Network Virtualization and Resource Allocation in OpenFlow-based Wide Area Networks

Pontus Sköldström
Acreo AB, Kista, Sweden
Email: ponsko@acreo.se

Kiran Yedavalli
Ericsson Research, San Jose, CA, USA
Email: kiran.yedavalli@ericsson.com

***Abstract*—In this paper we investigate a number of network virtualization models for OpenFlow networks. Specifically, we investigate three sub-modules of the system – the control channel and the software and hardware parts of an OpenFlow switch. We propose a number of extensions to the OpenFlow specification for this purpose and present a model of a full solution that compared to existing systems provides stricter isolation between different virtual networks while at the same time providing more flexibility.**

I. Introduction

OpenFlow [1] protocol based network architectures is an emerging area of research for service provider networks. At the core of this emerging architecture is the separation and centralization of the control plane from the forwarding elements in the network as opposed to the distributed control plane of current networks. A centralized "controller" uses OpenFlow protocol to control the forwarding elements to enable different network applications and services.

Network virtualization consists of multiple techniques applicable in many situations e.g. to improve network resource utilization through sharing, separation of traffic between different entities, and to simplify network management. Many techniques have been developed to virtualize different parts of the network - VPNs can create end-to-end virtual paths, with VRFs and VLANs its possible to create multiple forwarding tables in a single switch/router, and different tunneling techniques can be used to create multiple virtual links to share a single physical link.

An environment such as multi-tenant WAN or a fixed-mobile converged network, where multiple customers and competitors share a single physical network [2], [3], [4], through network virtualization, imposes many requirements on the virtualization system. Not only must the system make sure that the traffic is separated between customers, ideally no information should be able to leak between the virtual "partitions" (unless previously agreed), it must also enforce any established SLAs. In addition the system should be as flexible as possible in order to avoid costly interaction and coordination between the parties involved, reducing operational costs.

In this paper we present our work on network virtualization in the context of OpenFlow-based centralized control of service provider networks. We present an unified system level view of network virtualization encompassing the central controller, the datapath of the forwarding elements in the physical network, and the control channel between the controller and the forwarding elements. Further we present mechanisms to enforce resource allocation among the different "tenants" of the network in all the above three components of the system.

Network virtualization in the context of data center networks is well known and has been extensively studied in the context of OpenFlow [5] [6] [7]. Even though some of the ideas presented in these references are generic enough to be applied to service provider networks, each one of them provide a partial view of the network virtualization system. For example, while [7] and [8] describe control side virtualization, [5] describes forwarding datapath virtualization in the data center context.

The rest of the paper is organized as follows: in Section II we present five different virtualization models. We discuss datapath virtualization and resource allocation issues in Section III and control plane virtualization and resource allocation issues in Section IV. We describe OpenFlow protocol translations in the context of network virtualization in Section V and present a system level virtualization mechanism in Section VI before concluding in Section VII.

II. Virtualization models

A number of virtualization models in the context of OpenFlow can be envisioned. Essential to the virtualization process is a translation unit that maps identifiers between the real physical network and the virtual ones, similar to a memory management unit in a computer or one of many kinds of hypervisors used on virtual machine hosts. As with these hypervisors there are many different options on how and where to implement the translation unit, but it has to be located somewhere between the application logic and the physical hardware. In Figure 1 five options are shown:

ⓐ The Flowvisor [7] approach where the translation unit acts as a protocol proxy and is shared between multiple switches and multiple controllers.

ⓑ The translation unit is placed in the switch, it distinguishes between different connected controllers and performs the translation at the protocol level inside the *OpenFlow instance* on the switch.

ⓒ Each switch runs multiple OpenFlow instances, one per connected controller, and the translation is performed between each OpenFlow instance and the fast-path forwarding hardware.

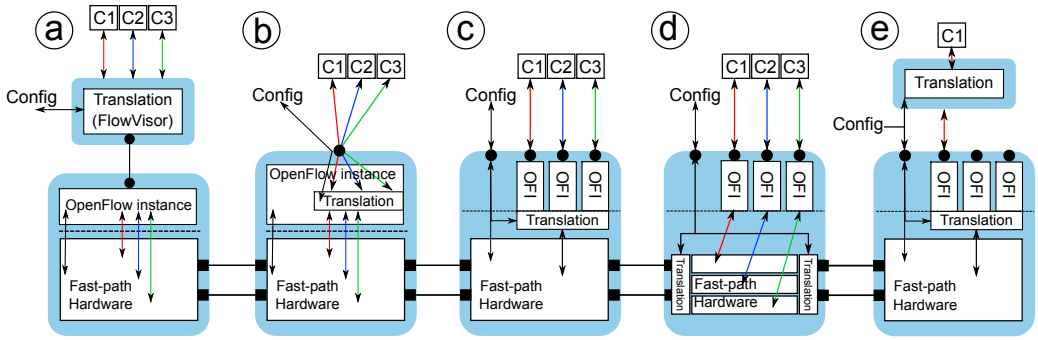

Fig. 1: Different placements of the translating functions in the system.

ⓓ Each switch runs multiple OpenFlow instances and the switch's datapath is partitioned into several parallel datapaths, one per instance. Translation is performed by influencing how ports are connected to the different parallel datapaths - the physical ports may be partitioned as well by using, for example using VLAN or MPLS labels.

ⓔ A model with multiple translation units, one responsible for virtualizing on the switch level, and another responsible for connecting multiple virtual switches and creating a virtual network representation. The virtual network topology can be partitioned, aggregated, or extended. Additionally, virtual links can be created through tunnels managed by the virtual topology translation unit and presented as physical links, similar to what is done in [5] and [8] (this can also be done on the switch level).

III. Datapath Virtualization and Resource Allocation

When implementing virtualization in an multi-tenant WAN or a fixed-mobile converged network environment, several resources of an OpenFlow switch [9] have to be separated and isolated from each other - FlowTable(s), Group Table(s), physical links, or parts of them, and any processing functionality (e.g. OAM processing or tunnel termination implemented as vendor extensions). Isolation includes not only direct separation, such as preventing a controller from updating a FlowTable belonging to another controller, but also indirect separation such as preventing one VN to use more than its allocated share of e.g. bandwidth.

A. Separation

To keep the Virtual Networks (VNs) separated at the link level there are two approaches - partitioning and encapsulation. With partitioning the link is split into multiple partitions by slicing the total flow-space, one partition per virtual network, for example by reserving a part of the total VLAN or IP address range per VN. To ensure that this separation holds all the VNs must be partitioned based on the same parameters of the flow-space, e.g. VLANs or IP addresses. However, these restrictions are only strictly link-local, that is for example, if the flow-space is partitioned based on the VLAN ID address space, the same VLAN ID could be reused for a different VN on the same switch.

With encapsulation, each VN in the network is assigned a link-local encapsulation ID, called a Virtual Network ID (VNID), a MPLS label for example, to which the VN traffic will be mapped and encapsulated when it traverses a shared link. Before transmitting a packet each switch adds a link-local encapsulation per VN, the encapsulation is used at the receiver to assign the packet to a certain VN and then removed before processing. This leaves each VN with a complete flow-space that has not been reduced to enforce separation, creating more flexibility at the cost of the extra processing needed to add and remove the encapsulation per hop. It also requires the switches to support an encapsulation protocol able to carry an Ethernet frame such as L2TP or MPLS.

When it comes to FlowTable(s) there are two different approaches to separation as well - flow-space partitioning or table partitioning. With flow-space partitioning the same mechanisms are applied as in the case of the partitioned link sharing above, each VN is only allowed to insert flow entries that **1)** have a match that is within the specific flow-space assigned to the particular VN, and **2)** does not have Actions that can move the packet into a different VN, for example by changing the field used to distinguish the VNs. Again, this approach has similar drawbacks as in the link case, mechanisms have to be devised to make sure that the partitioned flow-spaces do not overlap, which causes a loss of flexibility in the flow-space assignment. Additionally, enforcing that an Action does not move the packet to another virtual network could be difficult in some cases, for example if you de-capsulate a tunneled packet.

With table partitioning the FlowTable is divided into multiple tables, logically or physically. For example, OpenFlow specification 1.1 already mandates multiple FlowTables, thus, each VN could be assigned a group of distinct FlowTables from the available ones. Each VN could be assigned, say, up to 10 tables and this separation is easy to enforce through translation. While the protocol cannot address more than 2^8 tables this does not limit the potential number of VNs to 256. The FlowTable identifier has a local meaning in each OpenFlow protocol session, if the switch hardware can support more than 256 FlowTables they can all be used through translation, but *not* by a single OpenFlow session.

An example of table partitioning can be seen in figure 2 in

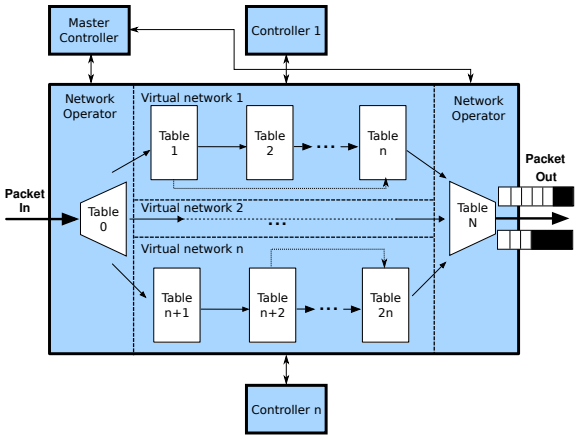

Fig. 2: FlowTable splitting by reserving a number of tables to different VNs. The ingress and egress tables are controlled by the virtualization system and used to classify incoming packets and apply per VN processing, e.g. QoS.

which two tables (the ingress and egress) have been reserved for use by a *Master controller*. The controller programs the ingress table with classification rules that determines which VN the incoming packets belong to and forwards them to the appropriate FlowTable for processing. Once the packets are processed in their allocated table(s) they are sent to the egress table for any needed post-processing, for example enqueuing them into different queues depending on which VN they belong to. Either there is a physical separation of tables or a translation unit is responsible for enforcing that the other controllers are only allowed to read and write the tables allocated to them. By mandating that all the flow packets pass through the egress FlowTable for re-application of per-VN encapsulation, the VN separation can be enforced throughout the network.

There are several ways of implementing the above ideas through certain modifications to the current OpenFlow semantics by restricting the use of certain Instructions/Actions and adding new ones. For example, the **Goto-Table** instruction can be used to make sure that the packets pass through the egress FlowTable as discussed above. Another example is to make sure that the Action-Set are implemented at the egress FlowTable and the Group Table is downstream to the egress table.

B. Resource Allocation

So far in this section we have only discussed the ability to distinguish between different VNs on the links and inside the switches, now we turn to enforcing resource usage. Enforcement of resource use is only needed if there is a competition for resources, and there are a number of components in a switch where competition for bandwidth and buffer space may occur. [10] discusses the common origins of congestion in a typical 10 Gigabit Ethernet switch; these will likely be similar in an OpenFlow switch, but of course depends on the particular implementation. Several of the stress points discussed are difficult to control, for example stress at the ingress packet buffer and packet classifier depends heavily on the complexity of and rate of incoming traffic. However, since the packet classifier specified by the OpenFlow specific is relatively simple, congestion in the classifier is quite unlikely. Multicast traffic may be a "good" source of congestion, not because of its complexity but rather since a single incoming packet multiplies and might end up in several line-card level queues as well as port queues.

By applying the classical QoS tools of classification, metering, coloring, policing, shaping, and scheduling at the ingress and egress ports we can limit congestion caused by high rate of incoming traffic, as well as limit the impact on downstream switches. In the context of virtualization the same QoS tools can be applied to ensure fair resource usage and isolation by different VNs, if designed with virtualization in mind.

The current OpenFlow 1.1 specification does not provide much support for QoS tools, however, some QoS support can be implemented by using one among the many FlowTables to perform classification into different QoS classes and the metadata field can be used to carry the classification through the pipeline and finally used to chose an appropriate port queue. In any case, assuming a virtualization model where OpenFlow is used to enforce fair resource allocation between different VNs requires extensions to the OpenFlow QoS support.

We suggest extensions to the OpenFlow 1.1 protocol to improve the QoS for fair resource allocation among VNs, first some extensions that could be implemented as Actions, and then improvements to the existing queuing mechanism:
Classification could be improved by adding an action that can set the service class in a variable in the per packet context, another option is to use part of the metadata to hold that information. In the first case it is easier to use the information in other QoS actions, but not possible to currently match on it in the FlowTables.
Metering and coloring could be implemented as an action using, for example, two-rate token buckets or virtual scheduling to assign colors. Color assignment could, like service class, be stored in the metadata field or in a separate variable. Further, meters could be placed in an hierarchy where they can "lend" bandwidth from the higher level meters (similar to [11], [12]), that would make it possible to guarantee a certain amount of policing bandwidth per VN which the corresponding tenant or operator may further subdivide in their own classes.
Policing could be implemented as a specific action that takes into account service class and packet color to drop packets that violate the bandwidth agreement, or through matching and dropping packets in a FlowTable. Since shaping requires buffering of violating packets it may be complicated to implement as an action since the packet has to leave the processing pipeline until it is no longer in violation, and then put back into the pipeline at the point it was put on hold.
Marking is already supported, however, a switch's internal priority mark would be useful in those cases where a packet should be given priority but not be modified.
Queuing and shaping could be improved by allowing queues

to be attached to other queues, not only prepended to ports. Similar to hierarchical metering, hierarchical queuing would make it possible to construct per VN egress traffic shaping that could be further subdivided in individual VNs.

IV. Control Channel Virtualization and Resource Allocation

To ensure fair allocation of resources on the control channel, the network that connects the controller to the switches (be it out-of-band or in-band) has to support mechanisms such as rate-limiting to prevent one VN from using all bandwidth on the control channel, for example by transmission of a large number of control packets to/from the controllers belonging to other VNs. Additionally, in both out-of-band and in-band scenarios, the control traffic may be competing with other traffic for network resources, for example management/configuration traffic using the same out-of-band network, or data traffic in the case of in-band control channels. In both cases network wide QoS reservations for the control traffic can solve the problem and provide fairness in use of the control channel resources.

Depending on the virtualization model used, local (per VN in the controllers and switches) rate-limiting may still be necessary. When using a Flowvisor [7] the control channels for the different VNs are multiplexed over a single TCP/SSL connection, it is impossible for intermediate network nodes to look inside it to differentiate between different control channels and enforce QoS for the different controllers. On the other hand, if the switches allow the different controllers to connect and control the VNs using e.g. different source or destination IP addresses or port numbers, the control traffic network can easily distinguish between the different connections and thus enforce QoS policies.

One could argue that the built-in TCP flow control is enough since it will push the TCP sessions towards an equilibrium of used bandwidth. However, it is likely that different tenants have different requirements on the amount of bandwidth needed in the control channel. For example, a tenant that needs to analyze data traffic in the controller will probably consume more bandwidth than one that does not, TCP flow control is not enough to satisfy such requirements.

A. Out-of-Band vs. In-Band Control Channels

An out-of-band control network has many advantages compared to the in-band counterpart in the OpenFlow case. It is simpler and easier to design in a reliable fashion. However, out-of-band control network might not be possible in some scenarios, for example, in a widely distributed OLTs in service provider access networks. Even if an out-of-band network is possible, it would be more expensive than an in-band solution due to an entire extra network and extra ports on the hardware, in addition to increased configuration complexity. Given these considerations, in-band control channels are as important as out-of-band channels, if not more.

In an in-band scenario the control channel is vulnerable to misconfigured FlowTable entries since all packets, including control packets, traverse them. For example, the in-band solution implemented by OpenvSwitch [13] requires the operator to pre-configure each switch with neighboring switches' as well as its own MAC addresses and the controllers IP addresses in order to install "invisible" FlowEntries that send control traffic to be processed by the local (non-OpenFlow) networking stack. This solution is quite fragile and configuration intensive, which could be operationally expensive in a large network. In the in-band case it is very important that sufficient QoS support is available so that control traffic can be guaranteed priority over data traffic. Additionally, QoS should not only be applied to control traffic entering the switch from the in-band channel but also to outgoing traffic, without the traffic necessarily passing through the FlowTable when leaving the switch. This may require that a invisible / non-mutable high-priority queue be reserved for traffic from the local network stack.

V. OpenFlow Instance and Translation

By *OpenFlow Instance* (OFI) we refer to the software part of the OpenFlow implementation of a switch that is running on the local switch CPU which is responsible for processing OpenFlow messages and performing everything else that is not handled by the hardware itself (e.g. implementing some Actions, keeping statistics etc). Depending on the virtualization model and the implementation, as discussed in Section II, it may also contain a translation unit that has to keep the different connected controllers separated. Running multiple OFIs as separate processes significantly reduces the complexity of serving multiple VNs and corresponding multiple controllers through a single OFI process. For further isolation, and thus appropriate allocation of limited switch CPU resources to the different OFI instances corresponding to different VN control channels, mechanisms such as light weight Linux Containers [14] that can streamline access to memory and processing power could be used.

LOCAL virtual ports can be used to send flows to the switch's local networking stack, e.g. to implement an in-band control network. In order to support the multiple channels corresponding to each OFI, a pre-determined range of LOCAL port numbers could be used, one for each OFI. Alternatively, if the different OFI instances are multiplexed through a single LOCAL virtual port using their respective IP addresses, it could lead to fairness and starvation issues.

The translation unit within the OFI is responsible for mapping the virtual identifiers into their physical counterparts as well as enforcing the separation between the different virtual "identifier spaces". The complexity of the translation unit depends on the choice of the virtualization model. For example, if per-VN encapsulation is used, similar to model ⓓ in Figure 1, the payload of the PACKET_IN and PACKET_OUT OpenFlow messages needs to be modified based on the port and VN they are associated with, and they should have the correct QoS markings applied for fair resource allocation. In contrast, correct translation and mapping of per-VN port numbers and port counters apply all the virtualization models

discussed in Figure 1. For example, for statistics monitoring, per-VN port counters could be implemented through per-VN FlowTable entries [15].

VI. A Full Virtualization System

In this section we combine the ideas discussed in the previous sections to present a complete network virtualization system that provides fair resource allocation both in the datapath and on the control channel, that is flexible and requires a low degree of interaction between the different virtual networks. Figure 3 illustrates such a system.

The datapath is built from model ⓒ to which we add encapsulation-based link separation combined with FlowTable partitioning. This results in a very flexible datapath virtualization system through the application of resource allocation and QoS mechanisms discussed in Section III-B. This can be achieved by dedicating the ingress and egress FlowTables on the datapath for this purpose. Link separation can be achieved, for example, through specific, pre-determined MPLS labels where at each link they are used to encode the VNID[1].

The OFIs could be separated through the OS's virtualization mechanisms and they connect to the datapath hardware through the translation unit. The OFI that handles the *Master Controller* (MC), usually operated by the network owner, configures the translation unit as well as all the ingress and egress FlowTables (shown as MC tables in Figure 3). This OFI should run at a higher priority on the local switch CPU so that the MC communications with the switch are given higher priority in order to handle special events such as failure scenarios. Configuration of the MC tables can be performed through the existing OpenFlow protocol, with minor restrictions enforced for each VN controller. The translation unit also needs configuration in order define the different virtual networks, and this can be achieved through non-OpenFlow channels or by OpenFlow protocol extensions containing the definition of a particular VN on a switch (e.g. VNID, MPLS labels belonging to it, ports belonging to it, etc).

The control network could be implemented as an in-band IP network and separated into two parts, the master control network and the virtual control channels. The virtual control channels are managed by the Master Controller which is responsible for routing, establishing QoS, and reconfiguring the control network in case of failures, topology changes etc. The master control network however needs to be bootstrapped through manual configuration or via a dynamic discovery protocol, which is area for our future work.

VII. Conclusion

In this paper we have reviewed a number of different virtualization models and pointed out areas in the current OpenFlow specification that need enhancements. Further, we have suggested improvements to the protocol, either through extensions or through restricting the usage of certain functionalities. Using the suggested improvements we have presented a highly flexible virtualization system that enables fair allocation of switch datapath and the virtual network control channel resources. As part of our future work we are in the process of implementing this virtualization system using the OpenFlow 1.1 reference switch implementation available from [9].

[1]MPLS replaces the existing EtherType with its own, therefore each VNID needs multiple MPLS labels assigned to it, one per used EtherType.

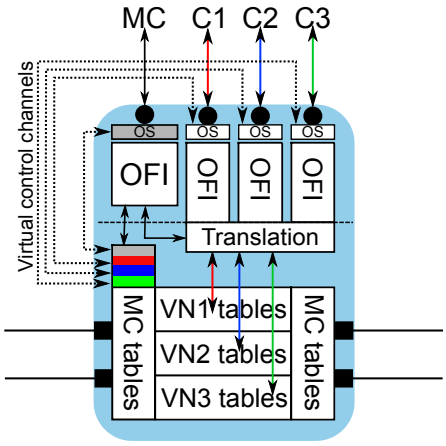

Fig. 3: A full virtualization system

Acknowledgment

This work was supported by the European Commission through the FP7 SPARC project.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] C. Larsen, G. Jacobsen *et al.*, "FTTH: The Swedish Perspective," *Broadband Optical Access Networks and Fiber-to-the-Home*, pp. 43–67, 2006.

[3] M. Forzati, C. Larsen, and C. Mattsson, "Open access networks, the Swedish experience," in *Transparent Optical Networks (ICTON), 2010 12th International Conference on*. IEEE, 2010, pp. 1–4.

[4] [Online]. Available: http://www.btwholesale-inspire.com/

[5] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 2010, p. 8.

[6] M. Gerola, "Enabling Network Virtualization in OpenFlow Networks through Virtual Topologies Generalization."

[7] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[8] E. Salvadori, R. Corin, M. Gerola, A. Broglio, and F. De Pellegrini, "Demonstrating generalized virtual topologies in an openflow network," in *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*. ACM, 2011, pp. 458–459.

[9] [Online]. Available: http://www.openflow.org/

[10] Ixia, "10-Gigabit Ethernet Switch Performance Testing." [Online]. Available: http://www.ixiacom.com/pdfs/library/white_papers/10ge.pdf

[11] M. Devera, "HTB Linux queuing discipline manual - user guide." [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

[12] I. Stoica, H. Zhang, and T. Ng, *A hierarchical fair service curve algorithm for link-sharing, real-time and priority services*. ACM, 1997, vol. 27, no. 4.

[13] "Open vSwitch: An Open Virtual Switch." [Online]. Available: http://openvswitch.org/

[14] "lxc Linux Containers." [Online]. Available: http://lxc.sourceforge.net/

[15] N. Foster, R. Harrison, M. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.