BNV: Enabling Scalable Network Experimentation through Bare-metal Network Virtualization

Pravein Govindan Kannan Ahmad Soltani Mun Choon Chan Ee-Chien Chang
School of Computing, National University of Singapore

Abstract

New paradigms and architectures, such as Software Defined Networking (SDN), have added an unprecedented increase in the rate of research and development conducted in the field of computer networks. With this increase, there is a rising need for platforms that can enable researchers and operators to experiment with various scenarios involving performance testing, SDN, security research, topology designs, etc. However, the available emulators fail to address fundamental needs of the experiments requiring diverse and scalable set of topologies.

In this work, we propose a novel approach to embed arbitrary topologies on a substrate network of programmable ToR switches using our network virtualization technique, called Bare-metal Network Virtualization (*BNV*). BNV is entirely software configurable and has been implemented on open source software and unmodified OpenFlow-enabled switches. The system has been deployed and currently running in a production testbed in National Cybersecurity Laboratory (NCL) for a year. Our evaluations show that BNV can support various data-center topologies with less number of switches which can facilitate building a high fidelity, repeatable and isolated experimentation platform for data-center, SDN and security research in computer networks.

1 Introduction

Network experimentation is an integral part of Network and Systems research. With new paradigms like Software Defined Networking, networks are becoming more programmable and customizable according to application requirements. Data-center architects and network operators, constantly work on optimizing network topologies and algorithms under various scenarios and to maintain the production network in a steady state [18].

Fidelity and repeatability are two of the essential requirements for any experimentation platform. Additionally, to mimic real-life production network scenarios, flexibility and scalability in emulation of topologies are essential at both the control-plane and data-plane. Unfortunately, none of the available tools can satisfy both fidelity and flexibility in the data-plane.

In most cases, experiments rely on network emulation tools like Mininet [13], NS, etc. These tools allow experimenters to create topologies very quickly on their host machines. However, since they run on the CPU of the machines they are hosted on, the network performance depends on the deployment environment. Hence, it is hard to achieve fidelity and repeatability in the experiments. Additionally, these tools cannot scale to emulate production networks [18, 26].

Network testbeds like CloudLab [21], DeterLab [19] provide an environment for network and security experimentation. However, since the network is considered as an infrastructure [1] and managed using VLANs, experimenters cannot program their network to perform SDN-based experiments.

In this work, we propose *BNV*, a network virtualization technique to bridge the gap between production networks and emulation tools by providing the following features:

Flexible network topology. BNV provides each tenant with their desired topology in a bare-metal fashion (virtualization is done at the switch hardware) while the underlying topology is entirely different. Hence, multiple arbitrary topologies can be provisioned over the hardware switches using BNV as a network hypervisor. Our technique makes emulating large topologies more accessible and cheaper compared to the actual cost of building one. The experimenters can design custom algorithms or protocols to program their allocated network.

Fidelity. BNV provides consistent performance at hardware line-rate. Such fidelity requirement is essential for network experimentation involving applications whereby the network configurations need fine-tuning for optimal performance and regulatory verification.

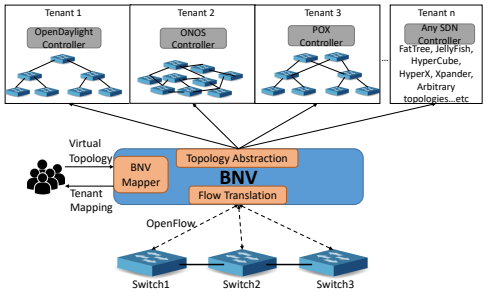


Figure 1: BNV System Overview

Isolation and performance guarantee for multiple tenants. BNV allows different tenants or experimenters with different application requirements and topologies to share the same network infrastructure while guaranteeing complete isolation and repeatable performance by implementing buffer and queue management.

To implement BNV, we address the following challenges:

1. What is the right substrate network for BNV with maximum flexibility and minimum wiring changes?
2. What is the algorithm needed to find the most efficient network embedding?
3. How to implement BNV using existing tools?

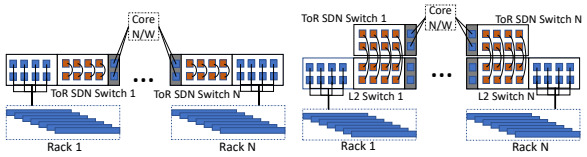
To address (1), we propose a novel topology wiring using loop-back links in switches, to emulate multiple arbitrarily connected virtual switches. To address (2), we formulate the problem as an integer linear programming (ILP) to maximize the scalability of experimental support and fidelity. For (3), we implement BNV over an existing network hypervisor OpenVirteX [7] and also integrated the platform with a bare-metal provisioning system DeterLab[19]. We have deployed BNV in a Cyber-Security testbed called National Cybersecurity Lab (NCL) [4] for over a year for experimenters to run SDN and other experiments.

Our evaluations have shown that BNV can support high fidelity virtualization of network topologies, and can virtualize networks consisting of more than 100 network devices using only five top-of-rack (ToR) switches, while supporting line-rate for high-performance testing of production workloads.

2 BNV Architecture

The main component of BNV which differentiates it from other network hypervisors is the one-to-many mapping which virtually slices the switches in the underlying physical infrastructure into multiple arbitrarily connected virtual switches (collection of physical switch ports), without losing the queuing behavior and providing fidelity in mapping. We use the terms physical or substrate topology interchangeably.

Figure 1 shows the architecture of BNV. BNV connects to the SDN switches via OpenFlow. Users submit



a. Static Loopback

b. Software-Configurable Loopback

Figure 2: Loopback Configuration in Network

their virtual topologies to the BNV Mapper which embeds the resources and finally creates the network for the tenant with the reserved physical resources.

BNV can create multiple tenants and supports use of any SDN controllers to control slice of the network independently, without interfering with the other tenants co-located on the same physical switch. We explain how BNV performs *topology abstraction* and *flow translation* in the following two sub-sections. We present the ILP based embedding of *BNV mapper* module in section 3.

2.1 Topology Abstraction

In a generic topology, switches are connected to hosts using relatively low-bandwidth links, and other switches (or core-network) using high-bandwidth links. We introduce additional link type to perform topology abstraction.The switches, apart from connecting to hosts or other switches, have special links which are **loopback links**, i.e., a link between two ports on the same switch. If a switch has X ports, H ports are connected to the hosts, C ports are connected to other switches, and L ports are loop-ports which form $\frac{L}{2}$ loopback links. The number of loopback links can be pre-configured as shown in Figure 2.a or dynamically configured by placing an L2-Switch or circuit switch between the host and the SDN Switch (ToR) as shown in Figure 2.b. With this configuration, we can dynamically configure the number of ports allocated to host and loopback at run-time through VLAN or circuit switch reconfiguration.

With the loopback links any two connected virtual switches can map to a single physical switch using the loopback link as their inter-switch link. Consider a substrate network as in Figure 3. The hosts are connected to the switches using 1Gbps links, and the inter-switch link has a bandwidth of 10Gbps. The virtual topology to be implemented is a triangular network (3 switches and 3 hosts) with 1Gbps links. The virtual topology mapping to the substrate is shown in Figure 3 where the links vL1 and vL2 are mapped to the CLink1, which is a backbone link in substrate network. vL3 in the virtual topology is mapped to the substrate network using loopLink1. Loopback links provide flexibility in abstraction while maintain the fidelity of the virtual links.

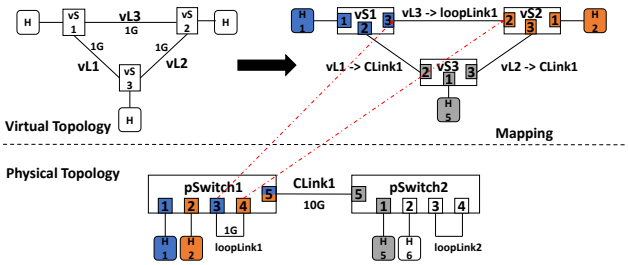


Figure 3: Mapping of Virtual Topology with Loopbacks

Table 1: Flow Translation using loopback link

Entity	FlowMod added to Switch
vS1	**pSwitch1** : {in_port=1,ip_dest:y.y.y.y, actions=meter:1,*lTag*:2,output:3}
vS2	**pSwitch1** : {in_port=4,ip_dest:y.y.y.y,*lTag*:2, actions=RemoveLTag,output:2}

2.2 Flow Translation

BNV performs flow-table translations to provide flowspace and dataplane isolation for virtualizing multiple-links to support arbitrary topologies. Packet tagging and metering are used to achieve topology virtualization. We maintain the virtual link information encoded in the packet header. We call it *lTag* for simplicity.

Refering to the virtual topology in Figure 3 with three links vL1, vL2 and vL3, the corresponding *lTags* being 1,2 and 3, We have two scenarios for mapping of the virtual links:

Case 1: Two connected virtual switches maps to different physical switches with a (in)direct link. (Example : vL1) In that case, any packet that travels vL1 is tagged with *lTag* 1 on the outgoing port of CLink1. Similarly, Flows are translated to attach the *lTag* in the match based on which virtual switch and virtual port the flowmod is meant for. Also, a drop-meter m is created to rate-limit the flows to the max-rate of 1G as in the virtual topology. For instance, assume a tenant adds a flow entry $\{in\_port:1, ip\_dest:x.x.x.x, actions=output:2\}$ to vS1 for routing a packet via link vL1. The translated flowmod is $\{in\_port:1, ip\_dest:x.x.x.x, actions=meter:1, lTag:1, output:2\}$.

Case 2: Two connected virtual switches maps to the same physical switches (vL3). Consider a FlowMod to enable ip-destination based forwarding from H1 to H2 are below:

$vS1:\{in\_port:1, ip\_dest:y.y.y.y, actions=output:3\}$
$vS2:\{in\_port:2, ip\_dest:y.y.y.y, actions=output:1\}$

Loopback link: LoopLink1 emulates vL3. The FlowMods that were pushed earlier can be implemented by performing the translation of the output ports to physical ports, and adding the appropriate lTag in the physical topology as in Table 1. Typically Loopback links can be dedicated to a single tenant and they can naturally emulate inter-switch links.

Metering: We employ two kinds of data-plane isolation using meters:

1. Shared links (cLink1): When multiple virtual links share a single physical link, they share the same queues. We mitigate the interference by restricting a single virtual link from exhausting the port buffers by partitioning the maximum burst size of a physical link (measured using techniques in [15]) to multiple virtual links.
2. Shared Switches (pSwitch1): When a single physical switch is sliced into multiple virtual switches, it's important to slice the buffer such that the virtual switches do not run out of buffer. In this case, we apply a meter to the flows from all ports of a virtual switch. We explain the implications and observations of metering in Section 4.3.

3 BNV Mapper

In this section, we present the formulation of the network embedder (BNV Mapper) as an ILP to maximize the fidelity of the embedding of the virtual topology and to minimize the overheads. The (user) input to the BNV Mapper is the virtual network topology that consists of a set of : 1) hosts V, 2) (virtual) switches S and 3) (virtual) links L. Each link is bi-directional and has a required bandwidth b. A link can be associated with either (1) two switches (corelink L_c) or (2) a switch and a host (hostlink L_h).

The user input is next converted into a form where the switches are broken down into (1) Core-link. A link between two switches is represented as a link between two switch-ports. (2) Host-link. A link between a switch-port and a host. In this way, each switch s can be represented as a set of switch-links or host-links. For a link that is associated with a switch, the required TCAM size $t(s)$ can be specified.

The physical topology consists of a set of: 1) physical server machines H, 2) physical switches R and 3) physical links Q. Similar to the virtual topology, Q can be categorized into corelinks Q_c and hostlinks Q_h. The available bandwidth of a link i is represented as B_i. Note that, Q_c comprises of both loopback links and backbone links (links between the switches).

We define a binary decision variable x_{iv}, which is set to one if a virtual corelink i is mapped to the physical corelink v, and zero otherwise. Similarly, y_{jw}, which indicates if a virtual hostlink j is mapped to a physical hostlink w. Mapping of hostlink automatically implies mapping of virtual host to physical server. The objective function is given below by Equation (1).

$$\min : \sum_{\substack{i\in L_c \\ v\in Y}} x_{iv} b_i + \sum_{\substack{m\in S \\ \{p,q\}\in R}} M^m_{pq} \qquad (1)$$

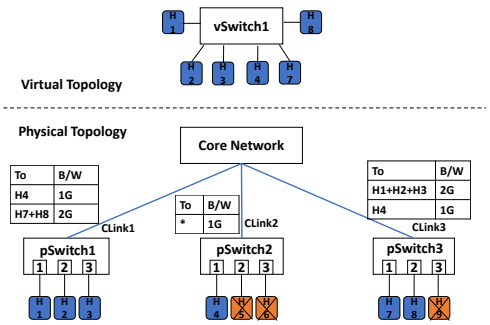


Figure 4: Allocation for big-switch abstraction

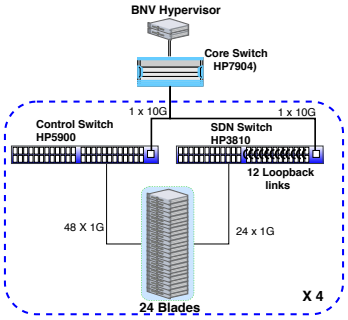


Figure 5: BNV testbed setup environment

The objective function has two terms. The first term represents the amount of resources to support mapping of corelinks ($x_{iv}b_i$) and the second term represents the resources needed to map a single virtual switch over multiple physical switches (M^m_{pq}). The overall goal is to minimize the usage of substrate backbone (core) links, since these links are (generally) under-provisioned relative to the hostlinks. This indirectly maximizes the usage of loopback links which provide higher fidelity in emulation. The constraints are:

$$\sum_{v \in Q_c} x_{iv} = 1, \forall i \in L_c \tag{2}$$

$$\sum_{w \in Q_h} y_{jw} = 1, \forall j \in L_h \tag{3}$$

$$\sum_{v \in Q_h} y_{jv} b_j \leq B_v, \forall j \in L_h \tag{4}$$

$$\sum_{v \in Q_h} y_{jv} c_j \leq C_v, \forall j \in L_h \tag{5}$$

$$\sum_{i \in L_c} \sum_{v \in Q^r_c} x_{iv} t(i) + \sum_{j \in L_h} \sum_{w \in Q^r_h} y_{jw} t(j) \leq N_r, \forall r \in R \tag{6}$$

$$z_{mn} = \sum_{v \in Q_c} x_{iv} b_i + \sum_{w \in Q_h} y_{jw} b_j, \quad \forall i \in L^m_c, j \in L^m_h, m \in S, n \in R \tag{7}$$

$$M^m_{pq} = \min(z_{mp}.z_{mq}), \forall m \in S, \{p,q\} \in R \tag{8}$$

$$\sum_{v \in Q_c} x_{iv} b_i + M^m_{p_v q_v} \leq B_v, \quad \forall i \in L_c, \forall m \in S, v \in Y, \{p_v, q_v\} \in R \tag{9}$$

Constraint (2) and (3) mandates each virtual corelink to be mapped to only one substrate corelink and similarly for hostlinks. Constraint (4) ensures that each physical hostlink is provisioned within it's capacity. Constraint (5) ensures that the physical host is not allocated beyond it's core capacity. The notation uses hostlink instead of host, however, in the model hostlink is synonymous to host. Also, Constraint (4) and (5) are needed only for VM-based provisioning like OpenStack. They are not needed for bare-metal provisioning methods since each virtual host is allocated an entire server blade. Constraint (6) makes sure we do not exceed the TCAM capacity bounds of the switch. TCAM specified for every virtual switch is split equally among it's corelinks and hostlinks for simplicity in allocation.

Big-Switch Abstraction. Constraint (7) defines a variable $z_{m,n}$ which depicts the total bandwidth of virtual links of virtual switch m that are mapped to physical switch n. Constraint (8) creates a R x R matrix for each virtual switch indicating the amount of intra-switch bandwidth between two physical switches mapped to a single virtual switch. It allocates the minimum link bandwidth generated by each physical switch for a given pair of physical switches mapping to same big virtual switch. Figure 4 illustrates an example of the embedding of a virtual switch with 5 hosts onto three substrate switches each mapping a different number of hosts. In Figure 4, each virtual switch is associated with a table containing the bandwdith to be limited for traffic to a particular set of hosts belonging to a physical switch. Finally, Constraint (9) makes sure the physical corelinks are provisioned within their capacity accounting both inter-switch links and intra-switch link utilization.

4 Testbed Implementation and Evaluation

We have implemented BNV over OpenVirtex [7] using OpenFlow 1.3. We have also integrated BNV with DeterLab [19] which provides bare-metal provisioning. We use Gurobi [3] as the ILP solver. The users submit their topology as a NS file[1]. The SDN switches are specified by defining the switch-type as "ofswitch".

BNV is deployed and currently functional on the production testbed of the National Cybersecurity Lab[2] for a year to provision SDN-based experiments with arbitrary topologies. The experimenters can completely use the functionalities supported by OpenFlow 1.3 (Meters, groups, etc.), and can develop and use custom network applications (BGP, congestion control, etc.). The network testbed layout is shown in Figure 5. The testbed

[1] Refer https://ncl.sg/bnvirtusage.pdf
[2] https://ncl.sg

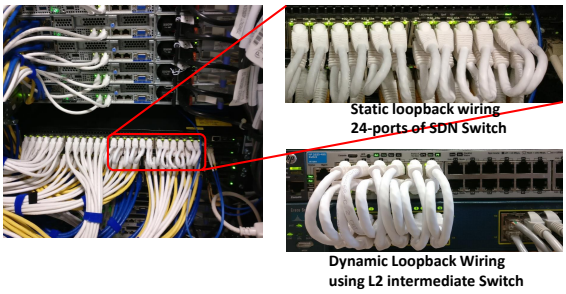
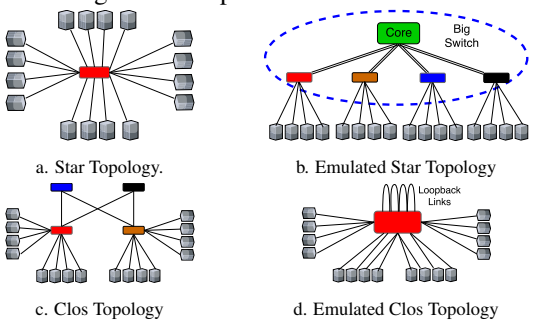


Figure 6: Loopbacks in Switches



a. Star Topology. b. Emulated Star Topology

c. Clos Topology d. Emulated Clos Topology

Figure 7: Topologies used for application fidelity

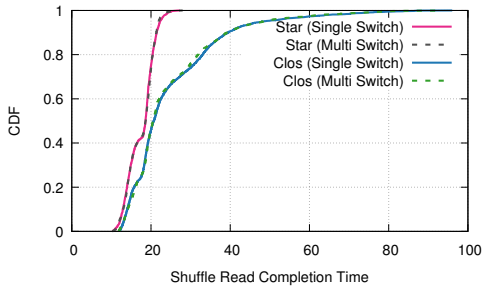


Figure 8: Performance comparison on emulation

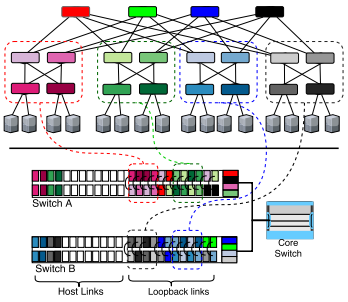


Figure 9: Mapping of FatTree4 on testbed

has four HP3800 SDN Switches, each connecting to a cluster of 24 Lenovo X3550 servers. The SDN switches are connected using a core-switch which is used only for L2 connectivity. The server blades are also connected to a control-switch, which is used for out-of-band management (IPMI, PXE booting). Each SDN switch has 12 loopback links and 1 10G uplink to the core-switch. Each loopback link is formed by connecting two ports using a short cable. Figure 6 shows the testbed with the static loopback wiring of 12 links per switch. We additionally perform software-configurable loopback for one cluster in our staging platform. In the following sub-sections we evaluate BNV's fidelity, isolation and embedding capabilities.

4.1 Performance Fidelity

In this section, we evaluate the fidelity of the virtual topology created by BNV by making use of one-to-many abstraction and many-to-one abstraction. The evaluation considers variations of two topologies: (1) a star topology with 1 switch and 16 hosts (Figure 7(a) and Figure 7(b)) and (2) a Clos topology with 4 switches and 16 hosts (Figure 7(c) and Figure 7(d)). We change the physical wiring of the testbed to implement the physical clos topology.

We run an Apache Spark application (wordcount on a 50GB file) on all 4 topologies and we plot the CDFs of shuffle read times on the four topologies in Figure 8.

We observe that the CDFs of the physical and virtual networks are very similar. The use of loopback links (Figure 7(d) vs Figure 7(c)) has the same application behaviour compared to actual topology. The use of longer physical hops and multiple physical switch to support one virtual switch (Figure 7(b) vs Figure 7(a)) do not affect the performance.

4.2 Topology Flexibility

Recent works on topology convertibility [28] demonstrate tangible gains in network performance by changing the network topology dynamically. BNV can leverage its inherent ability to map arbitrary network topology to allow researchers to experiment with different topologies quickly. This makes it possible to perform fine-grain optimization of topology for a work-load and ability to emulate traffic patterns reliably in a topology.

BNV can virtualize the network to support various topologies like FatTree [6], Clos, JellyFish[25], HyperX etc in a matter of a few seconds. We illustrate how a FatTree is mapped on to substrate topology in Figure 9.

We create four experiments with various topologies: Binary Tree (16 hosts, 15 switches), Star (16 hosts, 1 switch), FatTree with degree 4 (16 hosts, 20 switches), and JellyFish [25](16 hosts, 20 switches) and perform a wordcount application for a 50GB file in Apache Spark. We use custom partitioning in order to increase the inter-rack traffic. We perform 10 runs and plot the average shuffle read time for the four topologies in Figure 10.

We observe that tree topology has the longest shuffle read time due to its very limited bisection bandwidth. The star finishes fastest since it has full bisection bandwidth. FatTree and JellyFish perform somewhere in between as expected. ECMP [14] was used to split the traffic equally based on link utilization calculated at the controller in the case of FatTree and JellyFish.

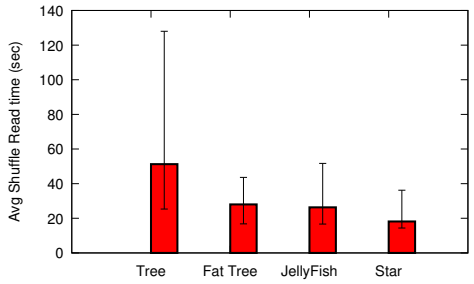


Figure 10: Apache Spark performance over various Topologies generated using BNV

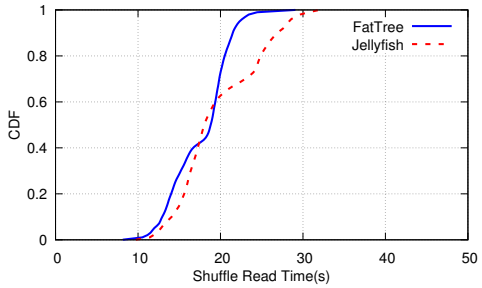


Figure 11: Comparison of FatTree and JellyFish for high intra-pod locality traffic

Now, taking a closer look at only two topologies: FatTree and JellyFish. FatTree and Jellyfish observe really close shuffle time. On an average, the shuffle time in JellyFish is about 7-8% lower. This is due to high inter-rack traffic. Then we vary the data placement in Spark to increase the intra-pod locality. We plot the CDF of the shuffle read times in Figure 11 for FatTree and Jellyfish. Interestingly, we observe an increased shuffle time (∼7.5%) in case of JellyFish compared to FatTree. To summarize our observation, JellyFish performs much better than FatTree when there are lot of inter-rack network traffic and FatTree performs better than JellyFish when there are lot of intra-rack exchanges. This performance behaviour observed is consistent with the observations made by recent work on convertible topologies [28]. Thus, BNV can achieve fidelity and flexibility in terms of its topology implementation.

4.3 Isolation

We spawn three experiments on BNV with the following topologies: 1) Expt-1: FatTree4 (16 hosts and 20 switches) running wordcount of 1GB file using Spark continuously with heavy inter-rack traffic, 2) Expt-2: JellyFish (random topology) topology with 16 hosts and 20 switches running iperf among all pairs and 3) Expt-3: Random topology using up the rest of the available switch-ports(8) and hosts(8) running ping between the pair of nodes with the longest hop-count. Initially, all experiments are idle with applications not running. The experimentation scenario is as follows :

1) Between 0-60 minutes, we run the application on each of the experiments for 20 minutes without interference.

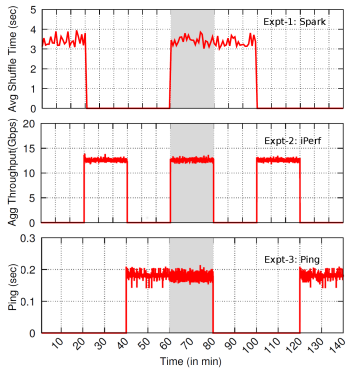


Figure 12: Concurrent experiments to check isolation

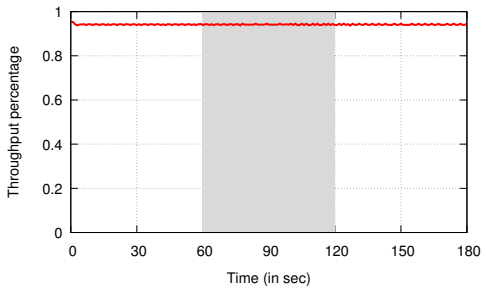


Figure 13: Observation of TCP traffic completely unaffected by UDP bursts in the same switch over other ports.

2) Between 60-80 minutes (shaded region), all three applications are run in parallel.

3) We repeat step (1) between 80-140 minutes.

We present the findings in Figure 12, where we plot different performance metrics such as average shuffle time for expt-1(top), aggregate throughput for expt-2(middle) and ping latency for expt-3(bottom). The shaded region (comprising 20 minutes) represents the portion of time the experiments were run in parallel, and the non-shaded regions depict the performance for individual runs. Observe that the applications see no perceivable difference in performance when the other experiments which share the same switches and backbone links are run.

Buffer partition: When two tenants share a physical link (typically backbone links), a burst of traffic from a single tenant could fill up the buffers and TX Queue, thus impacting the other tenants. Fortunately, we can achieve an approximate buffer partition by allocating the maximum burst for a particular virtual link or set of ports (using OpenFlow meters) during provisioning by considering the switch-port buffer-size to be one of the constraints in the BNV mapper. In the following, we perform two experiments to measure the impact of bursty traffic.

Bursty transmission on a specific output port does not affect the traffic on other switch-ports: We perform an experiment with two tenants allocated different slices of the same switch: one tenant performing UDP bursts (sending at the maximum rate) of 11 hosts to 1

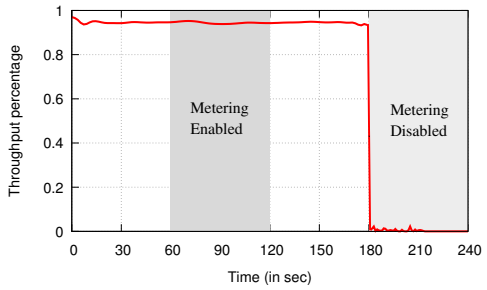


Figure 14: Observation of TCP traffic completely unaffected by UDP bursts in the same egress (shared)ports.

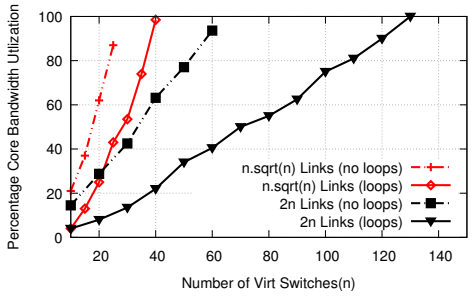


Figure 15: Mapping of Random Topologies

host, and the other generating a TCP traffic at line-rate on hosts running over two other ports. In Figure 13, the shaded region is the period where the UDP bursts occur. Clearly, the UDP traffic has no impact on the TCP traffic due to the buffer isolation provided. Although switch architectures [5, 11] perform buffer isolation inherently, in order to be generic and not rely on specific switch architectures, we allocate a burst-rate for all the ports belonging to each virtual switch, and apply the same meter to all the flows belonging to the virtual switch's ports. In this way, we guarantee that the amount of buffer used by a single (or a set of) flow(s) is bounded. Although, this implies that the switch-buffer may be under-utilized in certain circumstances, we reckon that this measure is necessary to guarantee isolation, fidelity and repeatability of experiments.

Burst transmission on shared ports (and links) minimally affects the traffic: We perform an experiment, where the backbone link is shared by two virtual links (either same or different tenants). vlink1 is allocated 1/10th of the bandwidth of the physical link, and the rest of the bandwidth is allocated to virtual-link2. We transmit TCP flow on virtual-link1 and 10 different UDP flows on vlink2 to saturate the physical link. We observe the impact of UDP traffic-bursts of vlink2 on vlink1. The results are shown in Figure 14. We can observe that the TCP throughput is not affected even during the presence of burst transmission. This is primarily because we limit the maximum burst per port. At the 180^{th} second, we turn off metering-based isolation. We notice that the TCP flow on vlink1 is completely starved due to bursty UDP flows on vlink2 in the absence of burst/ rate limiting.

4.4 Network Mapping Simulation

In this section, we evaluate through simulation the network embedding efficiency of *BNV* by considering a network consisting of 5 switches consisting of 48 down-link ports (1G each) of which, 24 ports to be connected to physical servers, and the 24 remaining ports are used form loopback links (12 loopback links per switch).

We embed random topologies with increasing number of switches and links to the physical topology. We consider two sets of random topologies with a fixed number of n switches: 1) Number of links = $2n$ and, 2) Number of links = $n\sqrt{n}$. We perform the network mapping with 10 set of topologies for each size of random topology. We plot the percentage of backbone link's bandwidth utilization against the topology size (number of switches) in Figure 15. For topologies with $2n$ links, the gain is clearly evident, as we are able to fit bigger topologies (up to 130 nodes and 260 links) using loopback links compared to only 60 nodes (and 120 links) and without loopback links, leading to approximately $2\times$ gain. Similarly, for topologies with $n\sqrt{n}$ links, we are able to fit topologies with 40 switches (252 links) with loopback links, compared to 25 switches (125 links) leading to $2\times$ gain. Additionally, we embedded topologies from Topology Zoo [16] and observed successful mapping for 96.5% of topologies.

Take-away: *BNV* can emulate a network topology with high fidelity and repeatability by maintaining the key characteristics of each topology while guaranteeing isolation for each experiment. *BNV* Mapper can embed a large collection of topology efficiently. The addition of loopback links increases the scalability significantly.

5 Related Work

Network Hypervisors: Network hypervisors like FlowVisor [24], OpenVirteX [7] and FSFW [2] are used in major testbeds like GENI [9]. However, these works can only provide either the corresponding physical topology or a subset of it. [23] explains ways to provide SDN experimentation in testbeds. However, supporting embedding of arbitrary network topologies is not explored by existing hypervisors or testbeds.

Switch Abstractions: NVP[17] performs network virtualization using virtual switches in the hypervisor. Since, it is based on hypervisor, it is hard to produce arbitrary topologies using software switches due to packet processing CPU overhead at the VMs or servers. Trellis [10], VINI [8] propose creation of arbitrary network topologies using containerization and tunnel abstractions. However, they cannot achieve line-rate needed for heavy workload testing. Similarly, Mininet [13] and Maxinet [27] are popular network emulation tools used to construct SDN-based virtual topologies on server/clusters. These are excellent tools for functional

testing, however they cannot scale to emulate large networks carrying traffic at line-rate. CrystalNet [18] provides an extensible testing environment to mimic production networks using a large number of virtual machines from public cloud infrastructures. However, it does not claim fidelity of network dataplane. BNV can bridge this gap by providing fidelity of the network data plane. Recently, MiniReal [26] has proposed an approach to achieve multi-switch abstraction using a single bare-metal switch by modifying the software of the switches and also loopback link. BNV is different in that it performs virtualization functioning as a network hypervisor and does not need to modify the switch software. Additionally, it can dynamically modify topologies at run-time, while providing repeatable performance.

Network Embedders: There exists a variety of virtual network embedding techniques [12, 22]. Other experimental setups like DeterLab [19] use Assign [20]. Assign [20] uses simulated annealing to get an embedding for a particular network topology. BNV Mapper, since it tries to embed arbitrary topologies, needs exact port mapping. Additionally, BNV Mapper has a different objective from the works in [12] to reduce the burden on the backbone links, and use loopback links instead.

6 Conclusion

We developed *BNV* (Bare-metal Network Virtualization), which provides high-fidelity network experimentation at data-plane and control-plane using programmable switches. *BNV* can support arbitrary network topologies using a unique method of creating loopbacks in switches in order to provide high fidelity. We propose an ILP-based formulation for efficient embedding of complex topologies to the substrate. We have built *BNV* on top of OpenVirtex, and deployed on a production testbed. *BNV* accurately emulates the characteristics of the topologies implemented over the substrate network while isolating multiple experiments.

Acknowledgement: This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity RD Programme (grant no. NRF2015NCR-NCR002-001).

References

[1] Cloudlab. http://docs.cloudlab.us/planned.html.

[2] Flowspace Firewall. http://globalnoc.iu.edu/sdn/fsfw.htm.

[3] Gurobi. http://www.gurobi.com/.

[4] National Cybersecurity Lab. https://ncl.sg/.

[5] Intel Ethernet Switch Family Memory Efficiency. White Paper, 2009.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[7] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. Openvirtex: Make your virtual sdns programmable. In *HotSDN*, 2014.

[8] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: Realistic and controlled network experimentation. In *SIGCOMM*, 2006.

[9] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. GENI: A federated testbed for innovative network experiments . *Computer Networks*, 2014.

[10] S. Bhatia et al. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. CoNEXT, 2008.

[11] S. Das. Broadcom smart-buffer technology. 2012.

[12] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys Tutorials*, 2013.

[13] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *CoNEXT*, 2012.

[14] C. Hopps. Analysis of an equal-cost multi-path algorithm, 2000.

[15] G. Jin and B. Tierney. Netest: a tool to measure the maximum burst size, available bandwidth and achievable throughput. In *ITRE*, 2003.

[16] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 2011.

[17] T. Koponen et al. Network virtualization in multi-tenant datacenters. In *NSDI*, 2014.

[18] H. Liu et al. Crystalnet: Faithfully emulating large production networks. In *SOSP*, 2017.

[19] J. Mirkovic, T. V. Benzel, T. Faber, R. Braden, J. T. Wroclawski, and S. Schwab. The deter project: Advancing the science of cyber security experimentation and test. In *IEEE HST*, 2010.

[20] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *SIGCOMM CCR*, 2003.

[21] R. Ricci and E. e. a. Eide. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *;login*, 2014.

[22] R. Riggio, F. D. Pellegrini, E. Salvadori, M. Gerola, and R. D. Corin. Progressive virtual topology embedding in openflow networks. *2013 IFIP/IEEE IM*, 2013.

[23] S. S R, J. Mikovic, P. G. Kannan, C. Mun Choon, and K. Sklower. Enabling sdn experimentation in network testbeds. SDN-NFVSec, 2017.

[24] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *OSDI*, 2010.

[25] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, 2012.

[26] S. Y. Wang and I. Y. Lee. Minireal: A real sdn network testbed built over an sdn bare metal commodity switch. In *ICC*, 2017.

[27] P. Wette, M. Drxler, and A. Schwabe. Maxinet: Distributed emulation of software-defined networks. In *IFIP Networking*, 2014.

[28] Y. Xia, X. S. Sun, S. Dzinamarira, X. S. Wu, Dingming an Huang, and T. E. Ng. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *ACM SIGCOMM*, 2017.