Software-Defined Data Centers

Brighten Godfrey
CS 538 April 11, 2018
Multi-Tenant Data Centers: The Challenges
Key Needs

Agility
Strength
Constitution
Dexterity
Charisma
Key Needs

Agility
Location independent addressing
Performance uniformity
Security
Network semantics
Agility

Agility: Use any server for any service at any time

- Better economy of scale through increased utilization
- Improved reliability

Service / tenant

- Customer renting space in a public cloud
- Application or service in a private cloud (internal customer)
Lack of Agility in Traditional DCs

Tenants in “silos”: VLAN associated with a particular IP prefix
Lack of Agility in Traditional DCs

Tenants in “silos”

Poor utilization
Lack of Agility in Traditional DCs

- Tenants in “silos”
- Poor utilization
- Inability to expand
Lack of Agility in Traditional DCs

IP addresses locked to topological location!
Key Needs

Agility

Location independent addressing
  - Tenant’s IP addresses should be portable anywhere

Performance uniformity

Security

Network semantics
Lack of Agility in Traditional DCs

Nonuniform performance

1:100 or worse oversubscription

Full line rate
Key Needs

**Agility**

Location independent addressing
- Tenant’s IP addresses can be taken anywhere

Performance uniformity
- VMs receive same throughput regardless of placement

**Security**

Network semantics
Lack of Agility in Traditional DCs

Untrusted environment
Key Needs

Agility

Location independent addressing
  • Tenant’s IP addresses can be taken anywhere

Performance uniformity
  • VMs receive same throughput regardless of placement

Security
  • Micro-segmentation: isolation at tenant or app granularity

Network semantics
Lack of Agility in Traditional DCs

x 1000s of legacy apps in a large enterprise...in a much messier topology
Key Needs

Agility

Location independent addressing
  • Tenant’s IP addresses can be taken anywhere

Performance uniformity
  • VMs receive same throughput regardless of placement

Security
  • Micro-segmentation: isolation at tenant granularity

Network semantics
  • Layer 2 service discovery, multicast, broadcast, …
Network Virtualization
Case Study: VL2
Case Study

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]

Influenced architecture of Microsoft Azure

Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

- Traffic volume between servers is 4x external traffic

Unpredictable, rapidly-changing traffic matrices (TMs)

![Graph showing traffic matrix distribution over time](https://example.com/traffic_matrix_distribution.png)

[Greenberg et al.]
Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

- Traffic volume between servers is 4x external traffic

Unpredictable, rapidly-changing traffic matrices (TMs)

[Greenberg et al.]
Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

- Traffic volume between servers is 4x external traffic

Unpredictable, rapidly-changing traffic matrices (TMs)

Design result: Nonblocking fabric

- High throughput for any TM that respects server NIC rates

[Greenberg et al.]
Motivating Environmental Characteristics

Failure characteristics

- Analyzed 300K alarm tickets, 36M error events
- 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

- “Scale out” instead of “scale up”
Cost MultiPath (ECMP), to spread the traffic and leverage the features of layer 3. Our measurements show that data centers tend to host a variety of services, and the topology is complex and heterogeneous.

In a traditional data center, the core routers carry traffic that is typically inter-rack and between servers. The devices are connected via switch-to-switch links, which are typically faster than servers.

However, the so-called "network" for virtualization or blob storage would be lost. Leveraging a trick used in many systems, routers with routing tables calculated by OSPF, thereby enabling tremendous volatility in their workload, their traffic diversity between servers. Our measurements show data center responsibilities between host and network — using a layer of software or APIs are needed.

We justify the design trade-offs by identifying the dominant design pattern of independent (e.g., random) traffic flows. The IP address changes do not impact the tier zero routers, and it is possible for the traffic to be delivered over the network without any centralized coordination or traffic工程 software.

The "network" for virtualization or blob storage would be lost. Leveraging a trick used in many systems, routers with routing tables calculated by OSPF, thereby enabling tremendous volatility in their workload, their traffic diversity between servers. Our measurements show data center responsibilities between host and network — using a layer of software or APIs are needed.

However, the so-called "network" for virtualization or blob storage would be lost. Leveraging a trick used in many systems, routers with routing tables calculated by OSPF, thereby enabling tremendous volatility in their workload, their traffic diversity between servers. Our measurements show data center responsibilities between host and network — using a layer of software or APIs are needed.

We justify the design trade-offs by identifying the dominant design pattern of independent (e.g., random) traffic flows. The IP address changes do not impact the tier zero routers, and it is possible for the traffic to be delivered over the network without any centralized coordination or traffic engineering software.

Separating names from locators: VL2 makes it easier to run legacy applications or APIs that rely on the IP address. The elements of the design enable VL2 to support key objectives, while eliminating the fragmentation and lack of semantics in the names.

VL2 makes it easier to run legacy applications or APIs that rely on the IP address. The elements of the design enable VL2 to support key objectives, while eliminating the fragmentation and lack of semantics in the names.
Routing in VL2

Unpredictable traffic

• Difficult to adapt

Design result: “Valiant Load Balancing” (at least as inspiration)

• Route traffic independent of current traffic matrix
• Spreads arbitrary traffic pattern so it’s uniform among top layer switches
Routing Implementation

Link-state network carrying only LAs (e.g., 10/8)
Embracing End Systems

As described in §4.1, we scale out the topologies to rapidly realize new functionality. For example, the VL's addressing scheme allows us to efficiently route packets to any server in the data center.

Routing Implementation

To meet the latter condition, we rely on TCP's end-to-end control plane messaging or churn. Furthermore, the routing design uses ECM link-state routing, equal-cost multipath (ECMP) forwarding, IP anycasting, and IP multicasting. VL in commodity switches: link-state routing, equal-cost multipath forwarding, IP anycasting, and IP multicasting.

We next describe each aspect of the VL's addressing scheme and routing logic. The VL's addressing scheme is designed to scale up individual network devices with more capacity using a large number of simple, independent (e.g., random) traffic patterns.

Randomizing to Cope with Volatility

Despite these techniques, we also realize n:m redundancy to improve reliability at higher layers of the hierarchy. Redundancy to n:m redundancy.

Conventional network (Figure, the counterpart of a non-blocking circuit switched network): VLB.

We use the high divergence and unpredictability of data-center traffic to our advantage. With no operational overhead, we randomize traffic patterns to cope with failure points, such as hardware bugs, and faulty components (e.g., ports).

Intermediate switches provides a richly-connected backbone well-suited for VLB. Embracing End Systems

To meet the latter condition, we rely on TCP's end-to-end control plane messaging or churn. Furthermore, the routing design uses ECM link-state routing, equal-cost multipath (ECMP) forwarding, IP anycasting, and IP multicasting. VL in commodity switches: link-state routing, equal-cost multipath forwarding, IP anycasting, and IP multicasting.

We next describe each aspect of the VL's addressing scheme and routing logic. The VL's addressing scheme is designed to scale up individual network devices with more capacity using a large number of simple, independent (e.g., random) traffic patterns.

Randomizing to Cope with Volatility

Despite these techniques, we also realize n:m redundancy to improve reliability at higher layers of the hierarchy. Redundancy to n:m redundancy.

Conventional network (Figure, the counterpart of a non-blocking circuit switched network): VLB.

We use the high divergence and unpredictability of data-center traffic to our advantage. With no operational overhead, we randomize traffic patterns to cope with failure points, such as hardware bugs, and faulty components (e.g., ports).
Routing Implementation

Link-state network carrying only LAs (e.g., 10/8)

Similar effect to ECMP to each rack

Smaller forwarding tables at most switches

Fungible pool of servers owning AAs (e.g., 20/8)
“All problems in computer science can be solved by another level of indirection.”

– David Wheeler

**App / Tenant layer**
- Application Addresses (AAs): Location independent
- Illusion of a single big Layer 2 switch connecting the app

**Virtualization layer**
- Directory server: Maintain AA to LA mapping
- Server agent: Query server, wrap AAs in outer LA header

**Physical network layer**
- Locator Addresses (LAs): Tied to topology, used to route
- Layer 3 routing via OSPF
End-to-end example

Application sends to AA 20.0.0.56
End-to-end example

Link-state network with LAs (10/8)

- Int (10.1.1.1)
- Int (10.1.1.1)
- Int (10.1.1.1)

Payload

- H(ft) 10.1.1.1
- H(ft) 10.0.0.6
- 20.0.0.55 20.0.0.56

Payload

- H(ft) 10.0.0.6
- 20.0.0.55 20.0.0.56

Payload

Host agent encapsulates

Application sends to AA 20.0.0.56

Directory servers

Q: Where is A: LA 10.0.06

AA 20.0.0.56

IP subnet with AAs (20/8)

[Greenberg et al.]
1. **Address resolution and packet forwarding**

   - **Application**: Sends packets to an AA for the destination AA addresses, on an underlying network that knows routes for other servers in the same service, while eliminating the ARP overhead that is required when servers share a single large IP subnet (i.e., the entire AA space).

   - **Routing Design**: The application sends a packet to an AA for the destination as shown in Figure 9.

   - **Packet Forwarding**: The router sends a broadcast ARP request for the destination Address Resolution Protocol (ARP) cache on the host generates a broadcast ARP request for the destination address.

   - **Access Control**: Via the directory service:
     - Requests and converts it to a unicast query to the VL network.
     - DLLA counseling table.
     - ARP cache.

2. **End-to-end example**

   - **Path**: From a source host, ECMP takes care of delivering packets encapsulated with multicast addresses, each associated with only as many Intermediate switches.

   - **ECMP vs. Anycast**: First, switches today only support up to 4096 paths. Second, they reduce overhead in the network control plane.

   - **Intermediate Switch**: Upon switch or link failures, ECMP will react, eliminating any ECMP being released by some vendors this year.

   - **Alternate Path**: If there are more paths available than ECMP can use, then VL can be used.

   - **Avoids Out-of-Order Delivery**: TCP detects a severe congestion event (e.g., a full window lost), TCP retransmits segments, but randomly chooses one of the equal-cost paths. Should it occur, initial results show the VL policy can enforce access-control policies. Further testing is required.

   - **Links between Servers**: By using VLB, ECMP reduces the number of links that need to be updated when the location of a server changes due to virtual-machine migration.

   - **Network Control Plane**: First, all Intermediate switches, decapsulated by the switch, delivered to the T

   - **Limited Availability Changes**: Due to switch/link failures, ECMP will react, eliminating any ECMP being released by some vendors this year.

   - **ECMP Release**: If there are more paths available than ECMP can use, then VL can be used.

   - **Avoids Out-of-Order Delivery**: TCP detects a severe congestion event (e.g., a full window lost), TCP retransmits segments, but randomly chooses one of the equal-cost paths. Should it occur, initial results show the VL policy can enforce access-control policies. Further testing is required.

   - **Limited Availability Changes**: Due to switch/link failures, ECMP will react, eliminating any ECMP being released by some vendors this year.

   - **Avoids Out-of-Order Delivery**: TCP detects a severe congestion event (e.g., a full window lost), TCP retransmits segments, but randomly chooses one of the equal-cost paths. Should it occur, initial results show the VL policy can enforce access-control policies. Further testing is required.
End-to-end example

Host agent encapsulates packets to an AA if the directory service refuses to provide information. Subsequent communication need not entail a directory lookup and converts it to a unicast query to the VL stack on the host, which generates a broadcast ARP request for the destination AA and delivers it to the destination as shown in Figure 4.2.3.

Address resolution:

Packet forwarding:

As servers can annotate endpoints with VLAs, each associated with only as many Intermediate switches as ECMP can accommodate. When an Intermediate switch determines that it has more paths available than ECMP can use, then VL forwards the anycast addresses from that switch to agents upon failures. Intermediate switch decapsulates.

Address resolution:

Packet forwarding:

To route traffic, a VL network, a VL instance of the directory service, and directo...
End-to-end example

Directory servers

Host agent encapsulates

Intermediate switch decapsulates

Destination ToR decapsulates again

Application sends to AA 20.0.0.56

[Greenberg et al.]
4.2.1 Address resolution and packet forwarding

**Address resolution:**

An application sends a packet to an AA for the first time. Since the AA does not know the destination address, it needs to resolve it. This process involves querying a directory service, such as DNS or a local cache, to look up the IP address associated with the destination.

**Packet forwarding:**

Once the destination address is resolved, the packet is forwarded to the next hop, which is typically a router or a switch. The packet is encapsulated with the destination AA's address, and the process repeats until the packet reaches its destination.

---

**End-to-end example**

The diagram illustrates how a packet is sent from an application to a destination AA, passing through intermediate switches. Each switch decapsulates the packet, looks up the destination AA, and encapsulates the packet with the next hop's address. This process continues until the packet reaches the destination AA.

**Key steps:**

1. Application sends a packet to AA 20.0.0.56.
2. Intermediate switch decapsulates the packet.
3. Intermediate switch looks up the destination AA and encapsulates the packet with the next hop's address.
4. Destination ToR switch decapsulates the packet again.
5. Host agent delivers the packet to AA 20.0.0.56.

---

For more details on the specific steps and processes involved, refer to the referenced paper by Greenberg et al.
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 decent (but far from perfect) load balance
- But, performance isolation among tenants depends on TCP backing off to rate destination can receive
- Leaves open the possibility of better 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

Host agents: dynamic “programming” of data path
VL2 Enduring Take-Aways

Scale-out nonblocking Clos network
ECMP for traffic-oblivious routing
Separation of virtual and physical addresses
Centralized control plane
Network Virtualization
Case Study: NVP
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

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

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen
NVP Approach to Virtualization

1. Service: Arbitrary network topology
NVP Approach to Virtualization

1. Service: Arbitrary network topology
NVP Approach to Virtualization

Service: Arbitrary network topology

Network Hypervisor

Physical Network: Any standard layer 3 network
Virtual network service

[Figure: Koponen et al.]
Virtual network service

[Figure: Koponen et al.]
Virtual network service

- VM
- Map
- ACL
- L2
- ACL
- Map
- ACL
- L2
- L3
- Logical Datapath 1
- Logical Datapath 2

Diagram showing the flow of packet processing through two logical datapaths interconnected by a logical router.
Virtual network service

There is much previous work on the problem of packet classification without TCAMs. See for instance [26, 27] for reasons discussed shortly. NVP also supports other tunnel types, such as GRE [8] for reasons discussed shortly.

Forwarding Performance

NVP uses an encapsulation method called STT [37, 38]. As a NIC processes an STT packet, among other things, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

To achieve efficient flow lookups on x86, OVS exploits different technique to classify packets quickly. Traditional physical switches generally classify packets against wildcard flows irrelevant parts of a packet header. OVS consists of a kernel module and a userspace program; the kernel module sends the first packet of each connection) will traverse exactly the same set of flow entries. In this case, the OVS flow table would hold full flow table, including wildcards, as many times as the presence of any IP encapsulation in the packet. That is, of flow classification on x86, NVP's encapsulation of all traffic can introduce significant overhead. This overhead

While exact-match kernel flows alleviate the challenges of large receive offload for encapsulated traffic with existing NICs, current Ethernet NICs do not support offloading in the pipeline submits the packet to the first table of the logical switch (with ACLs) to a logical router before being forwarded by a logical switch attached to the edge. This restriction means that the OVS flow table runs at the receiving hypervisor and there is a tunnel failure, multicast/broadcast, ARP, and QoS, for instance.

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

Each logical datapath: instead of encapsulating the packet and sending it over a tunnel, the final action of a logical datapath is mapped to tunnel headers. After each logical datapath, the logical forwarding decision is mapped to the next logical hop. The last logical decision is mapped to tunnel headers. Destination address, leaving the last logical L2 hop to be executed at the receiving hypervisor. Thus, in Figure 4: Processing steps of a packet traversing through two logical switches interconnected by a logical router (in the middle). Physical flows

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, the logical destination of the packet. That is, e.g., TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system's inability to enable standard NIC mechanisms exist for UDP traffic; the generalization of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar in CPU usage for high-volume TCP transfers. Similar
Virtual network service

There is much previous work on the problem of packet classification without TCAMs. See for instance [26] for reasons discussed shortly. OVS consists of a kernel module and a userspace program; the kernel module sends the first packet of each flow entries. OVS uses TCAMs, which are not available on the standard switches generally classify packets against wildcard flows irrelevant parts of a packet header. Traditional physical flow entries written by NVP can contain wildcards for any packet against its entire flow table in software. However, this restriction means that the OVS flow table cannot store entire flow entries for all interconnected logical datapaths. In this case, the OVS flow table would hold new flow into userspace, where it is matched against the flow table of the sending hypervisor needs only to have flows for its local VMs. To achieve efficient flow lookups on x86, OVS exploits the fact that all of the packets belonging to a single flow of traffic (or GSO) and handed over a large frame to the virtual NIC, MSS-sized packets and computes the TCP checksums for each packet on behalf of the OS. Large Receive Offload (LRO) allows hardware offloading mechanisms for encapsulated traffic. NVP also supports other tunnel types, such as GRE [9].

Figure 4: Processing steps of a packet traversing through two logical switches interconnected by a logical router (in the middle). Physical flows originating at a source VM first traverses through a logical switch (with ACLs) to a logical router before being forwarded by a logical switch attached to the destination VM (on the other side of the tunnel). This being forwarded by a logical switch attached to the destination address, leaving the last logical L2 hop to be executed at the receiving hypervisor. Thus, in Figure 4, processing steps of a packet traversing through two logical switches interconnected by a logical router are depicted how a packet is mapped from physical to logical, and vice versa. The packets pass through the virtual switch of the underlying hypervisor would have been able to see into the encapsulated packet. However, the operating system’s inability to enable standard NIC tunnel header insertion, but to pass them to the NIC; today’s NICs are simply not capable of seeing into the encapsulated packet. That is, NVP uses an encapsulation method called STT [8]. TSF is called Generic Segmentation Offload (GSO). Current Ethernet NICs do not support offloading in the presence of any IP encapsulation in the packet. That is, NVP places a standard, but fake, TCP header after the physical header including contextual information that specifies, among other things, the logical destination of the packet. As a NIC processes an STT packet, it places its contextual information into the encapsulated packet and compute their checksums before encapsulating them and passing them to the NIC; today’s NICs are simply not capable of seeing into the encapsulated packet. However, this restriction means that the OVS flow table cannot process packets in CPU usage for high-volume TCP transfers. Similar to achieve efficient flow lookups on x86, NVP’s encapsulation of all of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Similar TCP Segmentation Offload (TSO) allows to encapsulate an entire TCP packet, including its header, into a separate network layer packet. TSO is called Generic Segmentation Offload (GSO).
Virtual network service

There is much previous work on the problem of packet classification without TCAMs. See for instance [37] for reasons discussed shortly.

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, among other things, the logical destination of the packet.

TCP Segmentation Offload (TSO) allows packets into a single large TCP packet and, after verifying the operating system to send TCP packets larger than the physical MTU to a NIC, which then splits them into hardware offloading mechanisms for encapsulated traffic.

TSO is called Generic Segmentation Offload (GSO). NVP also supports other tunnel types, such as GRE [26] for reasons discussed shortly.

OVS, as a virtual switch, must classify each incoming flow (L2-L4 headers). Future packets in this same flow can then be matched entirely by the kernel. Existing work considers flow caching in more detail [22].

There are two standard offload mechanisms relevant to virtual network service: hardware offloading for encapsulated traffic with existing NICs, and passing them to the NIC; today's NICs are simply not designed to support these protocols.

The actual logical packet (starting with its Ethernet header) follows. As a NIC processes an STT packet, among other things, the logical destination of the packet.

As an optimization, we constrain the logical topology so as to avoid unnecessary computation on the hypervisor: instead of encapsulating the packet and then traversing each logical datapath, the packet would traverse each logical datapath by the kernel, which contain a match for every part of the packet against its entire flow table in software. However, this approach is expensive in CPU usage for high-volume TCP transfers. Similar to GSO, TSO minimizes CPU usage by offloading the NIC to build the TCP packet with its physical MTU (LRO) does the opposite and collects multiple incoming places a standard, but fake, TCP header after the physical header and compute their checksums before encapsulating them and passing them to the NIC.
Virtual network service

There is much previous work on the problem of packet classification without TCAMs. See for instance USENIX Symposium on Networked Systems Design and Implementation [11th USENIX Symposium on Networked Systems Design and Implementation].
Virtual network service

After each logical datapath, the logical forwarding decision is mapped to the next logical hop. The last logical decision is mapped to tunnel headers.

The receiving hypervisor is determined by the logical IP destination address, leaving the last logical L2 hop to be executed at the receiving hypervisor. Thus, in Figure 4, the last logical decision is mapped to the tunnel header.

There is much previous work on the problem of packet classification. We have found little value in supporting logical routers (or GSO) and handing over large frames to the virtual NIC, as an optimization, we constrain the logical topology to ensure that logical L2 destinations can only be present at the edge.

Failover, multicast/broadcast, ARP, and QoS, for instance, are not supported on logical L2 hops. For example, the sending hypervisor does not host any VMs attached to the last logical L2 hop. To achieve efficient flow lookups on x86, OVS exploits hardware mechanisms that are relevant in this setting. Physical traffic locality provides a significant reduction in the number of flow entries written by NVP can contain wildcards for any irrelevant parts of a packet header. Traditional physical flow entries written by NVP can contain wildcards for any irrelevant parts of a packet header. Traditional physical flow entries written by NVP can contain wildcards for any irrelevant parts of a packet header.

To overcome this limitation and re-enable hardware mechanisms exists for UDP traffic; the generalization of these mechanisms provides a significant reduction in the number of flow entries written.

Figure 4: Processing steps of a packet traversing through two logical switches interconnected by a logical router (in the middle). Physical flows are simplified example: we omit the steps required for failover, multicast/broadcast, ARP, and QoS, for instance. We consider a single packet originating at a source VM first traverses through a logical switch (with ACLs) to a logical router before the packet would traverse each logical datapath by the pipeline submitting the packet to the first table of the new flow into userspace, where it is matched against the full flow table, including wildcards, as many times as the kernel, which contains a match for every part of the packet on behalf of the OS. Large Receive Offload (LRO) does the opposite and collects multiple incoming places a standard, but fake, TCP header after the physical header including contextual information that specifies, among other things, the logical destination of the packet.

To break up the packets into standard MTU-sized packets and compute their checksums before encapsulating them and compute their checksums before encapsulating them and passing them to the NIC; today’s NICs are simply not capable of seeing into the encapsulated packet. Even if a VM’s operating system would have enabled TSO or GSO and handed over a large frame to the virtual NIC, the virtual switch of the underlying hypervisor would have the operating system’s inability to enable standard NIC mechanisms for UDP traffic; the generalization of these mechanisms provides a significant reduction in the number of flow entries written.

VXLAN [37, 38] is an example of a tunneling protocol that can be used to interconnect virtual networks across physical networks. VXLAN encapsulates L2 packets within an L4 header, allowing virtual networks to be extended across physical networks without modifying the underlying physical infrastructure. VXLAN is typically used in conjunction with software-defined networking (SDN) to enable the creation of virtual networks that can be easily configured and managed. VXLAN allows for the creation of virtual networks that can be easily configured and managed.
Virtual network service

There is much previous work on the problem of packet flow classification. We have found little value in supporting logical routers as a generalization of traditional physical routers. As an optimization, we constrain the logical topology of a tenant's virtual switched network to be single-rooted if possible. This restriction means that the OVS flow table of a sending hypervisor needs only to have flows for its edge. The receiving hypervisor is determined by the logical IP destination address, leaving the last logical L2 hop to be executed at the receiving hypervisor. Thus, in Figure 4, if a tenant's TCP connection (or GSO) and handed over a large frame to the virtual NIC, the NIC would fragment the packet and hand it over to the kernel, which contain a match for every part of the packet would traverse each logical datapath by the same principles as it traverses the pipeline of a single logical switch (with ACLs) to a logical router before being forwarded by a logical switch attached to the VM.

3.2

NVP also supports other tunnel types, such as VXLAN [11] and GRE [26] for reasons discussed shortly. The actual logical packet (starting with its Ethernet header including contextual information that specifies, among other things, the logical destination of the packet.) is then processed by the logical switch, which contain the L3/L4 headers. Future packets in this same flow (L2-L4 headers) will traverse exactly the same set of flow entries. OVS consists of a kernel module and a userspace program; the kernel module sends the first packet of each connection to the userspace program installs new flow into userspace, where it is matched against the flow table in software. However, OVS, as a virtual switch, must classify each incoming packet against its entire flow table in software. However, different techniques to classify packets quickly.

3.3

Traffic locality

switches generally classify packets against wildcard flows of the L3 routers of the logical topology; as well as those of the L3 routers of the logical topology; after each logical datapath, the logical forwarding decision is mapped to the next logical hop. The last logical decision is mapped to tunnel headers.
Virtual network service

---

To achieve efficient flow lookups on x86, OVS exploits

---

...as those of the L3 routers of the logical topology;

---

After each logical datapath, the logical forwarding decision is mapped to the next logical hop. The last logical decision is mapped to tunnel headers.

---

...x86 hardware where OVS runs, and so OVS must use a

---

...USENIX Association

---

...the receiving hypervisor is determined by the logical IP

---

...interconnected through logical switches without tenant VMs.

---

...program; the kernel module sends the first packet of each

---

...connections) will traverse exactly the same set of flow

---

...traffic locality

---

...different technique to classify packets quickly.

---

...using TCAMs, which are not available on the standard

---

...switches generally classify packets against wildcard flows

---

...flow entries written by NVP can contain wildcards for any

---

...packet against its entire flow table in software. However,

---

...OVS, as a virtual switch, must classify each incoming

---

...3.2

---

...between the second and third logical datapaths instead.

---

...runs at the receiving hypervisor and there is a tunnel

---

...the third logical datapath, then the third logical datapath

---

...executed at the receiving hypervisor. Thus, in Figure

---

...destination address, leaving the last logical L2 hop to be

---

...of a sending hypervisor needs only to have flows for

---

...such that logical L2 destinations can only be present at

---

...failover, multicast/broadcast, ARP, and QoS, for instance.

---

...being forwarded by a logical switch attached to the

---

...originating at a source VM first traverses through a

---

...pipeline submits the packet to the first table of the

---

...forwarding Performance

---

...To overcome this limitation and re-enable hardware

---

...offloading for encapsulated traffic with existing NICs,

---

...there are two standard offload mechanisms relevant to

---

...While exact-match kernel flows alleviate the challenges

---

...of flow classification on x86, NVP's encapsulation of all

---

...consider flow caching in more detail [26].

---

...full flow table, including wildcards, as many times as the

---

...program installs a new flow into userspace, where it is matched against the

---

...Virtual network service
Control abstraction
(sequence of OpenFlow flow tables)

Virtual network service

Packet abstraction
Tenant control abstraction

Network Hypervisor

Controllers

Open vSwitch

Tenant VM

server

Open vSwitch

Tenant VM

server

Open vSwitch

Tenant VM

server

Physical L3 Network

tunnel (GRE, VXLAN)
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 interferes with TCP Segmentation Offload (TSO)

- NIC can’t see TCP outer header
- Solution: STT tunnels adds “fake” outer TCP header
Discussion

Where’s the SDN?

- API to data plane
- centralized controller
- control abstractions

Why was micro-segmentation a “killer app” for SDN?

- Needed to automate control of a dynamic, virtualized environment, not suited to manual solutions

How does it compare to wide-area control in B4?
Industry Impact

Multiple vendors with software-defined data center “micro-segmentation” products

- VMware’s NSX
  - VMware claims more than 2,400 customers, $1B/yr sales
- Cisco’s ACI
- Startups vArmour, Illumio

Next time

- Programmable switches