Flexible Building Blocks for Software Defined Network Function Virtualization

(Invited Paper)

Aryan TaheriMonfared, Chunming Rong
Department of Electrical Engineering and Computer Science, University of Stavanger, Norway
{aryan.taherimonfared, chunming.rong}@uis.no

***Abstract*—Current virtual networks offered by IaaS cloud providers are not under complete control of their tenants. The virtual network configuration is carried out by the service provider, and the functionality is limited by the provider's offerings.**

This paper presents a new approach for building and maintaining tenant-programmable virtual networks. This type of virtual networks are the basic building blocks for network function virtualization, and significantly facilitates the implementation of network functions in software. Our approach gives tenants complete control over provisioned virtual network components, and simplifies the integration with on-premises resources. The implementation confirmed the practicality and scalability of the solution, at the cost of a small overhead.

I. Introduction

The cloud computing model provides on-demand access to a shared pool of resources [1]. Depending on the service model, resource types may vary. In the Infrastructure as a Service (IaaS) model, a customer (i.e. tenant) is supposed to have the most flexibility and control over provisioned resources such as virtual machines and virtual networks.

Network virtualization is a powerful abstraction of physical network substrates, and flexibility is an important aspect of it [2]. However, this has not been achieved in today's cloud services. Virtual networks are not under customers' control, and have a very limited flexibility. For instance, a customer can define a basic IP addressing scheme, or access control lists (ACLs) at most. Service providers are responsible for creating and maintaining virtual networks, in collaboration with infrastructure providers. In a public deployment, network functionalities are restricted to those exposed by the provider. While, in a private one, utilizing available functionalities requires tremendous cooperation of involved entities (e.g. operation center, internal tenants).

There has been some efforts among service providers for delivering reliable and efficient network services (including network virtualization), using Software Defined Networking (SDN) mechanisms. However, mechanisms are not exposed to customers, and they may just experience better guarantees and availability. There are several obstacles in exposing the functionality, including clashing customers' configurations (e.g. overlapping RFC 1918 [3] addresses) and fine-grained access control on APIs.

This paper proposes a new approach for the network virtualization, which is similar to the full machine virtualization technique, in terms of accessibility and isolation. In this approach, the cloud provider takes advantage of SDN mechanisms for creating virtual networks, and tenants utilize the same mechanisms to take over the control of them.

Each virtual network has a dedicated and isolated set of networking elements, that are directly accessible and fully controllable by the tenant. They have negligible performance overhead on other tenants and the infrastructure, while provide sufficient management access to the provisioned networking resources for tenants. For instance, tenants can create tunnels, tag interfaces, and utilize different forwarding protocols in their virtual networks, or seamlessly integrate them with their on-premises resources. Programmable virtual networks are the basic building blocks for delivering network-aware services or building specific-purpose overlays.

Section II discusses the background for cloud computing, and the new paradigms in networking. Section III explains our approach, its requirements, advantages, and disadvantages. Section IV studies scalability and performance of the solution. The related work to programmable virtual networks are briefly mentioned in Section V. The final remarks are pointed in Section VI.

II. Background

A. Virtual Networks in Infrastructure as a Service (IaaS) Model

A tenant is a customer of IaaS services and it can have multiple virtual networks. A virtual network may have multiple subnets, which are distinguished from each other using their Classless Inter-Domain Routing (CIDR) blocks. Overlapping CIDR blocks for different networks' subnets may or may not be supported, since it depends on the employed network virtualization technology.

In OpenStack, VMs are attached to an integration bridge, and communicate with other entities through a tunnel bridge. The integration bridge works as a virtual Top of Rack switch, and tags VMs traffic according to their corresponding tenants.

B. Software Defined Networking and OpenFlow

SDN facilitates new methods for managing and configuring networks. The key process in SDN is the abstractions between different layers of networking mechanisms. Shenker et al. [4], [5] introduces three level of abstractions in the network, *distributed state abstraction*, *specification abstraction*, and *forwarding abstraction*. The distribution abstraction provides a global view of the network, and hides the details of distributed states from the higher level mechanisms. The specification

QSHINE 2014, August 18-20, RHODES, Greece

DOI 10.4108/icst.qshine.2014.256308

abstraction builds a simple model of the network, and makes the abstract configuration decoupled from the physical infrastructure. The forwarding abstraction make the forwarding plane more flexible, by shielding the hardware details from the control plane. OpenFlow [6] is one approach for forwarding abstraction, which separates the control plane from the forwarding plane physically.

An OpenFlow switch has a set of flow tables and a group table. An OpenFlow controller can add, update, and delete flow entries in a flow table of the switch. Each flow entry has a matching pattern, a set of ordered actions, a priority, and counters. When a packet enters a port, and the highest-priority matching flow entry in the first table is found, corresponding actions are applied in order; Then the match data and action set are sent to the next table. An action can be forwarding the packet to one or more port(s), dropping, or modifying it [7].

C. Network Function Virtualization

European Telecommunications Standards Institute[1] defines Network Function Virtualization (NFV) [8] as a network architecture which utilizes virtualization for delivering network functions. Functions are realized in software that can be deployed, migrated, and replicated on standard hardware. The software is decoupled from proprietary hardware, and can evolve beyond hardware appliances' lifecycles.

III. Tenant Controlled Networks

In the IaaS model of cloud services, tenants don't have access to network devices, which connect their provisioned resources. They may only have a limited knowledge about the resource distribution, such as availability zones and regions. For instance, the underlying network topology, architecture, and other characteristics are not exposed to tenants.

Moreover, networking mechanisms (e.g. routing protocols) can not be modified and tenants are bound to decisions made by providers. A new network function, which is not supported by the provider, can only be realized in virtual machines. This has several drawbacks such as significant performance overhead, possible single point of failure, and placement challenges (e.g. for middlebox functions).

Providers can not give access to tenants for controlling their virtual networks, since there is no reliable method for segregating network configurations and enforcing policies. In addition, a tenant configuration may have significant impact on other tenants' networks, as well as the provider infrastructure.

In our proposed mechanism, a tenant manages a dedicated set of virtual network components using Open vSwitch DataBase (OVSDB) protocol [9], and programs the data planes through OpenFlow protocol [7], from a logically centralized SDN controller. The tenant's controller is directly connected to virtual switches, which means there is no additional cost for passing events and instructions through another software layer (e.g. infrastructure's SDN controller). Tenants' controllers are decoupled from the infrastructure provider's controller, so they can fail independently. Thus, a failure in the infrastructure's controller won't affect tenants' controllers – assuming there are fail-over forwarding mechanisms; And a failed tenant's controller won't impose overhead to the infrastructure's controller (e.g. polling costs, liveness checking).

The tenant's controller maintains a unified view of the virtual network, and it has insights into the topology, architecture, and resource distribution over the physical infrastructure. The information helps the tenant to implement new network functions efficiently, and deploy services properly.

It should be noted that this approach does not necessarily need tenants intervention. Those tenants who are not willing or do not have the competence, to control their virtual networks, can delegate the task to the provider or any other third party.

The solution is implemented on top of OpenDaylight [10] controller and its OVSDB plugin, with 2762 lines of code, which is available online[2].

A. Components

In addition to the common integration and tunnel bridges, each compute node has a pair of dedicated integration and tunnel bridges for each tenant's network. These tenant network bridges are created only on compute nodes which are hosting VMs from that tenant. Integration and tunnel bridges for tenant *xy* are denoted by *brint-xy* and *brtun-xy*. Virtual bridges in a compute node are represented by records in the Open vSwitch (OVS) Bridge table and are part of the same OVS instance.

Moreover, there should be a dedicated transport network for a tenant. This network type can be maintained by OpenStack Neutron or the infrastructure provider. For the sake of simplicity in this study, the latter is the case here.

A tenant transport network connects tenant's tunnel bridges together through endpoint interfaces (Figure 2). The interface (denoted by *ex-xy* for tenant *xy*) type is *Internal*, and it works as a connection between the bridge and the host kernel's TCP/IP stack. The tunnel endpoint interfaces and an interface (e.g. *eth1*) on the physical transport network are attached to the common tunnel bridge (Figure 1).

Equations (1), and (2) show the number of bridges in a compute node and the total number of them in the infrastructure. As it's shown, the number of bridges in a compute node increases linearly with the number of hosted tenants. Although, the number of bridges increases considerably, the overhead is not significant (Section IV).

$$|bridges_c| = 2 \times (|tns_c| + 1) \tag{1}$$

$$|bridges| = (\sum_{c \in cns} 2 \times (|tns_c| + 1)) + |ons|(2 \times |tns| + 3) \tag{2}$$

where:

tns = tenant networks set with a running VM
tns_c = *tns* on a compute node *c*
cns = compute node set
ons = OpenStack network server set

[1] http://www.etsi.org

[2] https://github.com/aryantaheri/ovsdb

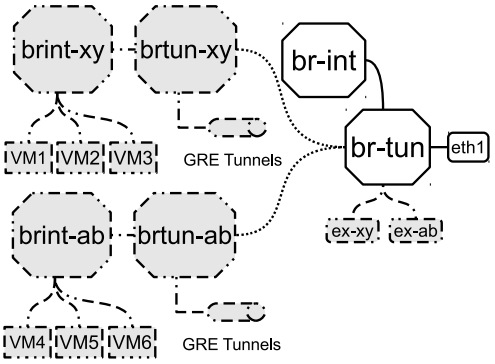


Fig. 1: Dedicated bridges for two tenants in a compute node.

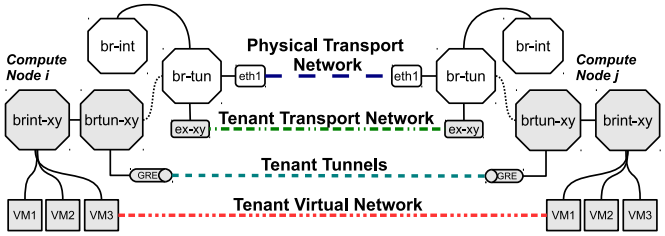


Fig. 2: Logical and physical networks.

When a new VM is scheduled on a compute node, the presence of tenant bridges is verified. If they didn't exist, the tenant bridge pair and the tunnel endpoint interface are created. Then, the VM's TAP device (i.e. virtual link layer device) is attached to the tenant integration bridge and appropriate forwarding rules are pushed.

B. Connectivity

VMs of a tenant in a node are connected to the tenant's integration bridge. This bridge works as a virtual ToR switch and provides intra-connectivity. However, VMs across compute nodes communicate through tenant's tunnel bridges, and this bridge type handles inter-connectivity. Integration and tunnel bridges of a tenant are connected using a pair of patch ports. Figure 1 depicts the networking components in a single compute node when two tenants are hosted.

C. Tunnels

When a tunnel endpoint interface is added, a *udev*[11] rule is triggered, and the interface IP address is pushed to the OVS Interface table. Upon the table modification, the SDN controller is notified, and establishes Generic Routing Encapsulation (GRE) [12] tunnels to discovered endpoints. A tenant's network has its individual tunnel set ($tnts_n$). Tunnels are only established between compute nodes which are hosting VMs attached to that network. Equations (3) and (4) show the upper bound of number of tunnels for a network, and total number of created tunnels in the platform.

$$|tnts_n| \leq \binom{|cns| + |ons|}{2} \tag{3}$$

$$|tnts| \leq |tns| \binom{|cns| + |ons|}{2} \tag{4}$$

where:

$tnts$ = tenant network tunnel set
$tnts_n$ = *tnts* for network *n*

D. Flow Programming

When there is no matching flow entry for a packet, the switch sends a Packet-In event to the controller to retrieve further instructions. The event may contain the full packet or some parts of the header. The processing time at the controller and the round trip time, for the event and its response, is costly. Thus, proactive flow programming of switches is the preferred method for an efficient forwarding. Therefore, flow rules are calculated in a deterministic way and pushed to the switches, when a new entity is added to a virtual network.

Once a VM's port is attached and tunnels are created, the controller pushes four types of flow rules (Table I) to relevant bridges. Whenever a packet traverses a tunnel, it's marked with the corresponding tenant network's segmentation ID (i.e. GRE tunnel key). The tenant tunnel bridge in the host of the new VM is called *local*, and tunnel bridges in other tenant hosts are called *remote*.

Rule types are local ingress, local egress, local flood, and remote egress.

1) The local ingress rule matches the traffic from tunnels on the tenant network's segmentation ID (i.e. GRE tunnel key), then tag it with the tenant's internal VID and forward it to the local integration bridge.
2) The local egress rule identifies other VMs' MAC addresses on the same virtual network at the remote sides, and choose designated tunnels for reaching them. The matching traffic's VID is removed and the traffic is forwarded through designated tunnels.
3) The local flood rule matches broadcast or unknown unicast traffic, and sends them out through all tenant's tunnels.
4) The remote egress rule is applied to the remote bridges. It matches traffic from remote tenant integration bridge on the destination MAC address, and the tenant's internal VID. Then the VID is popped and the traffic is forwarded through the designated tunnel.

The controller creates *O(N)* flow entries in each compute node, where N is the total number of instances which may belong to several tenant networks. Each tenant network tunnel bridge has $O(N_t)$ entries, where N_t is the number of instances on that network.

E. Packet Flow

Figure 3 depicts the packet flow between two VMs which are attached to the same tenant network, and hosted on two compute nodes. Traffic from VM3 on node *i* is tagged in *brint-xy* with the tenant network's VLAN ID on node *i*. If the destination VM is not hosted on the same node, *brint-xy* sends out the traffic to *brtun-xy*, otherwise it will be forwarded locally. When the packet matches relevant flow entries on

Rule	Flow	Match	Actions
Local Ingress	r-tun→l-tun→l-int	tunnel port, GRE key	push VID, fw to patch-port
Local Egress	l-int→l-tun→r-tun	patch port, VID, Dst MAC	pop VID, mark GRE, fw to tunnel-port
Local Flood	l-int→l-tun→r-tun	patch port, VID	pop VID, mark GRE, fw to tunnel-port
Local Egress	r-int→r-tun→l-tun	patch port, VID, Dst MAC	pop VID, mark GRE, fw to tunnel-port

TABLE I: Flow Rules

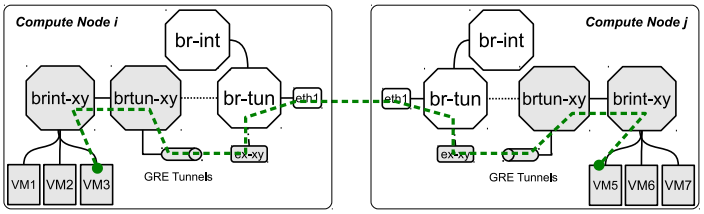


Fig. 3: Flows of packets between two VMs of a tenant across two compute nodes

brtun-xy, it is sent out through the GRE tunnel. Since, tunnel endpoints are on the same subnet as *ex-xy* interface, node *i* TCP/IP stack routes the packet using *ex-xy* interface. At this stage, the common tunnel bridge (*br-tun*) forwards the packet to the physical interface *eth1* on the transport network. Finally, the packet is routed through the physical transport network and reaches *eth1* interface on node *j*, and the reverse process happens there.

F. Trade-offs

The approach has five major benefits for an enterprise tenant. First, the tenant directly accesses the control and management planes using OpenFlow and OVSDB protocols. Second, each virtual network has a dedicated set of virtual components (e.g. virtual switches, interfaces, etc.). These two benefits facilitate the implementation of virtual network functions for a tenant. For instance, one can enforce the required middlebox functionalities to the provisioned virtual network, without involvement of the service provider (i.e. lock-in), or procurement of new proprietary appliances (i.e. additional costs).

Third, tenants' tunnel bridges are connected by GRE tunnels, which facilitate layer 2 isolation of inter-compute VMs' communications. Forth, SDN mechanisms can be used for a unified management of on-premises and off-premises resources. Fifth, virtual network topology and architecture are decoupled from the physical infrastructure, which makes possible transparent modification of the infrastructure.

This approach is not a panacea, and there is a trade-off between flexibility and performance. By introducing dedicated components, the start-up time increases and the implementation becomes more complex.

G. Performance Tuning

Initial results were not satisfactory. Thus, the following tuning were applied to improve the performance:

- MTU size on VMs: The GRE overhead causes the packets to be fragmented, and reduces the performance significantly. By decreasing the MTU size on VMs, GRE packets won't be fragmented.
- Vhost_net: It reduces the number of system calls and avoids context switching, by moving packets between the guest and the host system using the kernel instead of QEMU.
- Hardware Offloading: Where possible, the offloading functionalities of the Linux kernel is used to improve the performance and reduce the CPU load. Tuned parameters are RX and TX checksumming, Generic Segmentation Offload (GSO), and Generic Receive Offload (GRO).
- Kernel's IP neighbour table: The table size is increased to avoid overflow.

IV. Evaluation

The proposed approach must scale in a large infrastructure, with many virtual machines and virtual networks. Thus, several metrics have been measured including reachability time and available bandwidth. The experiment is carried out for a different number of instances and networks, which yields to a variety of instance distribution over compute nodes, and virtual networks. Each experiment is applied to two scenarios: first, tenants don't have full control (i.e. instances are attached to the common bridge), and second, tenants have full control (i.e. tenant instances are attached to dedicated bridges). These scenarios are denoted by CNB (Common Network Bridges) and DNB (Dedicated Network Bridges), respectively.

The virtual machine image used for creating instances is a customized version of CirrOS[13] based on Buildroot[14]. The VM type is m1.tiny with one VCPU, and 512 MB memory. The monitoring aggressiveness of experiments is decreased to moderate its negative impact on the overall performance.

The testbed has five compute nodes, one cloud controller node, one SDN controller, and one monitoring node. Each node has an AMD Opteron Processor 4180, 6 cores, 32 GB memory, and two Gigabit network interfaces. Management and data (VM inter-connection) networks are separated and use dedicated interfaces, which are connected to an HP 5406zl switch. All cloud nodes are running Ubuntu LTS 12.04, OpenStack Havana, and Open vSwitch 2.0.0.

A. Reachability Time

The first experiment measures the reachability time of an instance. The period between an instance spawn up request time (t_{rq}) and the first ICMP echo reply received time (t_{ier}) is called instance reachability time (t_r). The boot requests are sent from the platform controller node to the management API in one batch. And the ICMP echo requests are sent from the network controller node (i.e. an interface in the tenant network's namespace) toward the allocated IP addresses for each instance. This is an effective metric for the analysis of a variety of networking mechanisms in a virtualization platform. Since, the total overhead of not-networking components of the platform is uniformly reflected in this metric, and it can be masked among different mechanisms.

Reachability experiment results have eight fields, number of instances, number of networks, total number of records,

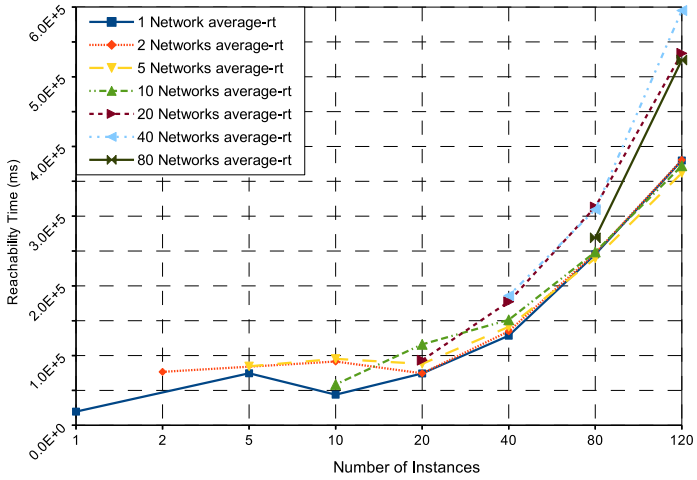


Fig. 4: Average reachability time for DNB

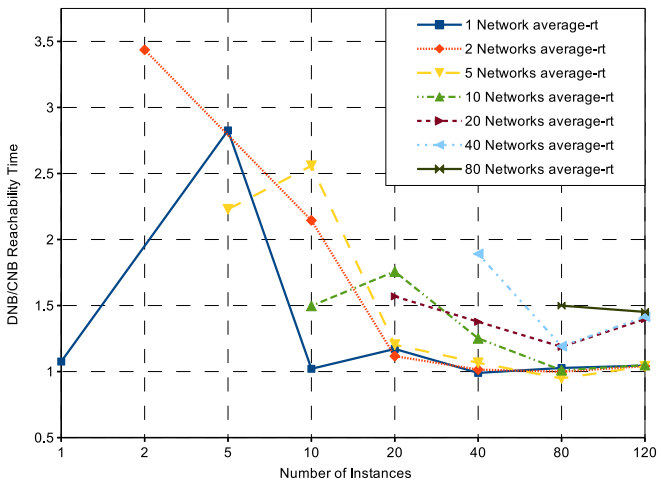


Fig. 5: Average reachability time comparison (DNB/CNB)

success records, failed records, missing records, average reachability time and the reachability time standard deviation.

Figure 4 depicts the average reachability time, when each tenant's network has a dedicated set of bridges and tunnels (i.e. DNB). The Y axis is the reachability time in milliseconds, and X axis is the number of instances. Each data series represents an experiment which has a constant number of networks. An experiment is divided into sub-experiments with a fixed number of requested instances. Sub-experiment's results are data points of the data series, and show the average reachability time for the given number of instances when they're distributed over the experiment's networks.

As depicted in Figure 5, DNB does not perform as well as CNB in this test. It takes longer for a VM to become reachable, when it is attached to a dedicated bridge, and utilizes dedicated tunnels. Although there is an extra start up cost in DNB, this overhead is much less significant when a large number of instances is requested (e.g. 80 instances). Particularly, when the instance distribution is uniform, the overhead is negligible for the last $n - |cns|$ instances, where n is the total number of instances. In DNB, approximately the first $|cns|$ instances faces the overhead of bridge and tunnel creation, and the remaining instances will use existing tenant network bridges.

B. Throughput

The available bandwidth for VMs in a cloud environment is a critical metric for the performance analysis. The experiment evaluates the available TCP and UDP bandwidth between the network controller and VMs, in both directions individually. Iperf is used for measuring TCP and UDP bandwidth, and for each direction the transmission period is 20 seconds.

The TCP experiment reports average and standard deviation of the available bandwidth for both directions and the combined one. The UDP experiment reports average and standard deviation of bandwidth, jitter, transferred datagrams, and out-of-order datagrams.

Figure 6 shows the average available bandwidth for TCP (6a) and UDP (6b), when dedicated bridges are utilized. The

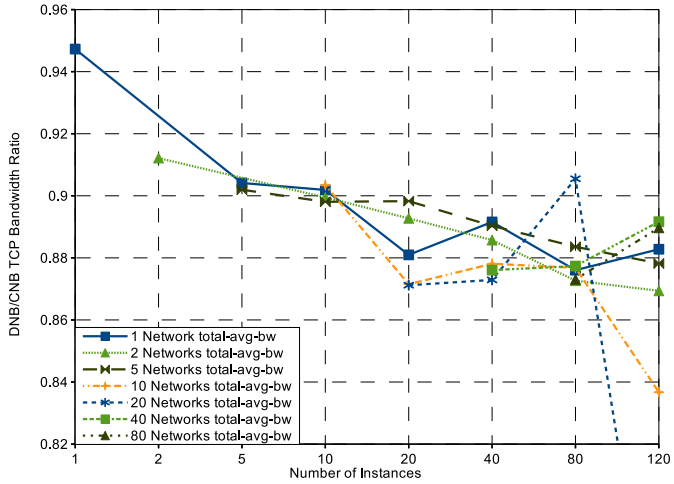
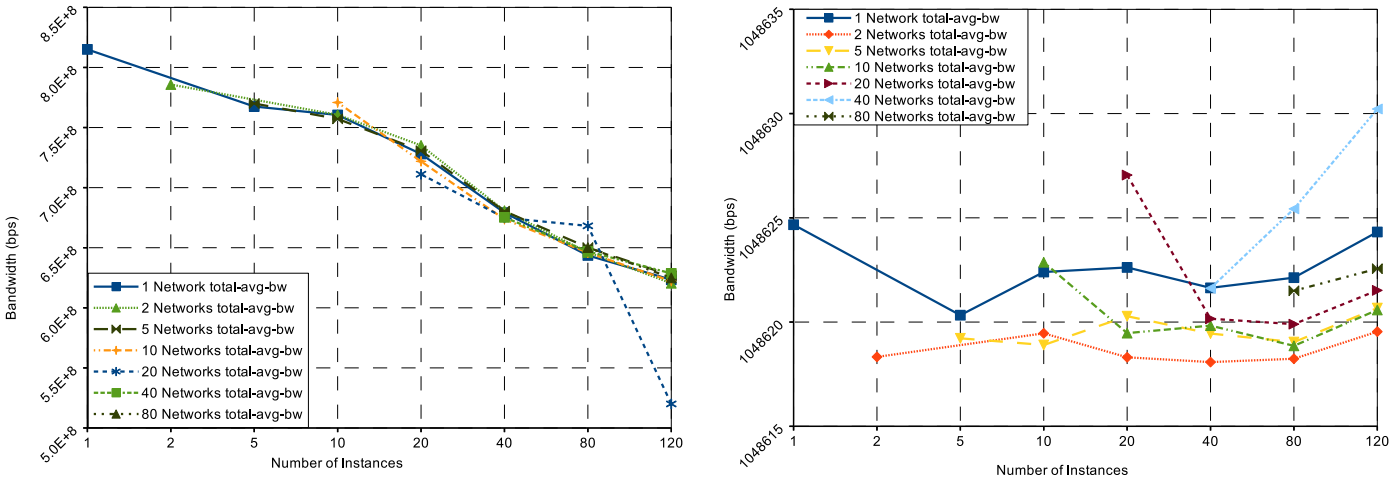


Fig. 8: Bidirectional TCP bandwidth comparison (DNB/CNB)

UDP throughput is consistent, and fluctuates around 1 Mbps for a different number of networks and instances. However, the TCP bandwidth decreases when the number of instances increases, and doesn't change significantly when the number of networks increases for the same number of instances.

Moreover, the TCP bandwidth in each direction for DNB is presented in Figure 7. It shows that the average throughput for the VM to controller direction (Figure 7b) is higher than the opposite one (Figure 7a). This is due to the VM's type (i.e. m1.tiny) and its processing power. Generally, RX operations are more demanding and CPU intensive; Therefore, the processing power can become a bottleneck for the ingress traffic toward a VM.

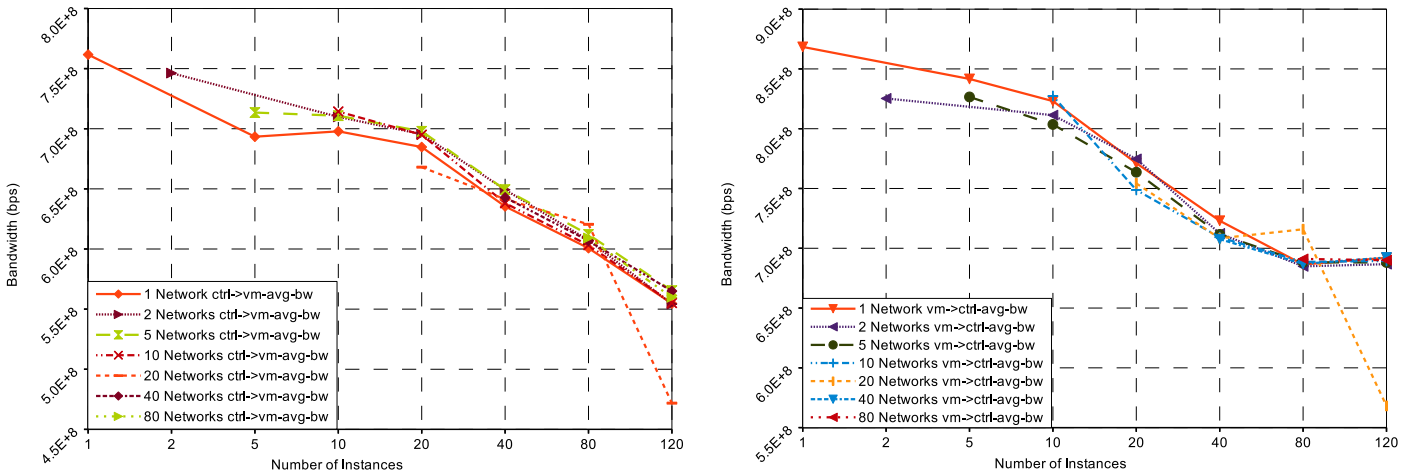
The same set of experiments are performed for CNB, and results are compared with DNB's experiments. UDP performance for DNB improved slightly, while its TCP performance degraded about 8% on average (Figure 8). Although, the performance overhead of DNB for TCP is not negligible, the delivered functionality is considerable.



(a) Average bidirectional TCP bandwidth

(b) Average bidirectional UDP bandwidth

Fig. 6: Average bidirectional TCP and UDP bandwidth for DNB



(a) Average TCP bandwidth from Controller to VMs

(b) Average TCP bandwidth from VMs to Controller

Fig. 7: TCP bandwidth breakdown for DNB

V. Related Work

Oktopus [15] is a system where a tenant expresses network requirements and gets a predictable environment in shared setting, while provider's revenue is not significantly affected. FlowN [16] uses VLAN tagging for creating virtual networks, and isolating address spaces. A virtual network topology is completely decoupled from the physical topology and remains unchanged even after changes in the provisioned resource distribution. Tenants of FlowN share a common controller, similar to container-based virtualization, and their VLAN IDs along switch mapping information are stored in a relational database (i.e. MySQL).

Keller et al. [17] propose an approach for the abstraction of a single router for each customer, and argue that it leads to a better control of the virtual network and reliable service delivery. CloudNet [18] and SEC2 [19] are architectures which utilize VPNs to seamlessly and securely connects provisioned cloud resources to on-premise resources. They isolate customers' virtual networks on the shared infrastructure using VLAN tagging.

CloudNaaS [20] delivers isolated virtual networks, with QoS, middlebox functions, and flexible address space. OpenFlow and VLAN tagging are used for programming flow rules on switches, and isolating traffics, respectively. These features are delivered by the service provider, and a customer doesn't have direct control of their resources.

Slicing the network (e.g. FlowVisor [21]) is another way of providing access to a part of the network, and giving complete access to tenant over that portion. However, network slices are coupled with the physical network topology, and a tenant can

see a subset of physical topology. Moreover, FlowVisor [21] doesn't support cloud platforms, and is not compatible with the recent versions of the OpenFlow protocol.

NVP [22] is a network virtualization platform for multi-tenant datacenters. It's based on a familiar abstraction, so tenants can apply their enterprise network policies to provisioned virtual networks, without modifications. Each flow entry's match and action fields are updated with the packet's metadata, to isolate logical datapaths. NVP uses a declarative language for expressing the logic, which makes the forwarding state computation decoupled from state transitions and event ordering. The language is only available for NVP developers, and can not be used by tenants. Software switching (realized by OVS) is also an important aspect of NVP, that facilitates fast innovation and design flexibility.

Our proposed approach differs from NVP in two aspects; First, NVP uses extra metadata in flow entries to isolate tenants while DNB creates a dedicated set of entities for each tenant. Although OVS uses a single underlying datapath across tenants' bridges in a compute node, they are isolated from a tenant's perspective. Second, in our approach, tenants control their virtual networks directly, and are not dependent on a proprietary solution. Thus, they can avoid vendor lock-in.

VI. Conclusion

This paper presented a new type of virtual network, with dedicated components for each tenant. Enterprise tenants can directly control their virtual networks, enforce policies, apply configurations, and engineer traffic, much like what they can do with on-premises resources. Switches in the networks support OpenFlow and OVSDB protocols, and any controller with compatible southbound APIs can configure them. The solution is implemented as a bundle for OpenDaylight controller, and experiments are performed in an OpenStack deployment. Our experience shows that the approach is not only practical, efficient, and scalable, but also has significant benefits for tenants. The performance overhead is not significant, and the deployment cost is moderated for a service provider.

Acknowledgement

The authors would like to thank Madhu Venugopal and Brent Salisbury for their insightful comments.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Information Technology Laboratory, Tech. Rep. SP 800-145, Sep. 2011.

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1432642

[3] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. d. Groot, and E. Lear, *Address Allocation for Private Internets*, ser. Request for Comments. IETF, Feb. 1996, no. 1918, published: RFC 1918 (Best Current Practice) Updated by RFC 6761. [Online]. Available: http://www.ietf.org/rfc/rfc1918.txt

[4] S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking, and the past of protocols," *Open Networking Summit*, 2011. [Online]. Available: http://www.opennetsummit.org/archives/apr12/site/talks/shenker-tue.pdf

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, p. 16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[6] 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, p. 69, Mar. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1355734.1355746

[7] Open Networking Foundation, "OpenFlow switch specification," Apr. 2013.

[8] ETSI, "Network functions virtualisation," Germany, Oct. 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[9] B. Pfaff and B. Davie, *The Open vSwitch Database Management Protocol*, ser. Request for Comments. IETF, Dec. 2013, no. 7047, published: RFC 7047 (Informational). [Online]. Available: http://www.ietf.org/rfc/rfc7047.txt

[10] "OpenDaylight." [Online]. Available: http://www.opendaylight.org/

[11] G. Kroah-Hartman, "udev – a userspace implementation of devfs," in *Proc. Linux Symposium*, 2003, p. 263271.

[12] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, *Generic Routing Encapsulation (GRE)*, ser. Request for Comments. IETF, Mar. 2000, no. 2784, published: RFC 2784 (Proposed Standard) Updated by RFC 2890. [Online]. Available: http://www.ietf.org/rfc/rfc2784.txt

[13] "CirrOS." [Online]. Available: https://launchpad.net/cirros

[14] "Buildroot." [Online]. Available: http://buildroot.uclibc.org/

[15] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, p. 242, Oct. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2043164.2018465

[16] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[17] E. Keller and J. Rexford, "The "Platform as a service" model for networking," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, p. 44. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863137

[18] T. Wood, A. Gerber, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "The case for enterprise-ready virtual private clouds," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855537

[19] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Secure cloud computing with a virtualized network infrastructure," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, p. 1616. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863119

[20] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: a cloud networking platform for enterprise applications." ACM Press, 2011, pp. 1–13. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2038916.2038924

[21] 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.

[22] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, p. 203216. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen