Virtual Network Diagnosis as a Service

Wenfei Wu[1], Guohui Wang[2], Aditya Akella[1], and Anees Shaikh[3]

[1]Univerisity of Wisconsin-Madison , [2]Facebook , [3]Google

1 Introduction

Today's cloud platforms allow multiple tenants to share a physical network upon which tenants may specify sophisticated virtual networks including virtual machines (VMs), virtual links and other network appliances, such as routers or load balancers. Virtual networks are realized by allocating and logically isolating resources in the physical infrastructure. VMs run atop hypervisors and connect to in-hypervisor virtual switches (e.g., Open vSwitch), virtual links can be implemented by tunneling protocols (e.g. GRE, VXLAN), and network services are provided by routing tenant traffic through software middleboxes in VMs. Examples that support such functionality include OpenStack Neutron, VMware/Nicira's NVP and IBM SDN-VE.

In this complex system, various network problems (e.g., due to misconfiguration, failures, bugs, etc.) occur in different layers. However, virtualization abstracts the underlying details, which prevents cloud tenants from having the needed visibility to perform troubleshooting and deploying existing solutions such as OFRewind, NDB, etc.. More specifically, tenants only have access to their own virtual resources, and crucially, each virtual resource may map to multiple physical resources, i.e., a virtual link may map to multiple physical links; tools inside the VMs (e.g. ping, traceroute) are usually point tools lacking the view of the whole virtual network, which causes inconvenience to the tenant. When a problem arises, there is no way today to systematically obtain the relevant data from the appropriate locations and expose them to the tenant in a meaningful way to facilitate diagnosis.

Diagnosing virtual networks in a large scale cloud environment introduces several concomitant technical challenges. First, the diagnosis approach should preserve the tenant's abstract view of the network, without leaking information about the infrastructure or another tenant's virtual networks. Second, as the cloud infrastructure is shared among tenants, the virtual network diagnostic mechanisms must limit their impact on switching performance and other tenant or application flows. Third, in a large-scale cloud, a large number of tenants may request diagnosis services simultaneously. Data collection and analysis should avoid imposing significant bottlenecks on troubleshooting and prevalent network traffic. Lastly, due to tunneling/encapsulation or packet transformation in middleboxes, identifying and correlating flows for different tenants becomes another challenge for the diagnostic service provider.

In this paper, we make the case for a virtual network diagnosis framework (VND) that enables a cloud provider to offer sophisticated diagnostic services to its tenants. Extracting the relevant data, exposing it to the tenant, and providing analytics interfaces, form the key capabilities of VND.

2 Design

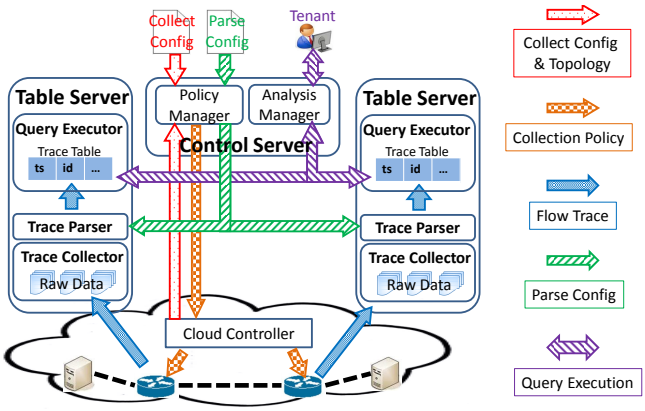

Figure 1: Virtual Network Diagnosis Framework

1) Appliance ***Node*** : ***lb***	Trace ID ***all***
2) Capture ***input***	Filter: ***ip.proto = tcp***
3) ***srcIP=10.0.0.6/32***	***or ip.proto = udp***
4) ***dstIP=10.0.0.8/32***	Fields: ***timestamp*** as ***ts***,
5) ***proto=TCP***	***ip.src*** as ***src_ip***,
6) ***srcPort=****	***ip.dst*** as ***dst_ip***,
7) ***dstPort=80***	***ip.proto*** as ***proto***,
8) Capture ***output***	***tcp.src*** as ***src_port***,
9) ...	***tcp.dst*** as ***dst_port***,
10) Appliance ...	***udp.src*** as ***src_port***,
11) ...	***udp.dst*** as ***dst_port***
(a) Collection	(b) Parse

Figure 2: A Diagnostic Configuration Example

VND is composed of a **control server** and multiple **table servers** (Figure 1). Tenants interact with the control server to perform diagnosis. When a tenant encounters problems in its virtual network, it can submit a **trace collection configuration** (Figure 2(a)) that specifies the flow of interest, e.g., related to a set of endpoints, or application types. The pattern may be specified at different granularity, such as a particular TCP flow or all traffic to/from a particular (virtual) IP

address. The tenant's traffic is transmitted by tunneling protocol, so the entire packet including all layers of the header and the payload can be collected.

The **policy manager** accepts the trace collection configuration, and obtains network topology and the tenant's logical-to-physical mapping information. This is assumed to be available at the SDN controller, e.g., similar to a network information base (not shown in the figure). The policy manager then computes a collection policy that represents how flow traces should be captured in the physical network. The policy includes the flow pattern, the capture points and the location of **trace collectors** in the physical network. The policy also has the routing rules to dump the captured flows into the collector. VND places the capture point and its table server locally with the problematic virtual appliance on the same hypervisor so as to reduce overhead to the network. Based on the policy, the cloud controller sets up corresponding trace collection rules on the capture points (e.g., matching and mirroring traffic based on a flow identifier in OpenFlow), starts the collectors in virtual machines and configures routing rules between capture points and collectors.

Trace collectors reside in the table servers to create local network taps to collect trace data. Cloud tenants also submit a **parse configuration** in Figure 2(b) to perform initial parsing on the raw flow trace. It has multiple parsing rules, with each rule having filter and field lists that specify the packets of interest and the header field values to extract, as well as the table columns to store the values. **Trace parsers** on table servers accept the parse configuration, and they parse the raw traffic traces into multiple text tables, called trace tables, which contain the packet records with selected header fields.

The trace tables are stored in the **query executors**. A query executor can itself be viewed as a database with its own tables; it can perform operations such as search, join, etc. on trace tables. Query executors across the table servers form a distributed database which supports inter-table operations. Tenants operate on their trace tables via an SQL interface from the **analysis manager** in the control server.

Various network diagnosis and monitoring operations can be developed using this SQL interface on trace tables. For example, statistics on a certain field such as IP or MAC can be computed by counting packets on that field; throughput can be calculated by aggregating packet payload size by timestamps; per-hop delay can be obtained by comparing packet timestamps in and out of that hop. Flows or packets in and out of a virtual appliance (e.g., middleboxes, tunnels) can be correlated by comparing their unique fingerprint in the header or the payload or their sequence of timestamps. Figure 3 is an example of RTT monitoring.

```
# T: <ts, id, srcIP, dstIP, srcPort, dstPort, seq, ack, payload_length>
1) create view F as select * from T where srcIP=IP1 and dstIP=IP2
2) create view B as select * from T where dstIP=IP1 and srcIP=IP2
3) create view RTT as select F.ts as t1, B.ts as t2 from F, B
        where F.seq + F.payload_length = B.ack
4) select avg(t2-t1) from RTT
```

Figure 3: RTT Monitoring

3 Implementation and Evaluation

We prototyped VND on a small layer-2 cluster with 3 HP T5500 workstations and 1 HP Procurve switch. Each workstation has 2 quad-core CPUs, a 10 Gbps NIC and 12 GB memory. Each physical host runs the KVM hypervisor and Open vSwitch to simulate the cloud environment. The trace collector and trace parser are implemented in python using the pcap and dpkt package. We use MySQL Cluster to achieve the functions of the query executor and the analysis manager.

We measured the overhead introduced by trace collection and data query. According to our measurement, the virtual switch OVS can process up to 18 Gbps network traffic; given that the NIC bandwidth is usually 10 Gbps, this leaves extra processing capability – up to 8Gbps – on OVS to perform trace replication. We measured the memory throughput for different trace capture rates, and concluded that each 1 Gbps of network traffic mirroring costs an extra 59 MB/s memory throughput. When a tenant performs a data query, the storage overhead is negligible because only a few fields of the packet header are extracted, and the network overhead is also negligible because the query command is send to distributed query executors and only the result is returned. Most of the network diagnosis/monitoring, such as throughput or RTT calculation, can be executed in real time.

We also evaluate VND's scalability through simulation. We simulate a data center with 10 K physical machines, using empirically observed data center workloads, and assume each server has the same processing capability as in our measurement described above. The results show that VND can satisfy the diagnostic requirement based on existing data center workload.

4 Conclusion

In this paper, we described the challenges involved in diagnosing problems in virtual cloud networks. We propose the VND framework to address these challenges by allowing cloud providers to offer sophisticated virtual network diagnosis services to their tenants. Our evaluation shows that by colocating flow capture points and table servers, VND can capture tenant traffic flows without impacting their performance, and the network diagnosis query can be executed quickly on distributed tables in response to tenant requests without introducing too much extra network traffic. This architecture scales to the size of a real data center network. To the best of our knowledge, ours is the first attempt at addressing the problem of virtual network diagnosis services in clouds, and we believe VND is a feasible and useful solution.