Intel Cloud Orchestration Networking

Spring Final Report

Matthew Johnson, Cody Malick, and Garrett Smith

Team 51, Cloud Orchestra

Abstract

This document contains our final report senior design. In includes an introduction to our project, the team, the client, its goals and purpose, and motivation. Included are the original requirements document, design document, and technology review. Also listed are short descriptions of our scope change, and updated gantt chart. Following are our full blog posts over the year, engineering expo poster, and project documentation. Last is our individual sections on what we learned.

CONTENTS

I. Introduction

Software Defined Network (SDN) implementations in practice today focus primarily on extremely large scale deployments where there are tens of thousands of data center servers. They use a topology that fully connects all servers with all servers, resulting in an extremely large and unwieldy mesh of tunnels. Beyond the mesh complexity itself, Address Resolution Protocol (ARP) table management, endpoint discovery, broadcast loop prevention and broadcast traffic management are also challenging in this complex topology.

In contrast, Ciao tightly integrates SDN to achieve a simpler overall implementation leveraging a limited local awareness of just enough of the global clouds state. Tenant overlay networks are used to overcome the above listed challenges in typical SDNs by using a distributed, stateless, self-configuring network topology running over dedicated network software appliances. This design yields a hierarchical SDN overlay without loops and meshes using Linux bridges interconnected by Linux native GRE tunnels. This has been shown to scale extremely well in an environment which consists of a few hundred nodes across a few server racks, which also happens to be the sweet spot of scale when it comes to most small and medium enterprises running private clouds today.

A. Purpose

The current implementation of Ciao tightly integrates software defined networking principles to leverage a limited local awareness of just enough of the global cloud's state. Tenant overlay networks are used to overcome traditional hardware networking challenges by using a distributed, stateless, self-configuring network topology running over dedicated network software appliances. This design is achieved using Linux-native Global Routing Encapsulation (GRE) tunnels and Linux bridges, and scales well in an environment of a few hundred nodes.

While this initial network implementation in Ciao satisfies current simple networking needs in Ciao, all innovation around software defined networks has shifted to the Open vSwitch (OVS) framework. Moving Ciao to OVS will allow leverage of packet acceleration frameworks like the Data Plane Development Kit (DPDK) as well as provide support for multiple tunneling protocols such as VxLAN and nvGRE. VxLAN and nvGRE are equal cost multipath routing (ECMP) friendly, which could increase network performance overall.

B. Goals

Our project is to first switch the Linux-created GRE tunnel implementation in Ciao to use GRE tunnels created by Open vSwitch. From that point we will switch the actual tunneling implementation from GRE to VxLAN/nvGRE based on performance measurements of each on data center networking cards. After this is completed, a stretch goal is to replace Linux bridges with Open vSwitch switch instances.

These goals changed somewhat by the middle of the Winter term. The primary goal now is to replace the Linux bridges with Open vSwitch switch instances because of an assumption that was found to be incorrect. It was assumed that we could create tunnel endpoints with Open vSwitch without using Open vSwitch bridges but Open vSwitch could not create tunnel endpoints with the Linux bridges Ciao uses. A full integration of Open vSwitch was required to use Open vSwitch created tunnels. Initially, we had planned on using a third party API, `libovsdb` to interface with the Open vSwitch management database [1]. While providing the necessary functionality, it added undocumented overhead. Specifically, all bridges and tunnels generated by Ciao had to be known about in the calling library. After extensive research and discussion with our client, we aimed to fully implement Open vSwitch into Ciao, rather than use it to exclusively create tunnels.

This scope change pushed the goal to switch the tunneling implementation to VxLAN/nvGRE based on performance measurements to stretch goal status. All scope change details were approved by our client.

C. Importance

As Ciao continues to grow as a project, the need for an improved networking interface will grow. Providing Open vSwitch implementation will, for the foreseeable future, keep Ciao on the edge of new innovations in software defined networking.

D. Team

Our team comprises three members: Garrett Smith, Matthew Johnson, and Cody Malick. All seniors in computer science, the team was formed prior to the start of fall term in 2016. Looking for challenging technical projects, the group approached Intel for project sponsorship.

E. Client

The project sponsor was the Open Source Technologies Group at Intel. Our primary staff sponsor was Rob Nesius, an Engineer Manager with the OTC.

1) Client Role: Intel took a very hands off approach to managing the project. While providing technical knowledge and assistance when needed, they very much left the team to manage itself. Manohar Castelino, a Principle Engineer at Intel and primary author of Ciao, provided a wealth of technical knowledge when requested.

2) Roles: While our team often worked on individual components together, often paired programming, each member had niche roles by the end of the year.

Garrett specialized in networking technologies and analytics. Throughout the year, he did a deep dive into networking technologies in order to write testing scripts for initial benchmarking. Using this knowledge, he provided great support to the team through understanding of SDN requirements throughout development.

Matthew's role in our group was as a Ciao domain specialist. With his understanding of the code base and constant forward momentum, we were able to plow through problems encountered throughout the year. He also acted as our primary communication point with the client.

Cody acted as the team's language evangelist. Having previous experience with the Go programming language, he helped get the team up and running quickly. Also digging into Open vSwitch, the primary implementation focus, was able to provide support through his understanding of needed interfaces.

II. Original Requirements Document

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 Open vSwitch GRE Tunnel: The first 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.

1.2.2 Test and Implement Best Performing Tunnel Implementation: Switch the tunneling implementation to VxLAN/nvGRE based on performance measurements of VxLAN and nvGRE on data center network cards.

1.2.3 Stretch Goal, Replace Linux Bridge: The final objective and stretch goal of the project is to replace the Linux bridges with Open vSwitch instances.

1.3 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].

1.4 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

1.5 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

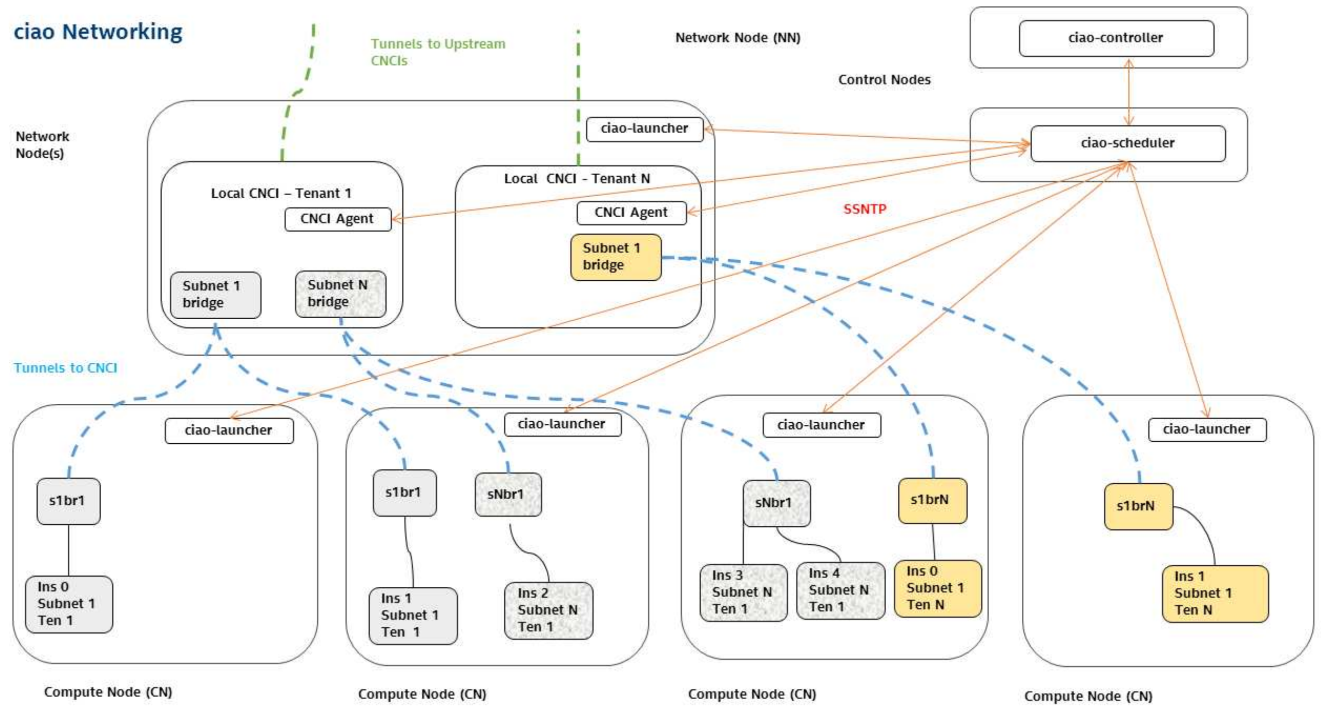
The Software Defined Network (SDN) we implement will be utilized by Ciao to transfer packets between compute nodes and control nodes on a cloud cluster. For this purpose the mode must be fully integrated into the Ciao infrastructure and must behave similarly to what is already in place.

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.

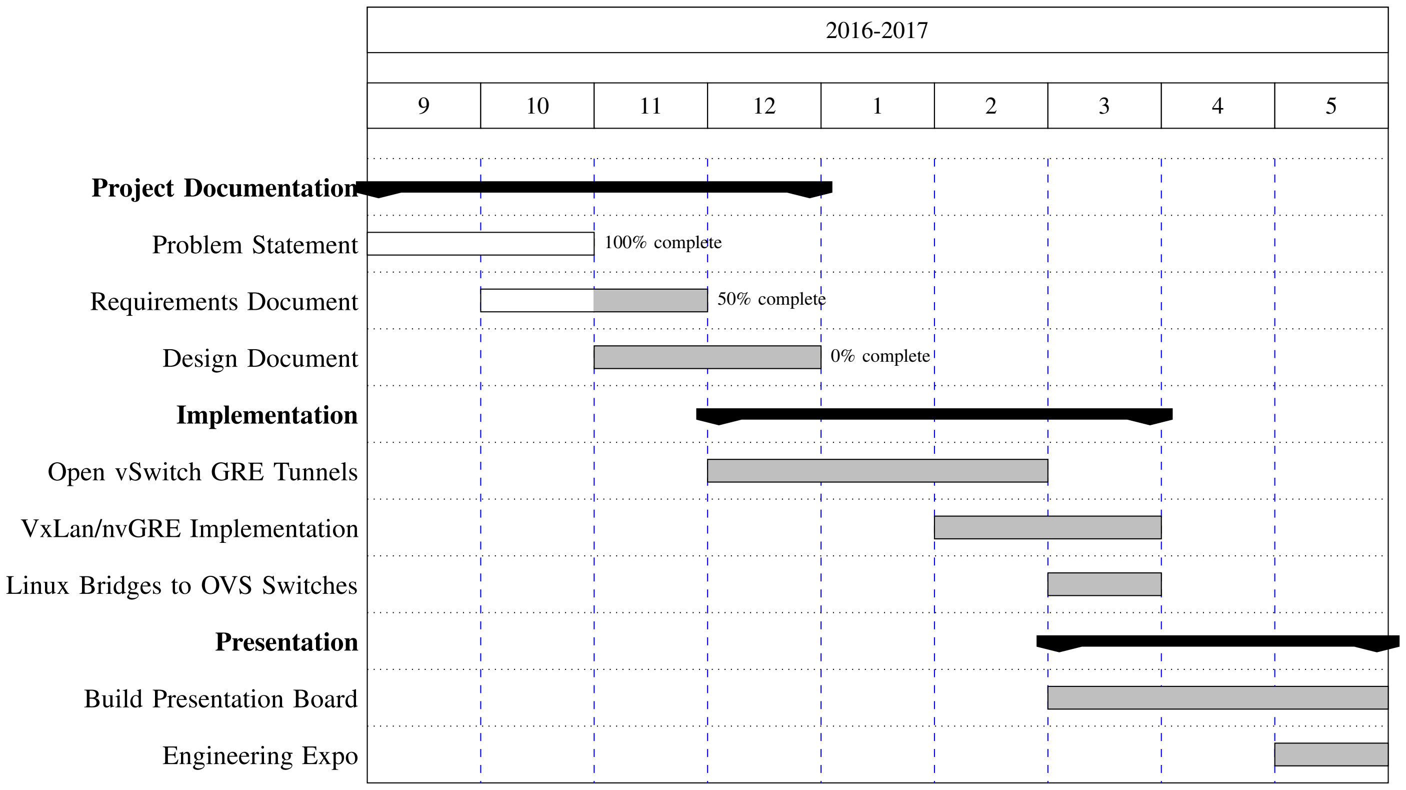
A stretch goal is to replace the currently-implemented Linux bridges with Open VSwitch switch instances.

The specific requirements for this project are as follows:

1) Documentation
   a) Problem statement
   b) Requirements document
   c) Design document
2) Implementation
   a) 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.
   b) 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.
   c) Optional: replace Linux bridges with OVS switches - if enough time remains, replace the Linux bridges with switches created by Open VSwitch.
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

III. Project Evolution and Changes

Our project goals over the year changed significantly. Large changing points were around week seven of winter term, as we found that our implementation plan was not feasible.

A. Scope Change

Originally, we were planning on replacing the Linux tunnels with Open vSwitch generated tunnels. This ended up not being possible, as the Open vSwitch management database requires a full stack implementation. Specifically, we had to generate the the original endpoint, known as a bridge, and tunnels with Open vSwitch. As a side effect of having to restart the project week seven of winter term, we moved VxLan and NvGRE implementation to a stretch goal. This was approved by the customer.

B. Table of Specific Changes

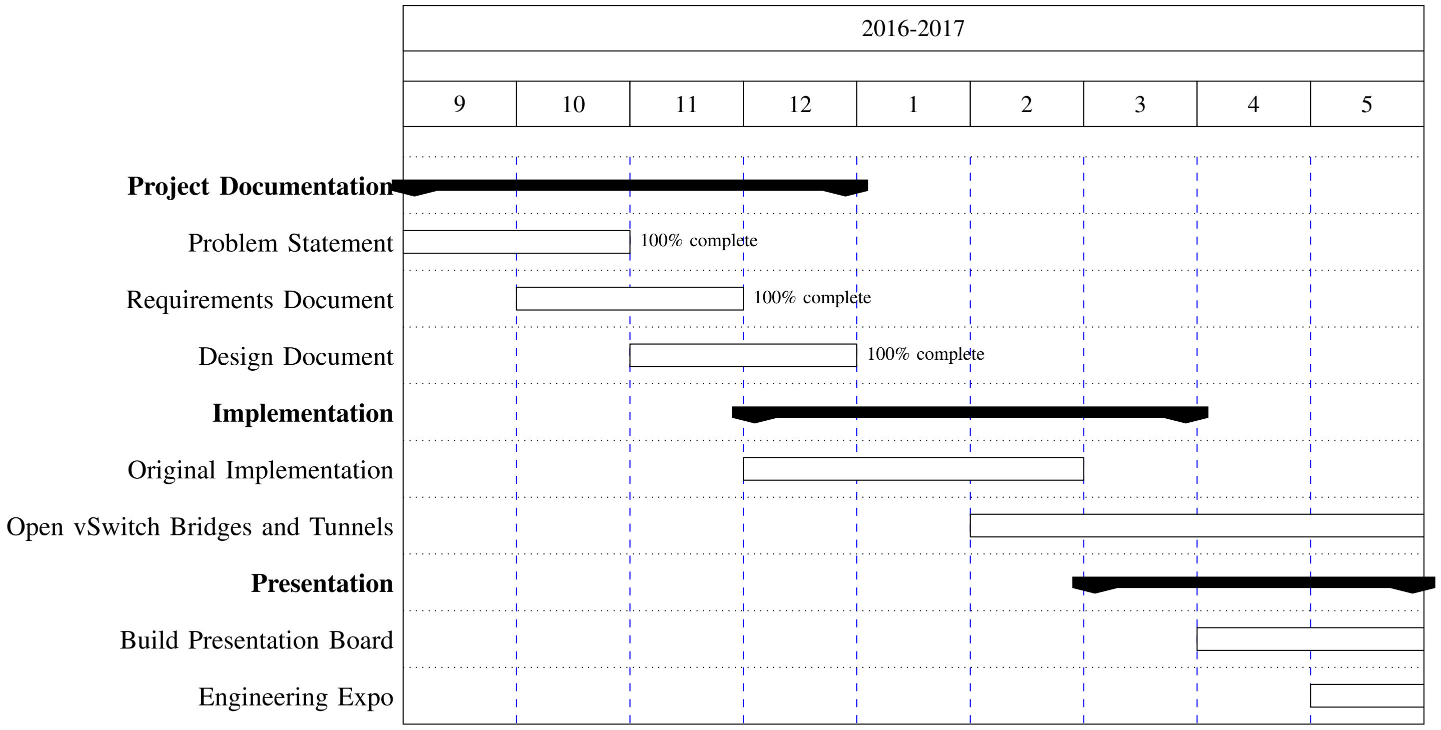
Shown below are the itemized list of changes make to our project after the initial design documents.

Scope Changes			
#	Original requirement	What happened to it	Comments
1	Implement OVS Tunnels into Ciao	Found goal to be impossible without bridges and tunnels both originating from OVS	Occurred week 7 of winter term, approved by client
2	VxLan/nvGre Implementation	Became a stretch goal due to time constraints on the project	Should be a simple change once our changes are merged into Ciao

1) Final Gantt Chart:

IV. Gantt Chart

Following is the actual distribution of work that occurred over the year. It is only slightly different than our original projections, mainly in spring term. Due to the scope change, we ended up working through the first four weeks of winter term to get the prototype in a working state. The list item ”original implementation” refers to work done before the scope change. Lastly, we group bridge and tunnel implementation due to the fact that one does not work without the other:



V. ORIGINAL DESIGN DOCUMENT

Intel Cloud Orchestration Networking Design Document

Abstract

This document outlines the design considerations for the implementation of Open vSwitch and other networking technologies in the Cloud Integrated Advanced Orchestrator (Ciao), Intel Corporation's advanced cloud orchestration software. It describes the various techniques, structure, and technology choices that will be used in the execution of our project.

Date of Issue: December 2nd, 2016

Issuing Organization: Oregon State University

Authorship: Matthew Johnson, Cody Malick, and Garrett Smith

Change History: First Draft, 12-02-2016

CONTENTS

I. Introduction

Our project is to first switch the Linux-created GRE tunnel implementation in Ciao to use GRE tunnels created by Open vSwitch. From that point we will switch the actual tunneling implementation from GRE to VxLAN/nvGRE based on performance measurements of each on data center networking cards. After this is completed, a stretch goal is to replace Linux bridges with Open vSwitch switch instances. This document outlines the steps, techniques, and methodology we will utilize to achieve each goal.

A. Purpose

The current implementation of Ciao tightly integrates software defined networking principles to leverage a limited local awareness of just enough of the global cloud's state. Tenant overlay networks are used to overcome traditional hardware networking challenges by using a distributed, stateless, self-configuring network topology running over dedicated network software appliances. This design is achieved using Linux-native Global Routing Encapsulation (GRE) tunnels and Linux bridges, and scales well in an environment of a few hundred nodes.

While this initial network implementation in Ciao satisfies current simple networking needs in Ciao, all innovation around software defined networks has shifted to the Open vSwtich (OVS) framework. Moving Ciao to OVS will allow leverage of packet acceleration frameworks like the Data Plane Development Kit (DPDK) as well as provide support for multiple tunneling protocols such as VxLAN and nvGRE. VxLAN and nvGRE are equal cost multipath routing (ECMP) friendly, which could increase network performance overall.

B. 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 such as the Data Plane Development Kit (DPDK).

C. Context

Our networking mode will exist within Ciao, a cloud orchestrator designed to be fast and easy to deploy. Ciao is sectioned into three parts, each with distinctive purposes [1].

Controller	Responsible for policy choices around tenant workloads [1].
Scheduler	The Scheduler implements a ”push/pull” scheduling algorithm. In response to a controller approved workload instance arriving at the scheduler, it finds a first fit among cluster compute nodes currently requesting work [1].

Launcher The Launcher abstracts the specific launching details for the different workload types (eg: virtual machine, container, bare metal). Launcher reports compute node statistics to the scheduler and controller. It also reports per-instance statistics to the controller [1].

Our networking mode must facilitate the communication of packets between all three levels of Ciao, as well as individual compute and network nodes and the Compute Node Concentrator (CNCI) [2].

Compute Node A compute node typically runs VM and Container workloads for multiple tenants [2].

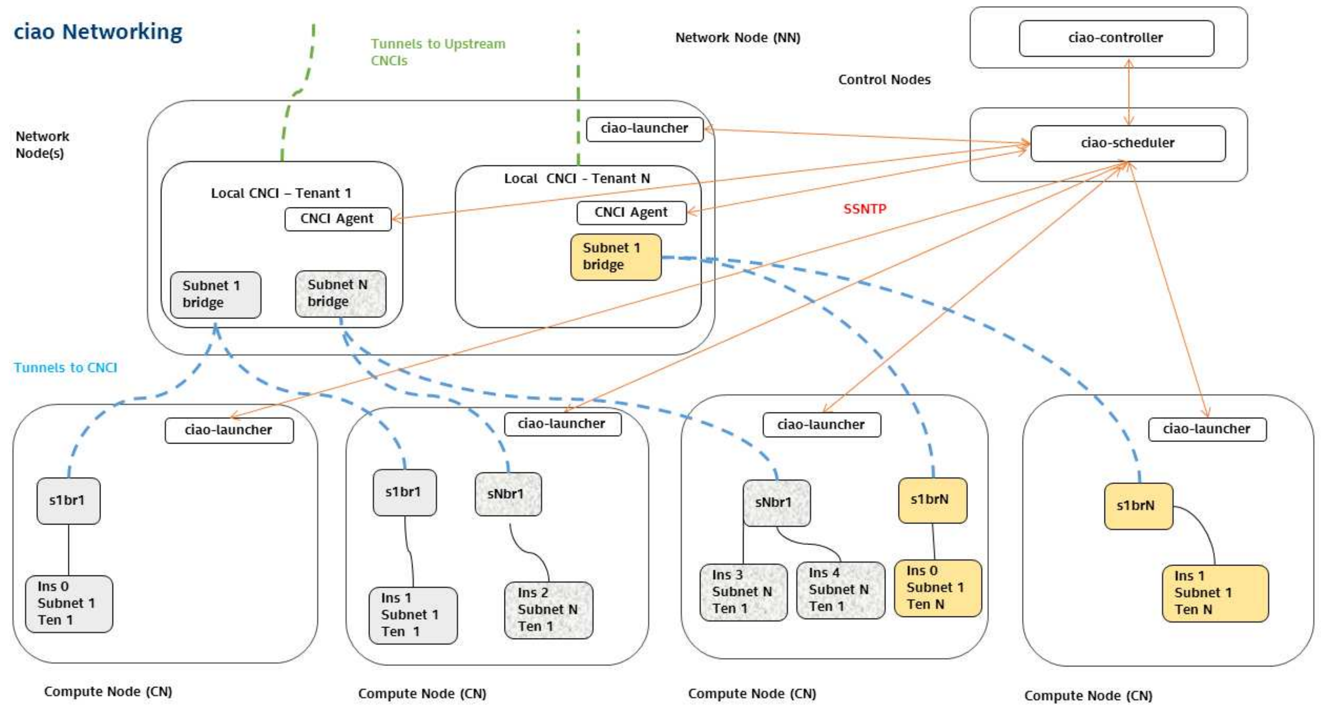
Network Node A Network Node is used to aggregate network traffic for all tenants while still keeping individual tenant traffic isolated from all other the tenants using special virtual machines called Compute Node Concentrators (CNCIs) [2].

Compute Node Concentrator (CNCI)

CNCIs are Virtual Machines automatically configured by the ciao-controller, scheduled by the ciao-scheduler on a need basis, when tenant workloads are created [2].

Specifically, the Ciao network components must communicate securely using the Simple and Secure Node Transfer Protocol (SSNTP). The network node aggregates traffic between compute nodes while keeping the tenant traffic isolated from other tenants in the cluster. Network nodes achieve this with CNCIs. A graphic of the lowest-level of this network configuration shows their relation to each other.

Fig. 1. Ciao Network Topology [3]



D. Summary

We will implement an Open vSwitch Generic Routing Encapsulation (OVS-GRE) mode in Ciao in order to leverage DPDK and other software defined networking technology innovations which are dependent on OVS. This document will outline our design strategy, design views, and design viewpoints for each component of our solution.

II. References

[1] T. Pepper, S. Ortiz, M. Ryan *et al.* (2016, sep) Ciao readme. [Online]. Available: https://github.com/01org/ciao/blob/master/README.md

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

[3] (2016, apr) Ciao network topology. [Online]. Available: https://github.com/01org/ciao/blob/master/networking/documentation/ciao-networking.png

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

[5] DPDK. What it is. [Online]. Available: http://dpdk.org/

[6] D. Thaler. (2000, nov) Multipath issues in unicast and multicast next-hop selection. [Online]. Available: https://tools.ietf.org/html/rfc2991

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

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

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

[10] (2016, nov) Open vswitch. [Online]. Available: http://www.openvswitch.org/

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

[12] 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

[13] 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

[14] *Standard for Information Technology–Systems Design–Software Design Descriptions*, IEEE 1016-2009, 2009.

[15] E. B. Pfaff, B. David. (2013, dec) The open vswitch database management protocol. [Online]. Available: https://tools.ietf.org/html/rfc7047

[16] Socketplane. (2016, dec) libovsdb. [Online]. Available: https://github.com/socketplane/libovsdb/blob/master/README.md

[17] ——. (2016, dec) play_with_ovs.go. [Online]. Available: https://github.com/socketplane/libovsdb/blob/5113f8fb4d9d374417ab4ce35424fbea1aad7272/example/play_wit

[18] C. Corporation. (2015, jan) Vxlan overview: Cisco nexus 9000 series switches. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.pdf

[19] D. J. Margaret Rouse. (2013, mar) Nvgre (network virtualization using generic routing encapsulation). [Online]. Available: http://searchsdn.techtarget.com/definition/NVGRE-Network-Virtualization-using-Generic-Routing-Encapsulation

[20] Y. W. P. Gard. (2015, oct) Nvgre: Network virtualization using generic routing encapsulation. [Online]. Available: http://ietfreport.isoc.org/idref/draft-sridharan-virtualization-nvgre/

[21] B. Mah. (2016, feb) iperf3: A tcp, udp, and sctp network bandwidth measurement tool. [Online]. Available: https://iperf.fr

[22] J. George, *qperf(1)*.

[23] B. Salisbury. (2016, mar) networkstatic/iperf3. [Online]. Available: https://hub.docker.com/r/networkstatic/iperf3/

[24] A. Schaaf, “arjanschaaf/centos-qperf,” jun 2015. [Online]. Available: https://hub.docker.com/r/arjanschaaf/centos-qperf/

[25] S. Yegulalp. (2015, jun) What’s the go language really good for? [Online]. Available: http://www.infoworld.com/article/2928602/google-go/whats-the-go-language-really-good-for.html

III. Glossary

Bridge	Software or hardware that connects two or more network segments.
Ciao	Ciao is a cloud orchestrator that provides an easy to deploy, secure, scalable cloud orchestration system which handles virtual machines, containers, and bare metal apps agnostically as generic workloads. Implemented in the Go language, it separates logic into ”controller”, ”scheduler” and ”launcher” components which communicate over the ”Simple and Secure Node Transfer Protocol (SSNTP)” [1].
Cloud	A huge, amorphous network of servers somewhere [4].
Cloud Orchestration	A networking tool designed to aid in the deployment of multiple virtual machines, containers, or bare-metal applications [1].
Compute Node Concentrator (CNCI)	Virtual Machines automatically configured by the ciao-controller, scheduled by the ciao-scheduler on a need basis, when tenant workloads are created [2].
Data Plane Developement Kit (DPDK)	DPDK is a set of libraries and drivers for fast packet processing. It was designed to run on any processors. The first supported CPU was Intel x86 and it is now extended to IBM Power 8, EZchip TILE-Gx and ARM. It runs mostly in Linux userland [5].
Equal Cost Multipath Routing (ECMP)	Equal cost multipath routing is a routing strategy in which next path routing for a packet can occur along one of several equal-cost paths to the destination [6].
Generic Routing Encapsulation (GRE)	Encapsulation of an arbitrary network layer protocol so it can be sent over another arbitrary network layer protocol [7].
Linux Bridge	Configurable software bridge built into the Linux kernel [8].

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 (CNCIs) [2].
nvGRE	Network Virtualization using Generic Routing Encapsulation [9].
Open vSwitch	Open source multilayer software switch with support for distribution across multiple physical devices [10].
OVS	Open vSwitch [10].
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 [11].
SSNTP	The Simple and Secure Node Transfer Protocol (SSNTP) is a custom, fully asynchronous and TLS based application layer protocol. All Ciao components communicate with each others over SSNTP [12].
Tunnel	Point to point network connection that encapsulates traffic between points [11].
VxLAN	Virtual Extensible Local Area Network [13].

IV. Body

A. Design Stakeholders

The stakeholders are Intel, and the members of the Oregon State University capstone group working on the project, Matthew Johnson, Cody Malick and Garrett Smith. Additional stakeholders are the existing and future users of Ciao.

B. Design Concerns

The design concerns are the platforms that need to be supported, the implementation of Open vSwitch created GRE tunnels, the implementation of VxLAN tunnels, the implementation of nvGRE tunnels, gathering performance metrics for the tunneling implementations, replacing the Linux bridges used by Ciao with Open vSwitch switch instances, logging, and testing. The capstone group members are stakeholders interested in all of the design concerns because they will be implementing all of the concerns. Intel is interested in all of the design concerns because they are the clients and they will be using Ciao.

C. Context Design Viewpoint

The context design viewpoint used here is from the IEEE 1016-2009 standard [14].

1) Context Design Concerns: The design concerns for the context viewpoint are related to overall compatibility both with the platform the software is running on as well as the software the networking mode is running in. It must also be compatible with existing networking protocols as used within Ciao already. Other major concerns include security and performance of the software defined network.

There is one primary use case initiated by a compute node sending a data packet to another compute node within the SDN.

Actor	Compute Node 1
Precondition	Compute Node 1 attempts to send data over network to Compute Node 2.
Postcondition	Compute Node 2 receives data sent by Compute Node 1
Main Path	1) Compute Node 1 encapsulates packet. 2) Compute Node 1 sends packet to Network Node. 3) Network Node routes packet via relevant CNCI to Compute Node 2. 4) Compute Node 2 receives packet and de-encapsulates.

2) Design Entities: The users of the networking mode we are implementing are the current and future users of Ciao, system administrators of small to medium enterprises running private clouds.

The actors of the networking mode are the components of Ciao networking, the compute nodes, network nodes, and CNCIs. These nodes handle the encapsulation, routing, and de-encapsulation of packets.

3) Design Relationships: Compute nodes both send and receive packets to other compute nodes. Network nodes aggregate this tenant network traffic while CNCIs within the network nodes keep tenant traffic isolated from each other.

4) Design Constraints: Design of the solution must integrate fully with Ciao as well as the Linux family of operating systems it runs on. This constraint demands specific technology choices as described below in the Context Design View section.

D. Context Design View

Compatibility with the rest of the Ciao system is paramount and drives many design decisions. The network implementation will be done in the Go programming language.

Because compatibility with the reset of the Ciao system is paramount, our software defined network will be written in the Go programming language and fully integrated in to Ciao. The Go programming language was selected for several reasons, including the efficiency of the language regarding both speed and memory, the concurrency capabilities, and the ease of implementation. Go was compared against C and Python as alternatives, and prevailed in every criteria except for availability of the language.

This network mode will be written as a standalone networking mode for Ciao as an additional option to the standard Linux bridges available now. For this reason, it must be fully integrated with the Ciao networking framework as it currently exists [2].

Since Ciao targets the Linux family of operating systems, our networking solution must also support Linux operating systems.

E. Interface Design Viewpoint

One of the main project goals is switching the GRE tunneling implementation from standard GRE tunnels to Open vSwitch generated GRE tunnels. The implementation of this object will be through designing and building an interface for Open vSwitch to hook into Ciao.

1) Design Concerns: Ciao will need to implement an interface for Open vSwitch in order to utilize the new feature set that OVS supplies. In order to access the OVS Management Database, we will need to interface with the Open vSwitch Database Management Protocol[15]. The following functions of the protocol will need to be created:[15]

Function Name	Parameters	Description
Insert	*table* : required *row* : required *id* : optional	The operation inserts specified value into table at row. If no id is specified, then a new unique id is generated.
Select	*table* : required *where* : required *columns* : optional	Searches *table* for rows that match all the conditions specified in *where*. If *columns* is not specified, all columns from the table are returned.
Update	*table* : required *where* : required *row* : required	Updates specified rows in a table. It searches *table* for rows that match all conditions specified in *where*.
Delete	*table* : required *where* : required	Operation deletes all the rows from *table* that match all the conditions specified in *where*

2) Design Elements: The primary method of interaction with Open vSwitch is through the Configuration Management Database, which provides a programmatic interface for updating and changing OVS on the fly. This will be the primary interface for our implementation.

F. Interface Design View

In order to interact with the interface using the Go programming language, the implementation will use the libovsdb library[16]. Libovsdb provides a direct interface for Go to access and modify the OVS Database configurations. The following is an example function call to call the above functions:[17]

Listing 1. Example insert operation in the OVS Database

```
// simple insert operation
```

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

Using the above code, slightly altered for each required operation, Go can be used to insert, select, update, and delete using a standard format.

G. *Interaction Design Viewpoint*

An important part of implementation is choosing a tunneling protocol to communicate between compute nodes. Currently, standard Generic Routing Encapsulation is used to create a virtual network. These tunnels are created at the Ethernet layer, and allow distributed systems to believe they are on the same physical network as another computer in a different location.

1) Design Concerns: While GRE tunnels get the job done, a new and more modern protocol is needed to support advanced virtualization features and scaling to large address spaces.

H. *Interaction Design View*

VxLAN is a relatively new virtualization standard developed by a few major players in the network industry. Cisco, VMware, Citrix, and Redhat all worked together to create this standard to resolve the major problem of massive virtual networks. VxLAN's primary advantage is that it has a massive address space, about sixteen million, and that its overall overhead increase is only fifty bytes [18]. Speed will be a deciding factor in which interface is chosen, so a smaller overhead is good.

NvGRE is another relatively new virtualization standard developed in tandem by Microsoft, HP, Intel, and Dell [19]. NvGRE sports a few of the same features as VxLAN, the primary difference between the two being the header field. They use different UDP port numbers and use a different bit to indicate that encapsulation has occurred [20]. The primary difference between the two is going to be overall performance in our testing environment, and in production.

Part of the implementation is selecting a more advanced tunneling protocol than standard GRE tunnels. The project will be implementing both nvGRE, and VxLAN, and testing the performance of both protocols in order to determine the best fit. Both protocols implement improved and more modern versions of tunneling software.

I. *Resource Design Viewpoint*

The resource design viewpoint used here is from the IEEE 1016-2009 standard [14]. We must provide explicit performance metrics for our SDN implementation. These metrics will be what we use to determine how our SDN implementation compares against the preexisting Ciao SDN implementation. The performance metrics must have numbers we can use to calculate actual and percentage differences in bandwidth and latency.

1) Design Concerns: The resource viewpoint was chosen because it covers the concerns of resource utilization and performance for the network and servers Ciao is running on. The specific resources we are interested in when gathering performance metrics for are network bandwidth and latency, and server CPU usage.

2) Design Elements: To compare our changes to Ciao with the current implementation, and decide which tunneling protocols to use in our final implementation we need to gather performance metrics. We will measure network bandwidth, latency, and CPU the usage of the hardware the SDN is running on before we make any changes, and using different tunneling protocols.

J. Resource Design View

To improve network performance we will use either the VxLAN or nvGRE tunneling protocol. We will gather performance metrics for the Ciao SDN when it is using the VxLAN and nvGRE tunneling protocols, and decide which protocol to use depending on which protocol performs better. Bandwidth and throughput are the most important performance metrics. To measure the performance metrics for the tunneling protocols we will use the iperf3 [21] and qperf [22] network tools. We will use iperf3 to measure network bandwidth and CPU usage. We will use qperf to measure network latency.

1) Design Elements:

a) iperf3: The iperf3 tool will be used to gather network bandwidth measurements, and the CPU usage of machines sending and receiving network traffic. We can run iperf3 in a Docker container. There is a public Docker image we can use so we will not need to build a Docker image for it [23]. The iperf3 tool must be run in a client server configuration. The docker image supports running iperf3 in either client or server mode. We will measure network bandwidth between different tenants on the network by running the iperf3 docker container in server mode on one tenant, and in client mode on other tenants. We can measure the CPU usage at the same time as the bandwidth because iperf3 will output both at the same time.

Listing 2. Running an iperf3 docker container in server mode

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

Listing 3. Running an iperf3 docker container in client mode

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

b) qperf: The qperf tool will be used to gather network latency measurements. We can run qperf in a Docker container. There is a public docker image we can use so we will not need to build our own qperf docker image [24]. The qperf tool must be run in a client server configuration. The docker image supports running in either client or server mode. To measure TCP latency the `tcp_lat` flag must be passed to the qperf client. To measure UDP latency the `udp_lat` flag must be passed to the qperf client.

Listing 4. Running a qperf docker container in server mode

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

Listing 5. 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
```

K. Design Rationale

Our design decisions were based largely around the goal of maintaining compatibility with Ciao and the platforms that Ciao runs on, as well as improving the capabilities and performance of Ciao's SDN. Implementation will be done in Go in order to integrate with the larger system and because the Go programming language is platform independent [25].

Open vSwitch was chosen because of the various features offered. OVS has support for packet acceleration frameworks such as DPDK and advanced SDN tunneling protocols such as nvGRE and VxLAN which would allow leverage of ECMP routing and potentially increase network performance.

V. Signatures

________________________ Robert Nesius, Engineering Manager

________________________ Matthew Johnson

________________________ Garrett Smith

________________________ Cody Malick

A. *Design Changes*

As stated in the scope change section, the primary changes were that we had to do a full stack implementation. To summarize the change, we originally were required to do a re-implementation of the linux-created GRE tunnels using Open vSwitch. After testing we discovered that Open vSwitch does not recognize linux-created bridges due to the internally-managed state in the OVS database. Therefore, all OVS-created GRE tunnels must attach to an OVS-created bridge. Originally, creating OVS bridges and integrating them into Ciao was considered our stretch goal. With this revelation we realized that creating the OVS bridges must be our first priority. This bumped the original second goal to test NVGRE and VXLAN protocols into stretch goal status and the GRE tunnel creation became our second goal. This approach required handling bridge creation and destruction as well as creation of GRE tunnels and attaching them to the OVS bridge objects. This was essentially integrating through the entire Ciao network stack.

VI. Original Tech Review

CS 461 - Fall 2016 - Technology Review

Matthew Johnson, Cody Malick, Garrett Smith
Team 51, Cloud Orchestra

Abstract

This document explores nine components of the software defined network that will be implemented by the Cloud Orchestration Networking Project sponsored by the Intel Corporation. For each component of the system, three separate technologies are evaluated and compared. Finally, a technology is decided upon for every component.

Contents

I. Introduction

The nine components of the software defined network we are implementing are defined in this document. For each component three technologies are evaluated and compared. We select the technology we will be using for that component based on criteria we have defined and explored.
The nine components of our system are:

Programming Languages	The language our solution will be implemented in. Authored by Matthew Johnson.
Logging	The logging library our project will use to record events. Authored by Matthew Johnson.
Functional Testing Framework	The testing framework used to ensure functionality of our network implementation. Authored by Matthew Johnson.
Packet Level Protocols	The protocol for the creation and forwarding of packets. Authored by Cody Malick.
Network Virtualization Implementation	The software implementation type of a networking switch. Authored by Cody Malick.
Network Bridge Implementation	The software implementation type of a networking bridge. Authored by Cody Malick.
Software Switch for the Compute Node Concentrator and Compute Nodes	The software implementation of the control and compute nodes. Authored by Garrett Smith.
Network Latency Measurement Tools	Analysis of network latency tools that will be used to test the implementation. Authored by Garrett Smith.
Network Throughput Testing Framework	The testing framework used to measure throughput of our network implementation. Authored by Garrett Smith.

II. Project Roles

A. Ciao and Golang Specialist

Matthew is our Ciao specialist and primary Golang engineer. With deeper knowledge and understanding of the Ciao architecture, as well as it's implementation language, his primary role will be handling details related to Ciao.

B. Network Specialist

Cody has great familiarity with network routing and protocol implementations. As the project is implementing pieces of a software defined network, Cody's primary role will be in understanding and implementing project details related to networking.

C. OVS and Testing Specialist

Garrett will be in charge of understanding Open vSwitch, one of the major components of the project. Almost every part of the project plugs into OVS in some way, shape, or form. He will also be working in-depth with testing tools to help ensure the project's quality at the end of the class.

III. Programming Languages

At the highest level, a main component of our software defined network implementation is the programming language it is written in. This is an important decision to make early in our design, as it affects all choices that follow.

A. Options

1) Go: Our first choice is the Go programming language. This is naturally the strongest choice as the rest of the Ciao infrastructure is written in Go. It would require a very strong argument to create a separate networking mode in another language. While technically possible, it would require a great deal of work to make the different pieces compatible with each other.
2) C: Other than interaction with the rest of the Ciao project, C is another natural choice. C has been around for decades, and has the capability to do nearly every computational and networking task. The libraries are extensive and available and the language is fast.
3) Python: Python is a choice here simply because of its ease of use. Python is very expressive and has nice libraries that abstract away the complicated details of software defined networking. The main downfalls of Python, however, are its reduced speed and space efficiency compared to Go and C. A result of writing a cloud orchestrator in Python is exemplified in the extremely complicated and slow Openstack project [1].

B. Goals for use in design

As stated, the choice of programming language will affect all aspects of our design for this project, from code structure to networking libraries and module design. This choice will easily have the largest impact on our project.

C. Criteria being evaluated

Important criteria to consider is the availability of necessary libraries and of the language and its dependencies itself, the inherent speed of the language to be used, the security features the language offers, the concurrency capabilities, and the overall ease of use.

1) Availability: The most available language in terms of libraries and the language itself (regarding its standard libraries) is obviously C because of how ubiquitous it is, how universally available it is, and how extensive its standard libraries are [2]. Close behind C in availability is Python. Python is nearly as available as C is because of how popular it has become in the last ten years [3]. Python has many libraries that provide simple abstractions to networking functionalities.

Of all these languages, Go is the least available as it is the youngest and least popular of the three. Go is not normally available by default on most operating systems and must be installed by the user. Go does, however, have available libraries that make it very easy to implement networking, as is evidenced by the extent to which they are used in the Ciao project currently [4].

The following figure demonstrates the popularity of C, Go, and Python in the United States in the last ten years.

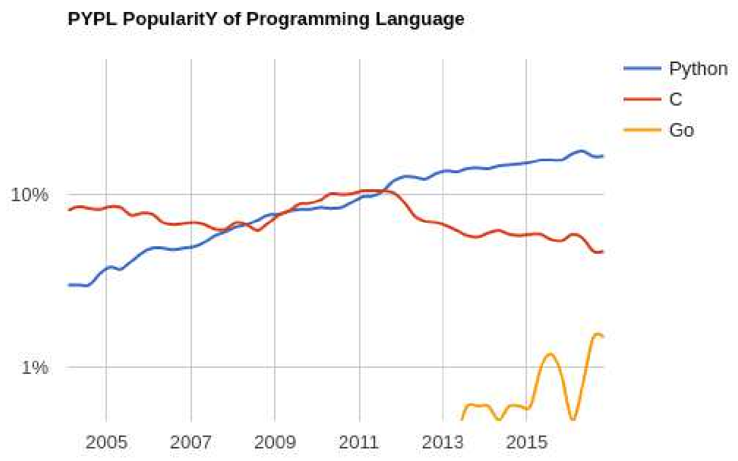


Fig. 1. Python, C, and Go popularity in the US [3]

2) Speed and Space Efficiency: One benefit of lower-level languages like Go and C is how they treat their variables. Go and C treat variables differently than some languages such as Python, which create overhead in order to track type information, and Java, which converts small ints to Integer class instances when placing them in a list. An example of this is in the representations of the same value in Go, Python, and C [5]:

```
var gocon int32 = 2014      // Go:   4 bytes
uint32_t gocon = 2014;      // C:    4 bytes
gocon = 2014                # Python: 24 bytes
```

Go performs comparably to C with regard to speed, as well [6], which is considerable since C is often the standard for fast programming languages. Compared to Python, as would be expected, Go and C can perform up to 45 times faster depending on the workload [6].

3) Concurrency: Concurrency is a key consideration for programming languages when implementing a software defined network. All operations must happen quickly and in parallel and must scale effortlessly. Therefore, it is necessary that all operations run in their own individual threads.

C has an extensive and established framework for parallel computing by utilizing pthreads. Mutexes can help the programmer protect against race conditions in their code, but the responsibility is up to the programmer to make their software thread-safe. Python has similar tools as C, but the parallelization is handled by a global interpreter lock (GIL). The GIL is a ”mutex that prevents multiple native threads from executing Python bytecodes at once.” GIL is necessary in python because the underlying C code that implements python is not thread-safe. The GIL ”prevents multithreaded CPython programs from taking full advantage of multiprocessor systems in certain situations” [7].

Go, on the other hand, combines the power of C and the ease of use and lock-handling of Python. You can use goroutines (functions that are capable of running concurrently with other functions) to create concurrency. Utilization of goroutines and other builtin language functionalities make concurrency very easy and lightweight in Go. An example from golang-book.com demonstrates how simple and lightweight threads in Go can actually be [8]:

```
package main

import "fmt"

func f(n int) {
   for i := 0; i < 10; i++ {
      fmt.Println(n, ":", i)
   }
}

func main() {
   go f(0)
   var input string
   fmt.Scanln(&input)
}
```

4) Ease of use: Python is by far the easiest to learn, use, and read. It focuses on ”readability, coherence, and software quality” and is recognized by many to be extremely easy to use [9].
C, being the oldest and lowest-level language of the three, is not a simple language to work with. Many things that are normally abstracted away in other languages are required to be programmed explicitly by the programmer. String manipulation is especially difficult in C.
Go is easier to learn than C and makes many improvements in terms of ease of use. It was even designed this way. Go was designed to make programming efficient in large-scale software development across teams with varying levels of experience and skill. It was designed with ”built-in concurrency and garbage collection” and includes ”rigorous dependency management.” [10]. These features are important for the work required by this project.
Another important note is that the rest of Ciao is written in Go, and while possible to create interfaces from C or Python for Go, it would be very difficult. Another option would be to re-implement Ciao in another language, which is so difficult and time consuming that it cannot even be considered as an option. With this consideration, Go is the clear winner in terms of ease of use.

D. Direct Comparison

Language	Availability	Efficiency	Concurrency	Ease of use
Go	3	1	1	1
C	1	2	2	3
Python	2	3	3	2

E. Selection

Based on the criteria explored above, the Go programming language is the language we are selecting to implement our solution in. The remainder of the technical review will be based on the assumption that we will be implementing our solution in Go.

IV. Logging

The logging libraries available for Go are almost innumerable, and most are very similar to each other. It is important that the library is fast, easy to use, and lightweight. Ciao currently uses the ’glog’ standard logging library, but due to the simplicity of logging it is feasible to choose another option.

A. Options

1) Zap Logger: Zap is a go logger that advertises ”blazing fast, structured, leveled logging in Go. According to benchmarks against other common logging libraries, Zap is indeed fast and lightweight in terms of management. There are major concerns here, however, including the overall size of the package along with its dependencies and the fact that it is in its beta phase [11].

2) Standard 'log' Library: The log library is standard in Go, and therefore very lightweight with no external dependencies. It is also simple to use with very many examples available [12]. It has less functionality than the zap logger, but we have very simple requirements.

3) Standard 'glog' Library: The glog library is very similar to the log library but adds a few features, like an improved leveled-logging functionality [13]. The main benefit here is that glog is used by the rest of the Ciao project, and is therefore trivial to setup and use in the networking portion [4].

B. Goals for use in design

Our goal for the logging module we select is to record important events and failures for debugging and functional testing.

C. Criteria being evaluated

1) Speed: As far as speed is concerned, the Zap logger appears to be the fastest of the options we are exploring, according to benchmarks [11]. Glog and log appear to be very similar, but glog makes a significant efficiency improvement by binding methods to booleans. This makes it possible to use glog "without paying the expense of evaluating the arguments to the log" [13].

2) Package Size: It is also important to consider the size of the dependency we are adding to Ciao. Zap, unfortunately, is a large package that is not already available in the project.

The log library, being part of the standard library, has no effective size since it is included with the language dependency. The glog package, as an external dependency, would normally lose outright in this category. However, the rest of the Ciao project already uses glog, so using it in our portion would not add a dependency to Ciao that does not already exist.

3) Ease of Use: Logging libraries are typically easy to use, and these three options are no exception. Zap is perhaps the most complicated to setup and use as it requires a structured log statement [11]:

```
logger.Info("Failed to fetch URL.",
   zap.String("url", url),
   zap.Int("attempt", tryNum),
   zap.Duration("backoff", sleepFor),
)
```

The standard log library is much simpler. An example of logging a statement in log [12]:

```
logger.Print("Hello, log file!")
```

The glog library is also very simple, but provides the option of leveled logs. The following is an example of logging an info and fatal statement, each in a single line [13]:

```
glog.Info("Prepare to repel boarders")
glog.Fatalf("Initialization failed: %s", err)
```

D. Direct Comparison

Logging library	Speed	Package Size	Ease of Use
log	3	1	2
glog	2	2	1
zap	1	3	3

E. Selection

Based on our evaluation of log, glog, and zap, glog comes out on top by a very small margin. The two simple packages, glog and log, were very close in every category, and it basically came down to the fact that glog is currently in use by ciao. We will be using glog for logging events in our portion of the project.

V. Functional Testing Frameworks

There are several testing frameworks available for Go. These range from the basic and simple 'testing' library to the more complicated and in-depth GoConvey and Ginkgo frameworks. Depending on our choice here, we could either add too much extraneous work getting a testing framework setup, or we could end up with simple tests that do not meet our needs.

A. Options

1) Standard ’testing’ library: The simplest and most lightweight option is the standard ’testing’ library. This can be expanded with the additional ’testify’ library, which adds functionality for mocking, assertions, HTTP protocol, and basic functions.
2) GoConvey: GoConvey is a testing framework that allows you to track your test status with a generated web UI. The tests can run automatically every time you save a .go file. GoConvey also generates an html test coverage report that allows you to review where your tests are falling short.
3) Ginkgo: GinkGo is a testing framework that allows you to write specifications for your tests. It is lighter-weight than GoConvey and works primarily from the command line. GinkGo takes a black box approach to testing.

B. Goals for use in design

The goal of the testing framework is to continuously verify our solution. Testing is important for any software, but it is particularly important in cloud orchestration networking. Our solution must be robust, and that can only be verified through rigorous testing.

C. Criteria being evaluated

The two important criteria when selecting a testing framework is the capabilities of the framework itself and the complexity of the framework. On one hand we want the tests to be powerful and support important features such as mocking and assertion. On the other hand we do not want the test framework to be so complicated that it is difficult to write tests.
1) Capability: The most important aspect of the testing framework is its ability to perform mocking. This is essential in unit testing. All three of the test frameworks allow the programmer to perform mocking.
Another important capability is assertion testing. Once again, all three support assertion tests.
A third important capability is randomization in testing. This is the best way to ensure corner cases are hit. This aspect, however, is provided by the Go standard library and therefore does not factor into this analysis.
One area where GoConvey shines is in its ability to perform testing analysis and output an easy-to-use web user interface. Ginkgo also provides some extra functionality related to black box testing.
2) Complexity: Of the three frameworks, the testing package is by far the simplest. Since it exists in the standard library, there is no overhead of installation or setup to get started writing tests [14].
Ginkgo is slightly more complicated to setup, since it is necessary to install and run a few Ginkgo-specific commands before ready to start writing tests. Once everything is ready, you can write tests, but must use Ginkgo-specific keywords and tests to utilize its full potential [15].
The most complex framework is the GoConvey suite. This is no surprise, considering how powerful it is. GoConvey requires the programmer to view the test results in a browser, as well [16]. This is difficult for our purposes, since we will be doing much of our development on remote machines via ssh, and will not have access to an X server to view the results.

D. Direct Comparison

Framework	Capability	Complexity
testing	3	1
GinkGo	2	2
GoConvey	1	3

E. Selection

Although the direct comparison reveals that all three are equally averaged to second place overall, the decision here is weighted towards the lowest in complexity. Our solution will be complex and difficult to implement, we do not want to have to worry about learning how to use a complicate testing framework on top of everything else. The standard ’testing’ package will be used for functional and unit testing of our solution.

VI. Packet Level Protocols

One of the main components of our system is deciding which protocol will be used to move data from one node to the next. Ciao provides its own data transfer protocol using SSNTP. Comparing how SSNTP works in general to other data transfer formats will provide insight into which protocol should be used.

A. Options

1) SSNTP: Simple and Secure Node Transfer Protocol is Intel’s solution for the transfer of data inside of Ciao networks. It is based on Transport Layer Security (TLS), the defacto standard for secure data transfer over the Internet. Part of what sets Ciao apart from the competition is its simplicity, as well as concise message format [17].

2) TLS: Transport Layer Security, or TLS, is the contemporary standard for secure data transfer across the Internet. It was built as an improvement upon the Secure Socket Layer protocol. It has great appeal as it is widely utilized, modern, and well studied [18].
3) SSL: Secure Socket Layer, or SSL, was the first version of what eventually became TLS following its third major revision. Originally utilized in Netscape, it wasn't widely utilized until it's third major revision. Secure Socket Layer encrypts at the application layer, allowing broad general use. While being somewhat older than the other protocols on the list, it is well understood and the security has been well proven [19].

B. Goals for use in design

The primary goal of the selected protocol is fast, stable, secure, and scalable communication between compute nodes and the orchestration node in Ciao. These are simple, but critical goals for the project.

C. Criteria being evaluated

The criteria for each protocol will be their security capabilities, protocol overhead, and ease of use. While stability is a concern, each protocol has shown stability through implementation in other applications many times over. This will be less of a focus for evaluation purposes.
1) Speed: Speed is a major consideration when working with a software defined network. Our final selection for a protocol needs to be fast. Any major delays in communication due to encryption or protocol overhead is a concern. While each implementation of the protocol will have its pros and cons, here the focus will be on application overhead to keep the metric in measurable territory.
SSL has two major components, the message protocol, and the handshake protocol. The handshake protocol is important as it ensures the overall security and authenticity of the communication between point A and point B [20].

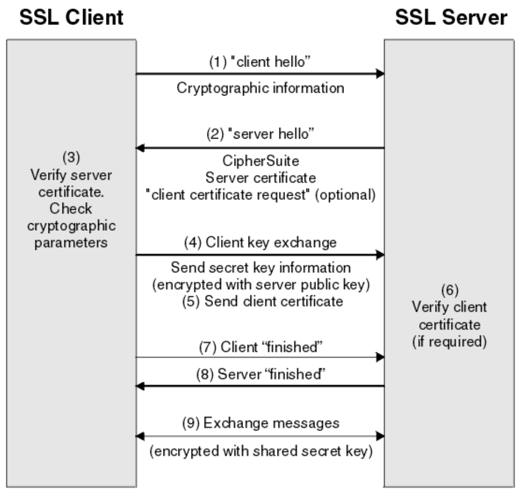

SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

Fig. 2. Simple SSL and TLS Handshake[21]

The SSL/TLS handshakes are identical. As far as overhead the two protocols are very similar. The primary difference between them is their security levels, which will be discussed in the following section.
The SSNTP equivalent of a handshake is called a SSNTP Connection. There is a fundamental difference that needs to be noted between SSNTP and the other two options. The difference is that the computers running SSNTP client software only have to do an authentication handshake, or SNTP Connection in this case, once. Once the client knows which server it is talking to, all connections no longer require a handshake step. This is a massive drop in needed overhead. [17]. Once the handshake is established, the client communicates with the server in an asynchronous fashion as needed.
2) Security: Security is important in choosing a communication protocol. If it wasn't, we could just use TCP, the protocol that all of these security protocols encapsulate.
SSL is unfortunately at the bottom of the list as far as the protocols listed here are concerned. Has a few vulnerabilities that have been found in the decade or so. While some of these are less of an issue, one attack in particular called POODLE has caused several major groups to call for SSL 3.0 to be deprecated [22]. While calling into question consideration for using this protocol on this system, it is still better than not having any security or encryption at all.

TLS is much better on this front as it is continually being updated. The latest release, version 1.2, uses AES, RC4, and a few other modern encryption ciphers [18]. Which encryption cipher is used is negotiated between the host and the client before communication begins.
SSNTP uses the TLS encryption protocols. On this front, it is equal to TLS.
3) Accessibility: All of the above protocols are well documented and easy to implement. In the scope of the project, however, SSNTP would be easier to use because other components of the system already use it. For that reason alone, it is simpler to get working for the project.

D. Direct Comparison

Protocol	Speed	Security	Accessibility
SSNTP	1	1	1
SSL	2	3	2
TLS	2	1	2

E. Selection

For our project, the technology for protocols will be SSNTP. It does what SSL and TLS do equally security wise, but is much more light weight and is more accessible in the context of the project.

VII. Network Virtualization

Because Ciao works off of a virtual network, it's important to pick the right tool for encapsulating packets to ensure they arrive at their destination. The standard virtual network tools have proven to be limited in their ability to scale. What the following tools do is allow for networks to scale to order of magnitudes higher than a standard virtual network would. As for virtual networks, virtual networks allow for virtual machines, or containers (docker, rkt) to be visible to the virtual network switch. This is powerful as a VM or container does not have a physical network card.
As we move to Open vSwitch, it gives us the option of using Network Virtualization using Generic Routing Encapsulation (nvGRE) or Virtual Extensible Local Area Network (VxLAN). Along with these two, a third option, Stateless Transport Tunneling (STT), will be contrasted to see which option fits the project needs best.

A. Options

1) VxLAN: VxLAN is a relatively new virtualization standard developed by a few major players in the network industry. Cisco, VMware, Citrix, and Redhat all came together to work on this standard to resolve the major problem of massive virtual networks. VxLAN's primary advantage is that it has a massive address space, about sixteen million, and that it's overall overhead increase is only fifty bytes [23]. Speed will be a deciding factor in which interface is chosen, so a fifty byte overhead is quite good.
2) nvGRE: NvGRE is another relatively new virtualization standard developed in tandem by Microsoft, HP, Intel, HP, and Dell [24]. NvGRE sports a few of the same features as VxLAN, the primary difference between the two being the header field. They use different UDP port numbers and use a different bit to indicate that encapsulation has occurred [25]. The primary difference between the two is going to be overall performance in our network setup.
3) STT: Stateless Transport Tunneling is a protocol developed by VMware to handle the same problem stated by the above protocols, but is primarily used to communicate between virtual switches. It is quite different than the other protocols as it carries much larger packets, up to sixty-four kilobytes, and it's overhead in general is much larger. This could be a major problem when it comes to choosing a protocol to use as overhead must be minimized.

B. Goals for use in design

The goal for these technologies is minimization of overhead while allowing for a virtual network to be mapped and allow for thousands of VMs and containers to become available easily. This problem of scalability is a major one, and one that has to be addressed when created large networks of computers hosting VMs or virtual machines.

C. Criteria being evaluated

The primary attribute being evaluated here is performance or lack of overhead. Performance and overhead go hand in hand, so they will be the primary metric of evaluation for this project. In terms of speed, all of these protocols move at the speed of hardware, so overhead makes sense for evaluation of the speed of the protocol. Less overhead equals more packets moved per second.

1) Overhead: NvGRE and VxLAN share almost identical header formats, with the primary difference in how they encapsulate their packets. NvGRE uses existing tunneling functionality in a TCP packet to encapsulate the data. All that is different is that is alters the twenty-four bits of data usually used for regular GRE to identify itself. This quite useful as it allows a great deal of existing hardware to support this ”new” protocol. The primary downside here being that loadbalancers and firewalls have to expand the the full packet, removing GRE, to inspect the packet. This could be a major slowdown if the packet inspection is a frequent activity [26].
VxLAN on the otherhand, while adding an additional fifty bytes of overhead to every packet, still requires the reversal of the encapsulation of every packet out of the VxLAN format. The other downside here being reduced compatibility with existing routing hardware. This is less of an issue for this project as all of our routing devices will be implemented in software [26].
STT has a significant disadvantage in the area of overhead when it comes to general use. It’s primary design purpose at VMware was to carry large amounts of data from point A to point B. In our use case, it doesn’t make much sense. Its overall packet header size will be between sixty-two bytes to eighty bytes. This is much larger than the other two. While this may not be a huge issue for VM to VM traffic, it becomes an issue when it travels across physical networks [27].

D. Direct Comparison

Protocol	Overhead
nvGRE	1
VxLAN	2
STT	3

E. Selection

For the purposes of the project, we will work with nvGRE in order to provide the lowest overhead. Live testing of nvGRE and VxLAN are part of the project. While it is difficult to tell during the design phase which will be better in production, on paper nvGRE seems to work better for our needs.

VIII. Network Bridge Implementation

Part of the project is generating GRE tunnels. Tunnels are very important when dealing with virtual networks. It allows a given VM to be mapped to the same network that another VM is on. This allows an arbitrary network to be created where a given set of VMs can be treated as a single network, regardless of their actual physical location. This allows a tenant to rent a set of VMs regardless of the set of servers they are on. While the GRE tunnels themselves are often identical, which is often the case when a standard exists, the features of the individual interfaces vary. There are two main options for this particular technology: Linux Bridges and Open vSwitch bridges. The third alternative, is not using them.

A. Options

1) Linux Bridges: Linux bridges are the standard for any user to create a network bridge from one network location to another. Linux bridges, as the name implies, ships with the Linux kernel. The big attraction to Linux bridges is that they are tried and tested, as well as being relatively simple. While being great for consistent deployment across devices, there are no updates occurring in the development area outside of the occasional security patch [28].
2) Open vSwitch Bridge: Open vSwitch is the only current area of major network innovation in the area of network bridging on Linux. While functionally, Linux bridges and OVS bridges are the same, OVS supports packet acceleration frameworks that Linux bridges do not [28].
3) No Bridge: The alternative to these two technologies is not having a bridge, and manually having to bridge networks with physical cabling. While possible, this is not a cost-effective method nor one that is easily accomplished without great manpower.

B. Goals for use in design

The goal in design of this system is to add an interface with the possibility for future expansion of capabilities as new systems and algorithms are created to improve bridging. For this implementation, larger feature sets, and an actively updated code base are ideal.

C. Criteria being evaluated

For this component, available features, as well as an actively updated code base are needed. This component is a preliminary for future implementation improvements, more concisely, it is groundwork.

1) Feature Set: In this metric, OVS vastly outstrips Linux bridges, and both Linux bridges and OVS are better than nothing at all. OVS has an actively updated code base, and has many more features available than Linux bridges [28]. One of the main features that make OVS bridges more appealing than Linux bridges are the availablilty of package acceleration frameworks. Not having any bridge at all, on the other hand, falls outside of our project requirements, and therefore is not a good option.

D. Direct Comparison

Bridge	Features
Linux Bridge	2
OVS Bridge	1
None	3

E. Selection

OVS is the right choice for this project, and the goals Intel has in the future for Ciao. It opens open a large feature set to be utilized, as well as opens Ciao up to benefit from any innovations made in OVS in the future.

IX. Software Switch for the Compute Node Concentrator and Compute Nodes

As part of creating a software defined network we need a software switch for the compute node concentrator(CNCI) and compute nodes(CN). Intel suggested we use Open vSwitch as part of the project requirements. The software switch needs to run on all of the platforms the Ciao supports, support GRE tunnels, VxLAN/nvGRE, and DPDK. I did some research on software switches and came up with LINCX and Lagopus as possible alternatives.

A. Options

1) Open vSwitch: Open vSwitch is an open source software switch written in platform independent C [29]. This means that it will run on all of the platforms supported by Ciao. It can operate as a kernel module paired with a daemon, or as user space application. For our use case we are interested in the kernel module paired with the daemon. Open vSwitch can be configured using a remote configuration server which works well with the CNCI and CN model. It has support for GRE tunnels, VxLan, nvGRE, and DPDK.
2) LINCX: LINCX is an open source software switch written in Erlang [30]. It requires the Erlang runtime be installed on the system it is running on. This means it should run on the platforms Ciao supports, but it adds another dependency. LINCX gets its configuration from a local configuration file. It does not have documented support for DPDK, or GRE tunnels.
3) Lagopus: Lagopus is an open source software switch written in C [31]. It only runs on Intel x86 machines. The only operating systems it officially supports are Ubuntu 14.04, Ubuntu 16.04, CentOS 7, FreeBSD, and NetBSD. It has support for DPDK, GRE tunnels, and VxLAN.

B. Goals for use in design

We are using a software switch as part of our SDN implementation to add functionality to Ciao. The software switch we use needs to support GRE tunnels, VxLAN, nvGRE, and DPDK to provide the additional functionality we are trying to add.

C. Criteria being evaluated

1) Supported Platforms: The software switch must support all of the platforms that Ciao currently supports so it does not reduce the number of platforms that Ciao can run on.
2) GRE Support: One of the requirements for the project is to use GRE tunnels. The software switch must support GRE tunnels to comply with the project requirements.
3) VxLAN Support: One of the requirements for the project is to use VxLAN. The software switch must support VxLAN to comply with the project requirements.
4) nvGRE Support: One of the requirements for the project is to use nvGRE. The software switch must support nvGRE to comply with the project requirements.
5) DPDK Support: One of the stretch goals for the project is to use DPDK with our SDN implementation. Even if we do not make it that far into the project the software switch still needs to support DPDK.
6) Configuration: One of Ciao's goals is minimal configuration. The software switch should be easy to configure, and needs to support remote configuration by the Ciao control nodes.

D. Direct Comparison

Name	GRE	VxLAN	Remote configuration	Language	DPDK support
Open vSwitch	Yes	Yes	Yes	C	Yes
LINCX	No	Yes	No	Erlang	No
Lagopus	Yes	Yes	Yes	C	Yes

E. Selection

We selected Open vSwitch for use as our software switch. This is the option that Intel wants us to use, and after researching other switches I agree with their choice. It has support for a wide variety of platforms unlike Lagopus which only officially supports Ubunutu 14.04 and Ubuntu 16.04, CentOS 7, NetBSD, and FreeBSD. LINCX supports any platform Erlang runs on, but does not have documented support for DPDK. In addition to the platform requirements, Open vSwich allows for configuration to be provided by a server. Both LINCX and Lagopus only support configuration via local configuration files. I could not find documented support for creating GRE tunnels with LINCX which removes it from the consideration. Combined with its lack of DPDK support it is not a good selection for this project. Lagopus has support for DPDK, as well as GRE tunnels and VxLAN, but it does not officially support all of the platforms we need it to. It also lacks the remote configuration support that Open vSwitch has. After taking all of this into consideration Open vSwitch is the best choice.

X. Network Latency Measurement Tools

One of our project requirements is to conduct network performance testing on our implementation. The purpose of testing network performance is to determine which tunneling protocols to use, and how our Open vSwitch implementation compares to the initial implementation. One of the metrics we are using to determine network performance is latency. Latency is how long it takes data to travel between hosts on a network [20].

A. Options

1) Wireshark: Wireshark is a network protocol analyzer that can be used to measure many network statistics including latency and throughput [32]. It has a command line interface called tshark. It supports saving network traffic to a capture file that can be analyzed at a later date. Wireshark only analyzes network traffic, it does not generate it. We would need to use Wireshark in combination with one of their recommended traffic generator tools to calculate latency [33]. Wireshark is very complex and has a lot of features we do not require for our use case. If we take the capture file it generates and open it with the GUI version of Wireshark we can generate graphs without feeding the data into another application.

2) Ping: Ping is a command line tool that can be used to measure the latency between two hosts [34]. It is a standard networking tool that is well tested and simple to use. The output is human readable, but not very machine readable. We would need to parse the output into a more computer friendly format before graphing it. It does not support throughput statistics so we would need to use another tool to calculate throughput. It can be configured to send a specific number of packets and will return the latency for each packet sent, as well as the minimum, average and maximum latency for the packets sent.

Listing 1. Example ping usage

```
[smithgar@flip3:~ ] > ping -c 5 oregonstate.edu
PING oregonstate.edu (54.244.95.93) 56(84) bytes of data.
64 bytes from ec2-54-244-95-93.us-west-2.compute.amazonaws.com (54.244.95.93):
    icmp_seq=1 ttl=42 time=10.6 ms
64 bytes from ec2-54-244-95-93.us-west-2.compute.amazonaws.com (54.244.95.93):
    icmp_seq=2 ttl=42 time=10.6 ms
64 bytes from ec2-54-244-95-93.us-west-2.compute.amazonaws.com (54.244.95.93):
    icmp_seq=3 ttl=42 time=12.8 ms
64 bytes from ec2-54-244-95-93.us-west-2.compute.amazonaws.com (54.244.95.93):
    icmp_seq=4 ttl=42 time=10.6 ms
64 bytes from ec2-54-244-95-93.us-west-2.compute.amazonaws.com (54.244.95.93):
    icmp_seq=5 ttl=42 time=11.1 ms

--- oregonstate.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 10.628/11.204/12.877/0.871 ms
```

3) Qperf: Qperf is a command line tool that can be used to measure network latency for both TCP and UDP packets [35]. It must be run in a client/server configuration. The client can then make requests to the server and receive latency statistics. The output is in human readable format. The following is an example taken from OpsDash showing how to use qperf to measure UDP bandwidth and latency [36].

Listing 2. Using qperf to measure UDP latency

```
root@node2:~> qperf -v 10.99.0.1 udp_lat
udp_lat:
    latency     =  46.7 us
    msg_rate    =  21.4 K/sec
    loc_cpus_used = 11.5 % cpus
    rem_cpus_used = 10 % cpus
```

B. Goals for use in design

This tool will be used to compare the latency of our SDN implementation against the starting Ciao SDN implementation.

C. Criteria being evaluated

The tool must work on Clear Linux which is the Linux distribution we are running Ciao on. It must allow us to measure network latency. It must have a command line interface. We must be able to capture the output and easily translate it into useful data. We should be able to graph the results.

D. Direct Comparison

Name	Output type	Single tool	TCP	UDP
Wireshark	Capture File	No, requires multiple tools	Yes	Yes
Ping	Human readable text	Yes	Yes	No
Qperf	Human readable text	Yes	Yes	Yes

E. Selection

I chose qperf as the tool for latency measurement. Wireshark requires more setup and configuration than qperf. Ping does not allow us to check the latency of UDP connections. We will need to install qperf on each of the machines we are performing network tests on by building it from source. I tested the installation process in a Docker container and it only takes a couple of minutes to download and build the source.

XI. Network Throughput Measurement Tools

As part of the network performance testing we are measuring network throughput. Throughput is how much data can be transferred between two hosts on a network in a fixed amount of time [20].

A. Options

1) iPerf3: iPerf3 is an open source tool for measuring network throughput [37]. It runs in a client server configuration on two hosts and measures throughput between the client and server. It can be configured to use random data, or to send a file across the network. It supports both TCP and UDP, as well as IPv4 and IPv6. It can be bound to specific network inerfaces on devices with more than one network interface. It can be configured to produce output in JSON format. This is an advantage because it is human readable, and easily parsible with many programming languages. Below is an example from the iPerf3 documentation of running it from the command line to measure the latency between hosts [37].

Listing 3. Sample iPerf 3 usage

```
node2> iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 60.0 KByte (default)
------------------------------------------------------------
[  4] local <IP Addr node2> port 5001 connected with <IP Addr node1> port 2357
[ ID] Interval  Transfer  Bandwidth
[  4] 0.0-10.1 sec 6.5 MBytes 5.2 Mbits/sec
```

```
node1> iperf -c node2
------------------------------------------------------------
Client connecting to node1, TCP port 5001
TCP window size: 59.9 KByte (default)
------------------------------------------------------------
[  3] local <IP Addr node1> port 2357 connected with <IP Addr node2> port 5001
[ ID] Interval  Transfer  Bandwidth
[  3] 0.0-10.0 sec 6.5 MBytes 5.2 Mbits/sec
```

2) Wireshark: Wireshark is a network protocol analyzer that can be used to measure many network statistics including latency and throughput [32]. It has a command line interface called tshark. It supports saving network traffic to a capture file that can be analyzed at a later date. Wireshark only analyzes network traffic, it does not generate it. We would need to use Wireshark in combination with one of their recommended traffic generator tools to gather network throughput metrics [33]. Wireshark is very complex and has a lot of features we do not require for our use case. If we take the capture file it generates and open it with the GUI version of Wireshark we can generate graphs without feeding the data into another application.
3) Netcat: Netcat is a tool that can be used to read and write data across networks from the command line [38]. When combined with timing software it can be used to calculate throughput. It is a standard networking utility and is well tested. The downsides to using Netcat are that you have to generate input using a tool such as dd.

Listing 4. Caclulating throughput with netcat and dd [39]

```
server$: nc -v -l 2222 > /dev/null
client$: dd if=/dev/zero bs=1024K count=512 | nc -v IP_OF_SERVER 2222
Result from dd:
536870912 bytes (537 MB) copied, 4.87526 s, 117 MB/s
```

B. Goals for use in design

We will be using this tool to compare the throughput of our SDN implementation against the starting Ciao SDN implementation.

C. Criteria being evaluated

The tool must work on Clear Linux which is the Linux distribution we are running Ciao on. It must allow us to measure network throughput for both TCP and UDP packets. It must have a command line interface. We must be able to capture the output and easily translate it into useful data. We should be able to graph the results.

D. Direct Comparison

Name	Output type	Single Tool	TCP	UDP
iPerf3	Human readable text or JSON	Yes	Yes	Yes
Wireshark	Capture File	No, requires additional tools	Yes	Yes
Netcat	Human readable text	No, requires additional tools	Yes	Yes

E. Selection

I chose iPerf3 as the tool for gathering network throughput statistics because it handles measuring throughput without any special configuration or other tools, and allows us to output the data in human readable format or JSON. This makes it easy to graph and analyze the results. Wireshark offers many of the same features as iPerf3 but it requires more set up and configuration, and does not output its results in as universal of a format. While Wireshark's built in graphing feature is useful, it does not outweigh the downsides. We would also need to use one of the recommended tools to generate network traffic [33]. Netcat must be used in combination with other utilities such as dd and does not output its results in as nice of a format as iPerf3.

XII. Conclusion

We explored nine components of our system and three technologies of each, resulting in an overview and comparison of twenty-seven technologies. We settled on one or more technologies per comparison. A summary of the selected technologies are listed below.

1) Programming Language - Golang
2) Logging Library - glog library
3) Functional Testing Framework - logging library
4) Packet Level Protocols - SSNTP
5) Switch Implementation - nvGRE
6) Bridge Implementation - Open vSwitch Bridge
7) Software Switch for CNCI and CNs - Open vSwitch
8) Network Latency Measurement Tool - Qperf
9) Network Throughput Measurement Tool - iPerf3

These selections were made based on research, and not trial implementation. These selections may change if blocking issues are discovered in the future.

XIII. References

References

[1] J. Bresler, "Tales from the trenches: The good, the bad, and the ugly of openstack operations," *Openstack Superuser*, jan 2015. [Online]. Available: http://superuser.openstack.org/articles/tales-from-the-trenches-the-good-the-bad-and-the-ugly-of-openstack-operations/
[2] tdammers. (2014, dec) What makes c so popular in the age of oop? top answer. [Online]. Available: http://softwareengineering.stackexchange.com/a/141345
[3] P. Carbonnelle. (2016) Pypl popularity of programming languages index. [Online]. Available: http://pypl.github.io/PYPL.html?country=US
[4] M. Castelino. (2016, may) Ciao networking. [Online]. Available: https://github.com/01org/ciao/tree/master/networking
[5] D. Cheney. (2014, jun) Five things that make go fast. [Online]. Available: https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast
[6] Stanford. The computer language benchmarks game. [Online]. Available: http://benchmarksgame.alioth.debian.org/u64q/compare.php
[7] python.org. (2015, jan) Globalinterpreterlock. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock
[8] C. Doxsey. (2016) Concurrency. [Online]. Available: https://www.golang-book.com/books/intro/10
[9] M. Lutz, *Learning Python*, 5th ed. O'Reilly Media, Inc, 2013.
[10] G. Rob Pike. (2012) Go at google: Language design in the service of software engineering. [Online]. Available: https://talks.golang.org/2012/splash.article
[11] Uber. (2016) zap. [Online]. Available: https://github.com/uber-go/zap
[12] golang.org. Package log. [Online]. Available: https://golang.org/pkg/log/
[13] golang. (2016) glog. [Online]. Available: https://github.com/golang/glog
[14] Google. Package testing. [Online]. Available: https://golang.org/pkg/testing/
[15] onsi. (2015, may) Ginkgo, a golang bdd testing framework. [Online]. Available: https://github.com/onsi/ginkgo
[16] smartystreets. (2016) Goconvey is awesome go testing. [Online]. Available: https://github.com/smartystreets/goconvey
[17] I. Corporation. (2016, sep) Simple and secure node transfer protocol. [Online]. Available: https://github.com/01org/ciao/tree/master/ssntp
[18] E. Rescorla. (2008, aug) The transport layer security (tls) protocol version 1.2. [Online]. Available: https://www.ietf.org/rfc/rfc5246.txt
[19] SSL.com. (2016, mar) What is ssl? [Online]. Available: http://info.ssl.com/article.aspx?id=10241
[20] K. W. R. James F. Kurose, *Computer Networking, A Top Down Approach*, 6th ed. Pearson, jan 2013.
[21] IBM. (2016, sep) An overview of the ssl or tls handshake. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm
[22] R. Barnes. (2014, oct) The poodle attack and the end of ssl 3.0. [Online]. Available: https://blog.mozilla.org/security/2014/10/14/the-poodle-attack-and-the-end-of-ssl-3-0/
[23] C. Corporation. (2015, jan) Vxlan overview: Cisco nexus 9000 series switches. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-729383.pdf
[24] D. J. Margaret Rouse. (2013, mar) Nvgre (network virtualization using generic routing encapsulation). [Online]. Available: http://searchsdn.techtarget.com/definition/NVGRE-Network-Virtualization-using-Generic-Routing-Encapsulation
[25] Y. W. P. Gard. (2015, oct) Nvgre: Network virtualization using generic routing encapsulation. [Online]. Available: http://ietfreport.isoc.org/idref/draft-sridharan-virtualization-nvgre/
[26] M. Sandbu. (2015, dec) Software-defined networking difference between vslan and nvgre. [Online]. Available: https://msandbu.wordpress.com/2015/12/03/software-defined-networking-difference-between-vxlan-and-nvgre/
[27] M. Terpstra. (2014, Jan) Stateless transport tunneling (stt) meets the network. [Online]. Available: http://www.plexxi.com/2014/01/stateless-transport-tunneling-stt-meets-network/
[28] R. Starmer. (2016, Jan) The battle of the switches: Ovs vs. linux bridge (or simplicity rules!). [Online]. Available: https://kumul.us/the-battle-of-the-switches-ovs-vs-linux-bridge-or-simplicity-rules/
[29] S. Finucane. (2016, nov) What is open vswitch? [Online]. Available: https://github.com/openvswitch/ovs
[30] A. Vasilenko. (2015, jan) Lincx - openflow software switch. [Online]. Available: https://github.com/FlowForwarding/lincx
[31] Y. Nakajima. (2016, nov) Lagopus software switch. [Online]. Available: https://github.com/lagopus/lagopus
[32] U. Lamping, R. Sharpe, and E. Warnicke. (2014, nov) Wireshark user's guide. [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/
[33] (2016, nov) Wireshark tools. [Online]. Available: https://wiki.wireshark.org/Tools
[34] *ping(8) - System Manager's Manual: iputils*, nov 2015.
[35] J. George, *qperf(1)*.
[36] *Measuring Network Performance in Linux with Qperf*, sep 2016. [Online]. Available: https://www.opsdash.com/blog/network-performance-linux.html
[37] B. Mah. (2016, feb) iperf3: A tcp, udp, and sctp network bandwidth measurement tool. [Online]. Available: https://iperf.fr
[38] C. Gibson, *Ncat Reference Guide*, jul 2013.

[39] J. Bowes. (2010, oct) Measuring network speeds with netcat and dd. [Online]. Available: https://jbowes.wordpress.com/2010/10/13/measuring-network-speeds-with-netcat-and-dd/

A. Tech Changes

Our tech stack changed in one area: we did not use `libovsdb`, the third party Golang library for Open vSwitch. We discovered that it added unnecessary overhead, and we opted to go for a direct interface with the OVS command line utility. After struggling with libovsdb integration into Ciao for several weeks, we communicated our issues to our client. Our client advised us to simplify our work and simply use the OVS utility. This simplified our work strategy considerably, since we did not have to manage table mutations for every network change, and instead could allow the utility to manage it internally.

VII. Blog Posts

A. Matthew

1) October 14, 2016: This is the first update for our project, so it will cover more than a week's worth of progress. First, a little background about our relationship with our client and how this project came to be...

a) Background: Over the Summer, 2016, I worked as a Software Engineering Intern for the Advanced Systems Engineering (ASE) group within the Open Source Technology Center (OTC) that is in turn within the Software Services Group (SSG) at Intel. So I was a SEI in ASE in OTC in SSG at INTC. If you've ever worked at Intel that sentence doesn't offend you... or maybe it does, I guess that depends on your experience at Intel.

I had the good fortune to work with some of the smartest people I've ever met. In an effort to continue working with them and continue challenging myself, I wrote a proposal requesting my group sponsor a Senior Capstone Project. I was successful. Not only did they enthusiastically embrace the idea, but provided enough quality hardware to be successful and enough mentorship to get us off the ground.

b) Progress: We have made some progress since being put on this project. A couple weeks ago we all traveled up to Hillsboro to meet with our clients, which turns out to include a couple Principal Engineers (PEs) who are miles more technical than us. During this meeting, one of the PEs was kind enough to spend an hour of his time explaining the more technical aspects of Ciao (I'm assuming I don't have to go into an explanation of our project here) and what they wanted us to do. Also during this meeting, we collaborated to write our problem statement due in this class. The next week we set up the hardware required to run Ciao in a cluster and began the process to register our hardware with the university so we could be granted IP addresses.

c) Issues: Since then we have been trying to jump through various hoops to get ethernet access on the hardware. This has been a continuing battle that we expect to be resolved soon. We need internet to use git, remote updates of Ciao and Clear Linux, and remote access to the cluster. We need *wired* internet because Clear Linux is a datacenter OS, and therefore does not even package the common wifi setup tools, such as `iw`, `wifi-menu`, `wpa-supplicant`, etc.

d) Next Steps: The next step is to install the Clear Linux OS on the five Intel NUCs in the cluster. Once that has been done we can set up Ciao and take initial measurements as a baseline. This baseline will be recorded so we can take further measurements once we implement our new networking mode using OVS.

– Matthew Johnson

2) October 21, 2016: Not much progress was made this week, though not for lack of trying.

a) Progress: This week was almost entirely devoted to figuring out networking for our hardware, 5 Intel NUCs. As mentioned previously, these NUCs will make up the control node and compute nodes for the cluster we will be running and testing Ciao on. After several days of debugging and a 22-email chain with the IT director at OSU, we are still blocked.

Other than debugging the networking with the IT director, we have successfully installed Clear Linux on all NUCs. This was done by bringing them all home and installing Clear on my home network. Tonight we will also receive feedback for our first draft of the Problem Statement. Hopefully we will have the feedback addressed by tomorrow morning (Friday, October 21st), since I will be driving up to Intel and will be able to get any changes approved and signed.

b) Issues: As I mentioned above, the issue with the networking is a big one. Hopefully we can get this figured out when the IT director returns on Monday.

c) Next Steps: The next step is to get networking figured out. This is a necessary step because we need to use the network to ssh into the NUCs, as well as communicate with the internet for github access and Clear Linux updates.

– Matthew Johnson

3) October 28, 2016:

a) Progress: This week we spent most of our time writing the rough draft for the requirements document. We have turned it in and it is definitely a good start. We will polish it over the next week to turn in next Friday.

Another big update for this week is that we have networking! I had another email conversation with Todd Shechter, who was at a loss as to why we could not access the network on our NUCs. He granted us access to the HP switch we had successfully connected via in the past. On Thursday, I went to move all our NUCs to the new switch. On a hunch I decided to try out the network one last time. They all worked! Our hardware is now set up and hopefully we will be able to get to work soon.

b) Issues: No issues currently, except for an increased workload from other classes. This being week 5, we all had midterms to contend with as well. Hopefully things calm down for a couple weeks.

c) Next Steps: The next step is to finish our requirements document and get it signed next week. We should also get Ciao installed on our NUCs and our cluster set up.

– Matthew Johnson

4) November 4, 2016:

a) Progress: This week we finished writing the requirements document that was due today. After the rough draft we turned in last week we continued working on it ourselves until Tuesday. On Wednesday Frank emailed us some suggestions (thanks Frank) and we addressed those right away. We got the document signed that afternoon by Rob Nesius, our client at Intel.

We will soon be receiving a book from Intel on SDN basics that will help us get started with implementation. We would have liked to be starting on implementation by now but the writing-intensive course load has tied up our time a bit.

b) Issues: Our workloads continue to be heavy, but that won't change. We are doing fine, completing our assignments on time, and have no immediate blockers.

c) Next Steps: Our requirements document is turned in, so now we are looking ahead to the design document. Hopefully we will also find time to start working on setting up our cluster in the next couple weeks.

– Matthew Johnson

5) November 11, 2016:

a) Progress: This week we started working on the technical review document due next Monday. This document outlines nine different components of our system. For each component we are exploring three different technologies that could be used to implement the component. Since our project is implementing a component of a larger system, it was difficult for us to come up with nine components and three technologies each (twenty-seven different options). We spent much of our week working together to figure out how to split the project up. We will all be working on our portions of the document over the weekend.

b) Issues: The biggest issue with the technical review document is that our system is a single component of a larger system, and is therefore difficult to split up into nine different parts.

c) Next Steps: The biggest looming deliverable is the technical review document due Monday. When that is complete we need to start on the very big design document due in a couple weeks.

– Matthew Johnson

6) November 18, 2016:

a) Progress: Over the last weekend we completed our technology review document, coming in at just over 16 pages total. Unlike several other groups we managed to complete and turn in our technology review on time. We are all satisfied with the result. Since then we have shifted to working on the design document that is due December 4th. This will be the largest document we produce this term and mostly marks the end of the documentation phase. The one document due after the design document is a term status report, which is mostly a combination of all the weekly updates we have written to give a coherent picture of our progress so far this year.

b) Issues: No issues currently.

c) Next Steps: The next big step is the large design document due at the beginning of next month. We will be working on that over the next couple of weeks.

– Matthew Johnson

7) November 25, 2016:

a) Progress: This was a short week with Thursday and Friday given over to the Thanksgiving holiday. We have transitioned to using the template requested by Kirsten for the design document and are ready to start with writing the document and preparing for our final presentation.

b) Issues: No issues currently.

c) Next Steps: The design document is our most immediate concern, followed closely by the end-of-term status report and presentation.

– Matthew Johnson

8) December 2, 2016:

a) Progress: This week we focused on the design document due Friday. We spent time researching design strategies and writing up our plan to execute. During this research we found a very useful Go library that interfaces with Open vSwitch. This library will simplify our implementation, allowing us more time to do network performance testing, which the client is very interested in.

b) Issues: No issues this week, we all worked hard and got the document done.

c) Next Steps: We are going to write the final report and do the final presentation this weekend. During Winter break we hope to start on the implementation.

– Matthew Johnson

9) January 13, 2017:

a) Progress: Over Christmas break Cody and I did some pre-work on the NUCs setting up our development environment and trying to get Ciao set up manually. We ran into some tough issues and communicated with the Intel team. The Intel team pointed us to a much more recently updated setup document that included automated deployment.

This week Garrett set up his development environment and we began to go through the automated deployment guide. After a show-stopper for much of this week we were able to continue with the deployment steps. Once deployed we will do our initial network measurements for baseline performance.

b) Issues: We ran into a show-stopper when Docker certification was broken in the latest version of Clear Linux (the target distribution). This was quickly fixed by the Clear development team and pushed external. We then had to remove the docker certification cache and everything worked fine again.

c) Next Steps: Next steps are to finish the Ciao deployment then gather initial network measurements. Once that is done we can implement our solution and test again.

10) January 20, 2017:

a) Progress: The docker issue was fixed and we were able to begin the Ciao deployment. We have been working through several issues, see below for details.

While Cody and I have been working on deployment, Garrett has been writing scripts to collect network performance data between nodes. He has made significant progress in parsing bandwidth data into csv format for graphical representation.

b) Issues: This week we continued to work on the Ciao deployment via automated ansible playbooks. We encountered several issues from the start and have worked through them one by one. Issues included errors in the ansible playbooks regarding yaml parsing and fqdn configuration. This type of issue was resolved by hardcoding the playbooks for our specific setup. We are now seeing a certification error that we will be working on this weekend.

c) Next Steps: Next steps are to finish the Ciao deployment and finish tooling for latency measurements.

11) January 27, 2017:

a) Progress: This week we were successful in deploying Ciao and beginning implementation of Open vSwitch components. Garrett has begun working out network measurements for our initial benchmarks. Progress is being made.

b) Issues: The first half of the week was spent debugging our Ciao deploy with members of the Ciao development team at Intel. After several email conversations and debug steps, we were advised to switch our operating system to Ubuntu because of various certification issues in Clear Linux. This, along with running from within a ciao-deploy docker container, fixed all of our issues. This has allowed us to move forward.

c) Next Steps: The immediate next step is to finish collecting network statistics and implement Open vSwitch-created GRE tunnels in Ciao (in the `gre.go` file).

12) February 3, 2017:

a) Progress: We have continued tentative development on the OVS modules for Ciao. Some of the necessary functions have been written, but it has been difficult to test this over the physical cluster. Garrett, who has worked on the OSU network in the past, thinks the network may not be assigning IPs to the cluster. The OSU network DHCP servers will not assign an IP unless the MAC address has been registered with the university. To get around this we may have to do single-vm setup or find a way to set up the cluster independent from the OSU network.

Ciao provides ciao-down, a tool that helps set up a single-vm environment for testing. This will be helpful once we get it spun up.

b) Issues: We are still having issues gathering initial network metrics due to inaccessible nodes on our physical cluster. Garrett is working through this but the best way to address this may be single-vm for now.

c) Next Steps: Getting ciao-down running is a priority so we can actually test the modules we are writing. We are also continuing to write the modules.

13) February 9, 2017:

a) Progress: We have a schedule we are following, and our current goal was to complete the OVS module by today. Although we have a module written, we are unable to get it to integrate properly with the rest of Ciao. The testing environment we are using is ciao-down, the single-vm setup for Ciao. I think we are misunderstanding some things in the calling hierarchy. I plan to work on it this weekend and communicate our issues to our client early next week. I'm attempting to set up a bridge with OVS as well as the GRE tunnel - which may not be strictly necessary. I thought it would make it easier.

b) Issues: As far as the physical cluster goes, we realized that the OSU network will not lease IP addresses unless the MAC address is registered with the University. Since Ciao uses the network's DHCP server to assign IP addresses to the nodes this is a big issue. Garrett has been setting up the physical cluster with a DHCP server running on the switch, with only our deployment NUC connected to the internet. He has been running into issues setting up the Keystone server once he took our controller NUC off the internet. He was trying to modify ansible tasks to get around the issue when I left campus today. If he isn't able to get this to work we will probably ask the university for a subnet we can run a non-MAC-locked DHCP server on, but Garrett has worked in OSU IT before and said we are unlikely to get permission for that. The number of errors the OSU network has generated for us has been a little frustrating.

c) Next Steps: I'll continue working on integrating the module. If I have no luck I'll work with the client next week to see what we are doing wrong.

14) February 17, 2017:

a) Progress: The majority of this week was spent preparing the Winter midterm report and presentation due on Friday.

Early in the week I met with Manohar Castelino, the networking expert on the Ciao project. Together we determined that OVS bridges are required to interface with OVS tunnels and this significantly changes the scope of our project. See our midterm report and updated requirements document for more detail. This makes the stretch goal, OVS bridge implementation, a primary goal along with creating the OVS tunnels.

b) Issues: No issues this week other than the scope change and associated goal reorder.

c) Next Steps: Next week we will continue implementing an entire OVS framework for Ciao, network bridges included.

15) February 24, 2017: This week we made considerable progress in writing a framework for the entire Open vSwitch network system. This was a new requirement identified only last week, causing a scope change for our project. Currently we have a framework written, but it is not integrated into Ciao. That is our future work. Currently Garrett is working on testing a DHCP server attached to an OVS bridge, which is required by the Ciao CNCIs.

a) Issues: No major issues this week.

b) Next steps: Integrate the new framework into Ciao and complete DHCP-OVS bridge testing.

16) March 3, 2017:

a) Progress: This week I moved houses, so my work was limited to the second half of the week. I worked on polishing the OVS framework for Ciao. I found that integrating into Ciao proper was extremely difficult because it is necessary to practically rebuild Ciao networking from the ground up. I spoke with Manohar Castelino about the scope of the project and he told us he is okay if we just call out to the OVS commandline tool instead of requiring direct API interaction. This has the potential to greatly simplify our project and hopefully we can get back on track.

b) Issues: Personal issues this week with two of three team members inhibited development. Those have been resolved and work continued in the latter half of the week.

c) Next Step: Exec command line tools to set up OVS instead of integrating complicated API calls into a complicated virtual network. I will meet personally with Manohar this next week as well.

17) March 10, 2017:

a) Progress: We made a lot of progress this week. I spoke with Manohar about the difficulties we were seeing integrating the OVS API into Ciao, and he advised me to use the OVS command-line interface. Once this decision was reached, we were able to quickly spin up some functions that accomplished what we had been trying to accomplish with libovsdb all along.

Once the functions were written I began integration work into Ciao itself. Right now the integration is in place, but I have had trouble seeing where the network mode is actually set in Ciao. This is my next step.

b) Issues: No significant issues for me this week.

c) Next Steps: The next step is to demo our progress to our TA and finish integration work into Ciao.

18) March 17, 2017:

a) Progress: This week we worked both on implementation and on creating a poster draft for the expo. We submitted the poster before the due date and switched our focus to implementation again.

I found an issue with our GRE tunnel creation code where we were not configuring the port properly with the IP address. A simple call to ifconfig configures the port.

Cody and Garrett found that we were not tracking bridge creation correctly in Ciao, causing Ciao to think that bridges were not created. They are working on this issue.

b) Issues: Ciao builds successfully, but it fails to actually connect the nodes. This is our ongoing work.

c) Next Steps: Continue to debug why Ciao is not connecting the nodes.

19) April 7, 2017:

a) Progress: Over Spring Break I was lucky enough to attend a three-day Ultimate Go programming language class provided by my work. Since this project is being completed in the Go programming language it was highly relevant.

When we arrived back for Spring term we started working again. We fleshed out some areas where Ciao needed to track our bridge creation better. I discovered on Wednesday that Open vSwitch was not being created on the compute nodes like we expected to.

b) Issues: The main issue causing failure was the failure to install OVS on the compute nodes. I emailed Manohar and we received a response with how that can be accomplished later in the week.

c) Next Steps: We really need to get this development finished as expo is fast approaching.

20) April 14, 2017:

a) Progress: This week I identified a bug in our implementation that was creating internal ports with the same name as the external-facing GRE ports. This was causing the external port creation to fail and preventing workload instances from being successfully created. When this was resolved (along with a couple other minor issues), we were able to create instances using OVS networking. While this represented a success and an important step towards completion, we are still having issues. The instances are ping-able (well, the compute node is, the instances are on separate ports, and ports are not ping-able), but we are unable to ssh into them. This is obviously a pretty big problem.

Cody and Garrett moved the NUCs to Cody's apartment to attempt physical cluster setup without OSU network restrictions. Once the physical cluster is set up we can gather network metrics on the hardware. We also worked on the poster draft and had our group picture taken.

b) Issues: Right now the main issue is a failure to connect the OVS bridge to the VNIC. The ovs-vswitchd logs indicate a netdev error, which I have been unable to root cause. The specific error is as follows:

```
2017-04-13T19:05:22.712Z|00016|netdev_linux|WARN|<bridge id>: obtaining netdev
    stats via vport failed (Invalid argument)
```

Then I see a lot of `file not found` errors with the bridge name as the file name.

c) Next Steps: Obviously, the next step is to figure out what the heck is going on with OVS here. We will also complete the poster and submit it Monday.

21) April 21, 2017:

a) Progress: This week we rolled back some recent experimental changes to Ciao's VNIC creation process in attempts to manage most of this via OVS. This was unsuccessful and caused the previously ping-able instances to be completely unavailable. We are now double checking that we are treating CNCIs and CNs the same way. A discrepancy here could cause problems.

We also have worked on finishing our poster and have responded to feedback from our client. Intel branding has been approved and the finishing touches to the poster are being made.

b) Issues: No new issues this week.

c) Next Steps: Finalize the poster and continue to debug the network issues.

22) April 28, 2017:

a) Progress: We worked on the poster this week as well as cleaning up our log statements for final code pull. The poster was approved by our client after some revision requests, which were made. The poster was submitted on Friday.

b) Issues: None

c) Next Steps: Wired review next week.

23) May 5, 2017:

a) Progress: This week we paired off with people from outside our group to conduct the Wired-style review. We also emailed our code to Manohar so he can look over what we have. We explained where we were stuck and asked him to look over it and see if there were any glaring errors. When we described our progress (that we can ping the instances but not ssh in) he said it is probably just a very small configuration issue that is preventing this from working fully.

b) Issues: None

c) Next Steps: Midterm report and video presentation coming up next week.

24) May 12, 2017:

a) Progress: This week we finished the Spring midterm video presentation and have been working on the written progress report. We are still awaiting feedback from Manohar on our code base if he has any.

b) Issues: No new issues this week. We thought we had broken the implementation again when demoing for our video presentation, but it turned out to be a simple configuration error.

c) Next steps: Turn in the progress report and prepare for expo next week.

25) May 19, 2017:

a) Progress: This week we submitted our midterm report/presentation on time. On Friday we presented at the Engineering Expo. The expo went well.

b) Issues: Nope

c) Next Steps: Final writing assignments to be assigned Wednesday.

26) May 26, 2017:

a) If you were to redo the project from Fall term, what would you tell yourself?: I would probably remind myself to test early and test exhaustively. Much of our early work was done in research, preparation, and setup. We found out halfway through Winter term that the specific technologies we were trying to integrate (Linux bridges and Open vSwitch GRE tunnels) were incompatible with each other due to the database nature of Open vSwitch. None of our research and proof-of-concepts with just Open vSwitch indicated this. If we had taken time early to test how our project would work with the exact combination of technologies we had to use, we could have caught our scope change earlier, possible Fall term instead of Winter term.

b) What's the biggest skill you've learned?: At the beginning of this year I knew *very* little about networking, particularly software-defined networking. Because of this project I have gained experience with Open vSwitch and cloud orchestration technologies. I had to go from 0-60 in a relatively short amount of time (well, maybe 0-55).

c) What skills do you see yourself using in the future?: I think Go programming will be pretty valuable in the future. It is a great new language that solves a lot of the security issues seen in C, while still providing that low-level control that makes C so attractive. I would like to do future projects using Go.

d) What did you like about the project, and what did you not?: I loved when things worked, I did not love when they did not work...

Seriously though, it was great to trouble-shoot some of the issues we were having and see positive results. Honestly, Open vSwitch is not terribly complicated to use, but integration with Ciao was exceedingly complicated. I think our remaining error has to do with Ciao integration rather than Open vSwitch issues.

e) What did you learn from your teammates?: I learned a ton about networking from Garrett. He came into this project with a much better understanding of things than I did, and he was able to correct many of my incorrect assumptions.

Cody has incredible organizational and people skills. He kept us on track for due dates and insisted on high quality for our written assignments. It is because of him our grades have been so high this year. Cody also is very knowledgeable in Golang, and served as a great human stack overflow when trying to remember go syntax.

f) If you were the client for this project, would you be satisfied with the work done?: I would be satisfied considering the scope change of the project. When we had to implement both bridges and tunnels using Open vSwitch, it changed how much of Ciao we needed to integrate with. The integration was the hard part, not the actual bridge and tunnel creation.

g) If your project were to be continued next year, what do you think needs to be working on?: There is still the issue of not being able to SSH into the instances that are created, so that needs to be addressed. Also, if feasible, converting the go execs to actual libovsdb API calls would be nice, but not necessary. I think the most important thing would be to complete our stretch goal. Originally our second goal, network measurements using NVGRE and VXLAN was moved to a stretch goal after our scope change.

B. Garrett

1) October 14, 2016:

a) Background: I'm interested in systems and infrastructure development and I would like to focus on it after I graduate. When Matthew asked me if I wanted to work on the Ciao project I was very excited. The project gives me the chance to work on a piece of infrastructure software, learn about software defined networking, and make a major contribution to an important open source project.

b) Progress: A couple of weeks ago our team drove up to Intel's campus in Hillsboro to meet with our clients. They went over Ciao's network stack with us, explained the project in depth, and answered questions. In addition to going over Ciao with us, they helped us write an abstract and problem statement for the project. We submitted the abstract and problem statement for the project on Thursday afternoon (2016/10/13).

Intel is lending us 5 NUCs (small form factor computers), and a managed Cisco switch to use as our datacenter for the project. Kevin McGrath is giving us access to his lab so we have a secure place to set up the hardware. We have the hardware physically in place in Kevin's lab, but we need wired network access to install Clear Linux on the NUCs and set up Ciao. We requested network access but the request has not yet been approved.

c) Issues: We are still waiting on access to OSU's wired network for the 5 NUCs and the switch that Intel are loaning us for the project. Network access is required to install Clear Linux, Ciao, and access the machines remotely. If we are unable to obtain network access for the hardware on OSU's network we will need to find somewhere else to house it.

d) Next Steps: We would like to get the network issues sorted out as soon as possible so we can finish setting up the NUCs and switch. Once we have network access we will install Clear Linux, set up Ciao, examine how Ciao's network stack currently works, and gather performance metrics. The performance metrics we gather will be used to compare Ciao's current network stack implementation against ours.

2) October 21, 2016: This week consisted of a lot of emails and not much progress. ### Progress #### Paperwork Our problem statement was reviewed and returned to us without any feedback. Cody emailed Kevin asking if we still need to resubmit the problem statement. We do.

a) Hardware and Software: In the land of NUCs and networks we are still having problems connecting the NUCs to OSU's network. Todd set us up with two 5 port switches in Kevin's lab, but we are having problems using them to connect to the network. Clear Linux requires network access to install, but network access requires us to set a hostname. We worked around the problem for NUC number 1 by bridging my laptop's wireless connection to the NUC over ethernet. This was a temporary workaround that let us install Clear Linux and set a hostname for the NUC. Unfortunately once the NUC was configured, we were still unable to obtain an IP address from the OSU network using the switches Todd provided. Several emails with Todd later we had NUC number 1 plugged into port 25 of the big 25 port switch in the middle of the lab, and connected to the network. This means the NUCs are configured correctly, and the issue likes somewhere in the network. At this point Todd emailed us that he was out of the office until next week, putting our network setup on pause. In the meantime, Matthew took the other 4 NUCs home and installed Clear Linux on them there.

b) Issues: As I said above, we are having problems getting the NUCs connected to OSU's network. Todd has been very helpful, but between email latency, and school and work schedules consuming chunks of the day the networking issues are taking a long time to resolve.

c) Next Steps: We need to work with Todd next week to get network access for the NUCs in Kevin's lab. If that goes through we will set up Ciao on the NUCs. Our problem statement does not require any edits so we will resubmit it. The requirements document is due on the 28th so we will be working on that.

–Garrett Smith

3) October 28, 2016:

a) Progress: The network issues have been solved. Matthew was going to move the NUCs from the small Cisco switches Todd provided us to the large HP switch in the middle of the room when Matthew discovered that the NUCs were able to connect to the network using the small switches. Many thanks to Todd Shechter for his assistance as we worked through the network issues. All five NUCs have access to OSU's wired network. Now we can access them remotely and configure Ciao.

In documentation land we submitted the rough draft of our requirements document.

b) Issues: We didn't have any issues with the Capstone project its self. Between midterms, homework for other classes, and attending the career fair I was very busy which left less time to work on capstone.

c) Next Steps: The requirements document is due next Friday (2016/11/4) so we will be working on it next week. We will also set up Ciao in its current form on the NUCs.

– Garrett Smith

4) November 4, 2016:

a) Progress: The final version of our requirements document has been signed and submitted. I think we would like to be furhter along in setting up the project and starting the implementation, but the writing requirements of the class have kept us busy.

b) Issues: There haven't been any issues with the class or eachother. We are making progress.

c) Next Steps: Next up is the design document. At some point we need to finish getting the NUCs set up with Ciao, but right now the class is focusing on documentation so we will be working on that primarily. Intel purchased each of us a copy of a book on software defined networking that is supposed to be arriving soon. I'm looking forward to digging into that and learning more.

– Garrett Smith

5) November 11, 2016:

a) Progress: This week we started work on the technology review. This involved splitting our project into 9 parts and assigning 3 to each member of the team. The goal of our project is to implement part of a larger system, so finding 9 smaller parts was difficult. After meeting two separate times we were able to figure out how to split the project into 9 parts. Each of us has 3 parts to work on now, and we will be completing those over the weekend.

b) Issues: There were no issues this week.

c) Next Steps: The technology review is due this coming Monday. We will be working on it over the weekend. After the technology review is complete, we will focus on the design document.

– Garrett Smith

6) November 18, 2016:

a) Progress: This past Monday (2016/11/14) we submitted the tech review document. It was a lot of work, but we got it in on time unlike many other capstone groups. I researched software switch options, network latency tools, and network throughput tools. For the software switch, Open vSwitch, the software switch recommended by Intel, was the best option. Nothing else I found came close in terms of features and portability. There were a lot of options for tools that measure network throughput and latency. I had a lot of choice when it came to choosing tools so I spent a lot of time reading recommendations, looking at all of the options, and comparing them to determine what would work best for us. Even though using the tools isn't part of the assignment, I was curious about how the networking tools I was researching worked so I set up and used some of them to get more of an idea of what I would be doing.

b) Issues: We didn't have any issues this week.

c) Next Steps: The design document is out next big assignment. We will be working on it over the next couple of weeks.

7) November 25, 2016: This week was shorter due to the Thanksgiving holiday.

a) Progress: This week was short due to the Thanksgiving holiday. We met and discussed how we will be completing the design document and progress report. We have the design document laid out using the template Kirsten wants us to use. We received feedback from Kirsten on the tech review. Some of the feedback was helpful generally like watching out for spacing issues.

In more of a personal progress report, I have been reading through the book on Software Defined Networking. It is very insightful, and I wish I had it at the start of the term.

b) Issues: No issues this week. The holiday is cutting into project time, but that is unavoidable.

c) Next Steps: I will continue reading through the book on software defined networking. The next big assignment that is due is the design document so we will be working on it next week.

– Garrett Smith

8) December 2, 2016:

a) Progress: This week we finished the design document. We each wrote up part of the design, then spent time editing and revising the document. We turned in the signed copy by the due date and did not need to turn in an unsigned copy. It feels good to have the first iteration of the design document complete because we are now in a position where we can work on implementing the project over winter break.

b) Issues: No major issues this week.

c) Next Steps: The progress report is due Wednesday next week so we will be focusing on that. We need to write both the presentation slides, and the document. Our plan is to meet up on Sunday to record the voiceover for our presentation.

– Garrett Smith

9) January 13, 2017:

a) Progress: I got my dev accounts set up on the NUCs. We met up and worked on getting Ciao set up.

b) Issues: We are unable to fetch remote docker images from docker.io because of a certificate error. For example, trying to fetch the Cio setup image results in the following error:

```
garrett@fw-dear205-ciao-nuc0 ˜ $ docker pull clearlinux/ciao-deploy
Using default tag: latest
Pulling repository docker.io/clearlinux/ciao-deploy
Error while pulling image: Get
   https://index.docker.io/v1/repositories/clearlinux/ciao-deploy/images: x509:
   certificate signed by unknown authority
```

At the time of writing this `swupd` reports our software is up to date.

```
swupd-client software update 3.7.4
  Copyright (C) 2012-2016 Intel Corporation

Attempting to download version string to memory
Update started.
Version on server (12680) is not newer than system version (12680)
Update complete. System already up-to-date at version 12680
```

Ciao has a nice automated setup utility that is run using docker. With docker down we can not use that. We were also planning on running the network analysis tools for latency and bandwidth measurement in docker containers. Docker not functioning is a blocker for us.

c) Next Steps:

- Gather network performance metrics for latency and bandwidth between nodes on the SDN.
- Implementation.

10) January 20, 2017:

a) Progress: Matthew and Cody have been setting up Ciao using the automated deploy docker image. They have had a few issues with the set up, but they are making progress.

I have been working on Ruby scripts to parse the output from iperf and qperf into CSV format so it is easier to work with. iperf was pretty simple to set up because it has an option to output its data in JSON format which is easy to parse. qperf does not have a computer friendly output option so the script for parsing its output is more complex. I am familiar with Ruby and it has strong native support for string manipulation so I chose it for this task.

b) Issues: We are having a cert issue with setting up Ciao that Cody and Matthew are working on.

c) Next Steps:

- Get the automated deploy working for Ciao.
- Finish up the script that parses the qperf output into a CSV.

11) January 27, 2017:

a) Progress: Great success! Ciao is up and running on the 5 Nucs. Now we can go full tilt on implementation.

b) Issues: We had some issues getting with the `getfqdn()` call returning the host name instead of the fully qualified domain name on Clear Linux, and a few issues getting the security certs set up for Ciao on Clear Linux. We switched to Ubuntu 16, after which we were able to get Ciao running.

c) Next Steps: I will be gathering network performance statistics over the next few days while we figure out how to set up OVS GRE tunnels.

12) February 3, 2017:

a) Progress: We worked on the development of an OVS module for Ciao so we can get OVS GRE tunnels implemented. Cody and Matthew have been spearheading that while I have been troubleshooting issues with the multi machine deployment of Ciao. More on those below. As a workaround, and to help with development we are going to set up Ciao-down which is a single machine deployment of Ciao.

b) Issues: Our deployment of Ciao on the 4 NUCs is not functioning correctly. We are unable to deploy new workloads which is preventing me from running gathering network performance statistics. When trying to run a workload the Ciao controller generates a 500 error: `{"error":{"code":500,"name":"Internal Server Error","message":"Unable to Launch Tenant CNCI"}}`. I dug into the source code and determined that this error message only occurs when the CNCI does not have an IP address.

I was lurking in the Ciao IRC channel when there was a discussion about moving CNCIs. One of the details of moving a CNCI is is that it has its IP address assigned by the underlying physical network. OSU's network will only assign IP addresses to machines with registered MAC address. This a a problem if we are going to be spinning up VMs on the fly. I have not confirmed this is the issue but it looks like a likely candidate. I emailed the Ciao developers and Tim Pepper gave me some tips for setting up more verbose error logging. I will be setting up more verbose logging and errors to try to determine if the CNCIs are not being assigned IPs, and to try to track down the root of the issue if that is not the problem.

c) Next Steps: We are going to get ciao-down running so we can run and test ciao on one machine. I will keep troubleshooting the multi machine deployment of ciao so we can gather performance metrics.

13) February 9, 2017:

a) Progress: Matthew and Cody have been working on implementing OVS created GRE tunnels. They have Ciao down running and have been developing using that.

I moved our Ciao cluster off of the OSU network onto a LAN running off of the switch Intel is lending us. I'm running the switch in layer 3 mode which allows me to use its built in DHCP server. I'm hoping this will resolve the problems getting IP addresses for the CNCI VMs. I have Nucs 0 through 3 connected to the LAN, and I'm working on getting Ciao running on them. Cody is currently using Nuc4 for his development environment.

b) Issues: I had some trouble getting the switch set up due to my unfamiliarity with its configuration software which took one development day to get working.

The current issue I am having is cleaning out the old configuration we had set for Ciao. The Ansible deployment is still using the old hostname (fw-dear205-ciao-nuc1.engr.oregonstate.edu) for Nuc1 when trying to connect to the Keystone server which is breaking the deployment. Git grep isn't showing the hostname as being hard coded anywhere in the config, but it is coming from somewhere.

c) Next Steps: Next week I will figure out the configuration issues and start gathering metrics. I also need to get an instance of Ciao down running so I can help out with development.

14) February 17, 2017:

a) Progress: This week we spent most of our time working on the midterm progress report.

Matthew met with Manohar Castelino, the Intel Ciao networking expert to discuss a possible change in the scope of our project. We need to implement our stretch goal of OVS bridges alongside of OVS tunnels because OVS tunnels require OVS bridges.

b) Issues: Nothing major this week.

c) Next Steps: Next week its back to development. I will also spend some time working on the network performance measurements.

15) February 24, 2017:

a) Progress: We made good headway on the OVS bridge implementation. I figured out part of the configuration with the Ciao cluster.

b) Issues: Nothing major this week.

c) Next Steps: Continuing implementation and testing.

16) March 3, 2017:

a) Progress: This week I got docker containers connected to an OVS bridge.

b) Issues: Nothing major.

c) Next Steps: Get a DHCP server connected to the bridge and see if I can have it assign IP addresses to the docker containers.

17) March 10, 2017:

a) Progress: This week we made some good progress. Right now we are calling the OVS CLI tool in Go to create bridges. See Cody's blog post for more details on how this works. This is easier to use, and faster on development time than using a library. It should also simplify the process of creating nvGRE and VxLAN bridges.

We discussed this with the client and they are fine with this approach.

I connected three docker containers to an OVS bridge, ran a DHCP server in 1 of them, and had the other docker containers request IP addresses from the DHCP server to confirm that the DHCP server in the CNCIs should work with the OVS bridges.

b) Issues: I spent a little bit of time fighting with Docker and dynamically assigned IP addresses. It ends up that

c) Next Steps: We need to add configuration to Ciao to enable switching between OVS and Linux bridges.

18) March 17, 2017:

a) Progress: This week we worked on the poster. We submitted our first draft with most of the information but no picture. Matthew has started commuting from Portland, so getting everyone together for a picture was not convenient this week. We will add one before the final poster submission is due.

We made more progress integrating the OVS bridges into Ciao. Cody and I looked into how the Bridge struct is used to create track bridges in Ciao and we are modifying it to have the option for OVS bridges.

```go
func NewBridge(id string, mode NetworkMode) (*Bridge, error) {
    bridge := &Bridge{}
    bridge.Link = &netlink.Bridge{}
    bridge.GlobalID = id //TODO: Add other parameters
    bridge.Mode = mode
    return bridge, nil
}
```

b) Issues: We weren't all able to meet to get a picture for the poster.

c) Next Steps: Finals are next week so we will be working on the progress reports.

19) April 7, 2017:

a) Progress: Over break Cody and I spent some time pair programming and made modifications to the Bridge struct that is passed around in Ciao. This struct is used to track bridges, and we had been bypassing it when they are created. We added a mode the the bridge to track OVS or Linux bridge type, and modified the `create()`, `destroy()`, `enable()`, and `disable()` methods to have different behavior depending on which mode is set.

b) Issues: We spent about 5 hours on Wednesday working together in a collaboration room and discovered problems with our dependencies. OVS is not being installed on tenant VMs. We emailed Manohar and received instructions for how the dependencies for tenants are configured so we are working on modifying those to get OVS installed correctly as a dependency.

c) Next Steps: Development, development and troubleshooting. We need to get our prototype working soon with expo fast approaching.

– Garrett

20) April 14, 2017:

a) Progress: We made some changes to the vnic (virtual nic) part of the Ciao networking library to allow us to attach the virtual interfaces to OVS bridges.

We switched to single VM for development, but we still need to get Ciao running on a cluster of machines. We moved the Nucs Intel is lending us for the project out of the lab at OSU and into Cody's apartment and onto his network. We have a lot more control over his network so hopefully we will have a smoother deployment experience there.

b) Issues: We are getting errors when we try to attach the virtual network interfaces to OVS bridges.

c) Next Steps: Get Ciao set up on the physical cluster running on Cody's home network.

21) April 21, 2017:

a) Progress: This week we continued looking into the issues with SSHing into the VMs.

We met with our client and Kirsten to discuss our progress and expectations for expo. Finally, we worked on the poster for expo.

b) Issues: No new issues, but we are still having issues SSHing into the VMs.

c) Next Steps: Finalize what we have an prepare for expo. Finish up the poster.

22) April 28, 2017:

a) Progress: This week we got the poster finalized and submitted.

b) Issues: Nothing major this week.

c) Next steps: Final code cleanup in preparation for the code freeze.

23) May 5, 2017:

a) Progress: This week we had the Wired style review to work on. That took a few hours, but I found it helpful because it gave me practice explaining our very technical project.

On Sunday we submitted our expo poster.

We did some code cleanup, and emailed Manohar with our progress.

b) Issues: Nothing new this week.

c) Next Steps: Next week we need to work on the midterm report and video.

24) May 12, 2017:

a) Progress: We did the midterm progress report and video this week.

b) Issues: Nothing this week.

c) Next Steps: Expo is next week so we will be doing that.

25) May 19, 2017:

a) Progress: This week was expo. I went to expo and presented.

b) Issues: My feet hurt.

c) Next Steps: Do whatever is left in the class.

26) May 26, 2017:

a) Post Expo Summary.: This is the last blog post after expo.

b) If you were to redo the project from Fall term, what would you tell yourself?: We had a big scope change because it was assumed that we could use Open vSwitch for just the tunnel endpoints. I would warn past me that this is an issue. To generalize the advice I would say to spend more time initially on proofs of concept and make sure that what we think we can do lines up with what we can do. We spent a lot of time researching and planning, but it would have been helpful to jump in with tools earlier and learn how they work and interact.

c) What's the biggest skill you've learned?: I learned a lot about networking, and specifically how software defined networks work. I know I will need networking knowledge for the job I have lined up after college, and working on this project gave me really good practical experience with software defined networks.

d) What skills do you see yourself using in the future?: Networking, networking, and more networking! I won't be focusing on networking, but as I said before I will need networking knowledge at my future job.

This was the first large project I have worked on in Go. I like the language and I plan on using it again for personal projects.

e) What did you like about the project, and what did you not?: Liked

Working on big software projects is always interesting to me, and I'm glad we were working on a real world project with real applications. Go is a great language. Before working on Ciao I had only touched it briefly for a small personal project, working on a large Go project taught me a lot about how it works. I have been interested in continuous integration and cloud management for a couple of years now so working on Ciao was great chance to get hands on development experience with cloud software. Who you are working with can definitely make or break the project. Matthew and Cody were great to work with, and I would definitely team up with them again for other projects.

I wish we had known about the single machine dev environment from the start. It would have made getting going easier, and relieved some of the headaches we had with initial setup. There wasn't anything big I disliked about the project.

f) What did you learn from your teammates?: Matthew was provided great leadership. I learned how to deal with scope change, and stay motivated when the project goes sideways like ours did with the scope change.

Cody has the most Go knowledge. We sat down together and did some pair programming which was very valuable. I would have spent a lot more time on some of the code I wrote without his help. He also has ruthless time management skills. He did an amazing job of keeping track of all of the due dates, and making sure we were on track.

g) If you were the client for this project, would you be satisfied with the work done?: Yes, we had some big scope changes, but those were communicated with our client. Getting Open vSwitch integrated into Ciao was the biggest part of the project and we were largely successful at that.

h) If your project were to be continued next year, what do you think needs to be working on?: We aimed for proof of concept and functionality first which means that there are a few places our code could be cleaned up. Some of the code we wrote is for lack of a better term, hacky. We are interacting with Open vSwitch using exec and the the OVS cli tools. That could be cleaned up and refactored.

We didn't hit all of our stretch goals. If someone else were to pick this up they should work on testing different tunneling protocols, and trying to get DPDK working.

i) Closing Thoughts: I learned a ton on this project. It was very challenging at times, but I'm glad I chose it because it was a unique learning experience. Matthew and Cody were great group mates, and I want to thank them for inviting me to work on the project with them. I also want to thank Rob Nesius and Manohar Castelino, and Intel for sponsoring the project. They were great work work with.

FIN,
Garrett

C. Cody

1) October 14, 2016: This is my first blog post of this exciting senior project. For detailed background information, see Matthew's update. He's done an excellent job writing up how the project came to be.

a) Background: I was approached by Matthew and invited onto the project. It is of great interest to me for a number of reasons: + I am, objectively, a complete greenhorn when it comes to infrastructure systems development. I often find the best way to learn is by finding a project, and grinding my nose against it for a time to learn. + There are few opportunities to work on something as bleeding edge as Ciao. There are even fewer opportunities to get work experience in developing software defined networks. Both of these opportunities appealed to me greatly. + I wanted to learn golang on my own time, and after finding out Ciao was in go, I could hardly refuse.

After accepting, Matthew, Garrett, and I all went up to Intel during the first week of classes and met the team. After a wonderful explanation of the project, and what was expected of us, we set off to start getting the project together.

b) Progress: We've set up the five NUCs, small Intel computers, that will act as our "cloud." Once the project is all setup, we will be simulating a production environment that we can test and deploy our project on. I would like to thank Intel for providing such a wonderful and high performance testing environment.

While visiting Intel, we also got our problem statement and abstract put together and approved by the customer. The team has been incredibly helpful, and efficient at getting us any information we need to get moving on the project.

c) Issues: The only wall that we've hit is getting the networking capability set up on campus. Because of the set up on campus, Network Services prevents deployment of random Ethernet devices. A pragmatic policy to be sure. Knowing all the mischief that college students get up to, I completely understand why this policy is in place. Because of this, we need to get approval from Network Services before our initial deployment of the Ciao infrastructure.

d) Next Steps: Once the network issues are out of the way, we can move forward with our initial testing setup with Ciao. Once that's set up, we can define what the requirements of the project are, and begin designing what exactly needs to be implemented.

Thanks for taking the time to read this, and look for future updates!

Ciao,

– Cody Malick

2) October 21, 2016: This week was a little less action packed than the last. Here's what's happening this week:

a) Progress: We had our problem statement reviewed, and we're waiting on the feedback from Kevin. That should arrive later tonight. We also spent some time getting the five Intel NUCs (Next Unit of Computing) set up in the lab. Matthew spent a good amount of time Wednesday getting them imaged with Clear Linux, Intel's flavor of Linux. We also got the base latex files for the requirements document pushed to a new branch of the docs repo, "client_req." We'll be updating that and merge it once the first version is complete.

We also looked through the basic requirements of the requirements document from IEEE.

b) Issues: Our issues with networking continue through this week. Todd, from the engineering network team, has been incredibly helpful and prompt in his responses. We plan on getting this issue resolved early next week.

c) Next Steps: After the network issues are ironed out, we'll get our test environment set up with Ciao, and get our requirements document written up.

Until next week,

– Cody Malick

3) October 28, 2016: This week we wrapped up fixing the networking and submitted our requirements document.

a) Progress: This week we submitted our finalized and signed copy of the problem statement. With that out of the way, we switched gears into the requirements document. We've worked through a rough draft, and will be fleshing out the details over the weekend. The requirements document has been good to read through, even in it's current form. It's given me a little more insight into what exactly the project is, and how we're going to go about design and implementation. Overall, I'm glad we're doing one.

This week we also got our networking issues resolved! This is great news as it will now allow us to completely configure and update our NUCs as needed without having to be in the lab.

We met with Frank on Tuesday and gave him the update on where we're at currently.

b) Issues: No notable issues this week. We're moving forward at a good clip.

c) Next Steps: We'll be completing the requirements document this coming week, as well as working on configuring the current implementation of Ciao on our five NUCs. We're also expecting some books from Intel that should give us more insight and information into software defined networks.

Until next week,

– Cody Malick

4) November 4, 2016: This week we got our requirements document reviewed and signed by our customer.

a) Progress: This week we submitted our finalized and signed copy of the requirements document. It's been a rather busy week, but our team has been working well together! We laid out a tentative plan for our next steps, which involve the design document and getting the books we need to being working with Software Defined Networks.

We met with Frank on Tuesday and gave him the update on where we're at currently.

b) Issues: No notable issues this week. We're moving forward at a good clip.

c) Next Steps: The next major steps are the tech review documents due next week, and begin work on the design document. We're supposed to be receiving a book from Intel soon as well which should help us get started.

Until next week,

– Cody Malick

5) November 11, 2016: This was a fairly tame week. Here's the update:

a) Progress: We've moved forward on our Tech Review paper. We have the outline detailed, and which person is working on each component. It was a little difficult for us to break up the project into the individual components. The main reason for the difficulty was that we are implementing a small subcomponent of a very specific system. We ended up figuring it out, and we'll all be writing our pieces this weekend.

b) Issues: No notable issues this week.

c) Next Steps: I will be working on the tech review this weekend, and we will have it turned in on Monday.

– Cody Malick

6) November 18, 2016:

a) Progress: It was a busy week! The team managed to get the Tech Review done on time. It ended up being quite a bit of work for each of the components the team was working on. My favorite section of the tech review was covering the network protocols section. I ended up choosing SSNTP, Simple Secure Network Transfer Protocol. The protocol was quite interesting as it was custom tailored to our project's needs. It was created by one of the principal engineers at Intel, who was also the creator of the project. It builds off of the security and handshake protocols that SSL and TLS established, but reduces the protocol overhead by only needing to do the initial handshake once. There were other sections, but they weren't nearly as interesting.

b) Issues: No issues this week, we're moving along very nicely.

c) Next Steps: We've started on the design document for the end of term. We'll be adding to that incrementally in the coming weeks.

– Cody Malick

7) November 25, 2016:

a) Progress: We had a short week this week. We've started on the design document. We will be making the transition over to the template provided by Kirsten for this assignment. In the last few documents, we had not used the exact template she provided. We will be doing so for the final documents this term.

Our team also got together and talked about how we were going to execute the final presentation video for a few minutes this week.

b) Issues: No issues this week. One thing I could use more of, however, is time!

c) Next Steps: With the template setup on the design document, we're ready to move forward with the bulk of the writing.

– Cody Malick

8) December 2, 2016:

a) Progress: This week we wrote the design document up, and got it signed just in time for submission on Friday. Writing the design document was helpful, as it forced the team to read through what libaries and interfaces in those libraries we need to use. As well as getting a somewhat formal design plan in place, it gave us a more informed overview of what peices of the project need to get done.

b) Issues: No issues this week! Everything is moving smoothly.

c) Next Steps: This weekend, the team is going to write the final report, and record the final presentation. After finals, we're planning on getting into the implementation bit of Senior Design before school starts.

See you in Winter Term!

– Cody Malick

9) January 13, 2017:

a) Progress: Over break, Matthew and I got our developer accounts set up on each NUC, and Garrett got his set up this week. We were only able to meet once this week due to the weather, but we laid out a couple things to do to get moving in development. I plan on building a more concrete list of action items that we can complete next week.

b) Issues: We had a show-stopping problem trying to get Ciao set up this week. The specific error and issues are detailed in Garrett's blog post. The issue has since been resolved, and we're moving on to finish the deployment of Ciao.

c) Next Steps:

- Finish Ciao deployment
- Create list of actions to move development forward
- Start development

10) January 20, 2017:

a) Progress: This week was rough for the team, in that we have encountered quite a few issues deploying Ciao. Matthew and I have spent most of our time this week working through configuration changes in Ciao, and while progress has been made, we have not yet successfully completed deployment.

Luckily, Garrett has been making great progress by writing some scripts that analyze network traffic statistics. Some progress has been made! Wohoo!

If we can't get this sorted beginning with next week, we will reach out to Intel and see what help they can provide.

b) Issues: As stated above, we encountered quite a few issues deploying Ciao. Specifically, when Ansible is running, we've run into about five or six major errors that halt production. While many of these have been bypassed through modification of configuration files, we have yet to have success in deployment.

c) Next Steps: We will continue to debug deployment, and start development on Ciao once we have a working test environment.

11) January 27, 2017:

a) Progress: A small amount of progress was made this week, but in a big way. We resolved our issues with ciao deployment! We were able to deploy ciao, and start initial development and network testing. We want to pull network statistics before we make changes, so that we can see what improvements are made by our implementation.

b) Issues: Continuing from last week, we were having quite a few issues deploying ciao. What we ended up finding out was that we were supposed to be running our ansible commands from inside of a docker container. Once that issue was resolved, deployment went relatively smoothly, thanks to Garrett and Matthew's hard work.

c) Next Steps: Next steps are implementing Open vSwitch, and generating GRE tunnels from OVS. We are excited to start development.

12) February 3, 2017:

a) Progress: This week, we started development on implementing the OVS interface into ciao. We're working on understanding libovsdb, the golang library we've chosen to ease development. We've written a couple functions, but have found a need to set up ciao-down, a single environment, development distribution that makes compilation and testing of ciao fast and easy.

I'm going to be working on setting that up over the weekend.

b) Issues: We are still having issues gathering initial network metrics. More on this from Garrett's blog post.

c) Next Steps: I'm working on setting up ciao-down on a development machine, and will be working on implementing OVS GRE tunnels over the weekend. We're aiming to have a basic prototype of this feature working by the end of next week.

Until then,

ciao

13) February 9, 2017:

a) Progress: This week we made progress into the first feature, implementing OVS generated GRE tunnels. Matthew and I pair programmed a couple times this week, but he made quite a bit of progress writing the code for bridge creation.

Outside of working with Matthew, I got a development environment set up for myself on NUC4, and can work on that remotely whenever I need. This is great news, because I hit a lot of road blocks on my personal machines trying to get this set up. I even went so far as to install linux on my laptop, but I was not able to get nested KVM enabled.

Garrett also got a working network test environment working. We should have initial network metrics soon.

b) Issues: I'm happy to say that our issues now are development related, not logistics. The current issue is that we need to find all the places that a bridge creation function call need to be filled in. I created a page to attempt to map out all those locations here

c) Next Steps: This coming week, we are getting our midterm progress report written, and our goal is to get OVS GRE bridges working.

See you next week.

14) February 17, 2017:

a) Progress: This week was focused primarily on writing the midterm progress report. Matthew met with Manohar, the main networking Engineer at Intel who designed Ciao.

b) Issues: No issues this week.

c) Next Steps: Next week we will continue work on implementing OVS into Ciao.

Ciao,

Cody

15) February 24, 2017:

a) Progress: We made some good headway into the bridges this week. I managed to figure out how to run a test on a single file, allowing us to see compilation errors. Before, we were just running the local VM to see if the code works. This is great, and allows us to see what state the code is in before deploying it into Ciao-Down.

We also got our new documents signed and dropped them off as requested.

b) Issues: No issues this week. We're continuing development.

c) Next Steps: Next week we are aiming to ship a working prototype of the OVS bridge.

Cody

16) March 3, 2017:

a) Progress: Unfortunately I was out sick most of the week. Personally, I did not make much progress. On Friday, Matthew and I briefly discussed the OVS management DB, and he's going to fire some questions off to the folks at Intel.

b) Issues: The big issue we've run into is that, from what we can tell, to get OVS working we have to completely overhaul the networking component of Ciao. This is quite the undertaking.

c) Next Steps: Once the questions are answered, we'll know more about what the next steps are.

Cody

17) March 10, 2017:

a) Progress: This was a very good week for progress. After some discussion with the client, we found that it was acceptable to use the CLI (command line interface) for Open vSwitch directly. In contrast to using a third party library, this is straightforward and trivial to call from Go. With this new approach, we were able to replicate quite a bit of the work that had been done thus far in libovsdb with direct command line calls.

Calling the CLI via go is easy. Here's an example of how we call the OVS command line interface:

```
func vsctlCmd(args []string) error {
   if _, err := exec.Command("ovs-vsctl", args...).Output(); err != nil {
      return err
   }

   return nil
}

func createOvsBridge(bridgeId string) error {
   // Example: ovs-vsctl add-br ovs-br1
   args := []string{"add-br", bridgeId}

   // Execute command
   if err := vsctlCmd(args); err != nil {
      return err
   }

   return nil
}
```

While it would be ideal to use an API, this method works well without providing unneccesary overhead to the project. We're implementing all the needed commands in Ciao, and are optomistic about getting a prototype working before end of term. Another advantage of this approach is that getting nvGRE and VxLan working are trivial. All it requires is setting a string in the command arguments.

b) Issues: I've had an issue with my development environment this week. Specifically that when I try to test the code new code, it isn't being properly copied to the test VM via ciao-down. I haven't run into this before, but I presume I'm doing something silly. I'll post an update on this once I have it fixed.

c) Next Steps: Moving into dead week, I'm confident of our ability to get a Ciao prototype working. I will be focusing on adding conditional logic where needed to get Ciao to call the new OVS functions we're written, rather than the GRE functions currently in place. Once this is done, Ciao will support Linux created bridges, as well as OVS created bridges.

Cody

18) March 17, 2017:

a) Progress: This week we worked the poster quite a bit. We got that done and submitted to Kevin and Kirsten. We also made some more progress on integrating OVS into Ciao. Matthew found that we were missing step in setting up the bridge, having the OS allocate an IP address. Using a call to `ifconfig`, the OS will now properly allocate.

We also found that a struct used internally by Ciao, `bridge`, is being used to track the state of the network internally. We were not using this object. We've made a small alteration to the struct by adding a `Mode` attribute. With that in place, we're working on switching over any appropriate areas to properly inform Ciao that a new bridge has been created by OVS.

b) Issues: No issues this week!

c) Next Steps: As we head into finals and spring break, the team will continue working towards getting a working prototype. I feel that we are quite close.

Saint Patrick's Day!

Cody

19) April 7, 2017:

a) Progress: Happy to be back for part three of senior design! Over Spring Break, Garrett and I got together and worked on implementing the rest of the code needed to get our new code using the existing `bridge` structure. After arriving back, the team met up for most of Wednesday for the first meeting of the term and development time.

b) Issues: On Wednesday, we work on development for about five hours. We discovered in this time that Ciao was not installing dependencies as we expected it to. Specifically, OVS was not being installed in the tenant VMs/containers. After thoroughly examining the code, we emailed Manohar for further assistance. We received a response this morning, and are working on getting that OVS installed.

c) Next Steps: As expo is only a few short weeks away, these next two weeks will be a time of intense development for all of us. We are determined to get a prototype working shortly.

Cody

20) April 14, 2017:

a) Progress: Good progress this week. For the first time, we've been able to spin up an instance running with OVS networking. While that was a great success, we've been unable to do anything beyond ping the machine.

On another note, we've moved the NUCs to my apartment to relieve ourselves of networking issues related to OSU's network. With no DHCP restrictions, we can freely spin up VMs and get connectivity to the web. Once we've sorted out the ssh issue with the VMs, we plan on gathering performance metrics with the new setup.

Lastly, we've revised the poster, and gotten a group picture taken for the poster. We'll be finalizing the current draft this weekend.

b) Issues: After doing some searching on what could be causing this issue, we spoke with the Intel team and looked into what the maximum transmission units were set to on the virtual machines. This ended up not being the issue. We're currently still debugging what the issue is.

c) Next Steps: This weekend and this coming week, we're working on getting our poster draft sent off to Intel and Kevin. We'll also be working getting the networking issue evaluated. Overall, a good week.

Cody

21) April 21, 2017:

a) Progress: This week we continued investigations into not being able to ssh into the VMs created. While pingable, they aren't very useful without ssh capability. This week we also finalized the last poster draft, and got approval for branding from Intel. We also met with Kirsten and Kevin separately to discuss expectations of our group at expo.

b) Issues: No new issues this week.

c) Next Steps: We'll be finalizing the poster this week, as well as continuing work on debugging the network issue.

Cody

22) April 28, 2017:

a) Progress: This week we worked on the poster submission, and making changes suggested by the client. We got the poster submitted for review by Kevin and Kirsten.

b) Issues: None

c) Next Steps: Final poster submission on Monday, as well as starting on the Wired-like review.

Cody

23) May 5, 2017:

a) Progress: This week was focused on writing the Wired-like review, and getting the poster submitted. We got the poster submitted for expo on time Sunday.

We also did code cleanup, and emailed Manohar about our current progress, and any suggestions he might have.

b) Issues: No new issues

c) Next Steps: Next week we'll be working on the midterm report, as well as the video submission.

Cody

24) May 12, 2017:

a) Progress: This week we worked on the midterm progress report. We recorded the video Wednesday, and got it edited up the same night.

b) Issues: No new issues

c) Next Steps: This weekend we'll be wrapping up the written report, and preparing for expo next week!

Cody

25) May 19, 2017:

a) Progress: I'm doing this post a bit early because of expo, but hey! Expo is this week. Outside of that, we submitted our midterm report, slides, and video on time.

b) Issues: Edit: My feet hurt after expo

c) Next Steps: Next week we have the last writing assignments to get done before the final. I'm also working with Matthew to return Intel's Nucs to them.

Cody

26) May 26, 2017:

So long, and thanks for all the fish.

It's been quite the journey working on senior design over the past nine months. As a team, we've worked hard, learned a lot, and broadened our horizons in computer science. As this is my last blog post, here are important questions to answer:

a) If you were to redo the project from Fall term, what would you tell yourself?: While we were quite good at researching and planning out the project, I would have told myself to do a proof of concept with Open vSwitch sooner than we did. More importantly, I would have told us to work with the Intel staff to get some help setting up our development environment. Due to the complex nature of developing a cloud orchestration system, we may have saved a couple weeks development time by getting help sooner.

b) What's the biggest skill you've learned?: The biggest skill I've learned on this project is how software defined networking works. By understanding the purpose as well as how it is implemented in a real system, I can take that skill and apply it in future software projects I work on.

c) What skills do you see yourself using in the future?: Outside of SDN, we learned quite a bit about working on a large Golang project. Go has become a language I use on personal projects, and have benefited quite a bit from picking up that skill alone. Other skills are project design (IEEE Design Papers), working with a real customer, and working through rough parts of the project with the customer to get desired results.

d) What did you like about the project, and what did you not?: I thoroughly enjoyed working with Matthew and Garrett, tackling *hard* problems while maintaining forward momentum. I feel that, given the learning curve of the project, that a good team can tackle any problem.

As for dislike, we encountered quite a few setup issues at the beginning of winter term. That was the least exciting portion personally.

e) What did you learn from your teammates?: From Matthew, I learned to stay focused, and never give up as we worked on tough patches of the project. He provided great leadership through never giving up, and keeping a positive attitude.

From Garrett, I learned to consider the project in a broader scope than I usually do. While paired programming together, we worked through the large portions of Ciao's code base, and often Garrett would consider sections of code I did not.

It's been a pleasure working with both of my teammates throughout the year.

f) If you were the client for this project, would you be satisfied with the work done?: I would be. Objectively, we have a bare bones prototype working. While this isn't quite where we wanted to be by the end of the year, we were able to learn the code base, the related technologies, the design paradigm, and produce a basic product.

g) If your project were to be continued next year, what do you think needs to be working on?: We implemented a somewhat hacky version of the prototype. We wrote the client by shelling out to command line to utilize Open vSwitch's command line utility, `ovs-vsctl`. While this is a workable solution, it is not optimal. Ideally, we could use a third party API and directly interface with the OVS management database. Next steps would be to research that approach, and replace our current implementation with that setup.

Lastly, I would like to thank Rob and Manohar again for sponsoring and supporting the project. It has been a memorable experience, and I have grown immensely as software engineer because of it.

Ciao,

Cody

VIII. Poster

Cloud Orchestration Networking

Increasing server-to-server speed, and increased scalability

Project Description

What is Ciao?

Ciao (Cloud Integrated Advanced Orchestrator) is a cloud orchestration system developed by Intel. A cloud orchestration system manages the allocation and use of resources in a cloud environment, such as compute cycles, networks, and storage. Ciao provides an easy to deploy, secure, scalable system which handles virtual machines (VMs), containers, and bare metals apps as generic workloads.

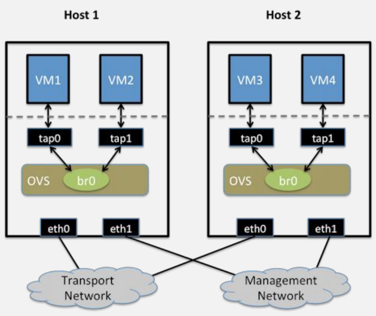
The Problem

Ciao uses standard Linux networking tools and network objects managed by the Linux kernel to create and manage software defined networks for VMs . The current implementation is not compatible with new areas of innovation in networking technologies.

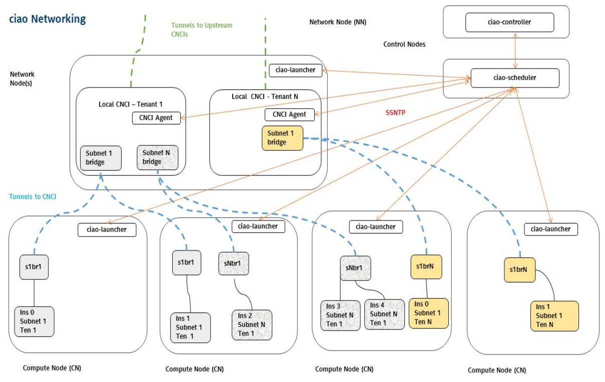
Proposed Solution

Our proposed solution is to explore implementing network creation and management in Ciao using Open vSwitch (OVS). OVS is a software implementation of a multilayer network switch that supports many standard network interfaces and protocols, with Virtual Extensible LAN (VxLAN) and Network Virtualization using Generic Routing Encapsulation (NVGRE) being the technologies of primary interest. With these technologies, OVS is used to dynamically create bridges and tunnel endpoints between nodes in the cluster being managed by Ciao.

Open vSwitch Tunneling Diagram



Source: http://docs.openvswitch.org/en/latest/_images/tunneling.png



Source: https://github.com/01org/ciao/blob/master/networking/documentation/ciao-networking.png

Findings and Results

The objective of the project was to explore full integration of the Open vSwitch (OVS) virtual switch platform into Ciao. These were our findings:

- Proof of concept of communication between two virtual machines using an OVS bridge on the host is successful.
- Open vSwitch does not recognize networking interfaces, bridges, or tunnel endpoints managed by the Linux kernel. Each of these must be created individually by OVS.
- A full replacement of tunnel endpoints and Linux bridges with Open vSwitch bridges and endpoints was required for Ciao integration.
- When the OVS network mode is selected via user configuration, the Ciao SDN will use OVS bridges and tunnel endpoints for controller and compute node network connections.
- Switching to innovative tunneling protocols such as VxLAN and NVGRE is as simple as a single parameter change with Open vSwitch tunnel creation.

Technical Details

Ciao Software Defined Network

- Simpler Software Defined Network (SDN) implementation. An SDN is a network layer defined in software rather than on physical switches and routers.
- Targets small or medium enterprise datacenters with several hundred servers
- Creates self-configuring, stateless, software defined overlay networks for tenants on top of existing networking hardware
- Uses Linux bridges and GRE tunnels to create overlays
- Problems:
  - Current SDN innovation is being done with Open vSwitch
  - Linux bridges and GRE tunnels used by Ciao are not compatible with Open vSwitch

The Team

Project Members and Sponsor

Team Members

Garrett Smith, Computer Science
smithgar@oregonstate.edu

Cody Malick, Computer Science
malickc@oregonstate.edu

Matthew Johnson, Computer Science
johnsma8@oregonstate.edu

Project Sponsor

Robert Nesius, Engineering Manager, OTC Advanced Systems Engineering, Intel Corporation

Special thanks to Intel Principal Engineer Manohar Castelino

IX. Project Documentation

A. Project Overview

Ciao is a cloud orchestrator, and requires a basic setup of five machines: a deployment node, a control node running Ciao, a network node, and two compute nodes. Because developing software on a cluster can be difficult, we use Ciao-Down, a CLI utility for single machine development.

B. Installation Instructions

The simplest way to run Ciao is in single VM mode with ciao-down. We had issues running Ciao in a multiple machine environment due to DHCP issues, but you are welcome to give it a try. The main issue we were seeing was caused by the OSU network. Ciao uses the external DCHP server to assign IP addresses to the instances it spins up. These instances have generated MAC addresses. The OSU network will only lease an IP address if the MAC address is registered with the university IT department. Our problem was that since the generated MAC addresses were not registered with the university the instances were never assigned IP addresses and were therefore inaccessible. If you want to run Ciao in a cluster Intel provides very detailed instructions. Manual deployment instructions can be found at https://clearlinux.org/documentation/ciao/cluster-setup.html. Instructions for deployment via an Ansible playbook can be found at https://clearlinux.org/documentation/ciao/deployment/deployment.html.

C. Running Ciao-Down

- Download the master version of Ciao by running `go get github.com/01org/ciao`
- Change directories into the Ciao git repo
- Add our Ciao repo as a remote with `git remote set-url origin git@github.com:matthewrsj/ciao.git`
- `Run git fetch` to get our changes
- Checkout the ovsdev branch with `git checkout ovsdev`
- Build ciao-down with `go get github.com/01org/ciao/testutil/ciao-down` and `$GOPATH/bin/ciao-down prepare`
- Connect to the ciao-down VM with `$GOPATH/bin/ciao-down connect`
- When you connect to the VM it will provide you with instructions to finish setting up your environment and run a workload.
- To power off the VM run `ciao-down stop`
- To remove the VM run `ciao-down destroy`

D. Project Software Requirements

- CPU that supports nested KVM. You must have nested KVM enabled as well.
- Ubuntu 16+, Arch Linux, Fedora. Other distributions may work but are not tested.
- Golang 1.8 or newer.
- Ciao-down requires several other binaries. If you are missing any ciao-down will prompt you to install them.

E. Additional References

X. "How did you learn new technology"

Due to the nature of the project, one of the best ways to learn was to dive in. While we were very hands on, the Ciao documentation was critical to learning and understanding the code base.

Open vSwitch was another challenging technology to pick up. Software defined networking is a highly technical subject. There were not a lot of specific websites that provide thorough examples and documentation on the technology. We did gain insight into OVS by reading the original IEEE RFC 7047 for the management database[2].

The official Golang documentation was another critical source of information for us. Without it, we wouldn't have been able to write the code that we did. It's available online at https://golang.org/ref/spec.

Last, but certainly not least, Manohar Castelino was a huge source of domain knowledge in relation to Ciao. A principle engineer and primary author of Ciao networking, Manohar was extremely helpful in helping us over difficult hurdles this year.

XI. "What did you learn?"

A. Matthew

At the beginning of the project I had a vary basic understanding of software defined networking, cloud orchestration or networking in general. Although I understood the concepts, I had no experience with the actual implementation details of any of the technologies we worked with.

In a general sense, I learned a lot about how software defined networking actually works in practice and specifically how Ciao's implementation works. Cloud orchestration is a complicated problem that Ciao is approaching in an innovative way.

To be more specific, I obviously learned a ton about Open vSwitch and other new and innovative SDN technologies such as VXLAN and NVGRE tunneling protocols. I also learned a lot about the Go programming language and learned to appreciate its error handling and overall philosophy. Some technical experience I gained that was indirectly related to our project was a better understanding of the various Linux networking tools available that we used to debug the issues we saw during project.

I also gained some non-technical skills during my work on this project. I learned to be more accepting of scope changes due to the major scope change we had during Winter Term. I also learned to never trust my assumptions about how something should work, because most likely those assumptions will be wrong. Connecting the two, I learned that incorrect assumptions can lead to major scope changes in your project if not careful.

About project work, I've learned that not everyone has the same understanding of the technical aspects of the project. For example, I had a much lower understanding of advanced networking than Garrett and a less complete grasp of the Go programming language. On the other hand, because I read through the Ciao code more closely I understand the inner-workings of Ciao better than either of them. When working on a project together this information must be freely shared in order to achieve a successful collaboration.

Project management and leadership is another area where everyone in the group gained experience. We all worked well together and different members took responsibility for different aspects of the project. We learned to rely on each other to stay on task and focused.

One thing that really helped us keep our work progressing was agreeing on a specific time to meet every week. Everyone on the team took it very seriously and showed up to all the team meetings. This kept everyone accountable for work being done and kept us motivated every week.

If I could do the project all over again, I would mistrust assumptions from the start of the project. This would have saved us several weeks of work before the scope change. Additionally, we tried to not bother our clients with technical questions too much since this was our project to complete. I did not want it to seem like we were relying on our very technical clients (one of them a Principal Engineer at Intel) to do our work for us. In hindsight it might have been better to ask those technical questions so they could at least point us in the right direction. While we did ask several technical questions throughout the year, asking more might have saved us some troubleshooting time.

B. Garrett

This project was very technical and as a result I received experience with a variety of tools, technologies, and technical concepts. Software define networking was the main technical point of the project, but along the way we worked with Go, Open vSwitch, general Linux networking utilities, network tunneling protocols, virtual machines, docker, and Ciao. The hands on experience I received with these tools and technologies is amazing.

To start with, setting up Ciao was great for learning how clouds are created and managed. To deploy Ciao to a cloud you as the user only have to install the Ciao software on one machine in your cloud. Ciao will take care of finding the other machines you want it deployed to, and installing its self. You set up your servers with a user and a key that can be used to SSH into the server , and give Ciao the user names and keys via configuration. It takes care of installing all of the components it needs to manage the machines. This makes a ton of sense looking back on it, but my initial thought was that you would need to set up a client on each machine in the cloud, and have it report back to the Ciao control node you are using to manage the cloud. This is a clever system, and one that makes a lot of sense once I saw how it works, but not one that was initially intuitive to me.

Prior to this project I had used Go for a couple of toy personal projects, but nothing serious. Working on Ciao gave me actual experience with Go in a large scale project. I learned how large Go projects are structures, and how the different packages are set up. The project also gave me practice code in Go which is the best way to get comfortable with a language.

Before we could replace the Linux bridges, and Linux created GRE tunnels that Ciao was using we had to understand what it was doing. This meant learning how Linux bridges work, and how to create GRE tunnels with the basic Linux utilities. Once we figured this out we where able determine what the networking sections of Ciao were doing, and where we needed to make our changes. That process taught me a lot about using Linux networking utilities such as `ip`.

Another important piece of technology I learned about was Open vSwitch. We had to know how to use it to integrate it into Ciao. I spent a few days running through some demos, and figuring out how to set up bridges and tunnels with Open vSwitch. I feel comfortable using it now, and I would consider using it in other projects if I needed a software switch.

On the non technical front I improved my communications skills because I had to be in regular contact with my team to keep them up to date on what I was working on, any issues I ran into, and my progress.

Project work is very different from school work. I learned just how much planning goes into a project before any work can start, and how structured it is. Working on the Capstone project involved a very constant stream of work which is different from traditional school projects which are short bursts of work. Overall I enjoyed working on the project. It was long enough lived that I felt invested int it, and I'm proud of my accomplishments which is not something I get very much from school work.

This was not my first time working on a team or on a large project, but it was my first time working on a project of this scale where I was one of the people in charge. There is a lot of planning involved, and you have to be flexible when something unexpected occurs. We had a large scope change and it threw our initial plan off track. Working around issues as they come up, and updating your plan if you need to is really important when managing a project.

Having a good team can make or break a project. We had a hard project, but we had an amazing team. Cody and Matthew were great to work with on a number of levels. We all stayed in frequent communication via group text chat, and met up regularly. This made it easy to keep up with who was working on what, and their progress. A structured schedule and regular communication make working in a group a lot easier.

If I were doing this all over again I would take more time to prototype and figure out exactly what technical questions we need to answer, and what issues we need to resolve before diving in. We had a scope change because we had to do a full replacement of the Linux bridges in Ciao with Open vSwitch instances which was a surprise part way through Winter term. I would try to prepare for that. In a more generalized sense I would try to figure out what assumptions we were making and figure out if they are correct as soon as possible.

C. Cody

This project encompassed a few major areas that I had no previous experience with. From a technical perspective, cloud orchestration and software defined networking were two big knowledge gaps for me. Also, as a side effect of learning SDN, I learned quite a bit about networking in general. While being familiar with the Go programming language, commonly referred to as Golang, I improved as a Go developer significantly this year.

“Cloud orchestration” is a bit of a buzzword. Going into the project, I had had no prior experience in the area. Jumping from no knowledge to working on the code base of a cloud orchestrator was quite a bit of fun. While we didn’t have to touch the algorithm that operated Ciao, we had to understand the fundamentally structure of the software we were altering. I picked up quite a bit of information reading through the Ciao docs, and through reading articles about basic goals of Ciao.

Learning software defined networking, as a topic, was pivotal to implementing the project. Ciao uses software defined networking to track the state of the network internally. With this capability, it can spin up VMs or containers at will, and allow the user to not worry about which server they run on. Understanding how SDN’s abstract the physical layer was interesting and fulfilling to learn. As a side effect of having to learn SDN, my general understanding of networking improved substantially.

Go has quickly become one of my favorite languages. It enforces simplicity and performance, while providing fantastic multi-threading and concurrency constructs. As our resident Golang expert, I had to convey knowledge to Garrett and Matthew as we moved through development. The best way of learning something is by explaining it to others!

From a non-technical standpoint, my ability to plan and design projects in general have improved over the year. While many did not enjoy the documentation portion of the project, I felt I grew significantly as, not only an engineer, but an architect.

Another area I felt greatly improved was public speaking. While there was a limited amount of it this year, standing up and explaining our project to people of different backgrounds and cultures was fun and challenging. Through explanation of the project to others, I gained a better understanding of our own project and how it fit into the bigger picture.

Working on projects is not a new experience to me, but I learned quite a bit about design this year. Rather than just jumping in, figuring out exactly what we wanted to do helped us in the long run. While we had a large scope change, we were able to use most of our original design to execute the project. Each member of our team took individual planning and execution rolls. One of the big keys to our success this year is that we planned out our weekly meeting times. More importantly, we stuck to those times. Every member of the group showed up and executed. While I feel that we spent more time planning than we needed to, it paid off significantly at the end of the year. We were much more relaxed when we knew exactly what was going to happen.

From a group work perspective, I found out what it meant to be part of a team that has synergy. In the past, I’ve worked with teams that worked well together, and others that did not. Matthew and Garrett were a pleasure to work with, and I have greatly enjoyed dividing class work, and tackling problems together. Overall, I think that the people you work with is far more important that the subject you’re working on.

Overall, I feel we did extremely well. If I were to do it over again, I would spend more time getting familiar with our code base in fall. Quite a few of our original roadblocks resulted from unfamiliarity with how Ciao is developed, and what tools were used in that development process.

XII. Conclusion

This year has been challenging, but a large growing experience for our team. Working on Ciao has given each of our team a chance to learn, from new technologies to design work. We would like to thank Kevin, Kirsten, and Intel for their support this year. We are satisfied with the work we've done this year, and are looking forward to our commencing our work in software engineering.

References

[1] Socketplane. (2016, dec) libovsdb. [Online]. Available: https://github.com/socketplane/libovsdb/blob/master/README.md

[2] E. B. Pfaff, B. David. (2013, dec) The open vswitch database management protocol. [Online]. Available: https://tools.ietf.org/html/rfc7047

XIII. Appendix A

The following listings illustrate bridge creation with OVS. A similar method is used for tunnel endpoint creation.

Listing 1. Calling the OVS command tool

```
func vsctlCmd(args []string) error {
   _, err := exec.Command("ovs-vsctl", args...).Output()

   if err != nil {
      glog.Error("vsctlCmd failed: " + err.Error())
      return err
   }

   return nil
}
```

Listing 2. Creating an OVS bridge

```
func createOvsBridge(bridgeId string) error {
   // Example: ovs-vsctl add-br ovs-br1
   args := []string{"add-br", bridgeId, "--", "set", "bridge", bridgeId,
      "datapath_type=netdev"}

   // Execute command
   if err := vsctlCmd(args); err != nil {
      return err
   }
   return nil
}
```

Listing 3. Destroying an OVS bridge

```
func destroyOvsBridge(bridgeId string) error {
   // Example: ovs-vsctl del-br ovs-br1
   args := []string{"del-br", bridgeId}

   if err := vsctlCmd(args); err != nil {
      return err
   }

   return nil
}
```

The following code listing is an example of how network mode switching was handled in Ciao.

Listing 4. Network mode switching

```
switch b.Mode {
case GreTunnel:
   bridge := &netlink.Bridge{LinkAttrs: netlink.LinkAttrs{Name: b.LinkName}}
   if err := netlink.LinkAdd(bridge); err != nil {
      return netError(b, "create link add %v %v", b.GlobalID, err)
   }

   link, err := netlink.LinkByName(b.LinkName)
   if err != nil {
      return netError(b, "create LinkByName %v %v", b.GlobalID, err)
   }

   brl, ok := link.(*netlink.Bridge)
   if !ok {
      return netError(b, "create incorrect interface type %v %v", b.GlobalID, link)
   }

   b.Link = brl
   if err := b.setAlias(b.GlobalID); err != nil {
      err1 := b.Destroy()
      return netError(b, "create set alias [%v] [%v]", err, err1)
```

```
    }

    return nil
case OvsGreTunnel:
    if err := createOvsBridge(b.GlobalID); err != nil {
        return err
    }

    return nil
default:
    return netError(b, "Create Bridge: unknown network mode %v, bridge %v", b.Mode,
        b.GlobalID)
}
```

This final listing shows the data structure used to store information on the bridge object.

Listing 5. Ciao bridge object and supplementary OvsBridge object

```
// Bridge represents a ciao Bridge
type Bridge struct {
    Attrs
    Link *netlink.Bridge
}

// OvsBridge represents a ciao bridge created via ovs-vsctl
type OvsBridge struct {
    Attrs
}
```

XIV. Appendix B

Fig. 1. The Golang Gopher