1. Floodless in Seattle: Kim et.al; SIGCOMM 2008
3. VL2: A scalable and flexible data center network SIGCOMM 2009
4. F10: A fault tolerant engineered network; Vincent Liu et al; NSDI 2013
5. Networking the cloud, Albert Greenberg, ICDCS 2009 Keynote talk
Data Center Costs

<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</td>
<td>Switches, links, transit</td>
</tr>
</tbody>
</table>

- Upwards of $1 to $4 B for mega data center
- Server costs dominate
- Network costs significant


*3 yr amortization for servers, 15 yr for infrastructure. 5% cost of money
Data center vs Enterprise

- **Enterprise: IT cost dominates**
  - 1 Human: 100 servers
  - Automation is partial, configuration, monitoring not fully automated

- **Data center: Other costs**
  - 1 Human: 1000 servers
  - Automation is mandatory, scale
Data center vs Enterprise

- Enterprise: Scale not present
  - Limited Shared resources
  - Isolation

- Data center: Scale out
  - 100000 servers
  - Upfront cost is high, leverage shared resources

- Scale up vs scale out
- Enterprise: a few high priced servers– scale up
- Datacenter: scale out, distributed workload, spread out a number of commodity servers
Data center vs Enterprise

- **Enterprise: CAPEX**
  - Capital expenditure borne by the enterprise
  - License and maintenance
  - Utilization not important

- **Data center: OPEX**
  - Pay per use for customers
  - Upfront cost is high, amortized over time and use
  - Utilization is very important
Architecture of Data Center Networks (DCN)

Internet

Data Center
Layer 3

Layer 2

A Single Layer 2 Domain

Key:
- BR = L3 Border Router
- AR = L3 Access Router
- S = L2 Switch
- LB = Load Balancer
- A = Rack of Servers
New Challenges to Ethernet, and SEATTLE as a solution

SEATTLE (Scalable Ethernet Architecture for Larger Enterprises)

**Motivation**
Neither bridging nor routing is satisfactory. Can’t we take only the best of each?

<table>
<thead>
<tr>
<th>Architectures Features</th>
<th>Ethernet Bridging</th>
<th>IP Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of configuration</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Optimality in addressing</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Host mobility</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>Path efficiency</td>
<td>✗</td>
<td>✅</td>
</tr>
<tr>
<td>Load distribution</td>
<td>✗</td>
<td>✅</td>
</tr>
<tr>
<td>Convergence speed</td>
<td>✗</td>
<td>✅</td>
</tr>
<tr>
<td>Tolerance to loop</td>
<td>✗</td>
<td>✅</td>
</tr>
</tbody>
</table>

**SEATTLE**

![Diagram of SEATTLE architecture](image-url)
Overview: Objectives

● Objectives
  ● Avoiding flooding
  ● Restraining broadcasting
  ● Keeping forwarding tables small
  ● Ensuring path efficiency

● SEATTLE architecture

● Evaluation

● Applications and Benefits

● Conclusions
Avoiding Flooding

- Bridging uses flooding as a routing scheme
  - Unicast frames to unknown destinations are flooded

- Does not scale to a large network

- Objective #1: **Unicast unicast traffic**
  - Need a control-plane mechanism to discover and disseminate hosts’ location information

“Don’t know where destination is.”

“Send it everywhere! At least, they’ll learn where the source is.”
Restraining Broadcasting

- Liberal use of broadcasting for bootstrapping (DHCP and ARP)
  - Broadcasting is a vestige of shared-medium Ethernet
  - Very serious overhead in switched networks

- Objective #2: Support unicast-based bootstrapping
  - Need a directory service

- Sub-objective #2.1: Yet, support general broadcast
  - Nonetheless, handling broadcast should be more scalable
Keeping Forwarding Tables Small

- Flooding and self-learning lead to unnecessarily large forwarding tables
  - Large tables are not only inefficient, but also dangerous

- Objective #3: Install hosts’ location information only when and where it is needed
  - Need a reactive resolution scheme
  - Enterprise traffic patterns are better-suited to reactive resolution
Ensuring Optimal Forwarding Paths

- Spanning tree avoids broadcast storms. But, forwarding along a single tree is inefficient:
  - Poor load balancing and longer paths
  - Multiple spanning trees are insufficient and expensive

- Objective #4: *Utilize shortest paths*
  - Need a routing protocol

- Sub-objective #4.1: *Prevent broadcast storms*
  - Need an alternative measure to prevent broadcast storms
Backwards Compatibility

- Objective #5: Do not modify end-hosts
  - From end-hosts’ view, network must work the same way

  - End hosts should
    - Use the same protocol stacks and applications
    - Not be forced to run an additional protocol
Basic Idea

- Provide a directory service based on consistent hashing
- Links state to build a topology among switches – efficient routing
- Each switch can store MAC → location
- MAC → (IP, location) stored in F(MAC)
- Each switch can store IP → MAC
- IP → (MAC, Location) stored in H(IP)
- Can also store location of services. E.g., DHCP_server,
Consistent hashing

- Hash keys and bucket-id to some uniform name space
- Assign key to the first bucket encountered in the name space
- Make collisions rare (for the bucket-ids)

When servers/buckets come and go small local movements
Maintain a directory to quickly locate server holding items
Consistent hashing

A key is stored at its **successor**: node with next higher ID
Single Hop Look-up

\( y \) sends traffic to \( x \)

Every switch on a ring is logically one hop away.
How does it work?

**Host discovery or registration**

Hash \( F(x) = B \)

Tunnel to egress node, \( A \)

Tunnel to relay switch, \( B \)

Tunnel to relay switch, \( B \)

Entire enterprise (A large single IP subnet)

Traffic to \( x \)

Notifying \(<x, A>\) to \( D \)

Store \(<x, A>\) at \( B \)

Optimized forwarding directly from \( D \) to \( A \)

Deliver to \( x \)

Switches

End-hosts

Control flow

Data flow
Terminology

Ingress applies a cache eviction policy to this entry.

shortest-path forwarding

Ingress (for \( x \))

Egress

Dst \( \langle x, A \rangle \)

Egress

Src \( \langle x, A \rangle \)

Relay (for \( x \))

\( \langle x, A \rangle \)
Responding to Host Mobility

when shortest-path forwarding is used

Old Dst

New Dst

Relay (for x)

when shortest-path forwarding is used
Unicast-based Bootstrapping: ARP

- ARP
  - Ethernet: Broadcast requests
  - SEATTLE: Hash-based on-demand address resolution

1. Host discovery
2. Hashing $F(IP_a) = r_a$
3. Storing $(IP_a, mac_a, s_a)$
4. Broadcast ARP req for $a$
5. Hashing $F(IP_a) = r_a$
6. Unicast ARP req to $r_a$
7. Unicast ARP reply $(IP_a, mac_a, s_a)$ to ingress

Owner of $(IP_a, mac_a)$

Switch
End-host
Control msgs
ARP msgs
Unicast-based Bootstrapping: DHCP

- DHCP
  - Ethernet: Broadcast requests and replies
  - SEATTLE: Utilize DHCP relay agent (RFC 2131)
    - Proxy resolution by ingress switches via unicasting

Diagram:
- **DHCP server** ($mac_d=0xDHCP$)
- **d** (Host discovery)
- **S_d** (Hashing $F(mac_d) = r$)
- **r** (Storing $(mac_d, s_d)$)
- **S_h** (Broadcast DHCP discovery)
- **h** (DHCP msg to $r$
- **6. DHCP msg to $r$**
- **7. DHCP msg to $s_d$**
- **8. Deliver DHCP msg to $d$**

Switch
- Control msgs
- DHCP msgs

End-host

Legend:
Ideal Application: Data Center Network

- **Data centers**
  - Backend of the Internet
  - Mid- (most enterprises) to mega-scale (Google, Yahoo, Facebook, etc.)
    - E.g., A regional DC of a major on-line service provider consists of 25K servers + 1K switches/routers
- **To ensure business continuity, and to lower operational cost, DCs must**
  - Adapt to varying workload → **Breathing**
  - Avoid/Minimize service disruption (when maintenance, or failure) → **Agility**
  - Maximize aggregate throughput → **Load balancing**
Conclusions

- **SEATTLE** is a plug-and-playable enterprise architecture ensuring both scalability and efficiency.

- **Enabling design choices**
  - Hash-based location management
  - Reactive location resolution and caching
  - Shortest-path forwarding

- **Lessons**
  - Trading a little data-plane efficiency for huge control-plane scalability makes a qualitatively different system
  - Traffic patterns are our friends
PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Networks Fabric

Radhika Niranjan Mysore, et.al,
Department of Computer Science and Engineering
University of California San Diego
SIGCOMM 2009
Motivation

- **Requirements for Data Center Networks (DCN):**
  - **R1:** Any VM may migrate to any physical machine without change their IP addresses
  - **R2:** An administrator should not need to configure any switch before deployment
  - **R3:** Any end host should efficiently communicate with any other end hosts through any available paths
  - **R4:** No forwarding loops
  - **R5:** Failure detection should be rapid and efficient

- **Implication on network protocols:**
  - A single layer2 fabric for entire data center (R1&R2)
  - Mac forwarding tables with hundreds of thousands entries (R3)
  - Efficient routing protocols which disseminate topology changes quickly to all points (R5)
## Recall: SEATTLE

<table>
<thead>
<tr>
<th>Layer</th>
<th>PlugNPlay</th>
<th>Scalability</th>
<th>Switch state</th>
<th>VM migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 2 (MAC)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Layer 3 (IP)</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

- OSPF among Switches
  - Links state broadcast to all switches
- Switch stores O(N) state
- Datacenter: Virtualization
- Each end host can have 10 to 20 virtual endpoints
- 100000 servers ➔ 2 M endpoints
SEATTLE vs Portland

• SEATTLE: 1-hop DHT; directory stores IP, MAC, location (Switch_ID) mappings
• Portland: Consider 1 fixed tree structure
• Use MAC address that encodes location!
Datacenter considerations

- Layer 2 approach:
  - Forwarding on flat MAC addresses
  - Less administrative overhead
  - Bad scalability
- Combine of layer 2 and layer 3:
  - VLAN
  - Resource partition problem

- End host visualization:
  - Needs to support large addresses and VM migrations
  - In layer 3 fabric, migrating the VM to a different switch changes VM’s IP address
  - In layer 2 fabric, migrating VM incurs scaling ARP and performing routing/forwarding on millions of flat MAC addresses.
Clos topology

(N x K)  (K x N)

1  (M x M)  1
2  2
3  3
4  4
.  K
M  M
Google DC evolution

<table>
<thead>
<tr>
<th>Datacenter Generation</th>
<th>First Deployed</th>
<th>Merchant Silicon</th>
<th>ToR Config</th>
<th>Aggregation Block Config</th>
<th>Spine Block Config</th>
<th>Fabric Speed</th>
<th>Host Speed</th>
<th>Bisection BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four-Post CRs</td>
<td>2004</td>
<td>vendor</td>
<td>48x1G</td>
<td>-</td>
<td>-</td>
<td>10G</td>
<td>1G</td>
<td>2T</td>
</tr>
<tr>
<td>Firehose 1.0</td>
<td>2005</td>
<td>8x10G 4x10G (ToR)</td>
<td>2x10G up 24x1G down</td>
<td>2x32x10G (B)</td>
<td>32x10G (NB)</td>
<td>10G</td>
<td>1G</td>
<td>10T</td>
</tr>
<tr>
<td>Firehose 1.1</td>
<td>2006</td>
<td>8x10G</td>
<td>4x10G up 48x1G down</td>
<td>64x10G (B)</td>
<td>32x10G (NB)</td>
<td>10G</td>
<td>1G</td>
<td>10T</td>
</tr>
<tr>
<td>Watchtower</td>
<td>2008</td>
<td>16x10G</td>
<td>4x10G up 48x1G down</td>
<td>4x128x10G (NB)</td>
<td>128x10G (NB)</td>
<td>10G</td>
<td>nx1G</td>
<td>82T</td>
</tr>
<tr>
<td>Saturn</td>
<td>2009</td>
<td>24x10G</td>
<td>24x10G</td>
<td>4x288x10G (NB)</td>
<td>288x10G (NB)</td>
<td>10G</td>
<td>nx10G</td>
<td>207T</td>
</tr>
<tr>
<td>Jupiter</td>
<td>2012</td>
<td>16x40G</td>
<td>16x40G</td>
<td>8x128x40G (B)</td>
<td>128x40G (NB)</td>
<td>10/40G</td>
<td>nx10G/nx40G</td>
<td>1.3P</td>
</tr>
</tbody>
</table>

Table 2: Multiple generations of datacenter networks. (B) indicates blocking, (NB) indicates Nonblocking.
Fat Tree topology

- Fat Tree Networks:
  - Split fat tree into three layers:
    - Labeled edge, aggregation and core
    - Split fat tree into k pods (k=4)
    - Each pod with $k^2 / 4$ hosts
    - Each source and destination has $k^2 / 4$ paths
    - B/W or capacity progressively increases higher towards the root
Positional addressing

POD Number
Positional addressing
Positional addressing
Positional addressing

PMAC:pod.position:port:vmid

00:01:03:01

48 bits PMAC address

03:01:02:04
Positional Pseudo MAC Addresses

- Pseudo MAC (PMAC) addresses encodes the location of the host
  - 48-bit: pod.position.port.vmid
  - Pod (16 bit): pod number of the edge switch
  - Position (8 bit): position in the pod
  - Port (8 bit): the port number it connects to
  - Vmid (16 bit): VM id of the host
  - Edge switches assign vmids to MAC addresses seen on its ports
Proxy-based ARP
Fabric Manager

- Characteristics:
  - Logically centralized user process running on a dedicated machine
  - Maintains soft state about network configuration information
  - Responsible for assisting with ARP resolution, fault tolerance and multicast

- Why centralized?
  - Eliminate the need for administrator configuration
Distributed Location Discovery

- Switches periodically send Location Discovery Message (LDM) out all of their ports to set their positions and to monitor liveness.

- LDM contains: switch identifier, pod number, position, tree level, up/down.

- Find position number for edge switch:
  - Edge switch randomly proposes a value in \([0, k/2-1]\) to all aggregation switch in the same pod.
  - The unused and not tentatively reserved ones are verified.

- Find tree level and up/down state:
  - Port states: disconnected, connected to end host, connected to another switch.
  - A switch with at least half of ports connects to end hosts is an edge switch, ports connect to other switches are upward.
  - A switch get LDM from edge switch is aggregation switch, ports connect to edge switch are downward, ports connect to core switches are upward.
  - A switch with all ports connect to aggregation switch is core switch, all ports are downward.
Location Discovery

1. Discover you are an edge switch – How?
2. Determine a Position number – choose random and verify by majority
3. Assign Pod Number: Each switch with Position = 0, Requests Fabric manager for Pod number
Implementation

- Scalability
  - Each host transmit 25, 50, 100 ARP requests/sec to fabric manager
Conclusions

- A scalable, fault tolerant layer 2 routing and forwarding protocol for DCN
- Based on fat tree network topology
- PMAC used to encode the location of the end host
- AMAC to PMAC translation needed
- Header rewriting
Virtual Layer 2: A Scalable and Flexible Data-Center Network
Albert Greenberg et.al., SIGCOMM 2009

Microsoft Research
Tenets of Cloud-Service Data Center

- **Agility**: Assign any servers to any services
  - Boosts cloud utilization

- **Scaling out**: Use large pools of commodities
  - Achieves reliability, performance, low cost

- Statistical Multiplexing Gain + Economies of Scale
VL2: basic idea

- Configure servers in such a way that they appear to be in one big IP subnet
- Avoid Broadcast (ARP) by using a directory service to convert IP to MAC address of RAC containing the server
- Use encapsulation to forward packet to ToR switch
- Aggregate switches uses IP anycast to do load balancing among many upward paths to Intermediate switch
Status Quo: Conventional DC Network

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

**Reference** – “Data Center: Load balancing Data Center Services”, Cisco 2004
Conventional DC Network Problems

- Dependence on high-cost proprietary routers
- Extremely limited server-to-server capacity
And More Problems …

- Resource fragmentation, significantly lowering cloud utilization (and cost-efficiency)
And More Problems …

- Resource fragmentation, significantly lowering cloud utilization (and cost-efficiency)

\[ \text{Complicated manual L2/L3 re-configuration} \]

\[ \approx 200:1 \]

\[ \text{IP subnet (VLAN) #1} \]

\[ \text{IP subnet (VLAN) #2} \]
And More Problems ...

- Resource fragmentation, significantly lowering cloud utilization (and cost-efficiency)
An Example VL2 Topology: Clos Network

- **D/2 switches**
- **D switches**
- **Intermediate node switches**
- **Aggregation switches**
- **Top Of Rack switch**

**Node degree (D) of available switches & # servers supported**

<table>
<thead>
<tr>
<th>D</th>
<th># Servers in pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>24</td>
<td>2,880</td>
</tr>
<tr>
<td>48</td>
<td>11,520</td>
</tr>
<tr>
<td>144</td>
<td>103,680</td>
</tr>
</tbody>
</table>

- A scale-out design with broad layers
  - Same bisection capacity at each layer ➔ no oversubscription
  - Extensive path diversity ➔ Graceful degradation under failure
Addressing and Routing: Name-Location Separation

Cope with host churns with very little overhead

**VL2** Switches run link-state routing and:

- Allows to use low-cost switches
- Protects network and hosts from host-state churn
- Obviates host and switch reconfiguration

Servers use flat names
Separating Names from Locations: How Smart Servers Use Dumb Switches

- Encapsulation used to transfer complexity to servers
  - Commodity switches have simple forwarding primitives
  - Complexity moved to computing the headers
- Many types of encapsulation available
  - IEEE 802.1ah defines MAC-in-MAC encapsulation; VLANs; etc.
Embracing End Systems

- Data center OSes already heavily modified for VMs, storage clouds, etc.
  - A thin shim for network support is no big deal
- No change to applications or clients outside DC
Use Randomization to Cope with Volatility

- Valiant Load Balancing
  - Every flow “bounced” off a random intermediate switch
  - Provably hotspot free for any admissible traffic matrix
  - Servers could randomize flow-lets if needed

Node degree (D) of available switches & # servers supported:

<table>
<thead>
<tr>
<th>D</th>
<th># Servers in pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>24</td>
<td>2,880</td>
</tr>
<tr>
<td>48</td>
<td>11,520</td>
</tr>
<tr>
<td>144</td>
<td>103,680</td>
</tr>
</tbody>
</table>
Traffic Forwarding: Random Indirection

Cope with arbitrary TMs with very little overhead

Links used for up paths
Links used for down paths

$I_{\text{ANY}}$ $I_{\text{ANY}}$ $I_{\text{ANY}}$

$T_1$ $T_2$ $T_3$ $T_4$ $T_5$ $T_6$

$I_{\text{ANY}}$ $T_5$ $z$ payload

$x$ $y$ $z$
Traffic Forwarding: Random Indirection

Cope with arbitrary traffic

- Harness huge bisection bandwidth
- Obviate esoteric traffic engineering or optimization
- Ensure robustness to failures
- Work with switch mechanisms available today

[ ECMP + IP Anycast ]

Links used for up paths
Links used for down paths

I\text{ANY}

T_1 T_2 T_3 T_4 T_5 T_6

I\text{ANY}

I\text{ANY}

x y z

payload
Does VL2 Ensure Uniform High Capacity?

- How “high” and “uniform” can it get?
  - Performed all-to-all data shuffle tests, then measured aggregate and per-flow goodput

<table>
<thead>
<tr>
<th>Goodput efficiency</th>
<th>94%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fairness$^§$ between flows</td>
<td>0.995</td>
</tr>
</tbody>
</table>

$^§$ Jain’s fairness index defined as $\frac{(\sum x_i)^2}{n \cdot \sum x_i^2}$

- The cost for flow-based random spreading

![Graph showing fairness index over time](image-url)
VL2 Conclusion

- VL2 achieves **agility at scale** via
  1. L2 semantics
  2. Uniform high capacity between servers
  3. Performance isolation between services

**Lessons**

- Randomization can tame volatility
- Add functionality where you have control
- There's no need to wait!
Addressing and Routing: Name-Location Separation

Cope with host churns with very little overhead

VL2 Switches run link-state routing and maintain only switch-level topology

Directory Service
...
  x \rightarrow ToR_2
  y \rightarrow ToR_3
  z \rightarrow ToR_3
...

Servers use flat names
F-10 A fault tolerant DC network fabric

- Heart beats to detect failures; exploit ECMP; H(ft)
- DA: Slow detection, recovery and suboptimal load balancing
Fat tree recovery

Lots of redundancy of the upward path; fast recovery
Fat tree recovery slow

NO redundancy of the way down;
Type A sub tree

Consecutive parents
Type B sub tree

Stride parents
Type AB sub tree

Alternate A and B sub trees
Alternate paths in AB tree

; route to child in opp type sub tree; more nodes have alternate paths
Features

- Local rerouting
- Pushback notification
- Centralized scheduler for global optimal traffic