Server Virtualization

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 server virtualization and the need of routing and switching for physical and virtual machines.

- Then we will discuss the two methods of virtualization: **(i) Docker based and (ii) Linux container based** and also address the problem of networking VMs and its **hardware based approach: SR-IOV, single-root I/O virtualization and software based approach: Open vSwitch**

Cloud Computing depends on Server Virtualization

- **In the context of server virtualization, how do we network the large number of VMs that might reside on a single physical server?** Cloud computing depends heavily on server virtualization for several reasons:

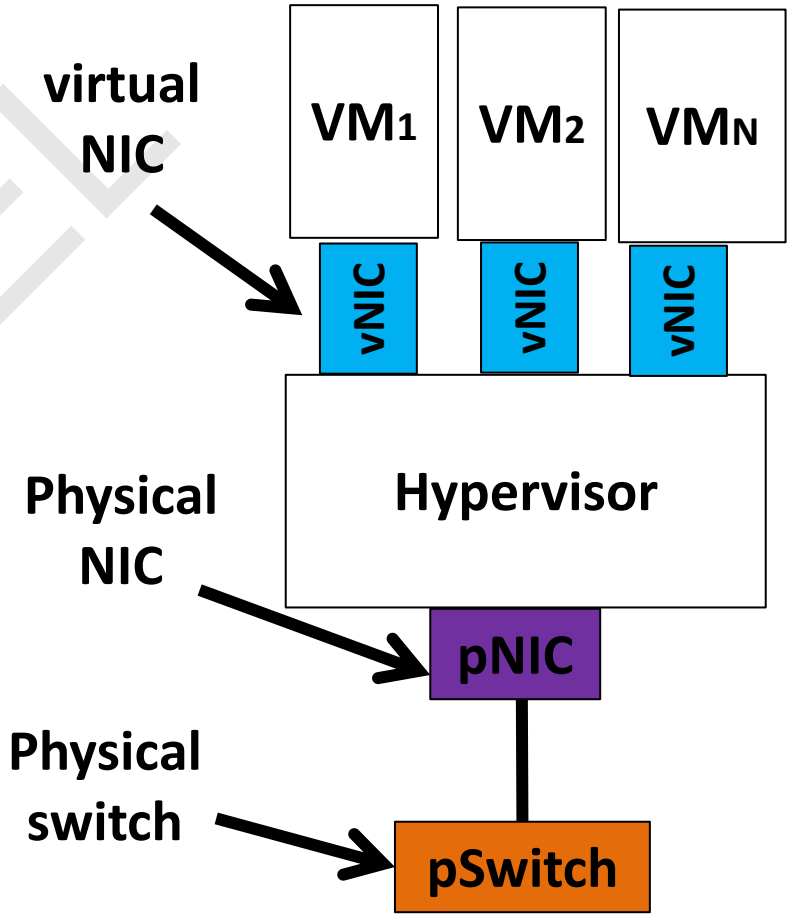
(i) **Sharing of physical infrastructure:** Virtual machines allow multiplexing of hardware with tens to 100s of VMs residing on the same physical server. Also, it allows rapid deployment of new services.

(ii) **Spinning up a virtual machine in seconds:** Spinning up a virtual machine might only need seconds compared to deploying an app on physical hardware, which can take much longer.

(ii) **Live VM migration:** Further, if a workload requires migration. For example, you do physical server requiring maintenance. This can be done quickly with virtual machines, which can be migrated to other servers without requiring interruption of service in many instances. Due to these advantages today, more endpoints on the network are virtual rather than physical.

Server Virtualization

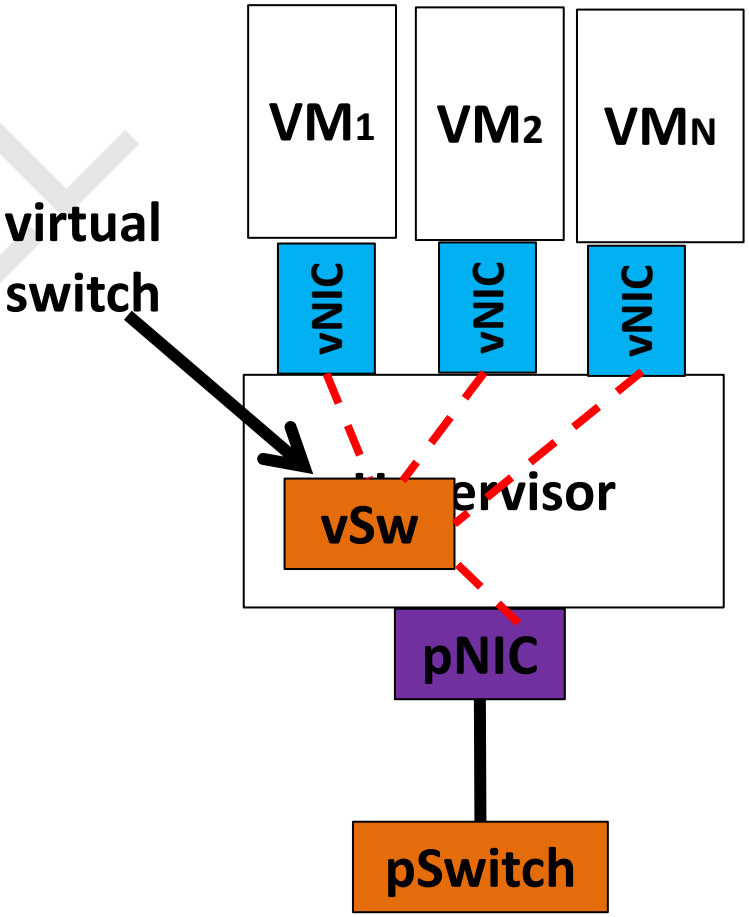
- The physical hardware is managed by a Hypervisor. This could be **Xen, KVM, or VMWare's ESXI,** or multiple such alternatives.
- On top of the Hypervisor runs several VMs in user space.
- The hypervisor provides an emulated view of the hardware to the VMs, which the VMs treat as their substrate to run a guest OS on.
- **Among other hardware resources the network interface card is also virtualized in this manner.** The hypervisor managing the physical NIC, is exposing virtual network interfaces to the VMs. **The physical NIC also connects the server to the rest of the network.**



Networking of VMs inside the Hypervisor

- **The hypervisor runs a virtual switch,** this can be a simple layer tool searching device, Operating in software inside the hypervisor.

- vSw is **connected to all the virtual NICs,** has them as the physical NIC, and moved packets between the VMs and the external network.



Alternate Methods of virtualization:

(i) Using Docker

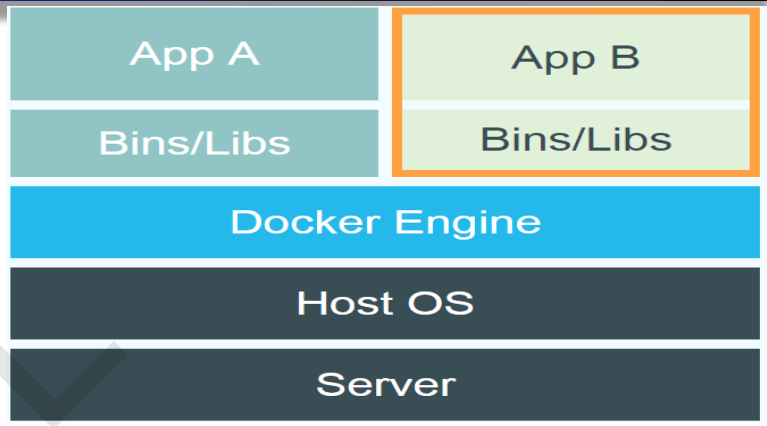
(ii) Using Linux containers

Docker

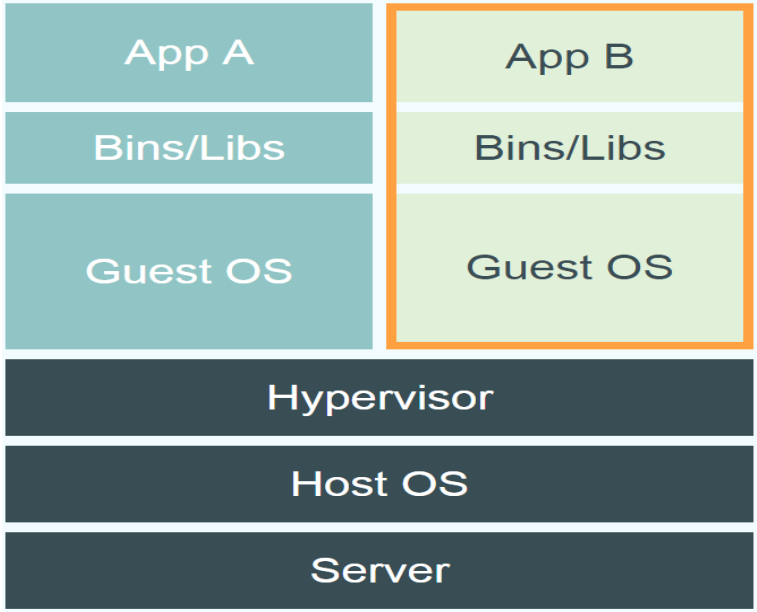
- **Docker is an open-source project that automates the deployment of applications inside software containers,** by providing an additional layer of abstraction and automation of OS–level virtualization on Linux.

- **The Docker Engine container comprises just the application and its dependencies.**

- **It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.** Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.



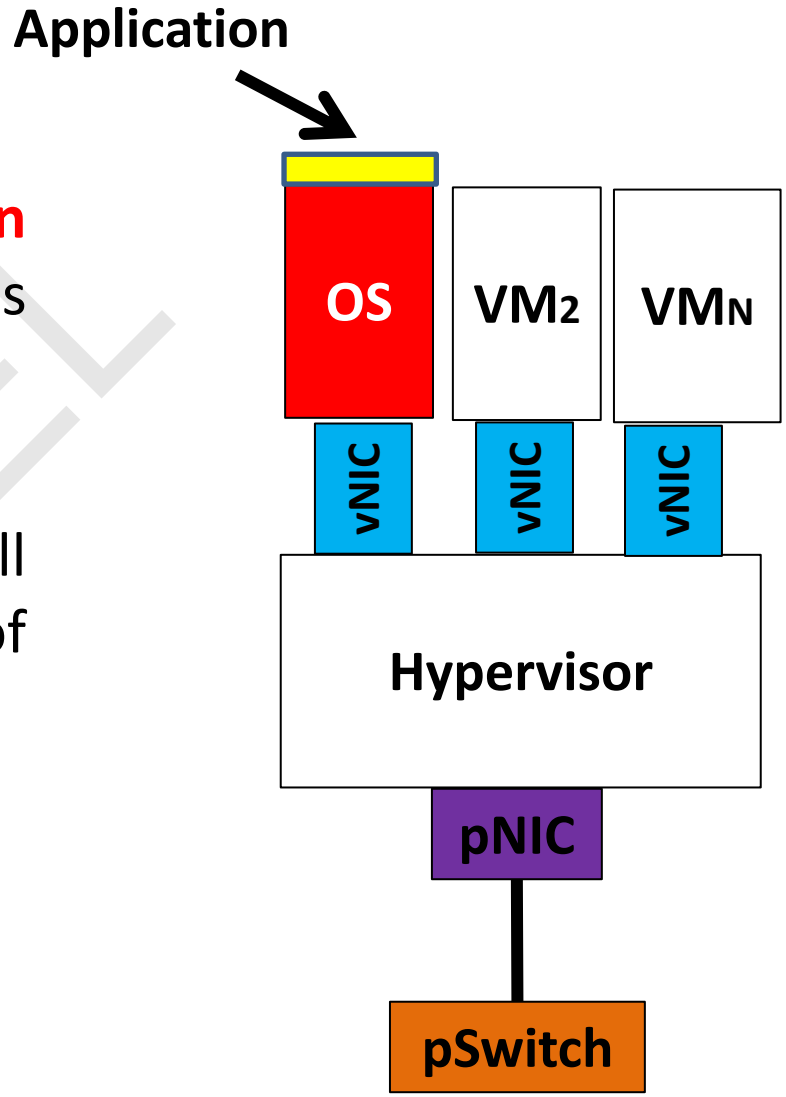
Docker



Virtual Machine

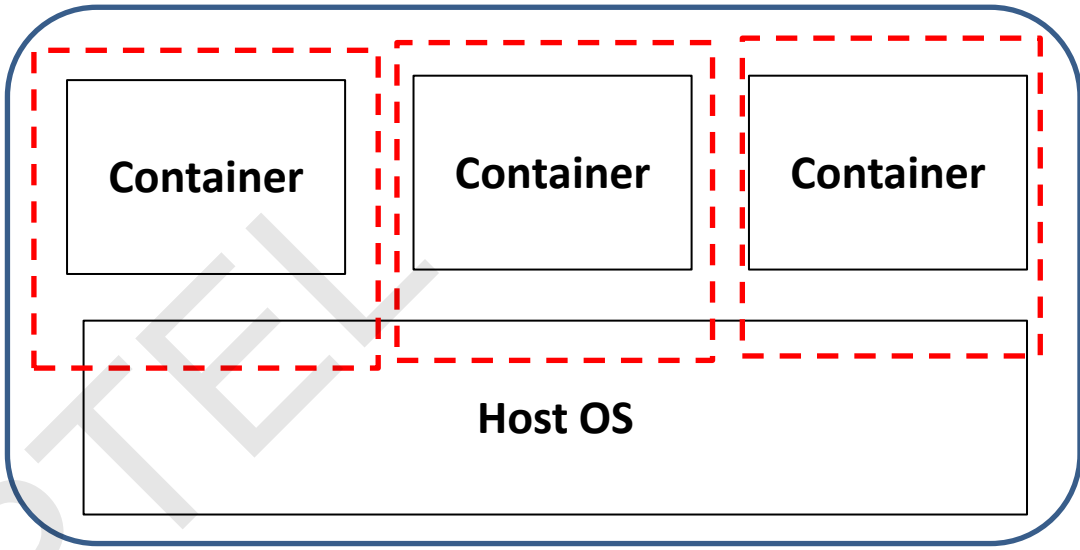
(i) Using VMs as Virtualization

- In this model, **each VM runs its own entire guest OS.** The application runs as a process inside this guest OS.
- This means that even running a small application requires the overhead of running an entire guest OS.

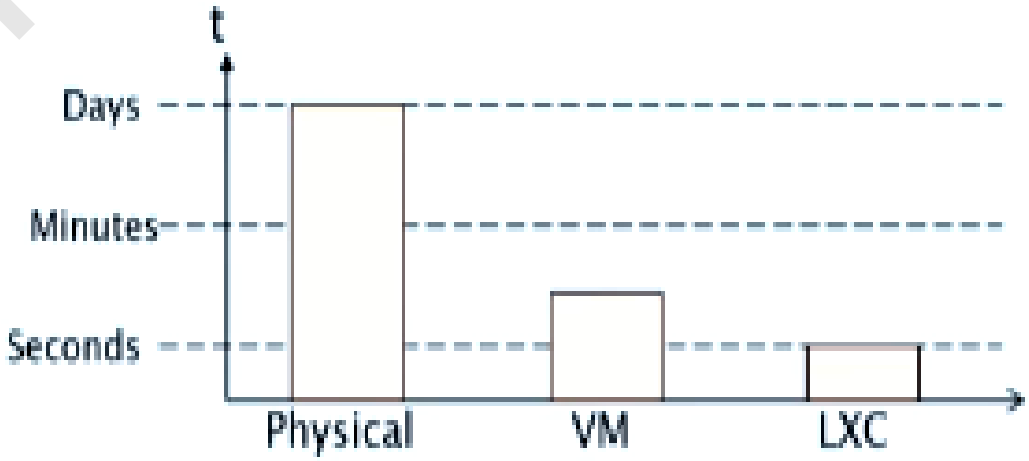


Linux containers as Virtualization

- Create an environment as close as possible to a standard Linux without separate kernel

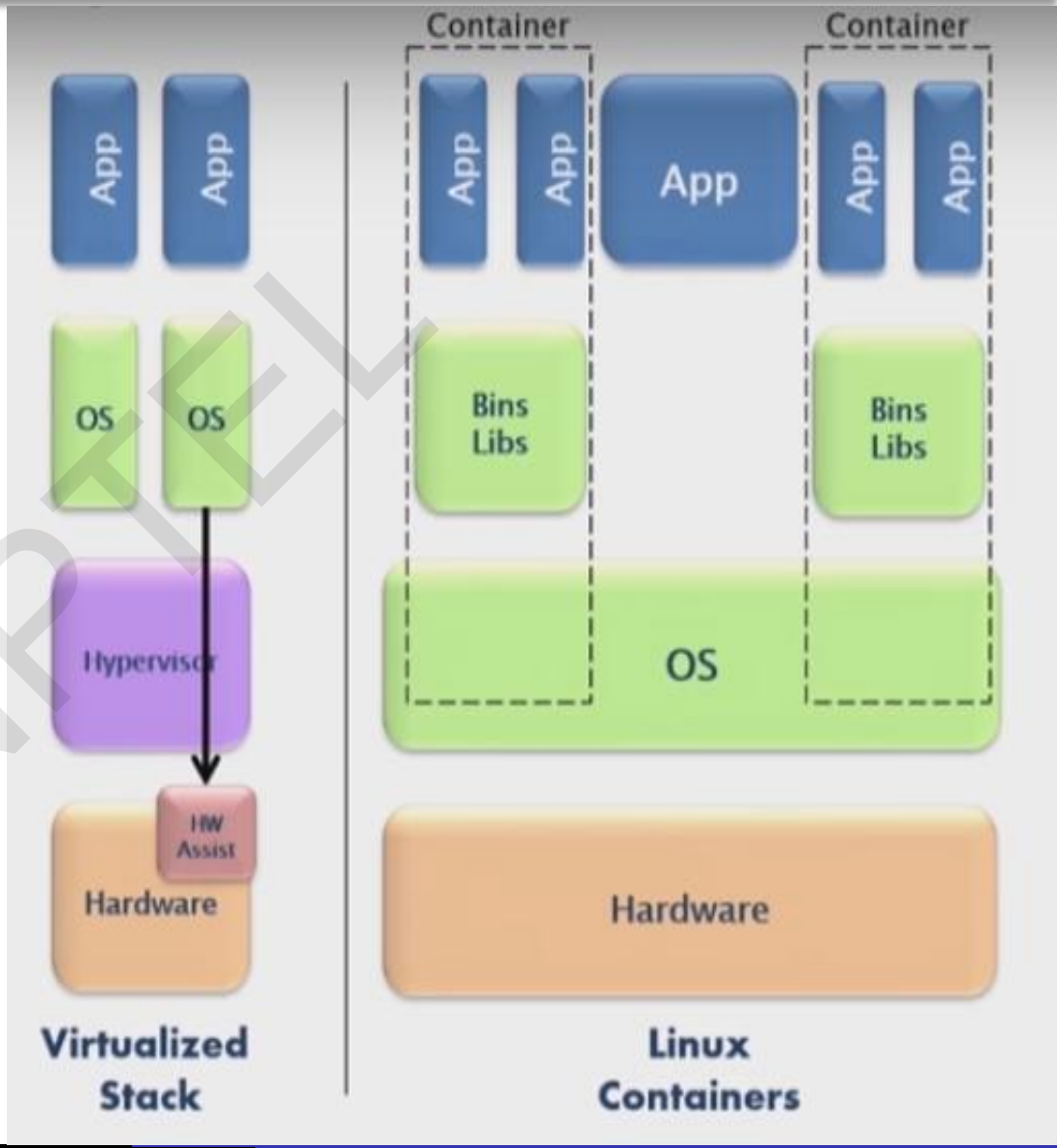


- **Uses kernel features for separation**
- **Near bare metal (physical hardware) performance**
- **Fast provisioning times (Building up/start-up time)**



Linux containers

- Containers run in host systems kernel
- Separated with policies
- Can use apps in host system
- Can install additional libraries and apps
- Portable between OS variants supporting linux container
- No overhead with hypervisor and guest OS kernel

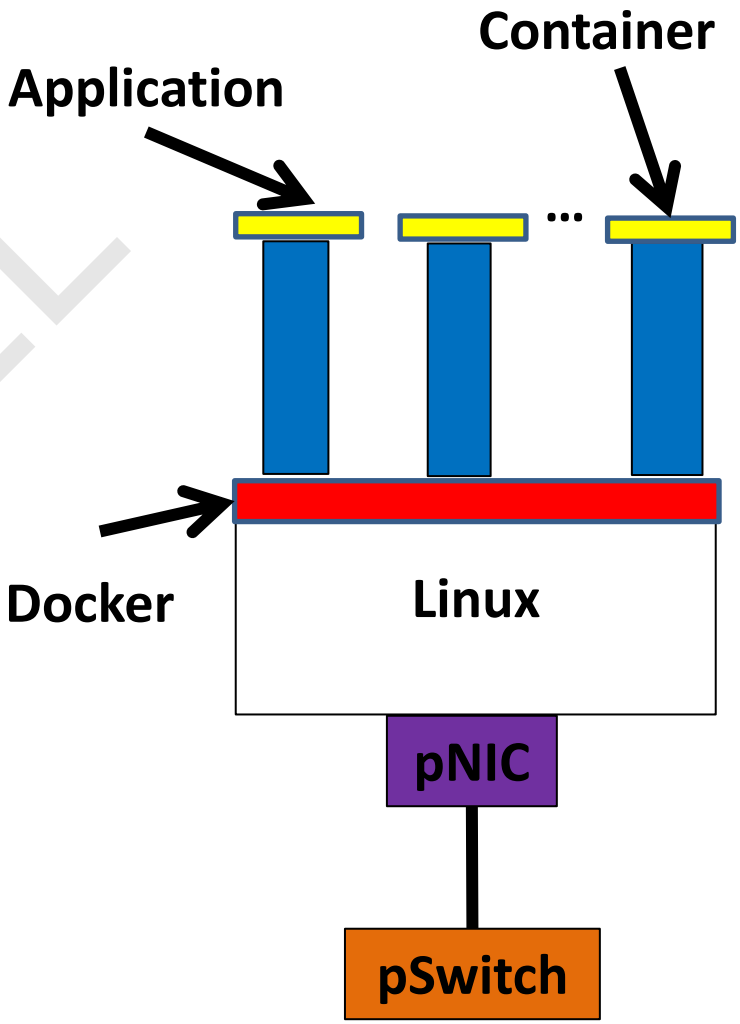


(ii) Using Linux containers

In this approach, an application together with its dependencies uses packages into a Linux container, which runs using the host machine's Linux stack and any shared resources.

Docker is simply a container manager for multiple such containers. Applications are isolated form each other by the use of separate namespaces. Resources in one application cannot be addressed by other applications. This yields isolation quite similar to VMs, but with a smaller footprint.

Further, containers can be brought up much faster than VMs. Hundreds of milliseconds as opposed to seconds or even tens of seconds with VMs.

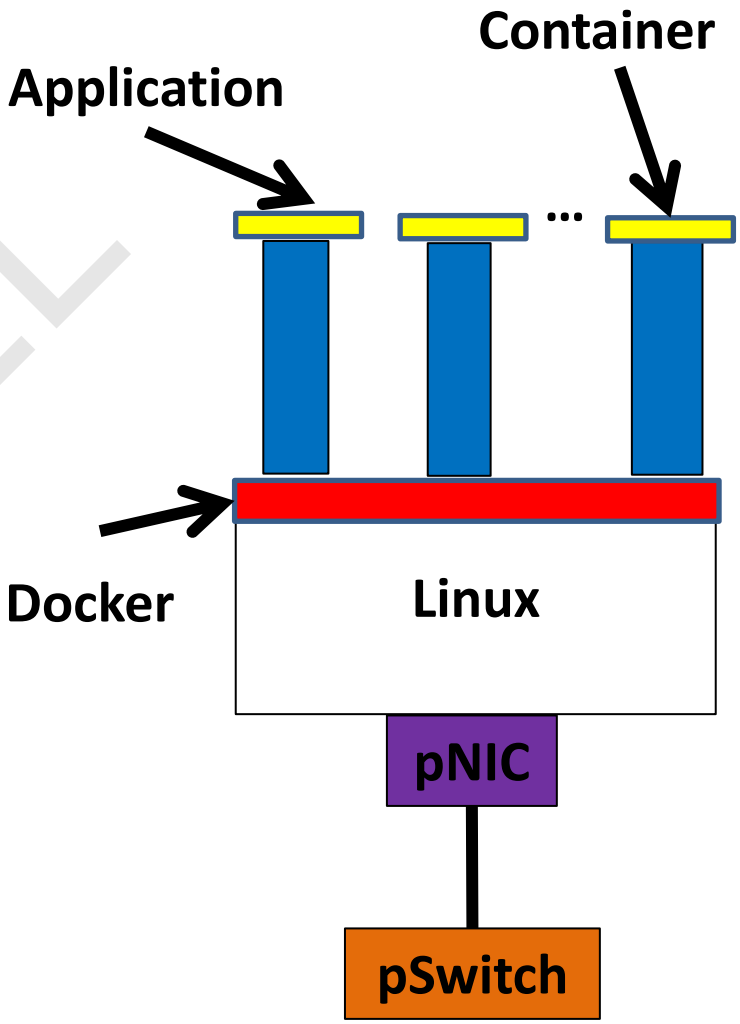


Networking with Docker

- **Each container is assigned a virtual interface.** Docker contains a virtual ethernet bridge connecting these multiple virtual interfaces and the physical NIC.

- **Configuring Docker and the environment variables decide what connectivity is provided.** Which machines can talk to each other, which machines can talk to the external network, and so on.

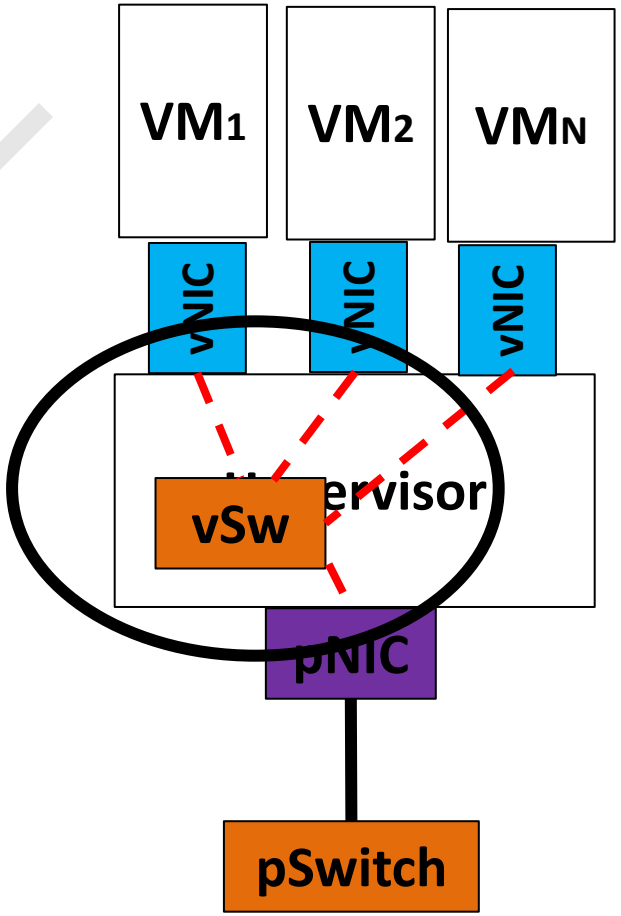
- **External network connectivity is provided through a NAT, that is a network address translator.**



Improving networking performance

- The **hypervisor runs a virtual switch to able to network the VM's** and **CPU is doing the work for moving the packets.**

CPU does the work!



Packet processing on CPUs

Flexible slow, CPU-expensive: Now packet processing on CPUs can be quite flexible because it can have general purpose forwarding logic. You can have packet filters on arbitrary fields run multiple packet filters if necessary, etc. But if done naively, this can also be **very CPU-expensive and slow.**

Packet forwarding: The packet forwarding entails: At 10Gbps line rates with the smallest packets, that's 84 Bytes. We only have an interval of 67ns before the next packet comes in on which we need to make a forwarding decision. Note that ethernet frames are 64 bytes, but together with the preamble which the tells the receiver that a packet is coming and the necessary gap between packets. The envelope becomes 84 bytes. For context a CPU to memory access takes tens of nanoseconds. So, 67 nanoseconds is really quite small.

Packet processing on CPUs

Need time for Packet I/O: Moving packets from the NIC buffers to the OS buffers, which requires CPU interrupts. Until recently a single X86 core couldn't even saturate a ten gigabits per second link. And this is without any switching required. This was just moving packets from the NIC to the OS. After significant engineering effort, packet I/O is now doable at those line rates. However for a software switch we need more.

Userspace overheads: If any of the switching logic is in userspace, one incurs the overhead for switching between userspace and kernel space.

Packet classification: Further, for switching we need to match rules for forwarding packets to a forwarding table. All of this takes CPU time. Also keep in mind that forwarding packets is not the main goal for the CPU. The CPU is there to be doing useful computation.

Approaches for Networking of VMs

There are two different approaches to address the problem of networking VMs:

(i) One, using specialized hardware:

- SR-IOV, single-root I/O virtualization

(ii) Other using an all software approach:

- Open vSwitch

(i) Hardware based approach

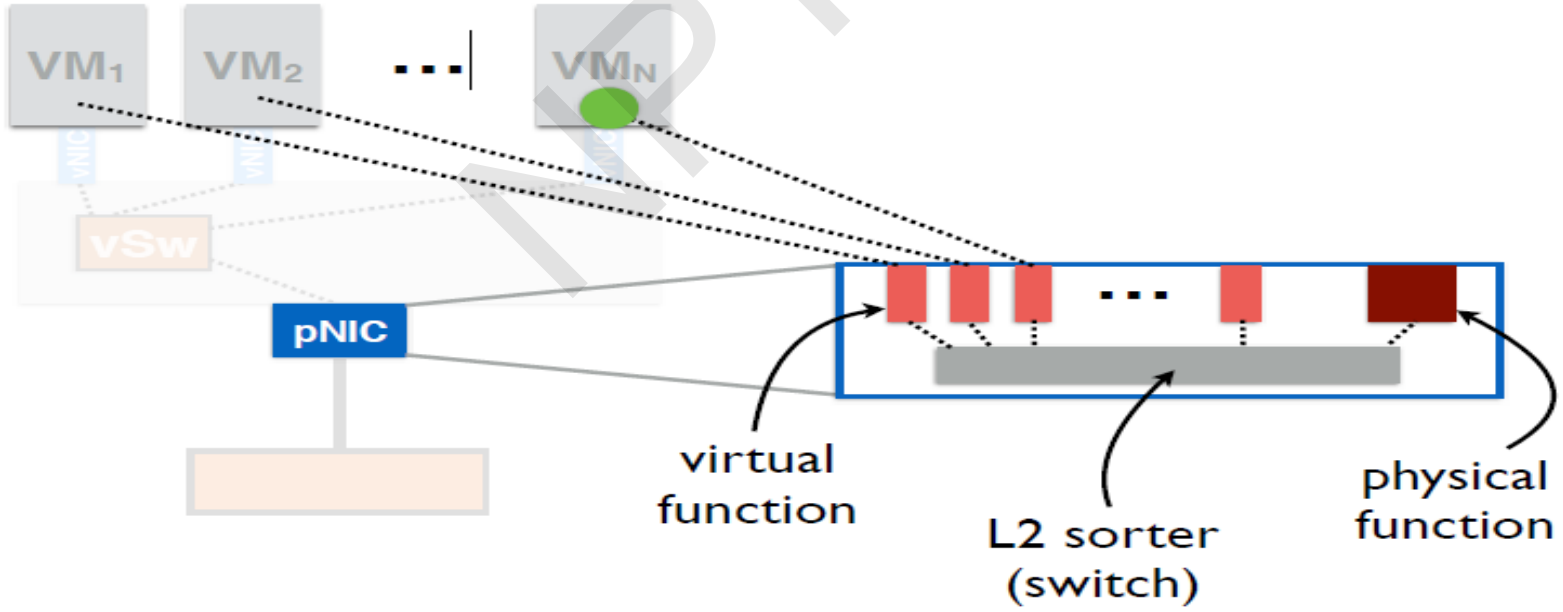
- **The main idea behind the hardware approach is that CPUs are not designed to forward packets, but the NIC is.**

- The naive solution would be to just give access to the VMs, to the NIC. But then problems arise. **How do you share the NIC's resources? How do you isolate various virtual machines?**

- **SR-IOV, single-root I/O virtualization**, based NIC provides one solution to this problem.

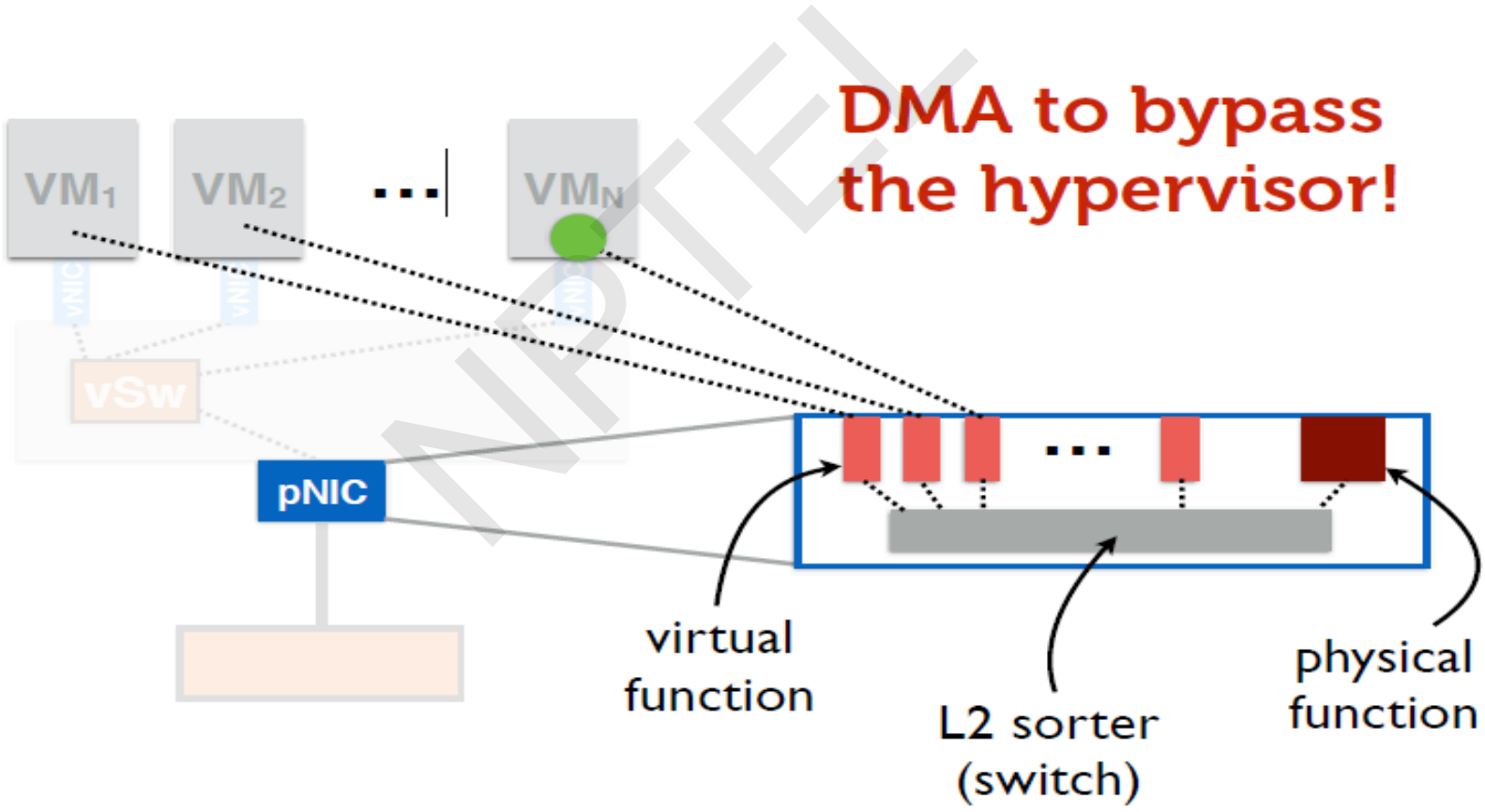
SR-IOV: Single-root I/O Virtualization

- **The SR-IOV, the physical link itself, supports virtualization in hardware.** So let's see inside this SR-IOV enabled network interface card.
- The NIC provides a physical function, which is just a standard ethernet port. In addition, it also provides several virtual functions which are simple queues that transmit and receive functionality.
- **Each VM is mapped to one of these virtual functions. So the VMs themselves get NIC hardware resources.**



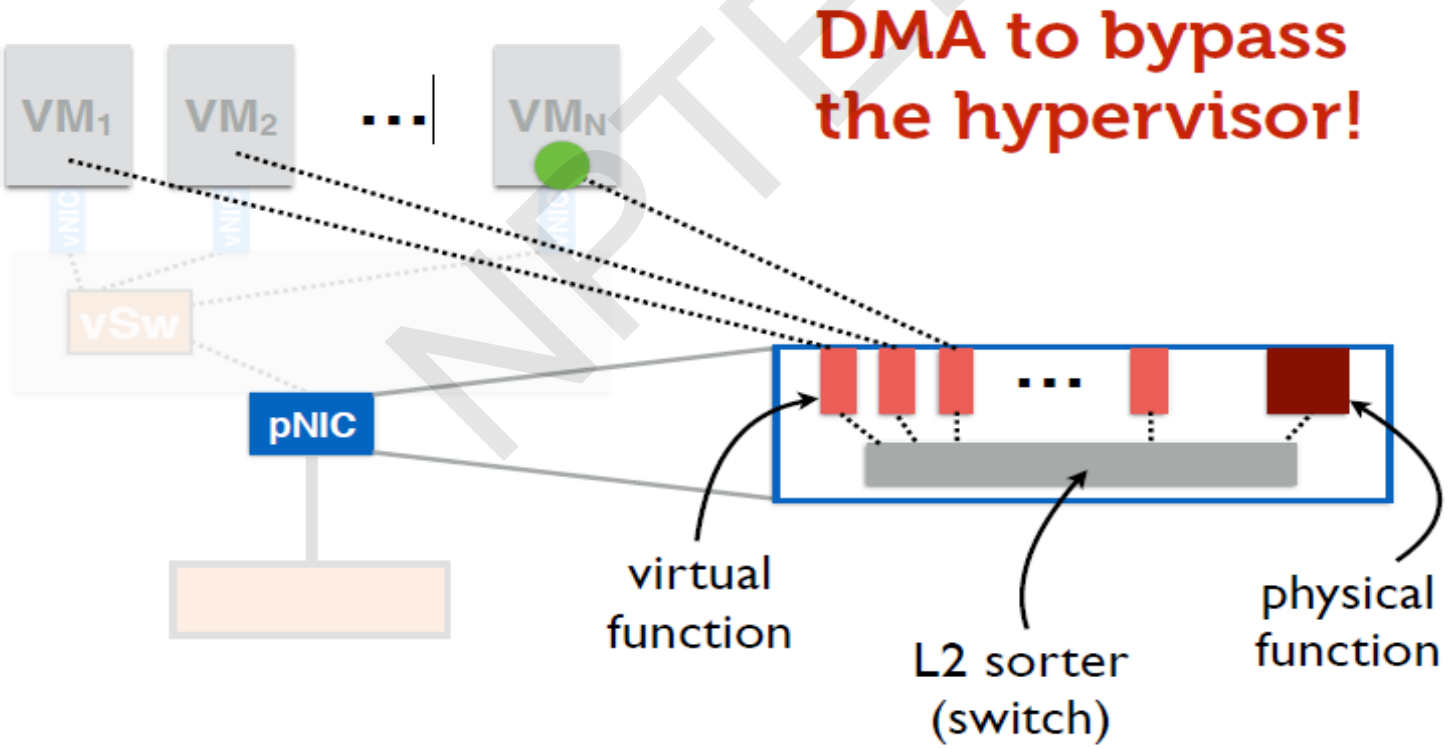
SR-IOV: Single-root I/O Virtualization

- **On the NIC there also resides a simple layer two, which classifies traffic into queues responding to these virtual functions**. Further, packets are moved directly from the net virtual function to the responding VM memory using **DMA (direct memory access).** This allows us to bypass the hypervisor entirely.



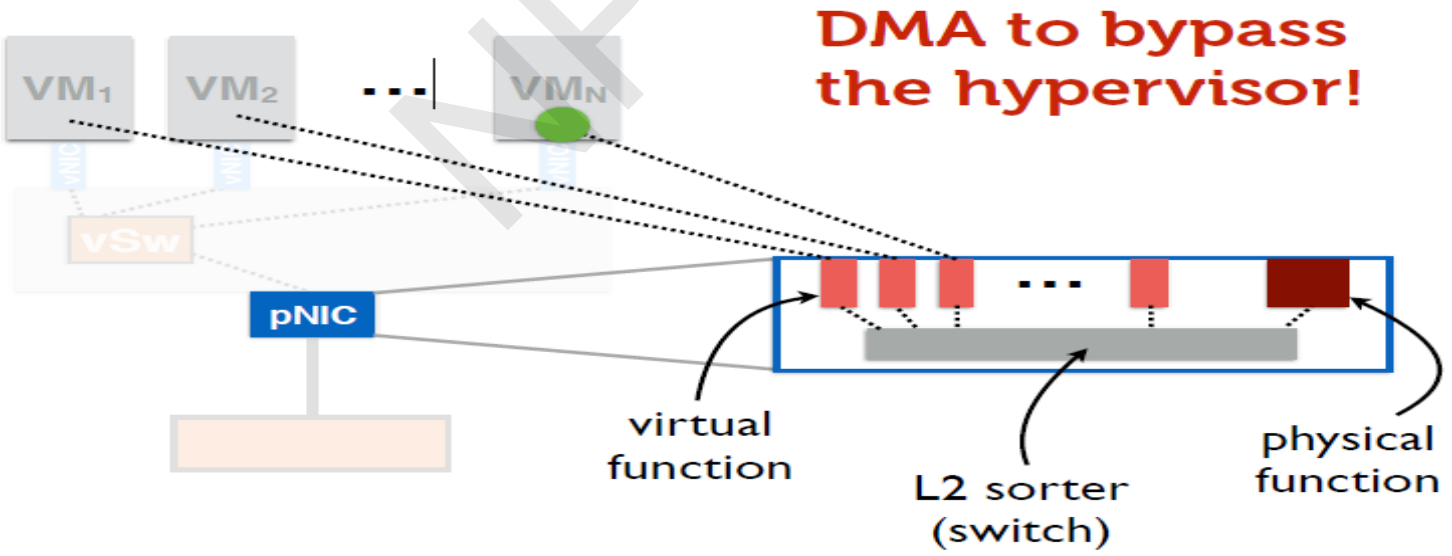
SR-IOV: Single-root I/O Virtualization

- **The hypervisor is only involved in the assignment of virtual functions to virtual machines.** And the management of the physical function, but not the data part for packets. The upshot to this is **higher through-put, lower latency and lower CPU utilization** and this give us close to native performance.



SR-IOV: Single-root I/O Virtualization

- There are downsides to this approach though. For one, **live VM migration becomes trickier**, because now you've tied the virtual machine to physical resources on that machine. The forwarding state for that virtual machine now resides in the layer two switch inside the NIC.
- **Second, forwarding is no longer as flexible.** We're relying on a layer two switch that is built into the hardware of the NIC. So we cannot have general purpose rules and we cannot be changing this logic very often. It's built into the hardware. **In contrast, software-defined networking (SDN) allows a much more flexible forwarding approach.**



(ii) Software based approach

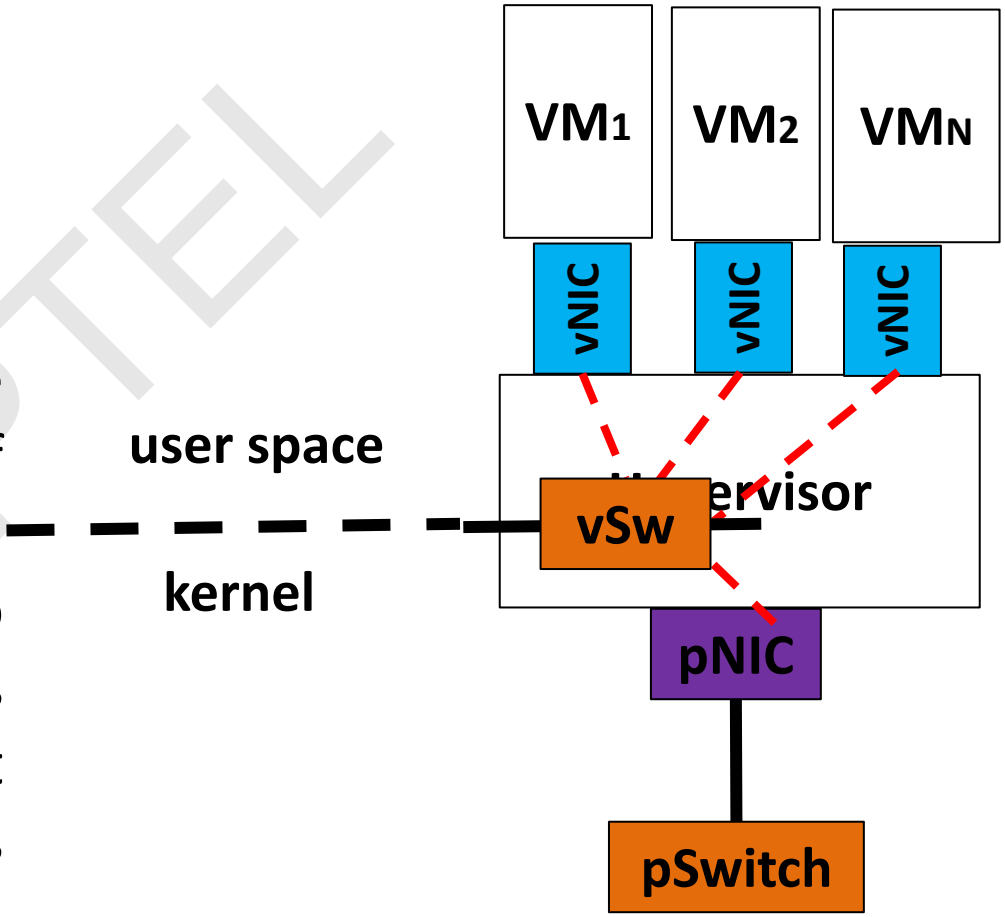
The software based approach that addresses:

(i) Much more flexible forwarding approach

(ii) Live VM migration

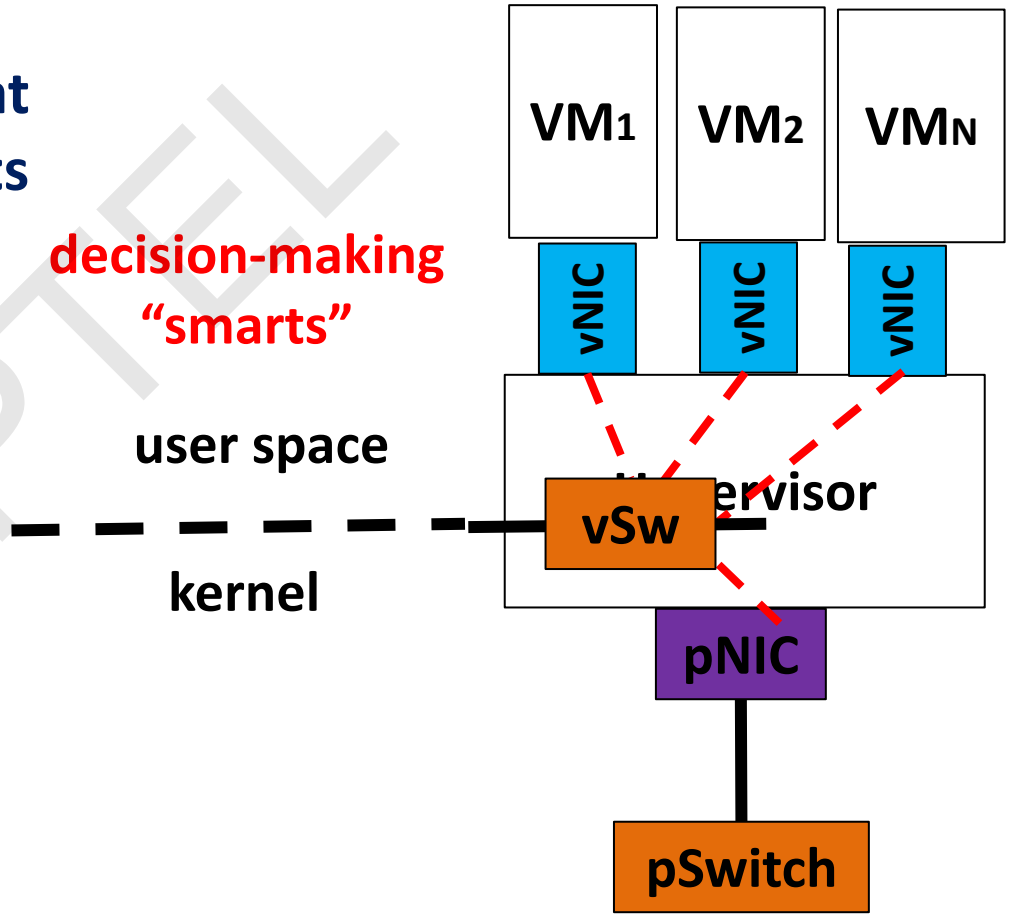
Open vSwitch

- Open vSwitch design goals are **flexible and fast-forwarding.**
- This necessitates a division between **user space** and **kernel space** task. One can not work entirely in the kernel, because of development difficulties. It's hard to push changes to kernel level code, and it's desirable to keep logic that resides in the kernel as simple as possible.



Open vSwitch

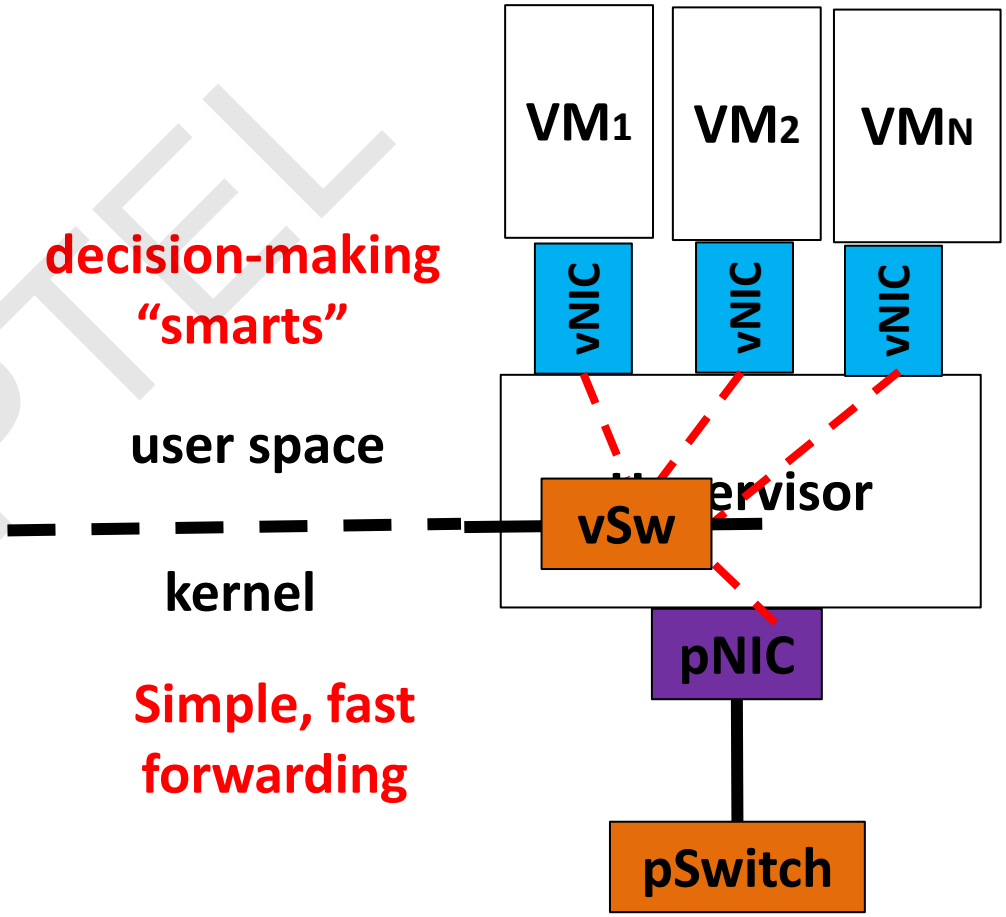
- So, the smarts of this approach, is the **switch routing decisions** lie in user space.
- This is where one **decides what rules or filters apply to packets of a certain type.** Perhaps based on network updates from other, possibly virtual, such as in the network. This behavior can also be programmed using **open flow.** **So, this part is optimized for processing network updates, and not necessarily for wire speed packet forwarding.**



Open vSwitch

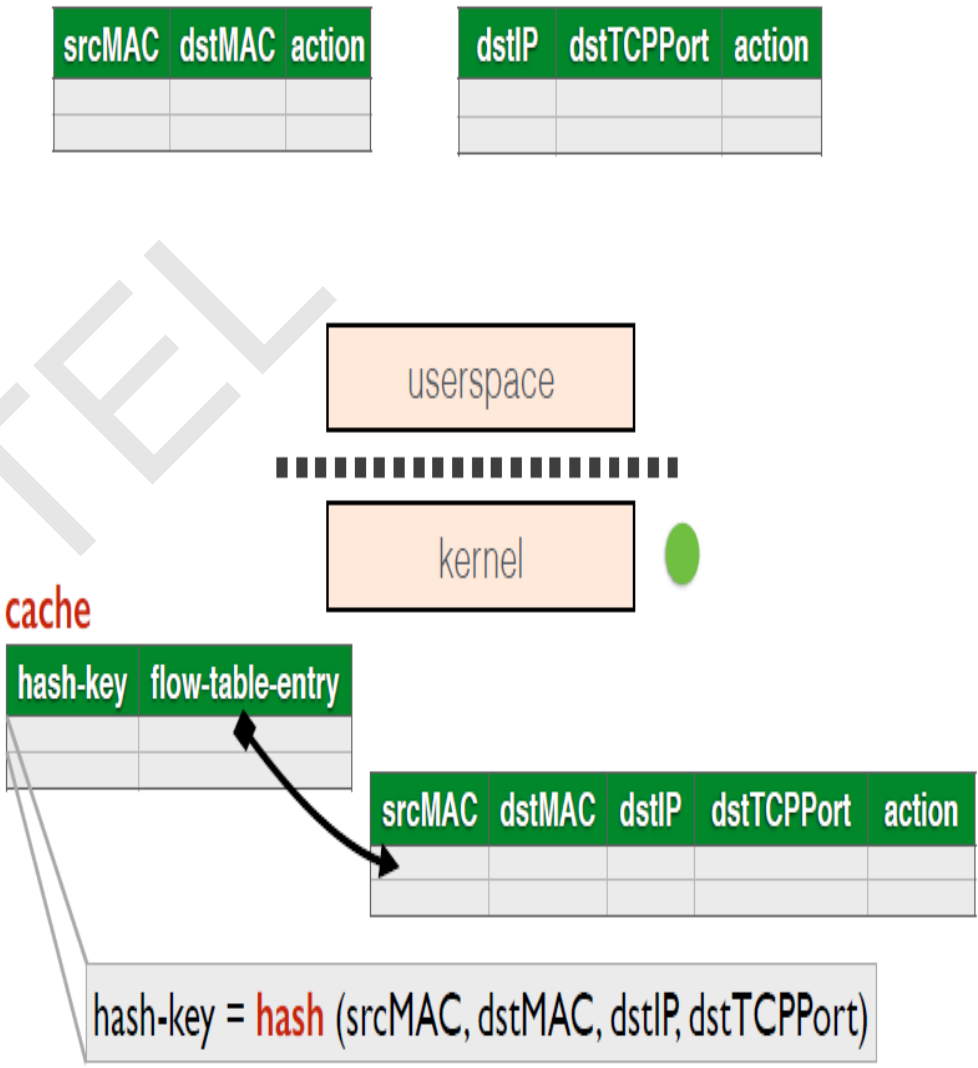
- **Packet forwarding**, on the other hand, is handled largely in the kernel, broadly.

- **Open vSwitch approach is to optimize the common case, as opposed to the worst case line rate requirements and caching will be the answer to that need.**



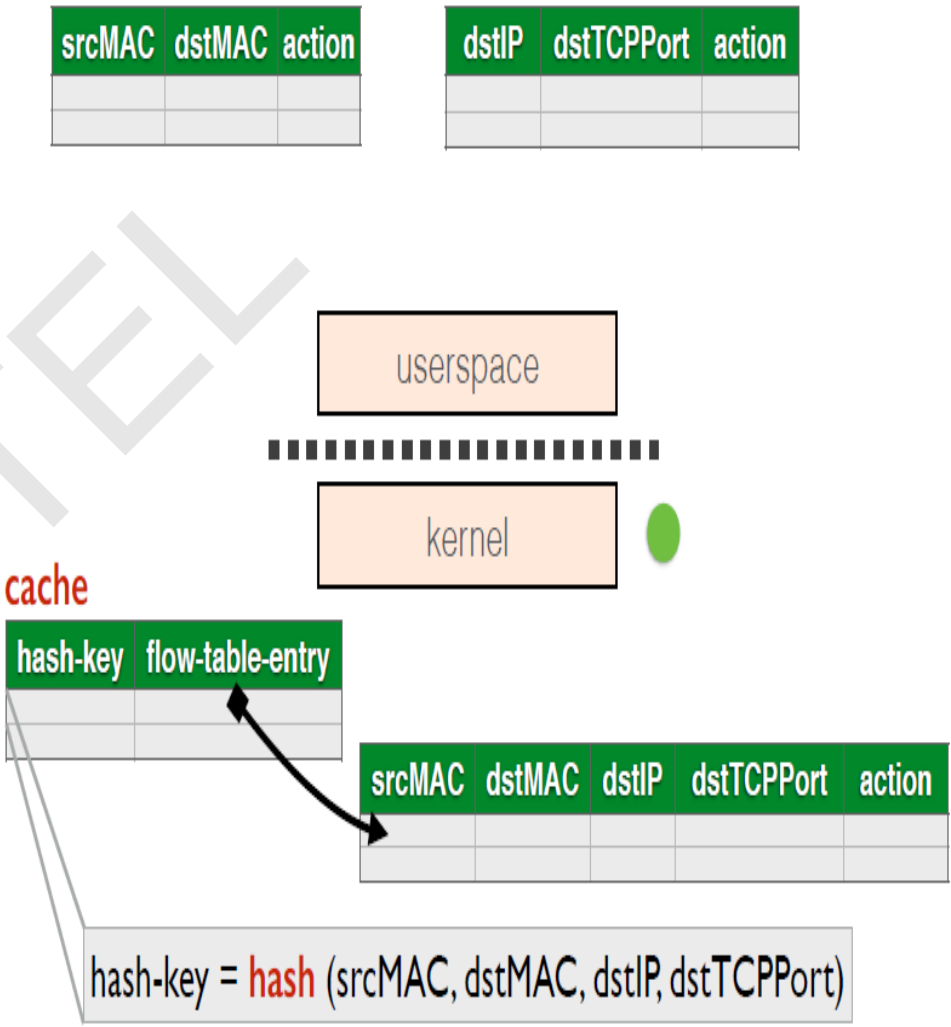
Inside Open vSwitch

- **The first packet of a flow goes to userspace here several different packet classifiers may be consulted.** Some actions may be based on MAC addresses and some others might depend on TCP PORTs, etc.
- The **highest priority matching action** across these different classifiers will be used to forward the packet.
- **Once a packet is forwarded, a collapsed rule used to forward that packet is installed in the kernel. This is a simple classifier with no priorities.** The following packets of this flow will never enter user space, seeing only the kernel level classifier.



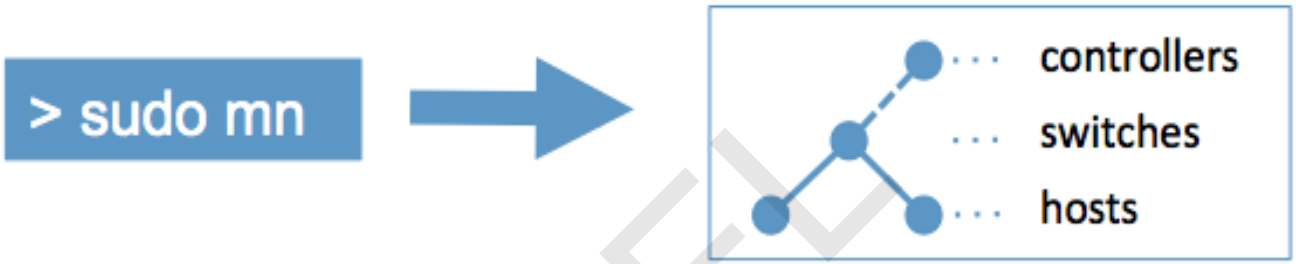
Inside Open vSwitch

- The problem though is when it's still running a packet classifier in the kernel in software. What this means is for **every packet that comes in, you are searching in this table for the right entry that matters and using that entry for forward the packet. This can be quite slow.**
- **Open vSwitch solves this problem is to create a simple hash table based cache into the classifier.** So instead of looking at this entire table and finding the right rule. You'll hash what fields are used to match packet, and the hash key is now your pointer to the action that needs to be taken. And, these hash keys and their actions can be cache.



Introduction to Mininet

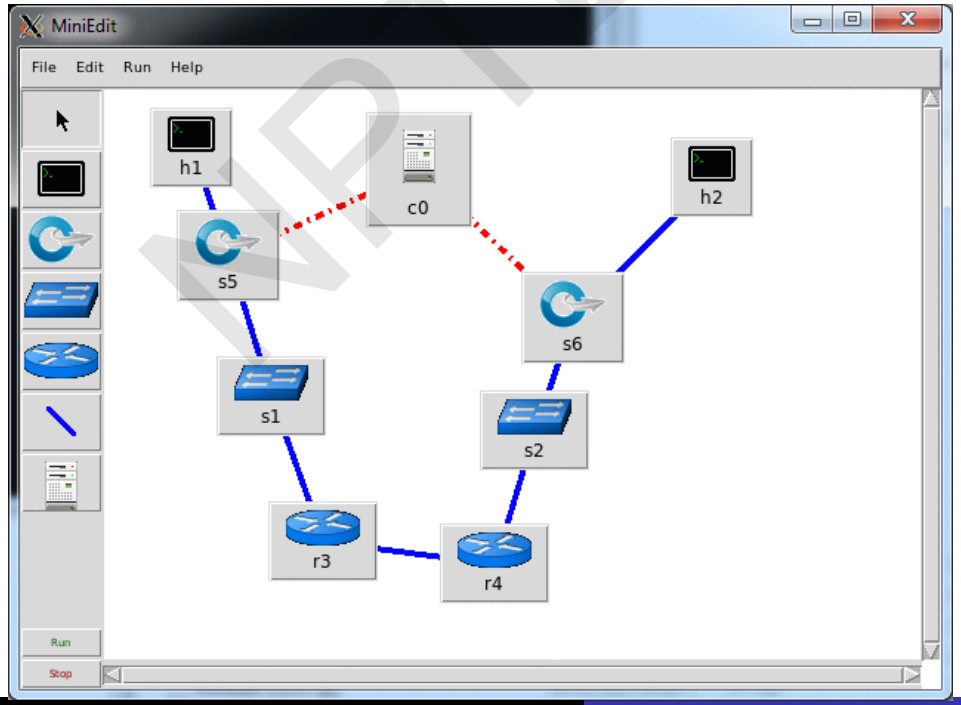
- Mininet creates a **realistic virtual network, running real kernel, switch and application code,** on a single machine (VM, cloud or native), in seconds, with a single command:



- It is a **network emulator** which creates realistic virtual network
  - Runs real kernel, switch and application code on a single machine
  - Provides both Command Line Interface (CLI) and Application Programming Interface (API)
    - **CLI: interactive commanding**
    - **API: automation**
  - **Abstraction**
    - Host: emulated as an OS level process
    - Switch: emulated by using software-based switch
      - **E.g., Open vSwitch, SoftSwitch**

Mininet Applications

- **MiniEdit (Mininet GUI)**
  - A GUI application which eases the Mininet topology generation
  - Either save the topology or export as a Mininet python script
- **Visual Network Description (VND)**
  - A GUI tool which allows automate creation of Mininet and OpenFlow controller scripts



Important Links for Mininet

- mininet.org
- github.com/mininet
- github.com/mininet/mininet/wiki/Documentation
- reproducingnetworkresearch.wordpress.com

Comparison

- The **hardware based approach** the **SR-IOV** takes **sacrifices flexibility and forwarding logic** for line rate performance in all scenarios. Virtually hitting native performance.

- While the **software based approach** **Open vSwitch**, the compromise made is to avoid targeting worst case performance, and **focusing on forwarding flexibility**.

References

- W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "**An updated performance comparison of virtual machines and Linux containers**," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, 2015, pp. 171-172.

- Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado, "**The design and implementation of open vSwitch**" In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15). USENIX Association, Berkeley, CA, USA, 2015, pp. 117-130.

Conclusion

- In this lecture, we have discussed **server virtualization and also discuss the need of routing and switching for physical and virtual machines.**

- We have discussed two methods of virtualization: **(i) Docker based and (ii) Linux container based.**

- To address the problem of networking VMs, two approaches are discussed: **(i) One, using specialized hardware: SR-IOV, single-root I/O virtualization and (ii) Other using an all software approach: Open vSwitch**

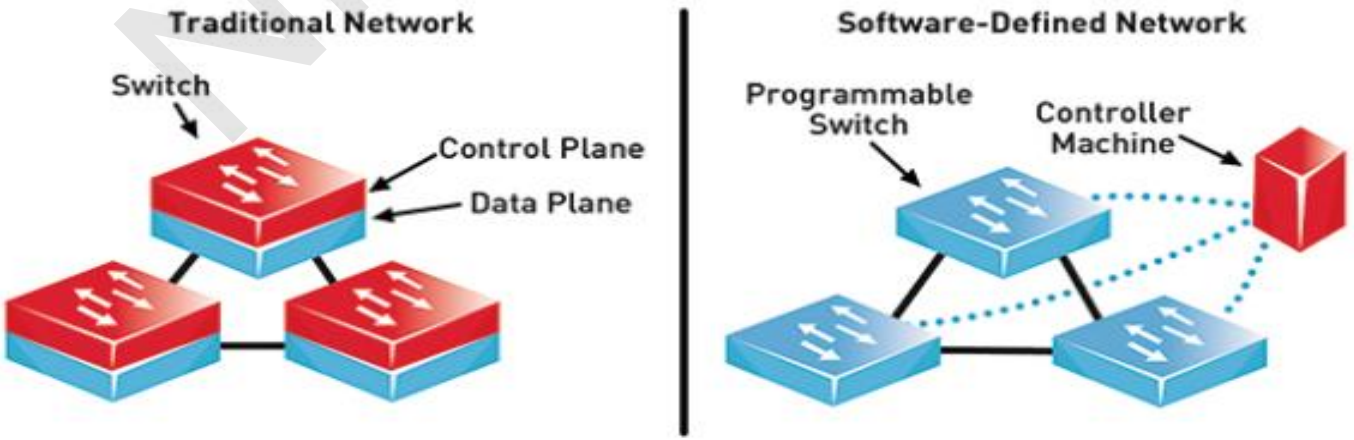
Software Defined Network
Application to Networking in the Cloud

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.

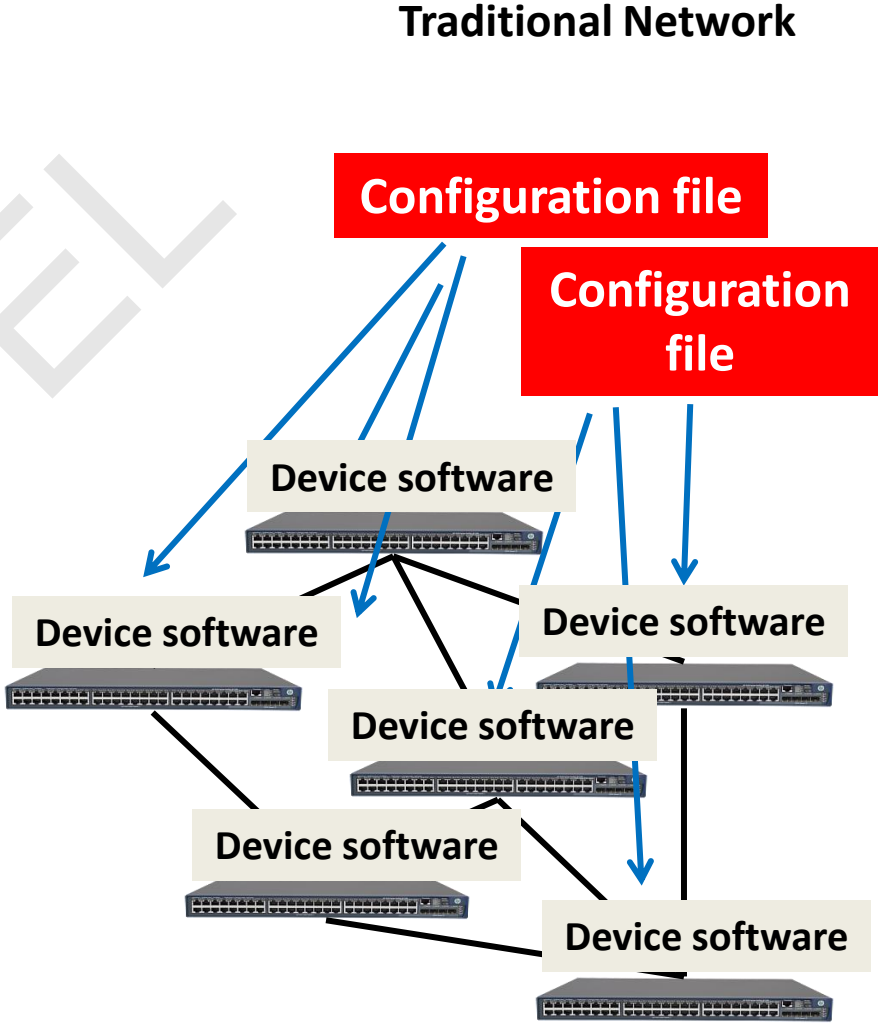
Traditional Network

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.**

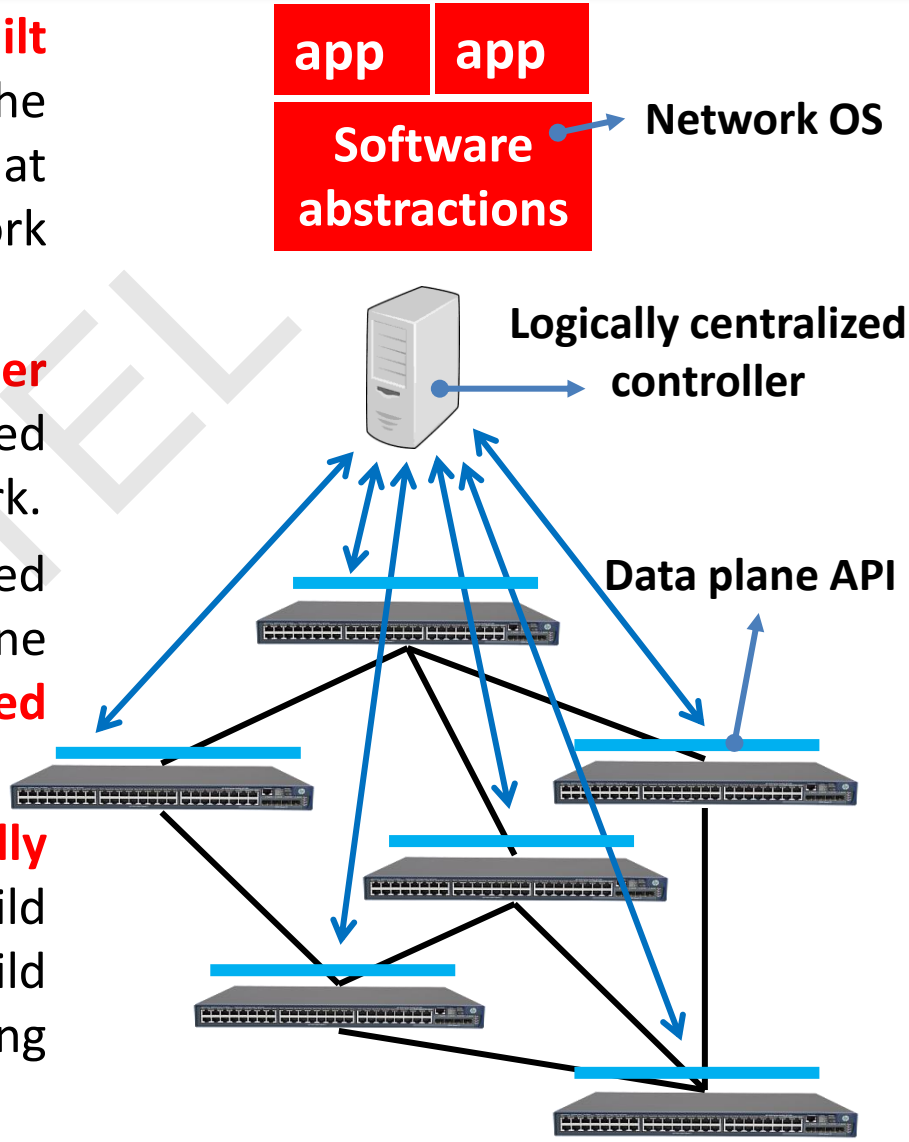
Once it reaches a device then these **configurations have to be interpreted by router software and implementation of protocols** that run distributed algorithms and arrive at routing solutions that produce the ultimate behavior that's installed in the network hardware.

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
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
        (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # 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)
```

Match specific set of packets

Construct action

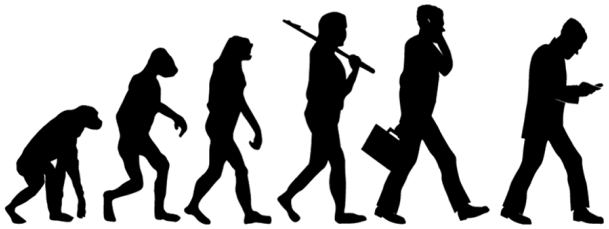
Install (match, action) in a specific switch

Key Ideas of SDN

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

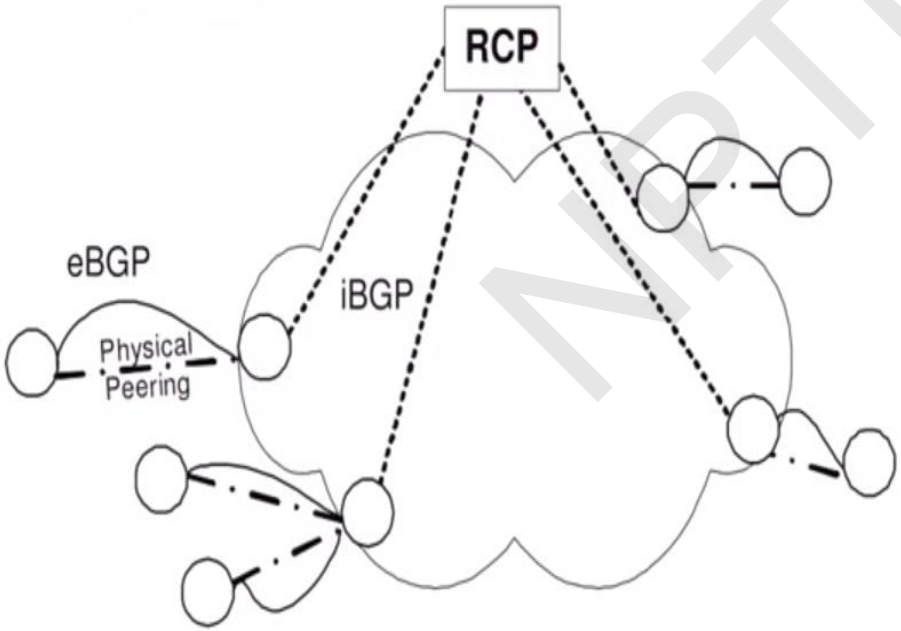
Evolution of SDN: Flexible Data Planes

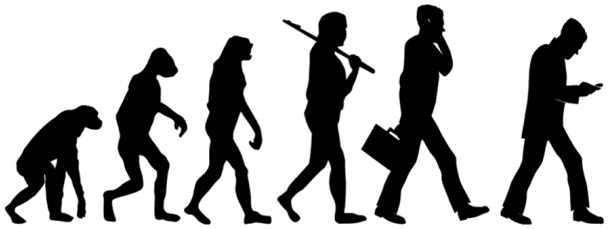
- Evolution of SDN is driving towards making the network flexible.
- **Label switching or MPLS (1997)** i.e. matching labels, executing actions based on those labels adding flexibility:-
- Lay down any path that we want in the network for certain classes of traffic.
- Go beyond simple shortest path forwarding,
- 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

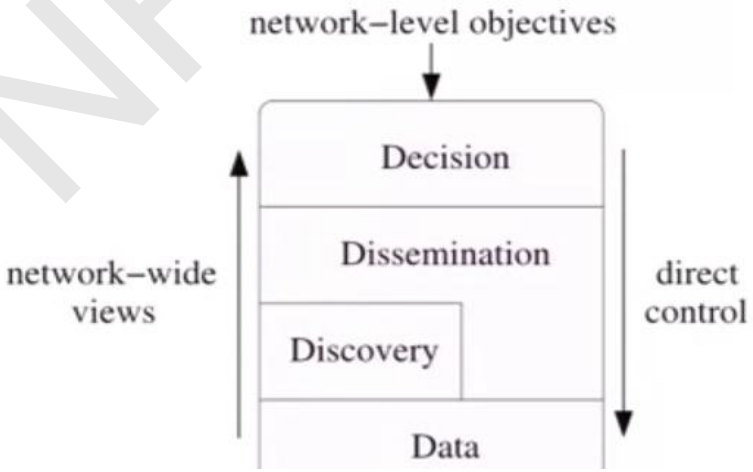
- **Routing Control Platform (2005) [Caesar et al. NSDI 2005]**
- Centralized computing to BGP routes, pushed to border routers via iBGP





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

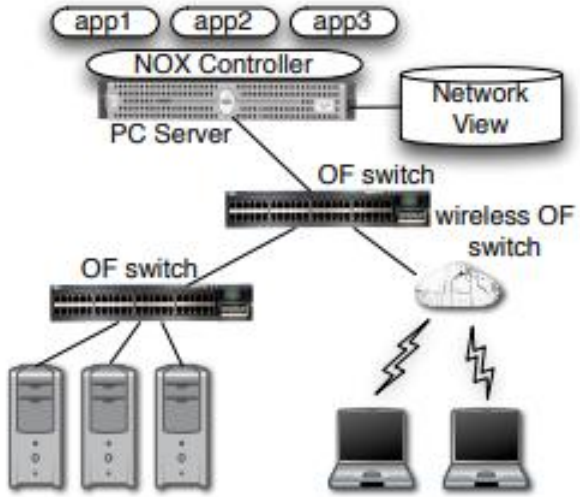
- **Ethane (2007) [Casado et al., SIGCOMM 2007]**
  - Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware.

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

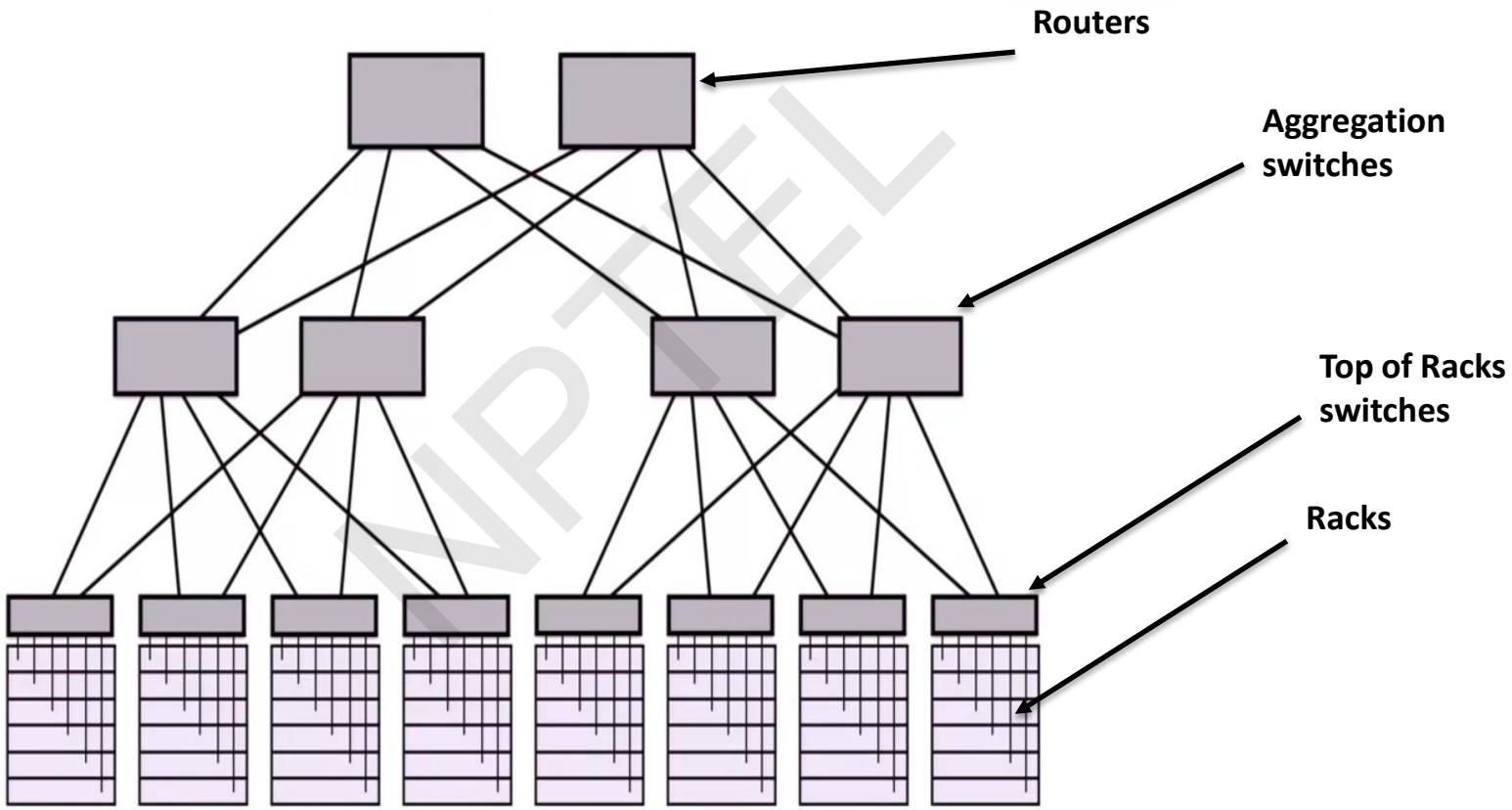
(v) Network semantics

(i) Agility

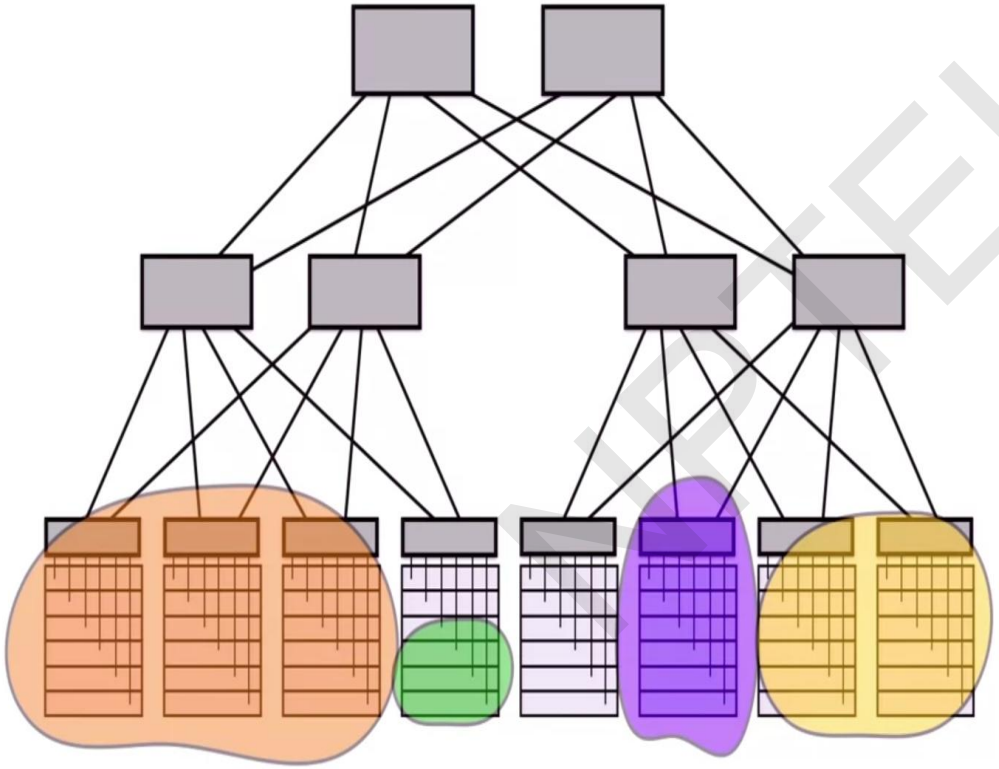
- **Use any server for any service at any time:**
  - **Better economy of scale through increased utilization:** Pack, compute as best we can for high utilization. If we ever have constraints then it's going to be a lot harder to make full use of resources.
  - **Improved reliability:** If there is a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



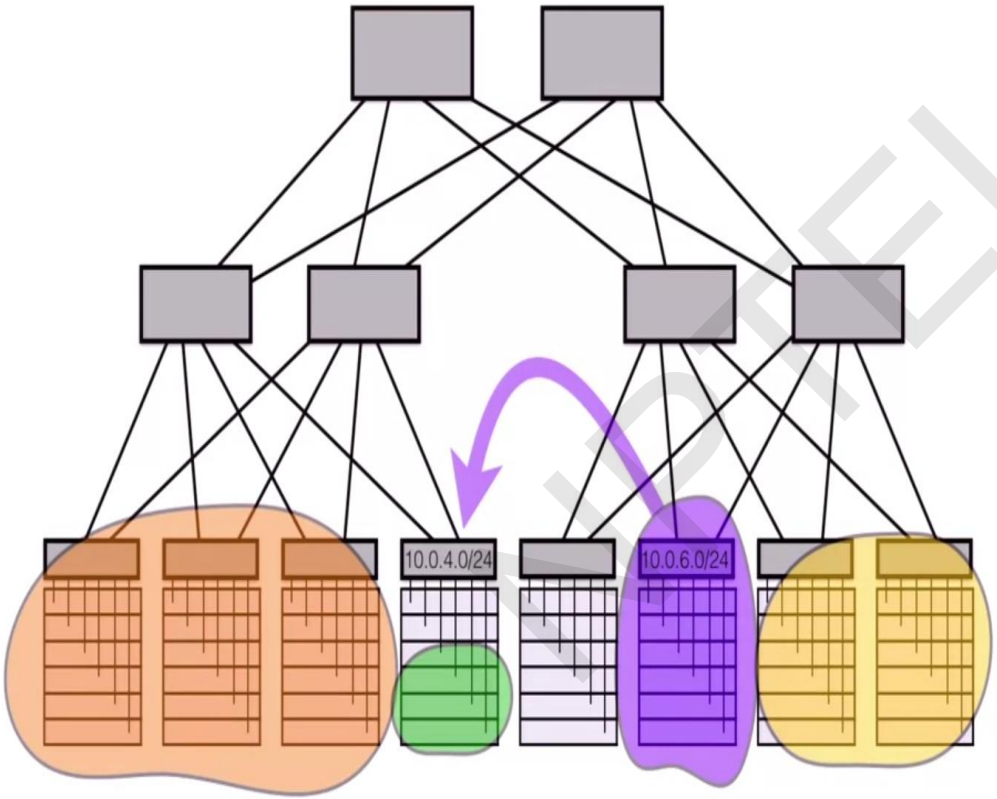
Lack of Agility in Traditional DCs



- **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



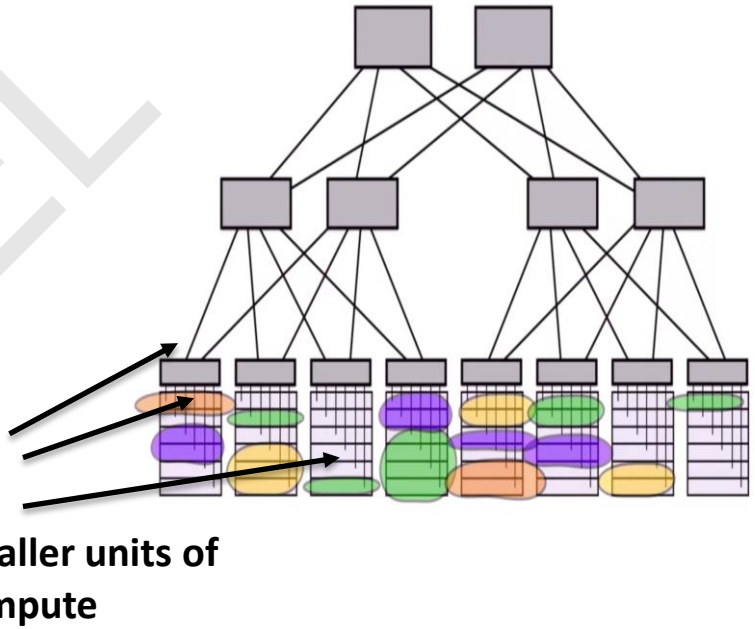
- **IP addresses locked to topological location!**

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.

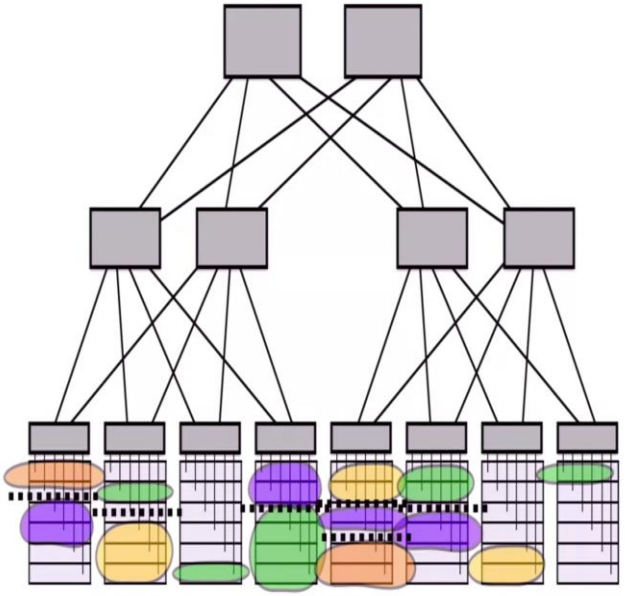
Key needs: Performance Uniformity

- **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

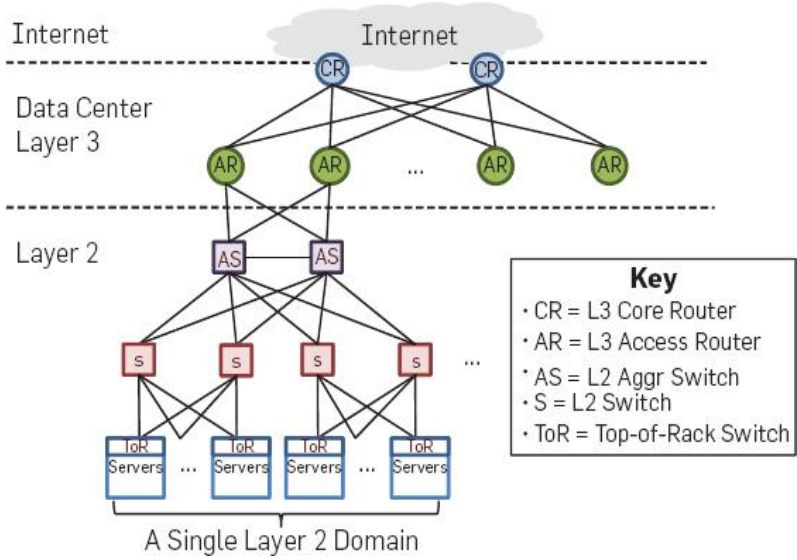
- **Network semantics:**
  - Match the functional service of a traditional data center
  - Not just Layer 3 routing services but also, Layer 2 services i.e. discovery services, multicast, broadcast etc. have to be supported.

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

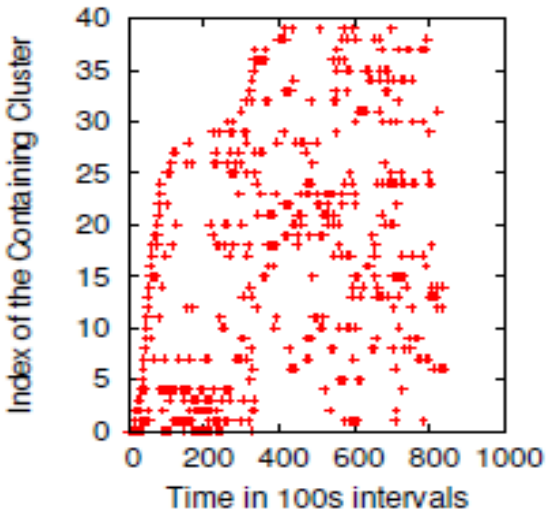
- Traffic volume between servers is 4 times larger than the external traffic

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

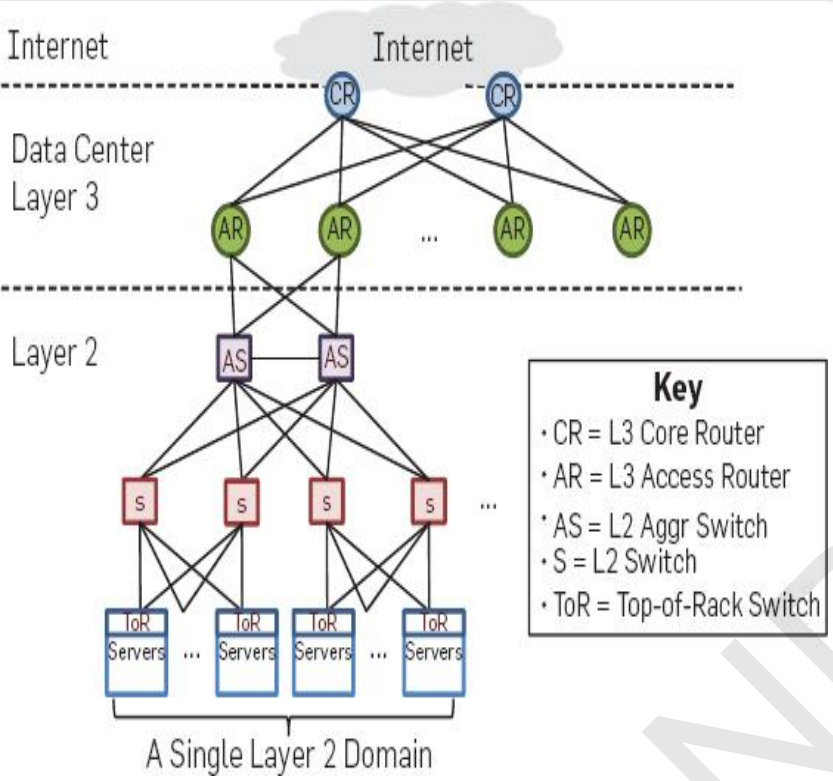
Failure characteristics:

- 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)

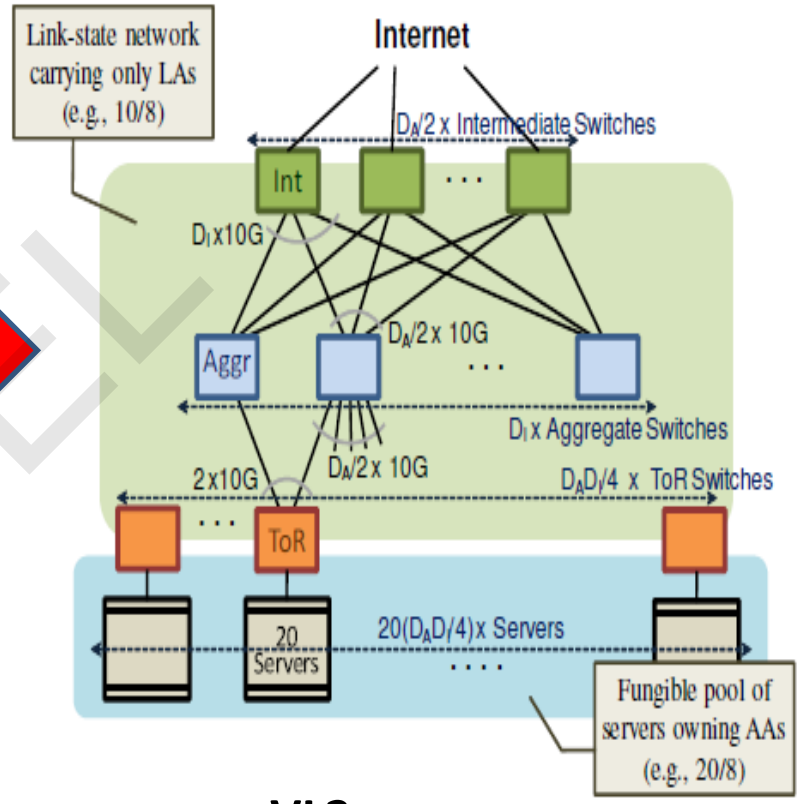
Design result: Clos topology:

- 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"

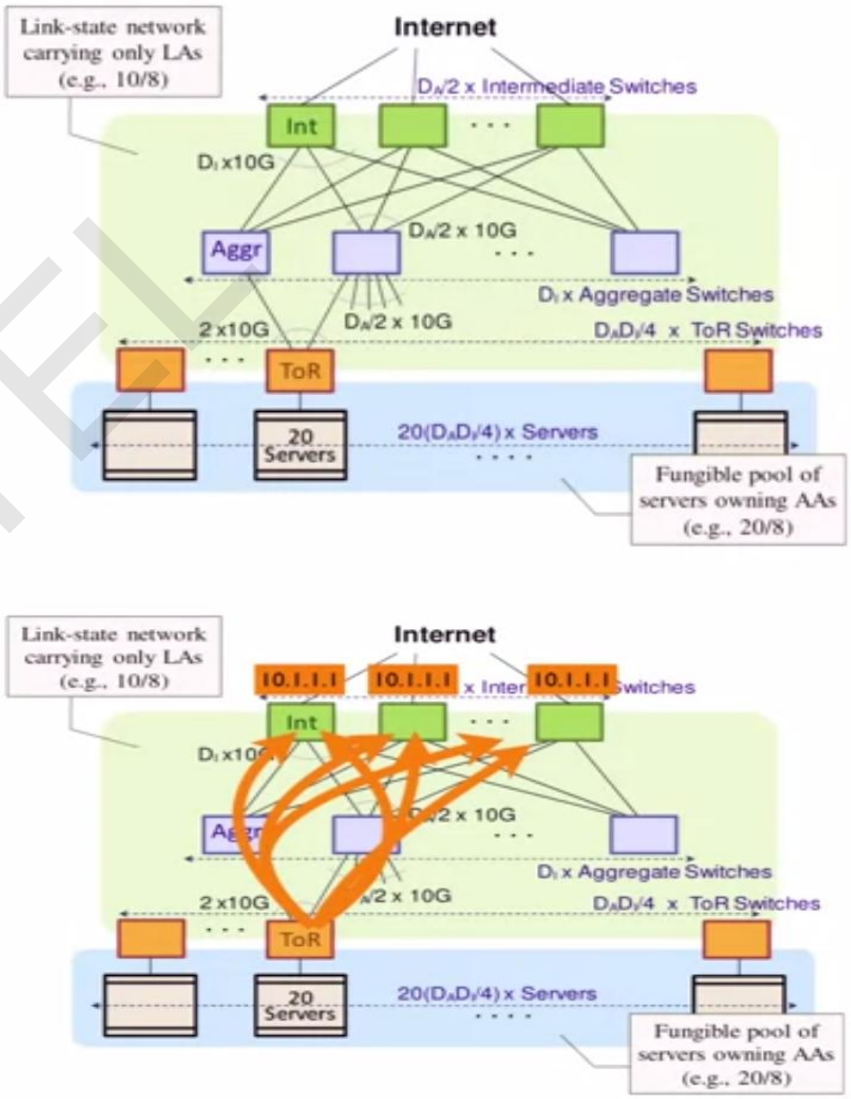
- For routing on hyper cubes take an arbitrary traffic matrix and make it look like a completely uniform even traffic matrix.
- Take flows and spreading evenly over all the available paths. Spread traffic as much as possible.
- Route traffic independent of current traffic matrix

Routing Implementation

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

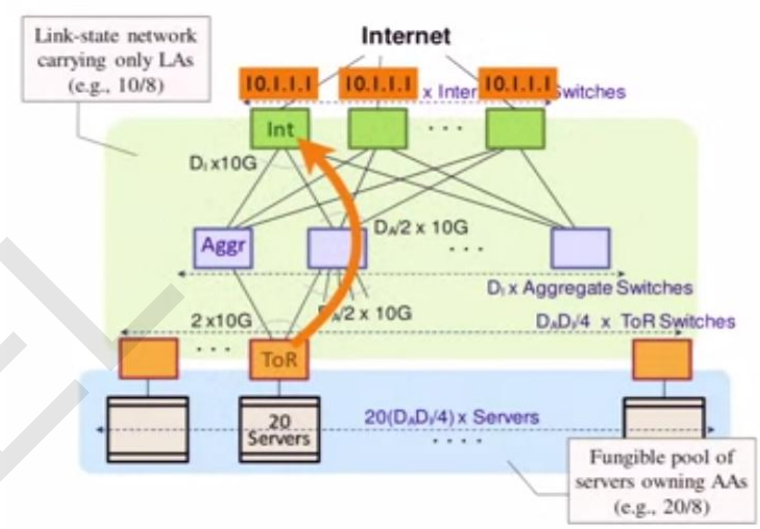
Now to do that what VL2 does is **it assigns those intermediate switches and any cast address.** The same any cast address for all of the switches.

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.**

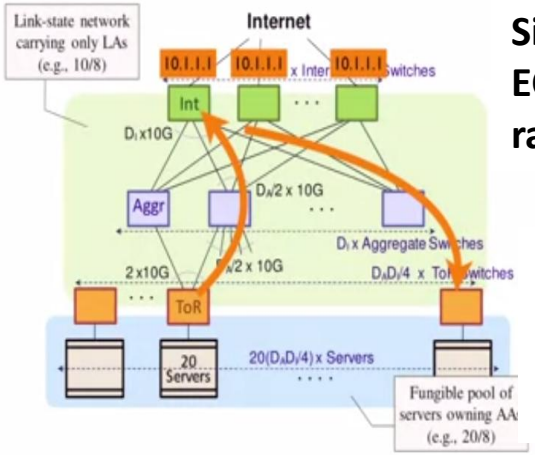


Routing Implementation

- 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.



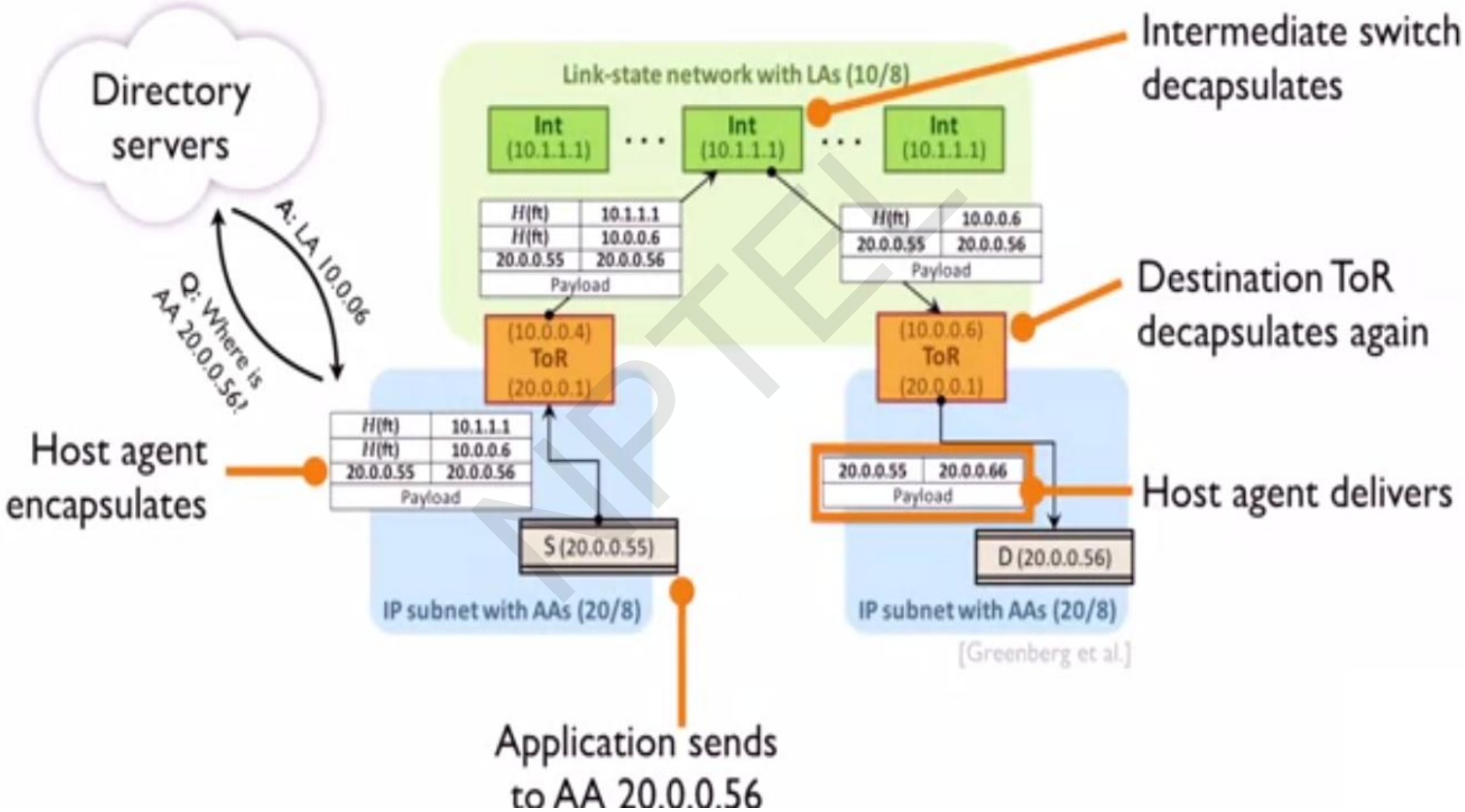
Similar effect to ECMP to each rack

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

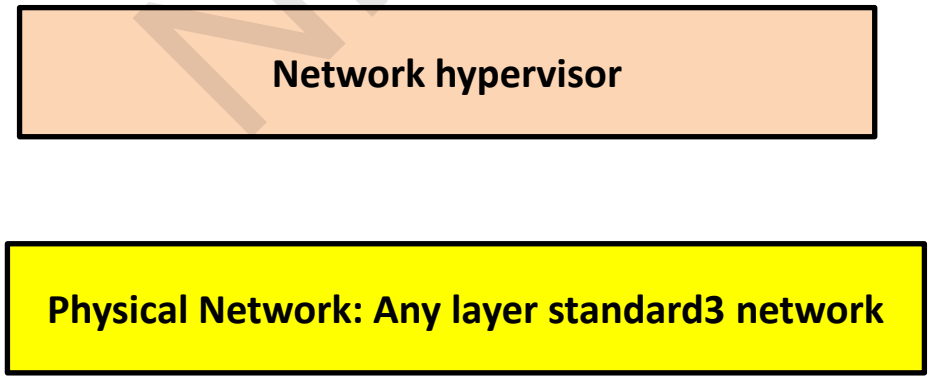
NVP Approach to Virtualization

- 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.

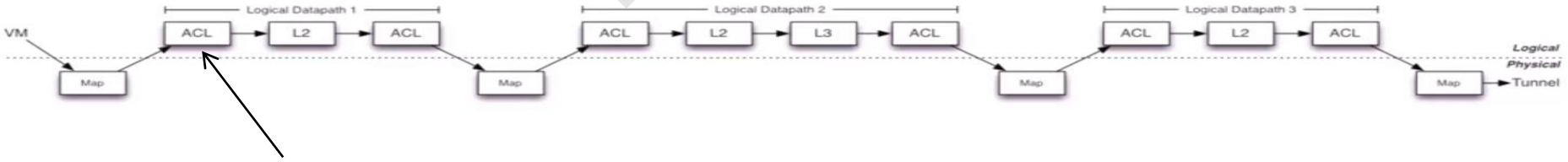
Service: Arbitrary network topology

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



Virtual Network Service

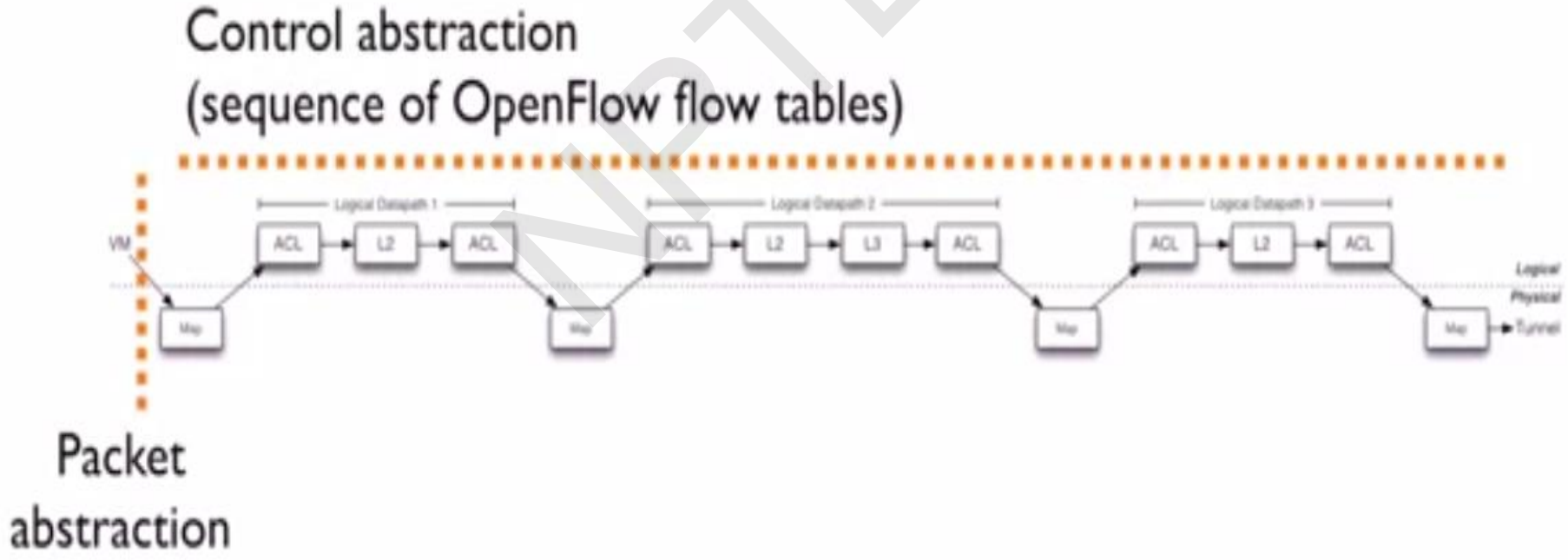
- Modeled as a sequence of data path elements that represent those switches.
- And these data path elements, each one of them is a OpenFlow forwarding table.
- It means the table will match on certain signature of packet headers and take certain resulting actions like dropping the packet, modifying certain 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.

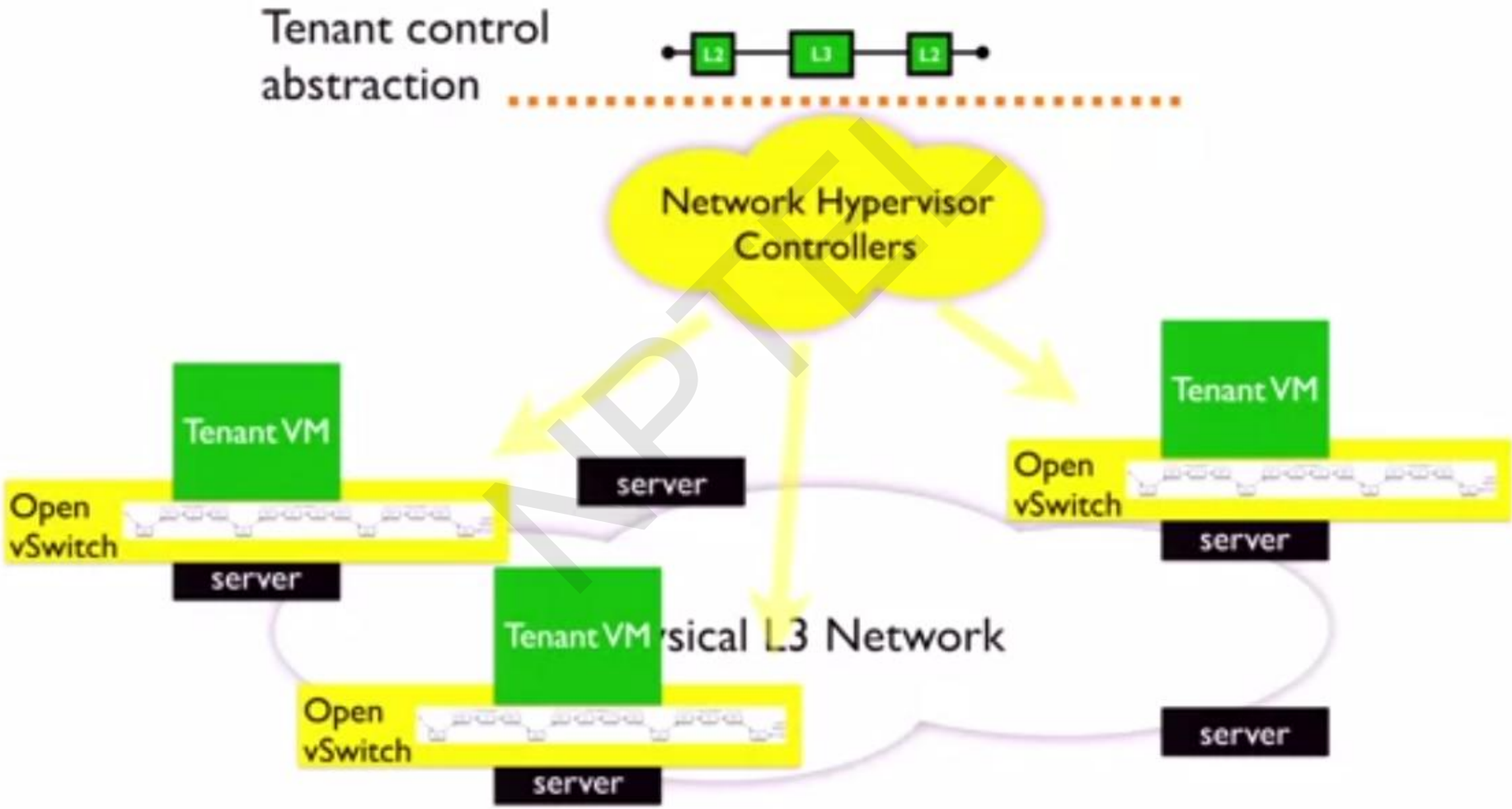


Access control Link Table

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**

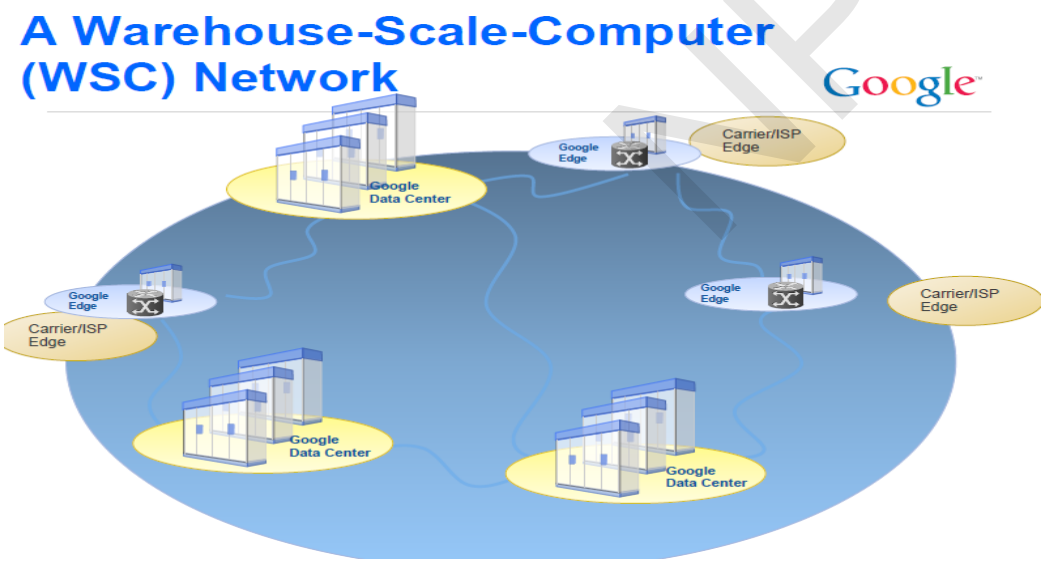
Geo-distributed Cloud Data Centers

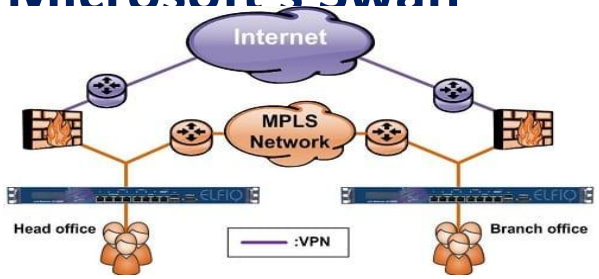
Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

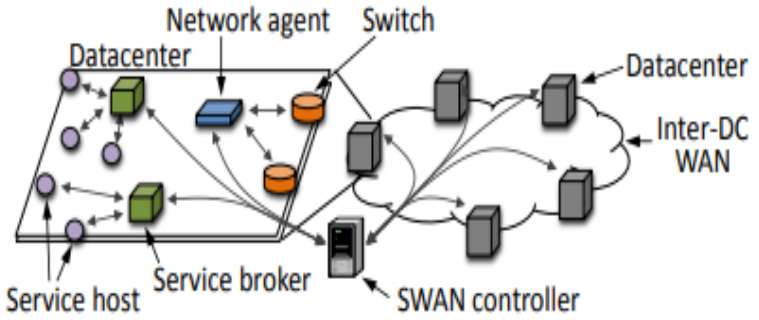
Preface

Content of this Lecture:

- In this lecture, we will study the **Geo-distributed cloud data centers**, interaction of data centers with users and with other data centers.
- We will also describe the data center interconnection techniques such as **(i) Traditional schemes such as MPLS, (ii) Cutting edge such as Google's B4 and (iii) Microsoft's Swan**

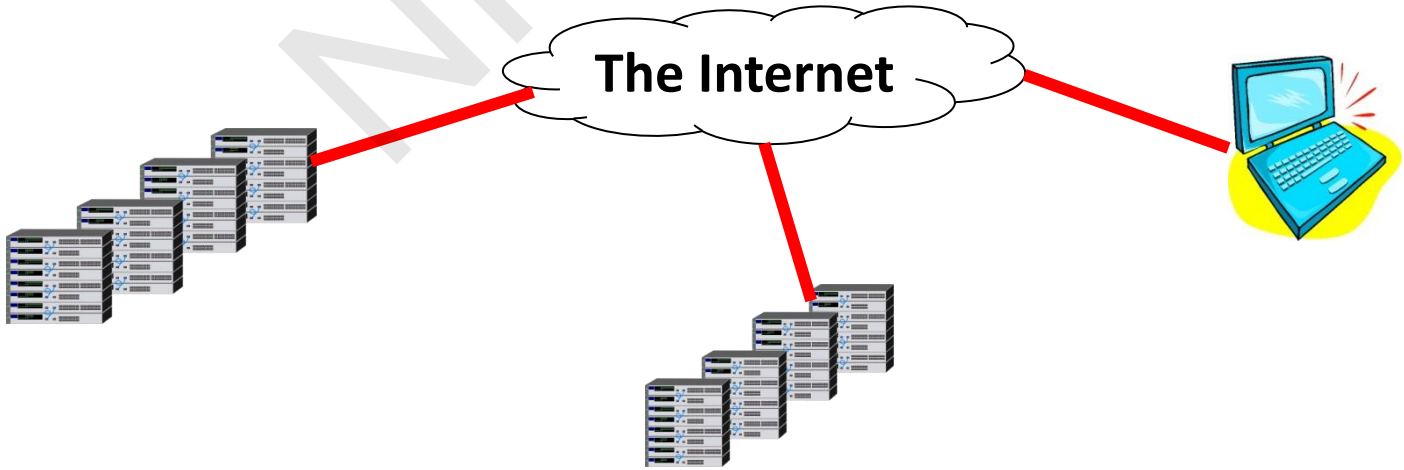






Inter-Data Center Networking: The Problem

- Today, the virtual use of any popular web application means to communicate with a server in a data center.
  - However the connectivity for this service depends on the internet
  - Internet becomes crucial in application service's performance
- Data centers also communicate with each other over the internet. **Example: replicating client data across multiple data centers.**
- In cloud scenario, the wide area connectivity or the internet is as crucial as the data center infrastructure.



Why Multiple Data centers ?

- Why does a provider like Google need such an extensive infrastructure with so many locations across a wide expanse of the globe?
- **Better data availability:** If one of the facilities goes down, due to a natural disaster, you could still have the data be available at some other location if it is replicated.
- **Load balancing:** Multiple facilities can spread incoming and outgoing traffic over the internet across a wider set of providers, over a wider geographic regions.
- **Latency:** If present in multiple paths of the globe then can reach clients in different locations at smaller distances, thus reduces latency.
- **Local data laws:** Several authority might require that companies store data from that country in that jurisdiction itself.
- **Hybrid public-private operation:** Can handle the average demand for service from the private infrastructure, and then offload peak demand to the public cloud.

Significant Inter-data center traffic

This paper was presented as part of the main technical program at IEEE INFOCOM 2011

A First Look at Inter-Data Center Traffic Characteristics via Yahoo! Datasets

Yingying Chen[1], Sourabh Jain[1], Vijay Kumar Adhikari[1], Zhi-Li Zhang[1], and Kuai Xu[2]

[1]University of Minnesota-Twin Cities
[2]Arizona State University

- **Study from five Yahoo data centers from 2011.**
- The study is based on **anonymized traces from border routers** that connect to these datacenters.

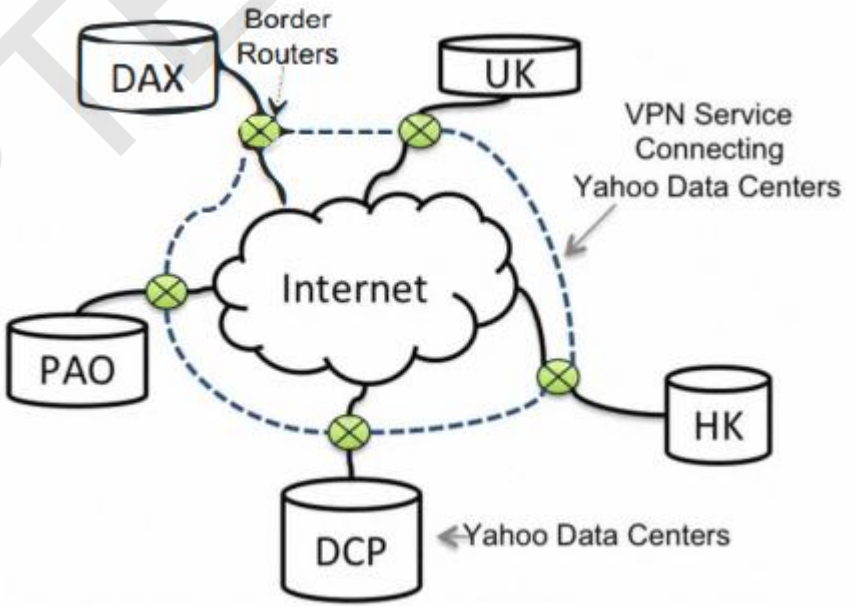
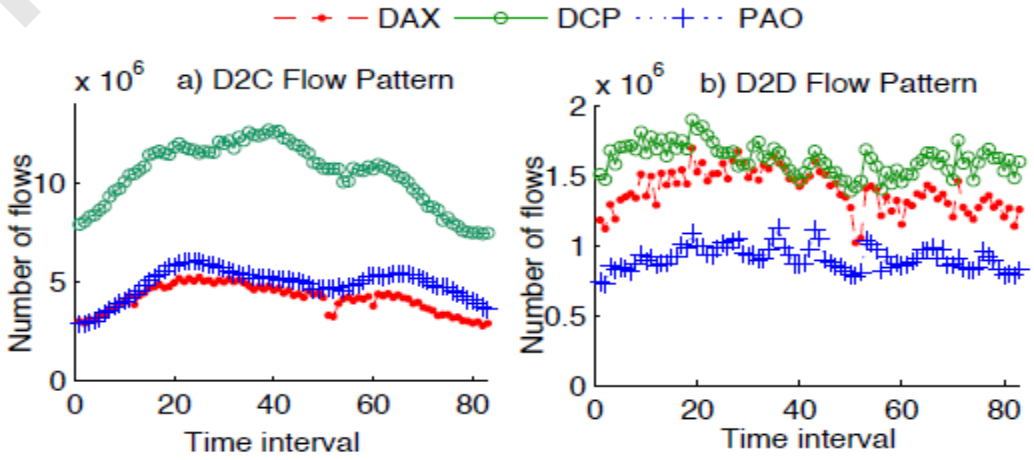


Fig.: Overview of five major Yahoo! data centers and their network connectivity.

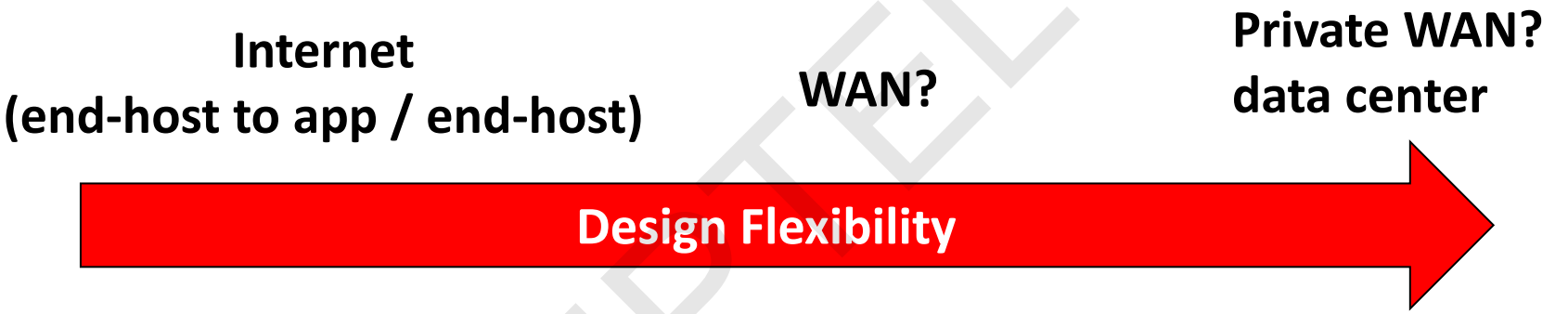
Significant Inter-data center traffic

- Here two plots showing the **number of flows between clients and the data centers.**
- On the right, between the data centers themselves. The three lines on each plot are for three different data center locations. Notice that the y-axis on these two plots are different. In terms of number of flows, the traffic between data centers is 10% to 20% of the traffic from data centers to clients.
- Flows between data centers could be very long lived and carry more bytes than those between clients and data centers.



Why are these networks different ?

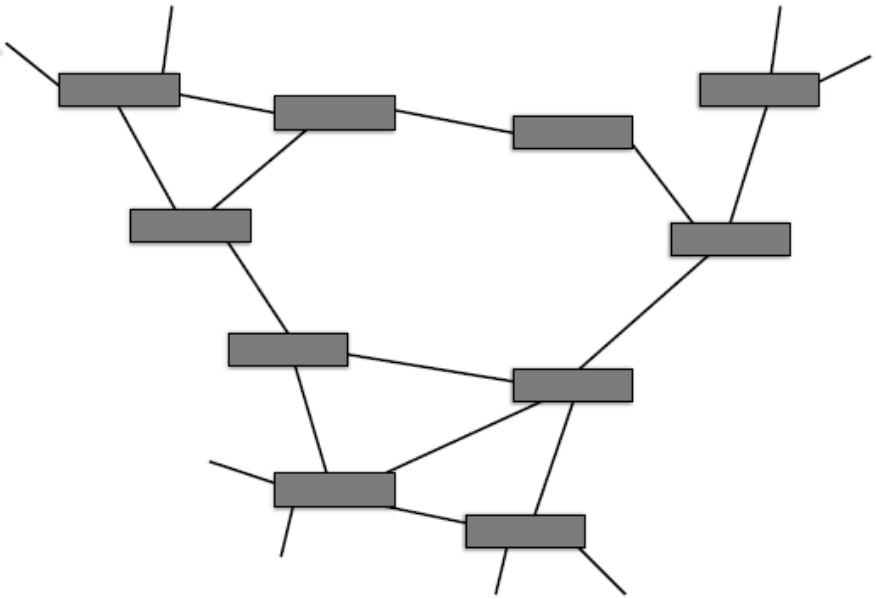
- Persistent **dedicated,100s of Gbps** connectivity between a (small) set of end-points



- Microsoft: "expensive resource, with amortized annual cost of 100s of millions of dollars"
- **[Achieving High Utilization with Software-Driven WAN, Hong et al., ACM SIGCOMM'13]**

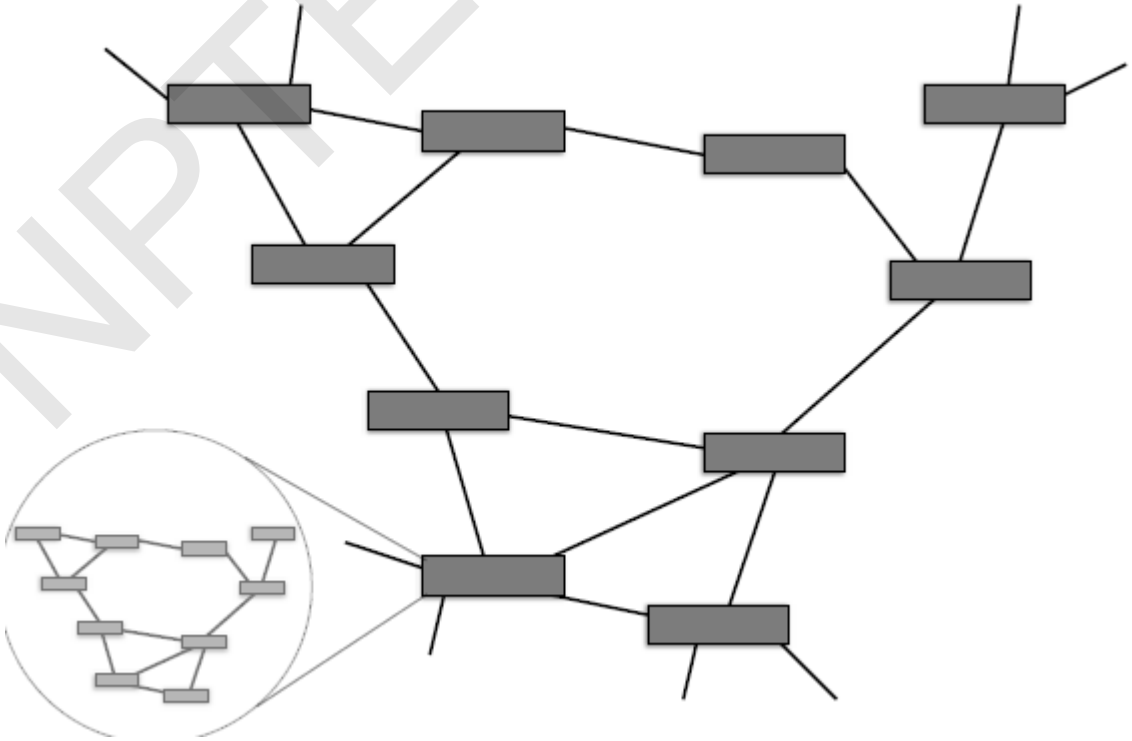
(i) MPLS: Traditional WAN approach

- The traditional approach to traffic engineering in such networks is to use **MPLS (Multiprotocol Label Switching)**
- Network with several different sites spread over defined area, connected to each other perhaps over long distance fiber links.



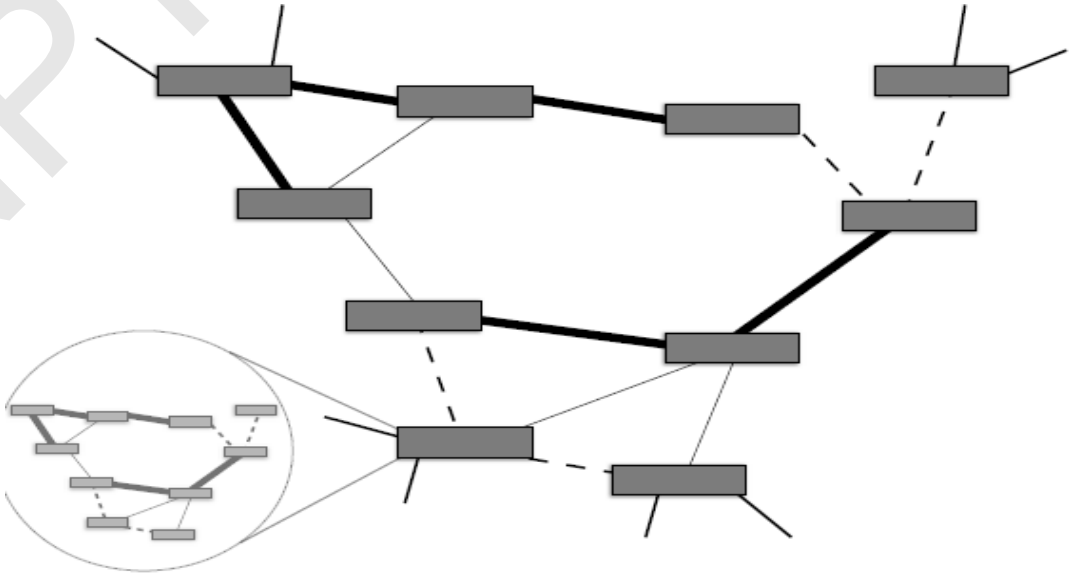
1. Link-state protocol (OSPF / IS-IS)

- Use **link-state protocol (OSPF or IS-IS)** to flood information about the network's topology to all nodes.
- So at the end of such a protocol, every node has a map of the network.



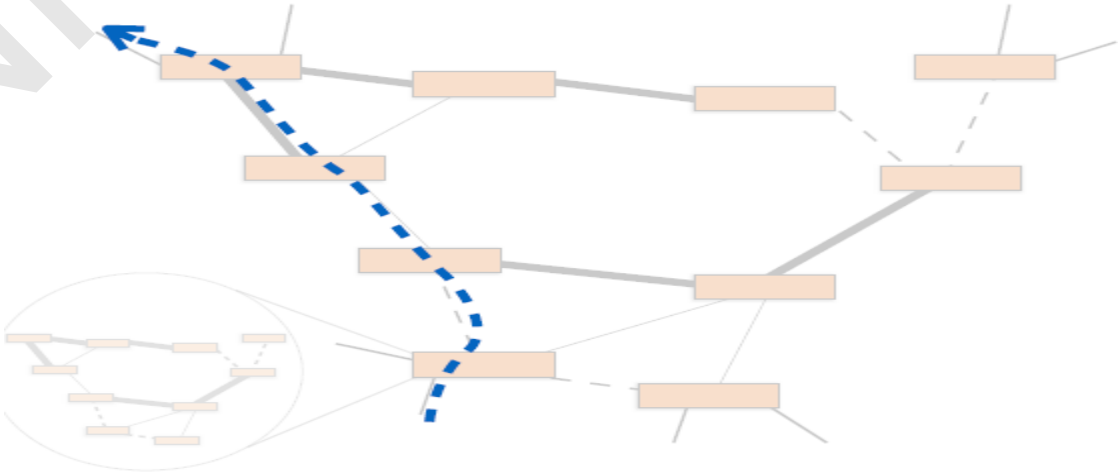
2. Flood available bandwidth information

- For traffic engineering, also **spread, information about the bandwidth usage** on these different links in the network.
- Given that there's already traffic flowing in this network, some links will have spare capacity and some won't.
- Both IS-IS and OSPF have extensions that allow the flooding of available bandwidth information together with their protocol messages.



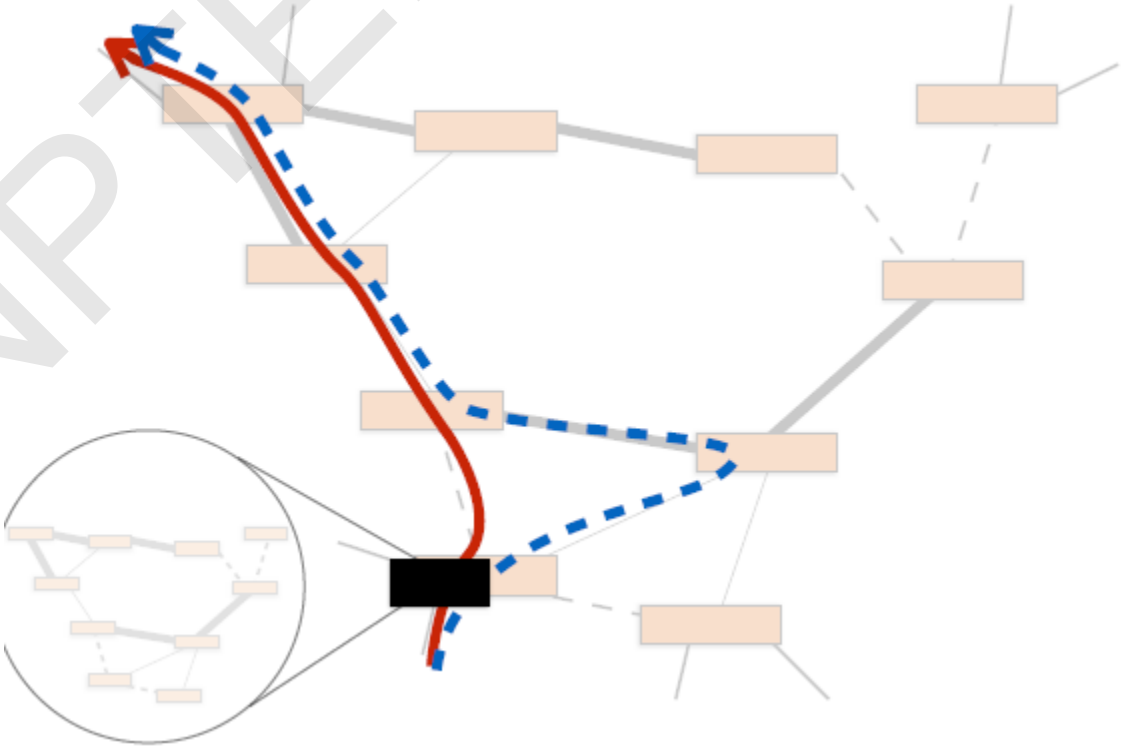
3. Fulfill tunnel provisioning requests

- Knowing the set of the network, when the router receives a new flow set of requests, it'll set up a tunnel along the shortest path on which enough capacity's available. It sends protocol messages to routers on the path setting up this tunnel.
- Further, **MPLS also supports the notion of priorities.** Thereby if a higher priority flow comes in with the request for a path, lower priority flows might be displaced. These flows might then use higher latency or higher cross paths through the network.



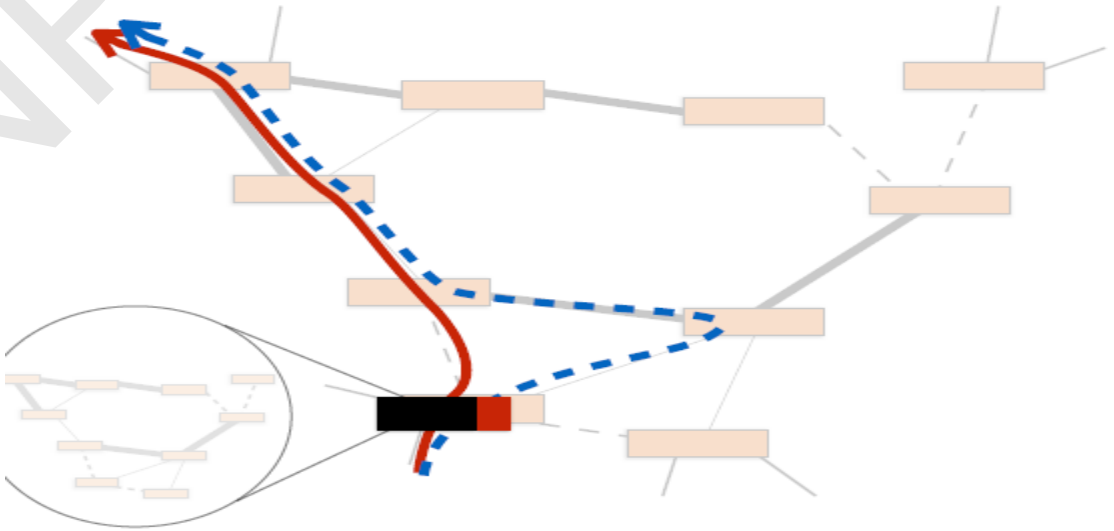
4. Update network state, flood information

- After a flow is assigned a terminal, the routers also update the network state.



4. Update network state, flood information

- When a data packet comes into the ingress router, the router looks at the packet's header and decides what label, that is what tunnel this packet belongs to. Then it encapsulates this packet with that tunnel's label and sends it along the tunnel.
- The egress router then decapsulates the packet, looks at the packet header again and sends it to the destination. **In this scheme, only the ingress and egress routers read the packet header.** Every other router on the path just looks at the assigned label.



Simple forwarding along the path

- Making forwarding along the path very simple. This is the reason the protocol is called **Multi-Protocol Label Switching (MPLS).**

- Also, MPLS can run over several different protocols, as long as the ingress and egress routers understand that protocol, and can map onto labels, that's why the name is: **multi-protocol label switching.**

Problem 1: Inefficiency

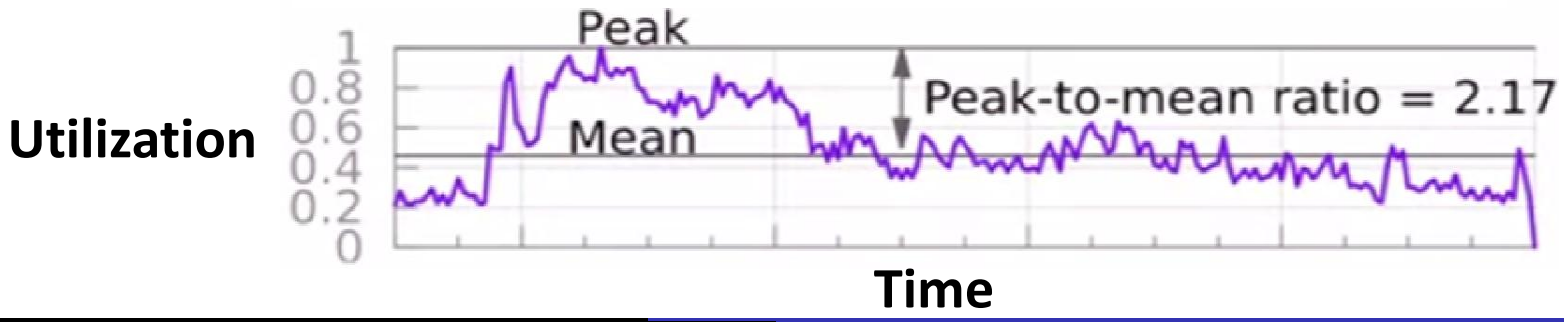
- First problem is **inefficiency in terms of usage of the expensive bandwidth.** Typically, these networks would be provisioned for the peak traffic.
- As this image shows here, if you have the traffic over time, the y-axis utilization, provision the network for peak traffic. Now, the mean usage of the network might be very small. In this example, it's **2.17 times smaller than the peak.**

ACM SIGCOMM, 2013

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

Microsoft



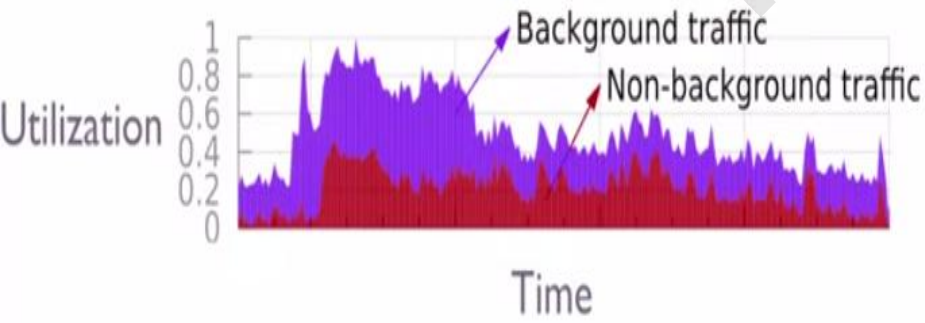
Problem 1: Inefficiency

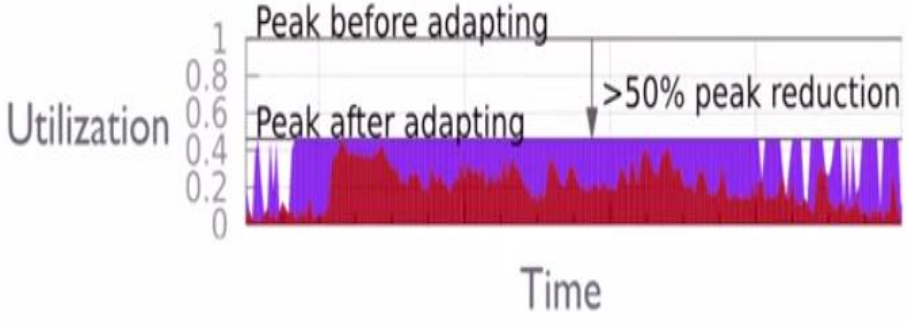
- Most of this traffic is actually **background traffic, with some latency sensitive traffic** as well.
- So you can provision for the peak of the latency sensitive traffic, and then fill the gaps with the background which is not latency sensitive.

ACM SIGCOMM, 2013

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft





Problem 1: Inefficiency

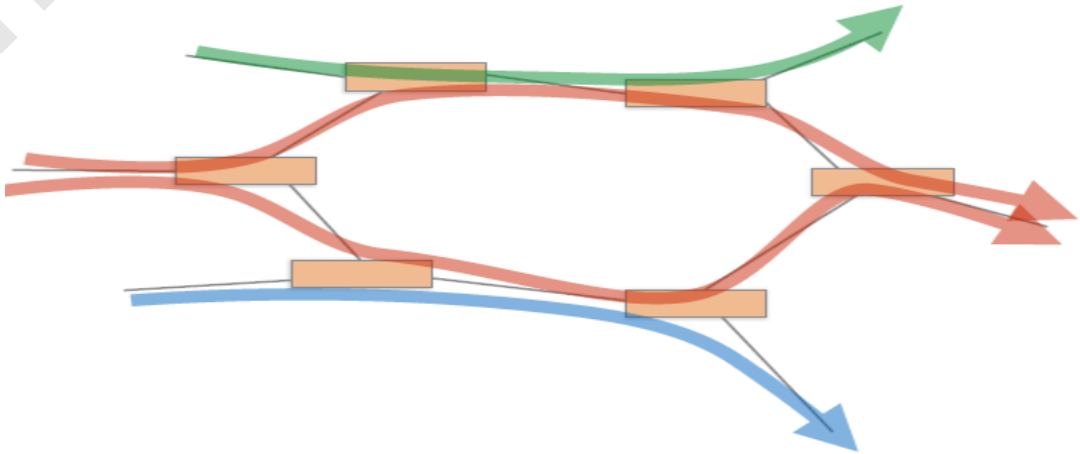
- So unless you differentiate traffic by service, you cannot do such an optimization. This is not easy to do with the MPLS approach because **it does not have a global view of what services are running in the network,** what parts of the network they are using and such.

- Also, a related point is that regardless of whether they are multiple services or not, MPLS, the routers make local greedy choices about scheduling flows. So **traffic engineering is sub optimal.**

- For these reasons, such networks typically run around 30% utilization to have enough headroom for these inefficiencies, and this is expensive.

Problem 2: Inflexible sharing

- Another big problem with the MPLS approach, is that it only provides link level fairness. So at any link, the flows can share capacity fairly. **But, this does not mean network wide fairness.**
- For example, we have the green flow sharing capacity across that length with a red flow. The blue flow also shares capacity with the red flow. But, the blue and green flows both get capacity half the red flow, because the red flow uses multiple paths. So we have link level fairness, but we do not have the network wide fairness.
- The network wide fairness is hard to achieve, **unless you have a global view of the network.**



Cutting-edge WAN Traffic Engineering (TE)

Google's B4

ACM SIGCOMM, 2013

B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh,
Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla,
Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

Microsoft's Swan

ACM SIGCOMM, 2013

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

Microsoft

1. Leverage service diversity: some tolerate delay

- **To get very high bandwidth utilization in the WAN** because of natural fluctuations over time. So, to leverage diversity in the services some services need a certain amount of bandwidth at a certain moment of time and they're inflexible and some other services can use to kind of fill in whatever room is left over.

- **For example, latencies instead of queries.**

2. Centralized TE using SDN, OpenFlow

- Software define networking approach gather information about the state of the network.
- Make a centralize decision about the flow of traffic and then push those decisions down to lower levels to actually implement them.
- But bringing all that information together in one place is a relatively complex decision.

3. Exact linear programming is too slow

- Traditionally, with a optimization technique like linear programming, which is a way to take a set of constraints on required amounts of data flow over parts of a network and come up with an optimal solution. But to apply it to the situation where we need to **make relatively quick decisions.**

- Part of the complexity comes from the multitude of services, the different priorities. If we have just one service, we could run it in flow algorithm, and that would be much faster.

- So it require something faster if it's not guaranteed to be exactly optimal.

4. Dynamic reallocation of bandwidth

- **The demands on the network change over time.** So to make continual decisions about what traffic is highest priority to move across which links at a given moment is a challenge with linear programming to make quick decisions.

- So these are online algorithms. But they're not online in the same way as things inside the data center might be.

- For example, Google runs its traffic engineering 500 or so times a day. So, it's not as fine grained as things we might need inside a data center. Traffic between these facilities is relatively stable it seems.

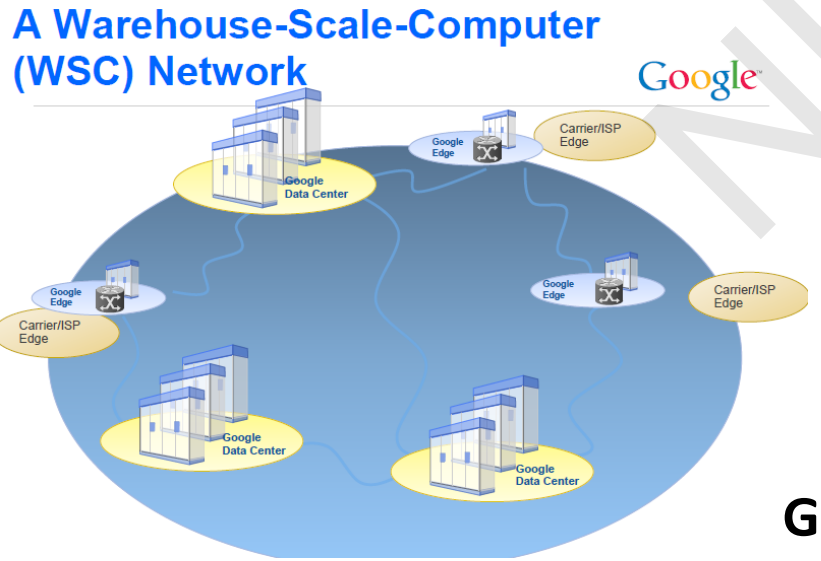
5. Edge rate limiting

- The commonality in the architecture is to implement an **enforcement of the flow rates.**

- So when the traffic enters the network and will do that at the edge, rather than at every hop along the path.

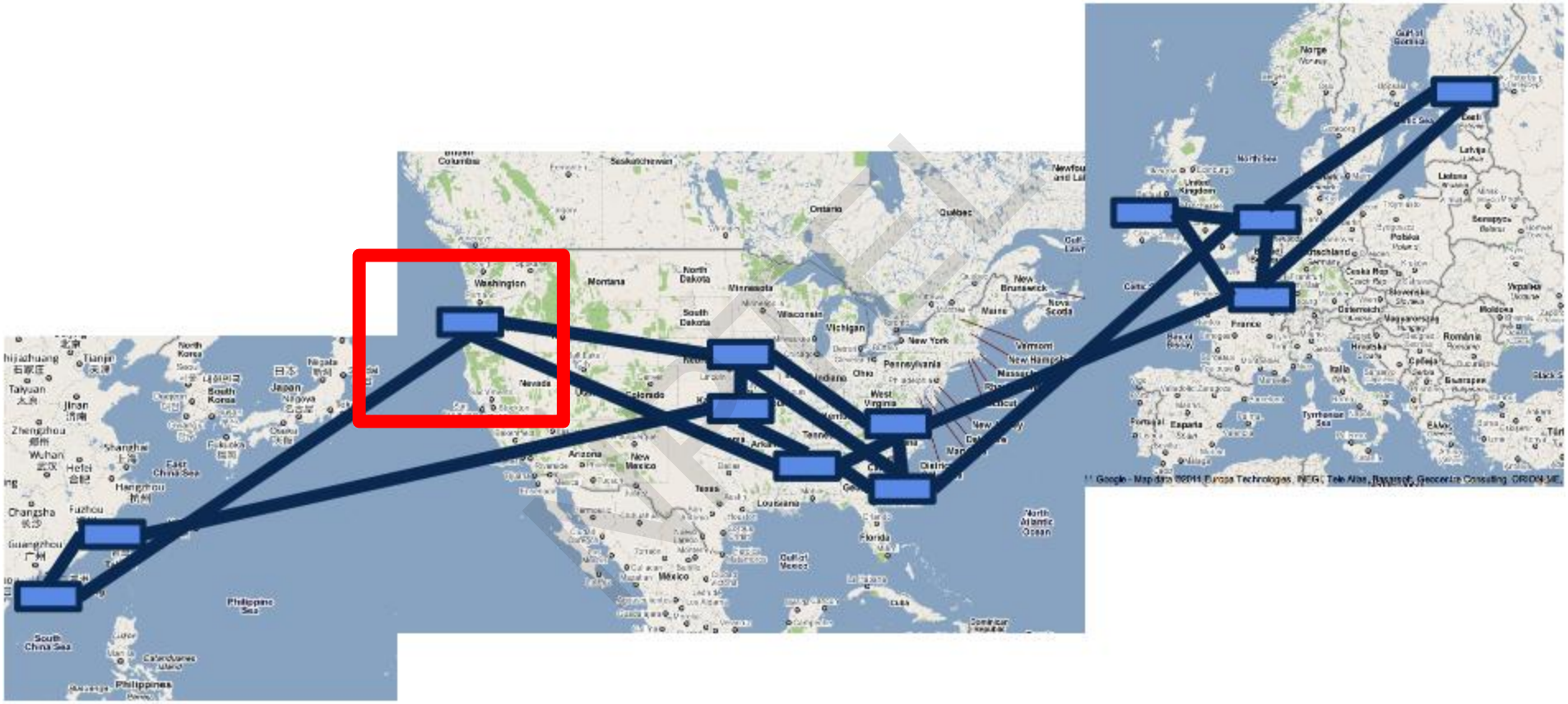
(ii) B4: Google's WAN (2011)

- **Google's B4 was the first highly visible SDN success.** It is a private WAN connecting Google's data centers across the planet. It has 12 locations spread across three different continents.
- It has a number of unique characteristics: **i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network,** which enables rate limiting and demand measurement at the edge



Google's B4 worldwide deployment (2011)

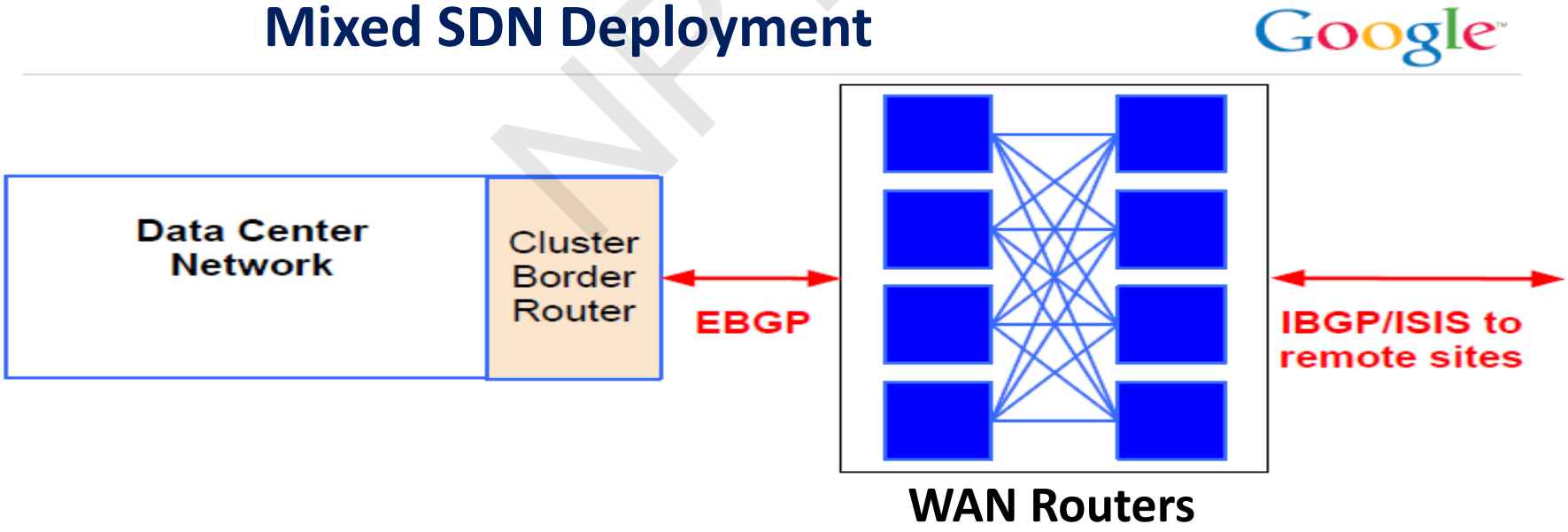
What happens at one site, one data center



Google's B4: view at one site

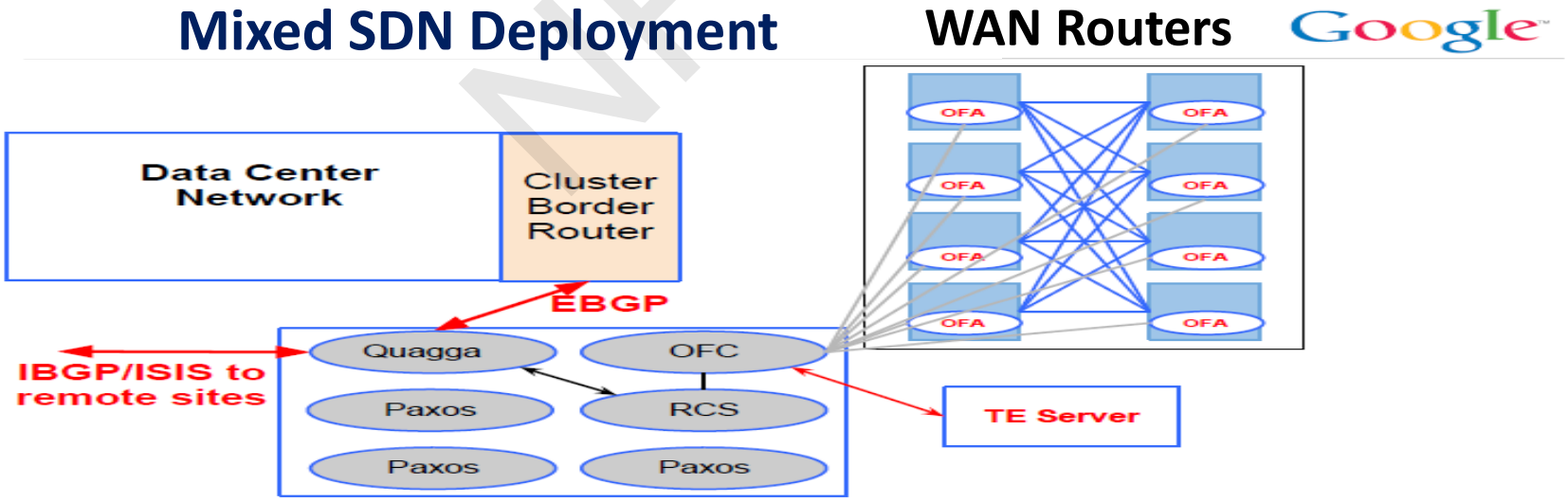
- **Inside one site, it has the data center network, of cluster & border routers**.

- In the traditional setting, these are connected using eBGP to run routers, which would then also interface using iBGP or IS-IS to the other data center sites.



Google's B4: view at one site

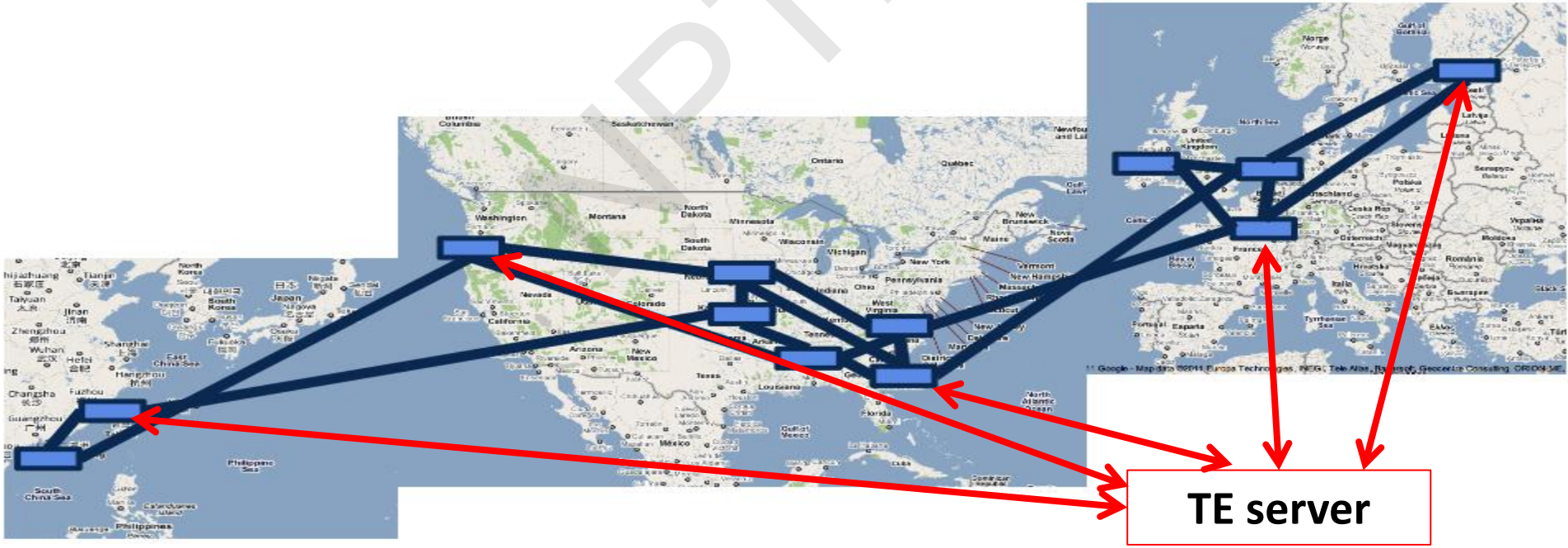
- For finer control over routing, this will be moved to a software router, i.e. **Quagga software switch that run it on a server.**
- The interface with open flow to set up routing rules on those routers. So, Quagga will run the routing protocols between the cluster border routers, and also the other sites and then, OpenFlow uses the routing information from Quagga and sets up forwarding rules in the WAN routers.
- Now we have software control and the traffic engineering (TE) server, which manages what rules exactly are installed.



Google's B4: Traffic Engineering

- The TE server Collects the topology information, the available bandwidth information and has the last information about flow demands between different sites.

- The TE server pushes out the flow allocations to the different data centers. At the data centers the multiple controllers then enforce those flow allocations on the centers.



Google's B4: Design Choices

- **BGP routing as "big red switch":**
- They also keep available the BGP forwarding state because each of their switches allows them to have multiple forwarding tables at the same time they can afford to have BGP forwarding tables.
- Also, in addition to the traffic engineering, if the traffic engineering scheme does not work. They can discard those routing tables and use the BGP forwarding state instead. **So the BGP routing tables serve as a big red switch.**

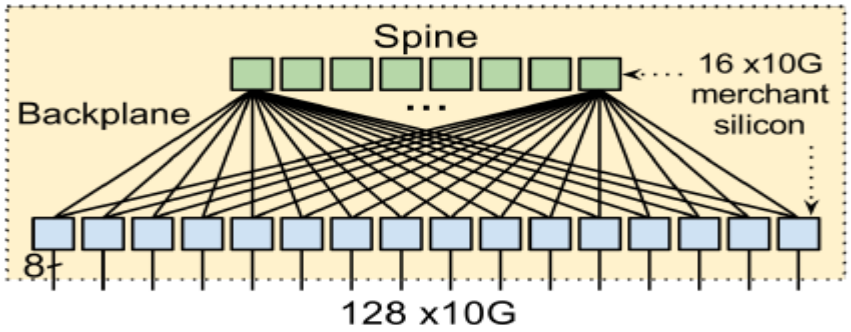
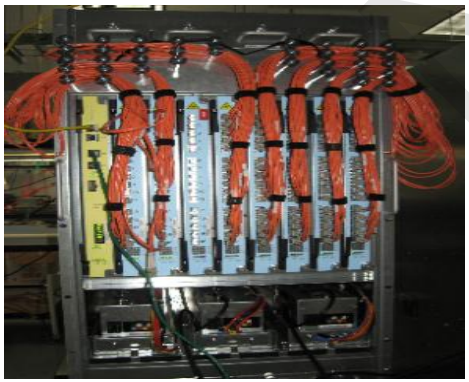


Figure: A custom-built switch and its topology

Google's B4: Design Choices

- **Hardware: custom, cheap, simple:**

- In 2011, there were not many software defined networking switches on the market. **So Google built their own using cheap commodity equipment.**



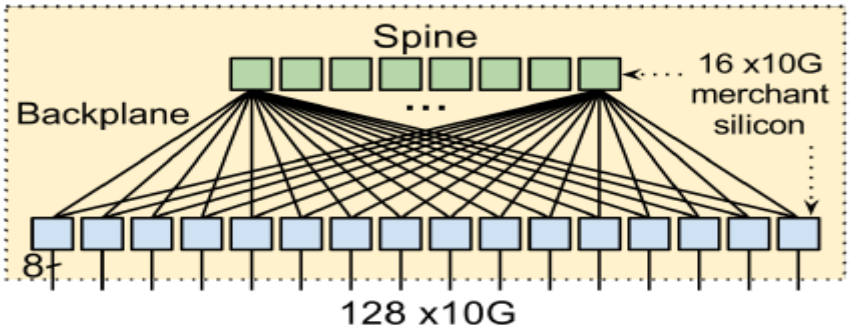


Figure: A custom-built switch and its topology

Google's B4: Design Choices

- **Software smart with safeguard:**

- Most of these smarts then reside in the software.
- Switches are simple in the software, which is the **open flow control logic at each site replicated for fault tolerant using paxos.**
- Further, for scalability of this system they use a hierarchy of controllers. **The software solution achieves nearly 100% utilization and solves the trafficking problem in 0.3 seconds.**

Google's B4: Design Choices

- **Hierarchy of controllers:**
- At the top level we have the **global controller**, which is talking to an **SDN gateway**. The gateway can be thought of as a **super controller** that talks to the controller's at all these different data center sites.
- Each site might itself have **multiple controllers**, because of the scale of these networks. **This hierarchy of controllers simplifies things from the global perspective.**

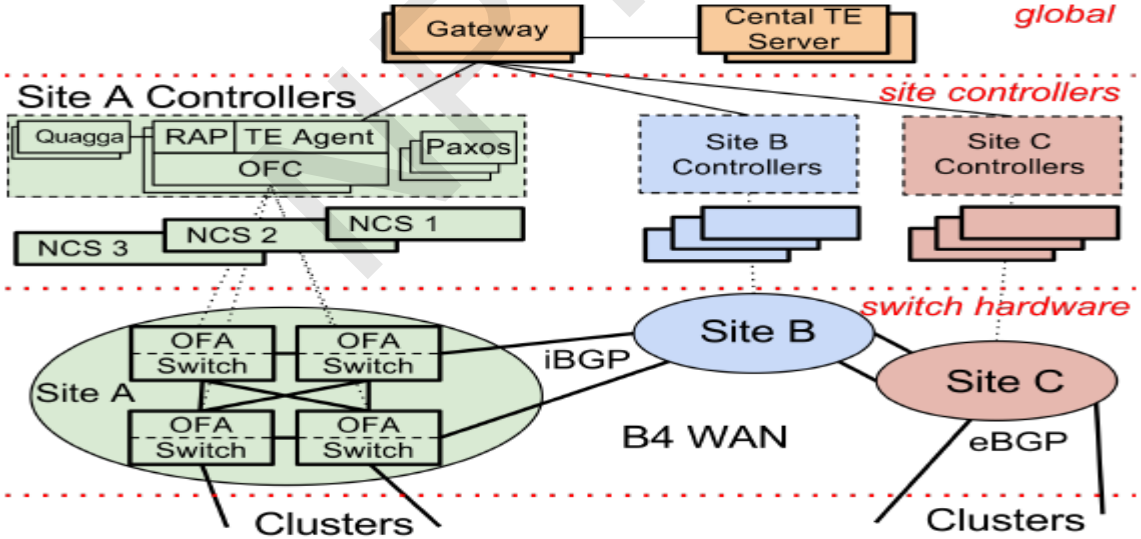


Figure: B4 architecture overview

Google's B4: Design Choices

- **Aggregation: flow groups, link groups:**

- Earlier, traffic engineering at this global scale is not at the level of mutual flows but of flow groups. That also helps scaling. Further, each pair of sites is not connected by just one link.

- These are massive capacity links that are formed from a trunk of several parallel high capacity links.

- All of these are aggregated and exposed to the traffic engineering layer as one logical link**. It is up to the individual site to the partition traffic, multiplex and demultiplex traffic across the set of links.**

(iii) Microsoft's Swan

- Microsoft has publicly disclosed the design for optimizing wide area traffic flow in their WAN.

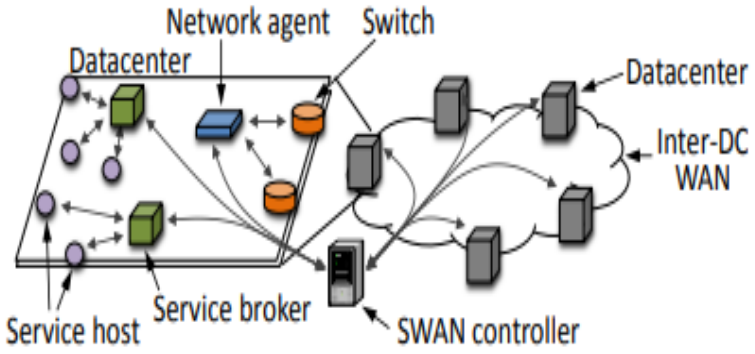
- Interesting feature in this design is the way **to make changes to the traffic flow without causing congestion.**

ACM SIGCOMM, 2013

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

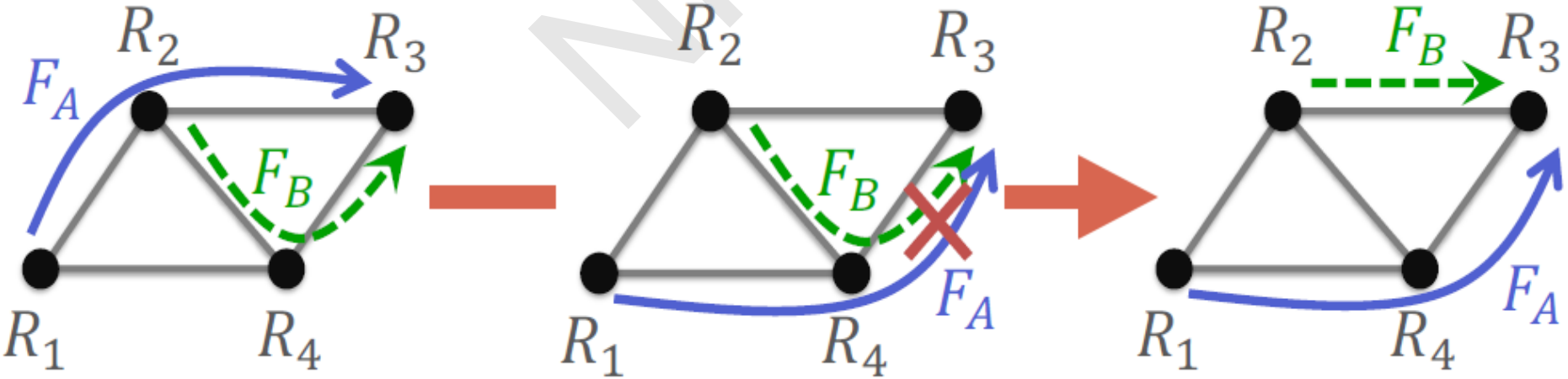
Microsoft



Architecture of Microsoft's Swan

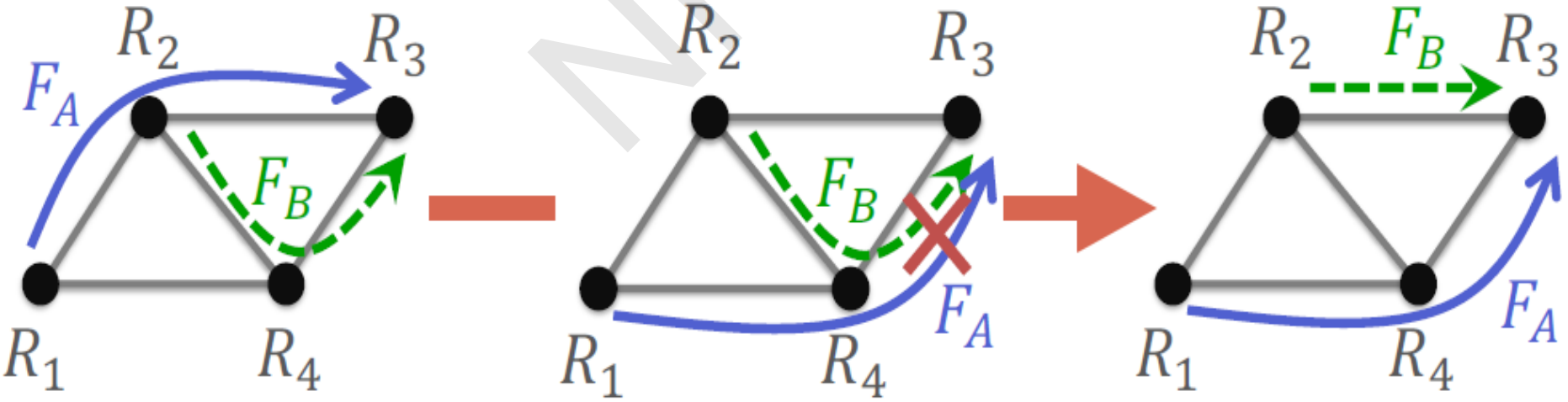
Microsoft's Swan

- The network on the left have two flows, the blue one and the green one that are each using in this simple example 100% of the bandwidth of the links that they flow across.
- To move from the left scenario to the right scenario. If we do using any naive way, because we can't control exactly when every packet will flow across every link. There's going to be some period of time where, due to timing uncertainty, we have both of the flows sharing to some extent one of the links.



Microsoft's Swan

- Can think of different ways to do this but we always going to run into some link that may get used by both lows at the same time. So, take a look at this design for SWAN there's an approach to making those updates with a certain amount of spare capacity so that congestion can be avoided.
- This approach that takes the optimization one step further **so that providing a pretty strong guarantee on lack of congestion even while the network data flow of changing.**



Conclusion

- In this lecture, we have discussed the **geo-distributed cloud data centers**, interaction of data centers with users and other data centers.

- Also discuss various data center interconnection techniques such as **(i) MPLS (ii) Google's B4 and (iii) Microsoft's Swan**