Cloud and containers

Ratul Mahajan
CSE 461
HUGE data centers (DCN)

- Thousands of routers
- Hundreds of thousands of servers

Connected by massive pipes

Microsoft and Facebook just laid a 160-terabits-per-second cable 4,100 miles across the Atlantic

*Enough bandwidth to stream 71 million HD videos at the same time*

By Thuy Ong | @ThuyOng | Sep 25, 2017, 7:56am EDT

Google’s Oregon DC
Inside a Google DC
DCN topologies

• Big iron → Commodity switches
DCN topologies

- Big iron $\rightarrow$ Commodity switches
- 1 Gbps $\rightarrow$ 10 Gbps $\rightarrow$ 40 Gbps $\rightarrow$ 100 Gbps (soon)
- Copper $\rightarrow$ Fiber
Oversubscription ratio

• Ratio of bisection bandwidth across layers of hierarchy
• Key design parameter that trades-off cost and performance
  • Higher oversubscription = lower cost but higher chance of congestion
DCN routing

• Spanning tree (L2) $\rightarrow$ OSPF/ISIS $\rightarrow$ BGP

• Each router acts as its own autonomous system (AS)
Backbone

- Provides global connectivity to DCs
Backbone

• Provides global connectivity to DCs

• May also have two backbones
  • A “public” backbone to connect to the outside world
  • A ”private” backbone for inter-DC connectivity

• Uses transcontinental and transoceanic fiber cables

• Routing: Distributed routing → SDN-based traffic engineering
SDN – Software Defined Networking

Decouple control and data plane
  • Control plane populates the data plane entries (routing)
  • Data plane forwards traffic (forwarding)

Traditionally, routing and forwarding are in the same device

Control plane separation opens up lots of new opportunities
  • Traffic engineering in backbones (next)
  • Network virtualization (later)
SDN-based traffic engineering

Centralized computation of forwarding tables
  • Compute “optimal” paths outside of the network
  • Based on estimated load; also factor in application priorities
What is in the box?
Router

A computer optimized for routing and forwarding

- Operating system to manage resources
- Routing protocol implementations (e.g., BGP, OSPF)
- Lots of ports (network interfaces, not TCP ports)
- Chip to forward traffic between ports at “line rate”
Router (2)

Traditionally, a hardware-software combo sold by a router vendor
- Cisco
- Juniper
- Arista
- ....

But moving toward open systems
- SONiC – open source router OS from Microsoft
- Running on “commodity” hardware
Configuring the router

Routers are not plug-n-play

• Configure IP addresses
• Configure which protocols to run
• Configure those protocols
• Configure management aspects, e.g., DNS servers, NTP servers

Configuration uses custom syntax:

• Example Cisco file:  
  https://github.com/batfish/pybatfish/blob/master/jupyter_notebooks/netwoks/example/configs/as1border2.cfg
Configuring the router (2)

Traditionally, configuration has been done manually
  • Figure out the change, reason about it manually
  • Log in to the router and apply the change
  • High risk of logical errors and “fat fingers”

Increasingly, more automation
  • Ansible, Batfish
Making a network out of routers

1. Get them connected
Making a network out of routers

1. Get them connected

2. Configure routers
   • Basic initial configuration provides connectivity to the router

3. Monitor, monitor, monitor

4. Configuration changes and maintenance
What is in this box?
Originally

```
Hardware
  OS
    Libs
      App
        App
```

To network
Then came virtual machines (VMs)

HW became too powerful
• Run multiple OSes on the same machine
• Cheaper that way

The hypervisor virtualizes the HW and fools the OS
• Provides isolation

The network thinks multiple hosts are connected
The hypervisor acts as a hub for inter-VM traffic
Forwarding between VMs involves a lookup from overlay address to underlay location.
Enter containers

Lighter-weight virtualization than VMs
• Libraries, not the full OS

Better isolation and packaging than apps
• Bundle the library versions you need
Container networking

Connect containers to the outside world and to each other

• Port conflicts among containers and other apps running on the same host
• High performance between containers on the same host
• (Virtual) private network between related containers (service mesh)
Container networking: Host

Containers share the IP address (and networking stack) of the host.

- Cannot handle port conflicts
- Minimal overhead
Container networking: Bridge

An internal network for containers on the same host.

- Use NATs for outside world
Container networking: Overlay

Create a private network across containers on different hosts
• VXLAN is a common way to do that
Enter microservices

Instead of developing a large monolithic application, structure the application as a bunch of communicating microservices

• Each microservice serves a (small) dedicated function, e.g., authentication
  • Can be written in any language
  • Can evolve independent of other microservices
  • Can be scaled independent of other microservices
• Each microservice gets a container

But now you may have lots of services across lots of containers

• Containers need to be deployed and scaled ➔ container orchestration
• Communication between services needs to be managed ➔ service meshes
Container orchestration (Kubernetes)

**Containers** are wrapped in **Pods** which are run on a **Cluster** of **Nodes**

Pods implement a **service**

https://sensu.io/blog/how-kubernetes-works
Service meshes (Istio)

“Application defined networking”
• Secure inter-service communication
• Load balancing for HTTP, gRPC, WebSocket, and TCP traffic
• Traffic behavior (routing rules, retries, failover)
• Access control, rate limits, and quotas
• Metrics, logs, and traces

What is not to like?

https://istio-releases.github.io/v0.1/docs/concepts/what-is-istio/overview.html
Service mesh overhead measurements