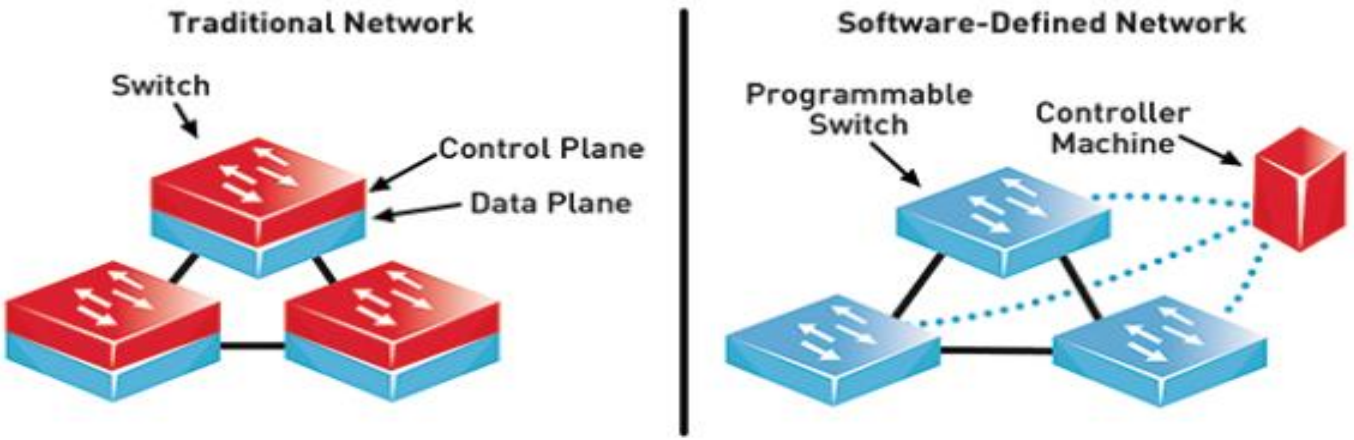
Software Defined Network
Application to Networking in the Cloud

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss the architecture of software defined networking and its applications to networking in the cloud.
- We will also discuss the network Virtualization in multi-tenant data centers with case study of VL2 and NVP



Need of SDN

The traditional networking problem that SDN (software defined networking) is addressing as:

I) Complexity of existing networks

- Networks are complex just like computer system, having system with software.
- But worst than that it's a distributed system
- Even more worse: No clear programming APIs, only **"knobs and dials"** to control certain functions.

II) Network equipment traditionally is proprietary

- Integrated solutions (operating systems, software, configuration, protocol implementation , hardware) from major vendors

RESULT: - Hard and time intensive to innovate new kinds of networks and new services or modify the traditional networks more efficiently.

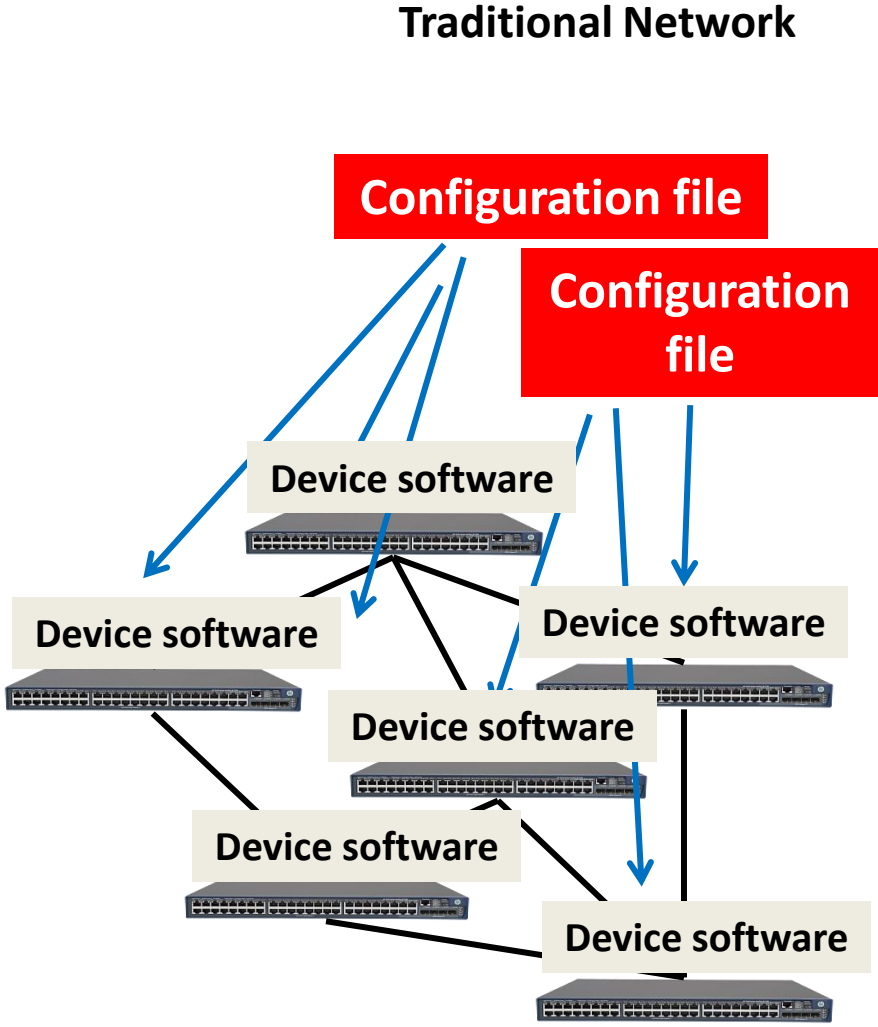
Traditional Network

In traditional network, **configuring a router**, for example, for BGP routing, in a configuration file.

Hundreds, thousands, tens of thousands network devices, router switches and firewalls throughout the network that will **get pushed out configuration to various devices on the network.**

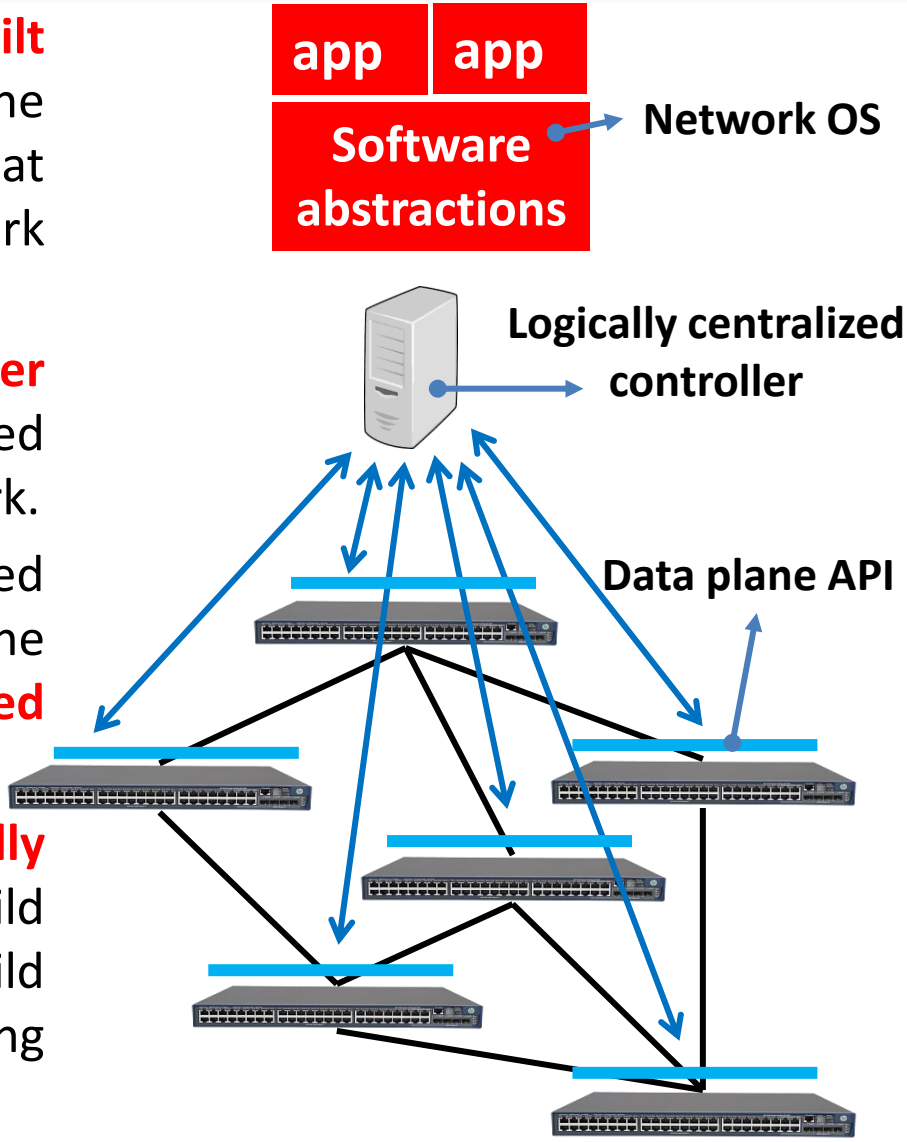
Once it reaches a device then these **configurations have to be interpreted by router software and implementation of protocols** that run distributed algorithms and arrive at routing solutions that produce the ultimate behavior that's installed in the network hardware.

So here the **policy is embedded into the mechanism.** It means the network routing to achieve low latency or high utilization of the network that control are baked into the distributive protocol that are in standardized implementations.



Software-defined network

- The traditional software and OSs are **built on layers and APIs.** So it begin close to the hardware build a low level interface that gives direct access to what the network switching hardware is doing.
- Then a **logically centralized controller** which communicates with distributed switches and other devices in the network.
- The goal of a logically centralized controller is to express our goal in one location and **keep the widely distributed switching gear as simple as possible.**
- **Put the intelligence in a logically centralized location.** On top of that, build software abstractions that help us to build different applications, a network operating system if you want.



Key Ideas of SDN

Key ideas software-defined networking architecture:

Division of Policy and Mechanisms-

- Low-level interface and programmatic interface **for the data plane**
- **Logically centralized controller that allows us to build software abstractions on top of it.**

Example: NOX

NOX is a very early SDN controller. **So to identify a user's traffic, a particular user or computer to send traffic through the network,** and that traffic through the network is going to be tagged with an identifier, a VLAN and that identifies that user.

So to instruct the network we **match a specific set of relevant packets and look at the location where this traffic comes in from as well as the MAC address.** And we're going to construct an action that should happen, in this case, tagging, adding that VLAN tag to the traffic.

And then we're going to **install that action on the specified set of packets in a particular switch.** So we're basically telling the switch, if you see this, do that.

In addition, commonly **SDN controllers have some kind of topology discovery, the ability to control traffic and monitor the behavior in the network.**

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

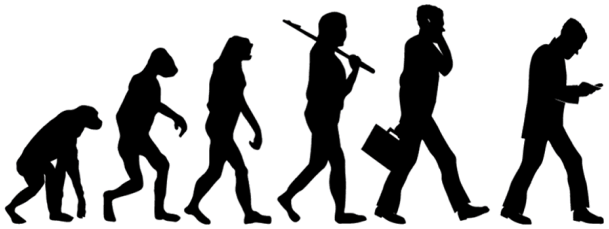
```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port                                  Match specific
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac          set of packets
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),               Construct action
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)           Install (match, action)
                                                              in a specific switch
    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

Key Ideas of SDN

- A programmatic low level interface with the data plane
- Centralized control
- Higher level abstractions that makes easier control.

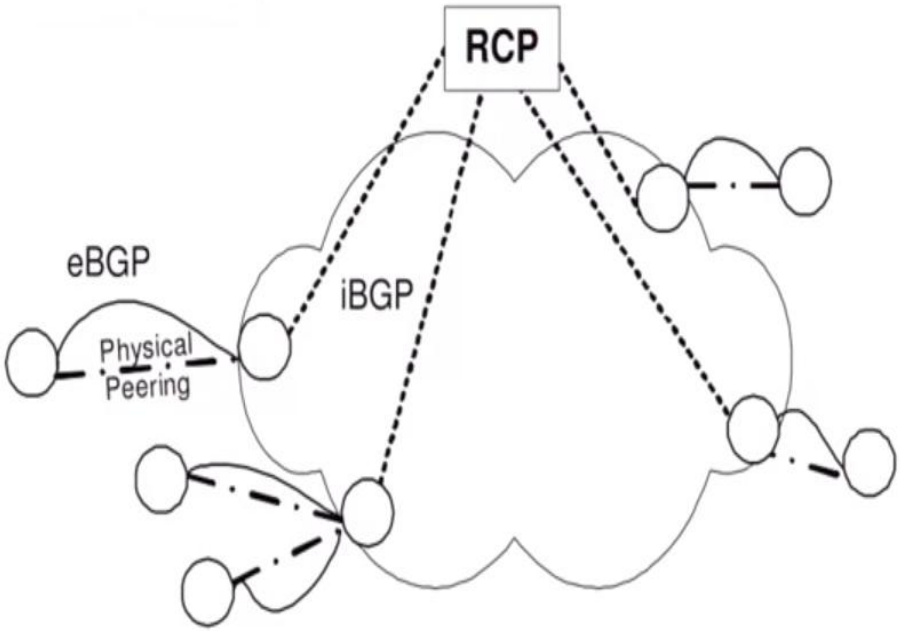
Evolution of SDN: Flexible Data Planes

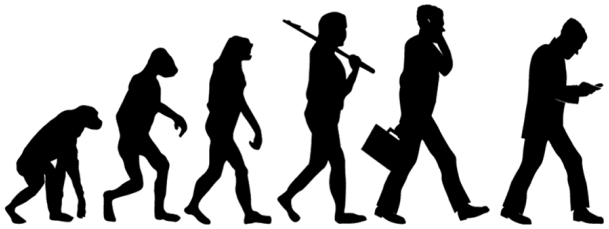
- Evolution of SDN is driving towards making the network flexible.
- **Label switching or MPLS (1997)** i.e. matching labels, executing actions based on those labels adding flexibility:-
- Lay down any path that we want in the network for certain classes of traffic.
- Go beyond simple shortest path forwarding,
- Good optimization of traffic flow to get high throughput for traffic engineering
- Setting up private connections between enterprises and distributed sites.



Evolution of SDN: Logically Centralized Control

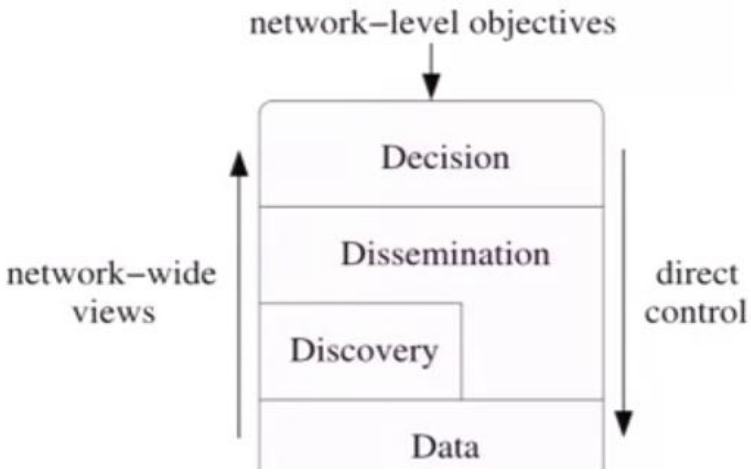
- **Routing Control Platform (2005) [Caesar et al. NSDI 2005]**
- Centralized computing to BGP routes, pushed to border routers via iBGP





Evolution of SDN: Logically Centralized Control

- **4D architecture (2005)**
  - A clean slate 4D Approach to Network control and management [Greensburg, Et. Al., CCR Oct 2005]
  - Logically centralized “decision plane” separated from data planes



Evolution of SDN: Logically Centralized Control

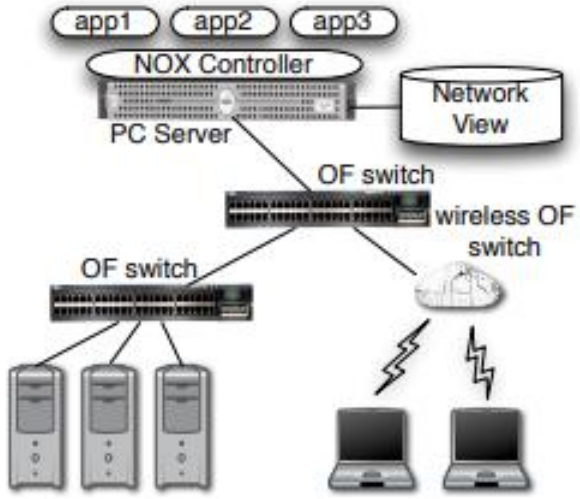
- **Ethane (2007) [Casado et al., SIGCOMM 2007]**
  - Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware.

Evolution of SDN: Logically Centralized Control

- **OpenFlow (2008) [McKeown et al. 2008]**
  - Thin standardized interface to data planes.
  - General purpose programmability at control.

Evolution of SDN: Logically Centralized Control

- Routing Control Platform (2005)
- 4D architecture (2005)
- Ethane (2007)
- OpenFlow (2008)
- **NOX (2008) [Gude et al. CCR 2008]**
  - **First OpenFlow (OF) controller:** centralized network view provided to multiple control applications s as a database.
  - Handles state collection and distribution.
- **Industry explosion (~2010+)**



SDN Opportunities

- **Open data plane interface**
  - **Hardware :** with standardized API, easier for operators to change hardware, and for vendors to enter market
  - **Software :** can more directly access device behavior
- **Centralized controller:**
  - Direct programmatic control of network
- **Software abstraction of the controller**
  - Solves distributed systems problem only once, then just write algorithm.
  - Libraries/languages to help programmers write net apps
  - Systems to write high level policies instead of programming

Challenges of SDN

Performance and Scalability

- Controlling the network through devices to respond quickly with latency concerns i.e. capacity concerns.

Distributed systems challenges still present

- Network is fundamentally a distributed system
- Resilience of logically centralized controllers
- Imperfect knowledge of network state
- Consistency issues between controllers

Architectural Challenges of SDN

- **Protocol to program the data planes**
  - OpenFlow ? NFV function ? WhiteBox switching ?
- **Devising the right control abstraction ?**
  - Programming OpenFlow : far too low level
  - But what are the right level abstractions to cover important use cases ?

The First Cloud Apps for SDN

- **Virtualization of multi-tenant data centers**
  - Create separate virtual network for tenants
  - Allow flexible placement and movements of VMs
- **Inter-datacenter traffic engineering**
  - Trying to achieve maximum utilization near 100% if possible.
  - Protect critical traffic from congestion.
- **Key-characteristics for above use cases**
  - Special purpose deployments with less diverse hardware.
  - Existing solutions aren't just inconvenient and don't work.

Multi-tenant Data Centers : The challenges

Cloud is shared among multiple parties and gives economy of scale. To share the cloud among multiple tenants, there's bit more work to do. So the key needs for building a multi-tenant Cloud data center are:

(i) Agility

(ii) Location independent addressing

(iii) Performance uniformity

(iv) Security

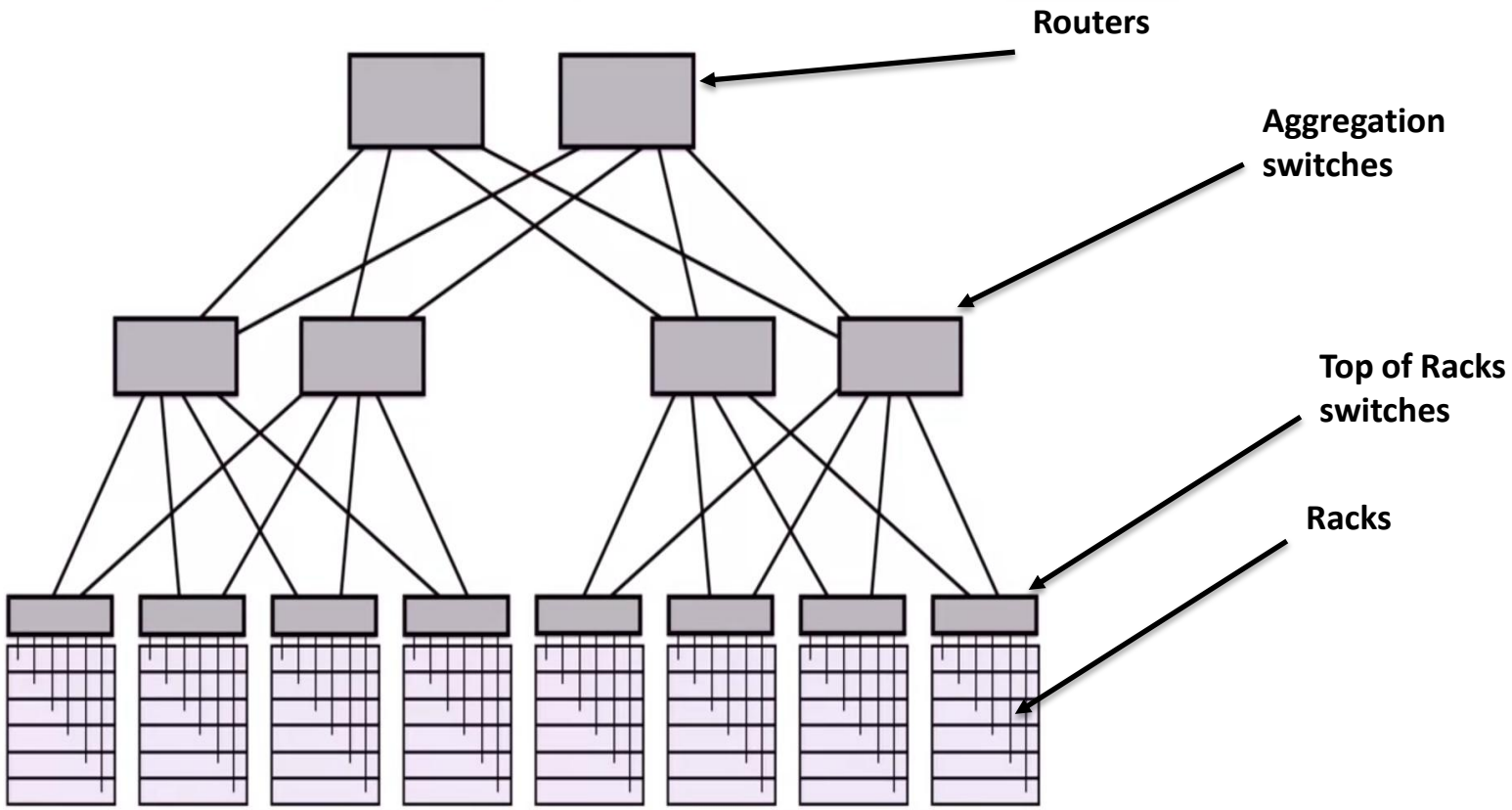
(v) Network semantics

(i) Agility

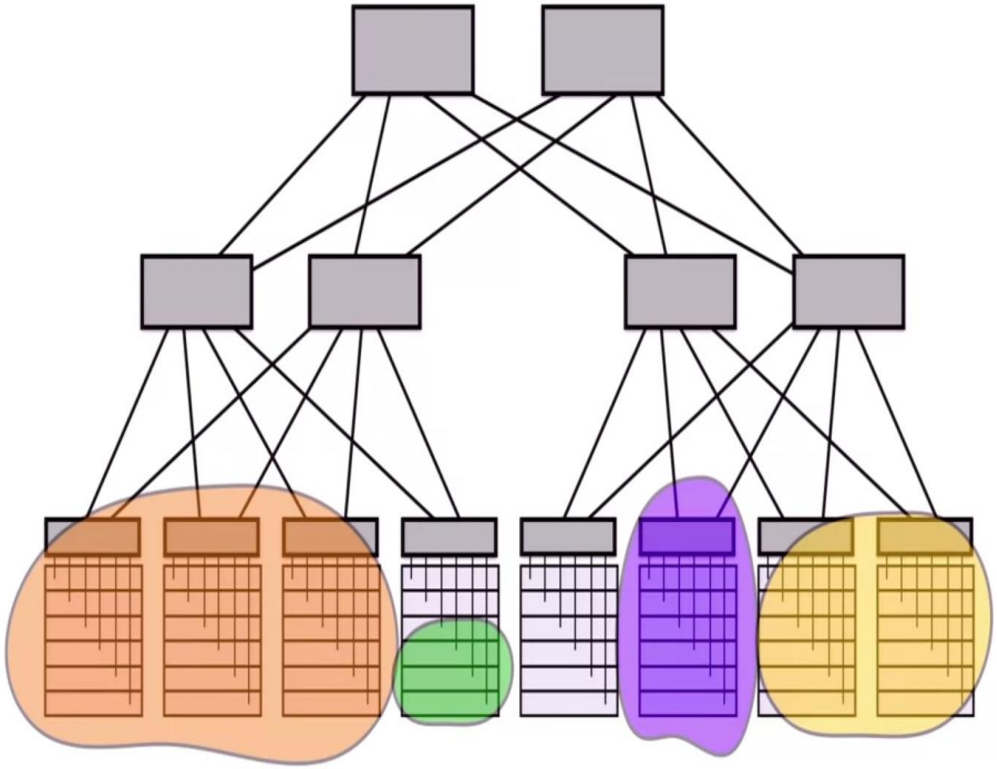
- **Use any server for any service at any time:**
  - **Better economy of scale through increased utilization:** Pack, compute as best we can for high utilization. If we ever have constraints then it's going to be a lot harder to make full use of resources.
  - **Improved reliability:** If there is a planned outage or an unexpected outage, move the services to keep running uninterrupted.

- **Service or tenant can means:**
  - A customer renting space in a public cloud
  - Application or service in a private cloud as an internal customer

Traditional Datacenters



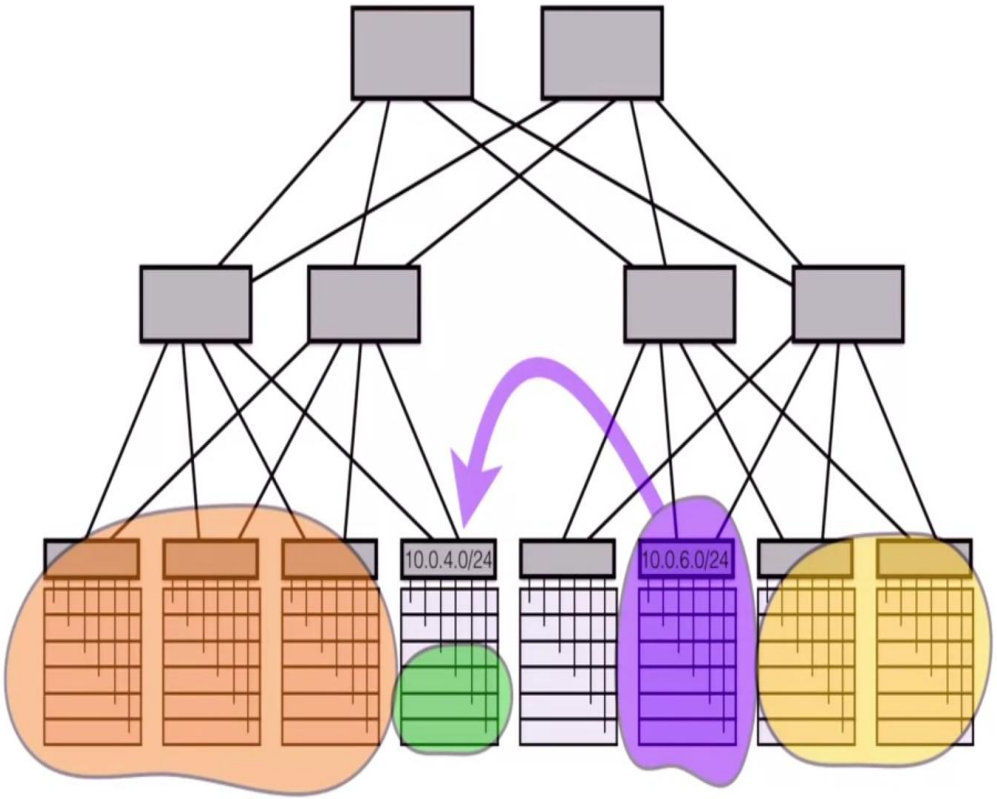
Lack of Agility in Traditional DCs



- **Tenant in "silos" –** Means one rack or a part of the cluster is devoted to a particular service

- **Poor Utilization**
- **Inability to expand**

Lack of Agility in Traditional DCs



- **IP addresses locked to topological location!**

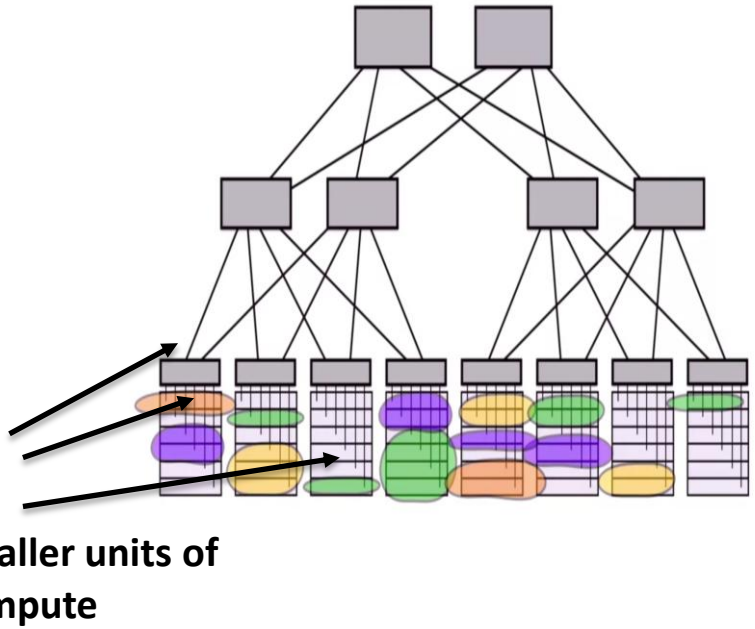
Key needs: Agility

- **Agility**
  - **Location independent addressing:** Racks are generally assigned different IP Subnets, because subnets are used as topological locators so that we can route. To move some service over there, we're going to have to change its IP address and it is hard to change the IP addresses of live running services.

  - **Tenant's IP address can be taken anywhere:** Tenant's IP address to be taken anywhere, independent of the location and the data center without notify tenants that it has changed location. Large over subscription ratio i.e. 100 % or greater if there's a lot of communication between both sides will be about a hundred times lower throughput than communicating within the rack.

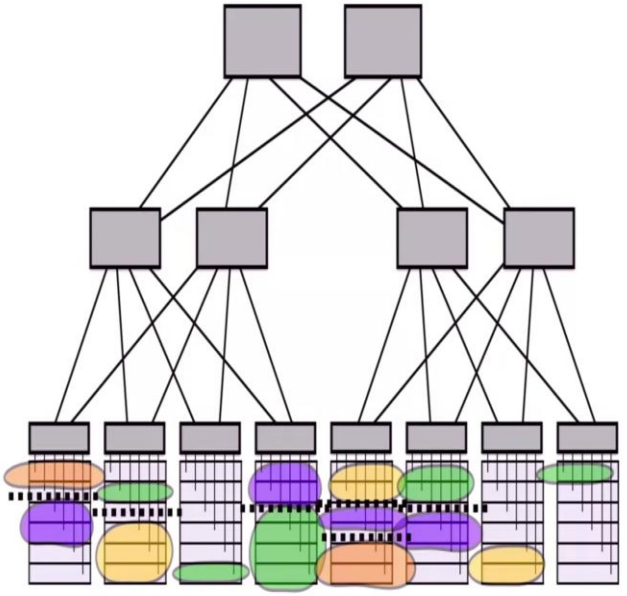
Key needs: Performance Uniformity

- **Performance Uniformity**
  - Wherever the VMs are, they will see the **same performance and latency.**
  - **Smaller units of compute** that we've divided our services into, and put them anywhere in the data center, and may be on the same physical machine.



Key needs: Security

- **Security:** Untrusting applications and users sitting right next to each other and can be inbound attacks. So to protect our tenants in the data center from each other in both the public data center as well as in the private cloud and you don't want them to have to trust each other.



  - **Micro-segmentation :** separation of different regions of a network.
  - Much finer grained division and control, of how data can flow.
  - Isolate or control just the data flow between pairs of applications, or tenants that should be actually allowed.

Key needs: Network semantics

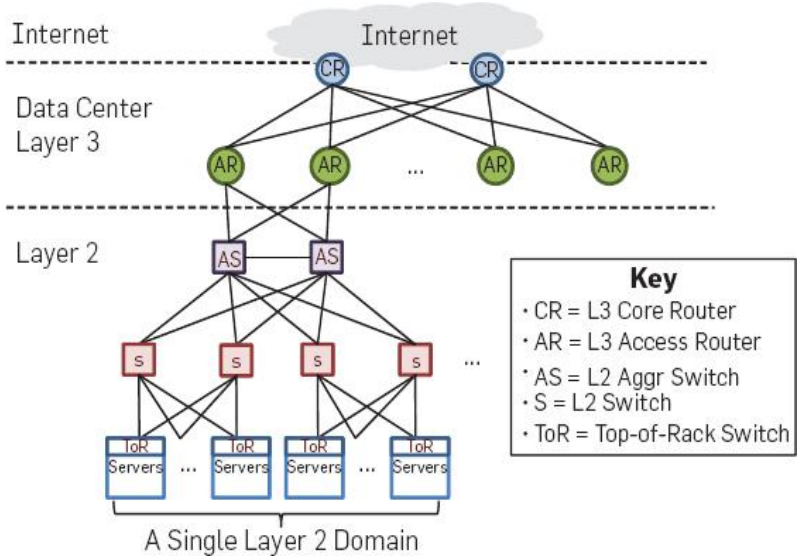
- **Network semantics:**
  - Match the functional service of a traditional data center
  - Not just Layer 3 routing services but also, Layer 2 services i.e. discovery services, multicast, broadcast etc. have to be supported.

Network Virtualization in Multi-tenant Data Centers Case Study: VL2

VL2: A Scalable and Flexible Data Center Network

Albert Greenberg
Srikanth Kandula
David A. Maltz
James R. Hamilton
Changhoon Kim
Parveen Patel
Navendu Jain
Parantap Lahiri
Sudipta Sengupta
Microsoft Research

[ACM SIGCOMM 2009]



Network Virtualization Case Study: VL2

Key Needs:

(i) Agility

(ii) Location independent addressing

(iii) Performance uniformity

(iv) Security

(v) Network Semantics

Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

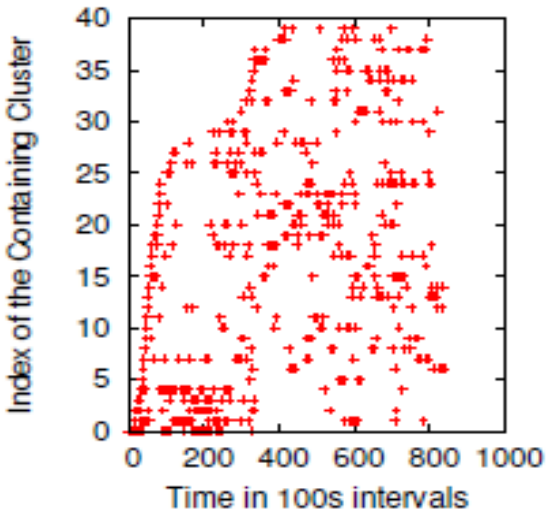
- Traffic volume between servers is 4 times larger than the external traffic

Rapidly-changing traffic matrices (TMs)

- i.e. Take traffic matrices in 100 second buckets and classify them into 40 categories of similar clusters of traffic matrices and see which of the clusters appear in the measurements
- So over time rapidly changing and no pattern to what the particular traffic matrix is.

Design result: Nonblocking fabric

- High throughput for any traffic matrices that respects server NIC rates.
- The fabric joining together all the servers, we don't want that to be a bottle neck.



[Greenberg et al.]

Motivating Environmental Characteristics

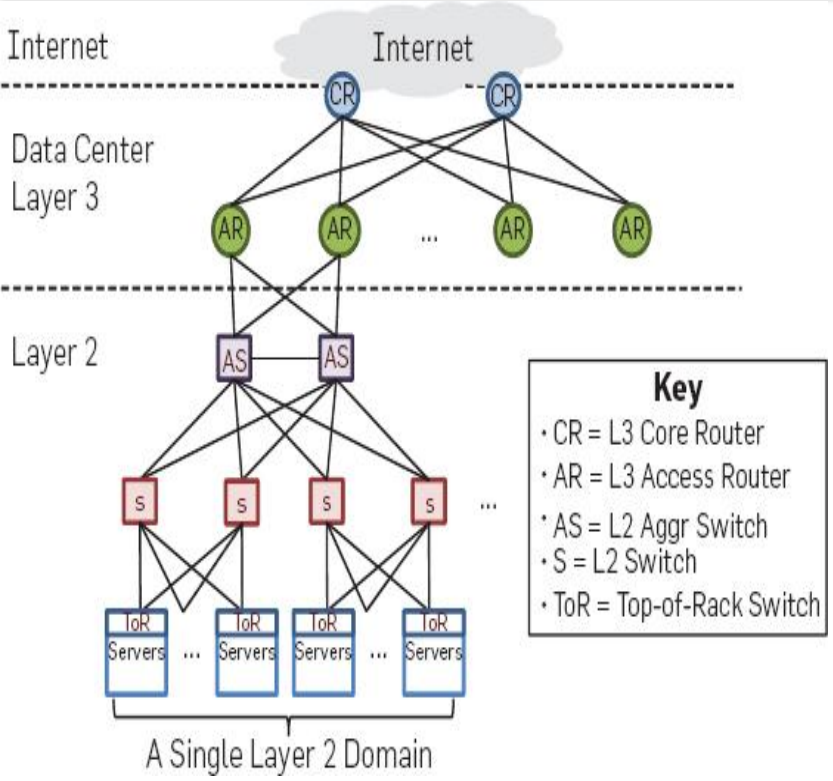
Failure characteristics:

- Analyzed 300K alarm tickets, 36 million error events from the cluster
- 0.4% of failures were resolved in over one day
- 0.3% of failures eliminated all redundancy in a device group (e.g. both uplinks)

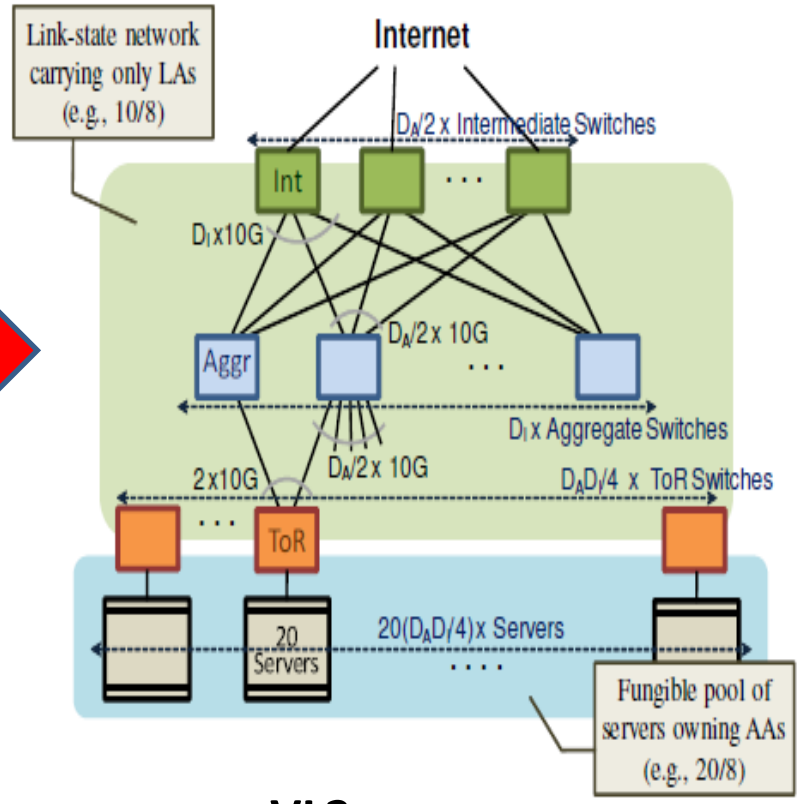
Design result: Clos topology:

- Particular kind of non blocking topology
- “Scale out” instead of “scale up”

VL2 physical topology



Traditional



VL2

An example Clos network between Aggregation and Intermediate switches provides a richly-connected backbone well suited for VLB. .e network is built with two separate address families—topologically significant Locator Addresses (LAs) and at Application Addresses (AAs).

Figures from Greenberg et al.

Routing in VL2

Unpredictable traffic

- Means it is difficult to adapt. So this leads us to a design that is what's called **oblivious routing**. It means that the path along which we send a particular flow does not depend on the current traffic matrix.

Design result: "Valiant Load Balancing"

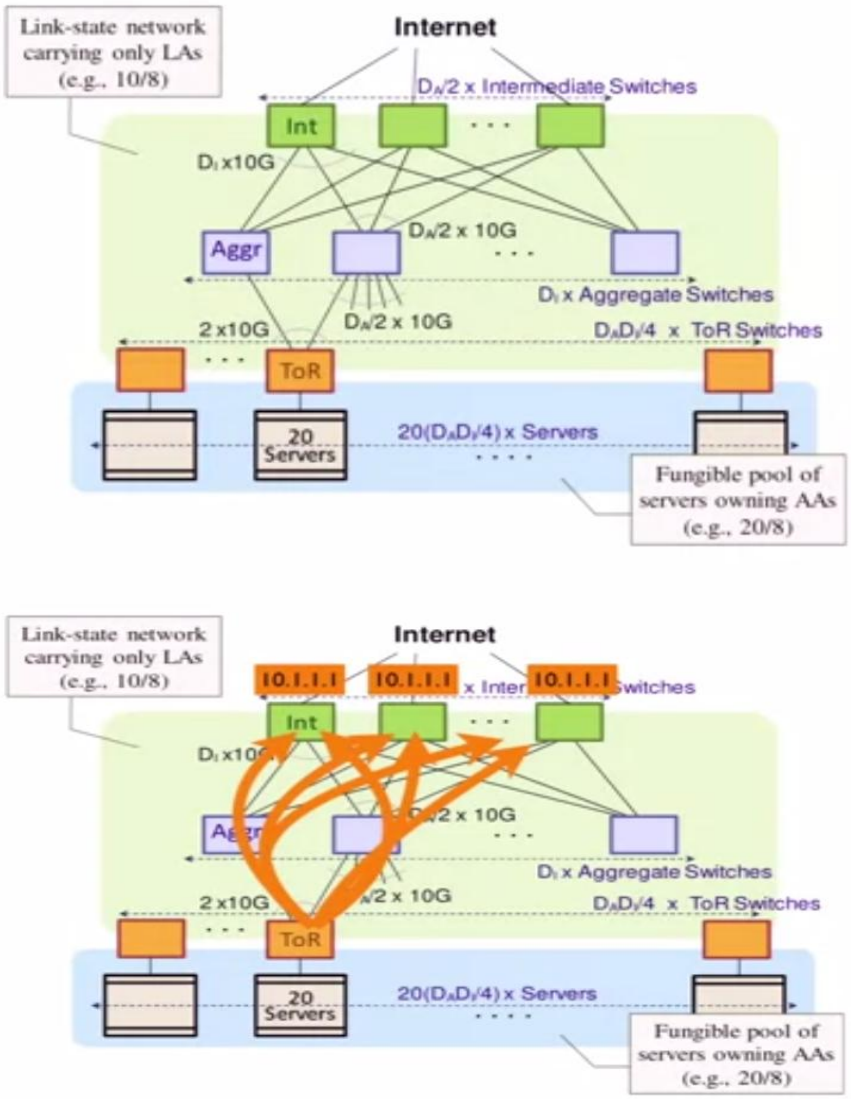
- For routing on hyper cubes take an arbitrary traffic matrix and make it look like a completely uniform even traffic matrix.
- Take flows and spreading evenly over all the available paths. Spread traffic as much as possible.
- Route traffic independent of current traffic matrix

Routing Implementation

Spreads arbitrary traffic pattern so it's uniform among top layer switches which are called intermediate switches.

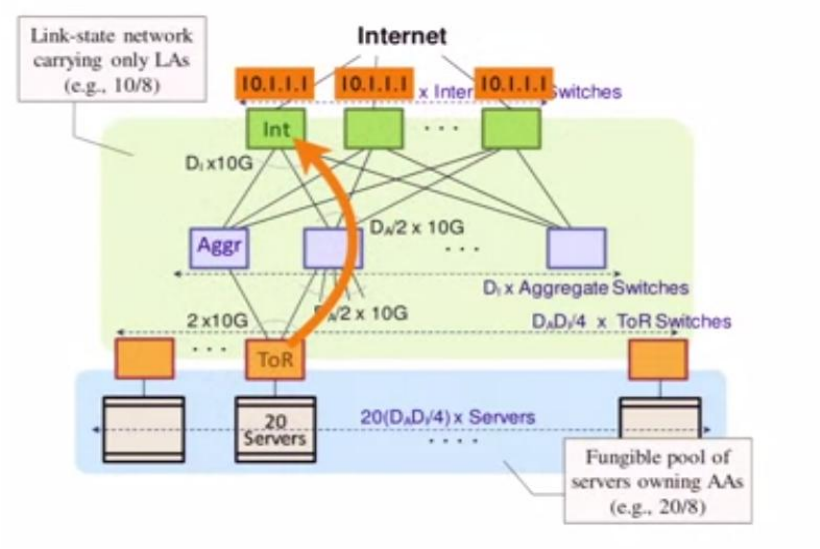
Now to do that what VL2 does is **it assigns those intermediate switches and any cast address.** The same any cast address for all of the switches.

So, then a top of rack switch can send to a random one by just using that single address. And if we are using **ECMP we will use a random one of those paths that are shortest.**

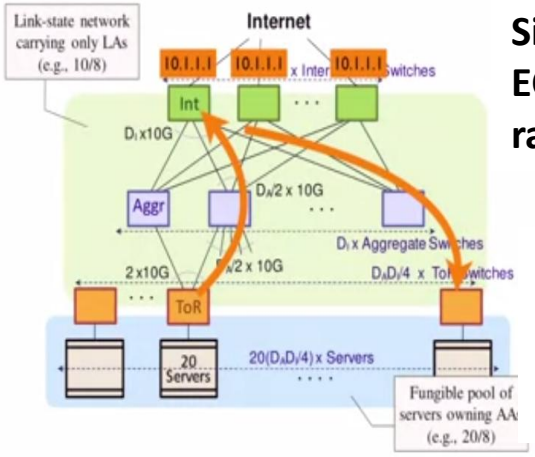


Routing Implementation

- As all of the paths are shortest because all of those intermediate switches are the same distance from the top of racks, So ECMP is going to give the full breadth of possible paths to any one of those switches, just by naming the single anycast address of all of the intermediates.



- **ECMP lets us select from one of those paths then one will be picked from any particular flow.** We send it to that intermediate switch. Now that outer anycast address is wrapping an inner header that actually has the destination address, in this design. So we'll forward it from there onto the destination.



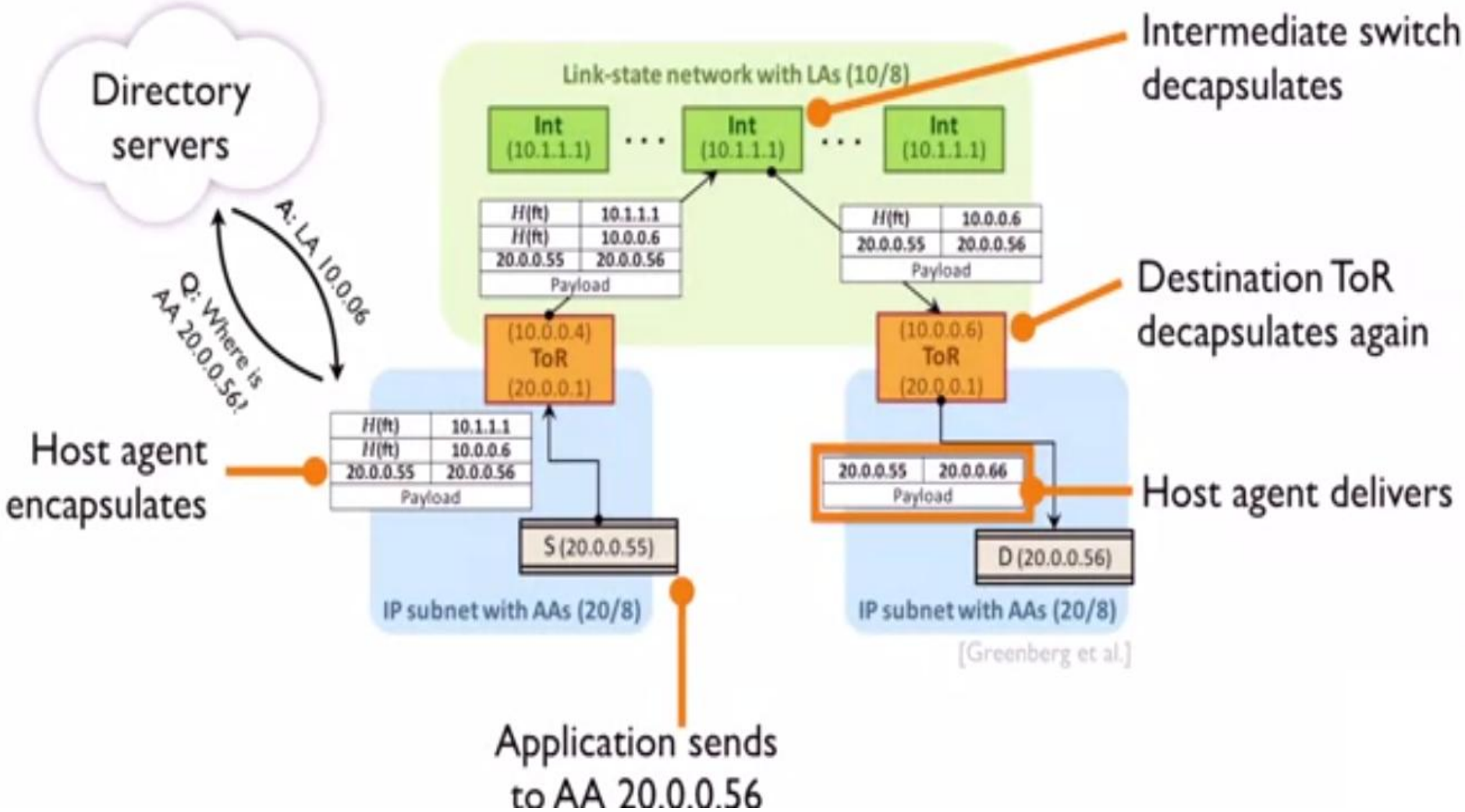
Similar effect to ECMP to each rack

Smaller forwarding table at most switches

Any service anywhere

App/Tenant layer	Application or tenant of the data center is going to see what's called application addresses. These are location independent, Same address no matter where the VM goes. And they're going to see the illusion of a single big Layer 2 switch connecting all of the application's VMs.
Indirection or Virtualization layer	Maintains a directory server that maps the application level addresses to their current locators. VL2 has agents that run on the server that will query the directory server and find that AA to LA mapping. And then when it sends a packet, it'll wrap the application address in the outer locator header
Physical network layer	Different set of IP addresses called locator addresses. Tied to topology used to route Layer 3 routing via OSPF

End-to-end Example



Did we achieve agility?

Location independent addressing

- AAs are location independent

L2 network semantics

- Agent intercepts and handles L2 broadcast, multicast

- Both of the above require “layer 2.5” shim agent running on host; but, concept transfers to hypervisor-based virtual switch

Did we achieve agility?

Performance uniformity:

- Clos network is nonblocking (non-oversubscribed)
- Uniform capacity everywhere.
- ECMP provides good (though not perfect) load balancing
- But performance isolation among tenants depends on TCP backing off to the rate that the destination can receive.
- Leaves open the possibility of fast load balancing

Security:

- Directory system can allow/deny connections by choosing whether to resolve an AA to a LA
- But, segmentation not explicitly enforced at hosts

Where's the SDN?

Directory servers: Logically centralized control

- Orchestrate application locations
- Control communication policy

Hosts agents: dynamic "programming" of data path

Network Virtualization
Case Study: NVP

Network Virtualization in Multi-tenant Datacenters

Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, and Rajiv Ramanathan, *VMware;* Scott Shenker, *International Computer Science Institute and the University of California, Berkeley;* Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang, *VMware*

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen

This paper is included in the Proceedings of the
11th USENIX Symposium on Networked Systems
Design and Implementation (NSDI '14).

April 2–4, 2014 • Seattle, WA, USA

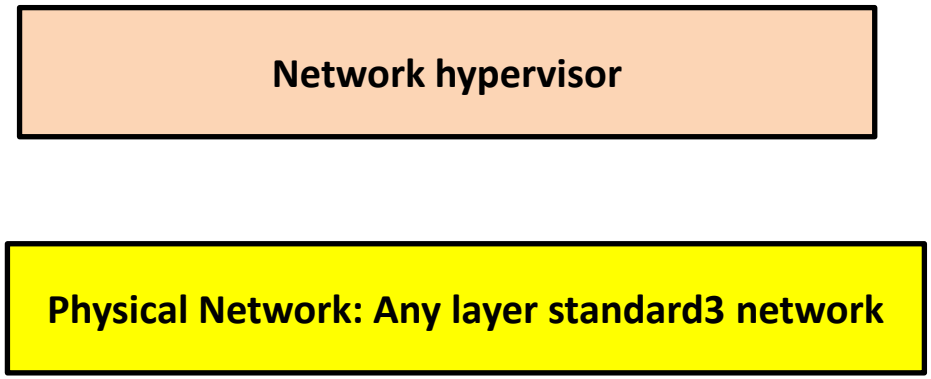
NVP Approach to Virtualization

- The network virtualization platform that was introduced in the paper **"Network virtualization in Multi-tenant Datacenters"** by Teemu Koponen et al. in NSDI 2014.

- And this comes out of a product developed by the Nicira startup that was acquired by VMware.

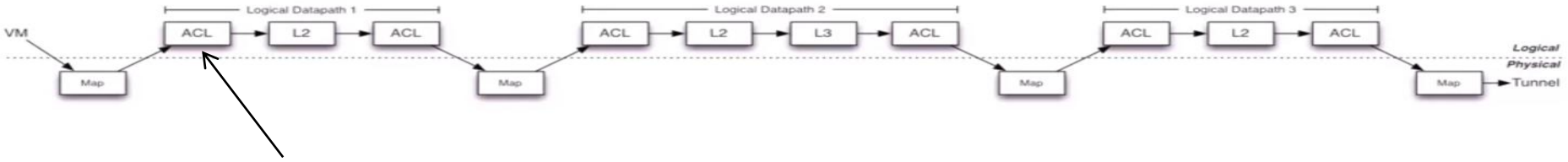
Service: Arbitrary network topology

- Replicate arbitrary topology
- Any standard layer 3 network
- Network hypervisor
- Virtual network tenants want to build



Virtual Network Service

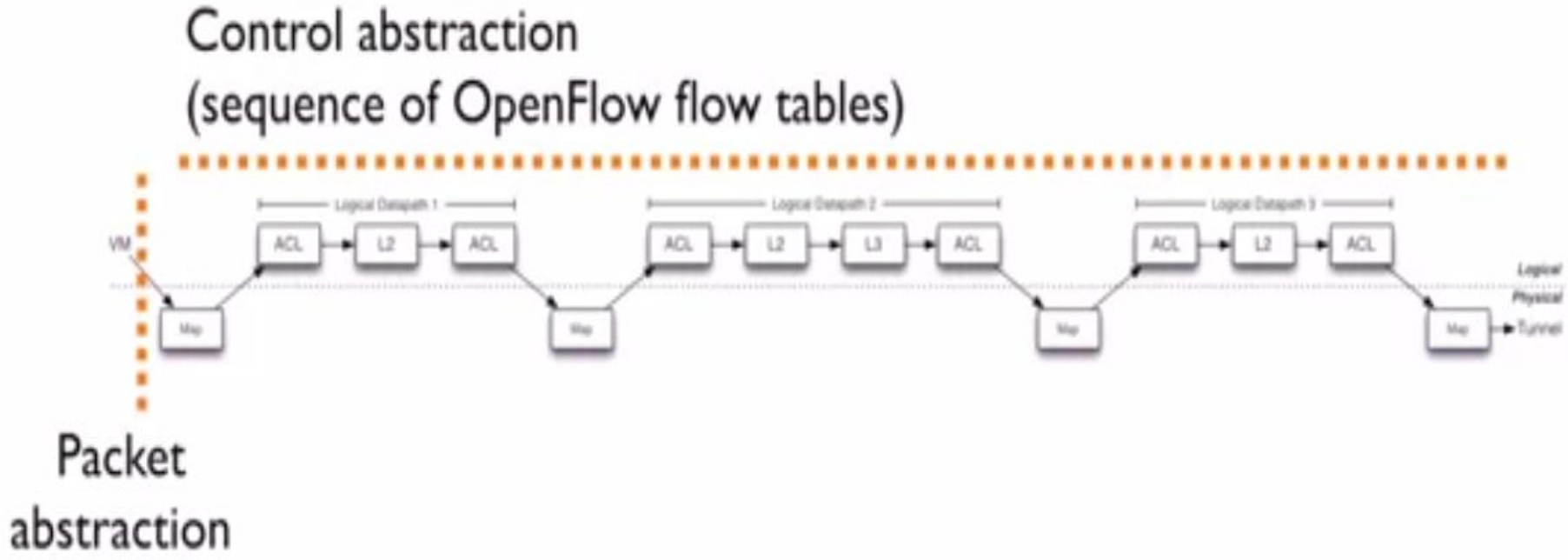
- Modeled as a sequence of data path elements that represent those switches.
- And these data path elements, each one of them is a OpenFlow forwarding table.
- It means the table will match on certain signature of packet headers and take certain resulting actions like dropping the packet, modifying certain fields in the packet, or forwarding the packet on.
- So the idea's that we can model the switching, and routing gear with the right sequences of the right OpenFlow tables that we set up.

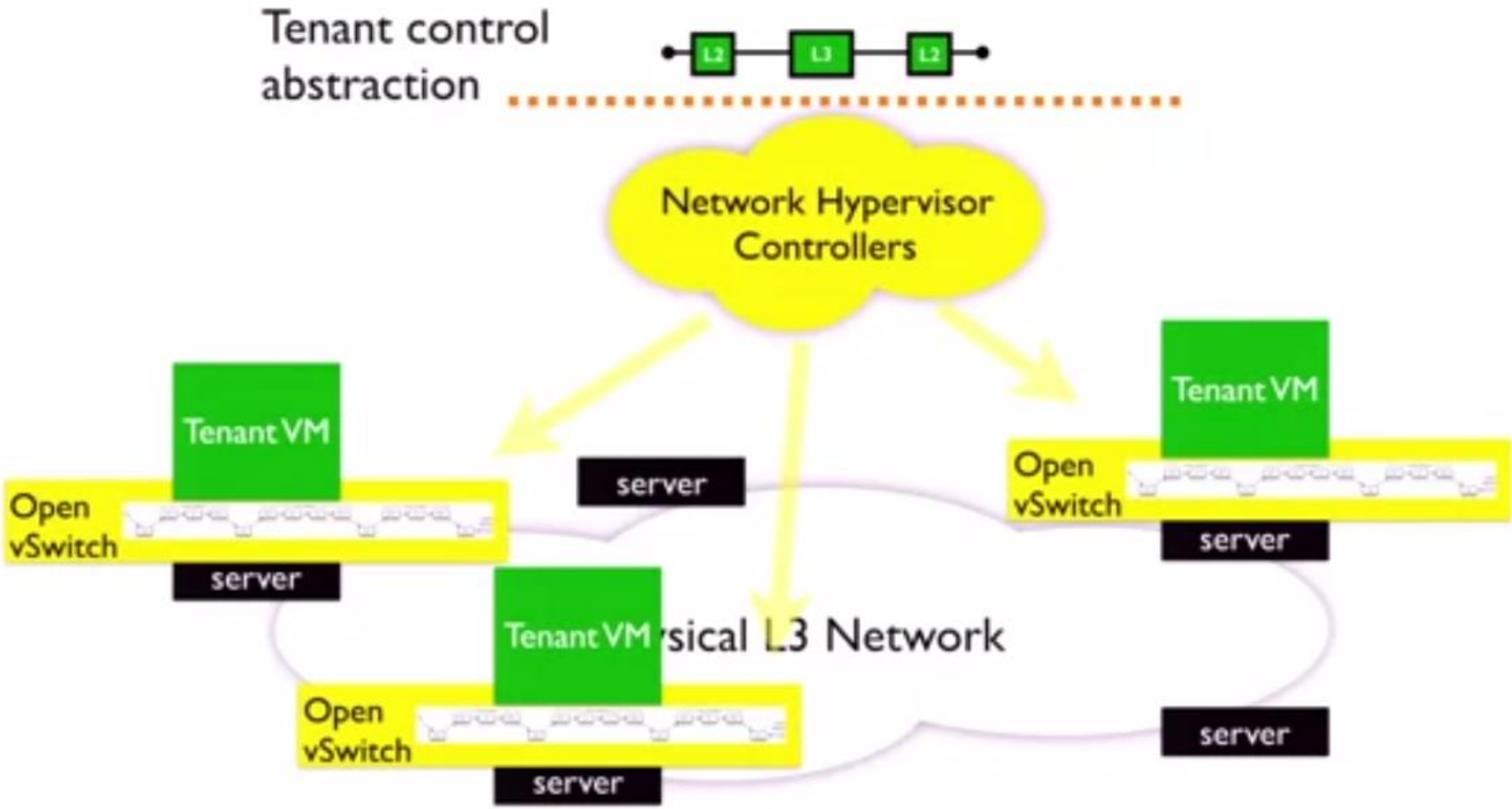


Access control Link Table

Virtual Network Service

- There's a **packet abstraction** where virtual machines are able to inject traffic into the virtual network.
- And there's a **control abstraction** where the tenant was able to define this entire virtual network pipeline, that sequence of OpenFlow flow tables.
- That's the interface that the tenant is given, at least the lowest level interface, that the tenant is given to be able to program their virtual network.




Tenant control abstraction
L2
L3
L2
Network Hypervisor Controllers
Tenant VM
Open vSwitch
server
Tenant VM
Open vSwitch
server
Tenant VM
Open vSwitch
server
sical L3 Network
server

Challenge: Performance

Large amount of state to compute

- Full virtual network state at every host with a tenant VM!
- $O(n^2)$ tunnels for tenant with n VMs
- Solution 1: Automated incremental state computation with nlog declarative language
- Solution 2: Logical controller computes single set of universal flows for a tenant, translated more locally by "physical controllers"

Challenge: Performance

Pipeline processing in virtual switch can be slow

- **Solution:** Send first packet of a flow through the full pipeline: thereafter, put an exact-match packet entry in the kernel

Tunneling interfaces with TCP Segmentation Offload (TSO)

- NIC can't see TCP outer header
- Solution: STT tunnels adds "fake" outer TCP header

Conclusion

- In this lecture, we have discussed the **need of software defined network, key ideas and challenges of software defined network.**

- We have also discussed the challenges in multi-tenant data centers i.e. **(i) Agility, (ii) Location independent addressing (iii) Performance uniformity, (iv) Security and (v) Network semantics.**

- Then we have discussed the **network Virtualization in multi-tenant data centers with a case study of VL2 and NVP**