Datacenter Network Virtualization in Multi-Tenant Environments

Viktor Goldberg[1], Florian Wohlfart[2], Daniel Raumer[2]
[1]Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften,
Boltzmannstr. 1, 85748 Garching bei München
[2]Technische Universität München, Network Architectures and Services,
Boltzmannstr. 3, 85748 Garching bei München
goldberg@lrz.de, wohlfart@net.in.tum.de, raumer@net.in.tum.de

Abstract: Outsourcing of computation and storage infrastructure into the cloud entails new challenges for the architecture and design of multi-tenant datacenters. With the evolution of virtualization techniques, tenant applications do not need to be operated on dedicated servers. Software switches hereby play an important role by mediating between physical infrastructure and virtualized applications. In this paper, we discuss the application of Open vSwitch, a software implementation of a network switch that is particularly designed for use in cloud environments. We analyze its features and performance in comparison to traditional concepts.

Software-defined datacenters provision pools of compute, storage and networking resources and distribute them to their customers [BBE+13b]. Today, primarily networking is seen as a barrier to software-defined datacenters [FRZ13]. Address spaces are tied from guest operating systems down to features that are installed on physical network devices, i.e. physical load balancers, result in slower provisioning and limited placement options. VMs (virtual machines) are geographically dependent on hardware features, vendors and the deployment on underlying fabrics. Programmatic provisioning, control and visibility, allow instant deployment of complex Layer 2 to Layer 7 topologies. From a provisioning perspective maintaining those services is operationally intensive and expensive [GHMP09]. Network Virtualization introduces a layer between physical networking equipment and the guest OS. The virtual overlay network can e.g. be created with a tunnelling technology called VXLAN (Virtual Extensible Local Area Network). We discuss a novel approach to software-defined datacenters as dealt with in the LRZ (Leibniz Super Computing Centre in Munich) infrastructure.

We describe the state-of-the-art in datacenter network virtualization in Section 1. Section 2 provides an overview to Open vSwitch. We look at selected features and the overall performance compared to cutting edge hardware switches and experimental ports like the DPDK vSwitch. Section 3 summarizes this paper.

1 Network Virtualization

An optimal resource allocation by the virtualized networks directly relates to financial savings [GHMP09]. Existing methods for network virtualization consist of plenty of primi-

tives and proposals, but are nothing more than point solutions, in the sense that only single aspects of networking are being virtualized [CKRS10]. VLANs (Virtual Local Area Networks) can virtualize the IP space by applying NAT (Network Address Translation) and therefore, share overlapping IP addresses with multiple tenants. With MPLS (Multi Protocol Label Switching), paths across multiple physical networks can be virtualized. VRFs (Virtual Routing and Forwarding) enables infrastructure operators to have virtual FIBs (Forwarding Information Bases) and therefore moves the routing processes into VMs. On behalf of OpenFlow the Data and Control Plane split paves the way for SDN (Software Defined Networking) [sdn].

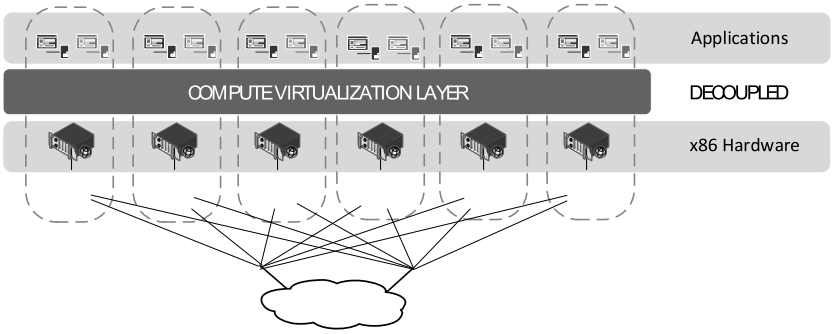

Figure 1: Virtualized multi-tenant topology of the Leibniz Supercomputing Centre

Figure 1 shows virtualized applications and physical machines as they are present in the multi-tenant datacenter of the LRZ. Multiple servers are connected by a physical network that run the tenants' workloads. Redundant physical paths provide connectivity with High Availability, High Reliability and High Bandwidth. On top of the physical layer, operators run the server virtualization software of their choice (like VMware ESX) to host the tenant VMs. If solely the described mechanisms are used to construct virtual environments for tenants, the result is fairly challenging to network operators [BBE+13a]. It is difficult to place or remove VMs around the datacenter because the tenants' addresses are chained to the physical network underlay – as shown by the dotted boxes. The addressing places constraints on where the VM can be moved, as flat IP networks and VLANs do not scale and policies cannot follow VM movements across the network automatically. It is difficult to provision new tenants, as this involves a lot of manual hardware configuration.

Multi-tenant datacenters suffer from the tenants' resources that are often coupled to the physical infrastructure and the resulting inflexibility to run the infrastructure in an efficient way [BBE+13b]. Virtualized software can be operated on commodity x86-hardware, just with a little memory and CPU overhead [CKRS10]. The most important leverage, is that physical machines and the software running on them are entirely decoupled [PPA+09]. Virtual workloads do not interfere with the hardware directly, but only with virtual devices, such as virtual NICs or virtual storage devices. A similar approach for networking is required. On top of the standard switches and routers that provide simple IP-to-IP connectivity a transparent layer is needed that allows to build virtual networks which have identical features to physical, so that workloads are not able to distinguish whether they are processed on physical or virtual instances [PGP+10].

In cloud and datacenter environments, the network virtualization layer consists of virtual switches (e.g., Open vSwitch) or virtual routers which are entirely deployed in software. On the hypervisor a small number of NICs is shared between all VMs running on the host. Hence, instead of supplying VMs with straight access to physical NICs, which limits the number of VMs to the number of available PIFs, they are connected via VIFs. In contrast to the simple network bridge, software switches provide a rich set of features, including interfaces like SNMP, SPAN, ERSPAN and CLI [ovsc]. The integration of software switches with the hypervisor (e.g., VMware NSX) even allows to get information from VIFs and machines that are difficult to determine by conventional switches [vmw]. For example, a virtual switch may determine which IP address or even multicast group is associated with a particular VM. With this knowledge broadcast and multicast storms can be cut down, if not avoided. A virtualized network environment extends the service possibilities of a multi-tenant datacenter to a new level: Cloud providers (like LRZ) can take advantage of their hardware resources while security in the network and isolation of multiple clients, which share one physical infrastructure, can be sustained [BBE+13b]. By removing all tenant services from the physical underlay network and porting them to a virtual deployment, cloud providers are also able to add network service features, e.g. to offer a single broadcast domain to tenants which are spread over multiple different IP networks by virtually tunnelling the whole tenant's overlay network - beyond Layer 2 and Layer 3 borders.

2 Open vSwitch

2.1 Design and Architecture

Open vSwitch (OVS) [PPA+09] is an open source multi-layer virtual switch that is used for network virtualization. We chose OVS as the central part of our analysis on virtual switches, because it is widespread, feature-rich and shows good performance. For large cloud frameworks like OpenStack [1] or OpenNebula [2] OVS serves as the backend. Like a physical switch it operates on Layer 2 but also has the ability to accomplish tasks on Layer 3 and Layer 4, for instance, make decisions not only based on MAC addresses but also considering IP addresses or L4 ports. OVS is a software implementation that runs on commodity hardware supporting different platforms, like Linux. It is split up into two major components – the controller which is the decision making engine, running in the userspace and a datapath that is implemented as kernel module for fast packet processing. OVS supports the OpenFlow protocol, but also adds features which are not present in OpenFlow (like QoS Control). The userspace module, which consists of a controller and tools, is licensed under the Apache license, whereas the `datapath` is under the GPLv2 for compatibility reasons with the Linux kernel [ovsb].

Comparing OVS to a physical switch, interfaces correspond to the slots (e.g., RJ-45 connectors) where cables are plugged in. A switch port can aggregate one or more interfaces using a technology called port bonding with a protocol like LACP (Link Aggregation Control Protocol). This allows to build load balancing and high availability environments. In switches, packets are forwarded on the basis of flows defined by source and destination

[1] http://docs.openstack.org/admin-guide-cloud/content/under_the_hood_openvswitch.html
[2] http://archives.opennebula.org/documentation:archives:rel3.8:openvswitch

MAC. In OVS flows can be identified by other, much more specific attributes, like Tunnel ID or IPv6 ND Target. This flexibility differentiates OVS from the existing bridging implementation, the Linux Bridge. The integration across multiple hypervisors, which allows network designers to span a virtual switch beyond the scope of a physical host, makes OVS a very promising technology in the context of SDN [lx-].

2.1.1 Forwarding Path

With the Linux bridge source and destination MAC address of a received packet is consecutively inspected, learned and forwarded [Ros13]. The packet never leaves the kernel. OVS design and architecture is different although it is still connected to the same place in the Linux kernel where the Linux bridging code is nested and resources are shared accordingly. The Linux networking stack is being bypassed. Path determination and packet processing decisions are made in the userspace.

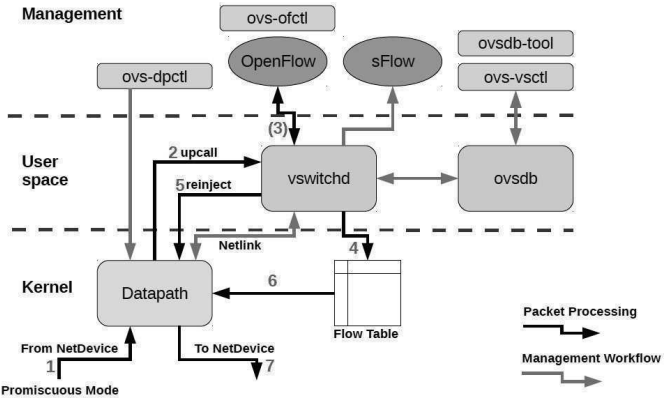

Figure 2: Architecture of Open vSwitch, packet processing and management workflows (from [wif])

The first packet to come in hits the datapath as shown in Figure 2 (1), where a flow table lookup is being performed to determine whether the datapath has any knowledge on what actions to perform with the packet. The first packet(s) in a flow is sent up to the *ovs-vswitchd* process (2) in the userspace. Major tasks of *vswitchd* are to communicate with SDN controllers using OpenFlow (6633/TCP) (3), interacting with the OVS database *ovsdb-server* via the management protocol (6632/TCP) and exchanging information with the kernel module using Netlink. When a packet from the kernel is received it is handed to the classifier which holds one or more OpenFlow tables and is a part of *vswitchd*. For example, if a simple Layer 2 virtual switch is implemented, it makes assumptions based on L2 addresses and then decides to forward on a specific port(s). Whatever decision is made for packets belonging to that flow, *vswitchd* has to instruct the datapath about what to do with it. Because the datapath is implemented to do lightweight forwarding, it does not keep state about the packet sent to the userspace earlier and therefore does not wait for a response. A newly generated flow table entry is added (4) and *vswitchd* is forced to inject (5) the processed packet back into the datapath. The recently inserted packet is matched against the new flow table entry of the datapath (6) and forwarded (7) to the destination [ovsa].

2.1.2 OVS Management

Plenty of tools exist to interact with OVS, configure and observe the state of the system as shown in the Management layer of Figure 2. The *ovsdb-server* holds the complete switch-level configuration and further communication details e.g., IP addresses between OVSDB and an OpenFlow controller. The state is durable, written to disk and therefore persistent upon restart. All properties like value constraints, weak references and garbage collection are met. Communication is handled via the OVSDB protocol that is based on JSON RPC. For configuration a connection to *ovsdb-server* needs to be established, which then connects to other parts of the system on behalf. A call to the database returns exactly then when the requested configuration action has finished with the information of failure or successful completion. This makes OVSDB very reliable in terms of remote management, as no concern needs to be put on the signalling. For efficiency reasons, a Unix socket is used for database configuration. This obtains the possibility to interact remotely using TCP and even SSL deliberately on top [ovsd, cla].

2.2 Features

OVS accomplishes simple tasks, analogous to what the Linux network bridge does: designating memory for network packets, buffering and forwarding packets in the kernel module and processing L2 frames from VIFs to PIFs, PIFs to VIFs and VIFs to VIFs.

2.2.1 Network Virtualization Overlays

With a high density of VMs in multi-tenancy clouds comes the need for tunnelling technologies, to separate tenants from each other. Major requirements include the need of L2 adjacencies across L3 boundaries, logical separation of virtual broadcast domains and decoupling of virtual networks from the physical infrastructure. Figure 3 depicts a stripped down cloud network topology with two physical servers pHostA and pHostB in separate physical locations. The underlay network provides basic IP End-to-End connectivity and consists of one router separating two L2 switches with the IP spaces 1.1.1.0/24 on the left and 2.2.2.0/24 on the right side. pHost A and pHost B both have IP addresses attached to their PIF from the networks they are connected to and run hypervisors (like KVM). Each server has a virtual switch running to provide connectivity for VMs. Both hypervisors run a multi-tenant switch (e.g., Open vSwitch) with tenants T1 and T2. T1.1 receives IP 10.0.0.1 from the vSwitch, T1.2 on the other physical end receives 10.0.0.2. The underlay network in-between has no routing knowledge for these specific networks, and additionally there is no L2 connectivity between T1.1 and T.1.2 as the router in middle forms a L3 boundary. The second tenant's IP space overlaps with T1 and therefore equal IP addresses are assigned to T2.1 and T2.2.

Although, there is the possibility to use traditional 802.1Q native[3] VLAN tagging in conventional L2 devices, scalability issues will remain considering there is no support for overlapping IP networks. Furthermore only a maximum of 4096 concurrent networks can

[3] An extension called Q-in-Q tunnelling, where a VLAN tag is encapsulated into an outer VLAN tag allows for tunnelling tenant networks through i.e. an ISP network but still fails over L3 paths.

coexist which is an insufficient number for large scale cloud deployments. Virtualization and therefore separation of the tenants' L2 networks allow for flexibility in IP addressing and likewise provide security through isolation of private traffic.

Through a tunnelling mechanism like VXLAN a virtual overlay network can be created, as in Figure 3 symbolized by the cloud and the curved connections.The vSwitches on both sides act as tunnel endpoints, called VTEPs (Virtual Tunnel End Point) in VXLAN. VTEPs require the information about the location of each VM of a tenant. For example the virtual switch on pHost A acting as a VTEP needs to learn that in order to reach T1.2, traffic has to be sent out towards pHost B at 2.2.2.2 which implies that VTEPs depend upon a mapping between VM L2 addresses and virtual network end point IP addresses. Mechanisms VTEPs utilize to derive these mappings include manual, push and dynamic learning. Manual configuration is usually not used in productive environments. By acquiring mappings in a pull/push process an external SDN controller is consulted (cf. Figure 3) or information is pushed towards it. In the dynamic learning process established flows are inspected and mappings extracted similar to L2 address learning. For example ARP broadcasts can be encapsulated into multicast messages to get across the network to VTEP subscribers [PPA+09]. The VXLAN Header holds the VNI (VXLAN Network Identifier) that allows the VTEPs to distinguish between different tenant domains. In the example of Figure 3 T1 holds VNI 10. VNIs work analogous to conventional VLAN tags, yet providing more scalability. 802.1Q tags have a maximum of 12 bits reserved for the ID which results in 4096 possible addresses. VNIs, however, have 24 bits reserved which allows for over 16 M different identifiers [MDD+14].

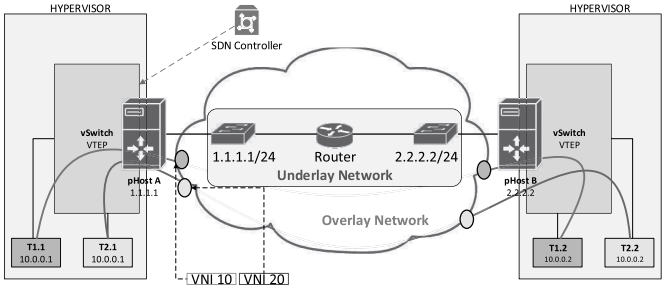

Figure 3: Two overlapping L2 tenant domains, tunnelled across an L3 network using VXLAN

Alternative solutions like GRE (Generic Routing Encapsulation) or LISP (Locator/Identifier Separation Protocol) for overlay tunnelling are also present in OVS [os-].

2.2.2 The Intelligent Edge

OVS supports the same features as high-end hardware switches – including features for the intelligent edge like LACP, 802.1Q, 802.1D STP, fine-grained ACLs, QoS control, BFD in the context of 802.1AG link monitoring and L4 hashing [os-, lis]. Intelligent edge networking defines the concept of shifting the logic from the core to the boundaries of the network. The edge position advances other in-network devices as it can summarize management of a great amount of virtualized hosts in comparison to an in-network device (e.g., traditional physical host). Many cloud networks face the problem of oversubscribed

PIFs. Often the guest VMs are not trusted which makes it indispensable to enforce traffic policies on the virtual node, before sending it out over the physical link. OVS has the ideal position for network control and visibility as different parts of the systems at the edge have the possibility to communicate between one another. For example by having a service VM that is running an IDS (Intrusion Detection System) and issues are detected by it, this machine has the possibility to communicate over an out-of-band networking subsystem with OVS and in consequence isolate traffic that has been determined to be compromised. This method can even be improved by introspection where a specific agent runs on the host VM collecting network data.

One more important feature is TCP pacing: When TCP traffic is sent from a VM, it may get sent as large TSO (TCP Segmentation Offload) segments and OVS now breaks them up into the according MTU size before sending them out, thus saving buffer windows by not filling them up. There is also the approach of doing the opposite which is required because all the clients communicating via TCP through one PIF end up filling the buffers while trying to maintain their state. Jitter is introduced into some sessions to break this up [ASA00].

Finally, the flowlet switching feature where the concept is to send a single TCP session over multiple links by consulting the RTT, keeps the segments in-order. Different links are chosen, once they have cleared the total RTT [SKK04].

2.3 Performance Analysis

In virtual switching environments packets are forwarded from VM to VM (VIF-to-VIF), VM to in-line network device (VIF-to-PIF) or in-line network device to VM (PIF-to-VIF). As shown in Figure 4, in a VIF to VIF scenario there is either the way of switching in the edge (1) or hairpinning (2) by directing the traffic through a hardware switch.

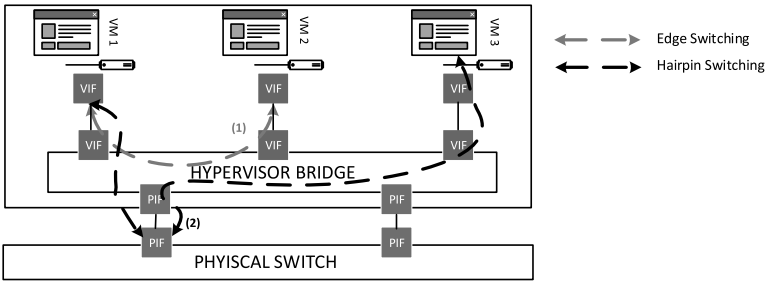

Figure 4: Two different switching approaches: Switching at the Edge (1) and Hairpinning (2)

Efficient processing of packets in switches like OVS is limited by multiple factors (e.g., CPU, #out ports). Adaptation to the system and the integration of packet IO frameworks like DPDK (Data Plane Development Kit) can be used to accelerate OVS. For example the Intel X710 40 GbE controller supports VXLAN offloading [Int14].

2.3.1 Evaluation

The performance of packet processing in software is only limited by the CPU when using common PC systems [ERWC15]. This means the performance of OVS is dependent on the system CPU and the number of cores used for parallel packet processing.

The maximum performance OVS achieves is a packet rate of 1.88 Mpps per CPU core (PIF-to-PIF) which is a meaningful improvement compared to the Linux bridging code that accomplishes a rate of 1.11 Mpps, and IP forwarding at 1.58 Mpps [ERWC14]. This best-case performance was measured with OVS running on a dedicated host (equipped with a 3.3GHz Intel Xeon E3-1230 V2 processor) attached to two physical network interfaces (Intel 10 Gigabit X520-SR2), where one serves as input and one as the output interface. The test traffic consists of one single flow (uniform UDP packets with 64 Byte packet length) and constant interframe gaps (constant bit-rate). Also in the context of PIF-to-VIF switching (OVS 0.85 Mpps, IP forwarding 0.78 Mpps, Linux bridge 0.74 Mpps) and PIF-to-VIF-to-PIF switching (OVS 0.3 Mpps, IP forwarding 0.19 Mpps, Linux bridge 0.2 Mpps), OVS always exceeds the performance of IP forwarding and the Linux bridge and therefore is well suited as an all-purpose switch.

Adaptation to the system and the integration of packet IO frameworks like DPDK (Data Plane Development Kit) can be used to accelerate OVS. DPDK vSwitch with a single core PIF-to-PIF throughput of 11.31 Mpps is considered as the fastest possibility of switching in software, yet the DPDK framework is not fully supported by OVS which results in regular instability and therefore is not qualified for production workloads [ERWC14].

2.3.2 Multithreading

Since megaflows (allow the kernel module to support a random number of bitwise wildcarding, instead of using an exact-match cache) were introduced in OVS 1.11 the total amount of packets that passed through the userspace – the major bottleneck of OVS – was radically diminished. Version 2.0 introduced multi-threading in *ovs-vswitchd* which now added the possibility to not only use OVS as a hypervisor switch but completely utilize it for PIF-to-PIF processing. This induces the ability to entirely substitute traditional switching appliances. The change in architecture allows the kernel cache to be extended to 200,000 flow entries in contrast to previous versions where the cache had a maximum capacity of 1000 flows [ovsa].

2.3.3 Scaling with multiple cores

After analyzing the per-core performance of OVS, this section focuses on the scalibility of OVS using multiple CPU cores in parallel. Therefore, we extend our scenario from 2.3.1. In addition to the existing traffic consisting of one flow, we create traffic patterns with up to seven equally distributed flows and ensure that each flow is processed by a dedicated CPU core.

Figure 5 shows that the performance of OVS scales almost linearly with the number of cores (represented by the number of flows). The packet size is insignificant unless the Ethernet Link limit is reached [ERWC14]. In Figure 5 that limit is 10 GbE. This effect

is explained by a comparative high memory bandwidth accessed directly by the NIC via DMA. By taking advantage of RSS (Receive Side Scaling), a technology used for hardware offloading, linear scaling of overall performance with the count of added CPUs is understood. The actual slight sub-linearity is caused by synchronization efforts between the cores (e.g. for statistics), and dynamic boosted frequencies in case of skewed loads. As rule of thumb about 1 Mpps per GHz and core can be forwarded on today's servers.

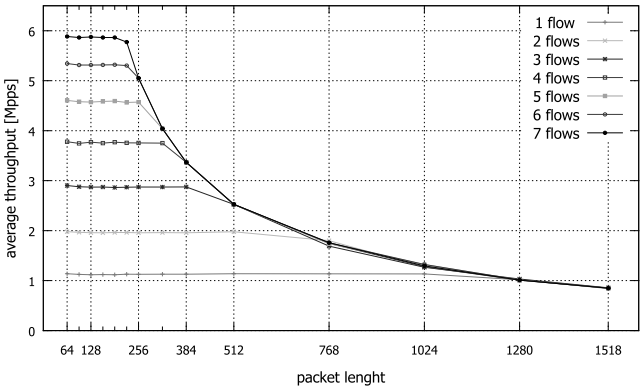

Figure 5: Throughput performance of OVS (10 GbE) PIF-to-PIF; various packet sizes and flows

3 Conclusion

Network operators in traditionally managed datacenters nowadays struggle with problems that arise in modern multi-tenant environments. Those legacy architectures lack behind in terms of efficiency, mobility and flexibility both for customers and providers. Hence, Datacenter Network Virtualization is expected to solve this issues while also providing security by proper isolation.

We have analyzed that advanced edges, supplied with a considerable amount of features, can impact the design of future network architectures. We have pointed out that Open vSwitch does not only support basic switching functionality but also has the ability to provide elaborate services, i.e. L2 tunnelling mechanisms through insecure L3 networks. Conventional switching and routing architectures can be simplified to provide IP End-to-End connectivity only, as this is the single requirement for virtualization overlays.

References

[ASA00] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *INFOCOM 2000. Proc. of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, 2000.

[BBE+13a] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data center network virtualization: A survey. *Communications Surveys & Tutorials*, 15(2), 2013.

[BBE+13b] Md. Faizul Bari, Raouf Boutaba, Rafael Pereira Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md. Golam Rabbani, Qi Zhang, and Mohamed Faten

Zhani. Data Center Network Virtualization: A Survey. *IEEE Communications Surveys and Tutorials*, 15(2), 2013.

[CKRS10] 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.

[cla] Keeping IT Classless, [SDN Protocols] Part 3 - OVSDB. `http://keepingitclassless.net/2014/08/sdn-protocols-3-ovsdb/`. visited 15-2-10.

[ERWC14] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Performance characteristics of virtual switching. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014.

[ERWC15] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems. In *Fourteenth International Conference on Networks (ICN 2015), Barcelona, Spain*, 2015.

[FRZ13] N. Feamster, J. Rexford, and E. Zegura. The Road to SDN. *Queue*, 11(12), 2013.

[GHMP09] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *Computer Communications Review*, 2009.

[Int14] Intel Ethernet Controller XL710 Datasheet Rev. 2.1. Intel, 2014.

[lis] Using LISP tunneling. `https://github.com/emaste/openvswitch/blob/master/README-lisp`. visited 15-2-10.

[lx-] Virtual Networking in Linux. `http://www.ibm.com/developerworks/linux/library/l-virtual-networking/`. visited 15-2-10.

[MDD+14] M. Mahalingam, D. Dutt, K. Duda, L. Agarwal, P.and Kreeger, T. Sridhar, M. Bursell, and C. Wright. RFC7348: Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. 2014.

[os-] Configuring the Quantum Open vSwitch Plugin. `https://wiki.openstack.org/wiki/Obsolete:ConfigureOpenvswitch`. visited 15-2-10.

[ovsa] Accelerating Open vSwitch to "Ludicrous Speed". `http://networkheresy.com/2014/11/`. visited 15-2-10.

[ovsb] OVS Documentation. `http://openvswitch.org/support/`. visited 15-2-10.

[ovsc] OVS: Features. `http://openvswitch.org/features/`. visited 15-2-10.

[ovsd] Ubuntu Manuals, ovsdb-server - Open vSwitch database server. `http://manpages.ubuntu.com/manpages/natty/man1/ovsdb-server.1.html`. visited 15-2-10.

[PGP+10] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby. Virtual switching in an era of advanced edges. In *Proc. 2nd Workshop on Data Center—Converged and Virtual Ethernet Switching (DCCAVES), ITC*, volume 22, 2010.

[PPA+09] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending Networking into the Virtualization Layer. In *Proc. ACM SIGCOMM Workshop Hot Topics in Networks (HotNets)*, 2009.

[Ros13] R. Rosen. *Linux Kernel Networking: Implementation and Theory*. Apress, 2013.

[sdn] OpenStack Taking Its Place in the Software-Defined Economy. `http://www.openstack.org/blog/2014/08/openstack-taking/`. visited 15-2-10.

[SKK04] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCP's burstiness with flowlet switching. In *Proc. ACM SIGCOMM Workshop Hot Topics in Networks (HotNets)*, 2004.

[vmw] vmWare NSX. `http://www.vmware.com/products/nsx`. visited 15-2-10.

[wif] Future Wireless Networking Project Wiki. `http://teampal.mc2lab.com/projects/fwn/wiki/Build_Open_vSwitch`. visited 15-2-10.