Last Week

- TCP Offloading
- Remote Direct Memory Access
- Network Virtualization
  - Software Defined Networking
  - OpenFlow
Today

- Wrap-up: Network Virtualization
- I/O Virtualization:
  - Device emulation
  - Paravirtualization
  - SR-IOV
  - IOMMU
OpenFlow
OpenFlow Building Blocks

- Controller talks to OpenFlow switch through a secure channel
- Switch contains:
  - One or more flow tables
  - A group table
- Flow tables:
  - Contain flow entries
  - Packets matched against flow entries
  - Flow entry determines which packet matches and what action will be taken
- Group table
  - Set of group entries
  - Each group entry has: identifier, type, counters and action bucket
  - Allows for additional action to be set on a packet: actions common for all packets of the same group

Slides adapted from Prof. Roscoe
Flow Table Entries

1. Forward packet to zero or more ports
2. Encapsulate and forward to controller
3. Send to normal processing pipeline
4. Modify Fields
5. Any extensions you add!

Switch Port | VLAN ID | VLAN pcp | MAC src | MAC dst | Eth type | IP Src | IP Dst | IP ToS | IP Prot | L4 sport | L4 dport
---|---|---|---|---|---|---|---|---|---|---|---

+ mask what fields to match
## Flow Table Entries: Examples

### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Flow Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>port3</td>
<td>00:20..</td>
<td>00:1f..</td>
<td>0800</td>
<td>vlan1</td>
<td>1.2.3.4</td>
<td>5.6.7.8</td>
<td>4</td>
<td>17264</td>
<td>80</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
</tr>
</tbody>
</table>
Pipeline Processing

- Flow tables sequentially numbered
- A flow table entry adds actions to an action set of a packet
- A flow table entry may explicitly direct the packet to another flow table
  - Can only direct a packet to a flow table with number which is greater than my own table's number
- If table entry does NOT direct a packet to another table, the pipeline stops and the associated action set is executed
OpenFlow Asynchronous Events

- Sent by the switch without being requested from the controller in case of a
  - Packet arrival for which there is not matching flow entry
  - Switch state change (e.g., new switch added)
  - Error
Example OpenFlow Applications

- Dynamic Access Control
- Seamless mobility
- Network virtualization
- Energy efficient networking
Dynamic Access Control

- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic
Seamless Mobility/Migration

- Observe hosts sends traffic from new location
- Modify flow tables to re-route the traffic
Saving Energy with OpenFlow (1)

- **Problem**: Power consumption of switches is high, even no ports are connected.
- **Key idea**: Use OpenFlow to dynamically turn off switching elements that are not needed.
Saving Energy with OpenFlow (2): The Network Power Knobs

- Actually, we have even more options: vary link speed, disable switch, move VMs, disable links
Remember: Network Virtualization Goals

- Give each application/tenant its own virtual network with its own:
  - Topology
  - Bandwidth,
  - Broadcast domain
  - ...

- Delegate administration for virtual networks
FlowVisor: Slicing the Network

- Divide the physical network into logical slices
  - Each slice/service controls its own packet forwarding
  - Give different slices to different application or owners
  - Enforce strong isolation between slices
- A network slice is a collection of sliced switches/routers
- Each slice believes it owns the data path
- Slicing Policy: specifies resource limits for each slice
  - Link Bandwidth
  - Topology
  - Maximum number of forwarding rules
Mapping packets to Slices

- TCP port#
- IP address
- MAC address

Slice 1
Slice 2
Slice 3
FlowVisor Implementation

- FlowVisor is an SDN/OpenFlow controller
  - Talks OpenFlow to the switches
- FlowVisor runs multiple OpenFlow controller, one for each slice
  - Talks OpenFlow to the 'Slice' controller
- FlowVisor intercepts and re-writes OpenFlow messages from the 'Slice' controllers
OpenFlow Challenges: Controller Delay and Overhead

- Controller is much slower than the switches
- Processing packets leads to delay and overhead
- Need to keep most packets in “fast path”
OpenFlow Challenges (2): Distributed Controller

- Controller is “single-point of failure” and potential bottleneck
- Partition or replicate controller for scalability and reliability
- Problems: keeping state consistent
Network I/O Virtualization
Data Transfer Non-Virtualized

1) Application: syscall, e.g., socket.write()

2) OS driver: issue PCI commands
   - Set up DMA operation

3) NIC:
   - transmit data
   - raise interrupt when done

Slides adapted from Prof. Roscoe
Virtualization and Hypervisors

Slides adapted from Prof. Roscoe
Virtualization and Hypervisors

How does network access work?

Slides adapted from Prof. Roscoe
Option 1: Full Device Emulation

- Guest OS unaware that it is being virtualized
- Hypervisor emulates device at the lowest level
  - Privileged instructions from guest driver trap into hypervisor
- Advantage: no changes to the guest OS required
- Disadvantage:
  - Inefficient
  - Complex

Slides adapted from Prof. Roscoe
Option 1: Paravirtualization

- Guest OS aware that it is being virtualized
  - Runs special paravirtual device drivers
- Hypervisor cooperates with guest OS through paravirtual interfaces
- Advantage:
  - Better performance
  - Simple
- Disadvantage:
  - Requires changes to the guest OS

Slides adapted from Prof. Roscoe
Paravirtualization with VirtIO

- **VirtIO**: I/O virtualization framework for Linux
  - Framework for developing paravirtual drivers
  - Split driver model: front-end and back-end driver
  - APIs for front-end and back-end to communicate

Slides adapted from Prof. Roscoe
Example: KVM Hypervisor

- Guest Application
- Guest Operating System
- QEMU
- KVM module
- Hypervisor
- Hardware

- Based on Intel VT-x
  - Additional guest execution mode
- I/O at guest OS trap into KVM (VM Exit)
- KVM schedules QEMU process to emulate I/O operation

Slides adapted from Prof. Roscoe
1) VirtIO-Net driver adds packet to shared VirtIO memory
2) VirtIO-Net driver causes trap into KVM
3) KVM schedules QEMU VirtIO Back-end
4) VirtIO back-end gets packet from shared VirtIO memory and emulates I/O (via system call)
5) KVM resumes guest

Slides adapted from Prof. Roscoe
Vhost: Improved VirtIO Backend

- Vhost puts VirtIO emulation code into the kernel
  - Instead of performing system calls from userspace (QEMU)

Slides adapted from Prof. Roscoe
Inter-VM communication

VM1

Guest Application
Guest Operation System

Guest Application
Guest Operation System

Guest Application
Guest Operation System

VM2

VM3

Hypervisor

NIC

Slides adapted from Prof. Roscoe
Inter-VM communication

How does inter-VM communication work?

Slides adapted from Prof. Roscoe
Switch Externally...

...either in

- **External switch:**
  - Simplifies configuration: all switching controlled/configured by the network
  - Latency = 2xDMA + one 2hops
- **NIC**
  - Latency = 2xDMA

Slides adapted from Prof. Roscoe
Switch in Hypervisor

- Uses host CPU cycles
- Latency = 1 software copy

Slides adapted from Prof. Roscoe
Switched Vhost in KVM

Slides adapted from Prof. Roscoe
Where are we?

- Option 1: Full emulation
  - No changes to guest required
  - Complex
  - Inefficient

- Option 2: Paravirtualization
  - Requires special guest drivers
  - Enhanced performance
Where are we?

- Option 1: Full emulation
  - No changes to guest required
  - Complex
  - Inefficient

- Option 2: Paravirtualization
  - Requires special guest drivers
  - Enhanced performance

Not good enough! Still requires hypervisor involvement, e.g., interrupt relaying

Slides adapted from Prof. Roscoe
Where are we?

- Option 1: Full emulation
  - No changes to guest required
  - Complex
  - Inefficient

- Option 2: Paravirtualization
  - Requires special guest drivers
  - Enhanced performance

- Option 3: Passthrough
  - Directly assign NIC to VM
  - No hypervisor involvement: best performance

Not good enough!
Still requires hypervisor involvement, e.g., interrupt relaying
Paravirtual vs Passthrough in KVM

Slides adapted from Prof. Roscoe
Challenges with Passthrough / Direct Assignment

- VM tied to specific NIC hardware
  - Makes VM migration more difficult

- VM driver issues DMA requests using VM addresses
  - Incorrect: VM physical addresses are host virtual addresses (!)
  - Security concern: addresses may belong to other VM
  - Potential solution: let VM translate it's physical addresses to real DMA addresses
    - Still safety problem: exposes driver details to hypervisor, bugs in driver could result in incorrect translations
  - Solution: Use an IOMMU to translate/validate DMA requests from the device

- Need a different NIC for each VM
  - Solution: SR-IOV, emulate multiple NICs at hardware level
Memory Address Terminology

- **Virtual Address**
  - Address in some virtual address space in a process running in the guest OS

- **Physical Address:**
  - Hardware address as seen by the guest OS, i.e., physical address in the virtual machine

- **Machine address:**
  - Real hardware address on the physical machine as seen by the Hypervisor

Slides adapted from Prof. Roscoe
Memory Address Terminology

- Virtual Address
  - Address in some virtual address space in a process running in the guest OS

- Physical Address:
  - Hardware address as seen by the guest OS, i.e., physical address in the virtual machine

- Machine address:
  - Real hardware address on the physical machine as seen by the Hypervisor
IOMMU

PCIe function (e.g. NIC)

Memory controller

Main memory

Slides adapted from Prof. Roscoe
VMM programs IOMMU with VM-physical to machine address translations

IOMMU

Memory controller

Main memory

PCle function (e.g. NIC)

Slides adapted from Prof. Roscoe
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

IOMMU

Memory controller

Main memory

Guest OS programs NIC with VM-physical address of DMA

PCIe function (e.g. NIC)

Slides adapted from Prof. Roscoe
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

IOMMU

Memory controller

Main memory

Guest OS programs NIC with VM-physical address of DMA

NIC issues a DMA request to VM physical memory

PCIe function (e.g. NIC)

Slides adapted from Prof. Roscoe
IOMMU

- IOMMU checks and translates to machine (real) address for transfer
- VMM programs IOMMU with VM-physical to machine address translations

- PCIe function (e.g. NIC)

- NIC issues a DMA request to VM physical memory
- Guest OS programs NIC with VM-physical address of DMA

- Memory controller
- Main memory

Slides adapted from Prof. Roscoe
IOMMU

VMM programs IOMMU with VM-physical to machine address translations

IOMMU checks and translates to machine (real) address for transfer

Memory controller

Guest OS programs NIC with VM-physical address of DMA

NIC issues a DMA request to VM physical memory

Memory controller accesses memory

PCIe function (e.g. NIC)
SR-IOV

- Single-Root I/O Virtualization
- Key idea: dynamically create new “PCI devices”
  - Physical Function (PF): original device, full functionality
  - Virtual Function (VF): extra device, limited functionality
  - VFs created/destroyed via PF registers
- For Networking:
  - Partitions a network card's resources
  - With direct assignment can implement passthrough

Slides adapted from Prof. Roscoe
SR-IOV in Action

Slides adapted from Prof. Roscoe
References

- I/O Virtualization, Mendel Rosenblum, ACM Queue, January 2012