Cloud Networking

Fog and Cloud Computing

Domenico Siracusa, Fondazione Bruno Kessler (FBK)

29/04/2022

Introduction

Preliminary information

- Part of the slides of this lesson are taken from other slidesets produced by Prof. Fulvio Risso and Prof. Gianni Antichi
- Material
  - Dan C. Marinescu, *Cloud Computing: Theory and Practice* (Chapter 5)
- Suggested readings
  - Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, *A scalable, commodity data center network architecture,* SIGCOMM 2008.
  - S. Floyd and V. Jacobson, *Link-sharing and resource management models for packet networks*, IEEE/ACM Transactions on Networking, vol. 3, 4, 365-386, 1995.
  - L. Rizzo and G. Lettieri, *VALE, a switched ethernet for virtual machines*, ACM CoNEXT 2012
  - Other references that you will find in the slides

Contents

- Introduction to data center networking
  - Requirements, architecture, topologies
- Networking in virtualised environments

Data center networks

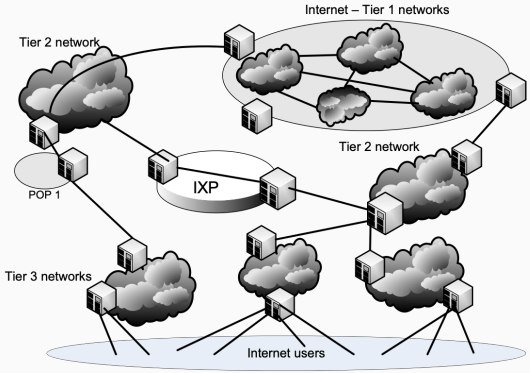
Clouds and networks

- Unquestionably, communication is at the heart of cloud computing
  - Interconnectivity supported by a continually evolving Internet made cloud computing feasible
    - Cloud is built around a high-performance interconnect
    - Servers of a cloud infrastructure communicate through high-bandwidth and low-latency networks
- Cloud workloads fall into four broad categories based on their dominant resource needs:
  - CPU-intensive
  - Memory-intensive
  - I/O-intensive
  - Storage-intensive
- Communication bandwidth goes down and the communication latency increases the farther from the CPU data travels

Relations between Internet networks i

- Three type of relations:
  - Peering - two networks exchange traffic between each other's customers freely
  - Transit - a network pays to another one to access the Internet
  - Customer - a network is paid to allow Internet access
- Networks have been commonly classified as:
  - Tier 1 - can reach every other network on the Internet without purchasing IP transit or paying settlements
  - Tier 2 - Internet service provider who engages in the practice of peering with other networks, but who still purchases IP transit to reach some portion of the Internet
  - Tier 3 - purchases transit rights from other networks (typically Tier 2 networks) to reach the Internet

Relations between Internet networks ii

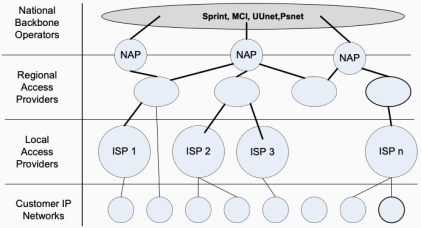


- Three classes of networks, Tier 1, 2, and 3; an IXP is a physical infrastructure allowing ISPs to exchange Internet traffic

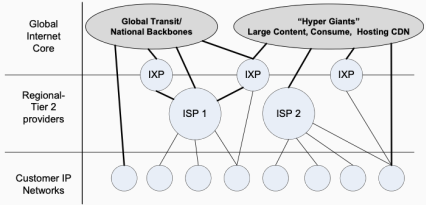
The transformation of the Internet i

- Web applications, cloud computing, and content-delivery networks reshaped the definition of a network
- Data streaming consumes an increasingly larger fraction of the available bandwidth as high definition TV sets become less expensive and content providers, such as Netflix and Hulu, offer customers services that require a significant increase of the network bandwidth
- The “last mile” - the link connecting the home to the Internet Service Provider (ISP) network is the bottleneck
- Several initiatives (e.g. Google Fiber Project) to bring 1Gb/s access speed to individual households through FTTH

The transformation of the Internet ii

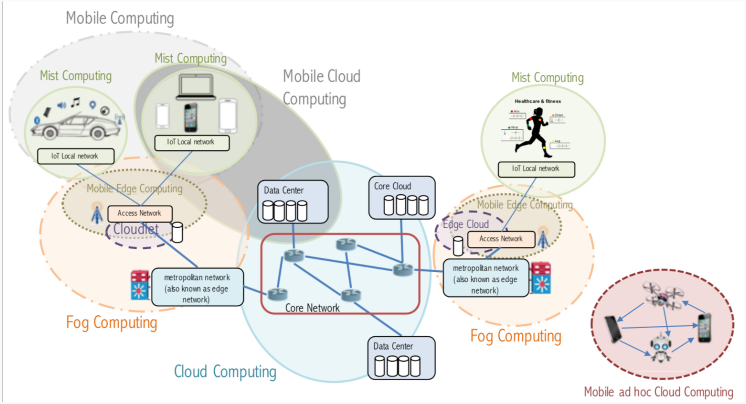


(a) Textbook Internet prior to 2007; the global core consists of Tier 1 networks



(b) The 2009 Internet reflects the effect of comoditization of IP hosting and of content-delivery networks (CDNs)

The transformation of the Internet iii



Data center networks: characteristics

- Huge scale (20k switches/routers)
- High bandwidth (10/40/100G and more)
- Very low round trip time (RTT, 1-10 microseconds)
- Limited geographic scope
- Limited heterogeneity
- Regular/planned topologies

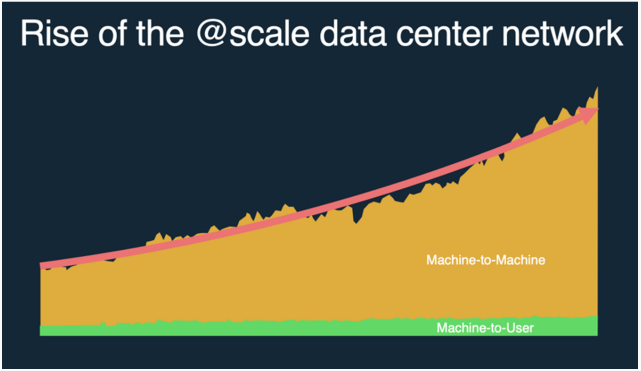
Degrees of freedom

- *Single* administrative domain
- Control over network and endpoints
- Control over placement of traffic source/sink
- In short: full control
  - Can deviate from standards
  - Can change addressing/congestion control
  - Can control routing (what traffic crosses which links)

Data center architecture: requirements i

- Applications:
  - Online analysis of data
    - Tons of small flows (e.g. 55% of flows 3% of bytes, 5% of flows 35% of bytes)
  - Low-latency user interaction
- Service model:
  - One specific application (e.g. Google, Facebook)
  - Cloud (computation/storage/infra as a service)
- Traffic directions:
  - Outward (e.g., serving web pages to users)
  - Internal computations (e.g. MapReduce for web indexing)
- Workloads often unpredictable:
  - Multiple services run concurrently within a DC
  - Demand for new services may unexpectedly spike

Data center architecture: requirements ii



- A. Moorthy (Facebook), "Connecting the World: A look inside Facebook's Networking Infrastructure", 2015

Data center architecture: requirements iii

- Example: data processing frameworks -> MapReduce
  - MapReduce is a framework for parallel computations that use potentially large data sets and a large number of nodes
  - Programmers use MapReduce to run models over large distributed sets of data and use advanced statistical and machine learning techniques to do predictions, find patterns, uncover correlations, etc.
- So?
  - Any-to-any communication -> high bandwidth is desirable
  - Latency is critical too

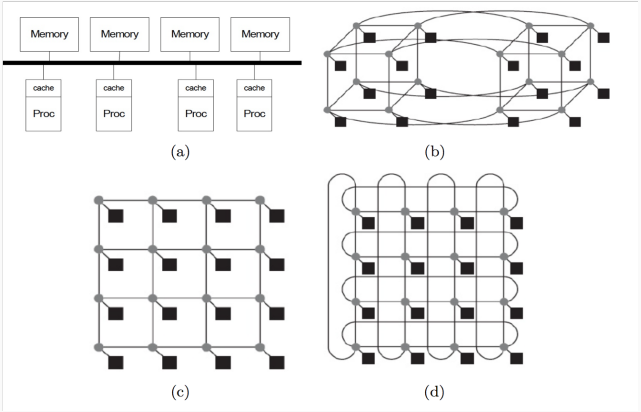
Interconnection networks: basic concepts i

- A network consists of *nodes* and links or *communication channels*
- An *interconnection network* can be:
  - Non-blocking if it is possible to connect any permutation of sources and destinations at any time
  - Blocking if this requirement is not satisfied
- *Switches* and communication channels are the elements of the interconnection fabric
  - Switches -> receive data packets, look inside each packet to identify the destination IP addresses, then use the routing tables to forward it to the next hop towards its final destination
  - An n-way switch -> has n ports that can be connected to *n* communication links
- The *degree of a node* is the number of links the node is connected to
- Nodes -> could be processors, memory units, or servers
- Network interface of a node -> hardware connecting it to the network

Interconnection networks: basic concepts ii

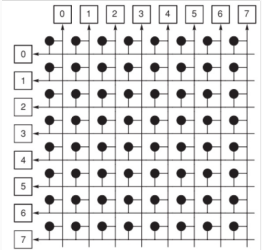
- Interconnection networks are distinguished by:
  - Topology - is determined by the way nodes are interconnected
  - Routing - routing decides how a message gets from source to destination
  - Flow control - negotiates how the buffer space is allocated
- The topology of an interconnection network determines:
  - Network diameter - the average distance between all pairs of nodes
  - Bisection width - the minimum number of links cut to partition the network into two halves
    - When a network is partitioned into two networks of the same size the bisection bandwidth measures the communication bandwidth between the two
    - Full bisection bandwidth: one half of the nodes can communicate simultaneously with the other half
- There are two basic types of network topologies:
  - Static networks where there are direct connections between servers
    - (a) Bus; (b) Hypercube; (c) 2D-mesh: (d) 2D-torus
  - Switched networks where switches are used to interconnect the servers

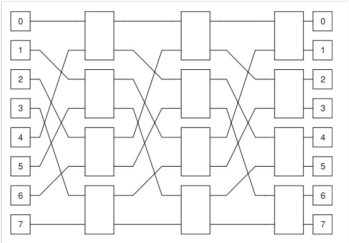
Interconnection networks: basic concepts iii



- Static networks: (a) Bus; (b) Hypercube; (c) 2D-mesh; and (d) Torus

Interconnection networks: basic concepts iv

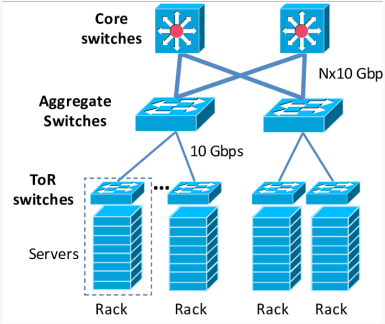




- Switched networks
  - (Left) An 8 x 8 crossbar switch
    - 16 nodes are interconnected by 49 switches represented by the dark circles
  - (Right) An 8 x 8 Omega switch
    - 16 nodes are interconnected by 12 switches represented by white rectangles

Cloud interconnection networks

- Network
  - 100000s servers (placed in racks) interconnected by
  - 1000s top-of-rack (ToR) switches interconnected by
  - 1000s aggregation switches interconnected by
  - 100s core switches interconnected to
  - Internet/backbone network
- Requirements for cloud interconnection:
  - Scalability
  - Low cost
  - Low-latency
  - High bandwidth
  - Location transparent communication



Location transparent communication

- Every server should be able to communicate with every other server with similar speed and latency
  - Applications need not be location aware
  - Reduces the complexity of the system management
- In a hierarchical organization true location transparency is practically not feasible
  - Cost considerations ultimately decide the actual organization and performance of the communication fabric

Costs

- Total cost varies
  - Upwards of $0.25B for mega data centers
  - Server costs dominate
  - Network costs significant
- From: "The Cost of a Cloud: Research Problems in Data Center Networks", ACM CCR 2009.

Amortized Cost	Component	Sub-Components
~45%	Servers	CPU, memory, disk
~25%	Power infrastructure	UPS, cooling, power distribution
~15%	Network	Switches, links, transit
~15%	Power draw	Electrical utility costs

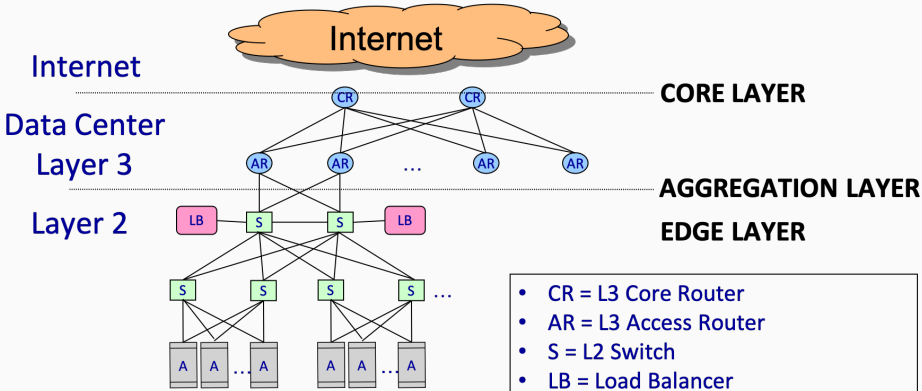
Servers costs

- Uneven application fit
  - Each server has CPU, memory, disk: most applications exhaust one resource, stranding the others
- Long provisioning timescales
  - New servers purchased quarterly at best
- Uncertainty in demand
  - Demand for a new service can spike quickly
- Risk management
  - Not having spare servers to meet demand brings failure just when success is at hand

Routers and switches costs

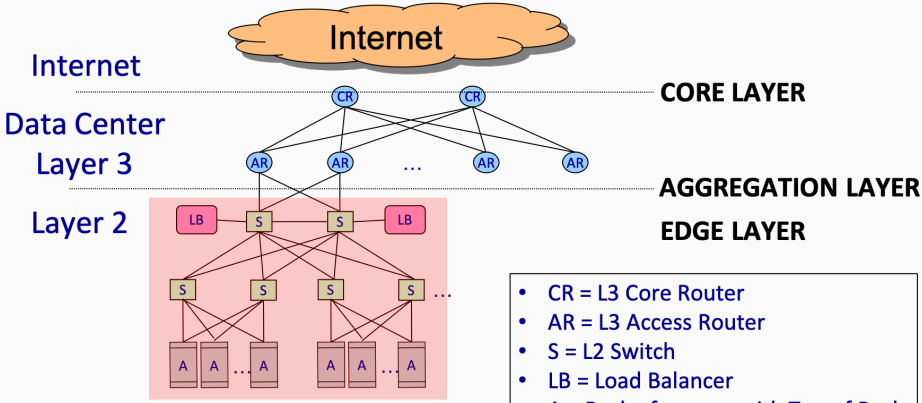
- The *cost* of routers and the number of cables interconnecting the routers are relevant components of the cost of interconnection network
- Better performance and lower costs can only be achieved with innovative router architecture
  - *wire density* has scaled up at a slower rate than processor speed and wire delay has remained constant
- Router – switch interconnecting several networks
  - *low-radix routers*: have a small number of ports
    - divide the bandwidth into a smaller number of wide ports
  - *high-radix routers*: have a large number of ports
    - divide the bandwidth into larger number of narrow ports
- The number of intermediate routers in high-radix networks is reduced
  - lower latency and reduced power consumption

Conventional data center network i



- CR = L3 Core Router
- AR = L3 Access Router
- S = L2 Switch
- LB = Load Balancer
- A = Rack of servers with Top of Rack switch

Conventional data center network ii

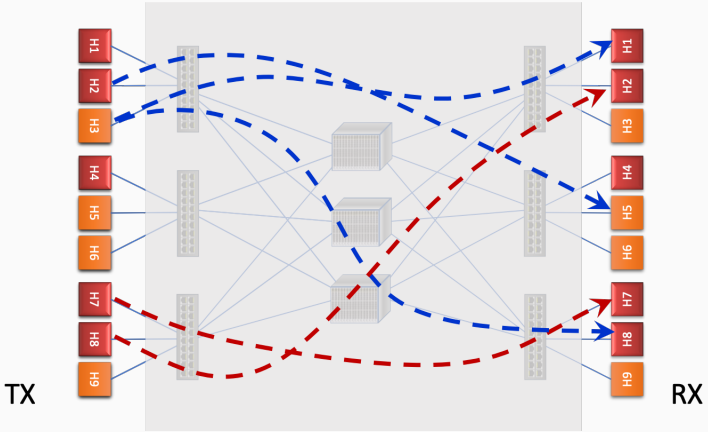


- CR = L3 Core Router
- AR = L3 Access Router
- S = L2 Switch
- LB = Load Balancer
- A = Rack of servers with Top of Rack switch

Layer 2 vs. Layer 3

- Ethernet switching (layer 2)
  - Pros
    - Fixed IP addresses and auto-configuration (plug & play)
  - Cons
    - Broadcast limit scale (ARP)
    - Spanning Tree Protocol
- IP routing (layer 3)
  - Pros
    - Scalability through hierarchical addressing
    - Multipath routing through equal-cost multipath
  - Cons
    - More complex configuration
    - Cannot migrate without changing IP address

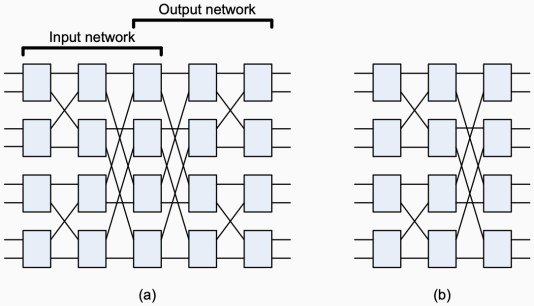
How does it look like: a giant switch



Topologies: Clos networks, InfiniBand, Myrinet, Fat trees i

- Butterfly network -> the name comes from the pattern of inverted triangles created by the interconnections, which look like butterfly wings
  - Transfers the data using the most efficient route, but it is blocking, it cannot handle a conflict between two packets attempting to reach the same port at the same time
- Clos -> Multi-stage non-blocking network with an odd number of stages
  - Consists of two butterfly networks
    - The last stage of the input is fused with the first stage of the output
  - All packets overshoot their destination and then hop back to it
    - most of the time, the overshoot is not necessary and increases the latency, a packet takes twice as many hops as it really needs
- Folded Clos topology -> the input and the output networks share switch modules
  - Such networks are called *fat trees*
    - Myrinet, InfiniBand, and Quadrics implement a fat-tree topology

Topologies: Clos networks, InfiniBand, Myrinet, Fat trees ii



- a) A 5-stage Clos network with radix-2 routers and unidirectional channels; the network is equivalent to two back-to-back butterfly networks
- (b) The corresponding folded-Clos network with bidirectional channels; the input and the output networks share switch modules

Fat trees i

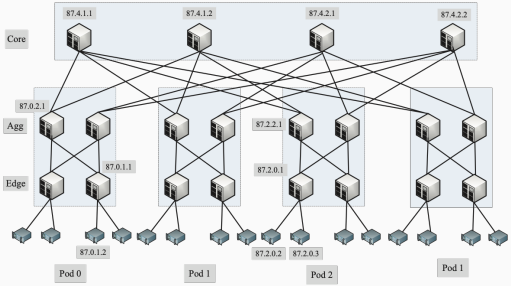
- Optimal interconnects for large-scale clusters and for Warehouse-Scale Computing
  - Servers are placed at the leafs
  - Switches populate the root and the internal nodes of the tree
  - Have additional links to increase the bandwidth near the root of the tree
- A fat-tree network can be built with cheap commodity parts as all switching elements of a fat-tree are identical

Fat trees ii

Fat-tree DCNs i

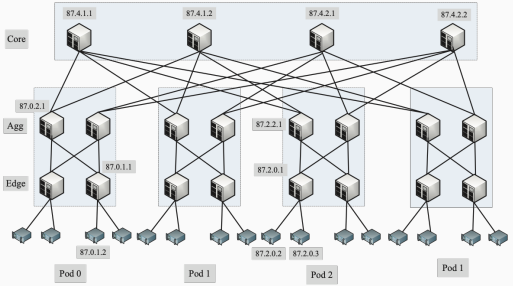
- Design principles:
  - The network should scale to a very large number of nodes
  - The fat-tree should have multiple core switches
  - The network should support multi-path routing
    - The equal-cost multi-path (ECMP) routing algorithm which performs load splitting among flows should be used

Fat-tree DCNs ii



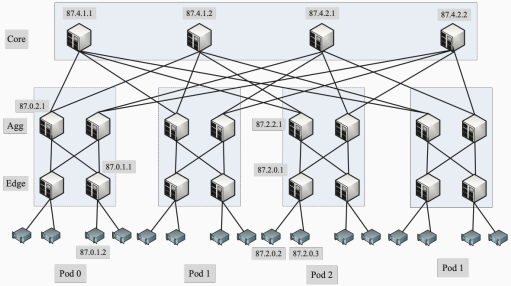
- Fat-tree interconnect consists of k pods
  - Each pod has two layers and k/2 switches at each layer
  - Each switch at the lower layer is connected directly to k/2 servers
    - Other k/2 ports are connected to k/2 of the k ports in the aggregation layer
  - # switches: k(k+1), # connected servers k^{2}
    - There are (k/2)^{2} paths connecting every pair of servers

Fat-tree DCNs iii



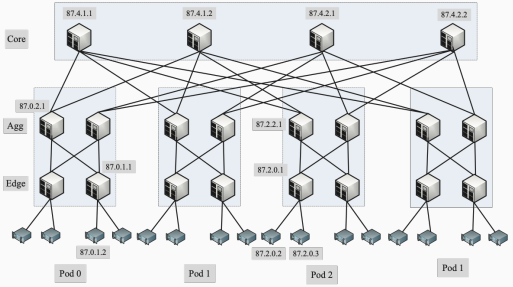
- Fat-tree interconnection network for k=4
  - Core, aggregation and edge layers populated with 4-port switches
  - Each core switch connected with one switch at the aggregation layer of each pod
- The network has four pods
  - Four switches at each pod, two at aggregation layer and two at the edge
  - Four servers are connected to each pod

Fat-tree DCNs iv



- IP addresses of edge/agg switches: *87.pod.switch.1*
  - switches are numbered left to right, and bottom to top
  - pod 2 switches (4 in number) have IP addresses 87.2.0.1 to 87.2.3.1
- Core switches IP addresses: *87.k.j.i* where k=4, *j* and *i* denote the coordinates of the switch in the (k/2)^{2} core switch grid starting from top-left

Fat-tree DCNs v



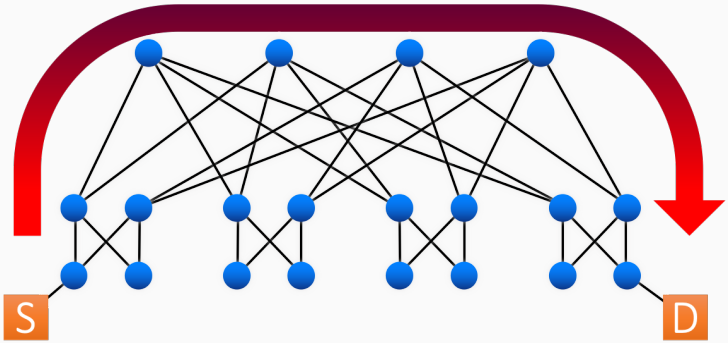
- Servers have IP addresses of the form *87.pod.switch.serverID*
  - serverID is the server position in the subnet of the edge router starting from left to right; e.g. IP addresses of the two servers connected to the switch with IP address 87.2.0.1 are 87.2.0.2 and 87.2.0.3

Open issues

- Workloads
  - Small flows (mice flows) requiring low-latency
    - e.g. query, coordination
  - Large flows (elephant flows) requiring high-throughput
    - e.g. data update, backup
- Open issues
  - How to balance these workloads
    - next slides
  - TCP does not perform really well in data center networks
    - Check: *V. Vasudevan et al., "Safe and effective fine-grained TCP retransmissions for datacenter communication", ACM SIGCOMM 2009*

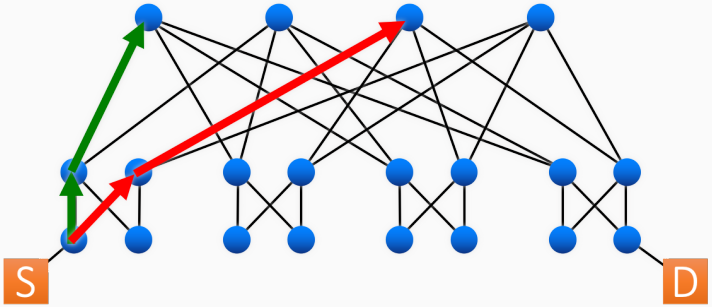
How to balance traffic flows? i

- Let us assume many flows from S to D
- Many equal cost paths going up to the core switches
- Only one path down from each core switch



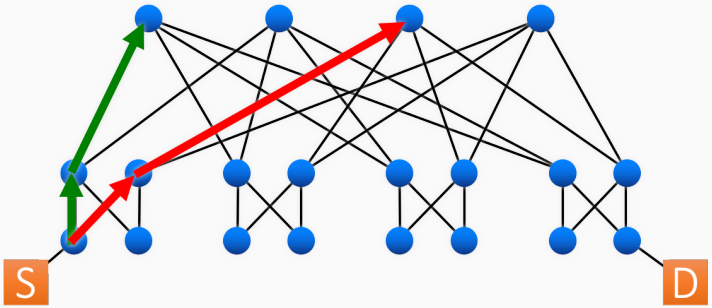
How to balance traffic flows? ii

- Equal-Cost Multi-Path routing (ECMP)
  - Randomly allocate paths to flows using hash of the flow
- Problems?



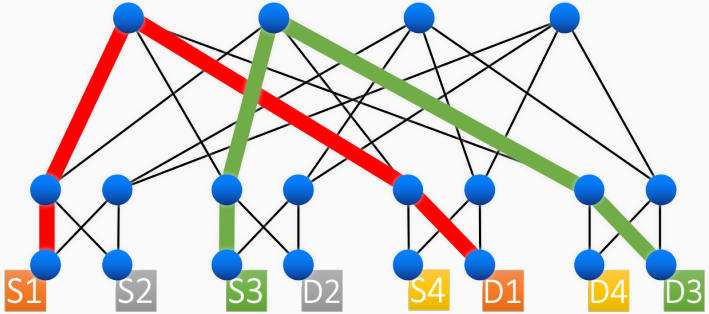
How to balance traffic flows? iii

- Agnostic to available resources
- Long lasting collisions between elephant flows



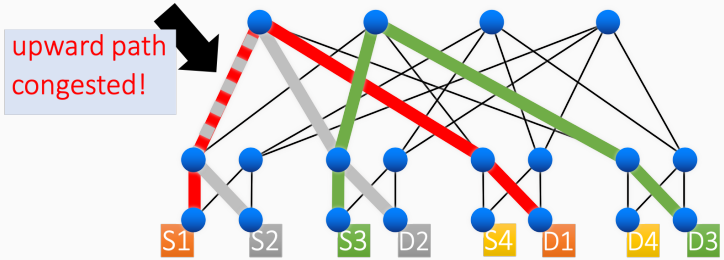
How to balance traffic flows? iv

- This is perfect :)



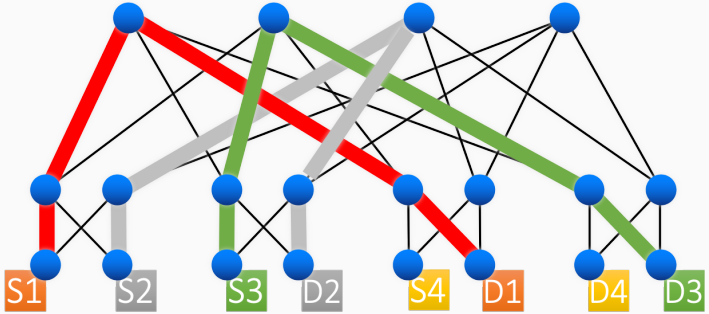
How to balance traffic flows? v

- This is not! :(



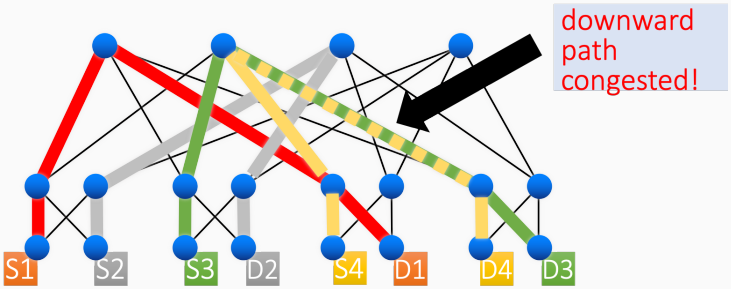
How to balance traffic flows? vi

- . . . and the network would still have more than enough capacity!



How to balance traffic flows? vii

- Warning: collisions can also take place on the downward path



From literature: Hedera

- *Mohammad Al-Fares, Hedera: Dynamic Flow Scheduling for Data Center Networks", NSDI 2010*
- Idea
  - Collect Flow information from switches to detect elephant flows
    - Flows exceeding a given threshold are considered large -> 10% of hosts' link capacity
  - Compute non-conflicting paths
  - Instruct switches to reroute traffic
  - What if only mice flows? ECMP!

Other solutions

- Why do not reroute based on link utilization?
  - *CONGA: Distributed Congestion-Aware Load Balancing for Datacenters – ACM SIGCOMM 2014*
  - Is it good for mice flows?
- What if we do based on switch queue occupancy?
  - *DRILL: Micro Load Balancing for Low-latency Data Center Networks - ACM SIGCOMM 2017*
  - Is it good for elephant flows?
- Why just not spread all the packets uniformly?
  - *Per-packet Load-balanced, Low-Latency Routing for Clos-based Data Center Networks – ACM CoNEXT 2013)*
  - Is it easy to implement congestion control?
- Can you solve all problems with routing?

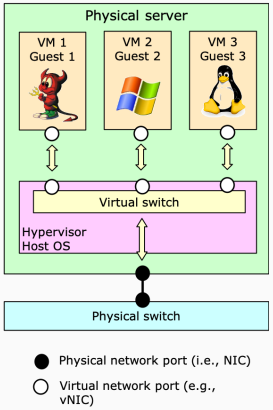
Summary

- Interconnection networks plays an essential role in the design of a data center
  - Complex requirements from applications, service models, traffic directions
  - Full control helped defining ad-hoc architectures
- Data center networks
  - Layered structure: edge, aggregation, core
  - Many topologies: fat tree quite relevant
- Open issues (research): achieve the following requirements together
  - Low latency (short messages, queries)
  - High throughput (continuous data updates, backups)
  - High burst tolerance

Networking in virtualized environments

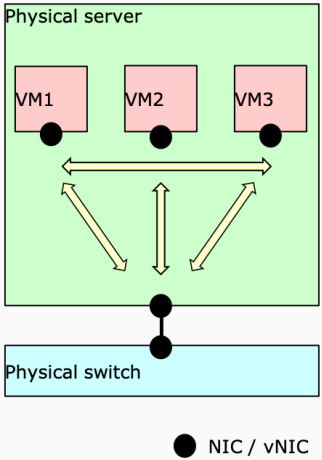
Computing virtualization in brief

- Capability to run multiple virtual machines on the same physical host
- Main software components
  - Virtual machines
  - Host OS / Hypervisor
  - Software (virtual) switch (or software bridge)
- All those components (including the virtual switch) are created by OS people



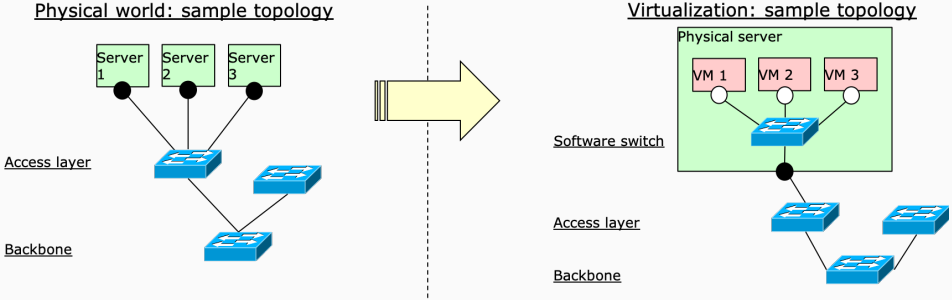
Virtualization and networking

- Virtualization introduces additional complexity for networking
- Communication requirement:
  - Deliver traffic from VMs/LXCs to the physical network (and vice versa)
  - Deliver traffic between VMs/LXCs within the same server
- Additional requirements
  - Assign IP addresses to VM/LXCs
  - Provide advanced features such as load balancing, firewall, etc.
- Note: no difference between VMs or LXCs from networking perspective



Main idea

- For the time being, let's consider a very simple networking model such as pure L2 (e.g., Ethernet only)
- Physical world: servers are connected to a switch, from here to the backbone
- Virtual world: vservers are connected to a switch, from here to the physical network



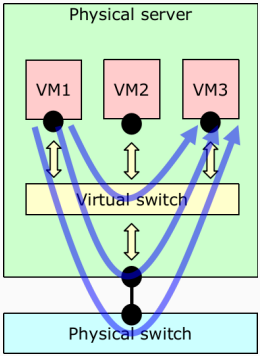
Challenges

1. Step 1: provide Ethernet (L2) connectivity to VMs/containers
   - VMs/containers can communicate together and to the external world
     - hardware and software switching
2. Step 2: provide IP connectivity to the VMs/containers
   - Native networking: use IP addresses "agreed" with the DC provider
   - Overlay networking: use whatever IP address, independently from the DC provider

North/South and East/West communications on a single server

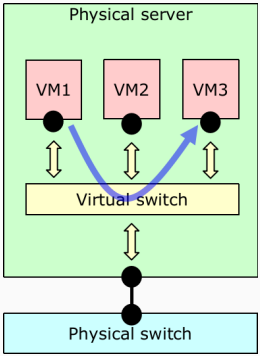
Options for North/South and East/West traffic

- Although previous slides suggest that this problem can be solved with a software switch (softswitch), in fact three main options exists:
  1. Virtual switch in the host OS / hypervisor
  2. In the NIC (using virtual queues)
  3. In the external switch (requires special support)



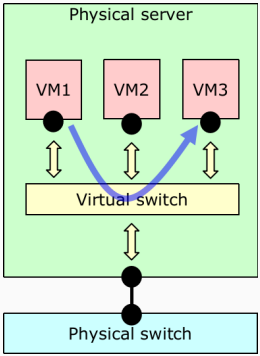
1. Host-based switching i

- Historically, the first solution used
  - In some cases the software switch is in the Host OS, in some other in the Hypervisor (e.g., in case the virtualization layer sits on top of the operating system such as in Virtualbox)



1. Host-based switching ii

- Strengths
  - High bandwidth (and reduced overhead) for inter-VM traffic
  - Enforces policies early
- Weaknesses
  - Additional processing overhead
  - Additional component (vswitch) to configure and monitor
  - Not clear who controls the switch: IT or networking people?
- Also called "software bridge" or "softswitch"

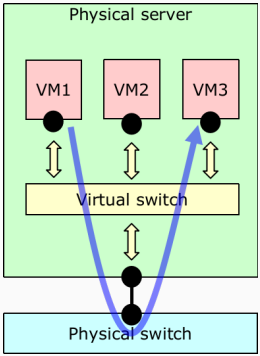


1. Host-based switching iii

- Centralized management
- Can implement the same features of hardware switches
  - Packet visibility, ACLs, Quality of Service
- Can support a large number of policy rules
- Consumes hypervisor CPU cycles
- Sometimes, even controlled by the same commands (e.g., Cisco IOS) and through the same management system that controls the rest of the infrastructure
  - vSwitch can be seen as a native piece of the infrastructure
- Possible hardware offloading can provide big wins
- Examples
  - VMware vSwitch, Cisco Nexus 1000V, Open vSwitch
- Physical NIC to operate in promiscuous mode (packets to VMs may be dropped because the destination MAC address does not match the MAC of the physical NIC)

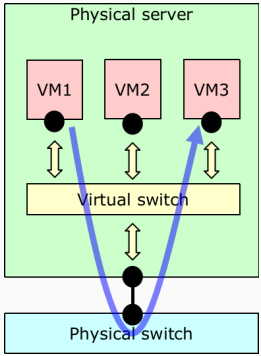
2. Hairpin switching i

- “hairpinning”: traffic makes a U-turn -> “hairpin”
- Use hardware that is already present
  - However, hardware must be compatible with hairpin switching: standard 802.1D bridges never forward a frame on the same port on which it has been received



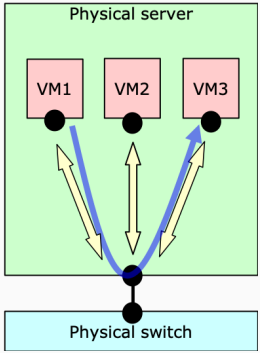
2. Hairpin switching ii

- Performance
  - Can leverage the power of the hardware (i.e., forwarding speed)
  - Not consume CPU cycles, consumes PCI bus BW (smaller than CPU-to-MEM one)
  - Traffic crosses twice intermediate components (hypervisor, NIC, physical link)
    - Best when having little VM-to-VM traffic
- Attractive when applying similar policies over all nodes
  - Clear separation between computing and networking domains
- No longer used



3. NIC switching

- Intermediate solution between host and hairpin switching
  - Intermediate also in terms of strengths and weaknesses
- Several NICs available supporting this feature
- Most important advantage: avoids the problem about who controls edge switches
  - NICs are not under the control of the Operating System, hence can be considered part of the network infrastructure
- More and more SmartNICs from datacenter vendors (e.g., Broadcom, Mellanox, Netronome, Pensando, etc.)



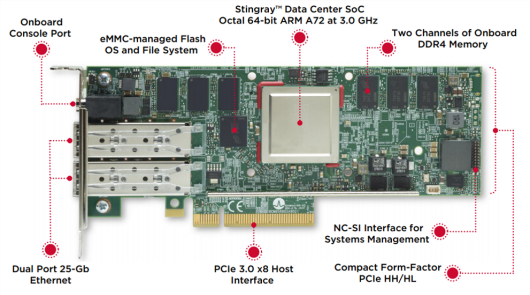
- SmartNIC is an intelligent server adapter (ISA) (e.g., NIC) that:
  - Can implement complex server-based networking data plane functions
    - e.g. multiple match-action processing, tunnel termination and origination, metering and shaping and per-flow statistics
  - Supports a fungible data plane either through updated firmware loads or customer programming, with little or no predetermined limitations on functions that can be performed
  - Works seamlessly with existing open source ecosystems to maximize software feature velocity and leverage

Why a SmartNIC?

- When amount of CPU spent in a server due to networking tasks becomes large, leaving little or no CPU left over to run applications (e.g., VMs/containers), consider offloading that computation to a discreet component
  - A similar pattern was observed with GPU, which were intended to offload graphical tasks in computers in which that load was significant
  - As a consequence, some servers may not benefit from SmartNICs (e.g., when the processing load due to networking is not significant), hence can stay with the traditional software switch
- The networking may be consuming a significant amount of CPUs
  - Widespread used overlay tunneling protocols (e.g., VxLAN) introduce a noticeable processing cost
  - Detecting and blocking DDoS attacks (which are often handled by datacenter servers; a "centralized" firewall may be difficult to scale) may consume a significant amount of CPU
  - Complex functions (switching, load balancing, etc) may consume a lot of CPU as well

SmartNIC example: Broadcom

- PS225 - Dual-Port 25GbE PCIe Ethernet SmartNIC
- Eight 64-bit ARM Cortex-A72 cores at 3.0 GHz with 16 MB cache
- 100 Gb/s cryptography engine with single-pass hashing and encryption/decryption
- ... and more -> `https://www.broadcom.com/products/ethernet-connectivity/smartnic/ps225`



Hardware vs Software switching: performance

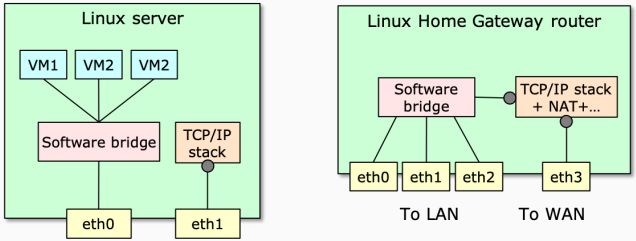
- Software switches have been demonstrated at Nx10Gbps
  - Although with significant CPU cost, stolen from VM processing
    - But... are we creating this infrastructure to achieve fast switching processing or to support many VMs?
- Software switches can drop traffic closer to the source
  - Important in clouds with over-subscribed links and untrusted sources
- Hardware offloading for software switches can provide big wins
  - Checksum offloading
  - SR-IOV may be even better
- Software switches are better for local VM-to-VM traffic

Summary

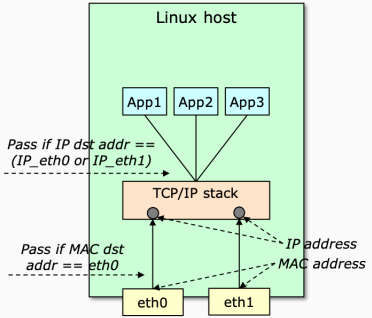
- Hardware switches attractive when applying similar policies over all nodes or in aggregate with little local VM-to-VM traffic
  - Clear separation between computing and networking domains
- Host switches provide more flexibility and fine-grained control at cost of hypervisor CPU cycles
  - Not clear separation between computing and networking domains
- Host switches is currently the most common solution
  - Easy to add new features (e.g., added value services such as mobility), without requiring explicit support from the network
  - Commercial interests are pushing for this solution, adding the most intelligence we can in them, leaving the network as a dumb pipe (just provide basic connectivity)

Software bridges (virtual switch) in Linux

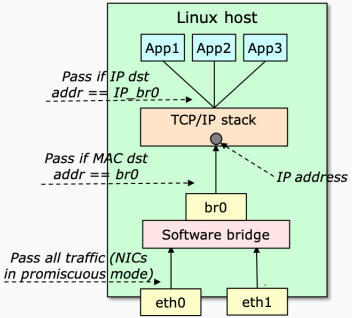
- Why do we need a software bridge?
  1. Provide intra-host connectivity to execution containers (e.g., different network namespaces, VMs, Docker, Linux Containers, etc)
  2. Handle IP addresses differently from the "official" IP model
     - "One IP address per NIC" rule
- Some examples:



- Linux allows to create a bridge across the interfaces, and one unique (virtual) interface that is valid for the entire host



Traditional networking: IP addresses assigned to the NIC



Bridged networking: IP addresses assigned to the bridge

Basic bridging networking in Linux ii

- In the past: the setup of a bridging process in Linux was a way to have an 802.1D bridge, running in software, that was part of a bigger network
  - We can modify the bridge code
  - We can play with a bridged network without having to buy a physical device
- Now: it represents a way to overcome one of the limitations of the IP protocol, which requires a distinct IP address on each interface
  - Since the default gateway is unique for the entire host, it is hard to use multiple paths
    - It requires an ICMP redirect or the manual definition of a specific route
  - If allowed by the STP, all the NICs can be used to rx/tx traffic

Software bridges in Linux

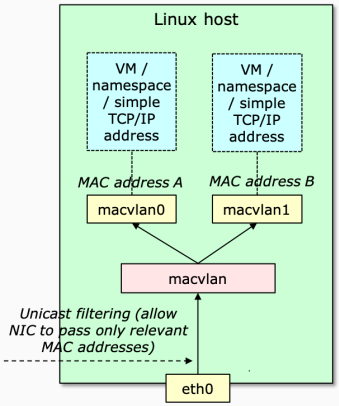
- Linux has three main types of software bridges
  - Linuxbridge
  - macvlan
  - Open vSwitch

Linuxbridge

- Most common software bridge: *linuxbridge*
- Behaves like a traditional hardware switch (i.e., IEEE 802.1D)
  - Has filtering DB (FDB), spanning tree (STP), etc.
- NICs in promiscuous mode allow to receive all the packets
  - Filtering based on the MAC address of the br0 virtual device
- Traffic can be sent to the network using both network interfaces
- The IP address configured on the bridge is reachable from both interfaces
  - Looks like a "loopback" address

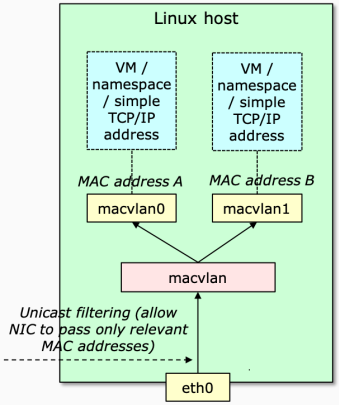
Macvlan i

- Implements a VLAN-like behavior by using MAC addresses instead of 802.1Q tags
- Looks like a (L2) subinterface derived from the main NIC
- Each macvlan interface
  - Has its own MAC address
  - Can have a distinct IP address
- Applications can bind to a specific interface / IP address
  - LXC guests can use one side of that interface, by moving the macvlan interface in their namespace



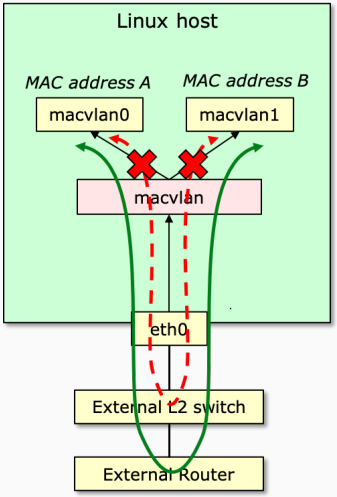
Macvlan ii

- Supports four operating modes
  - Private
  - Virtual Ethernet Port Aggregator (VEPA) - Default mode
  - Bridge
  - Pass-through
- Can configure the NIC to filter unicast packets based on MAC address instead of running in promiscuous mode (except for pass-through)
  - Requires explicit NIC support



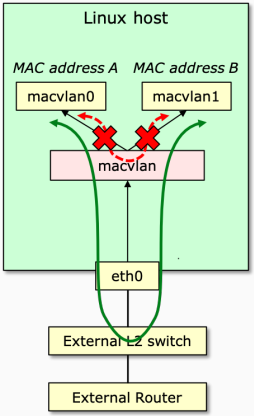
Macvlan: private mode

- VLAN-like behavior
- Forbids any inter-macvlan traffic
  - I.e., MAC A cannot send data to MAC B
- Traffic will be delivered only if it comes with a different source MAC address
  - E.g., from MAC A to a router and then to MAC B



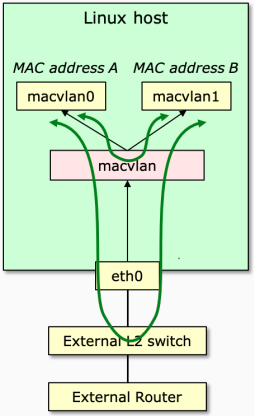
Macvlan: VEPA mode

- In this case MAC A is allowed to talk with MAC B only if the traffic comes from the external world
  - i.e., the Macvlan driver cannot send traffic from MAC A to MAC B
  - However, if the same frame is received from an external switch (it may also be the L2 switch on the NIC), the packet is delivered correctly
- The problem is that the external switch cannot be a traditional 802.1D bridge
  - 802.1D does not allow a frame to be sent back to the same interface on which it was received



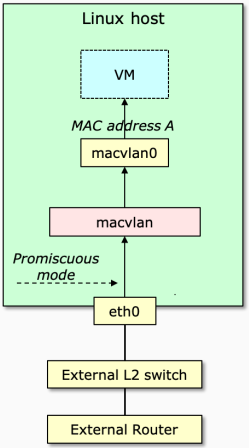
Macvlan: bridge mode

- Allows forwarding of all the traffic
- The Macvlan driver acts as a (simple) bridge
  - No filtering database (i.e., MAC learning)
  - No STP
  - Only one uplink



Macvlan: pass-through mode

- Supports only one macvlan device (single VM)
- Allow all traffic to be sent to the upstream component (e.g., VM)
  - Used mainly for VMs (as macvtap)
- NIC has to work in promiscuous mode
- Allow the upstream component to use any MAC address
  - No MAC filtering in the macvlan driver, nor in the NIC
  - Traffic is always sent upstream, and that component (e.g., the VM vNIC driver) will be in charge of the filtering
- MAC A is no longer used



Bridge vs Macvlan i

- Linux bridge
  - Behaves like a traditional hardware switch
- Macvlan: trivial bridge -> simple and fast
  - No need to do learning (or STP) as it knows every mac address it can receive
  - Uses less host CPU and provides slightly better throughput.

Bridge vs Macvlan ii

- Use Macvlan if:
  - Just need to provide egress connection to the physical network to your VMs or LXCs
- Use Bridge if:
  - Need to connect VMs or containers on the same host
  - For complex topologies with multiple bridges and hybrid environments
    - e.g. hosts in the same Layer2 domain both on the same host and outside the host
  - Advanced flood control, FDB manipulation, etc.

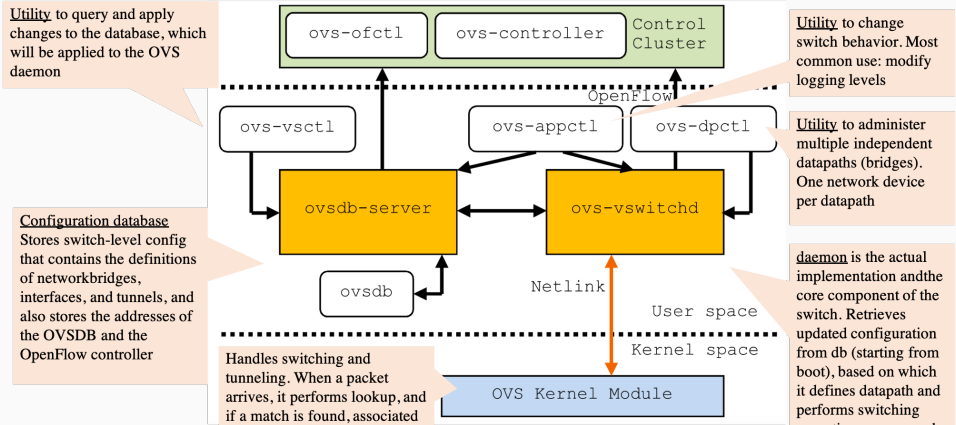
Open vSwitch (OvS)

- Open vSwitch (OVS) is a software implementation of a virtual multilayer network switch, designed to enable effective network automation through programmatic extensions
- Supports standard management interfaces and protocols such as NetFlow, sFlow, and more
- Designed to support transparent distribution across multiple physical servers by enabling creation of cross-server switches in a way that abstracts out the underlying server architecture
- Can operate both as
  - software-based network switch running within a virtual machine (VM) hypervisor
  - control stack for dedicated switching hardware

OVS implementations i

- Ported to multiple virtualization platforms, switching chipsets, and networking hardware accelerators
- Integrated into cloud computing software platforms and virtualization management systems (e.g. OpenStack, OpenNebula)
- Implemented in Kernel

OVS implementations ii

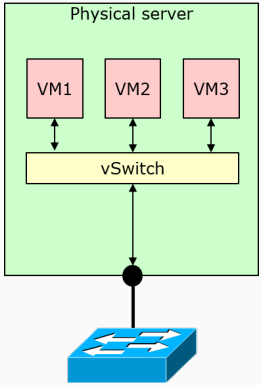


Source: M. Anan et al., *Empowering networking research and experimentation through Software-Defined Networking*, Journal of Network and Computer Applications, 70, 140-155, 2016

Single server: complex services

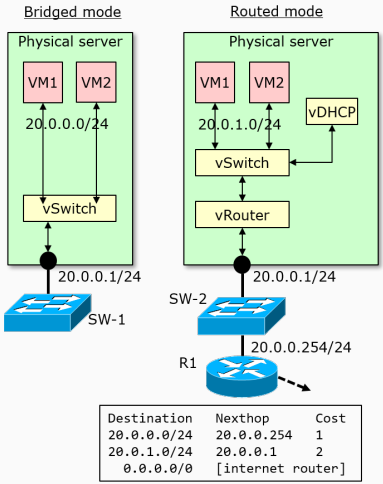
Several problems exists, not considered so far

1. Need to determine who provides IP address to VMs, and which addresses can be assigned
2. Providing connectivity may require more network services than just setting up a bridge (softswitch)
3. Multi-tenancy (i.e., strong network isolation) is a "must" in a virtual environment
4. Tenants may require additional network services within their own virtual setup
5. The server has its own TCP/IP stack (and applications, e.g., for SSH access, management, etc) that has to coexist with virtual network services



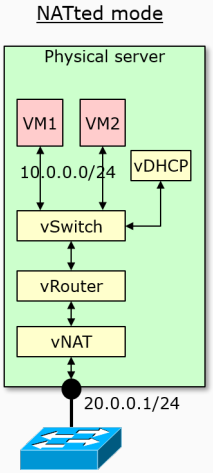
1. IP address assignment i

- Who provides IP addresses for VM/dockers?
- Option (1): Use addresses known by the physical network (direct routing)
  - Use the same addresses of the physical network (e.g., VMs attached in “bridged” mode, i.e., with pure L2 connectivity)
  - Pros: seamless inbound/outbound connections towards VMs
  - Cons: require the coordination of the (physical) network manager
    - This may limit the agility of the virtualization

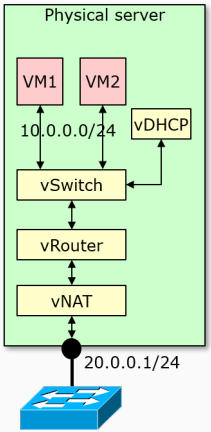


1. IP address assignment ii

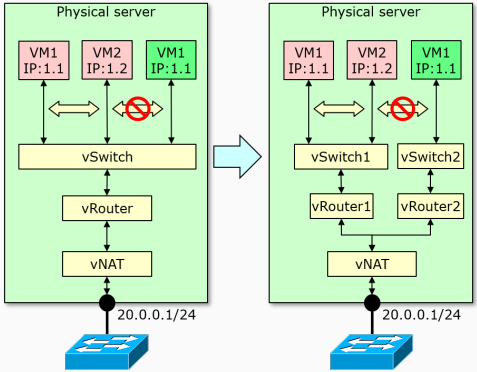
- Option (2): use private addresses and connect to the Internet through NAT
  - Pros: Work with any IP address range
    - Requiring no coordination with the (physical) network manager
  - Cons: Seamless outbound connectivity, but a NAT static rule is required to support inbound traffic
  - Cons: VMs reached with different IP addresses, depending if the sender is *inside* or *outside* the private network



- As evident from previous slide, additional network services may be required to connect the VM/Docker to the physical infrastructure
- If private addresses are used for VMs, hypervisor must also provide Router, NAT and DHCP
  - Those services are usually provided "transparently" by the infrastructure (e.g., hypervisor)
- Implementation
  - Leverage Linux kernel whenever possible (e.g., for bridging, routing, firewall/NAT (iptables))
  - Additional LXC/Docker can be used for other services (e.g., DHCP, with *dnsmasq*)

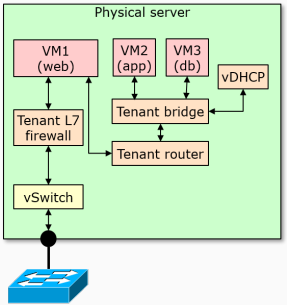


- Multi-tenancy (i.e., strong network isolation) is a "must" in a virtual environment
  - To protect communications (A cannot talk to B)
  - To prevent IP address conflicts (e.g., same IP address range for both tenants)
- Usually achieved by setting up multiple vSwitches and vRouters
  - VLANs are also possible, but why use VLANs when we can create as much virtual hardware as we want?



4. Tenant-defined network services

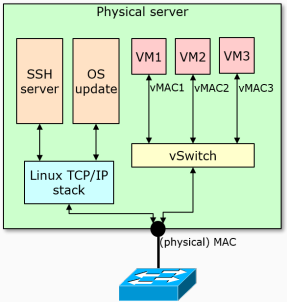
- Tenant may require a complex topology for its services
  - E.g., tenant deploys a service with 3 VMs (web server, application server, database), but only the web frontend (VM1) must be directly reachable from Internet, for increased security
- Implementation
  - Cloud controllers (e.g., OpenStack) provide some pre-defined services (e.g., router, bridge, NAT), often using Linux kernel functionalities or LXC/Docker
    - Tenant can simply use them as "black box", whatever implementation is used
  - Additional services can be set up by the tenant itself (e.g., VM with the proper software, such as for the L7 firewall)
    - For the hypervisor, those are just VMs



Note: for simplicity, this example does not support VM2/VM3 to connect to the Internet, e.g., to download OS updates

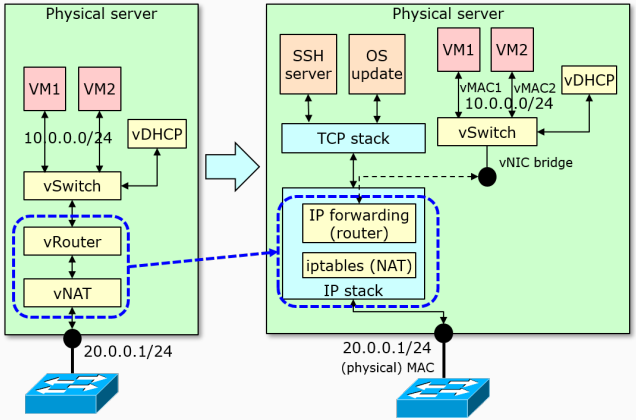
5. Co-existence of virtual services and host applications

- Server has its own TCP/IP stack (and apps, e.g., for SSH access, etc.) that must coexist with VMs
- Some observations
  - In case of "bridged" VMs, NIC MAC filtering (active by default) must be disabled to allow the vSwitch to receive inbound Ethernet traffic for VMs (with their own vMAC)
  - There should be a way to distinguish inbound traffic toward VMs (e.g., through MAC addresses or IP addresses) from the one directed to Linux
    - In case of VMs using private IP addresses it may be more difficult as we should use a L4 session table
- Linux networking stack: opportunity



Leveraging the kernel network stack for ancillary services i

- Nowadays, the Linux kernel is often used also to provide virtual network services
  - This justifies the large amount of work recently ongoing in the Linux kernel networking community
- How?
  - Complete example in next slide

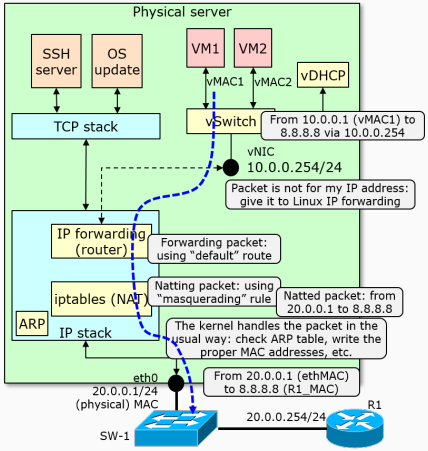


Leveraging the kernel network stack for ancillary services ii

- Packet sent from VM1 to Internet
  - Technically, to the default gateway
- The vNIC receives the packet, it detects that it is not the final recipient, so the packet is transferred to the Linux IP forwarding module
  - vNIC is device driver (runs in kernel)
- Kernel forwards it to the next hop based on the host routing table and ARP table (usual)

```
$ sudo ip route
default via 20.0.0.254 dev eth0 proto dhcp
20.0.0.0/24 dev eth0 proto kernel scope link src 20.0.0.1
10.0.0.0/24 dev vNIC0 proto kernel scope link src 10.0.0.254
```

```
$ sudo iptables -t nat -L
target      prot  source          destination
MASQUERADE  tcp   10.0.0.0/24   !10.0.0.0/24   masq ports: 1024-65535
MASQUERADE  udp   10.0.0.0/24   !10.0.0.0/24   masq ports: 1024-65535
MASQUERADE  all   10.0.0.0/24   !10.0.0.0/24
```

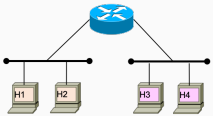


DC-wide services

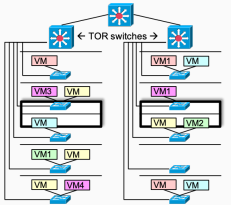
- When moving from a single server to a datacenter, in principle we must solve the same problems presented before
  1. IP addresses assignment
  2. Providing feature-rich network connectivity
  3. Support for multi-tenancy
  4. Support for tenant-defined network services
  5. Support (and integration) with the host TCP/IP stack
- The challenge is that everything must be provided "at scale"
  - Datacenters (clusters) can include thousand of servers

Tenant vs cloud manager view

- Tenant view
  - Tenants want to deploy their "virtual services" and must ignore the physical infrastructure (how many servers we have, how are they connected)
- Cloud manager view
  - Physical (physical network, servers and hypervisors) infrastructure must be able to provide the requested "logical" view
  - Additional problems (e.g., VM migration used to optimize computing tasks, energy consumption, etc.) may further complicate the picture



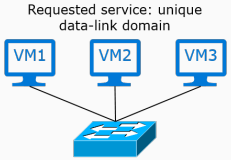
Physical world: the network derives from the physical topology



Virtual world: the network does not depend from the physical topology

Providing L2 connectivity to tenant services across DC

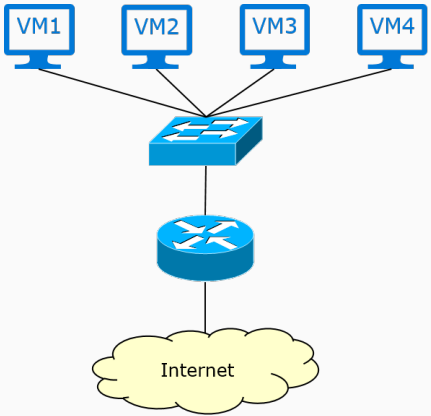
- Let's pretend the customer wants to have a vanilla L2 network, spanning among all its services across the entire datacenter
  - Frequent with OpenStack, where we want to provide highly customizable tenant networks
- In this case, the tenant wants to have control over IP addresses (it gets an L2 infrastructure, so L3 is under its own responsibility)
  - Consequently, IP addresses for VMs/containers are necessarily private
  - Decoupled addressing in the real (network) and virtual (VMs/containers) space
- Solution: tenant's traffic is tunneled between servers



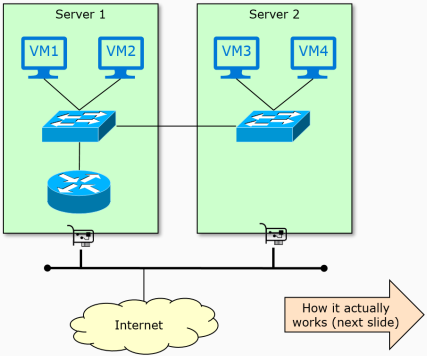
IP addresses assignment	(usually) private addresses
Feature-rich network connectivity	Required
Multi-tenancy	Required
Tenant-defined network services	Required
Integration with the host TCP/IP stack	Yes (usually tunneling)

Example: using tunnels to extend L2 tenant network across DC

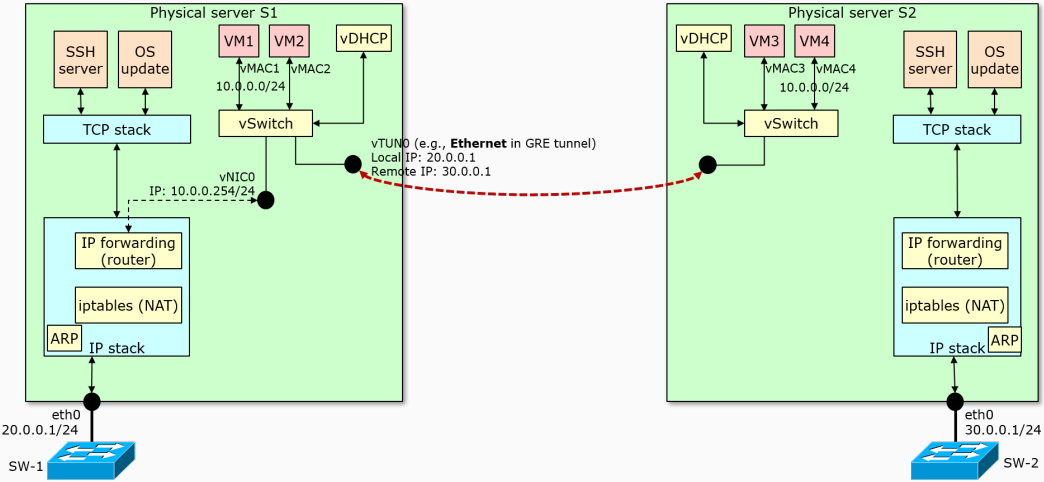
Requested service: tenant view



Requested service: cloud manager view

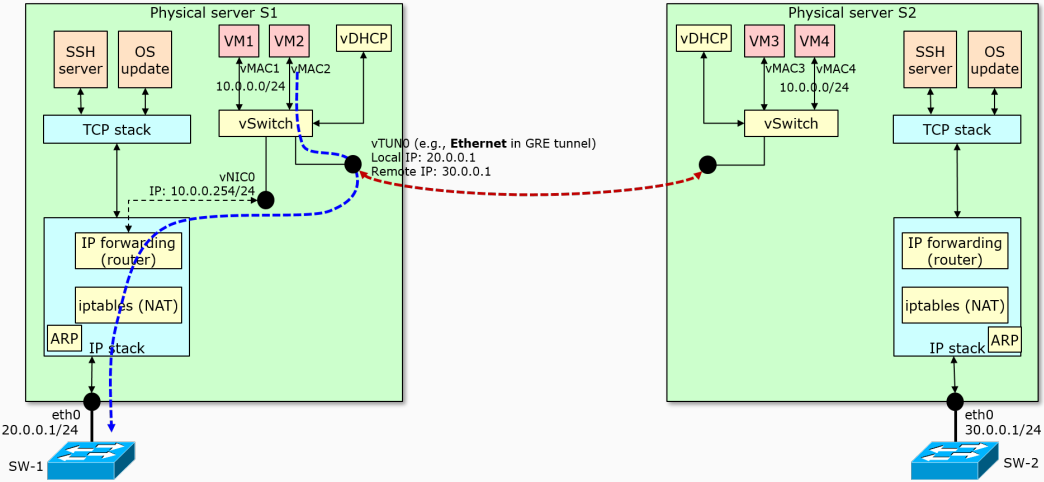


Example: using tunnels to extend L2 tenant network across DC i



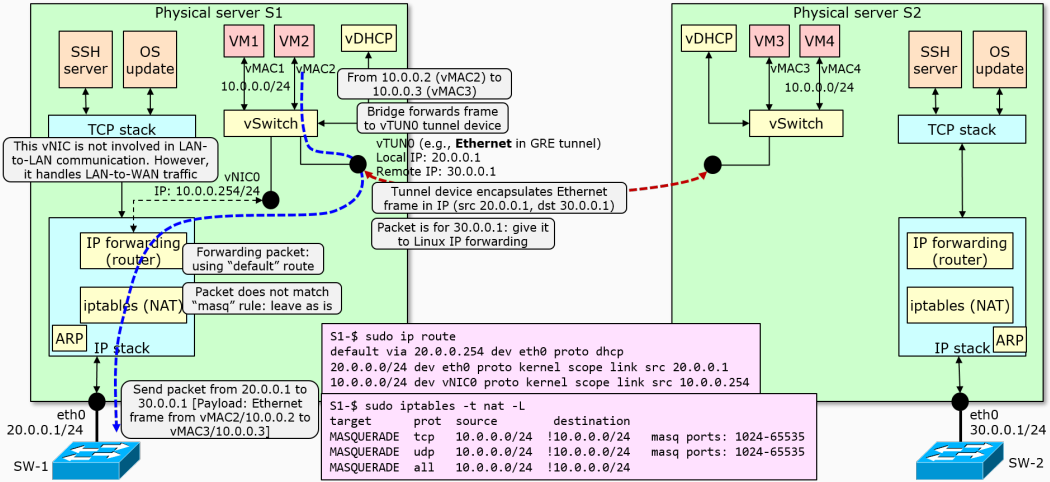
Routing and NAT tables referred to Server S1 – Very similar tables are on S2

Example: using tunnels to extend L2 tenant network across DC ii



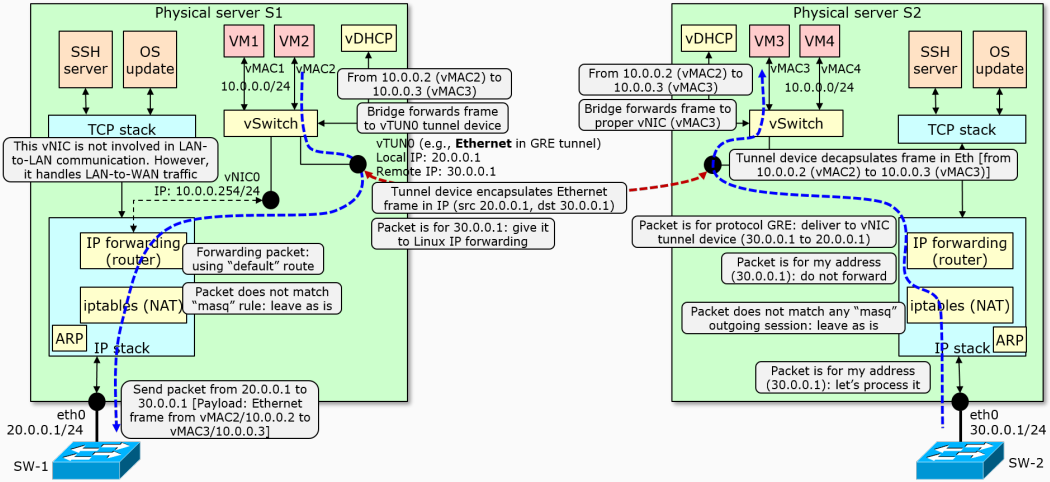
Routing and NAT tables referred to Server S1 – Very similar tables are on S2

Example: using tunnels to extend L2 tenant network across DC iii



Routing and NAT tables referred to Server S1 – Very similar tables are on S2

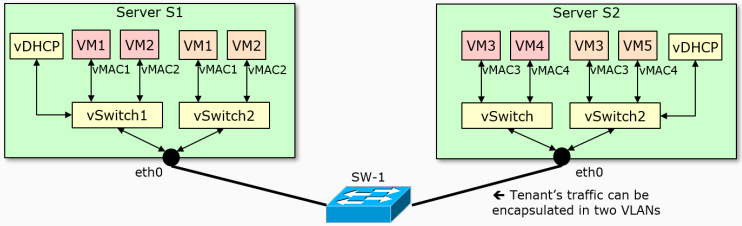
Example: using tunnels to extend L2 tenant network across DC iv



Routing and NAT tables referred to Server S1 – Very similar tables are on S2

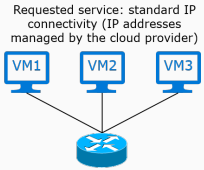
Would VLAN be appropriate to extend tenant's L2 networks?

- There are several reasons why this would not work properly
  - It require the cooperation of the (physical) network IT manager
  - VLANs are limited (4096); QinQ or similar should be used, complicating this solution
  - VLAN cannot be propagated with L3 physical networks (happens in large DCs)
    - Even worse, the physical network could a WAN link (e.g., geo-distributed datacenters)
  - What about the traffic toward Internet? (need a "gateway node", as in OpenStack)
  - What about the traffic to configure the server? (need another NIC, as in OpenStack)



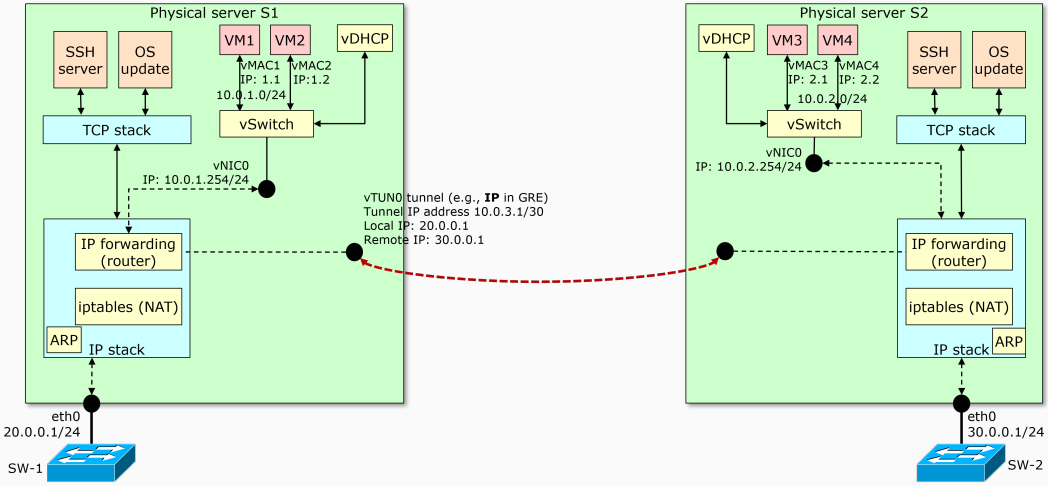
Providing L3 connectivity to tenant services across DC

- Let's pretend the customer just asks for L3 connectivity and does not care about having an L2 network nor which IP addresses are used
  - This represents the "Kubernetes" case, with loosely customizable tenant networks
- In case the orchestrator can deliver sophisticated network services to provide connectivity, which are not under the control of the tenant
  - The tenant cannot create its own network services either
- In this case, two working modes are possible: tunneling and direct routing



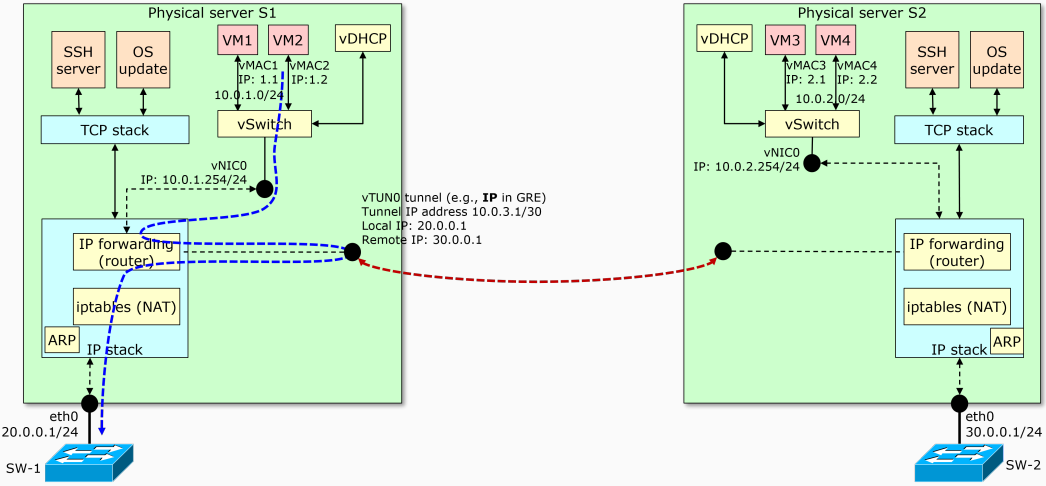
	First option	Second option
IP addresses assignment	Private addresses	Routable addresses
Feature-rich network connectivity	Required	Required
Multi-tenancy	Required	Required
Tenant-defined network services	Not supported	Not supported
Integration with the host TCP/IP stack	Yes (tunneling)	Yes (direct routing)

Example: using tunnels to extend L3 tenant network across DC i



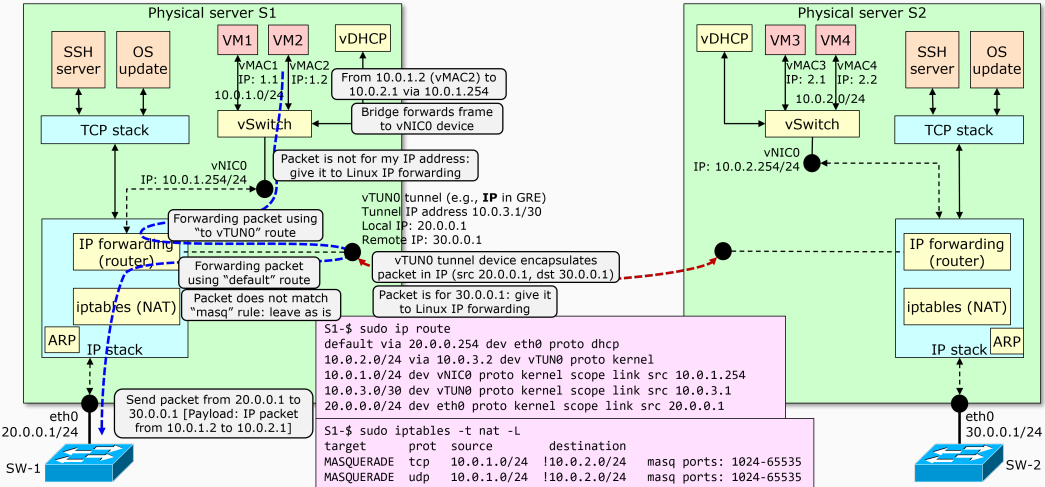
Routing and NAT tables referred to Server S1 – Very similar tables are on S2

Example: using tunnels to extend L3 tenant network across DC ii



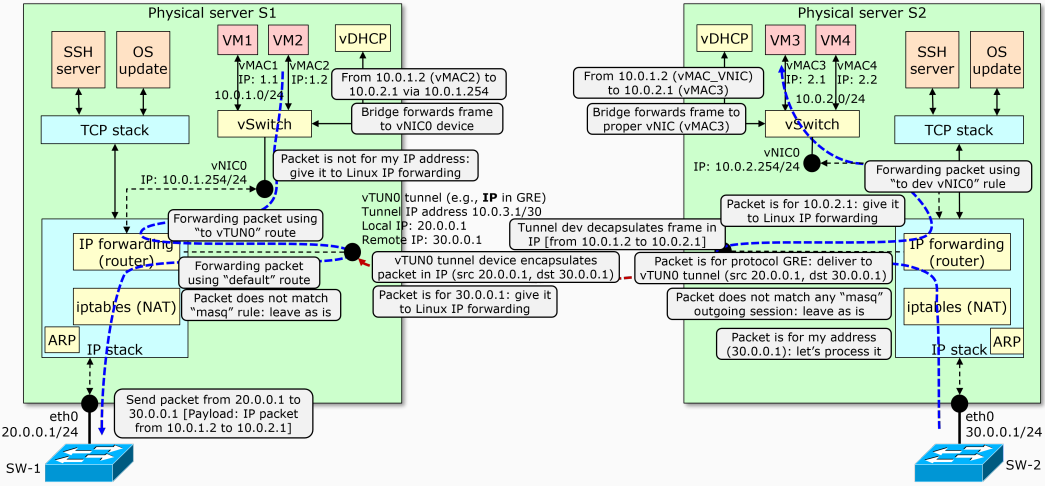
Routing and NAT tables referred to Server S1 – Very similar tables are on S2

Example: using tunnels to extend L3 tenant network across DC iii



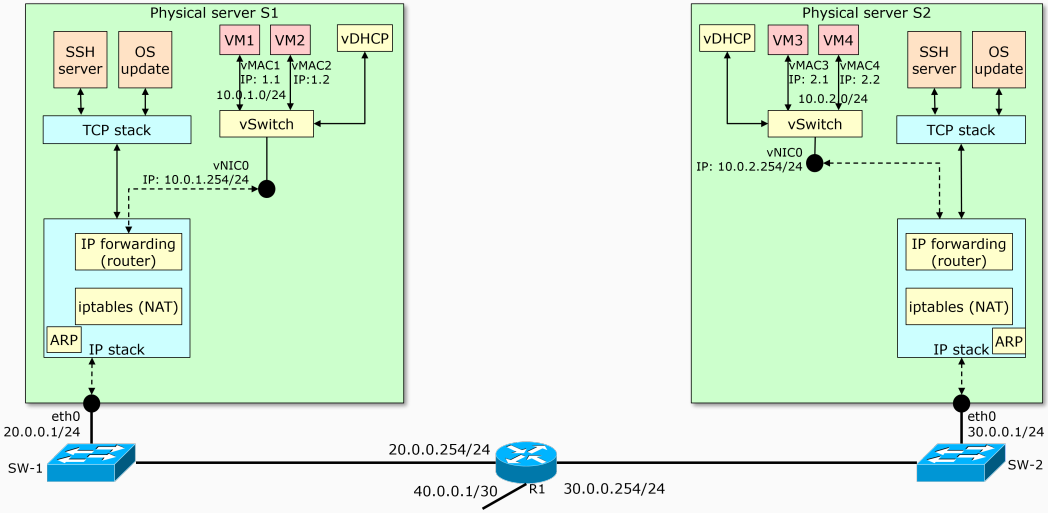
Routing and NAT tables referred to Server S1 – Very similar tables are on S2

Example: using tunnels to extend L3 tenant network across DC iv

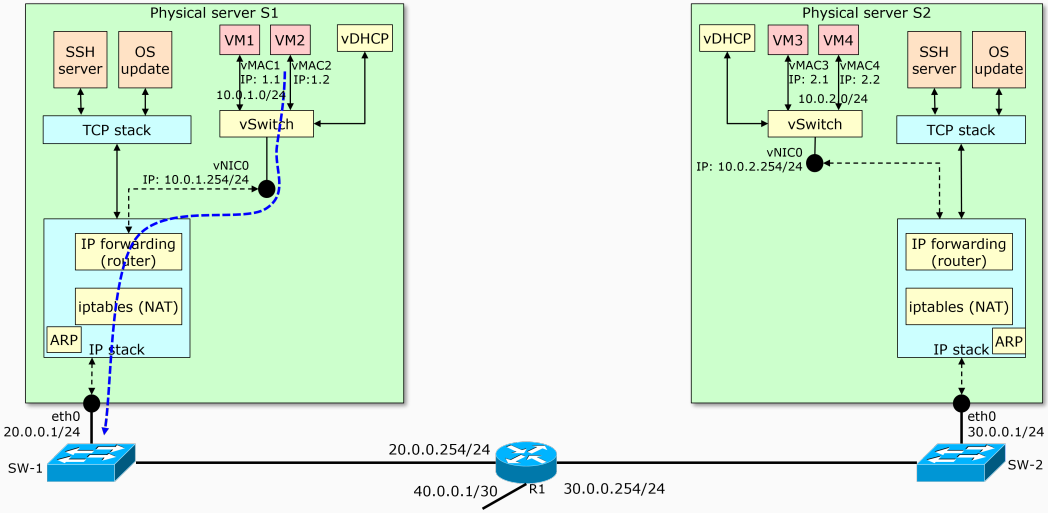


Routing and NAT tables referred to Server S1 – Very similar tables are on S2

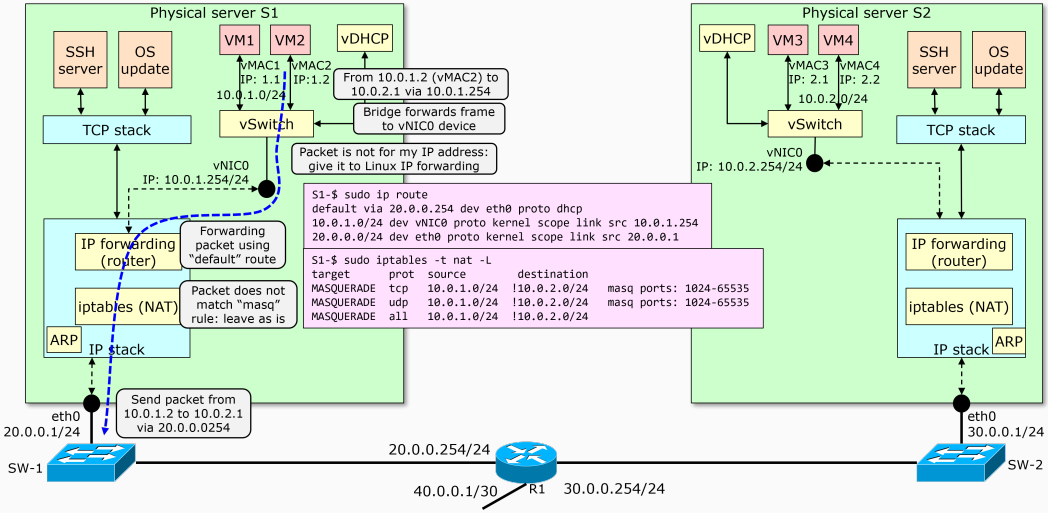
Example: using direct routing to extend L3 tenant network across DC i



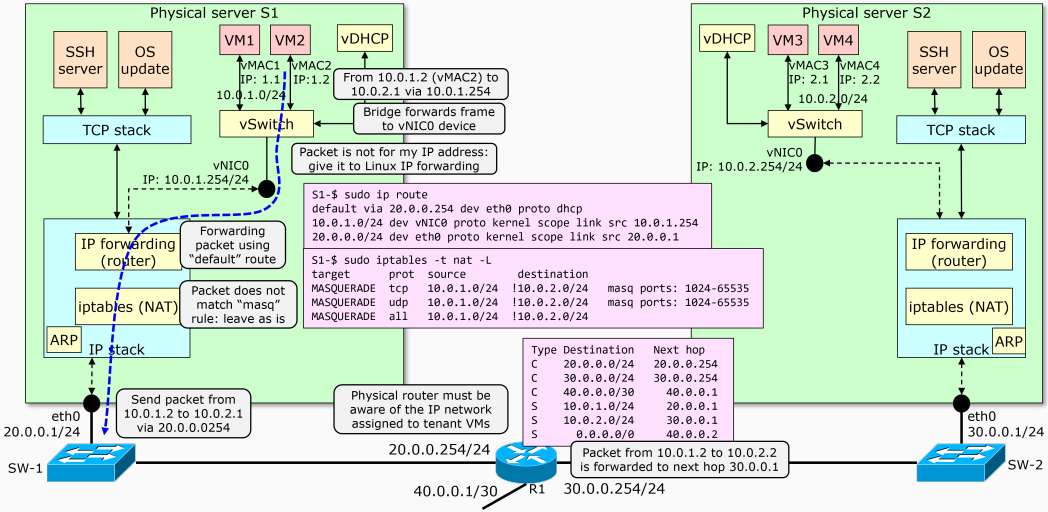
Example: using direct routing to extend L3 tenant network across DC ii



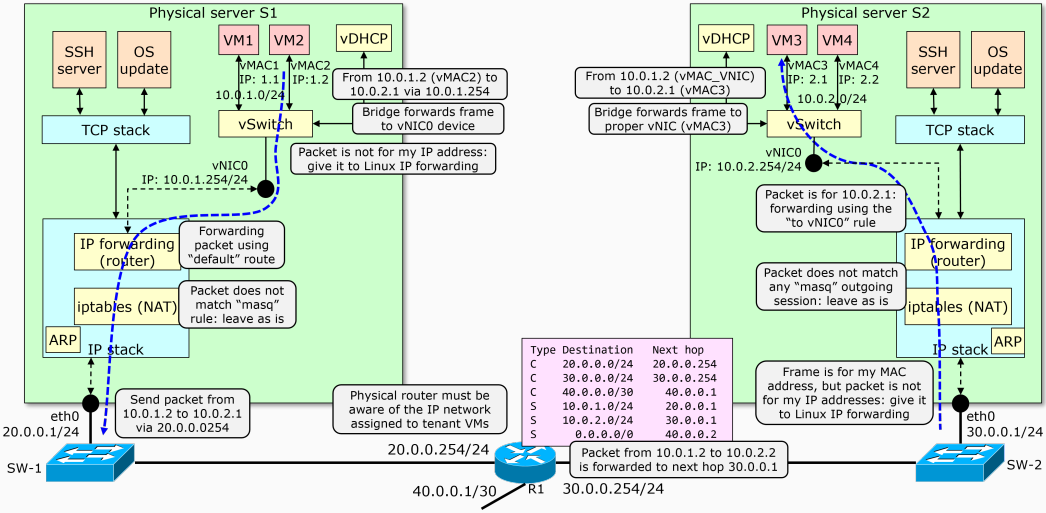
Example: using direct routing to extend L3 tenant network across DC iii



Example: using direct routing to extend L3 tenant network across DC iv



Example: using direct routing to extend L3 tenant network across DC v

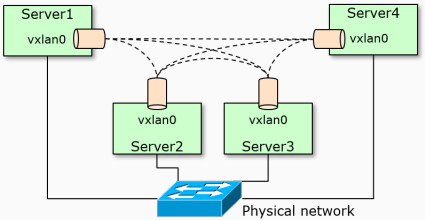


Some comments

- The frame/packet flow is "easy" to understand if you remember who you are, and the information available in that component (i.e., according to the specific context of the packet). For example:
  - When the packet is in the VM, you know only the IP routing table / ARP table of the VM
  - When the frame is in the bridge, you know only the filtering database
  - When the packet is the Linux forwarding module (networking stack), you know only the IP routing table / ARP table of the host
- The traffic may be handled in different way according to the destination
  - E.g., if the destination is another (local) VM, the packet may have to be tunneled
  - E.g., if the destination is the Internet, the packet may have to be NATted
- An orchestrator may be in charge of updating all the parameters required to scale with the number of servers (e.g., adding the required routes in the servers, creating the proper tunnels, etc.)

Tunnels and full mesh

- Point-to-point tunnels do not scale (a full mesh is required among servers)
  - This what is needed with tunneling technologies such as GRE (or Secure GRE), IPSec
- This is the reason often tunnels are created using the VxLAN technology, which provides two additional advantages
  - A transparent full-mesh solution, with a single endpoint for all the hosts belonging to the same VxLAN domain
  - VxLAN frames are encapsulated in UDP, hence the L4 port enables distributing the traffic across multiple parallel links
    - Each leaf switch has multiple connections towards the spine



Overlay vs Direct routing

Overlay

- Tunnels enables the deployment of tenant services without any interaction with the infrastructure provider
  - Useful in case of public datacenters
  - May trigger some performance problems due to the reduced MTU on the tunnel
- Preferred with OpenStack
- Also known as "*Overlay model*"

Direct routing

- Requires the collaboration of the infrastructure provider
  - Most cloud providers expose a software interface where each tenant (actually, the orchestrator such as Kubernetes) configures the required routes
  - No MTU problems
- Preferred with Kubernetes

Additional consideration for the overlay model

- Some orchestrators prefer to create the virtual network topology without touching the physical network
  - Network is just a "traditional" network, either L2 or L3
  - Network assigns IP addresses to the servers (in fact, to hypervisors) to transport traffic coming from virtual networks (which is, in fact, tunneled)
- Complete decoupling of physical and virtual networks
  - Virtual network traffic is independent from the physical topology
  - IP addressing in each virtual network is independent from the physical infrastructure
  - Traffic is moved from a software switch to another by tunneling
  - Cloud orchestrator do not need to touch (e.g., configure) the physical network; hence can be controlled by different parties (e.g., IT dept. the former, network admin the latter)
- Complete decoupling of VM orchestration (e.g., Vmware) from physical network (e.g., Cisco)

Summary

- Understanding how (virtual) networking works may be more difficult than originally thought
- Challenges
  - Need to consider multiple models: e.g., Overlay vs Direct routing
  - Need to mix virtual and real networking
  - Need to be aware of how Linux networking works
    - Not always obvious for a network engineer used to practice with hardware boxes
- The good
  - Usually networking works seamlessly
- The bad
  - When it does not work properly, debugging may be a nightmare