Demo/Poster Abstract: Efficient and Flexible Packet Tracing for Virtualized Networks using eBPF

Kun Suo[†], Yong Zhao[†], Wei Chen[‡] and Jia Rao[†]
[†]Department of Computer Science and Engineering, the University of Texas at Arlington
[‡]Department of Computer Science, the University of Colorado, Colorado Springs
Email: kun.suo@mavs.uta.edu, yong.zhao@mavs.uta.edu, cwei@uccs.edu, jia.rao@uta.edu

***Abstract*—As the scale of cloud systems continues to grow, virtualized networks are becoming increasingly important to the performance and reliability of the cloud. Despite many advantages, virtualized networks introduce additional layers of abstraction and are more difficult to monitor and diagnose performance issues compared to traditional networks. Furthermore, it is challenging to reason about the dynamic performance of virtualized networks. Therefore, there is a great need for fine-grained, user customizable, and reconfigurable network tracing. To address the above challenges, we propose *vNetTracer*, an efficient and programmable packet profiler for virtualized networks. vNetTracer relies on the extended Berkeley Packet Filter (eBPF) to dynamically attach user-defined tracing scripts into a live virtualized network without any changes to user programs nor restarting the monitored network. Through case studies, we demonstrate the effectiveness of vNetTracer in diagnosing various virtualized networking problems.**

I. Introduction

To adapt the rapid change of network infrastructure and support increasing applications in the cloud, many cloud providers have adopted virtualized network into their infrastructures for highly efficient and dynamic network services. However, how to efficiently monitor and improve the Quality-of-Service (QoS) of virtualized network is becoming a challenging and inevitable issue. Although many studies have proposed solutions to monitor and diagnose the network performance, there lacks an efficient and flexible tool to trace the network activities in a virtualized environment. Existing state-of-the-art works [2], [3], [4] either are based on the system logs or introduce significant modifications to the network stack, which not only incur additional overhead but also cannot be flexibly programmed for kaleidoscopic demands. Besides, many tools can only be used within certain domains, such as a guest OS, and their ability of tracing are severely limited in virtualized environments with many hardware and software boundaries.

In this paper, we propose a network tracing system named *vNetTracer*, which enables flexible and programmable network performance monitoring for virtualized networks in an efficient manner. First, vNetTracer can trace across boundaries in the distributed virtualized network infrastructure and incurs negligible runtime overhead in the highly consolidated and optimized systems. Second, based on the eBPF scripts, vNetTracer supports customized network packet tracing and can be programmed as various performance metrics for different requirements. Last, as software defined networks and services are usually changing rapidly and dynamically, vNetTracer can reconfigure tracepoints, tracing rules or actions at runtime for the diagnosis of bottlenecks or performance problems in virtualized systems. vNetTracer does not require modifications to applications and is transparent to the users. We adopted vNetTracer in real systems and it helped us monitor network performance, identify bottlenecks and even locate system bugs.

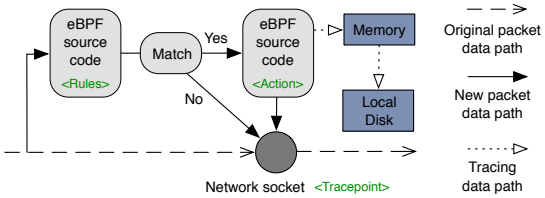

Fig. 1: How eBPF script works for packet filtering and tracing.

II. How vNetTracer works

Distributed tracing with low overhead In order to trace with high precision across the boundaries, vNetTracer adopts IP address and port number in the packet header to distinguish the network applications, and identifies individual packet by adding an unique packet ID. For the TCP packets, we used a 4-byte space in the options of the TCP header. For UDP packets, we used `__skb_put()` to allocate a 4-byte additional space to the original packets at the sender side. We generated a 32-bit random number as the packet ID in the space when the packet was copied from the user space to the kernel space. The UDP packet ID was then removed from the packet payload through `pskb_trim_rcsum()` before it was copied to the application buffer in the receiver side to guarantee the application transparency. At each tracepoint, vNetTracer attaches and executes Extended Berkeley Packet Filter (eBPF) scripts to collect data, which incurs negligible impact to the virtualized networks.

Performance metrics Besides the unique packet ID, vNetTracer also records the packet number, packet length and current system time for network measurement when a packet goes through the tracepoints. Based on these raw tracing data, we calculate the network performance metrics, including the throughput, latency, jitter and packet loss, for the applications. In addition, vNetTracer also provides advanced tracing data inside the virtualized networks. For instance, vNetTracer can decompose the abnormal network latency, attribute it into different layers of the network stack and locate the bottlenecks.

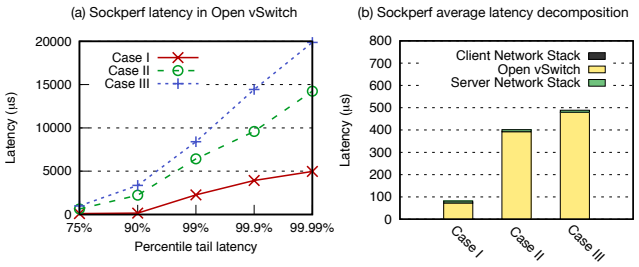

Fig. 2: Sockperf latency and its decomposition in the OVS.

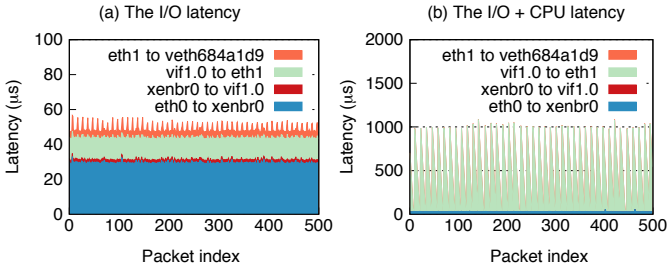

Fig. 3: Latency decomposition when the Sockperf VM runs alone or shares the physical CPU with a CPU-bound VM.

Flexibility Due to the dynamic network performance and capricious user requirements, vNetTracer attaches highly configured eBPF scripts inside the virtualized networks and collects tracing data based on user-defined rules. Figure 1 illustrates the workflow of tracing on a network device. When a packet goes across a network interface, the original process is just to pass it to the next layer or network device. However, when an tracing script is attached to the interface, the program will be first executed and checks whether the packet matches the user defined rules. If it is matched, the user defined actions are executed in the tracing script. Then the tracing raw data is copied into the local memory associated with this tracing script. If the above actions are finished or the rules are not matched, normal packet processing will proceed. The above processes execute at runtime and the tracing rules or actions can be reconfigured flexibly without rebooting the systems.

III. Preliminary Results

We ran representative network benchmarks in various virtualized environments with vNetTracer and made the following important observations:

Network delay inside Open vSwitch We created three VMs configured with four vCPUs and 4GB memory on a single physical server equipped with a dual ten-core Intel Xeon E5-2640 2.6GHz processor and 64GB memory. The hypervisor we used was KVM 2.6.1 and all the VMs were connected through Open vSwitch (OVS). We executed the Sockperf and iperf clients on VM0, another iperf client on VM1, and the Sockperf server as well as two iperf servers on VM2. As a comparison baseline, we only run the Sockperf application to measure the latency in an uncongested network, which was denoted as Case I in Figure 2(a). Next, we run the iperf client with Sockperf client simultaneously on the same VM and recorded the Sockperf latency as Case II. Finally, we added the second iperf client on another VM based on Case II and denoted such scenario as Case III. As illustrated in Figure 2(a), the tail latency of Sockperf in Case II and Case III increased significantly compared to the latency in the uncongested network. Similar problem can also be observed in the physical switches. To analyze the bottlenecks and locate the positions in the virtualized network path, we executed vNetTracer on the VMs and host machine, and bound the tracing scripts at application sockets and OVS ports. As the latency decomposition shown in Figure 2(b), the time spent inside OVS dominated the total transmission time. As more applications occupied the network path, the network became increasingly congested and the time in OVS, such as queueing delay and processing delay, increased significantly.

Tail latency in the hypervisor We created two VMs configured with one vCPU and 4GB memory on a single physical server. The hypervisor was Xen 4.8.1 and the scheduler was set as credit2. All the applications were running within containers on the VMs. First, we executed the Sockperf server side on one VM and sent requests to measure the latency as the baseline. Next, we executed a `while(1)` loop on another VM and pinned the vCPU of the two VMs on the same physical CPU core. The latency of Sockperf increased significantly when the I/O-bound VM shared the CPU resources with the CPU-bound VM. To analyze this problem, we used vNetTracer to attach eBPF script at the network ports shown in Figure 3(a) to trace the network. As shown in Figure 3(a), when the I/O-bound VM run alone, the client to server side transmission delay dominated the one way latency. In comparison, when the I/O-bound VM shared the same CPU core with the CPU-bound VM, the time spent between the backend `vif1.0` in Dom0 and frontend `eth1` in the server VM took more than 90% of the network latency, and this indicated the scheduling delay inside of Xen. We reported our findings to the Xen open source community and the above issues were confirmed by engineers from Citrix [1].

Inspired by these observations with vNetTracer, we proposed corresponding optimizations, such as limiting the network flow rate at the OVS ingress and tuning the credit2 scheduler in Xen, to solve the above issues. Our preliminary results show that the optimizations derived from our analysis with vNetTracer are effective in improving the application performance in the virtualized networks.

Acknowledgement

This work was supported in part by U.S. NSF grants CNS-1649502 and IIS-1633753.

References

[1] *Long tail latency caused by rate-limit in Xen credit2*. https://lists.xenproject.org/archives/html/xen-devel/2017-06/msg01410.html.

[2] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. In *Proceedings of USENIX OSDI*, 2014.

[3] K. Suo, J. Rao, L. Cheng, and F. C. M. Lau. Time capsule: Tracing packet latency across different layers in virtualized systems. In *Proceedings of the 7th ACM APSys*, 2016.

[4] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm. lprof: A non-intrusive request flow profiler for distributed systems. In *Proceedings of USENIX OSDI*, 2014.