An Analytics Approach to Traffic Analysis in Network Virtualization

Hui Zhang, Junghwan Rhee, Nipun Arora, Qiang Xu, Cristian Lumezanu, and Guofei Jiang
NEC Laboratories America, Princeton, New Jersey 08540
Email: {huizhang,rhee,nipun,qiangxu,lume,gfj}@nec-labs.com

***Abstract*—Network virtualization has been propounded as a diversifying attribute of the future inter-networking paradigm. However, monitoring and troubleshooting operational virtual networks can be a daunting task, due to their size, distributed state, and additional complexity introduced by network virtualization.**

We propose an analytics approach for the analysis of network traces collected across hypervisors and switches. To re-organize individual trace events into path-wise slices that represent the life-cycle of individual packets, we first present a trace slicing scheme. Then, we develop a path characterization scheme to extract feature matrices from those trace slices. Using those feature metrics, we develop a set of trace analysis algorithms to cluster, rank, query, and verify packet traces. We have developed the analytics approach in a SDN network management tool, and presented evaluation results to show how it can enable visibility and effective problem diagnosis in a SDN network.

***Keywords*—Network Virtualization, Software-Defined Networks, Network Management, Traffic Analysis**

I. Introduction

Network virtualization in multi-tenant data centers [10] provides the illusion of multiple, independent virtual networks on the same underlying physical hardware. The physical devices are responsible for forwarding packets while the software-based virtual networks provide a logical abstraction that allows tenants to deploy and manage diverse applications independently. Virtualizing the network enables the deployment of complex configurations and policies, that might not work together otherwise. Additionally, it improves network efficiency, and reduces operational costs [10].

Network virtualization comes at the expense of increased management complexity. For instance, Figure 1 depicts the networking architecture of a physical host in OpenStack [3] open-source infrastructure-as-a-service (IaaS) software. As shown in the figure, the physical host has 3 virtual machine (VM) guests running on it, which are configured with virtual network 1 (where `vm01` and `vm02` are connected to), and virtual network 2 (`vm02`, and `vm03`). For an Ethernet frame to travel from `eth0` of `vm01`, to the physical network, it must pass through nine devices inside the host: a TAP device for `vnet0` (implements the virtual network interface card), Linux bridges (which behave as hubs, and forward all received packets to all ports), `veth` pairs `qvbXXX` to `qvoXXX` , and `int-br-eth1` to `phy-br-eth1` (which are like virtual patch cables), an Open vSwitch [1] (OVS, configurable similar to any other physical switch), and finally the physical network interface card. The complexity involved in the transition of the packet over the "virtual interfaces" is quite significant and makes debugging and troubleshooting very difficult.

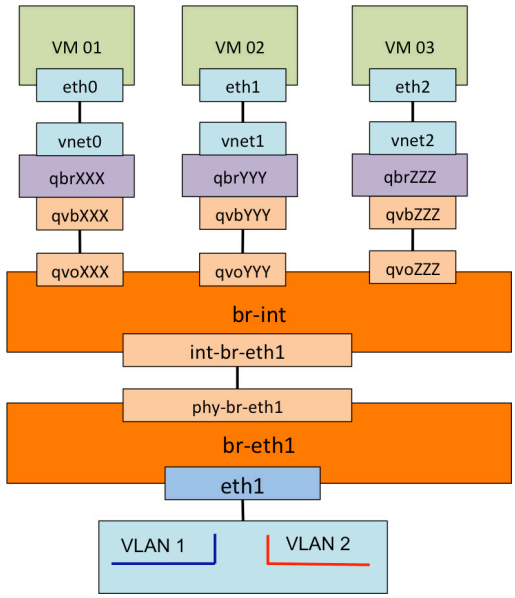

Fig. 1: OpenStack network architecture [3] on a physical host.

Based on our experiences in data center network management we have observed three main challenges in virtual networks.

Complex configuration. Network virtualization is enabled through a variety of mechanisms and devices, such as software switches deployed on hypervisors (*e.g.*, OVS), management protocols for programmable switches (*e.g.*, OpenFlow [2]), isolation and tunneling protocols for legacy networks (*e.g.*, VLAN, GRE). This generally requires non-trivial amount of network configuration efforts, such as tagging VLANs on legacy switches and adding tunnels in Open vSwitches. As the amount of work increases, so does the chance for misconfiguration. Thus, it is important to *identify the packet behavior as it traverses the network (in the data plane) rather than (or in addition to) examining what configuration operators have been applied (in the control plane).*

Miscellaneous network devices. Virtualization mechanisms often require the initialization of a large number of virtual network devices, each performing a specific function on traffic. Aggregating views from different network locations is essential in troubleshooting network performance degradations and failures. Existing solutions such as NetFlow offer only "interface-wise" visibility such as TCP flows traversing through the attached switch interface, the number of packets for a given flow. However, such visibility is not sufficient to reveal the life-cycle of an single packet. Simply combining distributed traces is insufficient because it requires domain knowledge to

account packet events from different traces to the same packet and recreate the packet's life-cycle. Thus, in order to fully conquer the complexities of virtual network monitoring, we need *a systematic approach to account for packets across all physical and virtual network devices*.

Assorted traffic groups. The operational granularity of networks is often global rather than local – a network update is not limited to an update of a single forwarding table entry, but a sequential updates of forwarding table entries along a routing path that certain packets have taken. For example, to resolve traffic congestion, it is effective to update certain source-to-destination routes or certain route segments. Besides, in a virtualized network, there is typically a large number of traffic groups to support: for instance, a tenant's traffic, a VM's traffic, one application's traffic, traffic of a priority class, etc., which indicates that a given traffic group may be likely buried in a large number of groups, and difficult to be examined. To achieve such operations above, we expect that *analytics-friendliness,* e.g.*, the ability to allow clustering, ranking, querying packets and routes*, is a desired feature.

In this paper we propose an approach for the summarization and analysis of virtual network traces collected across hypervisors and switches. It provides a new structure to network traces beyond 5-tuple IP flows, and offers both aggregate information and selective analysis for understanding the network's behaviors and performance along the time. The key contributions of the paper are:

- We propose a network trace modeling mechanism that includes a trace slicing scheme to re-organize individual trace events into route-wise slices which represent the life-cycle of individual packets. The mechanism also includes a path characterization scheme to extract feature matrices from these trace slices.
- We develop a set of packet analysis algorithms to cluster, rank, query, and verify packet traces based on feature metrics from trace slices.
- We developed the analytics approach in a NEC internal SDN network management tool, and presented evaluation results to show how it can enable problem diagnosis in a hybrid SDN network.

Ultimately, our goal is to help speed up diagnosis of data-path routing problems by leveraging new network analytics techniques for path-wise packet analysis.

II. Related Work

There are several existing network monitoring and diagnosis tools, which have not solved the discussed challenges and provide the desired feature of FlowView.

There are centralized software solutions that monitor network devices using NetFlow, SNMP or by simply collecting packet traces, and then aggregating statistical data. The tools based on NetFlow and SNMP cannot analyze traffic at the granularity of packets. Protocol analyzers, such as Wireshark [12], capture and view packets on specific interfaces. Network data visualization has a long history [4], [9], [14]; *e.g.*, NetViewer [9] focuses on large-scale network diagnosis tasks such as DDoS detection, and Visty [14] helps troubleshooting cross-layer network perform problems within a particular

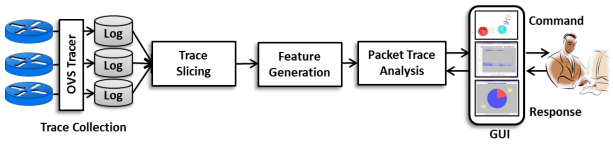

Fig. 2: Overview of FlowView.

machine. Most of those tools focus on IP flow dynamics at specific interfaces or hosts, and FlowView is complementary to them by providing packet analytics *across the network*.

In SDN trouble-shooting, VeriFlow [8] analyzes the configuration files pushed to network devices to infer forwarding paths and determine inconsistency. However, even if the configuration is correct, packets may not follow it due to bugs in switch software, conflicting rules or limited memory space to enforce all rules [6]. *ndb* [5] is a network debugger for software-defined networks that traces packet paths by emitting "postcards" from every switch that the traced packet traverses. The network controller collects all postcards and reconstructs the packet paths. While it is able to collect and correlate network wide information, *ndb* suffers from the increased overhead of logging information in the control plane. OFRewind [15] records guest network traffic by mirroring those packets on traversed switches and then replays them to identify operational problems. FlowView is complementary to them by helping operators to speed up diagnosing data-path routing problems with automated packet analytics *across the network*.

III. Design of FlowView

In this section, we present the design of FlowView. Figure 2 shows how FlowView processes packet information. It has four main components.

- **Trace collection**: This step collects the network packet traces from the target network. Various open-source packet capture tools (e.g., `libpcap`) or proprietary packet dump tools can be used for collecting traces from open vSwitches and physical network elements.
- **Trace slicing**: Each single trace presents an isolated view in each network infrastructure element. This step transforms such isolated packet events into event sequences, which groups all packet events that a packet invoked throughout the network. We call this information *a trace slice*.
- **Feature generation:** FlowView uses feature vectors in multiple levels of a network path, a link, and a packet to represent the characteristics of trace slices which are analyzed in the next step.
- **Packet trace analysis**: In this step, FlowView provides versatile data analytic techniques onto the trace slices such as clustering, ranking, query, and verification based on the feature matrix for network management such as performance profiling or troubleshooting.

A. Trace Slicing

Network traces collected from each router/switch show an isolated view on per-hop behavior. For troubleshooting network problems it is necessary to have a holistic view on the lifetime of a packet, which is a series of events collected from routers/switches that the packet goes through during its routing. We call the generation of this view, *trace slicing*,

Algorithm 1 Trace Slicing

H : Network element set, P : Packet trace set indexed by an network element, S : A network event, FS : A flow slice, FSS : A set of FS, T_D : A network diameter

```
1: function PREPROCESSING(P, H)
2:     M = ∅
3:     for h in H do
4:         for e in P[h] do
5:             [T_e, SW_e, Port_e, IO_e, H_e, P_e] = e; S_e = GenSignature(H_e, P_e)
6:             M.append([T_e, SW_e, Port_e, IO_e, H_e, S_e])
7:     Sort M by T_e in the ascending order
8:     return M
9: function FLOWSLICING(M)
10:    FSS = ∅; FoundFS = ∅
11:    for e in M do
12:        FSExist = false
13:        for FS in FSS do
14:            if S_e == FS.sig then
15:                e' = FS's last event
16:                if T_e' >= (T_e − T_D) then
17:                    FS.append(e); FSExist = true
18:                    continue
19:        if FSExist == false then
20:            FS = {e}; FS.Sig = S_e; FFS.append(FS)
21:    return FSS
```

1) Pre-processing the collected packet events: The packet events include diverse data fields. Before applying the slicing algorithm, the packet events need to be structured to ease the slicing procedure as shown in the `Preprocessing` function of Algorithm 1. For each packet event e, a packet signature S_e is created based on the header H_e and payload P_e. Like the idea in IP traceback [13], this signature is the invariant content in the packet that does not change through the routing process. In the header H_e, the IP flow information [source IP address, source port, destination IP address, destination port] is a part of the invariant content (for IP tunneling, the inner IP headers flow information is invariant). In the payload P_e, the whole content or a hash value of the content can be used as a part of the signature. The signature S_e is the combination of the invariant content created from both the header H_e and payload P_e.

After this step, packet events along with the newly generated signatures are sorted in an ascending order of the event time stamp T_e, and they are stored in an ordered list.

2) Slicing packet events: We define a unit of sliced packet events called a *flow slice (FS)* as a time-ordered event sequence data structure which includes all recorded packet events that a network packet invoked when it traverses the network. The `FlowSlicing` function in Algorithm 1 shows this mechanism. For each event in the ordered preprocessed event set, repeat the following steps; Search in the set FSS for any existing FS having the same signature as e's signature. If no FS found, create a new FS, insert e as the first event of this FS and label its signature as S_e, and continue; Otherwise, if for every FS found, its last packet events time stamp is earlier than e's time stamp by more than the threshold T_D, (the maximum time a packet can remain in the network and it is a function of the network diameter), then create a new FS, insert e as the first event of this FS and label its signature as S_e, and continue; Otherwise, for each found FS whose last packet events time stamp is not earlier than e's time stamp by more than T_D, append e as the last event of this FS. After a loop is finished, return FSS, a derived set of flow slices.

B. Trace Feature Generation

Once packet flow slices are generated we extract feature metrics which characterize the slices and assist the analysis tasks such as clustering, ranking, and query of packet traces. FlowView provides three kinds of features:

Path features: Each packet may have a diverse route. In order to precisely analyze each packet's behavior and represent its route, we defined features for the paths that the packets go through. One example is a feature vector for switch ids and ports which is defined as the set of switch ids and ports in a packet flow slice. It is a two dimensional matrix whose row represents a switch id and whose column represents a port. As another example, a feature vector for the number of path links in a slice can be easily determined using the count of the pairs of switch ids and ports.

Link features: Link features characterize the packet's per-link behavior. For instance, the feature for the packet delay in each link would be a set of link delays.

Packet features: Packet features capture the properties specific to the packet which are common in the perspective of a link or a path. The size, the protocol, the VLAN tag, the source or destination IP addresses would be the examples of packet features.

C. Trace Analysis

Based on the extracted features, FlowView provides four types of packet trace analyses: clustering, ranking, query, and verification. These four functions provide versatile packet trace analytics functions.

1) Clustering of trace slices: This procedure constructs the clusters of trace slices based on the similarity of features. One major usage of clustering is based on the feature vector for switch ids and ports which reflect the similarity of the end-to-end paths that the packets traverse in the network. This clustering enables us to determine the groups of different paths that the packets go through and perform behavior-based analysis of packets even without knowing the overall topology of the network. For the clustering method, we use connectivity based clustering [7] with a threshold in the distance function. This scheme uses an agglomerative method (bottom-up approach), and the single-linkage is used to connect clusters.

2) Ranking of trace slices: This procedure provides ranking of trace slices so that users can understand the significance of slices in a given ranking scheme of interest. Depending on users' interest various feature vectors can be used for ranking. For example, ranking based on the link delay will give us the network paths that are affected by delays.

3) Query of trace slices: In case that users have a specific dimension of analysis in mind and would like to find packets or paths for a certain condition as input, query function achieves such a goal. Given a set of query conditions, FlowView queries the set of slices, and reports the set that matches the conditions. For instance, if a user wants to understand whether there is any packet of a certain protocol or packet size, such information can be easily retrieved by FlowView.

4) Verification of trace slices: By building the paths that packets take in the data plane, users can have a global view and check a large diversity of conditions concerning network behavior which include end-to-end reachability, loop-freeness, routing behavior consistency. FlowView supports such verifications with invariant models [8]. For instance, FlowView can

automatically report the end-to-end reachability by verifying if the slice of a packet includes the OVSes in the two end hosts that serve the ingress and egress switches for that packet.

IV. Implementation and Evaluation

The current FlowView implementation collects network traces from OVS switches. Trace slicing, feature generation, and the internal engine of trace analysis are written in C++. In addition, we have a graphical user interface for trace analysis implemented in Java.

A. *Case Study: A hybrid SDN network*

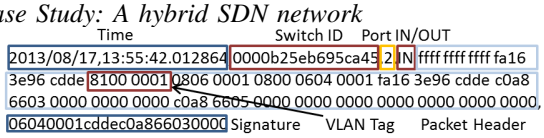

Fig. 3: An example of a structured packet event.

Trace information: The input trace data consist of a set of OVS traces in our hybrid SDN network [11]. This testbed is composed of 4 OVS switches, 2 NEC OpenFlow switches, and 1 legacy Juniper switch. Among such network devices, FlowView is deployed in the 4 OVS switches that act as the ingress and egress switches for VMs. Traces are periodically generated from this infrastructure for the purpose of accounting and troubleshooting. In particular, `ping` command was used to generate end-to-end traffic between VMs. Figure 3 shows an example of one FlowView packet event. A packet event e is defined as a 5-tuple object of time, switch ID, switch port, IN/OUT, packet header, and the signature.

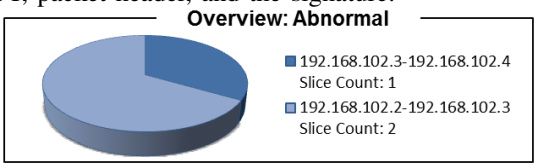

Fig. 4: The FlowView report on an abnormal network.

Abnormal network: A network anomaly occurred that the `ping` traffic in some nodes were not successful. We start troubleshooting by examining the FlowView summary report shown in Figure 4. A noticeable symptom was that the number of packet events were significantly more than the detected packet traces. We looked into those ARP packets and found they were mostly for the sending VM looking up the receiving VM's MAC address. Taking a close look at the ARP headers, we found that they all carried a VLAN tag. This hint led to the in-network VLAN configuration inspection, and turned out those ARP packets were dropped at the first physical switch which needs a VLAN trunk mode setup. Once the misconfiguration was fixed, the network restored the reachability for the VMs.

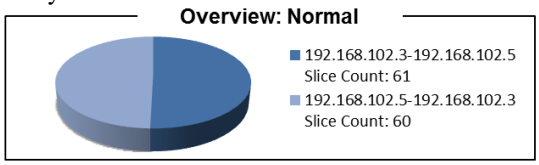

Fig. 5: The FlowView report on a normal network.

Normal network: A part of the FlowView summary report on the normal network is shown in Figure 5. It includes the reachability verification on the `ping` traffic between two VMs. Figure 6 shows the details of the path that one of packets in the trace has traversed.

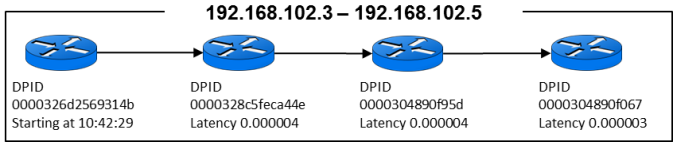

Fig. 6: The snapshot of a FlowView slice example.

V. Conclusion

FlowView is a tool for SDN analysis. Individual trace events from hypervisors and switches are re-organized into trace slices that represent the life-time of packets. FlowView offers versatile analytics features such as clustering, ranking, querying, verification as well as a graphical interface. We present our proof-of-concept and its concrete usage case that detects and troubleshoots the root cause of a network problem.

References

[1] Open vSwitch: An Open Virtual Switch. http://openvswitch.org/.

[2] OpenFlow specification in Open Networking Foundation. https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[3] OpenStack: Open Source Cloud Computing Software. https://www.openstack.org/.

[4] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Trans. Visualization and Computer Graphics*, 1(1):16–28, 1995.

[5] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the debugger for my software-defined network? In *Proc. ACM HotSDN*, 2012.

[6] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, and P. Kazemian. Leveraging sdn layering to systematically troubleshoot networks. In *Proc. ACM HotSDN*, 2013.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[8] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proc. USENIX NSDI*, 2013.

[9] S. S. Kim and A. L. N. Reddy. Netviewer: A network traffic visualization and analysis tool. In *Proc. USENIX LISA*, 2005.

[10] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *Proc. USENIX NSDI*, 2014.

[11] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang. Hybnet: Network manager for a hybrid network infrastructure. In *Proc. ACM/IFIP/USENIX Middleware*, 2013.

[12] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[13] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based ip traceback. In *Proc. ACM SIGCOMM*, 2001.

[14] K. Wongsuphasawat, P. Artornsombudh, B. Nguyen, and J. McCann. Network stack diagnosis and visualization tool. In *Proc. ACM CHiMiT*, 2009.

[15] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann. Ofrewind: Enabling record and replay troubleshooting for networks. In *Proc. USENIX ATC*, 2011.