CS 461 - Fall 2016 - Client Requirements Document

Matthew Johnson, Garrett Smith, Cody Malick

Cloud Orchestra

Abstract

This document outlines the requirements for the Cloud Orchestration Networking project sponsored by Intel Corporation. It formally defines the purpose, scope, description, function, use, constraints, and specific requirements of the project. Although there are no specific design decisions made, it will be used as a building block for the rest of the design, implementation, and testing process.

CONTENTS

1 Introduction

1.1 Purpose

Ciao is Intel's Cloud Integrated Advanced Orchestrator (Ciao). Ciao schedules and balances workload throughout a large set of servers. The job of distributing workload over a large number of servers is a difficult task. Ciao does this well, but has room for improvement in the area of network security and speed. Our project is to add additional functionality to Ciao's network systems, and enable technology that will increase the speed that these systems operate at.

While developing its Software Defined Network implementation into Ciao, Intel has found a need for a more advanced form of network bridge than the standard Linux bridge. The initial implementation using Linux bridges and GRE tunnels has worked well, but as further development was done on Ciao, the need for modern packet encapsulation and other innovative protocols was found to be needed. Implementing a network mode in Ciao utilizing Open vSwitch-created GRE tunnels would allow the network to utilize advanced networking techniques to increase performance, such as modern packet encapsulation methods. This addition would be used by those implementing Ciao in their own organizations or businesses to further increase speed and availability of features in their cloud.

1.2 Scope

Ciao exists as a cloud orchestrator for cloud clusters. It is inherently necessary for the separate nodes in the cloud cluster to be able to talk to each other. Without a reliable and secure software defined network Ciao would have little purpose. Utilization of Open VSwitch GRE tunnels allows Ciao to become more scalable and enables the inclusion of packet-acceleration technology.

The software we will be developing will be an Open VSwitch networking mode within Ciao, encapsulating the following goals:

1.2.1 Replace Linux Bridges with OVS Bridges: The first goal of the project is to replace all Linux Bridge calls in Ciao with OVS generated tunnels. A full implementation is needed in order to generate tunnels using the OVS management database.

1.2.2 Open vSwitch GRE Tunnel: The second goal of the project is to switch the current GRE tunnel implementation with the Open vSwitch created GRE tunnel. This will allow for newer packet encapsulation techniques to be used, as well as provide the option to test packet acceleration.

1.2.3 Stretch Goal: Test and Implement Best Performing Tunnel Implementation: The stretch goal for the project is to switch the tunneling implementation. The two options are VxLAN and nvGRE, and will the final implementation will be based on performance measurements of each protocol on data center network cards.

1.3 Notes on Scope Changes

At the beginning of Winter term, and during the writing of all our requirements documentation, it was assumed by us and our clients that GRE tunnels could be created independently of Linux bridges and that the new OVS tunnels could be connected to Linux bridges. There was nothing through the research we had done up to this point that implied differently. This was also the desired result by our client due to the flexibility of Linux bridges. Linux bridges are more versatile than OVS bridges so it was their hope to leave the bridge implementation alone for the first part of the project, then later add an alternate implementation in OVS that could be used if desired.

The assumption that OVS bridges and tunnels could be created independently and separately led to attempts to use OVS to do exactly that. When we tried this simple implementation of just the Open vSwitch GRE tunnels without creating OVS bridges, we encountered many errors with network setup. With what we know now these errors were very predictable. The errors we encountered drove us to look at other examples of open source projects using libovsdb and we noticed that everyone else was creating OVS bridges before they created OVS tunnels. We were unable to find any examples of mixing networking feature types.

As a result of the above discoveries, we communicated our concerns to our clients and tried to test manual creation of OVS GRE tunnels and Linux bridges. This was easy to do with the command line tool brctl to create the bridge and the Open vSwitch command line interface ovs-vsctl. First we created a Linux bridge with brctl called br1. Then we tried to connect a GRE tunnel created with ovs-vsctl to the bridge br1, but it turned out that Open vSwitch could not even see Linux bridges on the system.

Listing 1. No Linux Bridge and OVS Tunnel

```
mrsj@singlevm:~# sudo brctl addbr br1
mrsj@singlevm:~# sudo brctl show
bridge name     bridge id               STP enabled     interfaces
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
```

We communicated our findings to our clients and they agreed that this shows it is necessary to first create the OVS bridge if we want to use the OVS created GRE tunnels. This means it is now necessary to implement the entire network framework with Open vSwitch, a considerable change in scope. This was originally considered a stretch goal for the project, but has now changed to the primary goal, pushing the goal one back to goal two and goal two back to the stretch goal position. Our future work will be to create the OVS framework and create OVS bridges (previously the stretch goal), then create OVS tunnels to connect them (previously goal one). Finally we hope to complete the final goal of creating VxLAN and nvGRE modules (previously goal two).

This problem was identified during the beginning of week six of Winter term.

1.4 Definitions, acronyms, and abbreviations

Bridge	Software or hardware that connects two or more network segments.
Cloud	A huge, amorphous network of servers somewhere [1].
Cloud Orchestration	An easy to deploy, secure, scalable cloud orchestration system which handles virtual machines, containers, and bare metal apps agnostically as generic workloads [2].
CNCI	Virtual Machines automatically configured by the ciao-controller, scheduled by the ciao-scheduler on a need basis, when tenant workloads are created [3].

Generic Routing Encapsulation (GRE)

	Encapsulation of an arbitrary network layer protocol so it can be sent over another arbitrary network layer protocol [4].
Linux Bridge	Configurable software bridge built into the Linux kernel[5].
Network Node (NN)	A Network Node is used to aggregate network traffic for all tenants while still keeping individual tenant traffic isolated from all other the tenants using special virtual machines called Compute Node Concentrators (or CNCIs) [3].
nvGRE	Network Virtualization using Generic Routing Encapsulation [6].
Open vSwitch	Open source multilayer software switch with support for distribution across multiple physical devices [7].
OVS	Open vSwitch [7].
Packet Acceleration	Increasing the speed of the processing and transfer of network packets.
Packet Encapsulation	Attaching the headers for a network protocol to a packet so it can be transmitted using that protocol [8].
SSNTP	The Simple and Secure Node Transfer Protocol (SSNTP) is a custom, fully asynchronous and TLS based application layer protocol. All Cloud Integrated Advanced Orchestrator (CIAO) components communicate with each others over SSNTP [9].
Tunnel	Point to point network connection that encapsulates traffic between points [8].
VxLAN	Virtual Extensible Local Area Network [10].

1.5 References

[1] R. Munroe. (2011, jun) The cloud. [Online]. Available: http://xkcd.com/908/

[2] T. Pepper, S. Ortiz, and M. Simental. (2016, sep) Ciao project. [Online]. Available: https://github.com/01org/ciao/blob/master/README.md

[3] M. Castelino. (2016, may) Ciao networking. [Online]. Available: https://github.com/01org/ciao/blob/master/networking/README.md

[4] F. . T. Hanks, Li. (1994, oct) Generic routing encapsulation (gre). [Online]. Available: https://tools.ietf.org/html/rfc1701

[5] T. L. Foundation. (2016, nov) Bridge. [Online]. Available: https://wiki.linuxfoundation.org/networking/bridge

[6] M. Sridharan, A. Greenberg, N. Venkataramiah, K. Dudam, I. Ganga, and G. Lin. (2015, sep) Rfc7637. [Online]. Available: https://tools.ietf.org/html/rfc7637

[7] (2016, nov) Open vswitch. [Online]. Available: http://www.openvswitch.org/

[8] J. Kurose and K. Ross, *Computer Networking*, 6th ed. Pearson, 2012.

[9] S. Ortiz, J. Andersen, and D. Lespiau. (2016, sep) Simple and secure node transfer protocol. [Online]. Available: https://github.com/01org/ciao/blob/master/ssntp/README.md

[10] M. Mahalingam. (2014, aug) Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. [Online]. Available: https://tools.ietf.org/html/rfc7348

[11] (2016, apr) Ciao network topology. [Online]. Available: https://github.com/01org/ciao/blob/master/networking/documentation/ciao-networking.png

1.6 Overview

The following section describes more details about the product, including product perspective, specific requirements, functionality requirements, and any assumptions or dependencies used. The section is organized in the following fashion:

1) Overall Description
2) Product Perspective

3) Product Functions
4) User Characteristics
5) Constraints
6) Reliability
7) Security
8) Specific Requirements

2 Overall Description

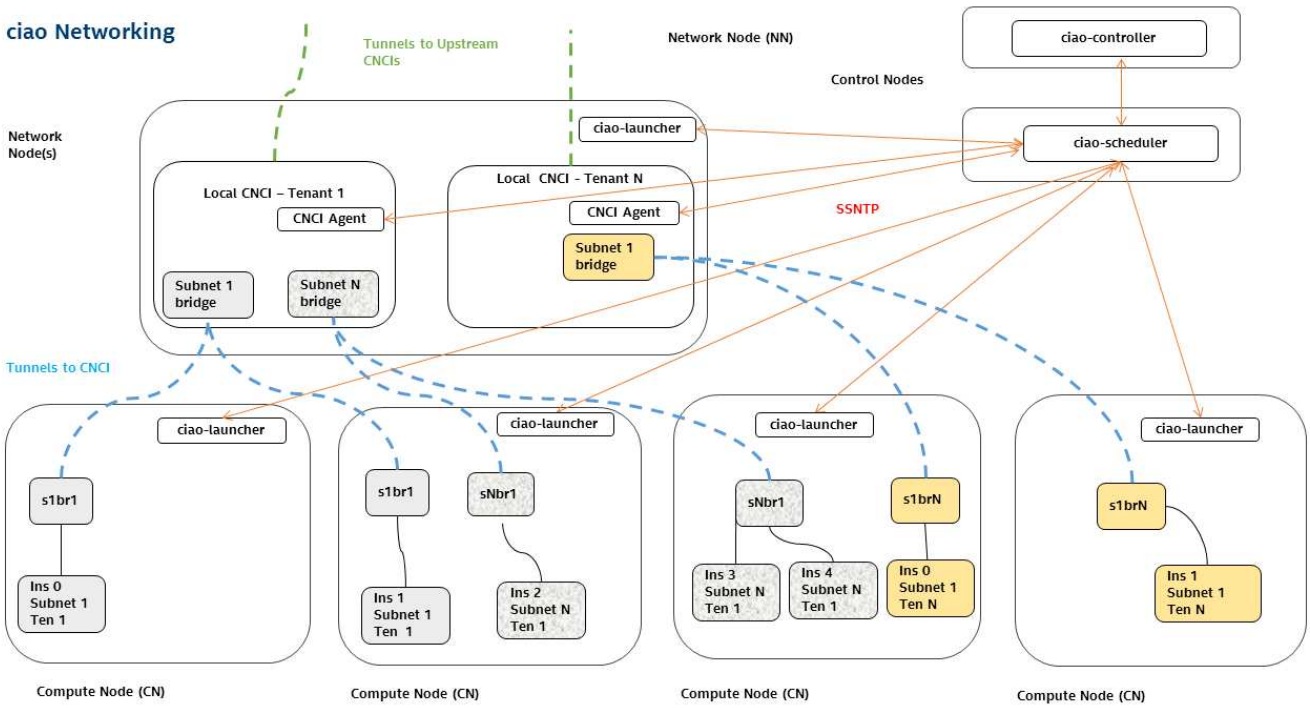
2.1 Product Perspective

The Software Defined Network (SDN) we implement will be utilized by Ciao to transfer packets between compute nodes and control nodes on a cloud cluster. For this purpose the mode must be fully integrated into the Ciao infrastructure and must behave similarly to what is already in place.

Because this software will be a component of a larger system, it must follow the design of that larger system and be fully integrated. As Ciao is implemented in the Go programming language, so this SDN implementation must be written in Go. The networking mode must route packets between ports on different nodes using networking protocols.

The SDN that is implemented must support the following structure of the cloud in Ciao:

Fig. 1. Ciao Network Topology [11]



Ciao has several functional components that must be worked with:

libsnnet Provides networking APIs to the ciao-launcher to create tenant specific network interfaces on compute nodes and CNCI specific network interfaces on a network node.

ciao-cnci-agent A SSNTP client which connects to the ciao-scheduler and runs within a CNCI VM and configures tenant network connectivity by interacting with the ciao-controller and ciao-launchers using the ciao-scheduler. The ciao-cnci-agent can also be run on physical nodes if desired.

docker-plugin Provides unified networking between VM and Docker workloads.

2.2 Product Functions

Ciao networking must support the following functionality.

1) Secure isolated overlay network - The networking implementation must provide each tenant with a secure isolated overlay network without the need for any configuration by the tenant and minimal configuration by the data center operator.
2) Auto discovery of compute/network nodes - Auto discover roles of nodes when they are attached to the network.
3) Diverse and scalable - Support large number of tenants with large or small number of workloads.
4) Work on different platforms - Operate on any Linux distribution by limiting the number of dependencies on user-space tools and leveraging Linux kernel interfaces whenever possible.
5) Migrate workloads - Provide the ability to migrate workloads from a Compute Node on demand or when a CN crashes without tenant intervention. Provide the ability to migrate CNCIs on demand or when a Network Node crashes.
6) Encrypt traffic - Provide the ability to transparently encrypt all tenant traffic even within the data center network.
7) Create GRE tunnels with Open VSwitch
8) Support for tenant and workload level security rules
9) Support for tenant and workload level NAT rules (inbound and outbound)

2.3 User characteristics

The users of this mode in Ciao will be educated and technically-minded data center administrators. They will be familiar with cloud technologies and networks, both software-defined and hardware-defined.

Due to Ciao's goal of minimum configuration, users are not required to have the knowledge base of a normal Openstack network administrator.

2.4 Constraints

2.4.1 High-order language requirements: As mentioned above, it is necessary to write this mode in the Go programming language in order to integrate with the rest of Ciao.

2.4.2 Reliability: The networking mode must be completely reliable and include the ability to migrate workloads when a compute or network node crashes with little-to-no interruption to the network capabilities for the tenants and must do so without tenant intervention.

2.4.3 Security: Traffic must be fully and transparently encrypted even within the data center network. It must utilize Simple and Secure Node Transfer Protocol (SSNTP) to ensure the security of traffic between the nodes.

2.4.4 Software Interfaces: Our networking mode must, as mentioned, be fully integrated with Ciao and able to facilitate communication between different compute and network nodes in a cloud cluster. Additionally, these connections must be secure and follow the SSNTP protocol.

The networking mode must also work in Ciao running on top of Clear Linux.

2.4.5 Hardware Interfaces: The network and control nodes will all be Intel NUCs and will be connected by ethernet ports and a switch. Other than the basic hardware requirements, there will be no other hardware interfaces, as this is a software defined solution.

2.5 Assumptions and Dependencies

A main assumption is that the Oregon State University will not interfere with our networking implementation. As we had issues with the OSU network in the beginning of this project, this is a valid concern.
Dependencies include Ciao and the physical hardware (5 Intel NUCs and a network switch) required to run the cluster. Additional dependencies are a Linux operating system, the Go programming language, Open VSwitch, and basic networking OS libraries.

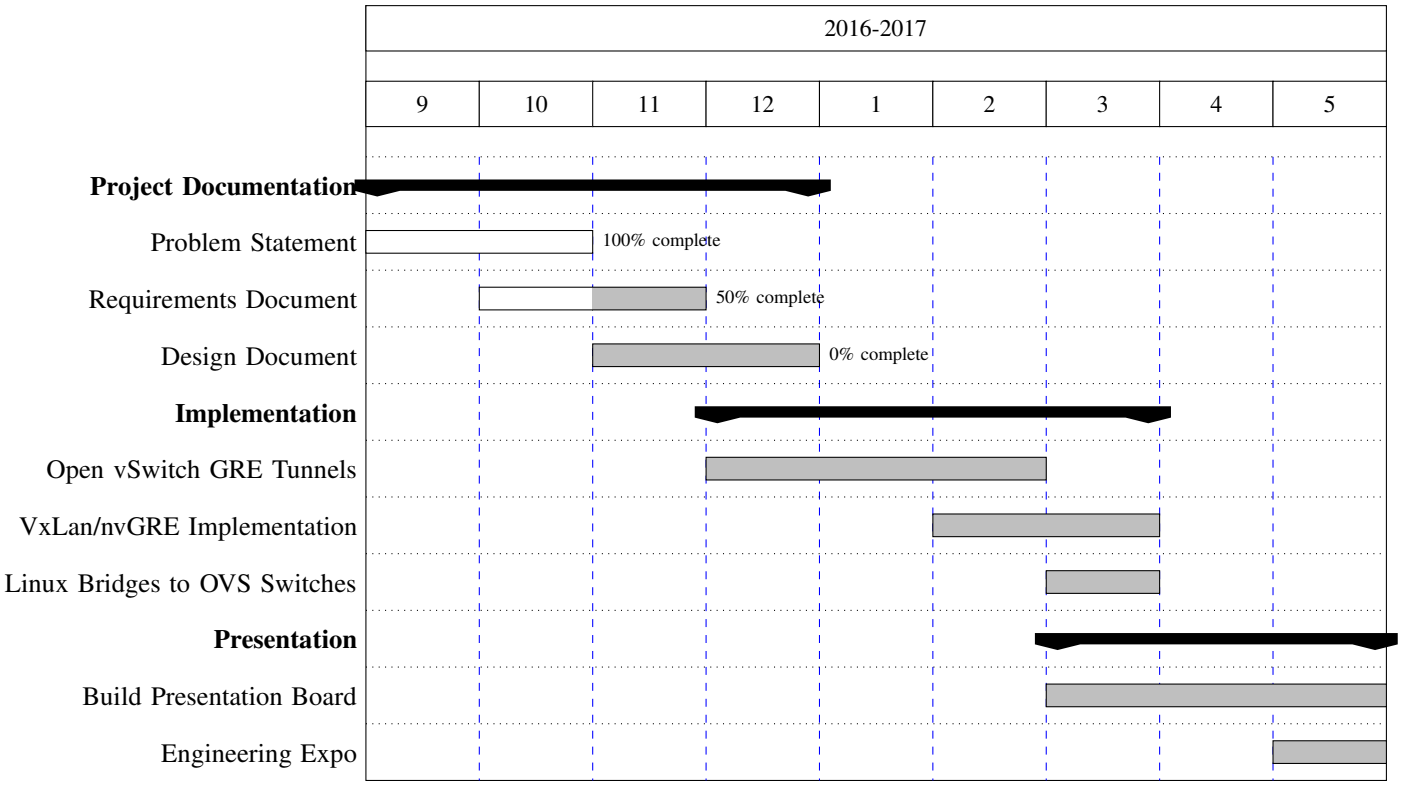
3 Specific Requirements

The main requirement is that Open VSwitch is used to create the GRE tunnels in the SDN implementation. A further goal is to switch the tunneling implementation to VxLAN or nvGRE based on performance measurements of each. The result of those performance metrics will dictate which is used.
A stretch goal is to replace the currently-implemented Linux bridges with Open VSwitch switch instances.
The specific requirements for this project are as follows:

1) Documentation
   a) Problem statement
   b) Requirements document
   c) Design document
2) Implementation
   a) Replace Linux Bridges with OVS Bridges Implement OVS Bridges into Ciao, and replace all Linux Bridge functionality.
   b) Open vSwitch GRE tunnels - Open VSwitch must be used to create the GRE tunnels in the SDN implementation. Currently, Ciao uses Linux-created GRE tunnels.
   c) Optional: VxLAN/nvGRE implementation - The second part is to switch the tunneling implementation to a VxLAN or nvGRE implementation based on speed performance. The faster implementation will be kept
3) Presentation
   a) Build presentation board
   b) Present at Engineering Expo

4 Gantt Chart

Following is the distribution of work scheduled for the duration of the project:


2016-2017
9
10
11
12
1
2
3
4
5
Project Documentation
Problem Statement
100% complete
Requirements Document
50% complete
Design Document
0% complete
Implementation
Open vSwitch GRE Tunnels
VxLan/nvGRE Implementation
Linux Bridges to OVS Switches
Presentation
Build Presentation Board
Engineering Expo