Rethinking Virtual Private Networks in the Software-Defined Era

Gabriele Lospoto*, Massimo Rimondini*, Benedetto Gabriele Vignoli† and Giuseppe Di Battista*
Roma Tre University
*{lospoto,rimondin,gdb}@dia.uniroma3.it †gabriele.vignoli@yahoo.it

***Abstract*—Multi Protocol Label Switching (MPLS) Virtual Private Networks (VPNs) have seen an unparalleled increasing adoption in the last decade. Although their flexibility as transport technology and their effectiveness for traffic engineering are well recognized, VPNs are difficult to set up and manage, due to the complexity of configurations, to the number of involved protocols, and to the limited control and predictability of network behaviors. On the other hand, Software-Defined Networking (SDN) is a consolidated, yet still emerging paradigm by which the control plane logic of a network device is implemented by an arbitrarily programmed software that runs outside the device itself.**

We conjugate the effectiveness of traditional VPNs with the programmability of SDN, proposing a novel and improved realization of MPLS VPNs based on SDN. With our approach, provisioning and setup of VPNs are accomplished by using a simple and flexible configuration language. Management and troubleshooting are facilitated because only a minimal set of technologies (notably, just MPLS) is retained. Control and predictability of network behaviors are enhanced by the centralized coordination enforced by the SDN controller. Besides illustrating our proposed approach and specifying the configuration language, we describe a prototype implementation of a controller and the outcome of tests we conducted in several configuration scenarios.

I. Introduction

Virtual Private Network (VPN) services based on Multi Protocol Label Switching (MPLS) play a key role in the business of Internet Service Providers (ISPs). They are offered to a wide population of customers to interconnect their geographically distributed sites and continue to have an increasing market share since the late 90s [1], [2]. Despite their maturity, they still have several critical aspects, implied by the plethora of networking technologies and protocols involved in their operation (MP-BGP, OSPF, LDP, etc.). First, this makes VPNs difficult to provision, configure, manage, and troubleshoot. Second, it is hard to cast a predictable behavior from the complex interactions of these technologies, especially considering that configurations are distributed on several devices. Third, they offer limited or inconvenient methods to control routing and traffic engineering decisions. In contrast, ISPs seek for simplicity of provisioning and configuration, trouble-free management, flexibility in accommodating increasingly complex customer requirements, controllability, and predictability.

Independently, a centralized routing approach called Software Defined Networking (SDN) [3] has been gathering increasing attention over the past years. SDN proposes a separation of the control plane from the data plane: a network device, called *datapath*, only has in charge the forwarding of the traffic, whereas all the control plane logic is managed by a piece of software, called *controller*, that runs on dedicate hardware. The controller sends to each datapath a set of rules, called *flow entries*, that are installed into local *flow tables*, determining the forwarding behavior of that datapath. When a datapath receives a packet that matches the conditions specified in a flow entry, it undertakes the actions specified in that entry, which may include forwarding, modification of packet headers, etc. If no flow entries are matched, the datapath forwards the packet to the controller, asking for the action to undertake. Nowadays, the most widely adopted specification of the SDN approach is OpenFlow [4], which includes a protocol for the communication between controller and datapaths.

This paper shows that SDN can give a fundamental contribution to tackle the mentioned shortcomings of MPLS VPNs. In principle, VPNs can be realized with SDN by implementing a controller that replicates the configuration interface and operation of the technologies involved in traditional VPNs. However, this approach has very limited advantages (perhaps only the centralized configuration), and inherits the drawbacks of the involved technologies. An alternative approach offering improved flexibility is to specify the configuration in the form of code fragments within the controller, for example by exploiting an API designed for the setup of VPNs: virtually arbitrary network behaviors and routing policies could be enforced in this way, but the highly increased configuration complexity may be barely tolerable by network administrators.

We leverage SDN to propose a completely different approach that preserves the current features of MPLS VPNs while bringing about a smooth provisioning, setup, and management experience for ISPs. We achieve this by introducing a simple and flexible language for a convenient centralized specification of VPN configurations. This specification is then directly translated into flow entries to be installed on datapaths. To the benefit of manageability, we drop most of the technologies currently involved in MPLS VPNs, retaining only those that are strictly required and reducing their usage to a bare minimum (we even abandon label switching). Controllability and predictability are also enhanced by the datapath coordination action enforced by the SDN controller. We have implemented a prototype of an OpenFlow SDN controller based on our approach and used it to perform extensive experiments in combination with Open vSwitch, one of the leading implementations of OpenFlow-compliant datapaths that is also used on commercial hardware such as Corsa Technology Inc. and Quanta Computer Inc. devices, as well as some Intel-based platforms.

The rest of the paper is organized as follows. In Section II we review contributions related with MPLS VPNs and SDN. In Section III we describe our approach compared with traditional

MPLS VPNs, and argue on its practical applicability. In Section IV we describe our VPN configuration language and how a VPN specification is translated into flow entries. In Section V we present the outcome of our experiments. Last, concluding remarks and lines for future research are in Section VI.

II. Related Work

MPLS VPNs are a long-established and extensively documented technology (for a comprehensive introduction, see [5]). However, their complexity of setup is a matter of fact (see, e.g., [6]) and is also confirmed by the existence of several ad-hoc management and monitoring tools such as [7]–[10]. Configuration automation solutions like, e.g., [11], can substantially cut the probability of errors but do not solve the underlying configuration complexity. In addition, current technologies for MPLS VPNs may exhibit inconsistent behaviors: for example, certain visibility constraints between VPNs may be difficult or even impossible to enforce (see [12]).

On the other hand, SDN is an approach supported by many vendors as well as by technical and research communities, including the Open Networking Foundation, the Open Networking Research Center (ONRC), the IETF, and the IEEE. Its most widely adopted realization is OpenFlow [4], whose most complete specification is 1.5.0 [13].

Despite the converging interest on SDN, there are really few attempts to import its flexibility in the implementation of VPN services. The most active research line is led by the ONRC [14]: in [15] they share our concerns on the inconveniences of the technologies currently used for VPNs and they implement an SDN-based MPLS traffic engineering solution that eliminates the need for intra-domain routing protocols. In [16] they demonstrate the feasibility of reimplementing several features of MPLS VPNs using an SDN controller. Similar results are presented in [17]. However, these contributions tend to focus more on traffic engineering aspects, are rather tied to the existing way of realizing VPNs, do not tackle the configuration difficulty problem, and lack a rigorous specification of SDN-related technical aspects. Although this research line is still alive, we are not aware of any progress on these contributions or deployments in operational networks.

We exceed existing contributions by rethinking VPNs with manageability as a driving principle: our approach simplifies the configuration by introducing a language that addresses only domain-specific concepts (e.g., customer sites, routing policies), and takes advantage of SDN to reduce the involved technologies to a very small set, thus abating or even eliminating the need for a complex VPN management system. While we do not pretend to replace sophisticated management suites, we believe the simplicity and clear specification of our solution can make it appealing for an ISP that is transitioning to SDN.

III. A Novel Approach to Realize VPNs with SDN

In this section we briefly review the current implementation of MPLS VPNs, discuss its inconveniences, and describe how we leverage SDN to address them and offer additional features.

A. Architecture and Technologies of VPNs Today

An MPLS VPN is a service that an ISP can offer to support the communication between geographically distributed sites of a customer network. Each site accesses the service by an interconnection between a *Customer Edge* (CE) router deployed in the customer's network and one or more *Provider Edge* (PE) routers which act as entry points to the ISP's network. Internal routers within the ISP are called *Provider* (P) routers and support exchange of MPLS-encapsulated traffic between different sites of the same customer. Traffic between different customers is kept separate by tagging packets with different MPLS labels, each identifying a VPN.

The setup of MPLS VPNs involves the following steps:

1) Setup of IP addresses for loopback interfaces of PE routers.
2) Setup of an internal routing protocol (e.g., Open Shortest Path First (OSPF) [18]) to ensure reachability between the loopback interfaces of different PE routers. This may involve basic traffic engineering parameters like link weights.
3) Setup of the Border Gateway Protocol (BGP) [19] to distribute routing information about customer networks among PE routers. The IP prefixes of each customer site are propagated on a full mesh of iBGP peerings established between PE routers (route reflectors can be used to improve scalability). Prefixes belonging to different customers are differentiated using the Route Distinguisher (RD) address qualifier offered by the MultiProtocol BGP (MP-BGP) [20] extensions, thus allowing address space overlaps between different customers. Optionally, prefixes can be tagged with a Route Target (RT) extended community to implement customized VPN topologies or allow communication between different VPNs. Optional NAT translation rules can be set up on some PE routers to grant customers with overlapping address spaces access to shared services or to the Internet.
4) Setup of the Label Distribution Protocol (LDP) [21] to assign MPLS labels. These labels identify paths between PE routers computed by the internal routing protocol.
5) Setup of MPLS forwarding. Packets going from a customer site to another travel in the ISP's network encapsulated in MPLS with two labels: one (inner) identifying the VPN and another (outer) identifying the label-switched path towards the PE router to which the target customer site is attached. Optionally, additional traffic engineering mechanisms (e.g., based on RSVP-TE) can also be deployed.

It is evident that even a simple VPN setup involves several technologies, exposing to the risk of configuration errors and complicating management and troubleshooting significantly. This is exacerbated by the fact that configurations are scattered on tens or even hundreds of devices, making their interplay extremely difficult to determine, especially in the presence of dynamic events (e.g., link failures). In addition, current technologies offer limited flexibility and control: for example, influencing routing paths by tuning OSPF weights is a hard task, sometimes accomplished by trial and error, and some integrity constraints just cannot be enforced (see, e.g., [12]).

B. Our Approach

We propose an SDN-based approach to realize VPNs that offers an extremely simplified configuration interface and preserves only the building blocks of a VPN that are strictly required to achieve the desired functionalities, thus gaining flexibility, controllability, and predictability.

In our approach CE routers are still standard IP routers set up with a default route pointing to the PE router they are connected to. On the other hand, we assume that all nodes in the ISP's network are SDN-enabled devices (datapaths) coordinated by an OpenFlow controller. Flexibility of management is improved in this scenario because the role of a network node (PE or P router) can be changed by just intervening on the SDN controller, without the need for costly firmware or hardware replacements. Because of its adequacy and widespread availability, we keep MPLS as the encapsulation technology for packets traversing the ISP's network.

We designed an SDN controller that orchestrates all datapaths in the ISP's network to set up VPNs without the need for additional technologies. Several configuration elements, as well as signaling and routing protocols, are no longer required, thus ruling out many subtle interactions and making the outcome of configurations more predictable. Referring to the steps in Section III-A, we make the following simplifications:

1) PE routers still need an IP address, but it is only used to contact the SDN controller over IP: their forwarding behavior is instead completely managed by the SDN controller.
2) Internal routing protocols are no longer required, because the SDN controller knows the network topology and can compute routing paths between PE routers as needed.
3) MP-BGP is also dropped, because address space overlaps between VPNs are handled by distinct flow entries installed on the datapaths. Any special routing policies (including, e.g., balancing on multiple uplinks) can be part of the controller's operational logic, therefore RDs and RTs are not needed either. NAT address translation rules are translated into flow entries as well.
4) LDP is also abandoned, because the assignment of MPLS labels is completely managed by the SDN controller.
5) We retain MPLS to forward packets from a PE router to another, but labels are pushed and popped by flow entries installed by the SDN controller based on VPN configurations and, possibly, on traffic engineering mechanisms.

In our approach, the setup of all VPNs is described in a single configuration specification, thus making it readily accessible. This specification contains information about each customer site, including the IP subnets it hosts (possibly learned by means of a routing protocol), the PE router it is attached to, and an optional specification of a PE router that is the egress point to the Internet. It also contains associations between customer sites and VPNs and a selection of the routing policy to be adopted for each VPN. Optionally, it contains NAT address translation rules and further policies for the communication between customer sites. We describe these information in detail in Section IV.

Our SDN controller consists of the components depicted in Figure 1, which realize the following functionalities.

Configuration parsing and basic setup – The controller fetches two kinds of configuration information. Global settings consist of a mapping between datapath IDs and symbolic names, used to build a datapath inventory and to conveniently reference devices in the rest of the configuration. VPN configurations, specified using the language in Section IV, instruct the controller about the VPN scenario to be realized.

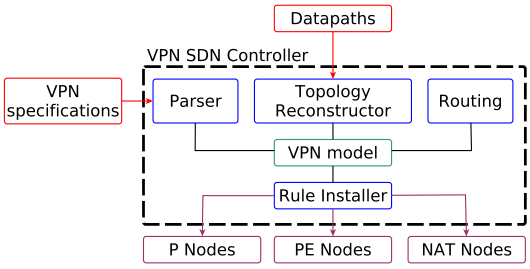

Fig. 1. Architecture of our SDN controller for MPLS VPNs.

The controller continuously watches these configurations for possible changes.

Topology reconstruction – The controller builds a complete representation of the network topology and keeps it up to date as network events (datapath/link failures or additions) occur. This activity is accomplished by exploiting the Link Layer Discovery Protocol (LLDP) [22] in the standard way envisioned by OpenFlow. When a link change event is triggered, the following functionality is activated.

Computation of routing paths – The controller (re)computes routing paths between PE routers as needed whenever a topology change is detected. If multiple paths are available between two PE routers, it selects a best path for each VPN, based on the routing policy specified in the configuration (e.g., shortest path). Although not within the focus of this paper, this policy may realize traffic engineering by taking into account the dynamic state of the network (e.g., link utilization).

The above components concur to maintain an internal representation of the VPN scenario that is used by another component to appropriately deploy OpenFlow rules:

Flow entries installation – VPN configurations are translated into flow entries to be installed on datapaths, as described in Section IV-B. Flow entries are always installed proactively, namely before any traffic is exchanged (we briefly discuss in Section IV-B how to improve scalability by installing some entries in reaction to observed network traffic). If a configuration change occurs, we assume that the controller clears all flow tables of all datapaths and immediately installs new flow entries according to the new configuration.

Unlike current MPLS VPNs, where a different MPLS label is used for each link and forwarding between PE routers happens by label switching, we use a fixed label along all the path. This is very beneficial for manageability, because the same label has the same semantic at any point in the ISP's network, and performance, because packet headers need not be rewritten. Similarly to some existing implementations, forwarding decisions on P routers are not affected by a packet's input port. Complex routing policies such as distinct routing paths per VPN or per service class, multipath routing, or destination-based load balancing can still be achieved by a suitably structured partitioning of the label space.

C. Applicability Considerations

Our approach can be readily implemented in existing networks provided that datapaths support release 1.1 of the

OpenFlow specification, (i.e., the earliest one that introduced support for handling MPLS labels). At the time of writing, several major vendors including Corsa Technology [23] Brocade [24], Huawei [25], Arista Networks [26], and NEC [27] already introduced OpenFlow 1.3 support in their product offering. Other vendors, as well as renowned silicon producers like Broadcom and Marvell, have OpenFlow 1.3 support in their roadmap (see, e.g., [28], [29]) or will do in the foreseeable future. Big market players such as Cisco and Juniper propose customized SDN implementations that offer similar functionalities under a different interface, but they also support open standards by means of OpenFlow compatibility modules (see, e.g., [30]). Recent releases of the OpenFlow specification are usually quickly adopted in leading software switch implementations such as Open vSwitch [31], Lagopus [32], and Switch Light [33], which are also designed to run on bare metal and can leverage packet processing libraries such as Intel's DPDK [34] in order to achieve production-level performance.

Due to hardware constraints, the size of flow tables is often very limited. While this is a realistic concern, in our opinion several arguments can mitigate it. First of all, even the most limited devices, which can accommodate few thousands of flow entries (see, e.g., [35]), can support the typical amount of MPLS VPNs operated by a small-sized ISP. Moreover, vendors move towards adopting more powerful silicon capable of handling tens of thousands of flow entries [36], and a suitable combination of optimization methods, like e.g. [37], and smart hardware solutions (see, e.g., [38], [39]) can push this limit up to millions of flow entries, practically ruling out scalability problems. Finally, such problems are insignificant for software switches, whose scalability limits can be overcome by affordable hardware upgrades, and whose performance are continuously pushed towards production level (see, e.g., [40]).

The transition from an existing operational network offering MPLS VPN services to an OpenFlow-enabled network should not be a concern either: there exist solutions, like RouteFlow [41], conceived to support the coexistence of OpenFlow and standard routing protocols. On the other hand, a project within the Open Networking Foundation is releasing use cases and guidelines to accomplish such transition (see, e.g., [42]). In addition, our approach can be applied regardless of whether controller-switch communication is realized out of band or in band. Finally, existing techniques to realize the SDN controller with a distributed architecture can be adopted to improve fault tolerance (see, e.g., [43], which also addresses network partitioning).

IV. A Configuration Language for VPNs

In this section we describe our VPN specification language. Although well-established configuration languages (e.g., YANG [44]) already exist, as far as we know this the first one specially designed for VPNs: as such it addresses domain-specific concepts (e.g., customer sites) as opposed to technical aspects (e.g., modules, namespaces, etc.), making the configuration much simpler. In this section we also explain how a configuration is turned into flow entries.

A. Configuration Primitives

First of all, a preamble in the configuration specifies, for each datapath, an association between its ID and a symbolic name: <datapath name="*sym_name*" dpid="*DPID*" />. For example: <datapath name="pe4" dpid="4" />. Following that, VPN configurations are specified by the following elements:

- A specification of the customer sites. It defines the IP subnets owned by each site (which may be automatically learned by means of a routing protocol like BGP), the PE router and port it is connected to, and, optionally, a PE router that acts as an egress point towards the Internet for that site.
  <customer-site name="*name*">
  <network subnet="*IP_subnet*" />...
  <connection pe="*PE_name*" port="*PE_port*" />
  <default pe="*egressPE_name*" />
  </customer-site>
  Example:
  <customer-site name="BigCompany_Rome">
  <network subnet="10.0.0.0/16" />
  <network subnet="10.172.0.0/16" />
  <connection pe="pe_rm" port="3" />
  <default pe="default_pe" />
  </customer-site>
- A specification of the VPNs to be set up in the network. For each VPN, it consists of a set of participating customer sites and of the policy to be applied to compute routing paths between PE routers (at present we envision only the shortest path policy, but more elaborate traffic engineering policies along the lines of [16] are of course possible).
  <vpn name="*VPN_name*">
  <site id="*site_name*" />...
  <routing type="*policy*" />
  </vpn>
  Example:
  <vpn name="BigCompany">
  <site id="BigCompany_Rome" />
  <site id="BigCompany_Tokyo" />
  <routing type="shortest-path" />
  </vpn>
- A specification of NAT address translation mechanisms. They are required when multiple VPNs with overlapping address space need to access the Internet or any shared services. Each customer site can use a different mechanism, and each mechanism comprises one or more translation rules that apply to single IP addresses or IP subnets.
  <nat egresspe="*PE_name*" egressport="*PE_port*">
  <mapping vpn="*VPN_name*" site="*site_name*">
  <rule private="*private_IP*" public="*global_IP*">...
  </mapping>...
  </nat>
  In this specification egresspe and egressport indicate the PE router and port that are connected to the Internet. Each mapping statement determines that the following IP address translation rules apply to traffic coming from a specific site of a specific vpn and directed towards the Internet. Each rule specifies that a packet originated from a private IP address must have its source address translated to a public IP address when sent to the Internet (packets received in response must have their destination address translated accordingly). private and public can be single IP addresses (*static NAT*). Alternatively, they can be two IP subnets of equal size, in which case IP addresses in the private subnet are mapped to IP addresses in the public subnet (*dynamic NAT*). Finally, private can be a wildcard '*' and public can

be an IP subnet, in which case any IP address belonging to the considered vpn and site is dynamically mapped to an available IP address in the public subnet (*full NAT*) when traffic is sent to the Internet. Example:

```
<nat egresspe="default_pe" egressport="1">
  <mapping vpn="BigCompany" site="BigCompany_Rome">
    <rule private="10.0.100.8" global="211.10.20.4" />
    <rule private="10.0.200.0/24" global="211.10.40.0/24" />
  </mapping>
  <mapping vpn="BigCompany" site="BigCompany_Tokyo">
    <rule private="*" global="190.25.0.0/16" />
  </mapping>
</nat>
```

In this example an IP address of site BigCompany_Rome in VPN BigCompany is statically mapped to another IP address when traffic from that site is sent to the Internet. Moreover, for the same site the controller will establish a mapping between addresses in subnet 10.0.200.0/24 and addresses in subnet 211.10.40.0/24. Last, the controller will dynamically remap any IP addresses belonging to site BigCompany_Tokyo of VPN BigCompany to an available address in subnet 190.25.0.0/16.

NAT translation rules are optional: a PE router can be the Internet egress router for a VPN even if no NAT rules are configured. Of course, if multiple VPNs with an address space clash need Internet access, response traffic from the Internet may be handled unpredictably in the absence of NAT rules.

Even if not explicitly stated, a configuration expressed with this language determines the role played by each datapath (P router, PE router, NAT), which in turn determines what flow entries the controller will install on that datapath. Actually, all the datapaths must at least be able to forward traffic between PE routers, namely must behave as P routers. The other roles are determined by the configuration context: for example, a datapath mentioned in a connection statement is to be considered a PE router. Another relevant aspect is that every packet coming from a customer site and entering the ISP's network must be assigned to the correct VPN. This is not difficult in general because, even if the customer site participates in multiple VPNs, the packet's destination IP address may belong to only one of these VPNs. An ambiguity arises if the source customer site and the destination IP address belong to more than one common VPN. We solve this ambiguity by assuming that the packet is sent to the IP address in the first VPN specified in the configuration.

With our language it is also possible to express complex routing policies. For example, allowing communication between customer sites in different VPNs is as simple as adding both sites to a new VPN created ad hoc. In a traditional MPLS VPN, this would require a selective tagging of IP prefixes with RTs and a leakage of tagged routes from one VPN to the other. Forbidding communication between customer sites is also possible, but a bit more difficult: to prevent traffic between customer sites S_1 and S_2 belonging to the same VPN V_1, one of the two sites, say S_1, can be moved to a separate VPN V_2. In addition, all the sites of V_1 except S_2 must then be added to VPN V_2 to permit communication with S_1. In order to make such configurations easier to specify and to support more flexible routing policies, we introduce an additional primitive:

```
<policy vpn="VPN_name">
  <deny from="from_subnet" to="to_subnet" />...
  <allow from="from_subnet" to="to_subnet" />...
</policy>
```

This specification blocks or allows traffic originated by IP addresses in subnet *from_subnet* and directed to IP addresses in subnet *to_subnet* within VPN *VPN_name*. With this primitive it is possible to selectively control communication between specific subnets of customer sites. Example:

```
<policy vpn="BigCompany">
  <deny from="10.10.100.0/24" to="10.10.200.1/32" />
</policy>
```

B. From Configurations to Flow Entries

We now describe how the SDN controller translates VPN configurations into flow entries to be installed on datapaths. Unless differently specified, all flow entries are installed proactively. We divide flow entries into classes according to their role and structure. Also, we call *ingress PE* the router through which packets from a customer site enter the ISP's network and *egress PE* the router through which these packets leave the ISP's network and reach their destination. Table I can be used as a reference to follow the description of the flow entries.

We first consider flow entries installed on P routers: these flow entries support delivery of MPLS-encapsulated packets between PE routers, therefore we call them *tunnel flow entries*. Each entry matches only on the outer MPLS label of a packet, which determines a path towards a PE router, and sends the packet out of the port that leads to that PE router along that path. If the P router under consideration is the penultimate hop along the path to the PE, the flow entry also pops the outer label before forwarding the packet. In principle, on a P router there is one tunnel flow entry for each PE router. However, since the same MPLS label is used along all the path and the union of all the paths to a PE router forms a tree, in most practical cases the number of installed flow entries is lower. On the other hand, the controller may also compute multiple paths (for example, one for each VPN) towards the same PE router to support traffic engineering: correspondingly, there can be multiple tunnel flow entries for the same PE router.

We now describe flow entries installed on PE routers, which belong to 3 different classes. These entries depend on VPN configurations and on a proper identification of the destination VPN and customer site of each packet, as specified in Section IV-A. The first class of flow entries supports connectivity between customer sites attached to the same PE router: we call them *local delivery flow entries*. Such a flow entry matches on the input port as well as on the source and destination IP subnet, and sends the packet out of the port connected to the target customer site. In the absence of configured policies, the controller only installs entries that support communication between sites (and prefixes) in the same VPN. On a PE router there is at most one local delivery flow entry for every pair of IP subnets in directly connected customer sites belonging to the same VPN. In practice, there can be fewer flow entries because two customer sites may share more than one VPN (a single entry for each pair of IP subnets is then enough to support communication for all these VPNs).

The second class of flow entries at PE routers supports delivery of packets between sites connected to different PE

TABLE I. AN OVERVIEW OF THE FLOW ENTRIES INSTALLED ON DATAPATHS IN THE ISP'S BACKBONE TO SUPPORT VPN SERVICES.

Entry class	Datapaths	Match conditions	Actions	Expected number of entries
Tunnel	P	MPLS label	If penultimate hop towards a PE, pop the label. Send packet out of the port along the path determined by the label.	One for each PE (or more, for traffic engineering)
Local delivery	PE	Input port, source and destination IP subnet	Send packet to the target customer site.	One for every pair of IP subnets in directly connected customer sites in the same VPN
Remote delivery	Ingress PE	Input port, source and destination IP subnet	Push one label for the VPN and one for the path to the egress PE. Send packet out of the port along the path to the egress PE.	One for every pair of IP subnets in remotely connected customer sites in the same VPN
Remote delivery	Egress PE	MPLS label, destination IP subnet	Pop label. Send packet to the target customer site.	One for every IP subnet of every VPN of directly connected customer sites
Default-out	Ingress PE	Input port, source IP subnet	Push one label for the path to the default PE. Send packet out of the port along the path to the default PE.	One for every IP subnet of every customer site that has a default PE
Default-out	Egress PE	—	Send packet out of the configured Internet-connected port.	One
Default-in	Ingress PE	Input port, destination IP subnet	Push one label for the VPN and one for the path to the default PE. Send packet out of the port along the path to the egress PE.	One for every IP subnet of every customer site that uses this specific PE as default PE

routers: we call them *remote delivery flow entries* and describe them separately for ingress and egress PE routers. A remote delivery flow entry at an ingress PE router matches on the input port as well as on the source and destination IP subnets. Upon successful match, it pushes two MPLS labels on the packet (one identifying the VPN and the other identifying the path to the egress PE) and sends the packet out of the port that is along the path to the egress PE. The SDN controller assigns a distinct label to each VPN and to each routing path: when installing remote delivery flow entries, the controller selects the VPN label to be applied as specified in Section IV-A, and selects the path label according to the computed routing paths (which, in turn, depend on the configured routing policies). For the count of remote delivery flow entries, the same considerations made for local delivery flow entries apply. A remote delivery flow entry at an egress PE router matches on the VPN label of a packet and on its destination IP subnet, it pops the label, and it sends the packet out of the port to which the target customer site is connected, which is determined based on the destination IP and the VPN. At a given PE router there will be one such flow entry for each IP subnet of each VPN of the customer sites that are connected to that router.

The third class of flow entries supports routing of network traffic from a customer site of a VPN to the Internet through the PE router that has been configured as default for that site. We call such entries *default-out flow entries*, and those supporting response traffic from the Internet *default-in flow entries*. We first describe the basic structure of these flow entries in the absence of NAT address translation rules. A default-out flow entry at an ingress PE matches on the input port (required because every customer site can use a distinct PE router as an egress to the Internet) and on the source IP subnet, it pushes one MPLS label indicating the path to the egress PE router (one label suffices because packets are not directed to a specific VPN but to the Internet), and sends the packet out of the port that is along the path to the egress PE router. Flow entries of this type are installed with a lower priority value so that they are matched only after any other flow entries, thus implementing a "default route". There is one such flow entry for every subnet of every customer site that has been configured for Internet access. A default-out flow entry at an egress PE just sends the packet out of the Internet-connected port, specified in the configuration. On a specific PE router there is just one such flow entry, and it is also installed with a low priority value. A default-in flow entry at an ingress PE router matches on the input port (the Internet-connected port) and on the destination IP subnet, pushes two labels on the packet (one for the VPN and one for the path to the egress PE), and sends the packet out of the port that leads towards the PE router that is connected to the destination customer site. If the destination IP address belongs to more than one VPN, the actual target VPN can be determined as assumed in Section IV-A, or proper NAT translation rules need to be set up to prevent any ambiguity. There is one default-in flow entry at an ingress PE router for every IP subnet of every customer site that uses that PE router as an exit point to the Internet. There is no need for specific default-in flow entries at an egress PE router, because their function is accomplished by remote delivery entries. If a customer site uses the PE router it is connected to as the default PE router, ingress and egress flow entries are condensed into a single entry: in this scenario a default-out flow entry matches on the input port and on the source IP subnet and sends the packet out of the Internet-connected port; a default-in flow entry matches on the Internet-connected port and on the destination IP address, and sends the packet out of the port connected to the relevant customer site.

Depending on their structure, the SDN controller can install flow entries at different moments: entries that are independent of the ISP network's topology, including local delivery flow entries and entries that are deployed on egress PE routers, can be installed immediately after the controller has fetched VPN configurations, and are never altered unless configuration changes occur. Any other entries, including tunnel flow entries and entries deployed on ingress PE routers, involve an output port in their actions and therefore depend on the network topology: as such, they can only be installed once the controller has computed paths between PE routers. If the controller recomputes these paths, for example because better ones become available, existing ones are disrupted (e.g., due to link failures), or because of traffic engineering policies, such entries need to be updated accordingly.

Our approach requires the installation of a number of flow entries that is comparable with the number of forwarding table entries of routers implementing traditional MPLS VPNs, and in general can be lower. Tunnel flow entries are at most as numerous as forwarding entries because, since we abandon label switching, there is no need for a distinct entry for each input port. Default-out flow entries at the egress PE are less numerous because one entry suffices for all VPNs. On the other hand, for some classes (local delivery and entries installed at ingress PE routers except default-in entries) we use a higher number of flow entries, because they also match on the source IP subnet. We made this choice to prevent forwarding

traffic from unintended VPNs, thus ruling out the integrity and isolation problems documented in [12]. If this is not a concern, the number of flow entries can be largely reduced.

We now describe NAT-related flow entries, which are implemented as a variation of other classes of entries at PE routers. The specification of NAT rules in the configuration is independent of whether NAT is implemented on ingress or egress routers (see [45] for a discussion of the two alternatives), but different flow entries are installed in the two cases. Implementing NAT at the egress requires the following changes. Default-out flow entries at any ingress PE router must also push the VPN label, and default-out flow entries at any egress PE router must also match on this label and on the source IP subnet: this is required to recognize the source customer site and apply the correct address translation rules. Before forwarding a packet, default-out flow entries at the egress PE must also replace its source IP address with a public address according to the configured rules. Similarly, when a default-in flow entry at an ingress PE router matches a response packet from the Internet directed to a mapped IP address, it must replace its destination IP with the original private address before forwarding the packet. On the other hand, NAT can be equivalently implemented at the ingress in the following way: each default-out flow entry at the ingress PE router of a customer site that is configured for NAT must also replace the source IP address of each packet with a public address before forwarding it. Similarly, default-in flow entries at the same router, which are implemented by remote delivery flow entries, must replace the destination IP address of a packet directed to a mapped IP address with the original private address. Regardless of where NAT is implemented, if a customer site uses the PE router it is attached to as the default PE router, then the entire address mapping takes place on that router.

In order to implement NAT, existing default-out and default-in flow entries must be replicated for each IP address that is subject to translation. In accordance with [45], implementing NAT on ingress routers can improve scalability in this scenario, because flow entries can be distributed across several devices. These considerations apply as long as flow entries are installed proactively. An alternative approach (the most viable one for full NAT) is to install NAT-related flow entries on-demand when traffic from new source IP addresses is observed, and to set them up with a timeout so that public IP addresses can be reused when this traffic is interrupted. With this solution, the number of NAT-related flow entries equals the number of entries in a traditional NAT translation table.

The last class of flow entries implements forwarding policies specified with the <policy> primitive defined in Section IV-A. We call such entries *policy flow entries* and assume they are deployed at ingress PE routers to which customer sites hosting the specified subnets are attached. For each <deny> policy, a flow entry is installed that drops any packets within the specified VPN (determined by the input port) that match the specified source and destination IP subnets. These entries are installed with a high priority, so that they override any other matching flow entries. For each <allow> policy, additional local or remote delivery flow entries are installed to support communication between the specified subnets.

In order to translate VPN configurations into flow entries, our controller reconstructs an internal representation of the network topology, as well as of the VPNs and the participating customer sites, and sends to the datapaths the appropriate flow entries for each class, as described above and summarized in Table I. Although conceived with OpenFlow primitives in mind, our language is independent of the specific technologies used to deploy VPN configurations: in principle, a specification expressed in our language could be implemented on network devices using other protocols like, e.g., NETCONF.

V. Experimental Evaluation

In order to validate the effectiveness of our approach, we implemented a prototype SDN controller (available at [46]) and performed several experiments.

A. Scenario and Technologies

We developed the controller using Ryu[1] [47], a Python-based framework which extensively supports OpenFlow specifications 1.0 through 1.4, enabling us to handle MPLS labels. We tested our controller with Open vSwitch[2] [31], one of the most feature-rich implementations of an OpenFlow-compliant software switch that is also used on a range of commercial hardware devices (e.g., Corsa Technology switches). We ran our tests inside Mininet[3] [48], a state-of-the-art emulator that is commonly adopted for experimenting with SDN.

Our prototype implements all the components described in Sections III-B and IV-B. At present configuration parsing, topology discovery (by means of an API offered by Ryu), routing path computation between PE routers, and all NAT types (static, dynamic, full) are supported. In addition, the controller reacts to topology changes: when a new link is detected, the Topology Reconstructor is notified (see Figure 1) and the Routing component recomputes paths between each pair of PE routers for which a path is not yet available. If a new path is found, the Rule Installer deploys flow entries on the relevant datapaths. When an existing link disappears, the Routing component determines whether there are currently used paths in which the disappeared link occurs. For every such path, the Rule Installer removes from the relevant datapaths all the flow entries that support that path and that are not used by any other paths (we use a reference counter to determine when an entry can be removed). Other features are unsupported or only partially implemented: our prototype controller understands a slightly simplified version of our configuration language, it does not detect on-the-fly configuration changes, it selects a random shortest path between PE routers, and it only allows the specification of a single IP subnet for each VPN of each customer site. We argue that none of these features affects the significance of our experiments.

In implementing the controller we had to face some technical issues. Due to the lack of a feature called "recirculation", in the official implementation of Open vSwitch it is currently impossible to look past MPLS labels in a match condition. We overcame this problem by using VLAN tags as VPN identifiers in place of MPLS labels. Accordingly, we split each flow entry matching both the VPN identifier and the IP subnet into two entries placed in separate flow tables: one entry that matches

[1]Ryu 3.8, latest Git snapshot as of April 2nd, 2014.

[2]Open vSwitch version 2.3.90, latest Git snapshot as of May 19th, 2014.

[3]Mininet version 2.1.0+, latest Git snapshot as of May 7th, 2014.

TABLE II. RESULTS OF OUR TEST RUNS WITH DIFFERENT TOPOLOGIES (FROM [50]) AND VPN CONFIGURATIONS.

Topology	Nodes	Links	Max flow entries	Convergence time (s)	Recovery time (s)
Att	25	56	98	0,8	0,03
Bell Canada	48	64	47	1,2	0,03
BT Europe	24	37	106	0,5	0,05
Evolink	37	45	63	0,7	0,1
NTT	47	63	67	0,8	0,04
Oxford	20	26	100	0,6	0,24
Renater2006	33	43	72	0,6	0,11
York	23	24	91	0,7	0,17

the VLAN tag, stores its value in a standard *Metadata* registry, removes the VLAN header, and points to the other flow table; another entry that matches the Metadata and the IP subnet and performs the required actions. We also experienced problems with the MAC address learning module of Ryu, which we worked around by populating the ARP caches of host machines in advance. Finally, due to a Mininet bug [49], for each test we had to update the association between Open vSwitch port numbers and virtual network interfaces. None of these problems is due to flaws in our approach or even OpenFlow itself: they are implementation-specific issues that do not impair the usage of our controller with hardware datapaths.

B. Experiments

We exploited our prototype to verify the functionality, scalability, robustness, and performance of our approach. For these purposes we used 3 groups of topologies and VPN configurations (the realism of these configurations was confirmed in a discussion with a medium-size ISP): a) A simple topology with 2 hosts in the same VPN, each attached to a PE router; the two PE routers were connected by 3 independent paths made up of 10 P routers each. We used this topology for functional tests. b) A more complex topology consisting of 12 P routers and 7 PE routers, each with one host attached and 2 of which performing NAT, totaling 25 links. We used variations of this topology to consider increasingly complex scenarios: at first just 2 customer sites in the same VPN, then more VPNs with overlapping IP address space, and finally VPNs with default PE routers and NAT translation rules (we modeled "the Internet" as a host connected to each default PE router). c) A range of real-world topologies from the Topology Zoo [50] project. For each of them we considered 100 VPNs, each consisting of 2 hosts in 2 different IP subnets. The hosts in each pair were attached to 2 distinct randomly picked network nodes which therefore acted as PE routers (at most 33 VPNs were set up on every PE); all the other routers were just plain P routers. We used these topologies for scalability tests.

For the functionality tests, we successfully verified connectivity between hosts in the same VPN and isolation between hosts in different VPNs, using a standard ping. We also verified that packets were encapsulated as expected. For the scalability tests, we assessed the number of installed flow entries and checked that it did not exceed the expected count stated in Section IV-B. In the most complex topology of group b) that we considered (9 VPNs, 2 customer sites for each VPN with one IP subnet each, and 9 static NAT rules) we had at most 36 flow entries on PE routers and at most 3 on P routers. Table II shows the maximum number of flow entries on any datapath for the topologies of group c). We believe these numbers show that our approach is pretty conservative in terms of consumed flow table space. To verify robustness to topology changes, we simulated several link failures, carefully selected to disrupt a high number of paths taken by network traffic. In the presence of such faults, packet loss may occur for at least two reasons. One is that packets may get trapped in a disconnected network region. The other reason is related with the behavior of datapaths: if an incoming packet does not match any flow entries, a datapath sends a request to the controller and only buffers the packet until a response is received. We ascertained that, if a matching flow entry is installed independently of this request-response transaction (like our controller does), the packet is lost. We verified the impact of link failures during a flood ping (i.e., a ping at the maximum rate made possible by the network) and during a TCP file transfer between two hosts: no more than 4 ICMP packets were lost in all our tests, and the transfer was never disrupted nor significantly slowed down. Our implementation can be further improved because it needlessly clears and reinstalls some flow entries involved in the changed paths. To evaluate performance, we started the controller in advance and estimated the time since the network was brought up to the moment in which all flow entries were installed: it never took more than 0.8s to install flow entries on all datapaths in group b) topologies; results for group c) topologies are in column "Convergence time" of Table II. We then estimated the time required to recompute paths and install new flow entries after link failures, which was always below 0.3s both for group b) topologies and for group c) topologies (column "Recovery time" in Table II). Considering the limits imposed by the emulation, we deem these figures rather promising for a production deployment.

VI. CONCLUSIONS AND FUTURE WORK

Taking advantage of the programmability of SDN, we propose a novel approach to realize MPLS VPNs, which supports easy provisioning and setup based on a simple and flexible configuration language, facilitates management and troubleshooting by dropping unneeded technologies, and improves controllability and predictability by enforcing a centralized coordination of the behaviors of network devices.

Although our language and approach are readily applicable, our research directions envision several improvements. Support for traffic engineering, albeit envisioned, is currently very limited: more elaborate routing policies could be introduced, and traffic could be dynamically re-routed depending on target QoS levels. Moreover, an incremental handling of installed flow entries is desirable, to support configuration changes as well as addition, removal, and migration of customer sites with minimal disruptions. A mechanism to learn customer IP prefixes, possibly by integration with existing protocols like BGP, should also be introduced. In addition, the language could be extended to support services such as Virtual Private LAN Service (VPLS). Finally, improvements to our prototype implementation and experiments on hardware testbeds or operational networks are likewise part of our work plan.

ACKNOWLEDGMENTS

We would like to thank the Unidata S.p.A. staff for the profitable discussions about VPN configurations and for allowing us to perform tests on a range of commercial devices.

REFERENCES

[1] AT&T, "Point of View: MPLS – A strategic technology," 2003.

[2] Frost & Sullivan, "MPLS/IP VPN services market update, 2014," 2014.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, 2013.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[5] L. Cittadini, G. Di Battista, and M. Patrignani, "MPLS virtual private networks," in *Recent Advances in Networking, Volume 1*, ser. ACM SIGCOMM eBook, H. Haddadi and O. Bonaventure, Eds. ACM, 2013, pp. 275–304.

[6] M. Mouzuddin and M. Shaikh, "Routing issues in deploying MPLS VPNs," in *Proc. MultiProtocol Label Switching Conference & Exhibition (MPLScon)*, 2005.

[7] Hewlett-Packard, "Intelligent Management Center – MPLS VPN Manager Software," Jan 2015. [Online]. Available: http://h17007.www1.hp.com/us/en/networking/products/network-management/IMC_MPLS_VPN_Software/index.aspx

[8] Cisco Systems, Inc., "Cisco Prime Provisioning," Jan 2015. [Online]. Available: http://www.cisco.com/c/en/us/products/cloud-systems-management/prime-provisioning/index.html

[9] Packet Design, "Route Explorer," Jan 2015. [Online]. Available: http://www.packetdesign.com/products/mpls-wan-explorer

[10] G. D. Battista, M. Rimondini, and G. Sadolfo, "Monitoring the status of MPLS VPN and VPLS based on BGP signaling information," in *Proc. IEEE NOMS*, 2012, pp. 237–244.

[11] W. Enck, T. Moyer, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, Y.-W. E. Sung, S. Rao, and W. Aiello, "Configuration management at massive scale: system design and experience," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 3, pp. 323–335, April 2009.

[12] R. Bush and T. Griffin, "Integrity for virtual private routed networks," in *Proc. INFOCOM*, 2003.

[13] Open Networking Foundation, "OpenFlow Switch Specification, version 1.5.0," Jan 2015.

[14] ONRC Research, "An SDN approach to MPLS traffic engineering and virtual private networks," Jan 2015. [Online]. Available: http://onrc.berkeley.edu/research_sdn_approach_to_mpls_traffic_engineering.html

[15] S. Das, A. Sharafat, G. Parulkar, and N. McKeown, "MPLS with a simple OPEN control plane," in *Proc. Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*, 2011.

[16] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNS with OpenFlow," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 452–453, 2011.

[17] S. Das, "PAC.C: A unified control architecture for packet and circuit network convergence," Ph.D. dissertation, Stanford University, 2012.

[18] J. Moy, "OSPF Version 2," RFC 2328, Internet Engineering Task Force, Apr. 1998.

[19] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Internet Engineering Task Force, Jan. 2006.

[20] T. Bates, R. Chandra, D. Katz, and Y. Rekhter, "Multiprotocol Extensions for BGP-4," RFC 4760, Internet Engineering Task Force, Jan. 2007.

[21] L. Andersson, I. Minei, and B. Thomas, "LDP Specification," RFC 5036, Internet Engineering Task Force, Oct. 2007.

[22] IEEE, "IEEE standard 802.1AB – station and media access control connectivity discovery," 2009.

[23] Corsa Technology Inc., "SDN done right," Jan 2015. [Online]. Available: http://www.corsa.com/sdn-done-right/

[24] Brocade, "Software-Defined Networking – OpenFlow," Jan 2015. [Online]. Available: http://www.brocade.com/solutions-technology/technology/software-defined-networking/openflow.page

[25] European Advanced Networking Test Center, "Huawei Technologies SDN showcase at SDN and OpenFlow world congress 2013," 2013. [Online]. Available: http://www.eantc.de/fileadmin/eantc/downloads/events/2011-2015/SDNOF2013/EANTC-Huawei_SDN_Showcase-White_Paper_Final_Secure.pdf

[26] Arista Networks, "Arista 7280E series," Jan 2015. [Online]. Available: https://www.arista.com/en/products/7280e-series

[27] NEC Corporation, "ProgrammableFlow PF5240 Switch," Jan 2015. [Online]. Available: http://www.necam.com/sdn/doc.cfm?t=PFlowPF5240Switch

[28] Broadcom Corporation, "OpenFlow – Data Plane Abstraction Networking Software," Jan 2015. [Online]. Available: http://www.broadcom.com/products/Switching/Software-Defined-Networking-Solutions/OF-DPA-Software

[29] Marvell, "Prestera DX packet processors," Jan 2015. [Online]. Available: http://www.marvell.com/switching/prestera-dx/

[30] Cisco Systems, Inc, "Cisco plug-in for OpenFlow," Jan 2015. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sdn/configuration/openflow-agent-nxos/cg-nxos-openflow.html

[31] "Open vswitch," Jan 2015. [Online]. Available: http://www.openvswitch.org

[32] "Lagopus switch," Jan 2015. [Online]. Available: http://lagopus.github.io/

[33] Big Switch Networks, "Switch Light," Jan 2015. [Online]. Available: http://www.bigswitch.com/products/switch-light

[34] Intel, "DPDK: Data Plane Development Kit," Jan 2015. [Online]. Available: http://dpdk.org/

[35] Brocade, "NetIron SDN Configuration Guide," Jan 2015. [Online]. Available: http://www.brocade.com/downloads/documents/html_product_manuals/NI_05500c_SDN/wwhelp/wwhimpl/common/html/wwhelp.htm#context=NI_SDNguide_html&file=OpenFlow_IPv4_IPv6.3.04.html

[36] Centec Networks, "CTC5162/CTC5163 product brief," Jan 2015. [Online]. Available: http://www.centecnetworks.com/en/ProductList.asp?ID=189

[37] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source Flow: handling millions of flows on flow-based nodes," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2010.

[38] NEC Corporation, "NEC ProgrammableFlow Univerge PF5240 product brief," Jan 2015. [Online]. Available: http://www.necam.com/docs/?id=5ce9b8d9-e3f3-41de-a5c2-6bd7c9b37246

[39] Huawei Technologies Co., Ltd., "SDN: The best answer to campus network challenges," Jan 2015. [Online]. Available: http://enterprise.huawei.com/en/about/e-journal/ict/detail/hw-311134.htm

[40] M. Carbone, G. Catalli, and L. Rizzo, "Improving the performance of Open vSwitch," in *Proc. EuroBSDCon*, 2011.

[41] CPqD - R&D Center for Telecommunications, "RouteFlow," Jan 2015. [Online]. Available: https://sites.google.com/site/routeflow/home

[42] Open Networking Foundation, "Migration use cases and method," Jan 2015. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf

[43] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. INM/WREN*, 2010.

[44] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, Internet Engineering Task Force, Oct. 2010.

[45] Cisco, "Cisco IOS NAT - Integration with MPLS VPN," Jan 2015. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/112084-ios-nat-mpls-vpn-00.html

[46] Roma Tre University – Computer Networks Research Group, "Software Defined Networking," Jan 2015. [Online]. Available: http://www.dia.uniroma3.it/~compunet/www/view/topic.php?id=sdn

[47] "Ryu controller," Jan 2015. [Online]. Available: http://osrg.github.io/ryu/

[48] "Mininet," Jan 2015. [Online]. Available: http://mininet.org

[49] "Mininet port numbering bug – GitHub bug report #312," Jan 2015. [Online]. Available: https://github.com/mininet/mininet/issues/312

[50] University of Adelaide, "The Internet topology zoo," Jan 2015. [Online]. Available: http://www.topology-zoo.org