Inter-controller communication in switching SDN architecture

Coroller Stevan, Goareguer Erwan, Pipereau Yohan
Télécom SudParis

Abstract - *Software Defined Networking can be used to perform traffic engineering and ensure Quality of Service in virtual SDNs, which are defined on top of wide layer 3 SDN infrastructures, mostly used by operators and huge service providers. In this survey, we explore different solutions to implement similar features (central management, virtual SDNs, controllers distribution, etc) in a small layer 2 research-oriented infrastructure.*

Keywords - Software Defined Networking, East/West Interface, Local Area Network

Table of content

Introduction

Our infrastructure is made up of three separate stacks of switches, each stack being connected to one SDN controller.

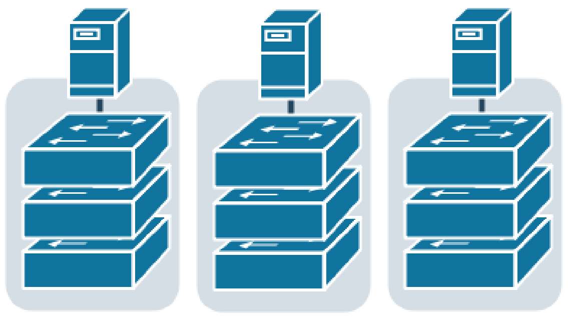


Figure 1: Initial infrastructure

Here are the questions we will address in this survey:

- How to centrally manage the whole infrastructure, using only one controller, in addition to the local control ensured by the three initial controllers ?
- How to allow users to use and configure a small part of the dataplane, without interfering with other users, nor being able to view the whole infrastructure topology ?
- How to allow such users to create an instance of the controller that best meets their requirements, independently of the controllers used to manage the whole network ?

All these questions rest upon one major question in SDN research : how to make two controllers communicate ?

This communication is not a mere peering of two controllers to allow the exchange of information. It is rather a connection of two domains through controllers to allow a remote control and view of a domain by another domain.

Thus, adding a controller to an existing infrastructure to control the data plane of a domain without being directly connected to it requires a two-step approach:

- The new controller needs to get the topology informations from the controllers directly connected to the data plane;
- And the new controller needs to be able to send OpenFlow rules to the data plane.

Communications between two controllers is called “East-West communication” in SDN litterature. It is important to keep in mind that at the time of writing, no standardized protocol exist to allow this communication.

In this article, we will give some details about clusterization of controllers. Most controllers use a different framework, thus a major problem is the incompatibility between two different model of controllers.

In order to get around this incompatibility, it is possible to rely on existing APIs such as :

- the Northbound REST API, which can retrieve a lot of information or send configuration but is totally different for every controllers. This could be implemented either as an integrated module on every controller or as a distributed scheduler given as an abstraction layer tailored to the specificity of every controllers' REST APIs.
- the Southbound BGP API, by using the BGP-LS extension to BGP to transfer topology information. Though, this protocol is limited to IGP topologies exchanges. In this configuration of switch domains, we do not have any Internal .
- the Southbound OpenFlow and LLDP API, which is the basis of network virtualization solutions like OpenFlow and OpenVirtex.

In most implementations, East-West interfaces have been built using Southbound API because, unlike the REST APIs, they were standardized.
Another determining element when comparing solutions is authenticating instances. Indeed, for REST queries this is pretty straightforward as they benefit from authentication schemes developed for other REST APIs problems.
However, it can become more complicated if you rely on a custom solution based on southbound APIs. This concern has not been studied thoroughly here.

As the previous solution are not convincing, we will dwell on virtualization techniques which offers another level of abstraction on the network. Here are the different techniques considered :

- Implementing an openflow controller which can act like an openflow proxy;
- Implementing the abstraction layer directly in the controller.

In this part, we will also go through various methods to slice the network.

I Clustering Controllers

Clustering controllers refers to the connection of multiple controllers of the same model (ex: clustering two ONOS instances). It is interesting in terms of:

- **scalability,** as creating multiple instances allows to divide the workload between all instances and attribute operations specific to each controller;
- **high availability,** to provide a fallback controller in case of crash;
- **data persistence,** as clusterization setup makes it possible to share data among multiple instances to prevent data loss.

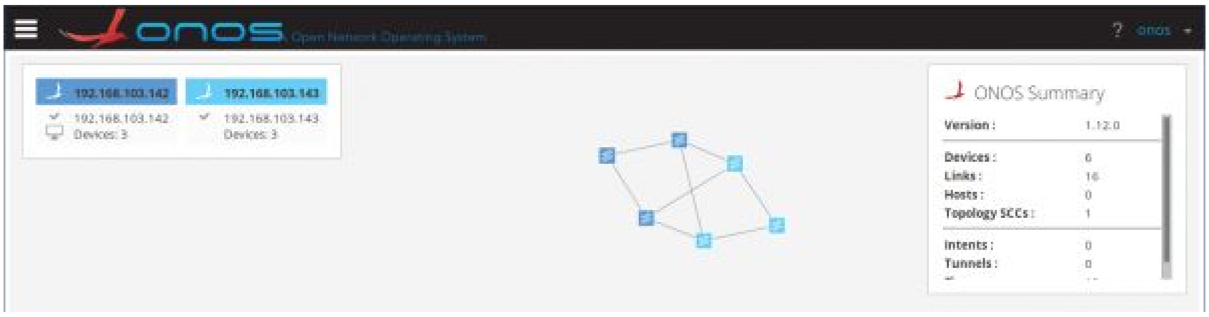
I.A ONOS cluster

ONOS offers a clustering solution to allow communication between two ONOS controllers. This feature can :

- retrieve information about the peer controller's topology;
- draw a graph of the topology of the two controllers and represent connection between the two domains;
- connect multiple controllers to a single switch to push Openflow rules;
- specify partitions to limit access to data to specific controllers.

Note that, if an extra controller is required, using ONOS clusterization solution imposes using nothing but ONOS controllers.

ONOS cluster nodes used to communicate with Hazelcast, but it now uses another framework for distributed systems named Atomix which uses TCP on port 9876.



```
Version :        1.12.0
Devices :        6
Links :          16
Hosts :          0
Topology SCCs :  1
Intents :        0
Tunnels :        0
```

Figure 2: Clustering two Onos instances

I.B OpenDayLight cluster

OpenDayLight cluterization solution uses TCP on port number 2550. OpenDayLight relies heavily on Akka for the clusterization, which is a toolkit for java and scala.

OpenDayLight does not simply offer a remote peering between controllers. It also distributes data on cluster nodes.

In order to better understand how this clusterization works, we will dig into OpenDayLight implementation a bit. OpenDayLight uses a middleware named MD-SAL consisting of two trees:

- **configuration Data Tree**, interface between users (consumer) and system services (providers);
- **operational Data Tree**, represents data state as perceived by components in the system.

OpenDayLight divides this datastore into shards. Then, subsets of shards are distributed to every cluster member. Shards are saved on persistent storage (hard drives) in order to reload them after the application is restarted. Shards are replicated on cluster nodes and expensive access to data (typically every remote data access) are logged.

The management of the data store and the construction of the cluster is handled by a cluster service. The effective storage relies on a three-phase commit protocol.

For the rest, opendaylight uses akka remoting to peer controllers and akka clustering to form the cluster abstraction.

I.C Conclusion

Onos and OpenDayLight relies on two different mechanisms to form their clusters. It is not possible to clusterize the two models of controllers simply. Nonetheless, clusterization between two controllers of the same model is highly optimized and it is recommended for performance and simplicity reasons.

II. NorthBound APIs

Controllers provide at least two Northbound APIs : REST and NETCONF (a NETwork CONFiguration protocol based on XML syntax). These northbound interfaces can be used to configure controllers and switches using Openflow rules.

II.A Inter-controller REST communication plugin

We have started studying a REST plugin to communicate between controllers using their own REST APIs. For communication between the same model of controller, clusterization is really more suitable so we will consider communication between two different model of controllers.

We need to use each controller’s REST API to create these plugins, and we need to implement a plugin for every controller; which requires a certain time to understand the controller implementation. Then, the problem is to create a long-term plugin for every controller. As controller tends to update quite frequently, these plugins would need to be updated with each controller update, which makes them difficult to maintain.. As controller REST APIs are not standardized, we would need to know the syntax for every controller we need to support. This is why the scheduler model is easier to upgrade and customize.

II.B Scheduler model

The scheduler model relies on an external application which can identify the model of the controller it needs to communicate with and then perform actions depending on its knowledge of the possible REST requests. This solution is still heavy as it grows proportionally with the number of controllers one wants to support. Yet, it can be developed in any language and it is really easy to modify the implementation in case the controller modifies its REST API.

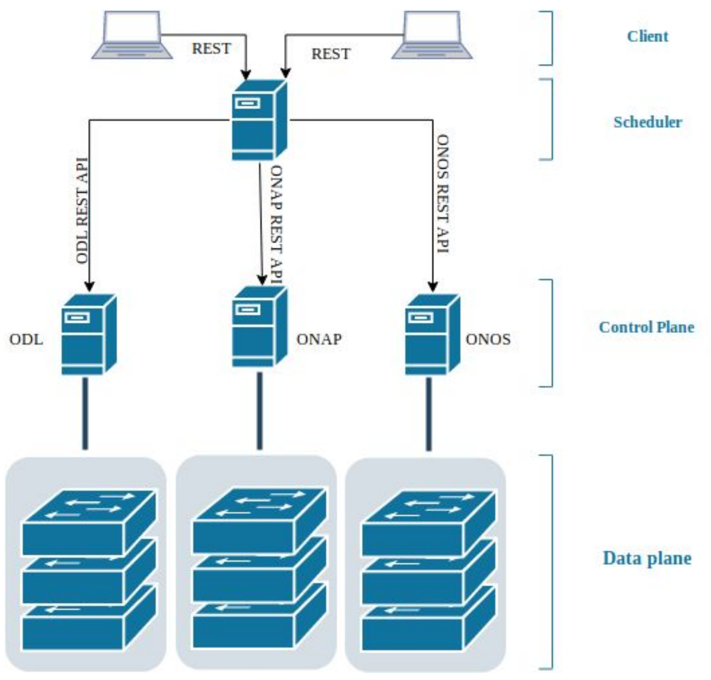


Figure 3: The scheduler model

III. Inter-cluster communication (with BGP-LS)

III.A BGP-LS protocol for inter-controller topology exchanges

BGP-LS stands for Border Gateway Protocol - Link State. This protocol is used to carry interior gateway protocol (IGP) link-state database through BGP.
It uses a new BGP Network Layer Reachability Information (NLRI) encoding format. BGP-LS has been designed to give visibility for LSDB (Link-State Database) and TED (Traffic Engineering Database) outside one area or Autonomous System AS.

BGP-LS implementation:
The new NLRI types introduced by BGP-LS are based on the TLV format (Type/Length/Value). The new NLRI are:
- Node NLRI
- Link NLRI
- Topology Prefix NLRI (IPv4/Ipv6)

The two BGP speaker open their communication with OPEN messages with the Link State Capability set.[1] Then the peers send BGP UPDATE messages containing the node and link NLRIs.

Most controllers including ONOS and OpenDayLight provide an implementation of BGP-LS.

[1] If you want to use a free AS number, you can pick one between 64512 and 65534.

BGP-LS usage in East-West communication:

BGP-LS is designed to exchange links and nodes information of an IGP[2] topology. This means that in our case BGP-LS cannot be used to send switch topology as it is not supported by the protocol.

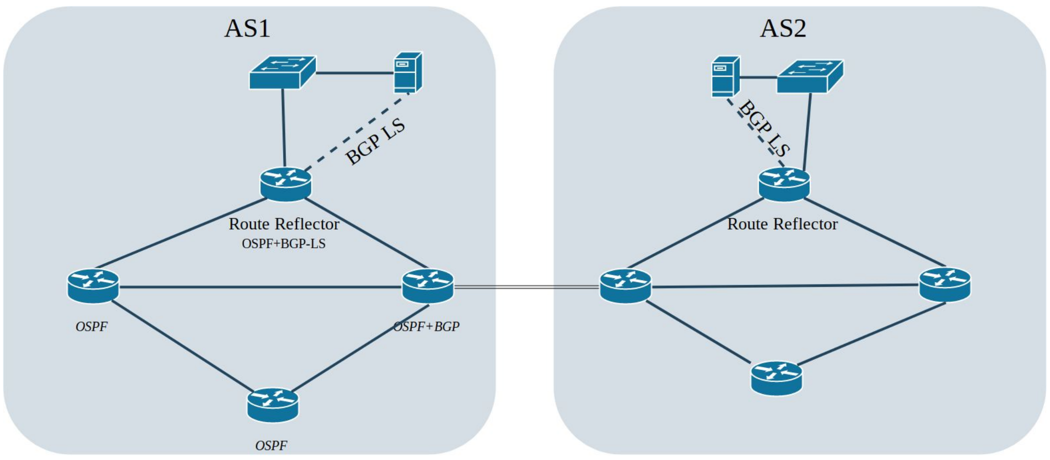


Figure 4 : BGP-LS architecture with SDN controllers

BGP-LS is well suited if one wants to exchange topology between two controllers in two different AS (Figure 4). The architecture would include at least an IGP area for each AS with a router supporting eBGP and the IGP protocol to communicate with the other AS. Inside each IGP area, there is a Route Reflector supporting BGP-LS and the IGP protocol. This route reflector is used by the controller to retrieve the topology using a BGP-LS tunnel.

III.B ONOS-ICONA and ODL-SDNi

ONOS-ICONA and ODL-SDNi are two plugins to share topology information between two AS. These plugins are used for inter-cluster communications and they use an East-West Interface.

In Figure 4 you can see how ONOS ICONA application allows communication between two Cluster. The interlink typically uses BGP-LS to exchange topologies.

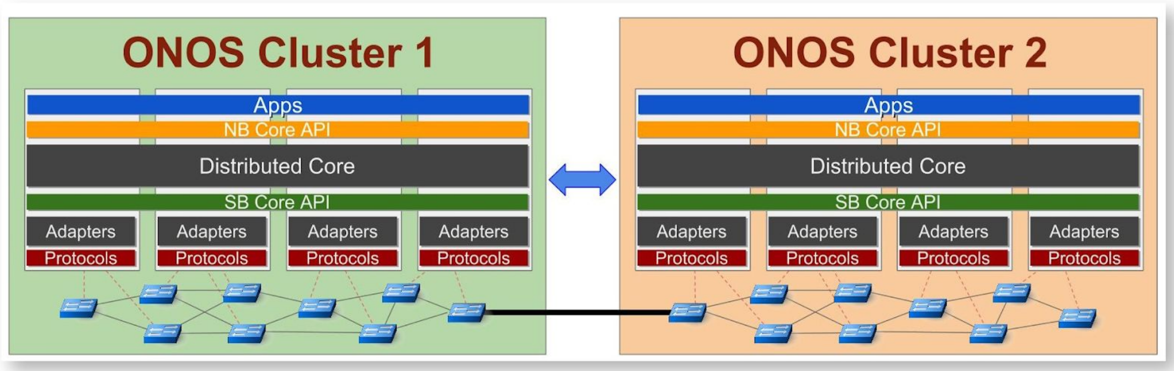


Figure 4: Inter-Cluster communication with ONOS-ICONA [7]

[2] OSPF, ISIS, …

These plugins features :

- sending and receiving network topologies;
- handling intents from remote controllers.

III.D Conclusion

In order to exchange **IGP** topology informations between two controllers, it is possible to use BGP-LS, a BGP extension. This solution is standardized and only requires a BGP-LS implementation of the protocol. It does not rely on the controller model which let the user free of the controller solution he wants to use. On ONOS and OpenDayLight, a tunnel can be easily built using REST API. Moreover, this solution allows the user to send information for inter-AS exchanges, so this solution does not suit a LAN topology.

However, this raises the question : How does OpenDayLight detect a Layer 2 topology? It appears to use LLDP, which brought us to consider building an LLDP proxy which would be connected to all controllers who want to get the layer 2 topology of a group of switches. We could then define the slicing of the group of switches a controller should be able to see. This bring us to the virtualisation part.

IV. Network Virtualization

Network virtualization is a generic term which can be used at various levels:

- **node-level virtualization :** with the use of an OpenVSwitch;
- **path-level virtualization :** used for node connection, it uses tunnel technologies;
- **domain-level virtualization :** used for tenant connection, generally it uses an overlay network.

Then, there are two approaches which can both be found in various technologies:
Hop-by-hop approach : Flows are handled by intermediate equipments and not just by endpoints. hop-by-hop requires a memory to buffer data on each node. Every intermediate verifies integrity. This approach is suitable for *Traffic Engineering* and *QoS*.
Overlay approach : A virtual network in which the separation of tenants[3] is hidden from the underlying physical infrastructure. This approach is suitable for *scalability*. Network overlays are used to build a virtualized network.
Both approaches are not incompatible!

IV.A FlowVisor

IV.A.1 What is it and how does it work ?

FlowVisor is a special SDN controller which aims at separating one physical SDN into multiple (potentially overlapping) virtual slices, a.k.a. virtual SDNs. “It acts as a controller for the switches and as an OpenFlow switch for the controllers”[1]. “The Prefix-based Layer 2 Network Virtualization (L2PNV) is an extension of FlowVisor, with the main objective of providing a level 2 virtualization engine without using VLAN, instead basing the virtual networks created in Media Access Control (MAC) on the address of origin and destination”[1].

From FlowVisor’s github project[17]:

- FlowVisor is a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers;
- FlowVisor creates rich slices of network resources and delegates control of each slice to a different controller;
- Slices can be defined by any combination of switch ports (layer 1), src/dst ethernet address or type (layer 2), src/dst IP address or type (layer 3), and src/dst TCP/UDP port or ICMP code/type (layer 4);
- FlowVisor enforces isolation between each slice, i.e. one slice cannot control another's traffic.

[3] set of resources (compute, network, storage, etc.) assigned to a group of users.

Although FlowVisor setup may be quite long, as it requires a lot of manual configuration, some tutorials are available on the internet to help network administrators to deploy FlowVisor[18].

In short, FlowVisor acts as an OpenFlow proxy that transparently rewrites and forwards OpenFlow messages, according to the slice policy[5].

IV.A.2 Does it match our needs ?

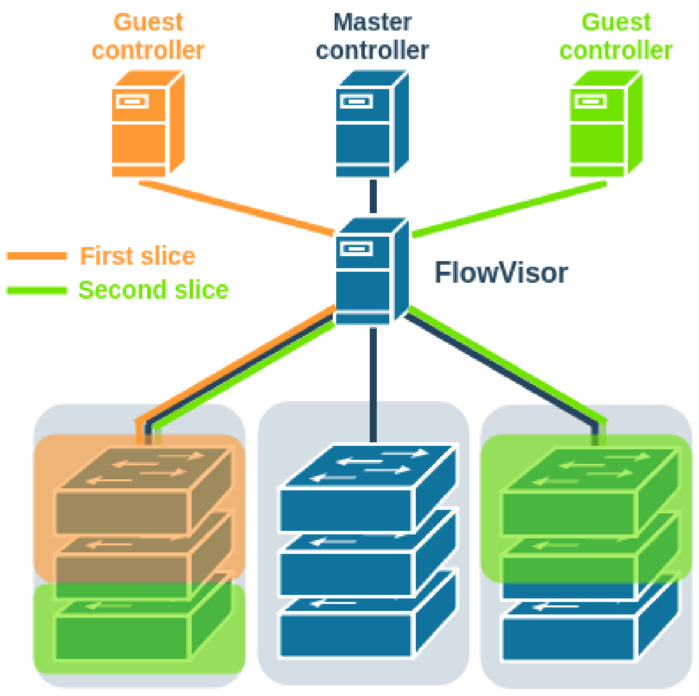


Figure 5: A FlowVisor infrastructure example

FlowVisor makes it possible to define multiple, isolated, possibly overlapping slices, or virtual networks. Thus, FlowVisor addresses all of our initial issues:

- We can define a single slice over the whole network, making it possible for one master controller to manage the whole network (default slice in above figure);
- It is possible to give a guest controller access to a part of the infrastructure, without interfering with other slices: guests can only configure switches' ports they are given access to, and controllers won't receive traffic from other slices' ports;
- FlowVisor is compatible with every controller that supports OpenFlow, which means guests can use the controller of their choice, independently of other controllers.

IV.A.3 Limits of FlowVisor and conclusion

While still being effective and functional, **FlowVisor is not maintained anymore and should not be run in production networks**[2] for three main reasons:

- FlowVisor only supports OpenFlow 1.0, while OpenFlow 1.5 has been released for more than two years now.
- It exists several vulnerabilities concerning slices' isolation[3][4].
- Flowvisor lacks scalability "since the recommended requirements for operating the FlowVisor are 4 processor cores and a 4GB Java heap" [1].

Yet, **FlowVisor has been used from 2009 and is still being used in multiple research networks** where slices' isolation vulnerabilities may not be a big concern.

Despite these counter-arguments, FlowVisor is the only solution that answers all of our needs. This is why **it remains one of the favored solutions** for the initial research infrastructure. In addition, the FlowVisor project is still open to contributions, so it is possible, and we would advise patching slices' isolation vulnerabilities and adding OpenFlow 1.3 or 1.5 support, before using it in production networks.

IV.B OpenVirteX

OpenVirtex (OVX) works very similarly to FlowVisor: it creates virtual networks called tenants, on top of your physical infrastructure, which allows you to control each part of the data plane with the controller of your choice. OVX is a special controller that creates a virtualization layer between data and control planes, while remaining transparent for both planes.

From OVX's FAQ: "In short, OpenVirteX gives you a full packet header space (a virtual copy for your own virtual network), whereas FlowVisor lets you divide up a single packet header space into subsets to assign to your slices." Thus, **OVX allows more freedom than FlowVisor** for network addressing, and let you create virtual topologies which, contrary to slices, do not have to be exactly identical to the underlying physical topology.

Just like FlowVisor, OVX setup can be quite long, but very complete tutorials exist[19], and OVX is also used in research networks[6].

In short, **OVX suffers from the same counter-arguments than FlowVisor**:

- Recommended requirements are 4 processor cores and 4GB java heap.
- OVX uses OpenFlowJ, an OpenFlow Java API which supports newest versions of OpenFlow, but the OpenFlowJ version used in OVX doesn't support OF 1.3 or more recent versions.

But, contrary to FlowVisor, **no resources isolation vulnerability has been discovered in OpenVirteX**, which makes OVX a safer solution than FlowVisor.

In conclusion, **OVX can be used safely in production networks**, and should require less contribution than FlowVisor to add support for newer versions of OpenFlow.

IV.C OpenDayLight Virtual Tenant Network

IV.C.1 What is it and how does it work ?

The OpenDayLight project (ODL) also developed a virtualization technology called Virtual Tenant network (VTN), integrated into their SDN controller as a plugin. VTN is separated into two applications: the VTN manager plugin for OpenDayLight, which exposes a REST API to manage VTN components, and an external application: the VTN coordinator, that exposes a REST interface for users to communicate with the VTN manager.

Just like OpenVirteX tenants, **VTN tenants perform a complete virtualization of the network**: the administrator defines a VTN policy in which he creates virtual switches and virtual bridges, and then connects these virtual equipments to physical ones. This kind of virtualization allows for the creation of virtual topologies which do not depend on the physical topology. In addition, **VTN can quickly map tenants to existing VLANs**, in order to isolate VLAN traffic into separated tenants.

Multiple OpenDayLight instances can share the same VTN policy, and additional controllers can be added to this VTN policy, which means VTN is a highly scalable technology. In our context, this means we can create an OpenDayLight instance for each group of switches, and each of these instances would have global vision of the whole virtual topology, which allows **distributed management of the network**. In addition, multiple VTN policies can be defined on top of one physical infrastructure, as indicated in figure 3. Traffic from one tenant can be completely isolated from another tenant, which allows multiple users to use the same underlying physical infrastructure without interfering with each other.

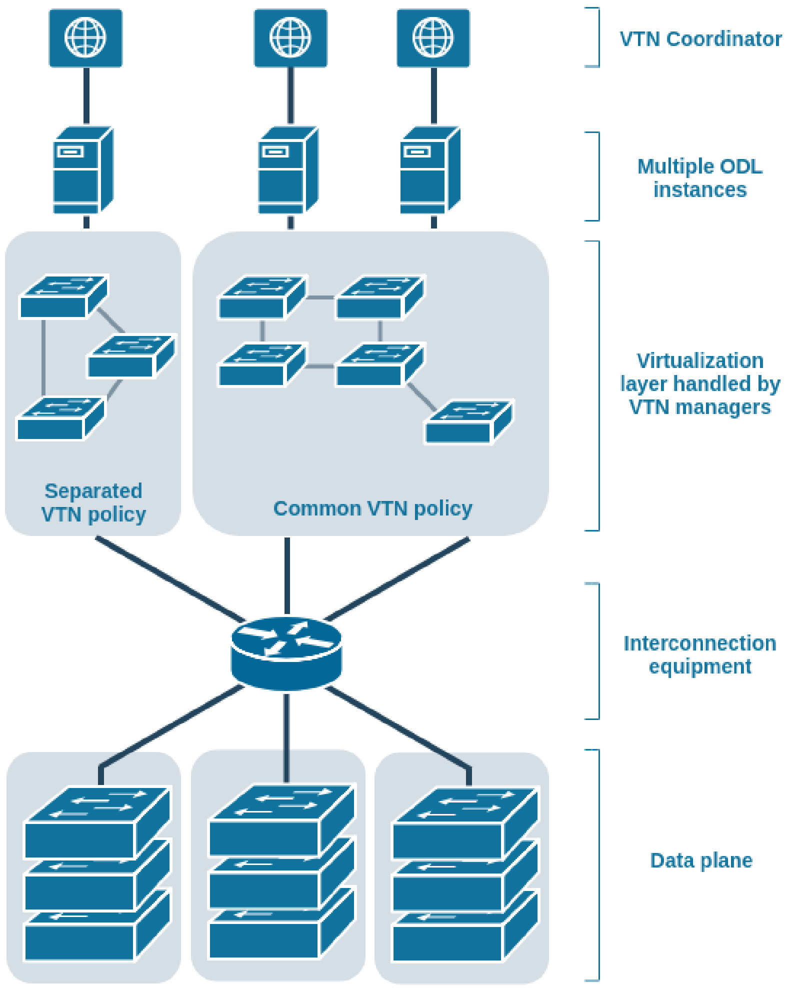


Figure 6: A VTN infrastructure example

IV.C.2 Does it match our needs ?

VTN perfectly answers our first question: a single controller can use the whole topology, while some other controllers can define smaller tenants, which cover a smaller part of the physical data plane.

However, a SDN controller can only connect to tenants thanks to the VTN manager plugin, designed for OpenDayLight. As a consequence, no other controller than ODL can connect to tenants, unless VTN manager plugins are developed for other controllers, which is highly unlikely to happen.

In addition, local controllers who only use a small part of the data plane with their tenant still have access to the whole physical topology, which means they can add other switches to their tenant and gain access to the rest of the data plane.

As a conclusion, VTN is one of the most mature technologies among those we studied in terms of SDN virtualization, but it has been designed to deal with issues of wide layer 3 infrastructures, used by operators and data centers, rather than to deal with small layer 2 infrastructure issues like those we are addressing in this survey.

Conclusion

There are many techniques available to answer the initial problem, here is some advice to choose the best one.

If you can use the same model of controller, controller clustering provides most of the features required with replication, high availability and access control. It is easy to setup and it can be automated with REST API on most controllers.

If the network you want to exchange with uses an IGP, we would definitely recommend the use of BGP-LS and allegedly an inter-cluster plugin if you are using cluster in your AS.

If you want to use different models of controller, you will need to use virtualization. OpenVirteX appears to offer better performance and security with support for newer version of OpenFlow than FlowVisor. Though this type of controller seems not to be supported anymore, it offers interesting features, it could be interesting to fork this project to develop something which suits your needs.

Finally, VTN can answer this problem though it is strongly linked with OpenStack and some hacking might be needed to make it work properly for your infrastructure. Moreover, it works only for OpenDayLight, and it is more directed towards layer 3 infrastructures.

Future Works

Though this project has presented various solutions and techniques, no performance comparison has been made between controllers like ONOS and OpenDayLight but it is very likely that in some situations one should be prefered. In particular regarding the clusterization choices, it is possible to dig into the algorithm used for clusterization and thus propose a suitable controller to use for specific applications.

We did not have time to look at the vulnerabilities reported for Flowvisor. There are two possible scenarios. Either these scenario concern directly the implementation of the controller and it would be a good idea to create a new ‘LLDP/OpenFlow Proxy’ similar to FlowVisor. Else, if vulnerabilities come from the concept of using a ‘LLDP/OpenFlow Proxy’, it would be interesting to give a feedback to the community about why this model can not be used anymore.

This paper has not emphasized the security concerns of communication between controllers. Though, it is relevant to look at possible Access control solutions for the different solutions we have listed such as VTN, REST APIs, BGP-LS, ...

References

[1] *NVP: A Network Virtualization Proxy for Software Defined Networking*, B. Pinheiro, E. Cerqueira, A. Abelem, **October 2016**

[2] *OpenFlow network virtualization with FlowVisor*, **2013**, by Sebastian Dabkiewicz.

[3] *Vulnerabilities and solutions for isolation in FlowVisor-based virtual network environments*, **2015**, by Victor T. Costa & Luís Henrique M. K. Costa.

[4] *FlowVisor Vulnerability Analysis*, **2017**, by Ying Qian, Wanqing You and Kai Qian.

[5] *FlowVisor: A Network Virtualization Layer*, **2011** by Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, Guru Parulkar

[6] *An Intent-based Network Virtualization Platform for SDN*, **2016**, by Yoonseon Han, Jian Li, Doan Hoang, Jae-Hyoung Yoo, and James Won-Ki Hong.

[7] *ICONA (Inter Cluster ONOS Network Application)*, **2016**, https://wiki.onosproject.org/pages/viewpage.action?pageId=4162523

[8] ICONA Gerrit for code review, **2016**, https://gerrit.onosproject.org/#/c/9534/

[9] ICONA: Inter Cluster Onos Network Application, Matteo Gerola, Michele Santuari, Elio Salvadori, Stefano Salsano, Pier Luigi Ventre, Mauro Campanella, Francesco Lombardo, Giuseppe Siracusano, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7116173

[10] *OpenDayLight wiki*, https://wiki.opendaylight.org/view/Main_Page

[11] *ONOS wiki*, https://wiki.onosproject.org/display/ONOS/Wiki+Home

[12] *A Hierarchical Control Plane for Software-Defined Networks-based Industrial Control Systems*, Béla Genge and Piroska Haller, **2016**

[13] *Distributed Hierarchical Control Plane of Software Defined Networking*, Prashant D. Bhole, Dinesh D. Puri, **2015**

[14] *ICONA: Inter-Cluster ONOS Network Application*, Francesco Lucrezia, Michele Santuari, Matteo Gerola

[15] *Inter-SDN Controller Communication: Using Border Gateway Protocol,* TATA Consultancy services, **2014**

[16] Towards Distributed Hierarchical SDN Control Plane, A. Koshibe, A. Baid and I. Seskar

[17] *FlowVisor GitHub project*, https://github.com/OPENNETWORKINGLAB/flowvisor/

[18] *FlowVisor configuration exercise,*
https://github.com/onstutorial/onstutorial/wiki/Flowvisor-Exercise

[19] *OpenVirtex official tutorial*, https://ovx.onlab.us/getting-started/tutorial/