Advanced Computer Networks
263-3501-00

RDMA, Network Virtualization

Patrick Stuedi
Spring Semester 2013
Last Week

- Scaling Layer 2
  - Portland
  - VL2

- TCP
  - Incast
  - Data Center TCP (DCTCP)
  - Multipath TCP
Today

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

- Benchmark:
  - Partition the network into two parts, src and dst
  - For each host in src connect to host in dst, avoid duplicates

- Regular TCP with ECMP flow hashing
  - Static hashing, creates hotspots (certain switches overly loaded)

- Multipath TCP with ECMP subflow hashing
  - Packets always travel on the least congested subflow
Network latencies in Data centers

Factors that contribute to latency in TCP datacenters
- Delay: cost of a single traversal of the component
- RTT: total cost in a round-trip traversing 5 switches in each direction

OS overhead per packet exchanged between two hosts attached to the same switch: 
\[(2 \times 15)/(2 \times 2.5 + 2 \times 15 + 10) = 66\% \]
Packet Processing Overhead

- **Sending-side:**
  - Data is copied from the application buffer into a socket buffer
  - Data is DMA copied into NIC buffer

- **Receiver side:**
  - Data is DMA copied from NIC buffer into socket buffer
  - Data is copied into application buffer
  - Application is scheduled (context switching)
Throughput and CPU load at 1Gbit/s and 10Gbit/s

- Throughput limited because of high CPU load

- RX side typically more CPU intensive because highly asynchronous
TCP Offloading

- What is TCP offloading
  - Moving IP and TCP processing to the Network Interface (NIC)

- Main justification for TCP offloading
  - Reduction of host CPU cycles for protocol header processing, checksumming
  - Fewer CPU interrupts
  - Fewer bytes copied over the memory bus
  - Potential to offload expensive features such as encryption
TCP Offload Engines (TOEs)
Problems of TCP offloading

- Moore’s Law worked against “smart” NICs
  - CPU's used to get faster
- Now many cores, cores don't get faster
  - Network processing is hard to parallelize
- TCP/IP headers don’t take many CPU cycles
- TOEs impose complex interfaces
  - Protocol between TOE & CPU can be worse than TCP
- Connection management overhead
  - For short connections, overwhelms any savings
Where TCP offload helps

- Sweet spot for TCP offload might be apps with:
  - Very high bandwidth
  - Relatively low end-to-end latency network paths
  - Long connection durations
  - Relatively few connections

- Typical examples of these might be:
  - Storage-server access
  - Cluster interconnects
User-level networking: Remove OS from the data path

- Transport offloading is not enough!
  - Still have system call overhead, context switch, memory copying

- U-Net:
  - Eicken, Basu, Buch, Vogels, Cornell University, 1995
  - Virtual network interface that allows applications to send and receive messages without operating system intervention
  - Move all buffer management and packet processing to user-space (zero-copy)

- traditional networking architecture
  - Kernel controls the network
  - All communication via kernel

- U-Net architecture:
  - Application access network directly via MUX
  - Kernel involved only in connection setup
U-Net Building Blocks

- **End points**
  - application’s handle into the network

- **Buffer area**
  - hold message data for sending or buffer space for receiving

- **Message queues**
  - hold *descriptors* pointing to buffer area
U-Net communication

- **Initialization:**
  - Create one or more endpoints
  - Register user buffers with endpoints and associated them with a *tag*

- **Sending**
  - Composes the data in the endpoint buffer area
  - Push a descriptor for the message onto the send queue
  - NIC transmits the message after marking it with the appropriate message tag.

- **Receiving:**
  - Push a message descriptor with pointers to the buffers onto the receive queue.
  - Incoming messages get de-multiplexed based on the message *tag*
  - Data is placed within the target buffer of the application by the NIC
History of User-Level Networking

- U-Net one of the first (if not the first) system to propose OS-bypassing
- Other early works
  - SHRIMP: Virtual Memory Mapped Interfaces, IEEE Micro, 1995
  - “Separating Data and Control Transfer in Distributed Operating Systems”, Thekkath et. al., ASPLOS'94
- Efforts of U-Net eventually resulted in the Virtual Interface Architecture (VIA)
  - Specification jointly proposed by Compaq, Intel and Microsoft, 1997
- VIA architecture has led to the implementation of various high performance networking stacks: Infiniband, iWARP, Roce:
  - Commonly referred to as RDMA network stacks
  - RDMA = Remote Direct Memory Access
RDMA Architecture

<table>
<thead>
<tr>
<th>RDMA verbs userlib</th>
<th>Socket layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>Kernel Mod</td>
<td>Ethernet</td>
</tr>
<tr>
<td>NIC Driver</td>
<td></td>
</tr>
</tbody>
</table>

- Traditional socket interface involves kernel
- RDMA interface involves kernel only on control path, but access the RDMA capable NIC (rNIC) directly from user space on the data path
- Dedicated verbs interface used for RDMA, instead of traditional socket interface
RDMA Queue Pairs (QPs)

Applications use 'verbs' interface to

- **Register memory:**
  - Operating system will make sure the memory is pinned and accessible by DMA
- **Create a queue pair (QP)**
  - send/recv queue
- **Create a completion queue (CQ)**
  - RNIC puts a new completion-queue element into the CQ after an operation has completed
- **Send/Receive data**
  - Place a work-request element (WQE) into the send or recv queue
  - WQE points to user buffer and defines the type of the operation (e.g., send, recv, ..)
Applications use 'verbs' interface to

- Register memory:
  - Operating system will make sure the memory is pinned and accessible by DMA

- Create a queue pair (QP)
  - send/recv queue

- Create a completion queue (CQ)
  - RNIC puts a new completion-queue element into the CQ after an operation has completed

- Send/Receive data
  - Place a work-request element (WQE) into the send or recv queue
  - WQE points to user buffer and defines the type of the operation (e.g., send, recv, ..)
RDMA operations

- **Send/Receive:**
  - Two-sided operation: data exchange naturally involves both ends of the communication channel
  - Each send operation must have a matching receive operation
  - Send WR specifies where the data should be taken from
  - Receive WR on the remote machine specifies where the inbound data is to be placed

- **RDMA (Remote Direct Memory Access):**
  - Two independent operations: RDMA Read and RDMA Write
  - Only the application issuing the operation is actively involved in the data transfer
  - An RDMA Write not only specifies where the data should be taken from, but also where it is to be placed (remotely)
  - An RDMA Read requires a buffer advertisement prior to data exchange
Example: RDMA Send/Recv (1)

- Sender and receiver have created their Qps and Cqs
- Sender has registered a buffer for sending
- Receiver has registered a buffer for receiving
Example: RDMA Send/Recv (2)

- Receiver places a WQE into its receive queue
- Sender places a WQE into its send queue
Data is transferred between the hosts
  - Involves two DMA transfers, one at the sender and one at the receiver
• After operation has finished, a CQE is placed into the completion queue of the sender
RDMA implementations

- Infiniband
  - Compaq, HP, IBM, Intel Microsoft and Sun Microsystems
  - Provides RDMA semantics
  - First spec released 2000
  - Based on point-to-point switched fabric
  - Designed from ground up (has its own physical layer, switches, NICs, etc)

- IWARP (Internet Wide Area RDMA Protocol)
  - RDMA semantics implemented over offloaded TCP/IP
  - Requires custom NICs, but uses Ethernet

- RoCE
  - RDMA semantics implemented directly over Ethernet

- All of those implementation can be programmed through the **verbs interface**
Typical CPU loads for three network stack implementations
Network Virtualization
Example: Enterprise Data Center

Physical Network

Data Center

- Aggr/Spine Switch
- Top-of-Rack Switch
- Servers (Physical or Virtual)

Guido Appenzeller [OpenSummit'11]
Network Virtualization
Challenge: Multiple Applications/Tenants

Physical Network

Guido Appenzeller [OpenSummit'11]
Network Virtualization

Challenge: Multiple Applications/Tenants

Guido Appenzeller [OpenSummit'11]
Network Virtualization: Goals

- Virtualize the network like we virtualize servers today:
  - Server virtualization: decouple workload from server hardware (e.g., CPU, Memory, I/O)
  - Network virtualization: decouple network services from physical network hardware
Network Virtualization: Goals (2)

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

- Delegate administration for virtual networks
Network Virtualization: Goals (2)

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

- Delegate administration for...
Network Virtualization: Goals (2)

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

- Delegate administration for virtual networks

How to get there?
- VLAN,
- SDN,
- Openflow,
- NOX,
- etc.
Virtual Local Area Network (VLAN)

- Remember from Lecture 2 (Principles):
  - Create multiple virtual LANs from a single physical network
    - Nodes in different LANs can't communicate
    - Broadcast isolation (e.g. ARP)
Defining VLANs

- **Static: Port-based**
  - Switch port statically defines the VLAN a host is part of

- **Dynamic: MAC-based**
  - MAC address defines which VLAN a host is part of
  - Typically configured by network administrator
802.1Q tagging

- VLAN tagging
  - Tag added to layer-2 frame at ingress switch
  - Tag stripped at egress switch
  - Tag defines on which VLAN the packet is routed
- Allows a single interconnect to transport data for various VLAN
- Trunk port: port that sends and receives tagged frames on multiple VLANs

4096 VLANs possible
Communication between VLANs

- Forwarding between VLANs requires going through a router
  - VLAN tag typically gets lost at router boundaries
802.1q Double Tagging

- Useful for Internet service providers
  - Allowing them to use VLANs internally while mixing traffic from clients that are already VLAN-tagged

- How it works
  - Tunnel one VLAN through another VLAN
  - VLAN tunnels add a second VLAN id to the frame (called the outer tag)
  - Switching of packets entering the tunnel is done based on the outer tag
  - Outer tag is removed at tunnel egress switch
Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
  - Requires administrator
- Number of VLANs limited (4096)
Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
  - Requires administrator
- Number of VLANs limited (4096)

We need something better
Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
  - Requires administrator
- Number of VLANs limited (4096)

We need something better: SDN/Openflow
Software Defined Networks (SDN)
Traditional Networks: Data Plane

- Task: Forward, filter, buffer, mark, rate-limit and measure packets

Jennifer Rexford [Princeton]
Traditional Networks: Control Plane

- Task: Track topology changes, compute routes, install forwarding tables
  - Example: Ethernet Spanning Tree Protocol

**Control plane:**
Distributed algorithms

Jennifer Rexford [Princeton]
Traditional Networks: Management Plane

- Task: Collect measurements and configure equipment
  - Example: SNMP

Management plane: Human time scale

Jennifer Rexford [Princeton]
Software Defined Networking (SDN): From a Bird's Eye View

Logically centralized control

Smart, slow

API to the data plane (e.g., OpenFlow)

Dumb, fast

Switches

Jennifer Rexford [Princeton]
Software Defined Networking:

- Architecture for future enterprise and data center networks
- Originally developed for University campus networks
  - To allow researchers testing out new features
- Ideas
  - Separate network intelligence from data path
  - Extract control functionality into a logically centralized controller
Building Blocks of SDN

Abstract Network View

Network Virtualization

Global Network View

Network OS
Building Blocks of SDN

At least one network OS, probably many

Abstract Network View

Network Virtualization

Global Network View

Network OS
Building Blocks of SDN

- **Abstract Network View**
  - At least one network OS, probably many

- **Network Virtualization**
  - Global Network View

- **Network OS**
  - Packet Forwarding
  - At least one network OS, probably many

- Functions ($f(View)$)
  - Control Programs

- Diagram Elements:
  - Yellow circles and arrows
  - Blue arrows
  - Red arrows
Building Blocks of SDN

- **Consistent up-to-date global network view**
- **At least one network OS, probably many**
- **Packet Forwarding**
- **Network Virtualization**
- **Abstract Network View**

- **Network OS**
- **Packet Forwarding**
- **Control Programs**

- **Global Network View**
Building Blocks of SDN

```
#include <netinet/in.h>
...
if (pkt->tcp->dport == 22)
    dropPacket(pkt);
...
```

Abstract Network View

Network Virtualization

Global Network View
Network OS

- Distributed system that creates a consistent up-to-date network view
- Runs on servers (controllers) in the network
- Uses forwarding abstraction to get/put state from/to switches
- Platform for running SDN applications (or control programs)
- Example controllers: NOX, Onix, Floodlight, … + more
Control program operates on view of network

- **Input:** global network view (graph/database)
- **Output:** configuration of each network devices

Control program is not a distributed system

- Abstractions hide details of distributed state

Event-driven programming: register handlers for events

Example: NOX control program to set VLAN tagging rules on user authentication
A Nice Analogy (1)

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

Specialized Applications
Specialized Operating System
Specialized Hardware

Open Interface
Windows (OS) or Linux or Mac OS

Microprocessor
A Nice Analogy (2)

Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation

Specialized Features
Specialized Control Plane
Specialized Hardware
A Nice Analogy (3)

Vertically integrated
Closed, proprietary
Slow innovation

Control Plane
Open Interface
Merchant Switching Chips

Horizontal
Open interfaces
Rapid innovation

Control program
Network OS
Forwarding abstraction

Specialized Features
Specialized Control Plane
Specialized Hardware

Monday 6 May 2013
OpenFlow
OpenFlow Basics

- Started in 2004 with the PhD thesis of Martin Casado
  - Now CEO at Nicira Networks
- Original motivation: drive innovation for new network architectures/protocols
- Gap in the tool space, current systems either
  - do NOT perform at the scale and speed we want OR
  - are NOT open to be programmed by researchers

<table>
<thead>
<tr>
<th></th>
<th>Performance Fidelity</th>
<th>Scale</th>
<th>Real User Traffic?</th>
<th>Complexity</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>medium</td>
<td>medium</td>
<td>no</td>
<td>medium</td>
<td>yes</td>
</tr>
<tr>
<td>Emulation</td>
<td>medium</td>
<td>low</td>
<td>no</td>
<td>medium</td>
<td>yes</td>
</tr>
<tr>
<td>Software Switches</td>
<td>poor</td>
<td>low</td>
<td>yes</td>
<td>medium</td>
<td>yes</td>
</tr>
<tr>
<td>NetFPGA</td>
<td>high</td>
<td>low</td>
<td>yes</td>
<td>high</td>
<td>yes</td>
</tr>
<tr>
<td>Network Processors</td>
<td>high</td>
<td>medium</td>
<td>yes</td>
<td>high</td>
<td>yes</td>
</tr>
<tr>
<td>Vendor Switches</td>
<td>high</td>
<td>high</td>
<td>yes</td>
<td>low</td>
<td>no</td>
</tr>
</tbody>
</table>
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
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!

+ 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>00:1f:..</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>
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.

![48-port Switch](image)

- ~150W nothing connected
- ~185W all 48 1G links on

**Power Meter**
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

Slices:
- 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: 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
References

- “Ethane: Taking Control of the Enterprise”, Sigcomm 2007
- “Can the Production Network Be the Testbed?”, OSDI 2010
- “A Survey of Virtual LAN Usage in Campus Networks”, IEEE Communications, 2011
### Flow Table Entries: Example (2)

#### Routing

<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>

#### VLAN 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>00:1f..</td>
<td>*</td>
<td>vlan1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6, port7, port9</td>
</tr>
</tbody>
</table>
FlowVisor Message Handling

Policy Check: Is this rule allowed?

Full Line Rate Forwarding

Policy Check: Who controls this packet?

Exception

OpenFlow Firmware

Data Path

Packet

FlowVisor

OpenFlow

Rule

Alice Controller

Bob Controller

Cathy Controller