CORD Reference Implementation

Ali Al-Shabibi, Andy Bavier, and Larry Peterson
Open Networking Lab

May 6, 2016

Introduction

CORD is a reference implementation of a cloud-based service delivery platform for network operators. A detailed case for CORD is laid out in a companion document [1], but in sum, it is designed to provide cloud economies by leveraging infrastructure constructed from a few commodity building blocks, and cloud agility by supporting the ability to rapidly deploy and elastically scale services.

CORD is a reference implementation, and as such (1) it is built from open source software components; (2) it is a fully integrated system that is sufficiently complete to support field trials; and (3) it exercises the full breadth of capabilities it is designed to support.

This note documents CORD's requirements and identifies the design decisions and technology choices that have shaped the reference implementation. As there is not yet an official release of CORD, this note should be read as preliminary.

Requirements

CORD has the following five requirements. The first three follow directly from the goal of of bringing cloud economies and agility to the Telco setting. The last two are about applying best practices in system design to this specific domain.

Economies of Commodity Hardware

CORD must run on commodity servers and white-box switches, and to the extent possible, leverage merchant silicon; it should not depend on proprietary or purpose-built

hardware to achieve high performance. That CORD leverages commodity hardware follows directly from its goal of supporting the economics of cloud infrastructure. And by implication, the software running on that commodity hardware must deliver the same performance and reliability as today's purpose-built hardware.

> *To this end, CORD provides mechanisms to virtualize commodity servers and control white-box switches, along with several open source exemplars of how software can leverage these mechanisms to achieve high performance and reliability.*

Enable Innovative Services

CORD must support a wide-range of services; it is not limited to access services, nor should it unnecessarily constrain how services are implemented. Specifically, CORD must support services drawn from across the following four dimensions: (1) both access services (e.g., Fiber-to-the-Home) and conventional cloud services (SaaS); (2) both services implemented in the data plane (NFV) and services implemented in the control plane (SDN); (3) both trusted operator-provided services and untrusted third-party services; and (4) both bundled legacy services and disaggregated greenfield services. Moreover, CORD must provide an agile framework for managing a set of services without regard for how the individual services achieve isolation, scale, performance, and high availability.

> *To this end, CORD defines a unifying service abstraction (and other supportive models) that encompasses the full range of service designs, along with mechanisms that map those abstractions onto virtualized commodity hardware.*

Extensible and Controllable

CORD is a configurable platform; it is not a point-solution. It must provide a means for the operator to specify the desired portfolio of services and the dependencies among those services. This includes the ability to configure CORD for different markets and access technologies: Residential, Enterprise, and Mobile. It must also provide a means to provision and parameterize those services to meet the operator's operational and business objectives.

> *To this end, CORD defines a Northbound Interface (NBI) that allows operators to configure, control, and extend a CORD deployment, and does so in a way that cleanly separates policy and mechanism.*

Multi-Domain Security

CORD must account for multiple domains of trust; it is not sufficient to only distinguish between CORD operators and CORD users. This includes a wide range of intermediate roles, including global operators, site-specific operators, service operators, service developers, service tenants (other internal services) and service subscribers (external principals).

> *To this end, CORD adopts best practices in secure system design: it minimizes the trusted code base, it provides mechanisms to mediate trust, and it provides mechanisms that support the principle of least privilege.*

Operational Robustness

CORD must account for partial and intermediate failure scenarios; it is not acceptable to ignore the operational realities of building a system by integrating multiple, independently developed software components. Without holding the reference implementation to the high bar of "production ready," CORD must be architected to account for the possibility that the operational behavior of the system is not always in sync with the desired state of the system.

> *To this end, CORD employs best practices in scalable cloud services: to treat error states as expected in a production system, and to provide built-in mechanisms to automatically recover from them without service disruption.*

Design and Technology Decisions

As a reference implementation, CORD goes beyond defining an abstract architecture to also making specific design decisions and technology choices. These decisions are subject to change, and individual components can be replaced with alternatives.

As an extensible platform, it is not practical to demonstrate all possible services and access technologies in a single configuration. The reference implementation focuses on a subset of services suitable for a deployment serving residential customers. We expect to add other services—and other reference configurations—over time, including M-CORD (a configuration of CORD suitable for a Mobile use case) and E-CORD (a configuration of CORD suitable for an Enterprise use case).

Two factors influence our technology decisions. The first is whether or not an individual component bakes in policy decisions that should be left to orchestration software layered on top of CORD. The goal is to cleanly separate policy (specified by the operator) from mechanism (implemented by CORD). The elements integrated into

CORD must be consistent with that goal. The second is to prioritize forward-looking design over backward compatibility. Specifically, CORD makes no effort to accommodate legacy hardware, which frees the design from constraints that might otherwise compromise CORD's ability to achieve its goals.

The rest of this section is organized around CORD's hierarchical structure, but it serves as little more than an project roadmap. Each of the elements introduced below is described in more detail in a companion Design Note..

Hardware – CORD POD

CORD runs on a collection of commodity servers and white-box switches, coupled with a "disaggregated packaging" of media access technologies like GPON. These hardware elements are then organized into a rackable unit—called a POD—that is suitable for deployment in a Telco Central Office. We have settled on a particular configuration for the reference CORD POD, but alternative configurations and sizings are also possible.

Commodity Elements

The reference CORD POD consists of three hardware elements:

- **Servers –** We have selected the Open Compute Project (OCP) qualified QUANTA STRATOS-S210-X12RS-IU server. Each server is configured with 128GB of RAM, 2x300GB HDDs, and a 40GE dual port NIC.
- **Switches –** We have selected the OCP-qualified and OpenFlow-enabled Accton 6712 switch. Each switch is configured with 32x40GE ports, and can double as both leaf and spine switches in the CORD fabric.
- **I/O Blades –** Celestica is the ODM for "OLT pizza box" that includes the PMC Sierra OLT MAC chip. This is a 1u-blade that includes 48x2.5Gbps GPON interfaces and 6x40GE uplinks.

The servers run Ubuntu LTS 14.04.3, and include Open vSwitch (OvS 2.3.0). The switches are based on ONF's Atrium software stack, which includes Open Network Linux, the Indigo OpenFlow Agent (OF 1.3), and the OpenFlow Data Plane Abstraction (OF-DPA), layered on top of Broadcom merchant silicon.

Although not included in the initial reference POD, I/O blades for 10G-PON and G.Fast are planned for the near future.

Configuration and Sizing

The reference CORD POD is sized to stress CORD's functionality and performance, as well as to serve in an AT&T field trial. Both larger and smaller configurations are also possible. The assembly instructions for the reference POD are given in a later section, but in summary:

- It is organized as two "virtual racks" each with two leaf/ToR switches and a set of servers/blades.
- It includes six Accton switches arranged in a leaf-spine configuration, with four serving as leaf (ToR) switches (two per virtual rack) and two serving as spine switches. The leaf switches use 24 ports as downlinks to the rack's servers and 8 ports as uplinks to the spine, while the spine switches use all 32 ports as downlinks to the leaf switches.
- It includes two Celestica OLT pizza boxes. All the I/O blades are co-located in the first "virtual rack" and connected to the corresponding pair of leaf switches.
- It includes six Quanta servers, with three serving as "head" nodes and three serving as "compute" nodes. The compute servers are spread across the two "virtual racks" and connected to the corresponding pair of leaf switches.

Note that the selected leaf switches have sufficient capacity to support up to 24 dual-port servers and the spine switches have sufficient capacity to support up to 16 racks.

Software – CORD Distribution

CORD is a system built by integrating other open source projects, each of which has been configured with its own set of parameters, model definitions, and software plugins. These software components collectively define CORD's extensible framework, on top of which the reference implementation includes a particular service portfolio.

Core Framework

CORD's core software framework is built from a combination of ONOS, XOS, OpenStack, and Docker (in addition to the server- and switch-specific software described above), as illustrated in Figure 1.

- **OpenStack** provides a core IaaS capability, and is responsible for creating and provisioning virtual machines (VMs) and virtual networks (VNs). CORD uses OpenStack's Nova, Neutron, Keystone, and Glance subsystems.

- **Docker** provides a container-based means to deploy and interconnect services. It also plays a key role in configuring and deploying CORD itself (e.g., the other elements—XOS, OpenStack, and ONOS—are instantiated in Docker containers on the POD head nodes).
- **ONOS** is the network operating system that manages the underlying white-box switching fabric, as well as the software switches (OvS) running in each server. It hosts a set of control applications that implement services and it is responsible for embedding virtual networks in the underlying network.
- **XOS** is a framework for assembling and composing services. It unifies data plane services supported by OpenStack and Docker, and the control plane services running on ONOS. XOS is an integrated system that defines a data model (built using Django) and a controller framework for backend components (written in Python with extensive use of Ansible).

There are three notable design decisions represented in these implementation choices. First, in support of the widest possible collection of services, the reference implementation supports services running in virtual machines (KVM), in containers running directly on bare metal (Docker), and in containers nested inside virtual machines (Docker in KVM).

Second, ONOS plays two roles in the reference implementation. The first is to manage the switching fabric and implement the virtual networks required by the rest of the framework. This functionality is implemented by a pair of ONOS applications (called Segment Routing and VTN, respectively), and it is accessed indirectly through OpenStack's Neutron API. The second is to provide a platform for hosting control programs that implement first-class CORD services. Two examples are given in the next subsection (vOLT and vRouter), both of which access ONOS using its *Flow Objectives* NBI [2].

Third, XOS effectively serves as the controller for CORD as a whole, and as such, it defines CORD's NBI. There is both a RESTful version of this interface and a model-based interface based on TOSCA (YAML). CORD also includes a collection of GUI-based views (e.g., one for network operators, one for third-party service developers, and one for residential subscribers).

Service Portfolio

CORD is designed to be extensible, with network operators specifying the service graph they wish to deploy in a given environment [3,4]. The reference implementation includes

the following set of services, the first three of which (vOLT, vSG, vRouter)[1] are targeted at Residential subscribers. We expect this set to grow over time, including the addition of services in support of Mobile and Enterprise customers (we refer to these as the M-CORD and E-CORD configurations, respectively).

- **vOLT –** Provides a *Subscriber VLAN* that connects residential customers to CORD. Implemented by a control application running on ONOS, and uses RADIUS for authentication [5].
- **vSG –** Provides a *Subscriber Bundle* that includes a set of consumer features (e.g., parental control, firewall). Implemented as a Docker container running directly on bare metal, with features implemented using various Ubuntu mechanisms (e.g., dnsmasq, iptables). Exploration of alternative ways to implement subscriber features is ongoing [6].
- **vRouter –** Provides a Routable Subnet by which a subscriber's container connects to the public Internet. Implemented by a control application running on ONOS, and uses Quagga to peer with other routers [7].
- **Ceilometer –** Provides a Monitoring Channel that can be queried for meters reported by various hardware and software resources. Implemented by a combination of OpenStack subsystems running in VMs and Docker containers [8].

Note that the reference implementation uses a proprietary CDN as an example third-party service. Not surprisingly, source code for this proprietary service cannot be included in the open source release. Our plan is to include a list of "CORD compliant" third-party services with each release, leaving it to organizations wanting to evaluate CORD in lab and field trials to separately negotiate an "evaluation license" for such services with the corresponding third-party service vendor.

POD Assembly

This section sketches the POD assembly instructions. More detailed information is available in the github repositories referenced below.

Hardware

The servers, switches, and OLT I/O blades in a canonical POD are assembled into two virtual racks (single physical rack) as illustrated in Figure 1. The figure is incomplete in

[1] In earlier implementations, vSG (virtual Subscriber Gateway) was called vCPE and vRouter was called vBNG.

that it does not show that Leaf-2 is connected to an upstream (legacy) router and the OLT blades are connected via GPON to residential gateways. Figure 1 is also limited in that is shows only the 3x40Gbps uplink from the each OLT blade to Leaf-1, but there is a parallel connection to Leaf-3. However, this second path is active only when the first fails; the two paths cannot be used simultaneously.

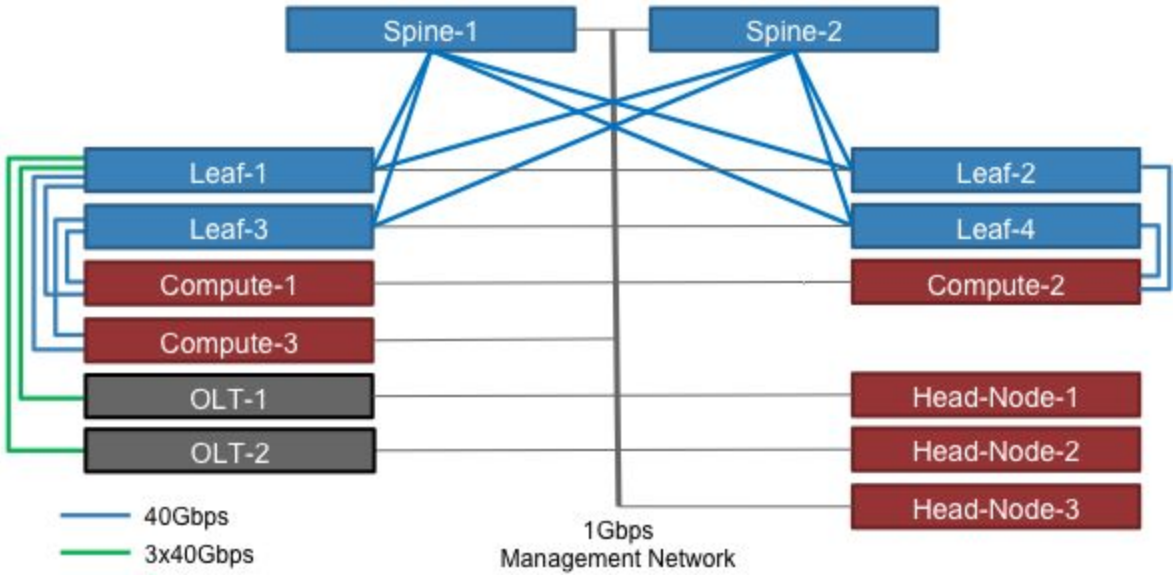

Figure 1. POD Hardware Configuration.

As depicted in Figure 1, the POD includes a global management network that connects all the servers and fabric switches to the control processes running on the CORD head nodes (it is implemented by a physical switch that is distinct from the white-box switches that implement CORD's data plane). There is also a local (per-server) management network that logically interconnects the instances running on that server, but these local management networks are not interconnected with each other or to the global physical management network. Instead, certain management functions are implemented by proxies or application bridges that are connected to both the global management network and each local (per-server) management network. For example, an ssh proxy is used to log into an instance for management purposes.

Software

Installing software on a CORD POD is a four-step process:

1. Install OpenStack.
2. Install ONOS-CORD and configure OvS on the nova-compute nodes to be controlled by ONOS-CORD (ONOS-CORD hosts the VTN and vOLT apps).

3. Install ONOS-Fabric and configure the white-box switches to be controlled by ONOS-Fabric (ONOS-Fabric hosts the Segment Routing and vRouter apps).

4. Install XOS and instantiate the CORD service graph.

The software installation process is still under active development, with the most recent procedures documented in the XOS repository on github:

https://github.com/open-cloud/xos/tree/master/xos/configurations/cord-pod

Instructions are included for building both a full POD and a “CORD-in-a-Box” development environment.

Infrastructure supporting Continuous Integration (CI) / Continuous Delivery (CD) is also under construction at https://gerrit.opencord.org.

Summary

The CORD reference implementation is currently undergoing active development, with the goal of having an official public release in mid-2016. Early drafts of the cited design documents will be posted to the CORD site (opencord.org) as they become available.

References

[1] Central Office Re-architected as a Datacenter (CORD). *CORD Design Notes* (June 3, 2015).

[2] CORD Fabric, Overlay Virtualization, and Service Composition. *CORD Design Notes* (March 2016).

[3] Service Assembly and Composition in CORD. *CORD Design Notes* (February 2016).

[4] Security in CORD. *CORD Design Notes* (March 2016).

[5] Virtual OLT (vOLT). *CORD Design Notes* (March 2016).

[6] Virtual Subscriber Gateway (vSG). *CORD Design Notes* (March 2016).

[7] Virtual Router (vRouter). *CORD Design Notes* (March 2016).

[8] CORD Monitoring Service. *CORD Design Notes* (November 17, 2015)