Topic 5: Cloud/Data Center Architecture:
Data Center Cluster Topology
Data Center Networking (L2 Semantics, ECMP LB)

Dec. 2, 2021

Y. Richard Yang
Computer Science Department, Yale University
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Final Project planning
  - Dec. 3: Checkpoint (2-page update)
  - Dec. 22: Due (report ~6 pages)
Cooperation and fault-tolerance through a reliable data store
- Different functions coordinate through the data store to divide tasks
- Replicas of same function coordinate for fault-tolerance

%kubectl -n kube-system get all
Recap: Controller-Manager+Scheduler+Etcd Work Flow

- Different components get different tasks by watching different labels/keys
- The controller-manager does not need to know who the scheduler is, who the node controllers (kubelet) are.
Recap: Etcd API
Recap: Raft Consistent, Replicated Log Protocol

- Replication and leader election to improve availability
- A combination of election rule, commitment rule, and update-follower rule to achieve consistency
Thinking Exercise: Microservices Control Architecture => 5GC Control Architecture
Thinking Exercise: Google Orion Network Control Architecture
Alternative Design: OpenDaylight Network OS

Offline: OpenDaylight Network OS Software Architecture
Offline: OpenDaylight Data Model Details

- Whole data store as a tree
- Two Logical Data Stores
  - config
  - operational
- InstanceIdentifier:
  - pointer to a node
(Offline) OpenDaylight: Data Operations Details

Operations must be included in transactions

```java
ReadWriteTransaction transaction =
dataBroker.newReadWriteTransaction();
Optional<Node> nodeOptional;
nodeOptional = transaction.read(
    LogicalDataStore.OPERATIONAL,
    n1InstanceIdentifier);
transaction.put(
    LogicalDataStore.CONFIG,
    n2InstanceIdentifier,
    topologyNodeBuilder.build());
transaction.delete(
    LogicalDataStore.CONFIG,
    n3InstanceIdentifier);
CheckedFuture future;
future = transaction.submit();
```

Datastore

network-topo

BGPv4

overlay1

nodes

nodes

n1

n2

n3

Operations must be included in transactions
Offline: ONOS System Architecture

Northbound - Application Intent Framework
(policy enforcement, conflict resolution)

Distributed Core
(scalability, availability, performance, persistence)

Southbound
(discover, observe, program, configure)

OpenFlow
(pluggable, extensible)

More API in future release

Source: ON.LAB
Offline: ONOS Software Architecture

The diagram illustrates the ONOS Software Architecture. It consists of several layers:

1. **Network Devices**: At the bottom, representing the network devices that are controlled by the software.
2. **Service Adaptation Layer**: This layer includes REST APIs and NETCONF, used for communication and configuration.
3. **Service Functions**: Includes components like Statistics Manager, Forwarding Rules Manager, and PCEP Exporter.
4. **Base Network Functions**: Such as Inventory Manager and Topology Exporter.
5. **Applications**: The top layer, where various applications can be built. 

The diagram also shows interfaces and protocols like OpenFlow 1.0/1.3, BGP-LS, PCEP, Netconf Client, OVSDB, and LISP. The Controller Platform and Southbound Interfaces & Protocol Plugins are also highlighted, indicating where the software interacts with the network and other components.
Microservices Pod Construction (Structure)

- **Initialize Pause network namespace**
- **Join Pause network ns**
- **Init Pause cont. net device**
- **Cross-node networking**
Microservices Pod Structure

- Demo (MacOS)
  - `kubectl get pod`
  - `docker ps`

```c
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define STRINGIFY(x) #x
#define VERSION_STRING(x) STRINGIFY(x)

#ifndef VERSION
#define VERSION "0.0.0"
#endif

static void sigsew(int signo) {
  perror(signame, "Shutting down, got signal");
  exit(0);
}

static void sigsegv(int signo) {
  if (waitpid(-1, NULL, 0) > 0) {
  }
}

int main(int argc, char **argv) {
  int i;
  for (i = 1; i < argc; ++i) {
    if (strlen(argv[i]) == 2) {
      printf("%s", VERSION_STRING(VERSION));
      return 0;
    }
  }
  if (getppid() == 1) {
    /* Not an error because pause sees use outside of infra containers. */
    fprintf(stderr, "Warning: should be the first process's");
  }

  if (sigaction(SIGINT, &struct sigaction{.sa_handler = sigint}, NULL) < 0)
    return 1;
  if (sigaction(SIGTERM, &struct sigaction{.sa_handler = sigterm}, NULL) < 0)
    return 2;
  if (sigaction(SIGHUP, &struct sigaction{.sa_handler = sigterm},
                .sa_flags = SA_NOCLOSE|SA_RESTART),
    NULL) < 0)
    return 3;

  for (;;) {
    pause();
    fprintf(stderr, "Error: infinite loop terminated\n");
    return 4;
  }
```

3b45e983c859 gcr.io/google_containers/pause-amd64:3.0 "pause" ...
3b45e983c859 gcr.io/google_containers/pause-amd64:3.0 "/pause" ...
dbfc35b00062 gcr.io/google_containers/pause-amd64:3.0 "pause" ...
c4e998ec4d5d gcr.io/google_containers/pause-amd64:3.0 "/pause" ...
508102acf1e7 gcr.io/google_containers/pause-amd64:3.0 "/pause" ...

Microservices Networking Construction

- Virtual ethernet pair
  - `ip link add veth0 type veth peer name veth1`
- Move one of them to different namespace
  - `ip link set veth1 netns netns1`
- Create a linux bridge
  - `ip link add name br0 type bridge` // or `brctl addbr br0`
- Add veth0 to the bridge
  - `ip link set dev veth0 master br0` // or `brctl addif br0 veth0`

- IP address management (ipam)
- Routing of overlay on the infrastructure
- Multiple designs: using etcd, BGP, ...
Summary of Progress So Far

- Topic 0: Network structure, layered software architecture
- Topic 1: Client-server network app protocol design, implementation and analysis (REST, HTTP)
- Topic 2: Reliable, efficient, secure network transport (SocketAPI, TCP, QUIC, Reno, Cubic, BBR, MTCP)
- Topic 3: Scalable, programmable networking (L2, OSPF, OLSRv2, BGP, 5GC, OpenFlow, SDN, P4...)
- Topic 4: Microservices architecture (Kubernetes, etcd, RAFT)
Topics 5/6: Network Systems in Cloud/Data Centers

- A large domain, applying the background in topics 0-4, with cloud/DC specialization
  - Topic 0: Network structure
    - Data center topology design
  - Topic 3: Networking control
    - Cloud/DC networking systems (e.g., VL2, Google B4)
  - Topic 2: Transport system
    - DC TCP, co-flow scheduling
  - Topic 1: Network app protocol
    - REST/HTTP => gRPC
    - Data analytics, distributed learning
Our Plan

- Today
  - Cloud/data center basic topology, networking and transport

- Next class
  - Cloud/data center resource scheduling
Outline

- Admin and recap
- Cloud data center (CDC) networking systems
  - Data center topology design (Clos, fat tree)
Big Picture: a “Lego” Problem

- Given small pieces (switches), construct a whole network
Conventional DC Construction

Internet

DC-Layer 3

DC-Layer 2

Key
- CR = Core Router (L3)
- AR = Access Router (L3)
- S = Ethernet Switch (L2)
- A = Rack of app. servers

~ 1,000 servers/pod == IP subnet

Evaluate if it is a good design: Lowest cost satisfying demands.

Reference – “Data Center: Load balancing Data Center Services”, Cisco 2004
## DC Construction Cost

<table>
<thead>
<tr>
<th>Amortized Cost*</th>
<th>Component</th>
<th>Sub-Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>~45%</td>
<td>Servers</td>
<td>CPU, memory, disk</td>
</tr>
<tr>
<td>~25%</td>
<td>Power infrastructure</td>
<td>UPS, cooling, power distribution</td>
</tr>
<tr>
<td>~15%</td>
<td>Power draw</td>
<td>Electrical utility costs</td>
</tr>
<tr>
<td>~15%</td>
<td>Network device</td>
<td>Switches, links, transit</td>
</tr>
</tbody>
</table>

*3 yr amortization for servers, 15 yr for infrastructure. 5% cost of money.


Typical design: find the most expensive components and make them efficient and lower cost.
Goal: Agility - Any Service, Any Server

- Turn the servers into a single large fungible pool
  - Dynamically expand and contract service footprint as needed

- Benefits
  - Lower server component cost
  - Achieve high performance and reliability
  - Increase service developer productivity

Agility: The same of most infrastructure projects
Achieving Agility

- **Workload management**
  - Means for rapidly installing a service's code on a server
  - *Virtual machines, disk images, containers*

- **Storage Management**
  - Means for a server to access persistent data easily
  - *Distributed filesystems (e.g., HDFS, blob stores)*

- **Network**
  - Means for communicating with other servers, regardless of where they are in the data center
Conventional DC Topology: Problem

Heterogenous server-to-server capacity
- Fundamentally a tree, the higher up in the tree, the more potential competition on resources, limiting server placement flexibility.
Conventional DC Topology: Problem

Poor reliability
- Fundamentally a tree, link failures in top of the tree can lead to large fraction loss/reliability issues
Inspiration of Modern Data Center Network: Clos Networks

Studied in 1953 by Clos on modular construction of flexible telephone switching network 

\(<m,n>\) ports interface switches: \(n\) facing users, \(m\) back switching.

Given \(src, dst\), flexibility is the first step.

Fix \(n\). the larger \(m\), the more flexibility, but the higher cost.

Q: How big is \(m\) so that each new call can be established w/o moving current calls?

If one can move existing calls, it is only \(m \geq n\): one can find route for any \(rn\) input -> \(rn\) output permutation (based on Hall’s Marriage Theorem)
Folded Clos (Fat-Tree) Topology

- One way to check a topology
  - Take any src ports set S, dst ports set D, the maximum flow from S -> D is limited by src/dst ports

Generic K-ary Fat Tree

- K-ary fat tree: three-layer topology (edge, aggregation and core)
  - k pods w/ each pod consisting of 2 layers of k/2 k-port switches
    - each edge switch connects to k/2 servers & k/2 aggr. switches
    - each aggr. switch connects to k/2 edge & k/2 core switches
  - each core switch connects to k pods

General Topology Design Check

- It can handle any traffic matrix $t$, as long as it is bounded by the port capacities
  - $T_{ij}$ as demand from port $i$ to port $j$
  - Sum of $t_{12} + t_{13} + \ldots + t_{1n} + t_{21} + t_{31} + \ldots + t_{n1} \leq C_1$, ...
**Generic K-ary Fat Tree**

K-ary fat tree: k pods w/ each pod consisting of 2 layers of k/2 k-port switches; each switch w/ half ports go up, half down

Ex: How many servers per pod?
A: $k/2 \times k/2 = k^2/4$

Ex: How many servers in total?
A: $k^2/4 \times k = k^3/4$

Ex: How many edge switches in total?
A: $k/2 \times k = k^2/2$

Ex: How many core switches in total?
A: $k^2/2 \times k/2 / k = k^2/4$
## Size Intuition

<table>
<thead>
<tr>
<th>k</th>
<th>#pods (k)</th>
<th>#servers/pods (k2/4)</th>
<th>#switches/pod (k/2*2)</th>
<th>#total core switches (k2/4)</th>
<th>#total switches (k2/4+k*k)</th>
<th>#total servers (k3/4)</th>
<th>#total links</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>64</td>
<td>16</td>
<td>64</td>
<td>320</td>
<td>1,024</td>
<td>3,072</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>256</td>
<td>32</td>
<td>256</td>
<td>1,280</td>
<td>8,192</td>
<td>24,576</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>1,024</td>
<td>64</td>
<td>1,024</td>
<td>5,120</td>
<td>65,536</td>
<td>196,608</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>4,096</td>
<td>128</td>
<td>4,096</td>
<td>20,480</td>
<td>524,288</td>
<td>1,572,864</td>
</tr>
</tbody>
</table>
Microsoft VL2 Topology

Assume
- Each Int switch has $D_I$ ports;
- Each Aggr has $D_A$ ports

Each Aggr switch uses half ports to connect to TOR switches, half to each Intermediate switch.

Each TOR connects to two Aggr switches.

Each Int switch connects to each Aggr switch.

$D_A/2$ Int switches

$D_I$ Aggr switches

$D_I D_A/4$ TOR

Each TOR connects $20$ servers

$20 (D_I D_A/4)$ servers

Q: TOR up link vs down link bw?
Microsoft VL2 Topology

<table>
<thead>
<tr>
<th>D=D_I=D_A (# of 10G ports)</th>
<th>Max DC size (# of Servers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>11,520</td>
</tr>
<tr>
<td>96</td>
<td>46,080</td>
</tr>
<tr>
<td>144</td>
<td>103,680</td>
</tr>
</tbody>
</table>

[Diagram of VL2 Topology with annotations: Int, Aggr, TOR, and 20 Servers]
Some Other Topologies

BCube

Jellyfish (random) [NSDI’12]
Offline: Facebook Layout

Long cables (fiber)

Offline: Single-Chip “Merchant Silicon” Switches

Image courtesy of Facebook
Summary: Datacenter Network Ultimate Goal

Provide the illusion of “One Big Switch”
Outline

- Admin and recap
- Cloud data center (CDC) networking systems
  - Data center cluster topology design (Clos, fat tree)
  - Data center cluster networking (routing) control (ECMP)
Does using fat-tree topology to inter-connect racks of servers in itself sufficient—we can use any networking control plane?

- How about the L2 control plane (flood+Learn)?
  - ARP flooding, spanning tree removes most capacities!

- How about a layer 3 IP routing control plane (e.g., OSPF)?
  - shortest path routing to each server
    - constructing a path for each server as a dst will need large flow tables
    - assume 10 million virtual endpoints in 500,000 servers in datacenter => 10 m entries, but typical switch has only 640K switch memory, for 32-64K flow entries
  - aggregation (e.g., each edge switch a subnet) to reduce flow table size
    - VM cannot move easily as address becomes locator
Outline

- Admin and recap
- Cloud data center (CDC) networking systems
  - Data center cluster topology design
  - Data center cluster networking (routing) control
    - Microsoft VL2 data center control
Microsoft VL2 Control Plane: Goals

- L2 control plane semantics with low control plane overhead
  - Easily assign any server to any service
    - Assigning servers to service should be independent of network topology
    - Configure server with whatever IP address the service expects
    - VM keeps the same IP address even after migration

- Uniform high capacity (fully utilize switch fabric):
  - Maximum rate of server to server traffic flow should be limited only by capacity of network cards

- Performance isolation:
  - Traffic of one service should not be affected by traffic of other services
VL2 Solution to Addressing and Routing: Name-Location Separation

Whole network as a L2 domain w/o CP flooding overhead.

Servers IP as flat name  Routing uses locator (ToR address)

VL2

Directory Service

...  

x → ToR_2

y → ToR_3

z → ToR_3

...

Lookup & Response

ToR_1  ...  ToR_2  ...  ToR_3  ...  ToR_4

ToR_3  y  payload

ToR_3  z  payload

x

yyz

z
Discussion

- Requirements on the Directory System?
- What is a possible design?
VL2 Directory System

Directory Servers

Read-optimized Directory Servers for fast lookups

Write-optimized Replicated State Machines using Paxos for reliable updates

1. Update

2. Reply

3. Replicate

4. Ack

5. Ack

6. Disseminate

Agent

“Lookup”

“Update”
Routing Design Option

Remaining issue: What are the path(s) for each srcTor/dstTor?
Assume
- 8 port switches
- Each link is 10G

Exercise: An example routing which can lead to contention/no isolation?
Assume
- 8 port switches
- Each link is 10G

Objective: Spread traffic so that no such contention can happen, as long as each host is bounded by interface card rate.
Solution: Valiant load balancing: uniform spread one more hop
Example: VLB Idea (Goal)

Assume
- 8 port switches
- Each link is 10G

For a1 -> a2 traffic: uniformly scatter to Int nodes: a1 -> \{c1, c2, c3, c4\} -> a2
Exercise: VLB Idea (Goal)

Assume
- 8 port switches
- Each link is 10G
VLB Intuition in VL2 Setting (Aggr-Int)

- Alg: spread traffic uniformly to the Int switches
- Q: Bound (assume DI = DA = 8; each link rate R):
  - a -> i traffic
    - $\frac{1}{4}$ of total upstream traffic from a ($\leq \frac{1}{4} \times 4 R = R$)
  - i -> a traffic (system enforce total->a <= a’s down stream)
    - $\frac{1}{4}$ of total traffic going down to a ($\leq R$)
Realizing VLB: ECMP (1)

Endhost picks a random Int (e.g., $I_0$) and encap

Switches use ECMP routing; Int switches and ToR switches do decap

Exercise: what are all upstream paths and downstream paths?
Realizing VLB: ECMP 1: Problems

Problem: Need to update each host if an Int switch changes state.
Problem: Each Aggr switch ECMP has no flexibility (fixed I). ECMP does not realize VLB.
Final VL2 Routing

VL2: All Int switches assigned the same anycast addr.
Ex: all upstream paths and downstream paths?
Intuition: ECMP in DC Topology

- Many equal cost paths going up to the core switches
  - ECMP tries to approximate VLB
- One path down from each core switch
Implementation Question

What are tables and actions at each network node?
Assume ECMP. Are there cases adding Iany is different from not adding?
Summary: Microsoft VL2 Data Center

Takeaways
- Separating names from locations
- Leveraging end systems
- Randomizing (through ECMP to approximate VLB) to cope with variability in traffic matrices

Issues of VL2?
ECMP Collision Problem in DC

- ECMP collisions possible in two different ways
  - Upward path
  - Downward path
Impacts of Collisions

- Average of 61% of bisection bandwidth wasted on a network of 27K servers
Randomization, Load Balancing, Flow Size, and Link Bandwidth (Intuition)

Load Balancing vs Item Sizes

20 × 1Gbps Uplinks

2 × 10Gbps Uplinks

11 × 1Gbps flows (55% load)

Prob of 100% throughput = 3.27%

Prob of 100% throughput = 99.95%