Making MPLS VPNs Manageable through the Adoption of SDN

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*—Virtual Private Networks (VPNs) implemented by Multi Protocol Label Switching (MPLS) tunnels appear in the service offer of many Internet Service Providers (ISPs). Due to the number of technologies that they involve and to the intricacy of their interactions, provisioning, setup, and maintenance of VPNs is a cumbersome task, whose complexity is usually mitigated by using advanced network management systems.**

We cut these difficulties at their roots by taking advantage of Software Defined Networking (SDN). We showcase the design and a prototype implementation of an SDN controller that, starting from a centralized specification of VPN settings expressed in a high-level simple and flexible language, automatically fills flow tables to implement the requested VPNs. Our implementation of VPNs with SDN promptly reacts to network dynamics (e.g., newly appeared links) and simplifies management a lot by dropping many unneeded technologies.

I. Overview and Related Work

Virtual Private Networks (VPNs) are pervasively used to interconnect geographically sparse sites of an organization. Most Internet Service Providers (ISPs) use Multi Protocol Label Switching (MPLS) as the underlying technology to offer VPN services on a medium and large scale. Despite the long-term consolidated architecture that MPLS VPNs are based on (for a comprehensive illustration see [1], [2]), their management raised concerns since early deployments (see, e.g., [3]), and their provisioning, configuration, and maintenance are difficult and error-prone even nowadays. The main reason is that MPLS VPNs rely on a combination of many networking technologies, whose interplay is hard to predict and control. ISPs commonly resort to technologies (e.g., NETCONF) and tools (e.g., [4]) that help automate management and cope with these difficulties, but a true understanding of the underlying mechanics is still a time-consuming and laborious task.

Software Defined Networking (SDN) [5], the recently advocated separation between the software that implements the control plane logic of a device and the hardware that accomplishes packet forwarding, offers an effective abstraction that supports designing network services with greatly increased flexibility. Taking advantage of this, we overhauled the current architecture of MPLS VPNs to reimplement them with SDN-capable devices. We first of all designed a simple and flexible configuration language for a high-level specification of VPN settings: it does not involve technical details such as peerings, areas, tunnels, etc., but rather addresses context-specific concepts such as customer sites, VPNs, edge (PE) and core (P) routers, as well as NAT address translation rules. The language finds its natural application in SDN, which enables a central specification of VPNs with little effort. We then dropped many technologies that are no longer needed with SDN (e.g., MP-BGP, OSPF, LDP, even label switching), retaining only those that are really essential for the operation of VPNs (notably, MPLS). Finally, we designed an OpenFlow SDN controller that, starting from a file of VPN specifications expressed using our language, installs entries in the flow tables of datapaths to implement the requested scenario. We argue that the simplification of eliminating most traditional VPN technologies far exceeds the overhead of deploying an SDN controller.

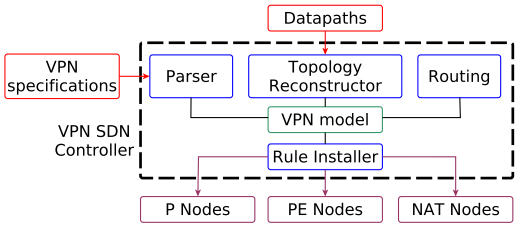

Fig. 1. Architecture of our prototype OpenFlow SDN controller.

We are aware of extremely few alternative SDN-based approaches for implementing MPLS VPNs. The most promising one, illustrated in [6], mainly focuses on traffic engineering aspects, lacks a rigorous description of SDN-related technical aspects, and is rather bound to the traditional way of operating MPLS VPNs, thus failing to address the configuration complexity problem. We therefore showcase a prototype SDN-based MPLS VPN implementation that has the following advantages: it starts from a set of simple and centrally specified VPN configurations, thus improving manageability; it reduces required technologies to a bare minimum (essentially, just MPLS), thus making network behaviors more predictable and simplifying troubleshooting; it takes advantage of the flexibility of SDN, so that each datapath can be repurposed to function as a PE router, a P router, or a NAT box without costly hardware replacements. Fault tolerance, a possible concern implied by the use of a centralized controller, can be ensured by using existing approaches (see, e.g., [7]).

For a complete illustration of the design of our SDN controller and of the VPN configuration language, see [8].

II. Architecture and Demo Setup

We implemented a prototype OpenFlow SDN controller for MPLS VPNs based on the comprehensive Ryu[1] framework, following the architecture in Figure 1. The controller consists of several components that operate in synergy to keep an

[1] http://osrg.github.io/ryu/

```
<vpns>
   <datapath name="p_node1" dpid="1" />
   <datapath name="p_node2" dpid="2" />
   <datapath name="p_node3" dpid="3" />
   <datapath name="p_node4" dpid="4" />
   <datapath name="p_node5" dpid="5" />
   <datapath name="pe_milan" dpid="6" />
   <datapath name="pe_rome" dpid="7" />
   <datapath name="nat_node" dpid="8" />
   <vpn name="example-vpn">
      <network subnet="10.0.0.0/24" pe="pe_rome"
               port="eth2" nat="nat_node" />
      <network subnet="10.0.1.0/24" pe="pe_milan"
               port="eth1" nat="" />
   </vpn>
   <nat name="nat_node" port="eth2">
      <mapping vpn="example-vpn" pe="pe_rome" type="static">
         <rule ip="10.0.0.1" gr="193.204.161.4" />
      </mapping>
   </nat>
</vpns>
```

Fig. 2. Example of VPN settings expressed in our configuration language.

internal representation of the network (the VPN model) up to date. A Topology Reconstructor takes advantage of an API offered by Ryu to continuously exchange information with each datapath in order to maintain a complete map of the network topology. This map is augmented with VPN configuration information (e.g., customer site connections, the desired routing policies, etc.) that a Parser retrieves from a specification written in a simple XML-based language. The best paths from a PE router to another are determined by a Routing module based on user-specified routing policies (e.g., shortest path, TE-based policies, etc.) As the VPN model is updated, its contents are translated into OpenFlow match-action entries that a Rule Installer component deploys in the flow tables of each datapath. These entries instruct the datapath to push/pop MPLS labels, as well as manipulate and forward packets based on its role of P router, PE router, or NAT. Network dynamics are also supported: a change in the network topology induces a recomputation of routing paths and a quick refresh of the flow entries installed in the datapaths.

Figure 2 shows a sample specification of VPN settings expressed in the configuration language supported by our prototype SDN controller: after assigning mnemonic names to datapaths (`p_node1`, `pe_milan`, etc.), it defines a single VPN by specifying the participating customer sites and the PE router ports they are attached to. It also specifies that a PE router (`pe_rome`) should mask a private IP address by mapping it to a public one (note that other mapping options are available besides this static mapping).

We demonstrate the operation of our controller in the Mininet[2] network emulation environment, which in turn is powered by Open vSwitch[3], an advanced implementation of an OpenFlow SDN datapath that is also adopted on a range of hardware devices. The scenarios we plan to consider range from simple topologies with a single VPN, useful to observe and understand the installed flow entries in detail, to complex setups derived from the topologies available at [9].

Our prototype SDN controller, which is publicly available for download at [10], implements all the components described at the beginning of this section with a few minor technical limitations: at present only shortest path routing is supported; the configuration language is a simplified version of the one described in [8] (notably, only a single subnet can be specified for each customer site); flow tables are completely cleared and repopulated when a change of the routing paths occurs; finally, the VPN specification file is not watched for changes, which cannot therefore be deployed on-the-fly. All these limitations are implementation-specific and have negligible impact on the effectiveness of the demonstration.

[2] `http://mininet.org/`

[3] `http://openvswitch.org/`

III. Challenges

Deployment of our MPLS VPN controller only requires OpenFlow datapaths that can handle MPLS labels as per the OpenFlow 1.1 specification (released in February 2011). We tested a range of device models produced by different vendors, finding that a good fraction provides this support, and we reasonably expect this fraction to increase over time.

During our experiments we also impacted several technical issues that deserve further investigation. Some of them (e.g., maximum size of an MPLS label stack, low switching performance) are specific of Open vSwitch and are no longer relevant if datapaths are hardware devices. Others, such as the maximum capacity of flow tables, are inherent in any realization of the SDN architecture, but vendors are progressively adopting components that overcome these limits[4]. Unfortunately, certain issues are hard to address. For example, matching on the IP header of an MPLS-encapsulated packet is necessary to deliver packets to the correct customer site of a VPN (based on the destination subnet), but is hard to achieve in current datapath implementations: as a workaround, in our prototype we used VLAN tags as VPN identifiers in place of MPLS labels.

Acknowledgments

We would like to thank Unidata S.p.A. for allowing us to perform extensive tests on a range of SDN-capable hardware.

References

[1] 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.

[2] E. Rosen and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)," RFC 4364, Internet Engineering Task Force, Feb. 2006.

[3] G. Huston, "MPLS – Is the emperor clothed?" ISP Column, Oct 2001.

[4] 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

[5] D. Kreutz, F. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," vol. 103, no. 1, Dec 2014.

[6] 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.

[7] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, "On the feasibility of a consistent and fault-tolerant data store for SDNs," 2013.

[8] G. Lospoto, M. Rimondini, G. Vignoli, and G. Di Battista, "Rethinking virtual private networks in the software-defined era," in *Proc. IM*, 2015.

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

[10] 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

[4] See, e.g., `http://www.corsa.com/products/dp6420/`