Basic network flows; OpenFlow as a datapath programming standard

http://zoo.cs.yale.edu/classes/cs434/

Geng Li

01/23/2017
CS434/534: Topics in Networked (Networking) Systems

Basic Network Workflows; OpenFlow as a Datapath Programming Standard

Geng Li
Computer Science Department
Yale University
205 Watson
Email: geng.li@yale.edu

http://zoo.cs.yale.edu/classes/cs434/
CS434/534: Topics in Networked (Networking) Systems

High-Level Language for Programmable Networks

http://zoo.cs.yale.edu/classes/cs434/

Y. Richard Yang

01/25/2017
Outline

- What is the data structure used in current systems?
- How is the data structure programmed currently?
- SDN and OpenFlow:
  - abstraction and extension of current data structures
  - a new way to program it
- How can the more general OF model be implemented efficiently?
Background: Current Model

- What happens when you visit mail.google.com
DNS: Domain Name System

Translates domain names to the numerical IP addresses

- DNS cache in Web browser
  - chrome://net-internals/#dns

- DNS cache in hosts file or the operating system
  - hosts: %systemroot%\system32\drivers\etc (Windows)
  - hosts: /etc/hosts (Linux)
  - pconfig /displaydns (Windows)

- DNS servers
Domain Name Space

- Query servers

- Root zone
  - .org zone
  - .com zone
  - ... (not fully visible)
  - .cn zone
  - others.com zone
  - google.com zone
  - others.google.com
  - mail.google.com
After getting IP address

- TCP connection
  - Transport layer (4)
- HTTP access
  - Application layer (7)
Datapath: Example 1 (same network): A→B

- Look up dest address in routing table
  - find dest is on same net
- Hand datagram to link layer to send inside a link-layer frame
Datapath: Example 2 (Different Networks): A→E

- Look up dst address in routing table
  - routing table: next hop router to dest is 223.1.1.4

- Hand datagram to link layer to send to router 223.1.1.4 inside a link-layer frame
Look Inside a Router

Two key router functions:
- run routing algorithms/protocol (RIP, OSPF, BGP)
- switching datagrams from incoming to outgoing ports
Input Port Functions

1. **Physical Layer**: bit-level reception
2. **Data Link Layer**: e.g., Ethernet
3. **Network Layer**: lookup output port using forwarding table

- **Line Termination**
- **Data Link Processing (protocol, decapsulation)**
- **Lookup, Forwarding Queueing**

Switch Fabric
Output Ports

- **Buffering** required when datagrams arrive from fabric faster than the transmission rate
- **Queueing** (delay) and loss due to output port buffer overflow!
- **Scheduling and queue/buffer management** choose among queued datagrams for transmission
Datapath: Example 2 (Different Networks): A→E

- Look up dest address in router’s forwarding table
- E on same network as router’s interface 223.1.2.9
- Link layer sends datagram to 223.1.2.3 inside link-layer frame via interface 223.1.2.9
Link Layer Services

- **Framing**
  - encapsulate datagram into frame, adding header, trailer and error detection/correction

- **Multiplexing/demultiplexing**
  - frame headers to identify src, dest

- **Media access control**

- **Forwarding/switching with a link-layer (Layer 2) domain**
  - in most link-layer, each adapter has a unique link layer address (also called MAC address)

- **Reliable delivery between adjacent nodes**
  - we learned how to do this already!
  - seldom used on low bit error link (fiber, some twisted pair)
  - common for wireless links: high error rates
Comparison of IP Address and MAC Address

- IP address is locator
  - address depends on network to which an interface is attached
  - introduces features for routing scalability
- IP address needs to be globally unique (if no NAT)

- MAC address is an identifier
  - dedicated to a device
  - flat

- MAC address does not need to be globally unique, but the current assignment ensures uniqueness
ARP: Address Resolution Protocol

- ARP Table: IP/MAC address mappings
- ARP is "plug-and-play":
  - nodes create their ARP tables without intervention from net administrator
- A broadcast protocol:
  - source broadcasts query frame, containing queried IP address
    - all machines on LAN receive ARP query
  - destination D receives ARP frame, replies
    - frame sent to A’s MAC address (unicast)
Recall Earlier Routing Discussion

Starting at A, given IP datagram addressed to E:

- look up net. address of E, find C
- link layer sends datagram to C inside link-layer frame; the dest. address should be C’s MAC address
Router vs. Switch

Layer 3 routing: Match on IP Prefix
Layer 2 switching: Match on MAC
Outline

- What are the data structure used in current systems?
Various of tables
  Fast-forwarding table
    • 5-tuple to identify a flow (source IP address/port number, destination IP address/port number and the protocol)
    • ...

Look up
Forward, switch, route...
Outline

- What is the data structure used in current systems?
- How is the data structure programmed currently?
How the tables are computed?

Routing algorithms/protocols

- Distance vector protocols
  - RIP...
- Link state protocols
  - OSPF...

Distance Vector

Link State
Distributed Computing

- Distributed computing is hard, e.g.,
  - FLP Impossibility Theorem
  - Arrow’s Impossibility Theorem

- Neighbors
- Network changes
- Interact with each other
  - By relay
  - Share local information
An Evolution View of Intradomain Routing Toward SDN
Outline

- What is the data structure used in current systems?
- How is the data structure programmed currently?
- SDN and OpenFlow:
  - abstraction and extension of current data structures
  - a new way to program it
**Software-Defined Networking (SDN)**

- Directly programmable
- Agile
- Centrally managed
- Programmatically configured
- Open standards-based and vendor-neutral

https://www.opennetworking.org/sdn-resources/sdn-definition
SDN: Separation of data and control planes

Traditional

SDN

Control
standard control protocol

Datapath
SDN: Programmable Network

- Easy to generate, add, modify and remove the table in hardware
- Now just defining a centralized control function
  - Configuration = Function(view)
What is OpenFlow?

- The first standard communications protocol defined between controller and switch.
How does it work? - Matching and Action

- Controller installs packet-forwarding rules
- Datapath performs forwarding
- Packet coming
- Matching
- Action
OpenFlow: Flow table

- contains a set of flow entries to apply to matching packets
**OpenFlow: Flow entry/rule**

- **match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table.
- **priority**: matching precedence of the flow entry.
- **counters**: updated when packets are matched.
- **instructions**: to modify the action set or pipeline processing.
- **timeouts**: maximum amount of time or idle time before flow is expired by the switch.
- **cookie**: opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.
- **flags**: flags alter the way flow entries are managed, for example the flag `OFPFF_SEND_FLOW_REM` triggers flow removed messages for that flow entry.
## OpenFlow: Match Fields

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Port</td>
<td>VLAN ID</td>
<td>VLAN pcp</td>
</tr>
<tr>
<td></td>
<td>MAC src</td>
<td>MAC dst</td>
</tr>
<tr>
<td></td>
<td>Eth type</td>
<td>IP Src</td>
</tr>
<tr>
<td></td>
<td>IP Dst</td>
<td>IP ToS</td>
</tr>
<tr>
<td></td>
<td>IP Prot</td>
<td>L4 sport</td>
</tr>
<tr>
<td></td>
<td>L4 dport</td>
<td></td>
</tr>
</tbody>
</table>

+ mask what fields to match

Source: Scott Shenker, UC Berkeley
## 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>

Source: Scott Shenker, UC Berkeley
### Examples

#### 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>5.6.7.8</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>*</td>
<td>*</td>
<td>00:1f..</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>vlan1</td>
<td>*</td>
<td>port6, port7, port9</td>
</tr>
</tbody>
</table>
OpenFlow: Flow entry/rule

- "Open" is real; "Flow" is fake

- **Flow**
  - are broadly defined
  - are limited only by the capabilities of the particular implementation of the Flow Table
OpenFlow: Action

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Action</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet + byte counters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>VLAN ID</th>
<th>VLAN pcp</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP ToS</th>
<th>IP Prot</th>
<th>L4 sport</th>
<th>L4 dport</th>
</tr>
</thead>
</table>

Source: Scott Shenker, UC Berkeley
OpenFlow: Table-miss

No match is found???

- A table-miss flow entry to process table misses
- May send packets to the controller, drop packets or direct packets to a subsequent table.
### OpenFlow: Flow entry/rule

<table>
<thead>
<tr>
<th>Reactive</th>
<th>Proactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>• First packet of flow triggers controller to insert flow entries</td>
<td>• Controller pre-populates flow table in switch</td>
</tr>
<tr>
<td>• Efficient use of flow table</td>
<td>• Zero additional flow setup time</td>
</tr>
<tr>
<td>• Every flow incurs small additional flow setup time</td>
<td>• Loss of control connection does not disrupt traffic</td>
</tr>
<tr>
<td>• If control connection lost, switch has limited utility</td>
<td>• Essentially requires aggregated (wildcard) rules</td>
</tr>
</tbody>
</table>
OpenFlow: Group table

- Enables additional methods of forwarding
  - Advanced
  - But required
OpenFlow: Group table

- A group table consists of group entries
- A group entry may consist of zero or more buckets
- A bucket typically contains actions that modify the packet and an output action that forwards it to a port
OpenFlow: Group table

- There are 4 group types
  - All *(Required)*
OpenFlow: Group table

- There are 4 group types
  - All (Required)
  - Select (Optional)
OpenFlow: Group table

- There are 4 group types
  - All (*Required*)
  - Select (*Optional*)
  - Fast failover (*Optional*)
OpenFlow: Group table

- There are 4 group types
  - All (Required)
  - Select (Optional)
  - Fast failover (Optional)
  - Indirect (Required)

<table>
<thead>
<tr>
<th>Flow</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow1</td>
<td>Group1</td>
</tr>
<tr>
<td>Flow2</td>
<td>Group2</td>
</tr>
<tr>
<td>Flow3</td>
<td>Group1</td>
</tr>
<tr>
<td>Flow4</td>
<td>Group1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Forward to port 2</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>
OpenFlow: Meter Table

- Enables OpenFlow to implement rate-limiting
- Each meter may have one or more meter bands.
- The bands define the behavior of the meters on packets for various ranges rate.
OpenFlow: Multiple Flow Tables

- **Pipeline**
  - Matching starts at the first flow table
  - may continue to additional flow tables

- **Why?**
OpenFlow: Multiple Flow Tables

Example: Cross product

One Table Design

<table>
<thead>
<tr>
<th>ethSrc</th>
<th>ethDst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>a₁</td>
<td>p₁</td>
</tr>
<tr>
<td>a₁</td>
<td>a₂</td>
<td>p₂</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>aₙ</td>
<td>aₙ</td>
<td>pₙ²</td>
</tr>
</tbody>
</table>

n² entries
**OpenFlow: Multiple Flow Tables**

- **Example: Cross product**

Table 1

<table>
<thead>
<tr>
<th>ethSrc</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>reg(_\text{srcCond})=(y_1) jump 2</td>
</tr>
<tr>
<td>a₂</td>
<td>reg(_\text{srcCond})=(y_2) jump 2</td>
</tr>
<tr>
<td>..</td>
<td>...</td>
</tr>
<tr>
<td>aₙ</td>
<td>reg(_\text{srcCond})=(y_k) jump 2</td>
</tr>
<tr>
<td>otherwise</td>
<td>drop</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>reg(_\text{srcSw})</th>
<th>ethDst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_1)</td>
<td>(a_1)</td>
<td>(p_{1,1})</td>
</tr>
<tr>
<td>(y_1)</td>
<td>(a_2)</td>
<td>(p_{1,2})</td>
</tr>
<tr>
<td>..</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(y_k)</td>
<td>(a_n)</td>
<td>(p_{k,n})</td>
</tr>
<tr>
<td>otherwise</td>
<td>drop</td>
<td></td>
</tr>
</tbody>
</table>

\(n + kn\) entries
OpenFlow: Protocol

- OpenFlow channel
  - the interface that connects Switch to Controller

- OpenFlow protocol supports three message types
  - controller-to-switch
  - asynchronous
  - symmetric
OpenFlow in the Real World

- Commercial OpenFlow switch - Physical
- Open vSwitch - Virtual
OpenFlow in the Real World

- Commercial OpenFlow switch - Physical
- Open vSwitch - Virtual
Open vSwitch

- **Overview**
  - follow the same thought and idea of OpenFlow
Linux Bridge Design

- Simple forwarding
- **Matches destination MAC address and forwards**
- Packet never leaves kernel

Source: Dean Pemberton, University of Oregon
Open vSwitch Design

- Decision about how to process packet made in userspace
- First packet of new flow goes to ovs-vsswitchd, following packets hit cached entry in kernel

Source: Dean Pemberton, University of Oregon
ovs-vswitchd in Userspace

- Core component in the system:
  - Communicates with outside world using OpenFlow
  - Communicates with ovsdb-server using OVSDB protocol
  - Communicates with kernel module over netlink
  - Communicates with the system through netdev abstract interface

- Supports multiple independent datapaths (bridges)

- Packet classifier supports efficient flow lookup with wildcards and “explodes” these (possibly) wildcard rules for fast processing by the datapath

- Implements mirroring, bonding, and VLANs through modifications of the same flow table exposed through OpenFlow

- Checks datapath flow counters to handle flow expiration and stats requests

- Tools: ovs-ofctl, ovs-appctl
OVS Kernel Module

- Kernel module that handles switching and tunneling
- Fast cache of non-overlapping flows
- Designed to be fast and simple
  - Packet comes in, if found, associated actions executed and counters updated. Otherwise, sent to userspace
  - Does no flow expiration
  - Knows nothing of OpenFlow
- Implements tunnels
- Tools: ovs-dpctl
Userspace Processing

- Packet received from kernel
- Given to the classifier to look for matching flows accumulates actions
- If “normal” action included, accumulates actions from “normal” processing, such as L2 forwarding and bonding
- Actions accumulated from configured modules, such as mirroring
- Prior to 1.11, an exact match flow is generated with the accumulated actions and pushed down to the kernel module (along with the packet)
Kernel Processing

- Packet arrives and header fields extracted
- Header fields are hashed and used as an index into a set of large hash tables
- If entry found, actions applied to packet and counters are updated
- If entry is not found, packet sent to userspace and miss counter incremented
Mininet

- Machine-local virtual network
  - great dev/testing tool
- Uses linux virtual network features
  - Cheaper than VMs
- Arbitrary topologies, nodes
Mininet

- Rapidly prototype, develop and test
  - Interestingly-sized networks (16-100 nodes) start up in seconds
  - No lengthy lab reconfiguration or rebooting required
  - Always-accessible network resources, in any topology, at essentially no cost
  - Designs that work on Mininet transfer seamlessly to hardware for full speed operation
Mininet

- Repeatably test, analyze, and predict network behavior
  - Easy replication of experimental and test results
  - Examine effects of code or network changes before testing/deploying on hardware
  - Allows automated system-level tests and experiments
  - Recreate real-world network and test cases for a variety of topologies and configurations
Mininet

- Quickly get up and running
  - Free and permissively licensed (BSD)
  - Minimal hardware requirements
  - Accessible to novices thanks to simple CLI
  - Smooth learning curve thanks to walkthrough, tutorial, examples and API documentation
  - Strong users and support community
Mininet

- Download: http://mininet.org/download/

- Tutorial: https://github.com/mininet/openflow-tutorial/wiki
Some Commands

- sudo mn --topo single,3 --mac --switch ovsk --controller remote
- sh ovs-ofctl dump-flows s1
- sh ovs-ofctl add-flow s1 in_port=1,actions=output:2
- sh ovs-ofctl add-flow s1 in_port=2,actions=output:1
- sh ovs-ofctl del-flows s1
- sh ovs-ofctl add-flow s1 "priority=0,action=normal" sh ovs-ofctl add-flow s1 "priority=100,eth_type=0x800,ip_dst=10.0.0.1,action=drop"
- sh ovs-ofctl add-flow s1 "priority=100,eth_type=0x806,dl_dst=00:00:00:00:00:02,action=drop"
Mininet

- Basic commands:
  - Display an xterm for switch s1
    - mininet> xterm s1
  - Inspect flow tables at switch xterm
    - dpctl dump-flows tcp:127.0.0.1:6634

- To view OpenFlow protocol messages, at mininet-VM xterm:
  - sudo wireshark &
  - Capture the interface to controller
  - In wireshark filter box, enter filter to filter OpenFlow messages: of
Mininet

Basic commands:

- Create a network consists of one OpenvSwitch, three hosts and is controlled by a remote controller with IP address 192.168.56.1
  - `sudo mn --topo single,3 --controller remote,ip=192.168.56.1 --switch ovsk`

- `mininet> help`
- `mininet> dump nodes`
- `mininet> h1 ping h2`
Outline

- What is the data structure used in current systems?
- How is the data structure programmed currently?
- SDN and OpenFlow:
  - abstraction and extension of current data structures
  - a new way to program it
- How can the more general OF model be implemented efficiently?
Pipeline Specialization

- Divide a single table into a pipeline, with specialization of types
  - Exact match $\gg$ lpm $\gg$ ternanry

OpenFlow building blocks

Monitoring/debugging tools
- oflops
- ndb

Applications
- Firewall
- Traffic Engineering
- Load Balancing
- Mobility

Controller
- Frenetic
- Floodlight
- OpenDayLight
- ONOS
- Ryu
- POX

OpenFlow

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

Software switches and experimental platforms
- NetFPGA
- Broadcom Ref. Switch
- OpenWRT
- OpenVSwitch

OpenFlow Switches