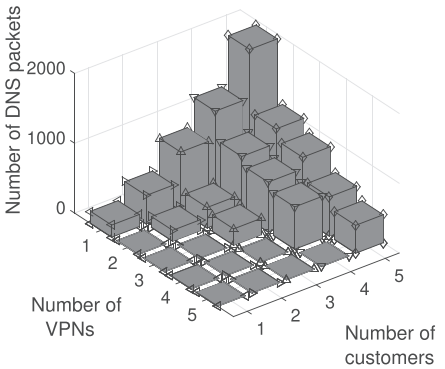
8.1 | Control plane impact

To evaluate the impact of our implementation on the control plane, we measure the amount of DNS and OpenFlow packets exchanged in the network to allow a source to exchange traffic with a destination. We count the number of DNS and OpenFlow packets on each interface of each OpenFlow-enabled switch in the network, namely OF1, OF2, and OF3. With respect to the OpenFlow packets, we point out that we only consider PacketIn, PacketOut, and FlowModification messages, since our implementation affects those types of messages (e.g., OpenFlow handshake and keepalive messages are independent by any controller implementations). Also, we consider both *Full* and *Partial configuration* scenarios, as discussed in Section 6.

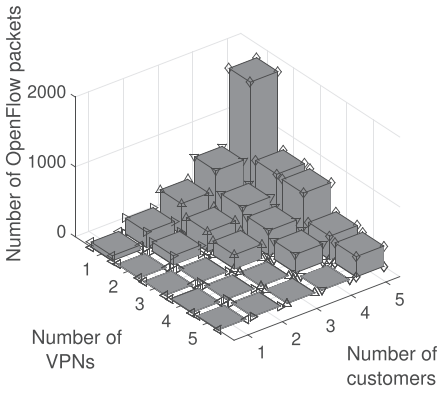
Figure 5 shows the total number of DNS (Figure 5A,B) and OpenFlow (Figure 5C,D) messages exchanged in a federated network with two ISPs. We observe that the number of messages (both DNS and OpenFlow) grows concerning the number of customers connected to each ISP, while it decreases when the number of VPNs increases. We ascribe this behavior to the fact that the number of messages strictly depends on the number of queries in the network. Indeed, if more queries are performed by many sources, more DNS packets are issued to get the IP address of each desired destination. Regardless of which scenario (*Full* or *Partial configuration*) we are considering, our SDN-controller has to interact with such DNS packets, implying an increasing number of OpenFlow messages. Those considerations are validated by looking at Figure 8. Indeed, we observe that the number of queries grows with respect to the number of customers (Figure 8B) while decreases with respect to the number of VPNs (Figure 8A). This is because increasing the number of



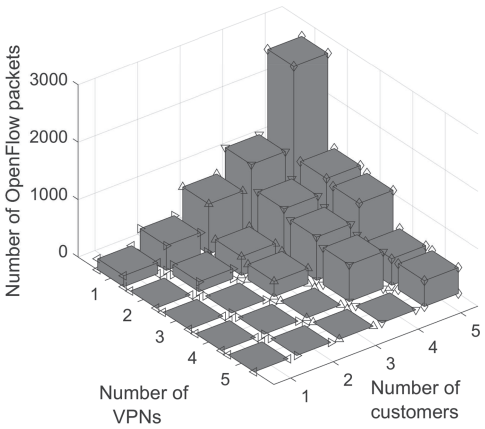
(A) Number of DNS packets exchange in the *Full configuration* scenario.



(B) Number of DNS packets exchange in the *Partial configuration* scenario.



(C) Number of OpenFlow packets exchange in the *Full configuration* scenario.



(D) Number of OpenFlow packets exchange in the *Partial configuration* scenario.

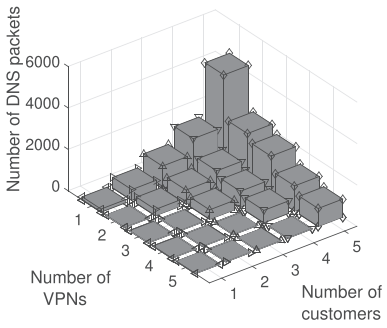
FIGURE 5 Control plane impact in a federated network consisting of two Internet service providers (ISPs)

VPNs means reducing the number of destinations per VPN, reducing the total number of queries, and—consequently—the total number of messages.

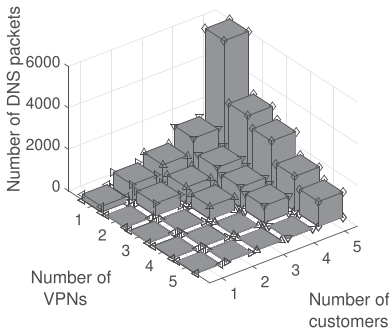
The same considerations are valid in the case of a federated network consisting of three ISPs. The total number of DNS and OpenFlow messages exchanged in the network is depicted in Figure 6, that is, Figure 6A,B for DNS and Figure 6C,D for OpenFlow packets. Of course, in this case, we observe a greater number of messages (almost 2,000). Such an increase with respect to Figure 5 has to be ascribed to the fact that adding a new ISP in the federated networks results in increasing the number of customers belonging to the federated network itself and, consequently, the total number of destinations. Since we already discussed how the number of destinations affects the number of queries and which is their impact on the total number of messages exchanged in the network, those results are perfectly aligned with what we expect, also considering results shown in Figure 8. It is worth to observe that the scalability of our framework is the same as the DNS service. Indeed, we do not add any DNS messages to the name resolution process, as in the *Partial configuration* scenario. Rather, in the *Full configuration* scenario, we prevent several DNS messages (e.g., DNS messages directed to the root name servers) to be forwarded in the network. On one hand, having the IP address of a local name server in the configuration saves a lot of DNS and OpenFlow messages. On the other hand, retrieving that information from the DNS resolution process is a plus from a configuration point of view (see Section 6), but it represents a cost considering the number of exchanged messages, that is—in any cases—the same of any DNS service.

Figure 7 depicts the controller processing time to handle DNS queries for both *Full configuration* and *Partial configuration* scenarios. The results show that increasing the number of DNS queries results in increasing the controller processing time. We claim that this behavior is due to the number of DNS messages the controller should handle. Note that the intercommunication delay between the controllers is ignored in this measurement.

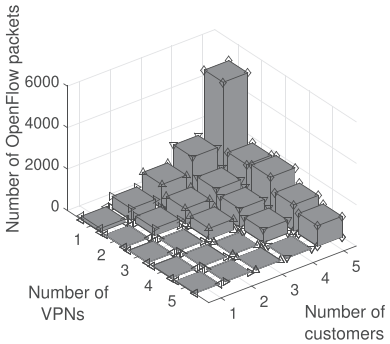
We also analyzed the correlation among the number of queries and the number of DNS messages. Pearson's correlation coefficient was used for this measurement. The correlation between the number of queries and the number of DNS messages in *Full* configuration scenario is 0.99, while for the *Partial* configuration scenario, this coefficient has a value of 0.90. This correlation for OpenFlow messages of *Full* configuration scenario is 0.99 and for *Partial* one is 0.90.



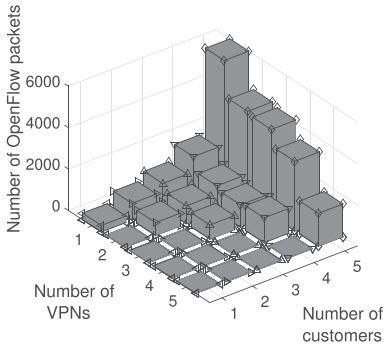
(A) Number of DNS packets exchange in the *Full configuration* scenario.



(B) Number of DNS packets exchange in the *Partial configuration* scenario.



(C) Number of OpenFlow packets exchange in the *Full configuration* scenario.



(D) Number of OpenFlow packets exchange in the *Partial configuration* scenario.

FIGURE 6 Control plane impact in a federated network consisting of three Internet service providers (ISPs)

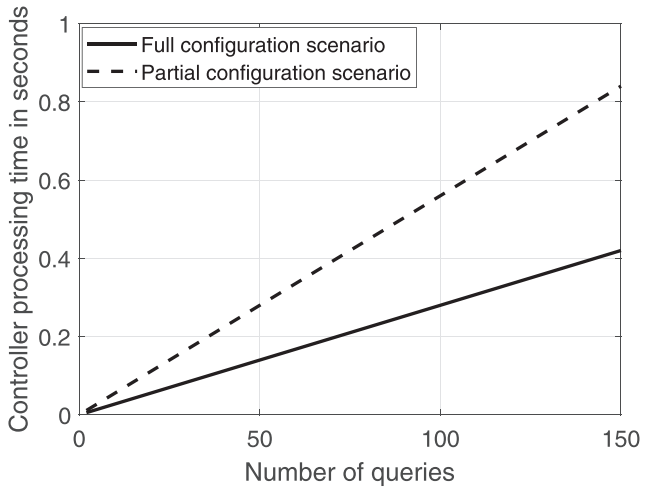


FIGURE 7 Controller processing time with varying the number of domain name system (DNS) queries

8.2 | Data plane impact

By default, our prototype SDN-controller installs two rules: one for ARP packets and another one for DNS packets. Indeed, all traffic belonging to those classes needs to be processed by the SDN-controller to allow the SDN-enabled switches to exchange traffic with the neighbor routers (ARP packets) and to allow the *DNS Handler* to properly steer the name resolution process (DNS packets).

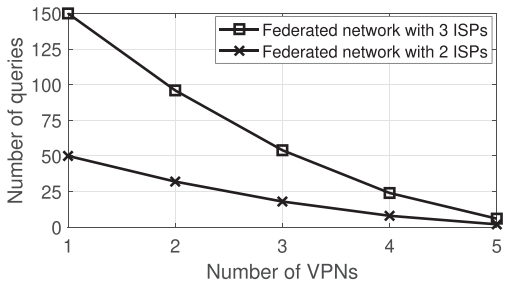
We also provide an analytical model to count the number of required rules to install on the SDN-enabled switches. Assume that C is the number of customers per ISP. First, we count the number of required rules for *Full* configuration scenario. As foregone, we need one rule to handle ARP packets. To handle DNS packets, we need $2\times C$ rules and for handling IP packets, the controller needs to install $4\times C+2$ rules. The number of OpenFlow rules in the *Full* configuration scenario (OF_{FC}) is

$$OF_{FC} = 1 + (2 \times C) + (4 \times C + 2) = 6 \times C + 3. \tag{1}$$

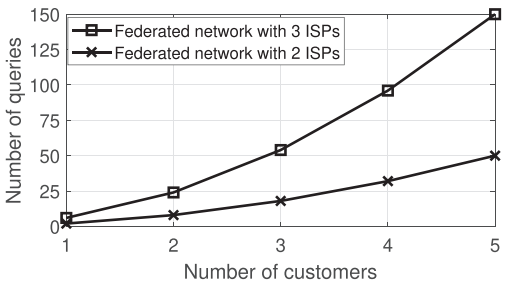
While for the *Partial* configuration scenario, we have the same number of rules for ARP and $2\times C+2$ rules for handling DNS packets. Finally, we need $4\times C+2$ rules for IP packets. The overall number of OpenFlow rules for the *Partial* configuration scenario (OF_{PC}) can be computed as follows:

$$OF_{PC} = 1 + (2 \times C + 2) + (4 \times C + 2) = 6 \times C + 5. \tag{2}$$

Primitive Handler – This component is in charge of the task of processing primitives sent by customers. After analyzing the content of each primitive, it writes proper information in the SDN-controller configuration, as shown in Section 5. Basically, this component does not have any impact on the data plane.



(A) Federated network with five customers per ISP.



(B) Federated network with a single VPN.

FIGURE 8 Number of domain name system (DNS) queries in the federated network

DNS Handler – This component is in charge of steering the name resolution process. After resolving a domain name, the *DNS Handler* installs a rule to send IP traffic toward the destination to the controller. This rule is needed since it triggers the *Routing Handler*, whose task is to act as shown in Section 6. Note that for each possible destination, one rule is needed. This shows the number of rules is linear concerning the number of destinations in the federated VPN.

Routing Handler – Routes in the networks are computed by this component. In particular, for each pair of end-hosts, it installs two rules. The first handles the traffic directed to the destination, whereas the second allows the traffic to come back from the destination to the source. Thus, the number of rules is quadratic with respect to all possible combinations among end-hosts. Since this situation is not so common in computer networks (destinations are less than sources), the number of rules is linearly proportionate to the number of pair ⟨source,destination⟩. Optimizations might be carried out attempting to reduce that amount of rules.

Finally, the evaluation shows that our system can manage federated networks with a small number of VPNs. Of course, having a single SDN-controller might be a limitation under different point of views (e.g., scalability and robustness). By increasing the number of SDN-controllers per ISP, our framework can handle a growing VPN demand (Figure 8).

9 | CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework enabling the fast creation of federated networks. We show that today's federated network architecture can be simplified by adopting SDN. Also, we demonstrate that our framework does not impact any existing configuration, as well as any existing architecture. It does not require architectural changes, except for the adoption of SDN-controllers, that is a reasonable assumption.

As research perspectives, we intend to go deeply in improving our current implementation, providing a more complete software enabling federations to use it to issue federated services. We believe that in a world where IPv4 address exhaustion is being a problem—also due to the slow IPv6 adoption[26,27]—our solution represents a valid alternative that allows ISPs to provide value-added services to their customers, without introducing any scalability issues. We would like to extend our framework to support data-plane programmability paradigms such as P4.[28]

ACKNOWLEDGEMENT

This work was partially funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (refs. 01IS18025A and 01IS18037A).

ORCID

Habib Mostafaei https://orcid.org/0000-0001-8282-1571

REFERENCES

1. Goiri I, Guitart J, Torres J. Characterizing cloud federation for enhancing providers' profit. In: 2010 IEEE 3rd International Conference on Cloud Computing. Miami, FL, USA; 2010:123-130.
2. Géant European Project. http://www.geant.net; 2020.
3. Beacon European Project. http://www.beacon-project.eu/; 2017.
4. Rosen E, Rekhter Y. BGP/MPLS IP Virtual Private Networks (VPNs). RFC 4364. The Internet Engineering Task Force (IETF); 2006.
5. Géant European Project VPN services. https://www.geant.org/Services/Connectivity_and_network/Pages/VPN_Services.aspx; 2020.
6. di Lallo R, Lospoto G, Rimondini M, Battista GD. Supporting end-to-end connectivity in federated networks using SDN. In: Proc. IEEE/-IFIP Network Operations and Management Symposium (NOMS 2016) Erol-Kantarci M, Jennings B, Reiser H, eds.; 2016; Istanbul, Turkey:759-762.
7. Mostafaei H, Lospoto G, Brandimartey A, Lallo RD, Rimondini M, Battista GD. Sdns: exploiting SDN and the DNS to exchange traffic in a federated network. In: 2017 IEEE Conference on Network Softwarization (NetSoft); 2017July; Bologna, Italy:1-5.
8. Villegas D, Bobroff N, Rodero I, et al. Cloud federation in a layered service model. *J Comput Syst Sci*. 2012;78(5):1330-1344. JCSS Special Issue: Cloud Computing 2011.
9. Koerner M, Gaul C, Kao O. Evaluation of a cloud federation approach based on software defined networking. In: 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops); 2015; Clearwater Beach, FL, USA:657-664.
10. Darzanos G, Koutsopoulos I, Stamoulis GD. Cloud federations: economics, games and benefits. *IEEE/ACM Trans Netw*. 2019;27(5):2111-2124.

11. Ferrer AJ, Hernández F, Tordsson J, et al. Optimis: a holistic approach to cloud service provisioning. *Future Gener Comput Syst*. 2012;28 (1):66-77.
12. Calheiros RN, Ranjan R, Beloglazov A, De Rose CésarAF, Buyya R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Pract Exper*. 2011;41(1):23-50.
13. Shen N, Farinacci D. LISP Multi-Provider VPN Use-Cases; 2014.
14. Farinacci D, Fuller V, Meyer D, Lewis D. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Internet Engineering Task Force; 2013.
15. Andrade I, de Alfonso C, Blanquer I. Defining and testing virtual federated networks across multiple cloud sites. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC); 2017; Valencia, Spain:107-112.
16. Moreno-Vozmediano R, Montero RS, Huedo E, Llorente IM. Implementation and provisioning of federated networks in hybrid clouds. *J Grid Comput*. 2017;15(2):141-160.
17. Fischer L, Belter B, Przywecki M, et al. Building federated research networks in europe. In: Terena Networking Conference. Prague, Czech Republic; 2010:1-13.
18. Przywecki M, Golub I, Paric D, et al. Federated pop: a successful real-world collaboration. https://geant3.archive.geant.org/Media_Centre/Media_Library/Media%%20Library/TNC2012%20Federated%20PoP%20full%20paper.pdf; 2012.
19. Kompella K, Rekhter Y. Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling. RFC 4761 (Proposed Standard), Internet Engineering Task Force; 2007.
20. Open Networking Foundation. Openflow switch specification, version 1.3.4; 2014.
21. Bind name server software. https://www.isc.org/downloads/bind/; 2020.
22. Ryu: component-based software defined networking framework. https://osrg.github.io/ryu/; 2020.
23. Mostafaei H, Lospoto G, di Lallo R, Rimondini M, Di Battista G. Sdnetkit: a testbed for experimenting SDN in multi-domain networks. In: 2017 IEEE Conference on Network Softwarization (NetSoft); 2017July; Bologna, Italy:1-6.
24. Netkit: the poor man's system to experiment computer networking. http://netkit.org; 2020.
25. Openvswitch. http://openvswitch.org/; 2020.
26. Google ipv6 statistics. https://www.google.com/intl/en/ipv6/statistics.html; 2020.
27. Ripe ipv6 statistics. https://stats.labs.apnic.net/ipv6/; 2020.
28. Bosshart P, Daly D, Gibb G, et al. P4: programming protocol-independent packet processors. *ACM SIGCOMM Comput Commun Rev*. 2014;44(3):87-95.

AUTHOR BIOGRAPHIES

Habib Mostafaei currently is a postdoctoral researcher at the Internet Network Architectures (INET) of Technische Universität Berlin. He received the PhD in Computer Science and Engineering from the Roma Tre University in 2019. Prior to the PhD education, he worked as a full-time faculty member at the Computer Engineering Department of Azad University (2009-2015). Currently, his main research fields include software-defined networking (SDN), inter-domain routing, and network measurements.

Gabriele Lospoto received his PhD in 2016 at Roma Tre University with the thesis "Improving flexibility, provisioning, and manageability in intra-domain networks," and he was a PostDoc research fellow in the Computer Networks group at the same university for the next 2 years. His research was focused on how to exploit software-defined networking in order to simplify service provisioning in production networks, especially in specific contexts such as federated networks. His research topics also included the study of formal methodologies to compare services implemented with different technologies. In 2018, he moved on industry, joining the Network Development and Engineering team at Unidata S.p.A., an Italian Internet Service Provider. He is currently involved in the design of the Unidata's network architecture and in the deployment of SD-WAN solutions.

Roberto di Lallo received a PhD in Computer Science and Engineering in 2018 at the Roma Tre University, defending the thesis "Internet eXchange Points: Current Challenges and New Opportunities". His research was focused on the impact of Internet eXchange Point on the Internet and on the practical applicability of SDN research. In 2018, he moved to industry, joining the Consortium GARR Distributed Cloud and Storage group, where he worked on Compute and Container Platform with a particular focus on the Federated Cloud model. Currently is a Cloud DevOps Architect at Accenture where designs cloud architecture solutions to let the client business growing adopting a DevOps approach.

Massimo Rimondini received a PhD in Computer Science and Engineering in 2007 at the Roma Tre University, defending the thesis "*Interdomain Routing Policies in the Internet: Inference and Analysis.*" Since then, he conducted academic research in various fields related to the Internet, the main ones being analysis of the stability of routing

protocols, traffic engineering, analysis of routing events, inference of network topologies and commercial relationships, network virtualization, and, recently, software-defined networking. In 2016, he chose to further enhance his experience in the field of computer networks by joining the engineering team of an Internet Service Provider (Unidata S.p.A.), which he currently leads and where he designs novel network architectures and services.

Giuseppe Di Battista is a Professor of Computer Science. He received the PhD in Computer Science from the University of Rome "La Sapienza" and is currently a faculty member in the Department of Engineering at the Roma Tre University. His current research interests include Computer networks, Graph Drawing, and Information Visualization. He has published more than 200 papers in the above areas and has given several invited lectures worldwide. His research has been funded by the Italian National Research Council, by the EU, and by several industrial sponsors. He has been national and/or local coordinator of many Projects of Relevant Italian National Interest (PRIN) of the MIUR.

How to cite this article: Mostafaei H, Lospoto G, Di Lallo R, Rimondini M, Di Battista G. A framework for multi-provider virtual private networks in software-defined federated networks. *Int J Network Mgmt*. 2020;30: e2116. https://doi.org/10.1002/nem.2116