Data Center Virtualization: VirtualWire

Hakim Weatherspoon

Assistant Professor, Dept of Computer Science

CS 5413: High Performance Systems and Networking

November 21, 2014

Slides from USENIX Workshop on Hot Topics in Cloud Computing (HotCloud) 2014 presentation and Dan Williams dissertation

Where are we in the semester?

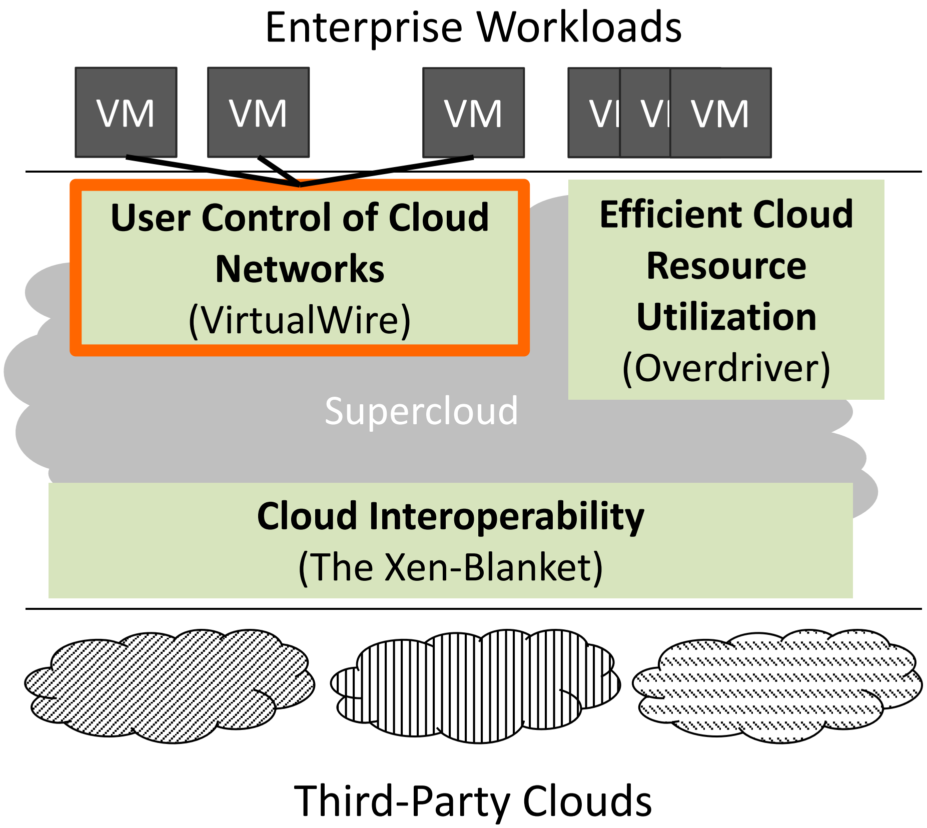
- Overview and Basics
- Data Center Networks
  - Basic switching technologies
  - Data Center Network Topologies (today and Monday)
  - Software Routers (eg. Click, Routebricks, NetMap, Netslice)
  - Alternative Switching Technologies
  - Data Center Transport
- Data Center Software Networking
  - Software Defined networking (overview, control plane, data plane, NetFGPA)
  - Data Center Traffic and Measurements
  - Virtualizing Networks
  - Middleboxes
- Advanced Topics

Goals for Today

- VirtualWires for Live Migrating Virtual Networks across Clouds
  - D. Williams, H. Jamjoom, Z. Jiang, and H. Weatherspoon. *IBM Tech. Rep. RC25378*, April 2013.

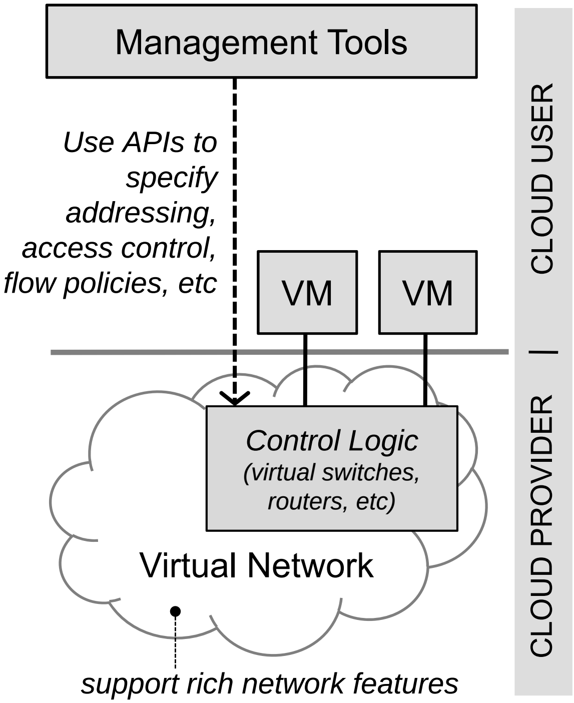
Control of cloud networks

- Cloud interoperability
- User control of cloud networks



current clouds lack control over network

- Cloud networks are provider-centric
  - Control logic that encodes flow policies is implemented by provider
  - Provider decides if low-level network features (e.g., VLANs, IP addresses, etc.) are supported


Management Tools
Use APIs to specify addressing, access control, flow policies, etc
VM
VM
Control Logic
(virtual switches, routers, etc)
Virtual Network
support rich network features
CLOUD PROVIDER | CLOUD USER

What virtual network abstraction should a cloud provider expose?

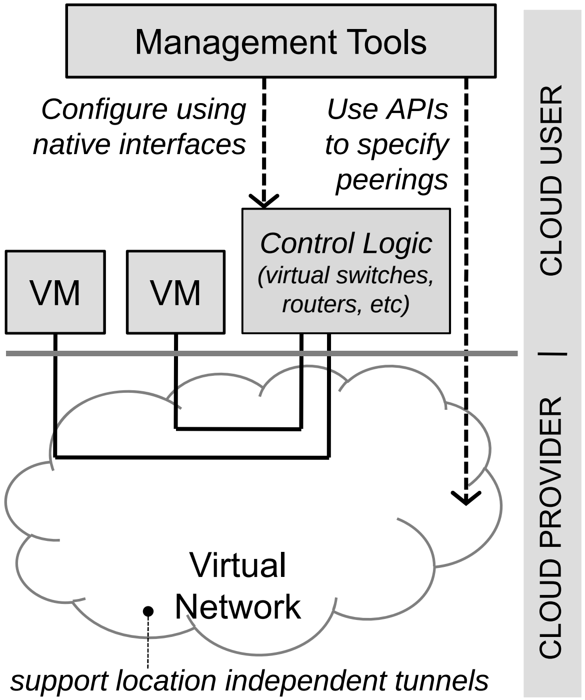
virtualwire

- Key Insight: move control logic to user

- Virtualized equivalents of network components
  - Open vswitch, Cisco Nexus 1000V, NetSim, Click router, etc.

- Provider just needs to enable connectivity
  - Connect/disconnect

- VirtualWire connectors
  - Point-to-point layer-2 tunnels

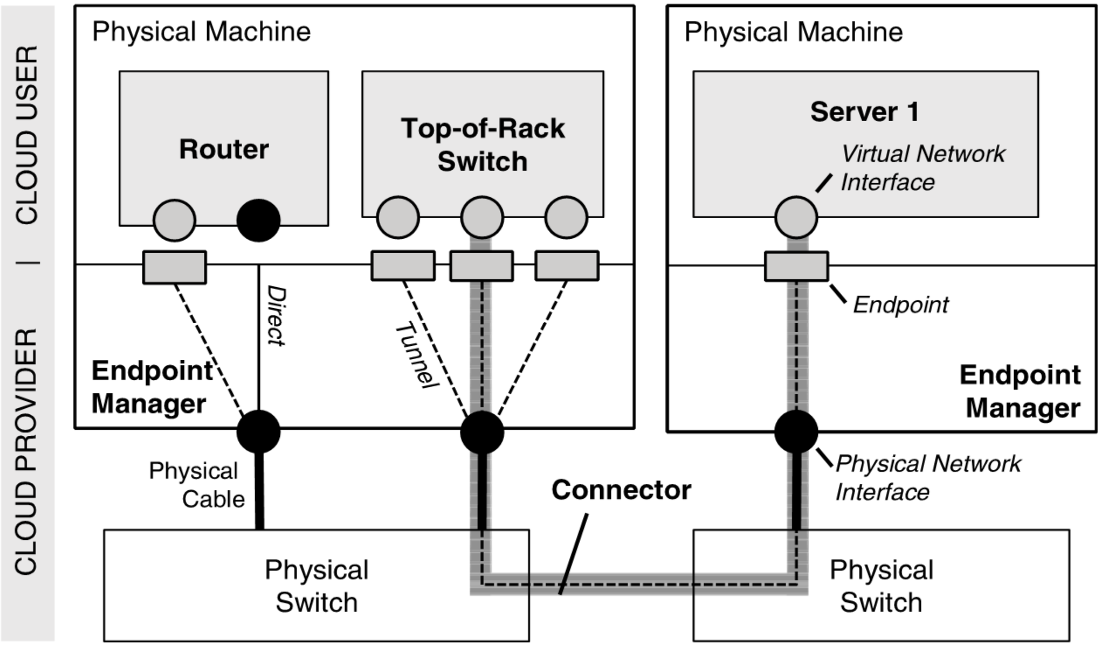

Management Tools
Configure using native interfaces
Use APIs to specify peerings
Control Logic (virtual switches, routers, etc)
VM
VM
Virtual Network
support location independent tunnels
CLOUD USER | CLOUD PROVIDER

Outline

- Motivation
- VirtualWire
  - Design
  - Implementation
- Evaluation
- Conclusion

VirtualWire connectors / wires

- Point-to-point layer-2 network tunnels
  - VXLAN wire format for packet encapsulation
- Endpoints migrated with virtual network components
- Implemented in the kernel for efficiency



VirtualWire connectors / wires

- Connections between endpoints
  - E.g. tunnel, VPN, local bridge

- Each hypervisor contains endpoint controller
  - Advertises endpoints
  - Looks up endpoints
  - Sets wire type
  - Integrates with VM migration

- Simple interface
  - **connect/disconnect**

VirtualWire connectors / wires

- Types of wires
  - Native (bridge)
  - Encapsulating (in kernel module)
  - Tunneling (Open-VPN based)

- **/proc** interface for configuring wires

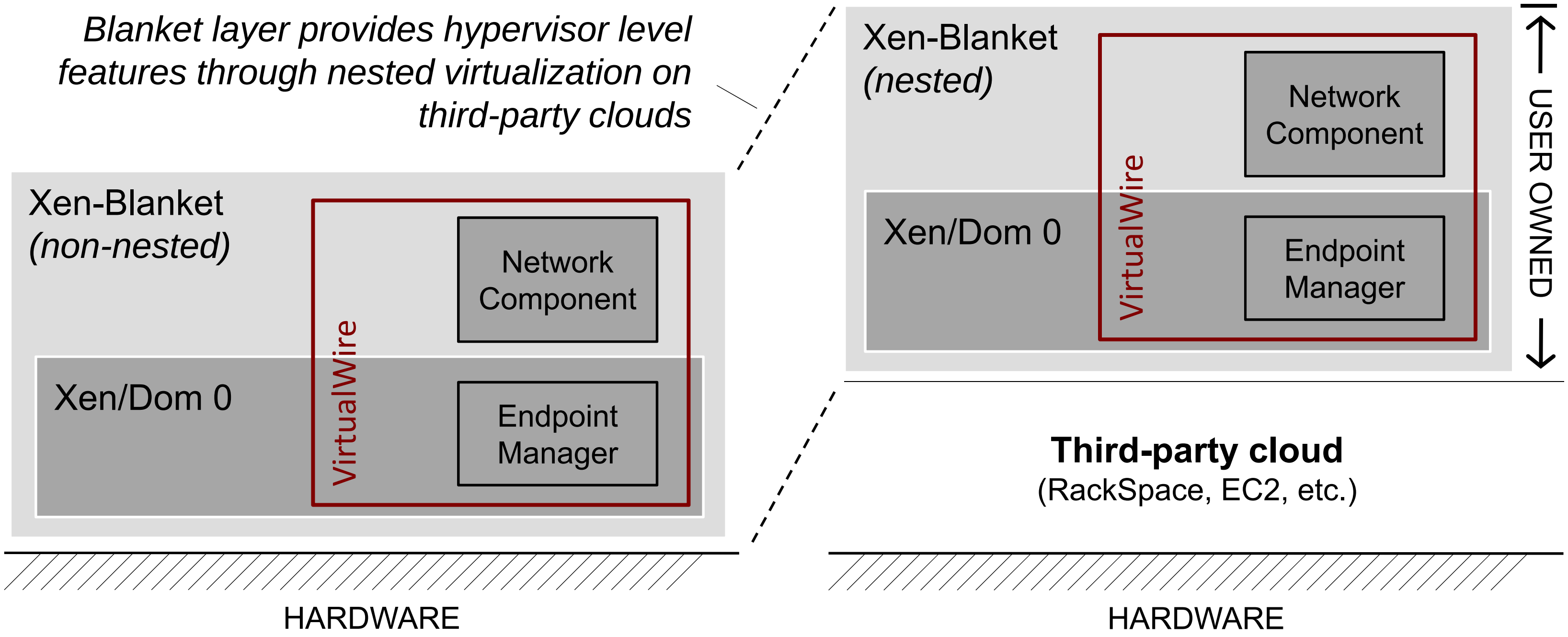
- Integrated with live migration

Connector Implementation

- Connectors are layer-2-in-layer-3 tunnels
  - 44 byte UDP header includes 32-bit connector ID

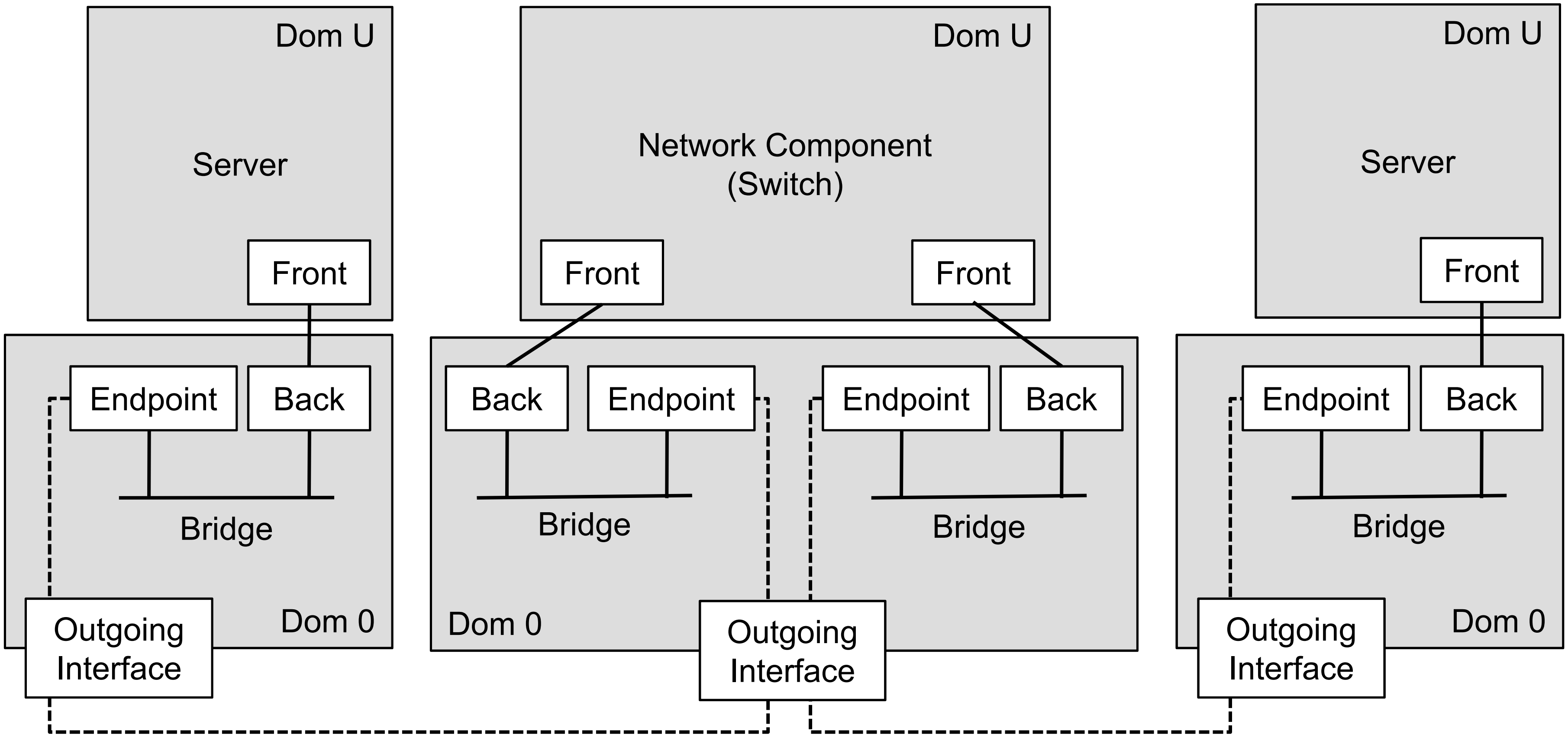
<table>
<tr><td></td><td colspan="5">Outer Ethernet
Header</td></tr>
<tr><td rowspan="5">Outer IP</td><td>Version</td><td>IHL</td><td>TOS</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Outer Source Address</td></tr>
<tr><td colspan="5">Outer Destination Address</td></tr>
<tr><td rowspan="2">Outer UDP</td><td colspan="3">Source Port</td><td colspan="2">Dest Port</td></tr>
<tr><td colspan="3">UDP Length</td><td colspan="2">UDP Checksum</td></tr>
<tr><td></td><td colspan="5">VirtualWire Connector ID</td></tr>
<tr><td rowspan="4">Inner Ethernet</td><td colspan="5">Inner Destination MAC Address</td></tr>
<tr><td colspan="3">Inner Destination MAC Address</td><td colspan="2">Inner Source MAC Address</td></tr>
<tr><td colspan="5">Inner Source MAC Address</td></tr>
<tr><td colspan="3">Optional Ethertype = C-Tag [802.1Q]</td><td colspan="2">Inner.VLAN Tag Information</td></tr>
<tr><td></td><td colspan="5">Original Ethernet Payload</td></tr>
</table>

virtualwire and the xen-blanket

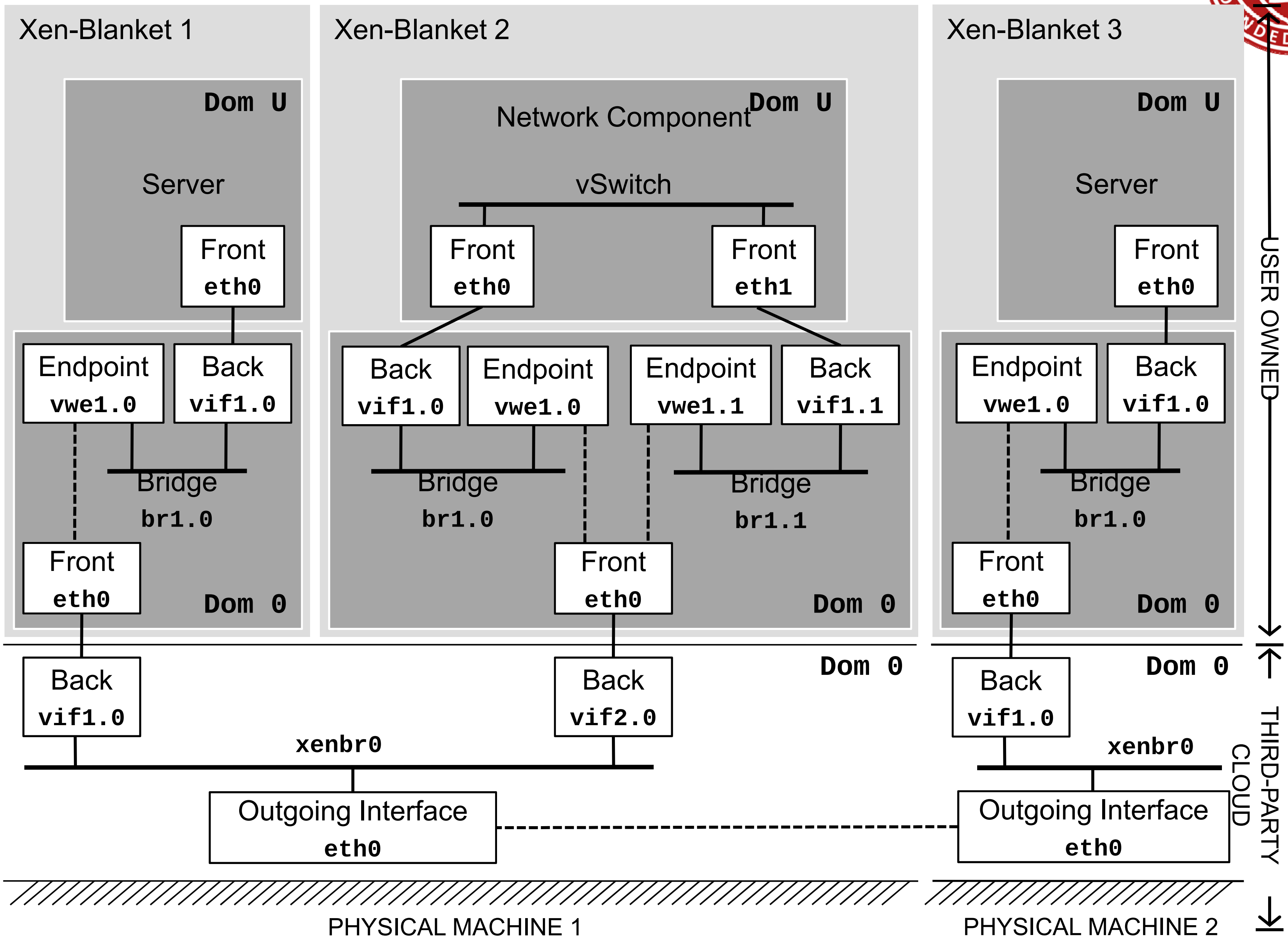


- Enables cross-provider live migration

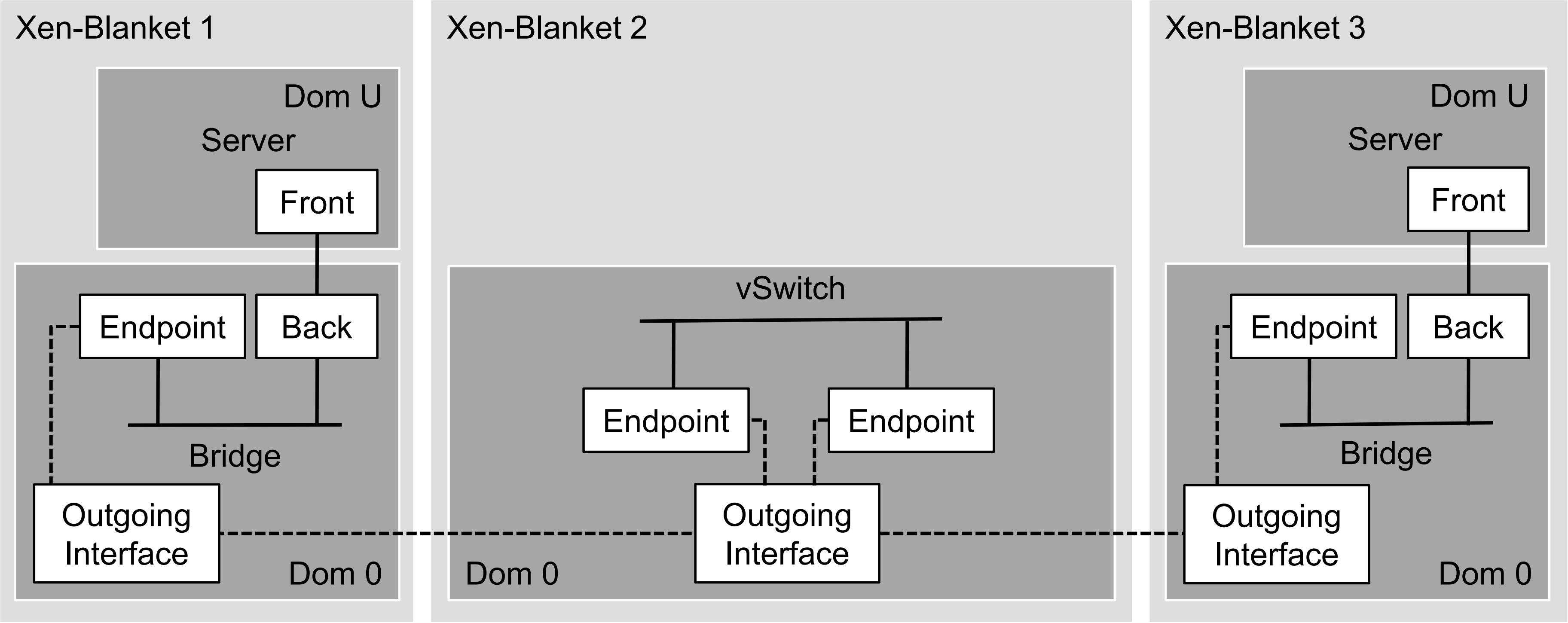
Implementation



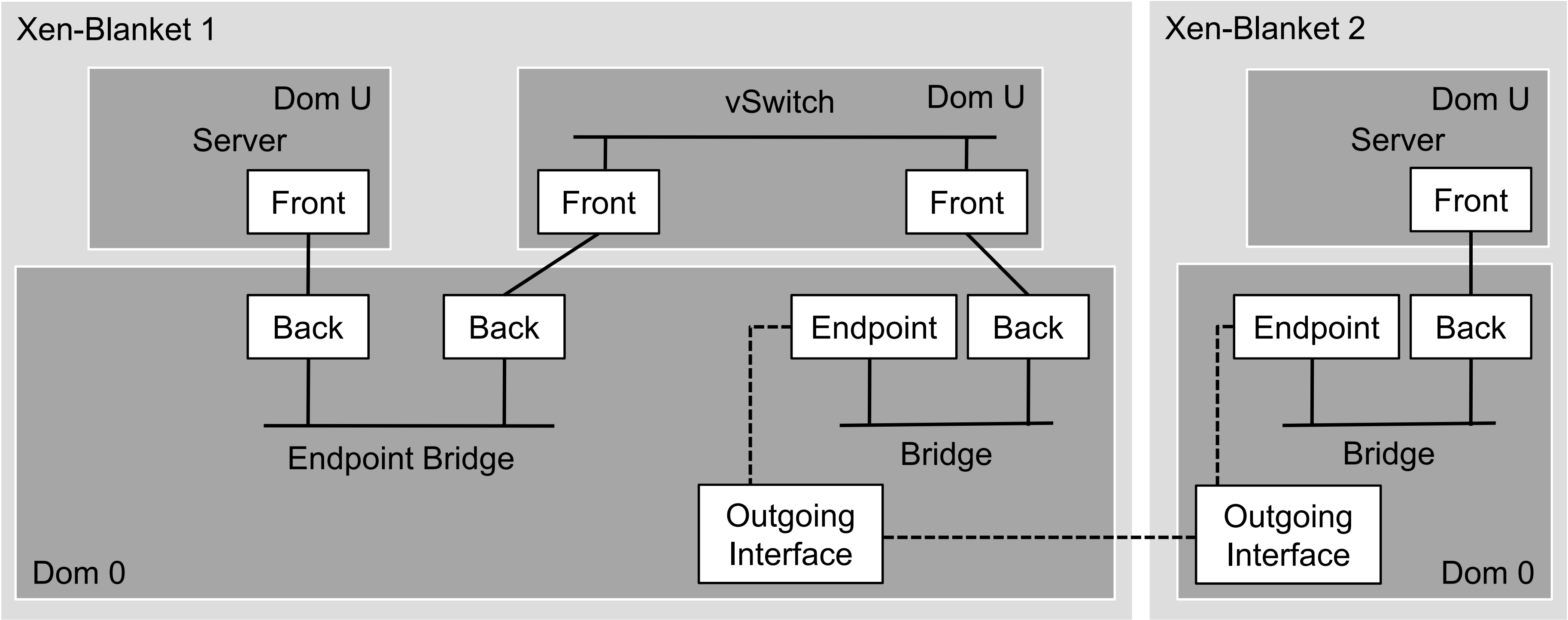
Implementation



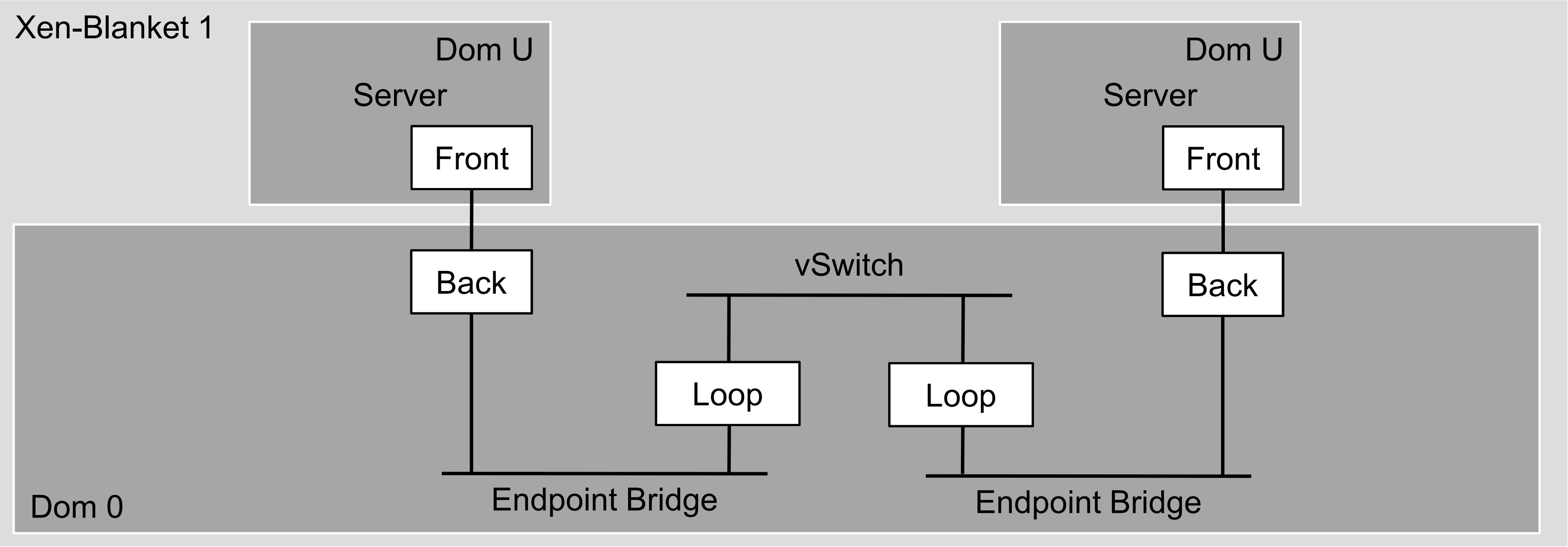
Optimizations



Optimizations



Optimizations

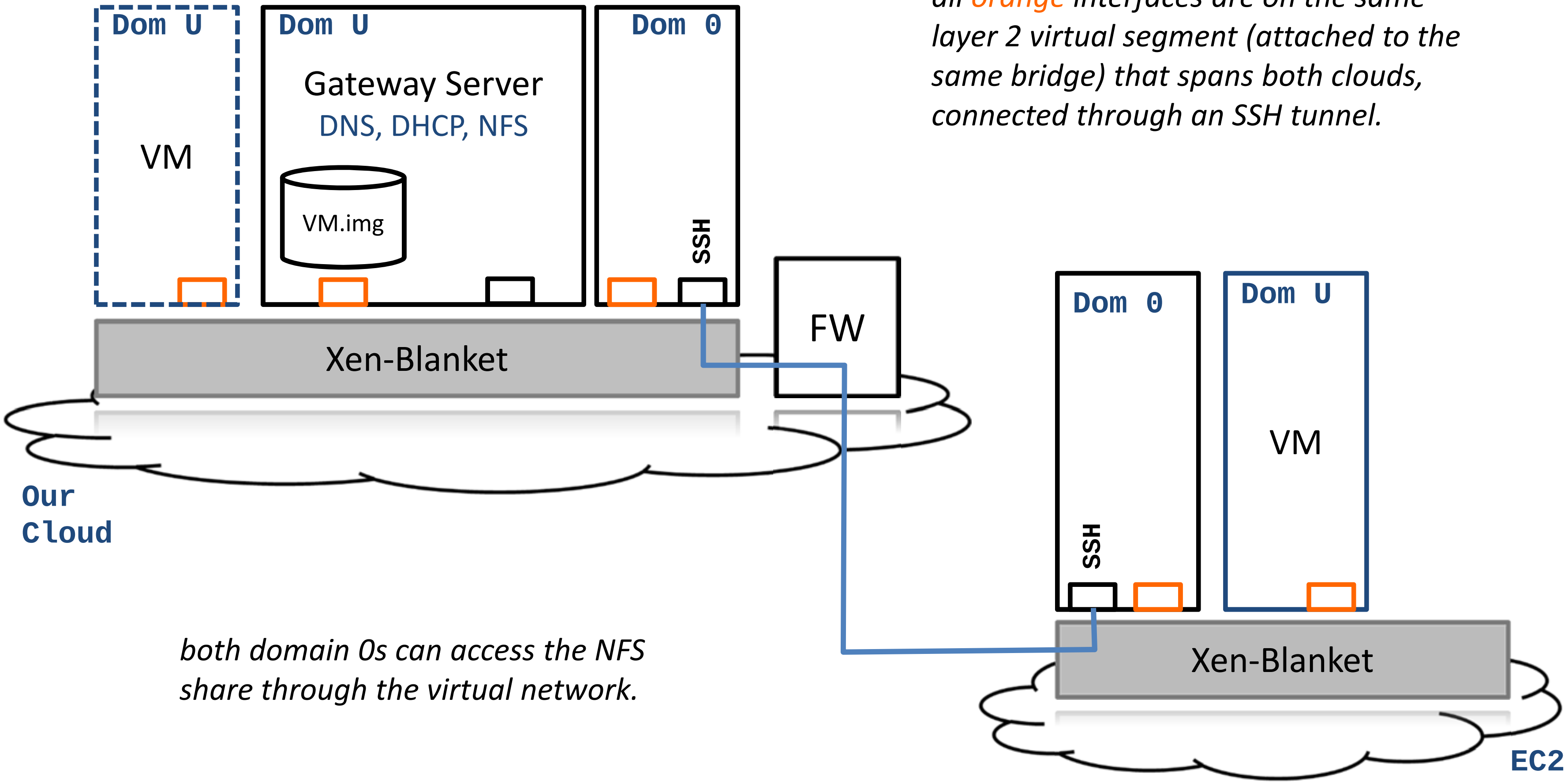


Outline

- Motivation
- VirtualWire
  - Design
  - Implementation
- Evaluation
- Conclusion

cross provider live migration

all orange interfaces are on the same layer 2 virtual segment (attached to the same bridge) that spans both clouds, connected through an SSH tunnel.

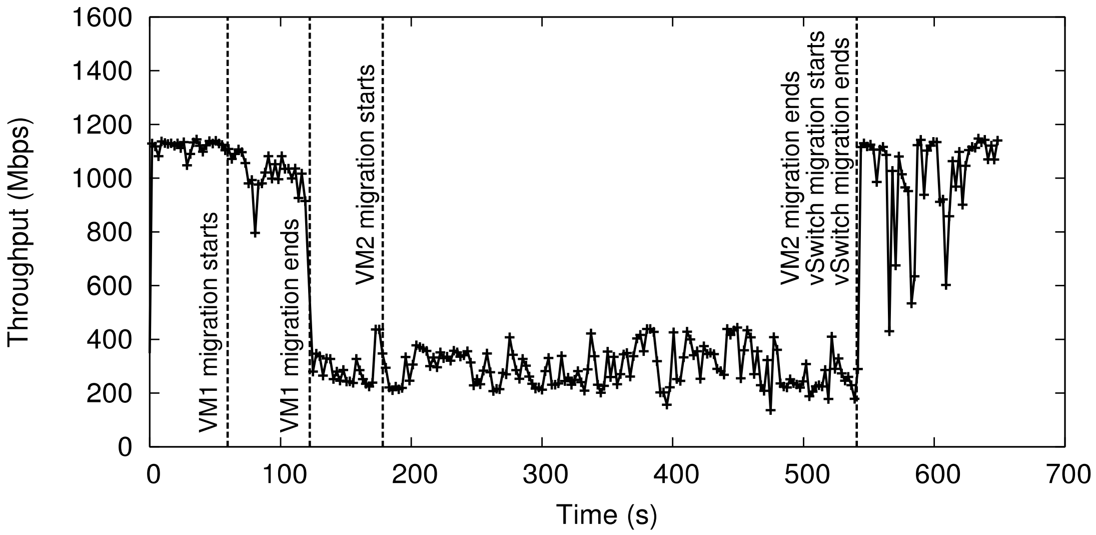


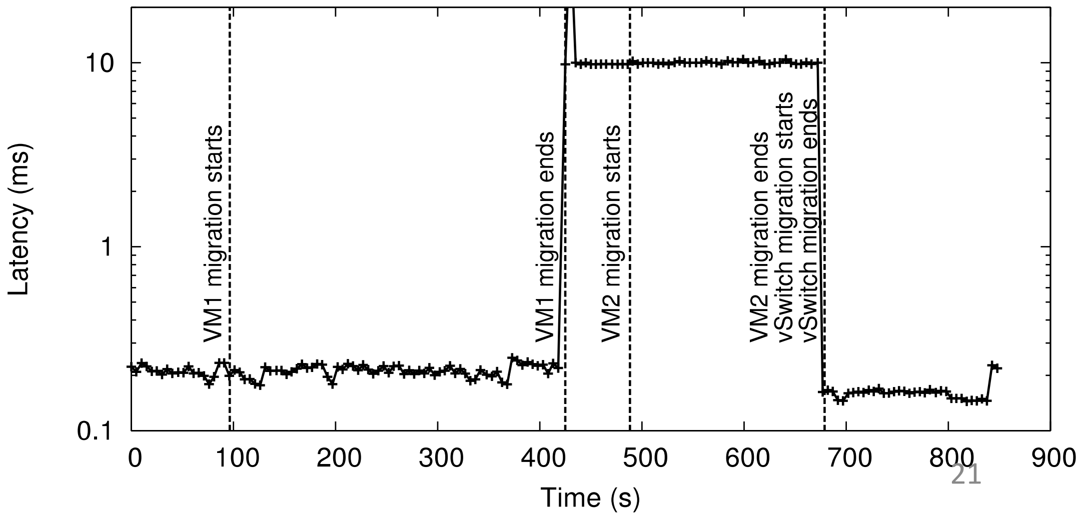
both domain 0s can access the NFS share through the virtual network.

- Amazon EC2 and local resources
  - EC2 (4XL): 33 ECUs, 23 GB memory, 10 Gbps Ethernet
  - Local: 12 cores @ 2.93 GHz, 24 GB memory, 1Gbps Ethernet
- Xen-blanket for nested virtualization
  - Dom 0: 8 vCPUs, 4 GB memory
  - PV guests: 4 vCPUs, 8 GB memory
- Local NFS server for VM disk images
- **netperf** to measure throughput latency
  - 1400 byte packets

cross-provider live migration

- Migrated 2 VMs and a virtual switch between Cornell and EC2
- No network reconfiguration
- Downtime as low as 1.4 seconds





Outline

- Motivation
- VirtualWire
  - Design
  - Implementation
- Evaluation
- Conclusion

performance issues

- Virtual network components can be bottlenecks
  - physical interface limitations
- Several approaches
  - Co-location
  - Distributed components
  - Evolve virtual network

Before Next time

- Project Interim report
  - **Due Monday, November 24.**
  - And meet with groups, TA, and professor
- Fractus Upgrade: Should be back online

- ***Required review and reading for Monday, November 24***
  - Making Middleboxes Someone Else’s Problem: Network Processing as a Cloud Service, Making middleboxes someone else's problem: network processing as a cloud service, J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. ACM SIGCOMM Computer Communication Review (CCR) Volume 42, Issue 4 (August 2012), pages 13-24.
  - http://dl.acm.org/citation.cfm?id=2377680
  - http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p13.pdf

- Check piazza: http://piazza.com/cornell/fall2014/cs5413
- Check website for updated schedule

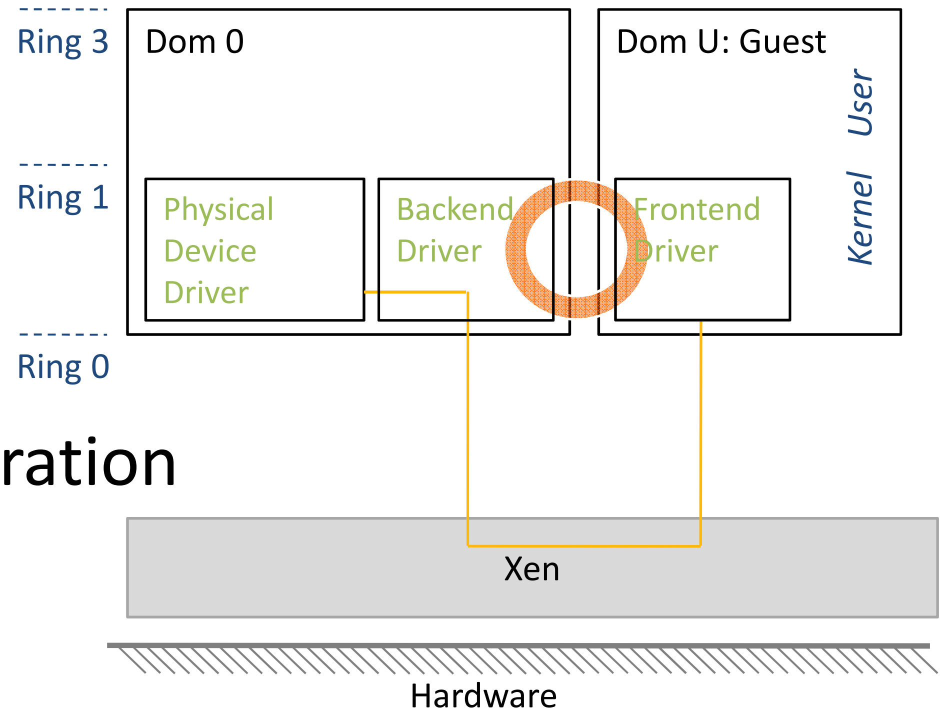
Decoupling gives Flexibility

- Cloud’s flexibility comes from decoupling device functionality from physical devices
  - Aka ***virtualization***
- Can place VM anywhere
  - Consolidation
  - Instantiation
  - Migration
  - Placement Optimizations

Are all Devices Decoupled

- Today: Split driver model
  - Guests don’t need device specific driver
  - System portion interfaces with physical devices

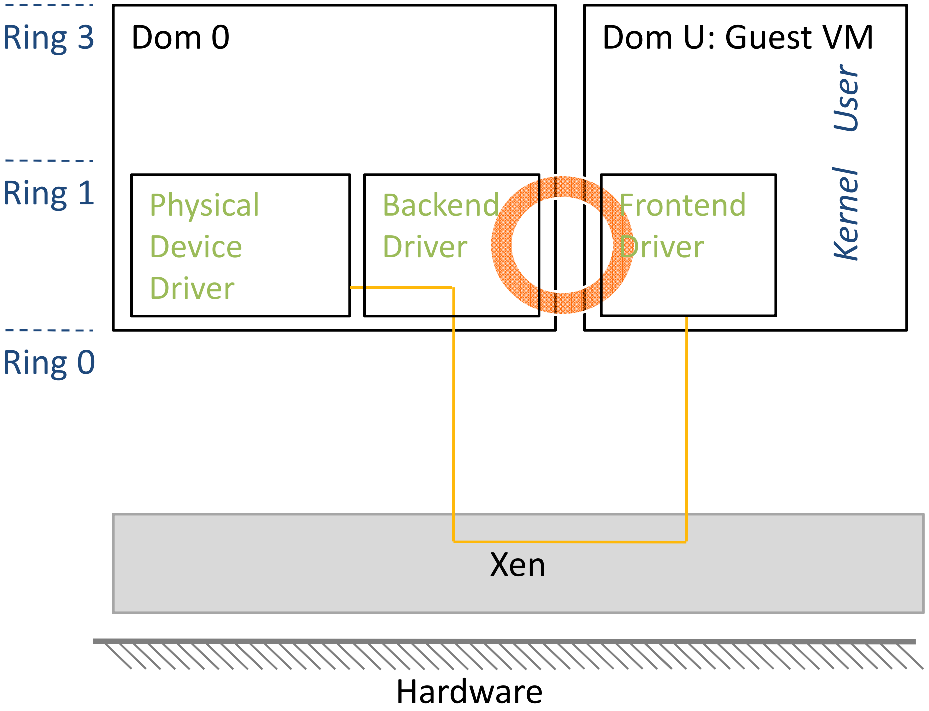
- Dependencies on hardware
  - Presence of device (e.g. GPU, FPGA)
  - Device-related configuration (e.g. VLAN)



Devices Limit Flexibility

- Today: Split driver model
  - Dependencies break if VM moves

- No easy place to plug into hardware driver
  - System portion connected in ad-hoc way

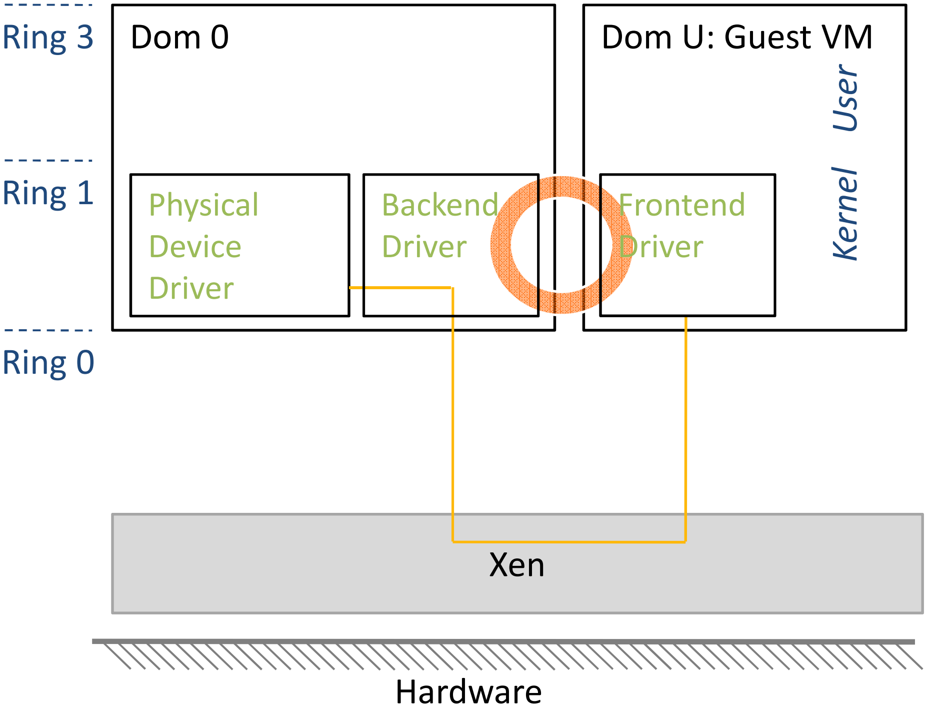


Split driver again!

- Clean separation between hardware driver and backend driver

- Standard interface between ***endpoints***

- Connected with ***wires***


Ring 3
Ring 1
Ring 0
Dom 0
Physical Device Driver
Backend Driver
Dom U: Guest VM
User
Kernel
Frontend Driver
Xen
Hardware