Virtualizing Open Access Networks

Pontus Sköldström, Anders Gavler, Viktor Nordell
Netlab, Acreo AB, Electrum 236, 16440 Kista, Sweden
Email: {ponsko,andgav,viknor}@acreo.se

***Abstract*—Existing network virtualization methods in OpenFlow networks are lacking in many respects, especially when applied in multi-tenant scenarios such as open access networks. In this paper we investigate how OpenFlow can be used to simplify the use and management of a shared network infrastructure in open access networks. We then present initial ideas on how to improve virtualization in OpenFlow networks by applying novel methods to virtualize the forwarding plane itself.**

I. Introduction

The open access business model is designed to reduce the cost of establishing new networks, increase competition and customer freedom, all by setting up a framework where multiple organizations fairly share a network infrastructure. This introduces a number of technological issues, some concerning the management of the network, some concerning the interaction between the shared network and existing networks of the organizations involved.

OpenFlow is a technology that could solve some of these issues; with some improvements it could become a carrier-grade technology that gives involved organizations access to Virtual Networks (VNs) on top of the shared infrastructure while at the same time reducing the costs of introducing novel services in the network.

The next two subsections will give a short background to the open access business model and virtualization. The section on OpenFlow will address a novel way of implementing a well isolated virtualization system in open access networks.

A. The open access business model

The open access business model [1] has been an ongoing development in countries like e.g. Sweden and the Netherlands for over ten years, and in the last couple of years it has been a hot topic on a larger scale as well [2] . The main focus has been on the business model and not on the technical aspects of a shared infrastructure platform.

The business model is based on a number of players active in this market. From a network perspective the scope is home, access, distribution, and to some extent the core network. The different players are modelled as: 1) a residential or business customer 2) the owner of passive network infrastructures, a passive infrastructure provider, 3) the owner of active network infrastructures, an active infrastructure provider, 4) an organization that can bring services into the network, a service provider (SP), and finally 5) a communications operator which is an entity that acts between the customer, the active infrastructure provider, and the service providers. Figure 1 show some of the possible combinations of this - from an open to a more vertically integrated model. A fully vertically integrated business model is when one organization holds all roles from 2 through 5. The open access business model has up till now mainly attracted smaller players and not e.g. large incumbent operators. This is currently changing as [2] is indicating.

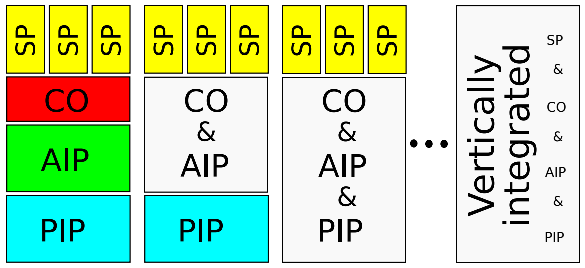

Fig. 1. Some constellations in the open access business model, from least to most integrated. The four operating roles can be seen, Service Providers (SP), Communications Operators (CO), and Active and Passive Infrastructure Providers (AIP and PIP).

As a model it has its short comings, some business related and some technical in nature. Incumbent operators commonly mention that the model leads to too high competition, too small customer slice, the "black box" problem of delivering services over another operator's network, and data and control and management plane transparency (e.g. freedom in use of protocols and addresses) . The first two are mainly business related and will not be discussed here.

The **black box problem** refers to the issue of end-user service support in open access networks which is handled by the service provider, but the service provider has much less visibility and control of the access network in an open access model than if the service provider owned the access network themselves (i.e. in a vertical business model). If the communications operator cannot provide troubleshooting tools to a service provider, it can lead to high support costs which in turn have a high effect on revenue.

Data, control and management plane transparency refers to that an SP should be able to use resource from the network in a transparent manner. An SP should be able to utilize e.g. the entire VLAN (Virtual LAN) or IP address-space without coordinating with other SPs as well as having the freedom of running whatever routing protocol, OAM (Operations, Administration, and Maintenance) protocol or application protocol it wants in the network. All this should be possible while still forwarding traffic in a secure and efficient way, without the possibility of affecting other SPs (e.g. fully separate QoS handling) or getting access to the other SPs traffic or network state.

There are of course service providers that do not want additional control. An example of this could be an IPTV service provider that is only interested in running an IPTV source and reaching to as many customers as possible without worrying too much about how the service is transported.

We believe that split architectures (i.e. separation of data, and control and management planes) together with virtualization is a promising way forward, OpenFlow is inherently a split architecture and could likely be extended to support strict enough virtualization.

B. Virtualization

With the rising popularity of virtualization in the operating system realm many have started to look at virtualization in networks with a more integrated view than what existing techniques offer. Existing techniques such as VLANs and Virtual Routing and Forwarding (VRF) allows the creation of logical topologies on top of physical networks in layer 2 and layer 3 respectively. Different kinds of Virtual Private Networks (VPNs) can be used to create virtual links or even virtual multipoint-to-multipoint connections. The paper [3] looks at the existing techniques and derives a number of architectural principles and design goals that should be met in order to merge and extend these techniques into a ***network virtualization environment*** capable of not only subdividing existing topologies or create virtual links but rather create *Virtual Networks* (VNs) (see figure 2). The goals and principles most relevant to the open access model are:

- **Flexibility** Guest networks should be able to implement arbitrary topologies, routing and forwarding functionality.
- **Isolation** Guest networks must be isolated from each to improve fault-tolerance, security, and privacy. We also include reservations in this concept, e.g. enforcing that guest networks do not use more bandwidth than agreed.
- **Programmability** Network nodes should be programmable in order to ensure flexibility, this is a way to implement e.g. customized routing protocols in the guest networks.

A network virtualization environment conforming to these points could prove very useful in an open access network. Not only can it provide a means to separate traffic from different providers but solve the open access problems listed earlier; different VNs can run different software (programmability) and act on different layers (flexibility), while not affecting each other (isolation).

In the next section we are addressing these aspects through a suggested implementation in an OpenFlow environment.

II. OpenFlow

An OpenFlow-enabled switch exposes an interface to the built-in forwarding tables which can be accessed by a Controller through the OpenFlow protocol. The controller can insert Match→Action rules in a so called FlowTable [4] in order to establish per flow based forwarding or manipulation.

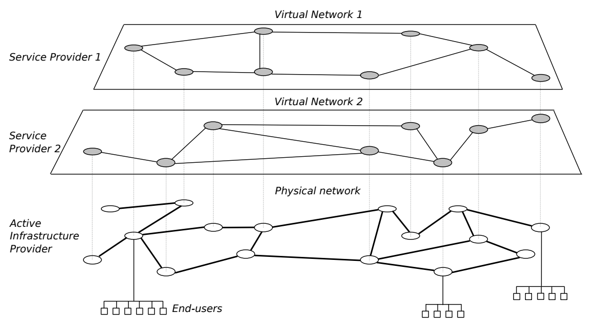

Fig. 2. Network virtualization environment hosting two virtual networks, each operated by different Service Providers.

A rule may match on a 15-tuple consisting of protocol header fields from the physical layer to the transport layer, e.g. incoming port, Ethernet addresses, MPLS labels, and TCP port numbers [5]. When a packet matches a particular rule a number of actions can be performed on the packet, e.g. modifying an address field and then transmitting the packet on physical a port. If a packet does *not* match any of the installed rules the packet is sent to the Controller for analysis. Based on the contents the Controller may e.g. insert a new rule or send the packet to a physical port. If an incoming packet does not match any installed rules it is sent to the controller for further analysis, this is a key feature for providing (limited) **programmability** in OpenFlow networks.

The latest OpenFlow version (v1.1) strings multiple FlowTables together to form a processing pipeline that can be used to perform more complicated operation then what was possible earlier (see figure 3). Multiple lookups and actions makes it possible to for instance use one table to decapsulate a packet from a tunnel and in the next table process packets transported in the tunnel (for details on the processing pipeline see [5]).

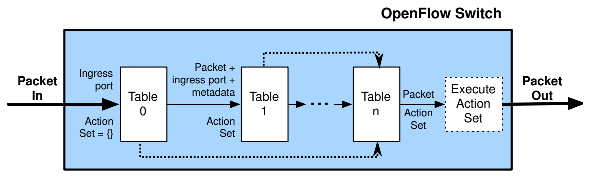

Fig. 3. Basic processing pipeline of OpenFlow 1.1. A packet can traverse multiple FlowTables and Actions can be executed during traversal or at the end of the pipe using the Action-Set. Some information are carried between tables, for example the original incoming port number.

A Controller typically provides a framework allowing a *network application* to interact with the connected switches. In the popular open-source controller NOX [6] a network application registers for certain events, for example to receive packets sent to the Controller. This modular approach makes it easy to extend and upgrade the network. For example,when doing IP Multicast one can deploy IGMP snooping (listening to IGMP messages on switches) in order to reduce the amount of broadcast traffic on Layer 2. In NOX this could be implemented by an application that traps IGMP traffic, performs the snooping and creates FlowTable entries, and then retransmits the IGMP message, without any changes to the switches.

The next sub-sections discuss different virtualization techniques, which provide **flexibility** to the OpenFlow system, and some of their problems.

A. Proxy-based virtualization

FlowVisor is an existing method for virtualizing OpenFlow networks, essentially by limiting a Controllers view of the physical topology. The FlowVisor proxies connections between multiple Controllers and forwarding elements and imposes restrictions on the communication based on predefined policies. For example one Controller can be given control over a part of the total 15-tuple address-space such as the VLAN range 100 to 200. If the Controller tries to send a command that affects a flow outside of this range the FlowVisor will reject it, i.e. not forward the command to a switch and instead return an error to the Controller.

A problem with this solution is the lack of isolation at the data plane level. While the FlowVisor can prevent Controllers from directly affecting each others network views there is no (good) mechanism to enforce isolation of the actual traffic, its difficult to prevent one SP to consume the entire bandwidth of a link. Another issue is more subtle and more of a management issue and that is how the different address-spaces are subdivided. To simplify management of the different network views they should all be defined based on the same part of the 15-tuple, e.g. VLAN ranges, to make sure they do not overlap. This reduces the flexibility of the system as part of the headers has to be reserved to define the views and can cause problems at the networks edge. For example VLAN #1 is typically used to contain management and routing traffic, colliding VLANs from different SPs could be mapped into their particular VLAN range through translation at the edges but this quickly turns into a complicated network management issue. These issues make the proxy-based solution a viable alternative but unnecessarily inflexible and complicated to implement in an open access scenario.

B. Encapsulation-based virtualization

Instead of differentiating traffic based on already existing header values, as in the FlowVisor approach, one can use per-hop encapsulation. In this approach multiple overlay networks are established in the network by creating multiple tunnels on all links between forwarding elements, where the forwarding elements in turn treat packets differently depending on which tunnel they arrived in. In OpenFlow this can roughly be implemented by encapsulating packets with a per-VN MPLS label at the ingress switch in the network, the label is later used by the network to differentiate traffic of different VNs. When an encapsulated packet enters a switch the first FlowTable matches the VN-label, pops it, and forwards the packet to a subset of the FlowTables that are assigned to one VN. Once the packet has been processed, it is correctly encapsulated again before the packet is allowed to leave the forwarding element (see figure 4). We will refer to the first table, that separates traffic onto the VN specific tables, as the demux-table and the last table that reassigns the VN-labels as the mux-table. In [7] they describe a similar method in an OpenFlow context, however without going into details on how it is actually implemented.

The encapsulation based approach allows an SP full use of its set of FlowTables, it can modify any header fields without reduced flexibility. Next section identifies a number of possible solutions of how to implementing this approach.

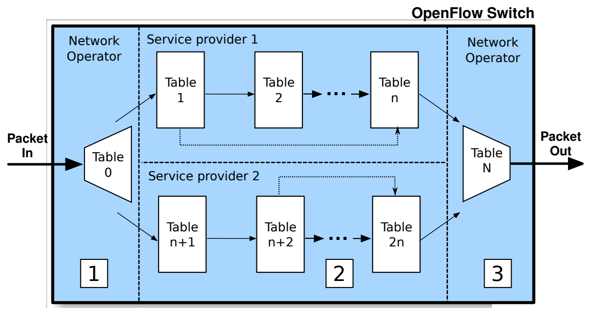

Fig. 4. Using two tables for demultiplexing/multiplexing traffic belonging to different VNs. [1] Matching and popping a VN MPLS label, [2] sending the packet through a VN pipeline, [3] pushing the label back on.

III. Encapsulation-based virtualization solutions

We have identified three different ways of implementing this model in OpenFlow 1.1 by adding minimal features to the protocol and/or adding restrictions to the available actions in the processing pipeline.

1) Explicit mux-table via Metadata In this approach we reserve two network provider tables, one for demultiplexing incoming packets and one for multiplexing outgoing traffic. There is also a number of tables reserved for each VN, into which the packets are demultiplexed from the first table. We then reserve part of the metadata and metadata mask fields which we use to keep track of either which table the packet last visited or which VN it belongs to. When the packets hit the last table, the multiplex-table, the combination of metadata and EtherType is used to push the correct VN label back on again. To ensure correct operations we'd have to make sure that all packets pass through the multiplex-table, this can be done by adding restrictions to the *Output* action. By replacing any output actions with an instruction to go to the multiplex table and putting the Output action in the Action-Set we can enforce that all packets traverse the mux-table before leaving the switch. Replacing actions could be handled by an external proxy like the FlowVisor, or be behaviour built-in to the switch.

This method may be unnecessarily expensive however, primarily since it requires one extra table lookup (compared to following methods).

2) Implicit mux-table via proxy Another method is to implement the multiplex-table implicitly by examining the rules that are installed by the service providers and transparently insert actions that push the VN label back where appropriate.

This would typically be if there's no *Goto-table* instruction or if there is an *Output* action since those would terminate the processing pipeline. However, we cannot simply insert a push MPLS label action directly since there may be other MPLS actions in the Action-Set, either we have to restrict the use of the Action-Set (no MPLS operations) or introduce a new instruction that executes the Action-Set and then lets us push the VN label.

This method, while restricting the use of the Action-Set, seems to be quite light-weight in implementation and the amount of new features needed.

3) Implicit mux-table via Action-Set A third way to implement the mux-table implicitly would be by letting the first table add an push MPLS label action to the Action-Set. Since the Action-Set is executed at the end of the pipeline the network operator can enforce that the correct label is put back once the packet leaves the switch. However, one would have to modify the pipeline rules to restricts the use of the Action-Set in the VN specific FlowTables by first disallowing the use of MPLS related actions in the action set, and by not allowing use of the Clear-Actions instruction (which empties the Action-Set).

This method seems to be the least intrusive, only introducing few restrictions on the Action-Set processing.

There three solutions all share a problem concerning how MPLS labeling affects the EtherType. When a MPLS label is pushed onto an Ethernet frame the existing EtherType is replaced with the MPLS EtherType. The content of the Ethernet frame can then no longer be inferred from the EtherType but has to be associated to the Label Switched Path (LSP) itself in order to restore the original EtherType at the LSP egress. This complicates our virtualization model by requiring us to have multiple labels per VN, one for each EtherType that is used by the VN. In practice this is probably not a problem, typically the number of different EtherTypes is relatively low, so the number of required extra FlowTable entries is probably not large.

IV. Isolation issues

While we have solved the basic isolation by providing something similar to VRF on the OpenFlow level, there are still many ways the VNs can interfere with each other e.g. through bandwidth use on the switch as well as on the control channel(s), switch CPU usage for example through Vendor extensions (non-standard OpenFlow extensions). While these are all related to consuming to much of some kind of resource another isolation issue is FlowTable privacy, one SP should not be able to read the contents, or statistics, of another SPs FlowTable.Most of these issues are addressed in [8], and similar solutions can be applied in our case. We will however elaborate on the bandwidth isolation problem. In [8] they addressed this issue by mapping traffic to VLAN priority classes, which results in a quite coarse and limited way of isolating bandwidth shares. It also requires that packets are modified as they traverse the network, something we wish to avoid. A more flexible way is proposed in [9], this proposal adds the ability to create fine tuned rate-limiters that can e.g. be applied to traffic on flows going through the switch or sent by the switch to the controller(s) by a new Action. In our case this could be utilized by creating per-VN rate-limiters that are executed in the demux-table, thus limiting the amount of traffic allowed to pass through a switch on a per-VN basis. Similarly traffic going to the different controllers can be limited to avoid starvation.

V. Conclusions

We have presented initial ideas on how to improve forwarding plane virtualization in OpenFlow through minor modifications in a way that both increases isolation and improves flexibility. We believe that flexible virtualization combined with other OpenFlow features, such as the ability of running different applications per VN, makes an improved OpenFlow a good candidate for a network adapted to some of the needs of the open access business model.

As future work we are planning to prototype one of the solutions and compare it to existing solutions. For example, these solutions should add less latency to the control channel compared to e.g. FlowVisor but they add extra FlowTable lookup(s) and the encapsulation/decapsulation of packets per hop. This probably introduces extra latency in the forwarding path, which has to be weighted against the added benefits.

We also plan to investigate how many different EtherTypes that are actually in use in typical open access networks, in order to reduce the amount of FlowTable entries it may be worth to move to a different encapsulation format.

Acknowledgment

This work was supported by the European Commission through the SPARC and OASE projects.

References

[1] 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.

[2] T. M. Dirk Breuer, "Research Questions in the Business Case of FTTH." [Online]. Available: http://www.ecoc2010.org/contents/attached/c20/WS_7_Breuer.pdf

[3] N. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009. [Online]. Available: http://www.mosharaf.com/wp-content/uploads/nv-overview-commag09.pdf

[4] 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.

[5] O. S. Consortium, "OpenFlow Switch Specification v1.1.0," 2011.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008. [Online]. Available: http://noxrepo.org/doc/nox-ccr-final.pdf

[7] 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.

[8] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Stanford University, Tech. Rep., 2009. [Online]. Available: http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf

[9] J. Tourrilhes, "Rate Limiter Proposal," Retrieved April 18, 2011, from http://www.openflow.org/wk/index.php/Rate_Limiter_Proposal.