Towards Building Secure and Reconfigurable Virtual Networks on Multi-Tenant Data Centers

Jinwoo Kim
School of Software
Kwangwoon University
jinwookim@kw.ac.kr

Jaehyun Nam*
Department of Computer Engineering
Dankook University
jaehyun.nam@dankook.ac.kr

***Abstract*—Network virtualization (NV) has been widely used today in data centers to meet the multi-tenancy requirement—a key to achieving Infrastructure as a Service (IaaS) in a modern cloud environment. However, even though NV is a popular solution employed by network operators, we argue that most solutions still have challenges in terms of (i) configuration, (ii) management, and (iii) security; all of which hinder the deployment of secure and practical *virtual networks* that tenants wish to construct. To bridge the gaps, we propose *LinkWire*, a new NV system that produces secure and reconfigurable virtual networks. For this, *LinkWire* leverages extended Berkeley Packet Filter (eBPF), the recently-adopted in-kernel programmable networking technology. We also illustrate several use cases to show how our system brings benefits.**

***Index Terms*—Network Virtualization, Network Configuration, Extended Berkeley Packet Filter**

I. Introduction

The recent popularity of IaaS (Infrastructure-as-a-Service) is the key to bringing cloud success. IaaS allows tenants to configure their infrastructures and develop applications without considering physical constraints. According to Gartner's report [1], the IaaS market grew by 41.4% in 2021, dominating the other two cloud services (i.e., PaaS and SaaS) due to the widespread use of virtualization technologies. For instance, host-level virtualization (e.g., virtual machines, containers) and network-level virtualization (e.g., VMware NSX) facilitate sharing of physical resources between tenants.

Cloud tenants often utilize virtual machines (VM) as nodes that run specific network functions, i.e., network function virtualization (NFV). In the context of IaaS, they are used for diverse use-cases, such as building an SDN (Software-Defined Networking) network, a BGP VPN, or a security service chain (e.g., IDS/IPS). Those use cases require tenants to construct *virtual networks*[1] over VMs (assigned to a tenant) by relying on network virtualization (NV) techniques [2]–[4].

In order to meet the tenants' requirements, virtual networks must be constructed flexibly and reliably. However, we tackle that existing NV solutions have several limitations:

Configuration Challenge. When using NV solutions, tenants should manually configure a virtual link between VMs, specifying how to en/decapsulate packets through a tunneling protocol, which is not problematic in traditional data centers where a small number of hosts reside. However, a typical multi-tenant data center has many VMs (nodes). Creating a virtual network (modeled as an undirected graph) with n virtual nodes theoretically requires configuring $2n(n+1)$ virtual links at the worst. Thus, tenants have to spend much time on manual configurations.

Management Challenge. Virtual networks should be updated when tenants want to deploy a new network topology or the physical topology changes. Adjusting virtual links between nodes involves the re-installation of complicated tunneling and segmentation rules (e.g., IP addresses and routing policies). As VMs are normally distributed across multiple physical hosts, tenants need to address rule conflicts that may occur between different network subnets. It is difficult to resolve the rule conflict problem from the current decentralized environment.

Security Challenge. It is possible that an attacker manages to compromise a VM, aiming at performing lateral movement within a target data center. Hence, a tenant could employ network segmentation to prevent attackers from accessing the tenant's virtual network. However, while network segmentation solutions isolate traffic between different segments, they do not do so between VMs (i.e., per virtual link). Thus, attackers could mount network attacks *within* a segment, such as eavesdropping, spoofing, and man-in-the-middle attacks.

In this paper, we propose a new NV solution, called *LinkWire*, that allows tenants to build secure and reconfigurable virtual networks. Our key idea is to leverage eBPF (extended Berkeley Packet Filter), an instruction set and execution environment supported by the Linux kernel. With eBPF, one can implement a general program executed in low-level network stacks. By utilizing this benefit, we aim to address the challenges mentioned above as follows: First, *LinkWire* automatically produces eBPF rules according to the tenant's specifications for a virtual network. Following this, our eBPF programs (installed at each VM) then en/decapsulate packets to build a virtual link (i.e., tunnel) between VMs. Second, *LinkWire* keeps monitoring the tenant's specifications and the physical topology changes to update their associated rules without conflicts. Third, *LinkWire* achieves the inter-VM isolation by utilizing two Linux kernel hooks: (i) XDP (eXpress DataPath) and (ii) TC (Traffic Control). XDP is the hook where eBPF programs can perform fast RX packet

* Jaehyun Nam (jaehyun.nam@dankook.ac.kr) is the corresponding author.
[1]We define a *virtual network* by a graph that consists of virtual nodes and links, virtually built upon physical (or substrate) nodes and links.

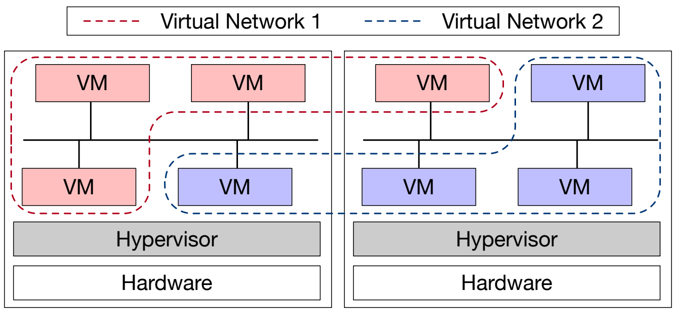

Fig. 1. Example virtual networks (i.e., network segments) built by existing NV solutions.

processing, enabling us to implement early packet inspection and filtering. With TC, *LinkWire* implements encapsulation of TX packets for creating end-to-end tunnels (i.e., virtual links).

Our contributions are summarized as follows:

- Design of *LinkWire*, a new NV system that can automatically create virtual networks across VMs according to a tenant's specification.
- Practical and secure NV methodology that utilizes in-kernel network stacks, being able to run independently to the underlying hypervisors.
- Demonstration of its utility with diverse deployment scenarios, such as building an SDN network, a BGP VPN, and a security service chain.

II. Background and Motivation

This section presents the background and motivation to understand the need for *LinkWire*.

A. Network Virtualization

In general, a hypervisor refers to software that allows to create and run multiple virtual machines (VM) upon a physical host. Its noticeable feature is to provide host virtualization that emulates guest operating systems via resource sharing for underlying hardware, achieving multi-tenant environments. While containers are considered an alternative to VMs today, VMs still play an important role in cloud data centers due to their better isolation and security.

In addition, tenants often want to construct different network topologies for different workload types. However, it is important to note that existing hypervisors do not support the creation of distinct network domains across other hosts. To complement this drawback, several network virtualization solutions have been proposed, such as NVGRE (Network Virtualization using Generic Routing Encapsulation) and VXLAN (Virtual eXtensible Local Area Network). The common goal they aim to achieve is to group VMs into the same broadcast domain (or segment) regardless of the location in a physical network. Thus, it enables tenants to build an (isolated) layer-2 *virtual network* over a layer-3 network (see Figure 1). Those solutions are widely used in modern data centers to achieve multi-tenancy.

Unfortunately, whereas NV solutions could isolate a broadcast domain, we observe that they are incapable of isolating

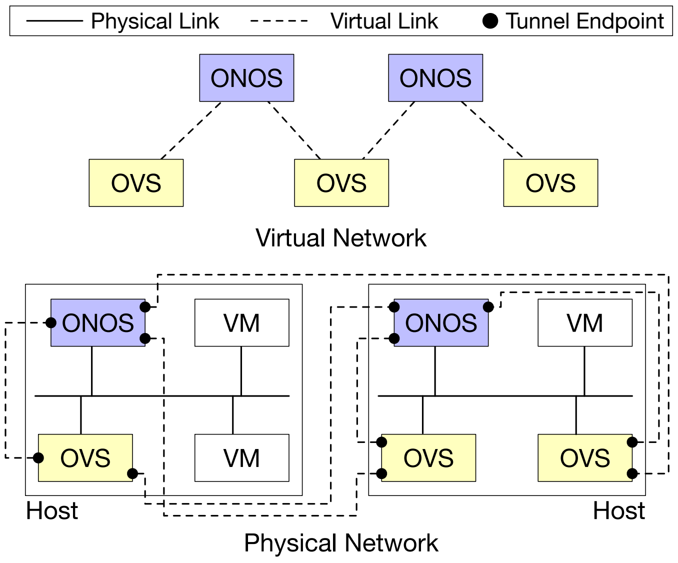

Fig. 2. A motivating example of building an SDN (virtual) network built upon the physical network.

end-to-end communication (i.e., virtual link) between VMs. This is because most NV solutions rely on tunneling between hypervisors that have tunnel endpoints[2]. Thus, they could create tunnels between hosts, but could not do so between VMs. From a topology view, it is equivalent to just having a bus network topology of VMs; thus, tenants could not design a virtual network topology that consists of diverse VM links.

B. Motivation

One could argue that it is possible to run VPN (Virtual Private Network) protocols (e.g., IPSec, PP2P, L2TP) to create an isolated virtual link (i.e., tunnel) between VMs. This way, tunnel endpoints place on VMs instead of a hypervisor. However, we reveal that they have several limitations in terms of security and practicality. In what follows, we discuss the difficulty of generating a virtual network across VMs with those solutions by introducing a motivating scenario.

Cloud tenants often want to build an overlay SDN network to control traffic between their VMs in a centralized manner. In doing so, it is common to run software-based SDN switches (e.g., OVS [5]) and a controller cluster (e.g., ONOS [6]) for flexibility and scalability. One important issue tenants should consider is how to construct a secure SDN control path, which is responsible for control channels (e.g., OpenFlow) between controllers and switches. From a security perspective, attackers must not distinguish the control path from the data path because they can conduct harmful attacks that target the control channels or the controllers if they know so [7]. To achieve this goal, it is desirable to build an isolated SDN control path.

However, building isolated SDN control paths upon VMs requires complicated link configurations, as shown in Figure 2. To establish virtual links between VMs, a tenant has to perform

[2]Note that tunnels are established through the encapsulation of a packet with additional headers, which are inserted and removed at the *tunnel endpoints*.

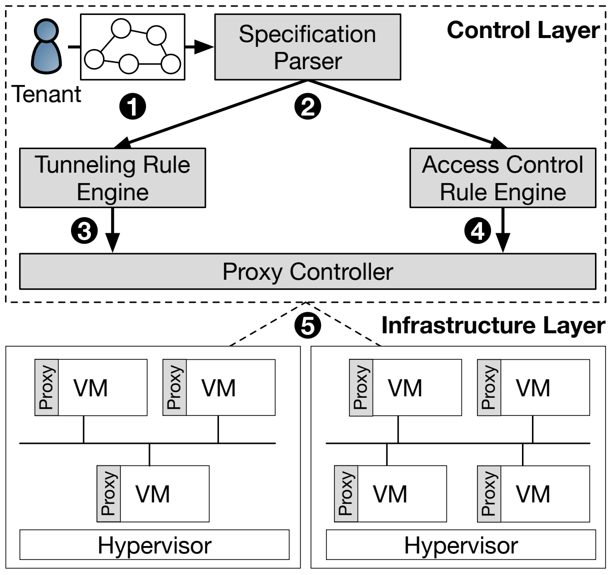

Fig. 3. The overall architecture of *LinkWire*.

VM-by-VM configurations for tunnel endpoints. To formalize this problem, suppose that we want to build fully connected control paths between controllers and switches. This can be cast to drawing a complete bipartite graph that consists of m controller nodes and n switch nodes, having mn links. Theoretically, a tenant has to configure $2mn$ tunnel endpoints per VM, which is time-consuming and error-prone.

III. System Design

This section presents the design of *LinkWire*.

A. Architectural Overview

Figure 3 illustrates the overall architectural of *LinkWire*, which consists of two high-level components: (i) the *control layer* and (ii) the *infrastructure layer*. The former is responsible for orchestrating topology management jobs in a centralized manner, while the latter is in charge of running VMs and executing commands the control layer instructs. To deploy *LinkWire*, tenants should pre-install the *proxy* for each VM at the infrastructure layer to communicate with the control layer. Because the proxy is designed to utilize the Linux kernel's low-level network stacks (i.e., XDP and TC), tenants do not need to modify any service or OS running at VMs and the underlying hypervisors. Once the proxy is installed and a communication channel is established, the control layer adds it to a node pool.

B. LinkWire Workflow

At the high level, *LinkWire* operates with the following procedure: (1) *LinkWire* first takes the tenant's topology specification as an input, and the *specification parser* analyzes it to obtain information about virtual links and routing rules. (2) They are delivered to the *tunneling rule engine* and the *access control rule engine*, respectively. (3) The tunneling rule engine generates configurations and rules for creating virtual links (i.e., tunnels) between VMs. (4) Next, the access control rule engine produces blocking/allowing rules according to the membership of a virtual network, which will be installed in VMs. (5) Those rules are sent to each VM via the *proxy controller* that is responsible for keeping control channels with proxies. Note that the control channels are fully encrypted by TLS/SSL.

C. Topology Abstraction

As mentioned before, *LinkWire* takes a tenant's topology specification as an input. One challenge is how to provide an intuitive abstraction for tenants so that they can compose a virtual network easily. A natural solution for this is to adopt a *graph abstraction* that specifies VMs by a set of *vertices* and virtual links by a set of *edges*. To build a graph, a tenant can choose vertices by seeing the list of currently available VMs provided by *LinkWire* and drawing edges between them.

LinkWire allows a tenant to specify various constraints on the vertex and edge labels. For example, a tenant can specify physical constraints on a virtual network, such as the maximum number of virtual links (e.g., "VM A cannot create more than 10 virtual links.") and access control rules (e.g., "VM A should not communicate with VM B."). *LinkWire* allows tenants to write all the information with graph description languages (e.g., GraphML, DOT).

When a graph is submitted to *LinkWire* at an initial time, it should be fully analyzed to extract information necessary for the subsequent modules. For this purpose, *LinkWire* uses a depth-first search (DFS) algorithm that traverses all nodes and links. When visiting each vertex, *LinkWire* checks whether a vertex or edge has a label. If so, it extracts constraints from the label. *LinkWire* terminates the traversal if all vertices and edges are covered.

LinkWire also should be able to reconfigure virtual networks on the fly when a tenant resubmits a different specification. An efficient solution to doing the task is to partially update configurations currently applied to the infrastructure layer instead of creating new ones at scratch. To do so, *LinkWire* compares a newly submitted graph with the old one to find a *delta graph* (i.e., a subgraph that only has newly added links) between them. Once obtained, *LinkWire* performs partial updates by only inspecting the delta graph.

D. Building Virtual Links

Here, we aim to construct inter-VM virtual links that meet *(i) security* and *(ii) reconfigurability*: For security, the packets interchanged between VMs must be hidden from an attacker who may analyze packet headers. Thus, *LinkWire* needs to present a way of building a secure forwarding path between VMs, which does not reveal VM identities. However, from a practical perspective, it is difficult for a tenant to attach an additional spare cable in a data center for network isolation. In addition, for reconfigurability, the virtual links should be deployed conveniently and updated fast enough. Modifying physical connectivity would require the tenant's significant time and effort.

Our approach is to leverage existing tunneling protocols (e.g., NVGRE, VXLAN, IP in IP) to construct virtual links compatible with the IP network stack. This way, the tenant does not need to care about the deployment of *LinkWire*, and the virtual network can be reconfigured by simply updating tunneling rules. To eliminate the need for manual configuration jobs, *LinkWire* automates the configuration process for all eBPF programs installed at each VM. Once a tenant gives his/her preferred tunneling protocol along with the topology specification (that specifies IP addresses), *LinkWire* generates suitable eBPF rules. The rules are updated into the maps of an eBPF program, attached in the Linux TC hook, as the eBPF program needs to en/decapsulate packets at TX and RX paths.

E. Inter-VM Access Control

Although *LinkWire* encapsulates packets, an attacker may inject his/her packets into a (non-authorized) virtual network by guessing its subnet. To prevent this, *LinkWire* generates a whitelist from the topology specification. Thus, the eBPF rules are updated into an eBPF program attached to the Linux XDP hook. The XDP is the earliest hook of the RX path; thus, we can achieve high-performance packet filtering at the lowest network layer. At the XDP, the eBPF program reads the rule and only allows the packet whose IP/MAC addresses belong to the whitelist.

IV. Use Cases

This section presents the possible use-cases of *LinkWire*.

A. SDN Network

As shown in Section II-B, deploying an SDN network topology requires complicated manual configurations with existing NV solutions. In contrast, *LinkWire* allows a tenant to build an SDN network easily. In the specification, the tenant defines the control path and data path with different IP address ranges (i.e., subnets) for building an out-of-band control channel[3]. *LinkWire* then automatically generates distinct tunneling rules so that the paths use different encapsulation headers.

B. BGP VPN

Many enterprise tenants employ VPN to build a private WAN that connects geographically distributed networks. For this purpose, they run BGP processes (e.g., Quagga [8]) and establish BGP connections from VMs to their VPN sites (outside data centers) [9]. When establishing BGP connections, a tenant needs to secure a BGP channel; otherwise, an attacker can inject a malformed packet (i.e., a BGP poisoning attack). Such threats could be avoided through *LinkWire*, which performs strict access control at the XDP layer for unauthorized access.

[3]Note that the out-of-band control channel means that the SDN control path is (physically or virtually) isolated from the data path.

C. Security Service Chain

NFV is widely employed in data centers due to its cost efficiency and flexibility. With NFV, tenants could deploy any (software-based) network function on the VM they want to run, which is cheaper and more elastic than proprietary hardware. Typically, a tenant runs network functions on a few VMs (i.e., network choke points) for resource saving. As network traffic may not go through those VMs, the tenant needs to determine a *service chain* of traffic, specifying a packet processing sequence between VMs. Whereas composing the service chain is a complicated task in existing NV solutions, *LinkWire* facilitates the daunting job by automatically producing routing rules.

V. Conclusion

In this paper, we present *LinkWire*, a new NV system for secure and reconfigurable virtual networks built on multi-tenant data centers. We sketch the concept of *LinkWire*, system design, and its potential use cases. In future work, we will implement *LinkWire* and evaluate its security and performance by comparing it to existing NV solutions.

Acknowledgment

This research was supported by the MSIT (Ministry of Science and ICT), under the National Program for Excellence in SW (2017-0-00096), supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

References

[1] "Gartner forecasts worldwide public cloud end-user spending to reach nearly \$500 billion in 2022," 2022, https://www.gartner.com/en/newsroom/press-releases/2022-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-500-billion-in-2022.

[2] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, N. Mckeown, and G. Parulkar, "Can the production network be the testbed?" in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[3] H. Wang, A. Srivastava, L. Xu, S. Hong, and G. Gu, "Bring your own controller: Enabling tenant-defined sdn apps in iaas clouds," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[4] G. Yang, B.-Y. Yu, S.-M. Kim, and C. Yoo, "Litevisor: A network hypervisor to support flow aggregation and seamless network reconfiguration for vm migration in virtualized software-defined networks," *IEEE Access*, vol. 6, pp. 65 945–65 959, 2018.

[5] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[7] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, and Y. Yang, "The crosspath attack: Disrupting the sdn control channel via shared links," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 19–36.

[8] "Quagga software routing suite," 2022, https://www.quagga.net/.

[9] "Bgp vpn interconnection service overview," 2022, https://docs.openstack.org/networking-bgpvpn/ocata/overview.html.