Intel Cloud Orchestration Networking

Matthew Johnson, Cody Malick, and Garrett Smith

March 24, 2017

Table of Contents

Team Cloud Orchestra

Our team is implementing a software defined network based on Open vSwitch for a cloud orchestration technology called Ciao.

Our client is Rob Nesius, Engineering Manager at Intel Corporation.

Scope

What is a cloud?

Figure 1: The Cloud - Randall Munroe [1]

"A huge, amorphous network of servers somewhere"

Scope

The network connecting the nodes must be controlled. In our case, this is achieved by the Cloud Integrated Advanced Orchestrator, or Ciao.

Ciao provides an easy to deploy, secure, scalable cloud orchestration system which handles virtual machines, containers, and bare metal apps agnostically as generic workloads [2].

Scope

The servers, or nodes, in these clouds need to talk to each other, hence the need for a network solution.

Ciao uses a software defined network to allow different compute nodes to communicate with each other.

Software defined networking is a software abstraction of physical networking solutions, allowing for “a more scalable and centralized network control architecture” [3].

Project Goals

Goal One

Switch Ciao Generic Routing Encapsulation (GRE) tunnel implementation to use Open vSwitch-created GRE tunnels.

Generic Routing Encapsulation (GRE) GRE encapsulates an arbitrary network layer protocol so it can be sent over arbitrary network layer protocol [4].

Open vSwitch (OVS) Open vSwitch is a "multilayer virtual switch. . . designed to enable massive network automation through programmatic extension" [5].

Project Goals

Goal Two

Switch the tunneling protocol implementation to best-performing network virtualization technology based on our performance measurement of each on data center network cards.

Project Goals

Goal Three (optional)

Replace the Linux bridge implementation with OVS switch instances.

Project Design

Open vSwitch uses a database to manage configuration while running. The configuration can be updated on the fly by accessing its management protocol using the Open vSwitch Database Management Protocol, defined in RFC 7047[6]

OVS Database Management Protocol

One of the main goals of the project is to implement OVS built network bridges programmatically. The protocol uses JSON to build its commands.

Listing 1: Example insert OVS DB Management Protocol

```
{
  "op": "insert",
  "table": "Bridge",
  "row": 2,
  "uuid-name":
     00000000-0000-0000-0000-000000000000
}
```

Libovsdb

Libovsdb is an open source library that provides a Go programming language wrapper around the OVS Database Management Protocol. This tool is quite helpful because it allows us easy access to the Management Protocol, without having to manually create JSON. Here is an example:[7]

Listing 2: Example insert operation using libovsdb

```
// simple insert operation
insertOp := libovsdb.Operation{
    Op:   "insert",
    Table:    "Bridge",
    Row:      bridge,
    UUIDName: namedUUID,
}
```

Replacing GRE

Generic Routing Encapsulation is one of the original solutions for point-to-point virtual network tunneling. It was developed by Cisco, but has one major issue: scaling.

nvGRE and VxLAN

Two alternative tunneling protocols to replace GRE:

- nvGRE: Network virtualization standard created in tandem by HP, Dell, and Intel
- VxLAN: Network virtualization standard created in tandem by Cisco, VMware, Citrix, and Redhat

Overall performance and overhead are similar on paper, will require testing to see which is the best fit for our implementation

Network Performance I

We need to test the performance of our SDN implementation to make a decision between the VxLAN and nvGRE protocols. We also need to compare how the Ciao SDN performs before and after we make modifications. The two network performance metrics we are testing are bandwidth and latency.

Bandwidth Testing I

We are using iperf3 for bandwidth testing [8]. It is a well documented, easy to use tool for measuring bandwidth. It must be set up to run in client/server mode. There are premade Docker containers we can use to run iperf3 [9].

Listing 3: Running an iperf3 Docker container in server mode

```
docker run -it --rm --name=iperf3-server -p
    5201:5201 networkstatic/iperf3 -s
```

Listing 4: Running an iperf3 Docker container in client mode

```
docker run -it --rm networkstatic/iperf3 -c
    172.17.0.2
```

Bandwidth Testing II

Listing 5: iperf3 output

```
Connecting to host 172.17.0.2, port 5201
[  4] local 172.17.0.3 port 38464 connected to 172.17.0.2 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  1.38 GBytes  11.8 Gbits/sec    0    720
     KBytes
[  4]   1.00-2.00   sec  1.37 GBytes  11.8 Gbits/sec    1    720
     KBytes
[  4]   2.00-3.00   sec  1.35 GBytes  11.6 Gbits/sec    1    819
     KBytes
[  4]   3.00-4.00   sec  1.23 GBytes  10.5 Gbits/sec    0    824
     KBytes
[  4]   4.00-5.00   sec  1.27 GBytes  11.0 Gbits/sec   92    592
     KBytes
[  4]   5.00-6.00   sec  1.37 GBytes  11.8 Gbits/sec    0    592
     KBytes
[  4]   6.00-7.00   sec  1.21 GBytes  10.4 Gbits/sec    0   1.77
     MBytes
```

Bandwidth Testing III

```
[  4]   7.00-8.00   sec  1.37 GBytes  11.7 Gbits/sec    0   1.77
    MBytes
[  4]   8.00-9.00   sec  1.36 GBytes  11.6 Gbits/sec    0   1.77
    MBytes
[  4]   9.00-10.00  sec  1.37 GBytes  11.8 Gbits/sec    1   1.77
    MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  13.3 GBytes  11.4 Gbits/sec   95
                sender
[  4]   0.00-10.00  sec  13.3 GBytes  11.4 Gbits/sec
                receiver

iperf Done.
```

Latency Testing I

We are using qperf for latency testing [10]. It is a well documented, easy to use tool for measuring latency. Much like iperf3 it must be set up to run in client/server mode. Again, much like iperf3 there are premade Docker containers we can use to run qperf [11].

Listing 6: Running a qperf docker container in server mode

```
docker run -dti -p 4000:4000 -p 4001:4001
    arjanschaaf/centos-qperf -lp 4000
```

Listing 7: Running a qperf docker container in client mode to measure TCP and UDP latency

```
docker run -ti --rm arjanschaaf/centos-qperf
    172.17.0.2 -lp 4000 -ip 4001 tcp_lat udp_lat
```

Latency Testing II

Listing 8: qperf output

```
tcp_lat:
    latency =  91 us
udp_lat:
    latency =  81.7 us
```

Stumbling Blocks I

The one major problem we had during the fall quarter was getting the Intel NUCs connected to the campus network. We required network access to install Clear Linux on the NUCs, to access them remotely via SSH, and to perform actions that require internet access such as accessing the Ciao git repository hosted on GitHub. Intel wants us to store the NUCs in a secure location. Kevin McGrath let us use his lab which has restricted access.

We set the NUCs up in Kevin's lab on campus, and had to work with Todd Shecter, the head of the OSU EECS IT department to get permission to connect them to the campus network, and register them so they would be able to connect to the campus network. This took three weeks between technical issues and communication delays because most of our communication was done by email.

Stumbling Blocks II

- We first emailed Todd with an explanation of our project and requested network access. He requested a meeting to better understand what we were doing.
- After explaining our project to Todd in person, he gave us permission to connect the NUCs to the campus network, and said he would help.
- To connect the NUCs to the network Todd required the MAC address of each device.
- None of the NUCs had an OS on them, and the MAC address was not listed on the case, or in the BIOS. We had to boot each NUC from a USB flash drive to get the MAC address.
- We provided Todd with the MAC addresses of each NUC so they could be connected to the campus network.
- We waited for Todd to register the NUCS on the campus network.

Stumbling Blocks III

- After Todd registered the NUCs they were not being assigned an IP address by the campus network.
- As a workaround so we could install Clear Linux we bridged the network connection from Garrett's laptop to the NUCs.
- We emailed Todd explaining the IP address issue, and he had us go through several troubleshooting steps, none of which worked.
- Todd had us connect one of the NUCs to a different switch in the lab which resolved the issue.
- We were going to move the NUCs over to the other switch, but it took a couple of days before we had time to do this, and during that time the campus network started assigning IP addresses to the NUCs.
- Once the NUCs had network access the problem was resolved.

Winter Term

The next few slides cover the progress we made by the end of winter term.

Docker Certificate Issues

Clear Linux was managing Docker certificates incorrectly which halted our work until the next release of Clear Linux.

Problems with Clear Linux Certificate management

Clear Linux manually manages the trust store, which sometimes causes keystone certificates to be overwritten.

Problems with getfqdn()

On Clear Linux the `socket.getfqdn()` function, which is supposed to return the fully qualified domain name was only returning the host name. We reported this the Clear Linux Developers.

```
garrett@fw-dear205-ciao-nuc4:~$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more in
>>> import socket
>>> socket.getfqdn()
'fw-dear205-ciao-nuc4.engr.oregonstate.edu'
>>>
```

Migrating to Ubuntu 16

We migrated to Ubuntu because of the issues with Docker certificates and `getfqdn()` on Clear Linux. Most of the Ciao team develops on Ubuntu.

Network Performance Measurement Scripts

- Garrett wrote scripts to parse the output from iperf and qperf into csv format.
- Scripts were written in Ruby because of its strong support for parsing strings, and good JSON library.

Problems Running Workloads

- After we set up Ciao, we ran into issues running workloads.
- When Ciao sets up a CNCI it uses DHCP to assign an IP address to it.
- OSU's network will not assign IP addresses to the CNCIs.
- To work around this we moved the Nucs off of OSU's network.
- We are running a DHCP server on the Cisco switch we are using for the LAN

Listing 9: CNCI error

```
{"error":{"code":500,"name":"Internal Server
    Error","message":"Unable to Launch Tenant CNCI"}}
```

Open vSwitch - libovsdb I

It was originally our plan to implement with libovsdb.

Listing 10: libovsdb insert operation

```
"op": "insert",
"table": "Port",
"row": <port information>,
"type": "gre",
"uuid-name": "uuid"
```

But! We actually need to create an OVS bridge for this to work.

Open vSwitch - OVS bridge first I

Listing 11: No Linux Bridge and OVS Tunnel

```
mrsj@singlevm:~\$ sudo brctl addbr br1
mrsj@singlevm:~\$ sudo brctl show
bridge name    bridge id               STP
    enabled    interfaces
br0            8000.000000000000       no
br1            8000.000000000000       no
docker0        8000.02427ef725d4       no
mrsj@singlevm:~\$ sudo ovs-vsctl add-port br1
    gre0 -- set interface gre0 type=gre
    options:remote_ip=192.215.10.0
ovs-vsctl: no bridge named br1
```

Open vSwitch

Original Goal: implement GRE tunnels with Open vSwitch by the end of week five

Scope Change: cannot implement GRE tunnels in OVS without first implementing bridges in OVS, originally a stretch goal.

New Goal: Implement bridges in OVS, then implement tunnels in OVS. Finally, implement VXLAN and nvGRE tunneling protocols as a stretch goal.

Progress

Internal Functions

Internal functions were written in Ciao to control OVS via the ovs-vsctl tool.

Progress

Ciao Integration

We have integrated the internal function calls into Ciao in the compute node (cn) module, the compute node concentrator (CNCI) module, and the controlling network module (network).

Progress

Testing

ciao-down (test utility for Ciao) successfully builds our integrated network module, but fails on the verify script.

Next Steps

Integrating new network mode into the Ciao `bridge` object. Will allow the internal network map to track new bridges and tunnels created by OVS, and update them appropriately.

References

- R. Munroe. (2011, jun) The cloud. [Online]. Available: http://xkcd.com/908/
- T. Pepper, S. Ortiz, M. Ryan *et al.* (2016, sep) Ciao readme. [Online]. Available: https://github.com/01org/ciao/blob/master/README.md
- P. Goransson and C. Black, *Software Defined Networks, A Comprehensive Approach*. Morgan Kaufmann, 2014.
- F. . T. Hanks, Li. (1994, oct) Generic routing encapsulation (gre). [Online]. Available: https://tools.ietf.org/html/rfc1701
- (2016, nov) Open vswitch. [Online]. Available: http://www.openvswitch.org/
- E. B. Pfaff, B. David. (2013, dec) The open vswitch database management protocol. [Online]. Available: