NETWORK VIRTUALIZATION
BASED ON
SOFTWARE DEFINED NETWORK
SOFTWARE DEFINED NETWORK

Introduction
Motivation
Concept
Open Flow
Virtual Switch
We have lost our way

Routing, management, mobility management, access control, VPNs, ...

Million of lines of source code 5400 RFCs Barrier to entry

500M gates Bloated Power Hungry

10Gbytes RAM

Many complex functions baked into the infrastructure

OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, MPLS, redundant layers, ...

An industry with a “mainframe-mentality”
• Lack of competition means glacial innovation
• Closed architecture means blurry, closed interfaces
• Vertically integrated, complex, closed, proprietary
• Not suitable for experimental ideas
• Not good for network owners & users
• Not good for researchers
Glacial process of innovation made worse by captive standards process

- Driven by vendors
- Consumers largely locked out
- Lowest common denominator features
- Glacial innovation
SOFTWARE DEFINED NETWORK

Introduction
Motivation
Concept
Open Flow
Virtual Switch
Trend

Computer Industry

Virtualization layer

Windows (OS)  Linux  Mac OS

x86 (Computer)

Network Industry

Virtualization or “Slicing”

NOX (Network OS)

OpenFlow

Network Industry
The “Software-defined Network”
The "Software-defined Network"

1. Open interface to hardware
2. At least one good operating system
   Extensible, possibly open-source
3. Well-defined open API

Network Operating System

- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
Isolated “slices”

Many operating systems, or Many versions

Open interface to hardware

Virtualization or “Slicing” Layer

Open interface to hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware
Consequences

More innovation in network services

- Owners, operators, 3rd party developers, researchers can improve the network
- E.g. energy management, data center management, policy routing, access control, denial of service, mobility

Lower barrier to entry for competition

- Healthier market place, new players
SOFTWARE DEFINED NETWORK

Introduction
Motivation
Concept
Open Flow
Virtual Switch
Traditional network node: Router

- Router can be partitioned into control and data plane
  - Management plane/ configuration
  - Control plane / Decision: OSPF (Open Shortest Path First)
  - Data plane / Forwarding
Traditional network node: Switch

- Typical Networking Software
  - Management plane
  - Control Plane – The brain/decision maker
  - Data Plane – Packet forwarder
SDN Concept

- Separate Control plane and Data plane entities
  - Network intelligence and state are logically centralized
  - The underlying network infrastructure is abstracted from the applications
- Execute or run Control plane software on general purpose hardware
  - Decouple from specific networking hardware
  - Use commodity servers
- Have programmable data planes
  - Maintain, control and program data plane state from a central entity
- An architecture to control not just a networking device but an entire network
Control Program

Control program operates on view of network

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

Control program is not a distributed system

- Abstraction hides details of distributed state
Forwarding Abstraction

Purpose: Abstract away forwarding hardware

Flexible
- Behavior specified by control plane
- Built from basic set of forwarding primitives

Minimal
- Streamlined for speed and low-power
- Control program not vendor-specific

OpenFlow is an example of such an abstraction
OpenFlow Basics

Control Program A  Control Program B

Network OS

OpenFlow Protocol

Ethernet Switch
OpenFlow Basics

“If header = $p$, send to port 4”
“If header = $q$, overwrite header with $r$, add header $s$, and send to ports 5,6”
“If header = $?$, send to me”
**Match** arbitrary bits in headers:

<table>
<thead>
<tr>
<th>Header</th>
<th>Data</th>
</tr>
</thead>
</table>

Match: 1000x01xx0101001x

- Match on any header, or new header
- Allows any flow granularity

**Action**

- Forward to port(s), drop, send to controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate
General Forwarding Abstraction

- Small set of primitives
  “Forwarding instruction set”
- Protocol independent
  Backward compatible
- Switches, routers, WiFi APs, basestations, TDM/WDM
SOFTWARE DEFINED NETWORK

Introduction
Motivation
Concept
Open Flow
Virtual Switch
What is OpenFlow

• OpenFlow is similar to an x86 instruction set for the network

• Provide open interface to “black box” networking node
  ▪ (ie. Routers, L2/L3 switch) to enable visibility and openness in network

• Separation of control plane and data plane.
  ▪ The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry
  ▪ The control path consists of a controller which programs the flow entry in the flow table

• OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries
OpenFlow Consortium

http://OpenFlowSwitch.org

• Goal
  ▪ Evangelize OpenFlow to vendors
  ▪ Free membership for all researchers
  ▪ Whitepaper, OpenFlow Switch Specification, Reference Designs
  ▪ Licensing: Free for research and commercial use
OpenFlow building blocks

- Monitoring/debugging tools:
  - oftrace
  - oflops
  - openseer

- Stanford Provided Applications:
  - ENVI (GUI)
  - LAVI
  - n-Casting
  - Expedient

- Controler:
  - NOX
  - Beacon
  - Trema
  - Maestro
  - ONIX

- Slicing Software:
  - FlowVisor

- Commercial Switches:
  - HP, NEC, Pronto, Juniper, and many more

- Stanford Provided OpenFlow Switches:
  - Software Ref. Switch
  - NetFPGA
  - Broadcom Ref. Switch
  - OpenWRT
  - PCEngine WiFi AP
  - Open vSwitch
Components of OpenFlow Network

- **Controller**
  - OpenFlow protocol messages
  - Controlled channel
  - Processing
    - Pipeline Processing
    - Packet Matching
    - Instructions & Action Set

- **OpenFlow switch**
  - Secure Channel (SC)
  - Flow Table
    - Flow entry
<table>
<thead>
<tr>
<th>Name</th>
<th>Lang</th>
<th>Platform(s)</th>
<th>License</th>
<th>Original Author</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow Ref.</td>
<td>C</td>
<td>Linux</td>
<td>OpenFlow License</td>
<td>Stanford/Nicira</td>
<td>not designed for extensibility</td>
</tr>
<tr>
<td>NOX</td>
<td>Python, C++</td>
<td>Linux</td>
<td>GPL</td>
<td>Nicira</td>
<td>actively developed</td>
</tr>
<tr>
<td>Beacon</td>
<td>Java</td>
<td>Win, Mac, Linux, Android</td>
<td>GPL (core), FOSS Licenses for your code</td>
<td>David Erickson (Stanford)</td>
<td>runtime modular, web UI framework, regression test framework</td>
</tr>
<tr>
<td>Maestro</td>
<td>Java</td>
<td>Win, Mac, Linux</td>
<td>LGPL</td>
<td>Zheng Cai (Rice)</td>
<td>includes emulator, regression test framework</td>
</tr>
<tr>
<td>Trema</td>
<td>Ruby, C</td>
<td>Linux</td>
<td>GPL</td>
<td>NEC</td>
<td>includes emulator, regression test framework</td>
</tr>
<tr>
<td>RouteFlow</td>
<td></td>
<td>Linux</td>
<td>Apache</td>
<td>CPqD (Brazil)</td>
<td>virtual IP routing as a service</td>
</tr>
</tbody>
</table>
Secure Channel (SC)

- SC is the **interface** that connects each OpenFlow switch to controller.

- A controller **configures and manages the switch** via this interface.
  - Receives events from the switch
  - Send packets out the switch

- SC **establishes and terminates the connection** between OpenFlow Switch and the controller using the procedures
  - Connection Setup
  - Connection Interrupt

- The SC connection is a **TLS connection**. Switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key.
Flow Table

- Flow table in switches, routers, and chipsets

Flow 1.

- Rule (exact & wildcard)
- Action
- Statistics

Flow 2.

- Rule (exact & wildcard)
- Action
- Statistics

Flow 3.

- Rule (exact & wildcard)
- Action
- Statistics

Flow N.

- Rule (exact & wildcard)
- Default Action
- Statistics
Flow Entry

- A flow entry consists of
  - **Match fields**
    - Match against packets
  - **Action**
    - Modify the action set or pipeline processing
  - **Stats**
    - Update the matching packets

In Port | Src MAC | Dst MAC | Eth Type | Vlan Id | IP Tos | IP Proto | IP Src | IP Dst | TCP Src Port | TCP Dst Port
--- | --- | --- | --- | --- | --- | --- | --- | --- | --- | ---

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

1. Packet
2. Byte counters
### 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>00:1f:..</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td><strong>port6</strong></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><strong>port6</strong></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>
### 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>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></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>*</td>
<td>*</td>
<td>00:1f..</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6, port7, port9</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>vlan1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>
OpenFlow Usage

Controller

Peter's code

OpenFlow Protocol

Peter
Usage examples

Peter’s code:

- Static “VLANs”
- His own new routing protocol: unicast, multicast, multipath, load-balancing
- Network access control
- Home network manager
- Mobility manager
- Energy manager
- Packet processor (in controller)
- IPvPeter
- Network measurement and visualization
- ...
Separate VLANs for Production and Research Traffic

Normal L2/L3 Processing

Flow Table

Controller

Research VLANs

Production VLANs
Dynamic Flow Aggregation on an OpenFlow Network

• Scope
  ▪ Different Networks want different flow granularity (ISP, Backbone,...)
  ▪ Switch resources are limited (flow entries, memory)
  ▪ Network management is hard
  ▪ Current Solutions: MPLS, IP aggregation
• How do OpenFlow Help?
  - Dynamically define flow granularity by wildcarding arbitrary header fields
  - Granularity is on the switch flow entries, no packet rewrite or encapsulation
  - Create meaningful bundles and manage them using your own software (reroute, monitor)
Virtualizing OpenFlow

- Network operators “Delegate” control of subsets of network hardware and/or traffic to other network operators or users
- Multiple controllers can talk to the same set of switches
- Imagine a hypervisor for network equipments
- Allow experiments to be run on the network in isolation of each other and production traffic
Switch Based Virtualization

Exists for NEC, HP switches but not flexible enough

Production VLANs

Research VLAN 1

Research VLAN 2

Flow Table

Controller

Controller

Normal L2/L3 Processing
FlowVisor

- A network hypervisor developed by Stanford
- A software proxy between the forwarding and control planes of network devices
FlowVisor-based Virtualization

Topology discovery is per slice

OpenFlow Switch

OpenFlow Switch

OpenFlow Switch

OpenFlow Switch

OpenFlow FlowVisor & Policy Control

OpenFlow Protocol

OpenFlow Protocol

OpenFlow Protocol
Separation not only by VLANs, but any L1-L4 pattern

- OpenFlow Switch
- OpenFlow Switch
- OpenFlow Switch
- OpenFlow Switch

Broadcast
Multicast
http Load-balancer

OpenFlow Protocol
OpenFlow Protocol
OpenFlow Protocol

dl_dst=FFFFFFFFFFFFF
tp_src=80, or tp_dst=80
FlowVisor Slicing

- Slices are defined using a slice definition policy
  - The policy language specifies the slice’s resource limits, flowspace, and controller’s location in terms of IP and TCP port-pair
  - FlowVisor enforces transparency and isolation between slices by inspecting, rewriting, and policing OpenFlow messages as they pass
FlowVisor Resource Limits

- FV assigns hardware resources to “Slices”
  - **Topology**
    - Network Device or Openflow Instance (DPID)
    - Physical Ports
  - **Bandwidth**
    - Each slice can be assigned a per port queue with a fraction of the total bandwidth
  - **CPU**
    - Employs Course Rate Limiting techniques to keep new flow events from one slice from overrunning the CPU
  - **Forwarding Tables**
    - Each slice has a finite quota of forwarding rules per device
Slicing

Slice: OpenRoads
Slice: Aggregation
Slice: Production
Slice: PlugnServ
Physical Network
Slice: OpenPipes
FlowVisor FlowSpace

- FlowSpace is defined by a collection of packet headers and assigned to “Slices”
  - Source/Destination MAC address
  - VLAN ID
  - Ethertype
  - IP protocol
  - Source/Destination IP address
  - ToS/DSCP
  - Source/Destination port number
FlowSpace: Maps Packets to Slices

- TCP port#
- IP address
- MAC address

- Slice 1
- Slice 2
- Slice 3
FlowVisor Slicing Policy

- **FV intercepts OF messages from devices**
  - FV only sends control plane messages to the Slice controller if the source device is in the Slice topology.
  - Rewrites OF feature negotiation messages so the slice controller only sees the ports in its slice
  - Port up/down messages are pruned and only forwarded to affected slices
FlowVisor Slicing Policy

• FV intercepts OF messages from controllers
  ▪ Rewrites flow insertion, deletion & modification rules so they don’t violate the slice definition
    • Flow definition – ex. Limit Control to HTTP traffic only
    • Actions – ex. Limit forwarding to only ports in the slice
  ▪ Expand Flow rules into multiple rules to fit policy
    • Flow definition – ex. If there is a policy for John’s HTTP traffic and another for Uwe’s HTTP traffic, FV would expand a single rule intended to control all HTTP traffic into 2 rules.
    • Actions – ex. Rule action is send out all ports. FV will create one rule for each port in the slice.
  ▪ Returns “action is invalid” error if trying to control a port outside of the slice
FlowVisor Message Handling

Policy Check: Is this rule allowed?
Policy Check: Who controls this packet?

Full Line Rate Forwarding

OpenFlow Firmware

Exception

Packet

Data Path
SOFTWARE DEFINED NETWORK

Introduction
Motivation
Concept
Open Flow
Virtual Switch
INTRODUCTION

• Due to the cloud computing service, the number of virtual switches begins to expand dramatically
  ▪ Management complexity, security issues and even performance degradation

• Software/hardware based virtual switches as well as integration of open-source hypervisor with virtual switch technology is exhibited
Software-Based Virtual Switch

- The hypervisors implement vSwitch
- Each VM has at least one virtual network interface cards (vNICs) and shared physical network interface cards (pNICs) on the physical host through vSwitch
- Administrators don’t have effective solution to separate packets from different VM users
- For VMs reside in the same physical machine, their traffic visibility is a big issue
Issues of Traditional vSwitch

• The traditional vSwitchs lack of advanced networking features such as VLAN, port mirror, port channel, etc.

• Some hypervisor vSwitch vendors provide technologies to fix the above problems
  ▪ OpenvSwitch may be superior in quality for the reasons
Open vSwitch

• A software-based solution
  ▪ Resolve the problems of network separation and traffic visibility, so the cloud users can be assigned VMs with elastic and secure network configurations

• Flexible Controller in User-Space
• Fast Datapath in Kernel

Server

Open vSwitch Controller

Open vSwitch Datapath
Open vSwitch Concepts

- Multiple ports to physical switches
  - A port may have one or more interfaces
    - Bonding allows more than once interface per port
- Packets are forwarded by flow
- Visibility
  - NetFlow
  - sFlow
  - Mirroring (SPAN/RSPAN/ERSPAN)
- IEEE 802.1Q Support
  - Enable virtual LAN function
  - By attaching VLAN ID to Linux virtual interfaces, each user will have its own LAN environment separated from other users
Open vSwitch Concepts

- Fine-grained ACLs and QoS policies
  - L2-L4 matching
  - Actions to forward, drop, modify, and queue
  - HTB and HFSC queuing disciplines
- Centralized control through OpenFlow
- Works on Linux-based hypervisors:
  - Xen
  - XenServer
  - KVM
  - VirtualBox
Open vSwitch Contributors (Partial)
Packets are Managed as Flows

- A flow may be identified by any combination of
  - Input port
  - VLAN ID (802.1Q)
  - Ethernet Source MAC address
  - Ethernet Destination MAC address
  - IP Source MAC address
  - IP Destination MAC address
  - TCP/UDP/... Source Port
  - TCP/UDP/... Destination Port
Packets are Managed as Flows

• The 1st packet of a flow is sent to the controller
• The controller programs the datapath's actions for a flow
  ▪ Usually one, but may be a list
  ▪ Actions include:
    • Forward to a port or ports
    • mirror
    • Encapsulate and forward to controller
    • Drop
• And returns the packet to the datapath
• Subsequent packets are handled directly by the datapath
• KVM and Xen provide Live Migration
• With bridging, IP address migration must occur with in the same L2 network
• Open vSwitch avoids this problem using GRE tunnels
Hardware-Based Virtual Switch

• Why hardware-based?
  ▪ Software virtual switches consume CPU and memory usage
  ▪ Possible inconsistency of network and server configurations may cause errors and is very hard to troubleshooting and maintenance

• Hardware-based virtual switch solution emerges for better resource utilization and configuration consistency
Virtual Ethernet Port Aggregator

• A standard led by HP, Extreme, IBM, Brocade, Juniper, etc.

• An emerging technology as part of IEEE 802.1Qbg Edge Virtual Bridge (EVB) standard

• The main goal of VEPA is to allow traffic of VMs to exit and re-enter the same server physical port to enable switching among VMs
• VEPA software update is required for host servers in order to force packets to be transmitted to external switches

• An external VEPA enabled switch is required for communications between VMs in the same server

• VEPA supports “hairpin” mode which allows traffic to “hairpin” back out the same port it just received it from--- requires firmware update to existing switches
Pros. and Cons. for VEPA

• **Pros**
  - Minor software/firmware update, network configuration maintained by external switches

• **Cons**
  - VEPA still consumes server resources in order to perform forwarding table lookup
با تشکر