Design of a Real-Time Traffic Mirroring System

Liang-Min Wang
Intel Corp
liang-min.wang@intel.com

Tim Miskell
Intel Corp
timothy.miskell@intel.com

John Morgan
Intel Corp
john.m.morgan@intel.com

Edwin Verplanke
Intel Corp
edwin.verplanke@intel.com

***Abstract*—Network monitoring and analysis is a vital component of network infrastructure reliability, availability, and sustainability. As network bandwidth consumption continues to grow exponentially, real-time traffic data analysis becomes increasingly challenging and expensive. In many cases, monitoring can only be achieved via hardware Test Access Point (TAP) devices. Due to the intrusiveness and inflexibility of deploying such hardware, this approach is intractable in an SDN environment, where network resource allocation programmability is key to network orchestration and deployment. In our prior work we developed a technology to improve VIRTIO port mirroring via network device mirror offloading. In this paper, we extend our mirror offloading work to include a novel mirror tunnel design. Mirror tunneling technology transports traffic from any device capable of supporting a software TAP to a mirror tunnel device that supports mirror hardware offloading. In addition, we develop a mirror controller based upon SR-IOV technology and the kernel netdev interface. With minimal changes required for the traffic source network function, the host system can dynamically reconfigure traffic mirror management settings via pre-existing kernel-based Internet Protocol (IP) management tools.**

Keywords— SDN, Port-Mirroring, SR-IOV, NETDEV.

I. INTRODUCTION

Network bandwidth demand continues to grow exponentially, particularly with the sudden surge in employees and students working remotely due to the COVID-19 pandemic, further increasing strains on limited networking resources. Resource virtualization is critical to sustain network infrastructure deployments, e.g. factoring in peak time demand along with resource allocation. This requirement has further amplified adoption of SDN-based designs [1]. Moreover, network service deployments based upon SDN orchestration allows services to be deployed on-demand. Dynamic resource allocation drives the need for real-time resource auditing, which is usually achieved by means of monitoring services. Prior art presented in [16, 17, 20, 21] have provided ample use cases on how monitoring can be used for the purposes of network management. Nevertheless, real-time monitors demand dedicated hardware (HW) resources for the purposes of capturing traffic. As a result, existing monitoring tools are mostly implemented on routers and switches. Monitoring protocols and RFCs including SNMP [2], RMON [3], and NetFlow[4] all propose methods for traffic mirroring via a router or switch system. Implementing traffic monitoring over a non-router device continues to be a challenge due to the requirements of real-time packet capture services [7,8]. In addition, deploying a traffic monitor in an SDN environment adds another layer of complexity given that the traffic source and the mirroring service may reside in two different virtualized environments, i.e. VMs or containers. The challenges related to real-time mirroring between network services of different virtualized address domains can be summarized as follows:

(1) **Source Traffic Mirroring.** The mirroring service should introduce minimal impact on the source network function.

(2) **Transporting Mirrored Traffic.** The mirroring service should be able to deliver packets to monitoring devices via intra-domain networking.

(3) **Monitoring Service Provisioning.** The network monitor function (NMF) should deployable on-demand based upon the run-time workload and throughput level requirements.

Hardware based approaches including routers, switches, and Field Programmable Gate Arrays (FPGAs) can provide dedicated resources to meet the first two requirements. However, deploying scalable monitoring services on-demand over fixed function accelerators is challenging and requires separate resources for each TAP request. On the other hand, software (SW) implementations provide greater flexibility, particularly in the available approaches for meeting the third requirement. However, the incurred overhead, in terms of mirroring along with encapsulation and decapsulation for transportation [19] may be excessive depending on the source network service. As a result, these approaches either are relegated to low throughput environments or require the use of down sampling, e.g. sFlow, prior to analysis. For of our approach we expand on prior work [5] by leveraging improvements to VIRTIO mirroring performance via hardware offloading (HWOL). We present a novel "mirror tunneling" technique that extends HWOL to virtually *any* device.

The remainder of this paper is organized as follows. Section II provides context for the advanced NIC features that our design is based upon. Section III provides details regarding our design, followed by Section IV which contains our performance results. Section V concludes the paper with a summary of our key contributions along with our future work.

II. Background for NIC Offloading

In this section, we provide a description of key HW and SW technologies for which our design is based upon. In this case, we employ three advanced NIC features and one SW feature available in user-space drivers, e.g. the Data Plane Development Kit (DPDK) [18].

(1) **Device level network function traffic mirroring.** This is an advanced feature supported by the network devices discussed in [12]. This mirroring feature occurs between *ports*, i.e. network functions, in the same device. As expounded in [6], HWOL can be used to improve VIRTIO port mirroring performance. As described in [9], traffic from the network service function (NSF) residing in one VM, VM1, is mirrored to the NMF in another VM, VM2, for analysis. With para-virtualization technology [11], each virtual device, i.e. VIRTIO, is hosted by a backend driver, i.e. VHOST. As a result, port mirroring is achieved through packet copy operations that occur in the backend driver. Compared to the case when mirroring is disabled, the associated overhead results in an approximately 70% decrease in packet forwarding throughput [6,9], which can be reduced by employing device level network function traffic mirroring. The throughput improvement is achieved by replacing the VIRTIO-based NMF with a SR-IOV NMF and replacing data copy with device mirroring. As a result, traffic is mirrored from the NSF to the NMF without costly data copying. To take advantage of this feature the backend driver must execute on a network device that inherently supports traffic mirroring. Since port mirroring is triggered whenever there are ingress or egress packets, this approach can result in artifacts, i.e. the mirrored traffic generated in the backend device. To resolve the issue, the design presented in [9] introduces a VLAN-based traffic mirroring transportation framework that allows the switch to discard or forward the mirrored traffic depending on a pre-defined set of flow table rules.

(2) **Device network function transmit loopback.** Our design employs transmit loopback, a device level feature, so that mirror operations via HWOL no longer require the use of routers and switches, as in the aforementioned VLAN-based approach [9]. Transmit loopback is a standard NIC feature that is designed primarily for self-test purposes. For devices that support transmit loopback on a per-VF basis [13], this feature can be used to *gate* the artifacts back to the RX interface rather than continually transmitting the traffic over the external interface. With the mirror tunnel approach introduced in this design, the tunnel device effectively becomes a *passive* device. Specifically, since we loop TX traffic back to the RX interface, our design introduces no change to the actual traffic in the network.

(3) **Device network function link event communication.** We employ the PF-to-VF link status change event notification mechanism to implement a centralized traffic mirroring controller that can be used by any SR-IOV capable device along with a hypervisor. The notification system available in the Linux kernel allows an SR-IOV enabled NIC device to configure each Virtual Function into one of three link modes: enabled (always on), disabled (always off) or auto (the VF follows the PF link state). Our design leverages the first two modes to enable and disable mirror tunneling. By combining this PF management capability with a SW TAP we can realize the first requirement, i.e. mirroring traffic while minimizing impact on the source network function. When ingress and/or egress traffic are captured via a SW TAP the metadata of the source traffic is modified as the traffic is mirrored. The metadata enables HW to perform per-packet VLAN insertion upon transmission. To avoid introducing artifacts on the source traffic, the metadata is restored after the packets are transmitted via the mirror tunnel device.

(4) **DPDK pre-TX and post-RX packet processing.** For the current design, the SW TAP is achieved through the pre-TX and post-RX driver callback interface provided by DPDK [18]. However, the same technique can be applied through other user-space or kernel-space drivers, e.g. drivers that employ eBPF filters [10] to process pre-TX and post-RX traffic for other network functions. The SW TAP includes a per-descriptor, i.e. per-packet, VLAN insertion transaction that is supported by HWOL to the network device. By leveraging programmable VLAN insertion, our design enables efficient scaling of the NMF. Specifically, by allowing applications to insert different VLAN tags the monitoring service can leverage this for the purpose of scaling computational resources. As a result, our design meets the third requirement, i.e. providing the ability to scale resources in support of the NMF. As described further in [9], VLAN tags can also be used to support the intra-domain transportation of mirrored traffic. Therefore, our design effectively meets the second requirement for a real-time monitoring system.

III. Design

Figure 1 presents the framework for the implementation of a traffic mirroring system via mirror tunnel devices. In this case, the traffic from the NSF is captured and sent to the NMF. For an SDN or cloud environment, the NSF and NMF can reside in different VMs, or one may reside on the host while the other resides in the guest. This diversity in configuration requirements makes deploying *HW* TAP devices challenging if not impossible. In our design, the service network interface, which forwards traffic to the NSF, can be either a physical or logical network interface. For the purposes of this paper, we refer to a network interface that is directly supported by a NIC as a *physical* network interface, which includes both PF and VF functions, i.e. in the case where device virtualization technology is applied. By contrast, a *logical* network device is a SW emulated network device, e.g. a VIRTIO interface, indirectly managed by a backend physical device driver [11]. With reference to Fig. 1, there are two ethernet bus devices: (1) the NSF interface and (2) the NMF as well as the mirror tunnel device interface.

Mirroring traffic through a *passive* "mirror tunnel" device is an innovation that serves as an extension to our prior work [5] involving *active* devices. The use of a passive interface, i.e. a *tunnel* device, is distinct from traditional deployments, in which each active network device is provided a certain amount of computational resources to manage traffic for one or more NSFs. A passive device is a network instance that does not consume any computational resources, e.g. cores normally assigned to DPDK PMD drivers [18] or interrupt service routines normally allocated for kernel drivers. As described in Fig. 1, a passive device acts as a "tunnel" that allows SW TAPs to transmit packets. The packet transmission is executed via the NSF driver; therefore, the tunnel device requires no dedicated resources. Mirroring begins with the capture process, after which the packets are transmitted via the passive mirror tunnel interface. When mirror HWOL is enabled and configured, the transmit logic inside the mirror tunnel network interface triggers a HW copy of the packet stream to the pre-configured destination network interface allocated to the NMF. The final component of the design is an innovative run-time mechanism designed to enable and disable forwarding mirrored traffic to an external interface, e.g. Ethernet bus 2. In contrast, the design presented in [5] always forwards mirrored traffic.

As shown in Fig. 1, our design introduces three control elements:

(1) **Software TAP controller.** The controller is designed to manage enabling and disabling SW TAPs. The mechanism is created via IP management tools supported by the Linux kernel and the underlying user-space DPDK libraries [18]. Specifically, the controller leverages `ip` commands to update the link state setting for the mirror tunnel interface, i.e. the VF. The PF sends link state change event notifications to the appropriate VF. Our design adds a pre-registered link status change callback routine to the driver. Our design registers and unregisters the SW TAP depending upon the current setting for the VF link status.

(2) **Mirror offloading controller.** This feature controls mirror HWOL of the NIC and is accomplished by means of a Virtual Function daemon (VFd) [14]. VFd is a kernel sysfs extension designed to allow the PF to manage VF configuration. There are three mirror HWOL features supported in VFd: ingress, egress and VLAN based mirroring. Our design employs VLAN mirroring to enable traffic classification, for which the mirroring logic in the NIC inspects the tags of the traffic and forwards packets with matching VLANs to the pre-configured VF(s).

(3) **Outbound mirrored traffic controller.** As depicted in Fig. 1, HW mirroring is only triggered when we enable transmission from the mirror tunnel network interface. This control provides administrators the ability to loopback outbound traffic. In the work presented in [10], we discuss a VLAN based L2 switching algorithm to forward mirrored traffic to a remote site according to a coordinated flow forwarding policy. For cases where mirrored traffic is to remain in the local network, this control prevents outbound traffic by enabling transmit loopback via VFd.

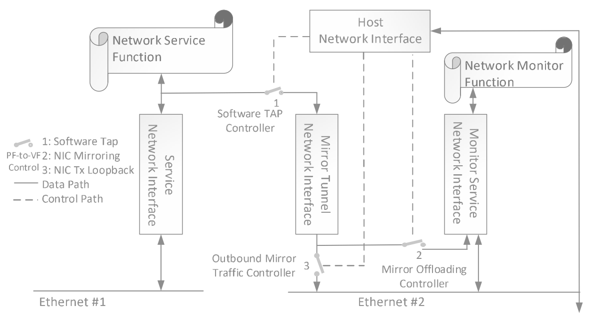

Figure 1 Traffic Mirroring via Mirror Tunnel Devices

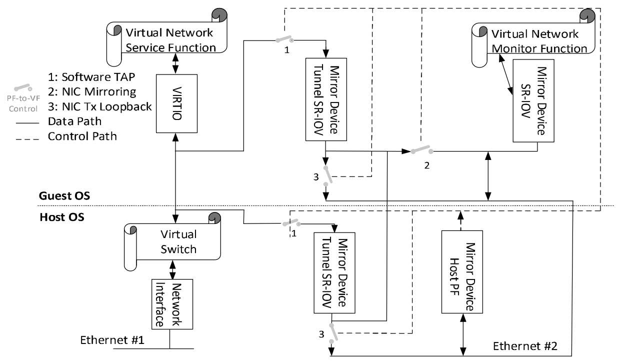

Figure 2 Logical Network Interface Mirror Tunneling

Figure 2 presents a mirror tunneling use case over a virtual switch (vSwitch) system where a Virtual Network Function (VNF) is implemented via logical network devices. For this use case, both the NSF and the NMF are deployed in a VM. The mirror tunnel devices are collocated with the NSF, which is necessary since the SW TAP leverages a source packet pointer buffer that in turn removes the need for copy operations, i.e. requirement 1. We note that inter-VM packet transfer without a mirror tunnel requires the use of expensive L3 overlays. In a vSwitch system, the virtual NSF manages traffic through a para-virtualization based logical interface, i.e. VIRTIO [11]; the backbone of many vSwitch designs [15]. For a VIRTIO interface there are two possible TAP points for extracting virtual NSF traffic. Specifically, the SW TAP can either be applied to the front-end interface, i.e. virtio-net, or the backend driver, i.e. vhost net. Figure 2 presents both options. Based on our prior work [5], mirror HWOL is implemented through a backend driver to take advantage of the centralized mirroring configuration interface provided by the vSwitch controller. The de-centralized approach of implementing a SW TAP through a front-end driver has the advantage of not requiring modifications to the vSwitch implementation.

IV. PERFORMANCE BENCHMARK

In this section we present the performance data collected for our mirror tunnel design when realizing the topology presented in Fig. 3, i.e. VHOST mirroring via vSwitch. For this case, which involves the use of VNFs with logical network interfaces, we modify OvS [9] to support our mirror tunnel design. Furthermore, we leverage the DPDK `l2fwd` application as our VNF, and compare the baseline results with enabling mirroring on the backend and frontend drivers. For completeness, we also collect results using the unmodified, built-in OvS mirroring mechanism. For VIRTIO mirroring, we continue to leverage `l2fwd`. In this case, we extend the application to accept mirror configuration settings as input parameters. As mentioned in the previous section, VFd serves as an aid for our management interface.

Figure 3 presents the test setup for VHOST mirroring. In this case, we use the Maximum Receiver Rate (MRR) test results for a unidirectional flow to measure the overhead under worst-case scenario conditions. Specifically, since each VIRTIO port by default trains to 10 Gbps full duplex, we wish to sufficiently stress the device under test. Note that VIRTIO interface vnic3 is used for default mirroring while interfaces VF2, i.e. the mirror tunnel interface, and VF3, i.e. the mirror network interface, are used for our design. The topology for VIRTIO mirroring is identical with the exception that VF0 is attached to the vProbe VM.

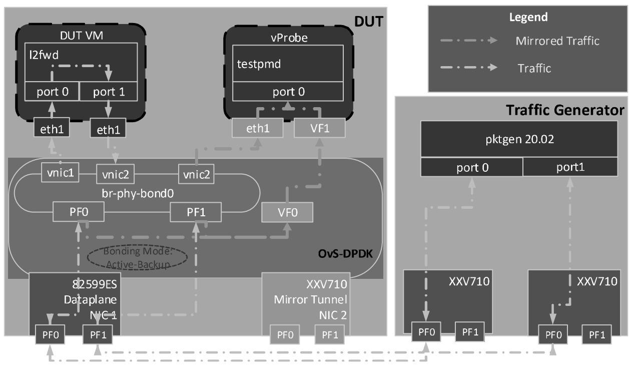

Figure 3 Test Network Topology for VHOST Mirroring

Note that the measured overhead for mirroring includes two data sets: (1) VIRTIO and (2) VHOST mirroring. We observe that traffic mirroring through the backend (BE) driver produces twice as much overhead as the frontend (FE) driver. We plan to investigate this issue further with the HW performance monitors supported by x86 processors. Based upon the results shown in Fig. 4 we observe that although throughput significantly degrades for the default OvS port mirroring configuration, performance markedly improves and is ostensibly restored by means of our design.

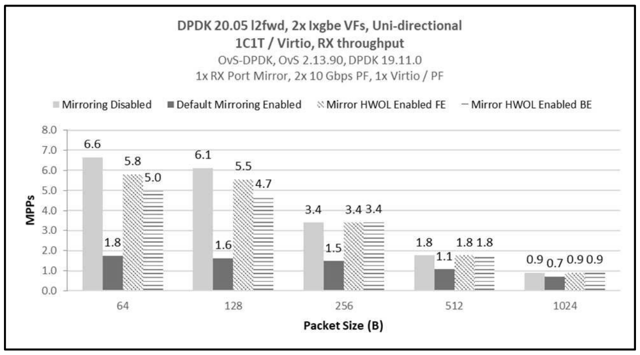

Figure 4 VHOST (OvS) and VIRTIO Port Mirroring

V. Conclusion

In this paper, we demonstrate a novel approach to the design of a SW-based monitoring system that reimagines a NIC with mirror HWOL capabilities as a portable, scalable SW TAP, thus enabling high throughput mirroring on virtually *any* system. This design takes advantage of HWOL features to capture traffic in high throughput environments with minimal impact to the source NSF. Our mirroring service is run-time reconfigurable and can be deployed either on the host or guest domains. We present a mirroring service management system that can be deployed in several environments since it is based upon common Linux networking tools available in most kernels, i.e. `ip`. We showcase the ability of our mirror tunnel design to be deployed over both physical and logical devices. Currently, we are actively engaged with various communities and exploring upstreaming our contributions to the corresponding mainstream Open Source projects.

References

[1] D. Kreutz, et al., Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE, vol. 103, no. 1, Jan 2015

[2] SNMP, Simple Network Management Protocol, RFC 1157

[3] RMON, Remote Monitoring, RFC 1757

[4] Cisco Systems, Introduction to Cisco IOS NetFlow – A Technical Overview, Whitepaper, May 2012.

[5] L.M. Wang, T. Miskell, P. Fu, C. Liang and E. Verplanke, Port mirroring via NIC offloading, IEEE NOMS 2020

[6] Cisco Switch Design, https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html

[7] A. Cecil, A Summary of Network Traffic Monitoring and Analysis Techniques,https://www.cse.wustl.edu/~jain/cse567-06/net_monitoring.htm

[8] C. So-In, A Survey of Network Traffic Monitoring and Analysis Tools, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.4989&rep=rep1&type=pdf

[9] T. Miskell, L.M. Wang, E. Finn and M. Mehan, Port Mirroring Offload, Open vSwitch and OVN 2020 Fall Conference

[10] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak and G. Carle, Performance Implications of Pakcet Filtering with Linux eBPF, 2018 30th Int. Teletraffic Congress ITC30, Vienna, Austria.

[11] R. Russell, “virtio: towards a de-facto standard for virtual i/o devices.” ACM SIGOPS Operating Systems Review, July 2008.

[12] Intel Ethernet Controller X710/XXV710/XL710 Datasheet

[13] README in i40e driver, https://sourceforge.net/projects/e1000/files/i40e%20stable/2.11.25/

[14] L.M. Wang, A. Zelezniak, E. S. Daniels, T. Miskell and L.D. Chen, Build an SR-IOV Hypervisor, IEEE IFIP 2019

[15] P. Emmerich, d. Raumer, F. Wohlfart and G. Carle, Peformance Characteristics of Virtual Switching, 2014 IEEE Int. Conf. on Cloud Netowrking

[16] S. Jeong, J. You and J. W.K Hong, Design and Implementation of virtual TAP for SDN-based OpenStack Networking, IEEE IFIP 2019.

[17] TAP as a Service, https://docs.openstack.org/dragonflow/latest/specs/tap_as_a_service.html.

[18] DPDK, http://www.dpdk.org

[19] Cisco, Implementing Data Center Overlay Protocols, https://www.ciscopress.com/articles/article.asp?p=2999385&seqNu

[20] E. S. Daniels, Reflections on Mirroring with DPDK, DPDK Summit 2017, https://www.slideshare.net/LF_DPDK/lfdpdk17reflections-on-mirroring-with-dpdk

[21] Y.Y. Yang, W.H. Chen, C.T. Yang, S.T. Chen, and F.C. Jiang, Implementing of a Real-Time Network Traffic Monitoring Service with Network Functions Virtualization, 2015 Int. Conf. on Cloud Comp. and Big Data.

[22] PCI-SIG, SR-IOV Specification, https://pcisig.com/sites/default/files/specification_documents/ECN_SR-IOV_Table_Updates_16-June-2016.pdf