CS434/534: Topics in Network Systems

High-Level Programming for Programmable Networks

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

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

- Admin and recap
- SND Datapath model
  - Basic datapath model: flow tables
  - Improved datapath model: flow table pipelines
- SDN programming
  - Higher-level logic driven user programming
L3: Example 1 (same network): A→B

- Look up dest address in routing table
  - find dest is on same net (what is the matching alg?)
- Hand datagram to L2 to send inside a link-layer frame
ARP Protocol

- **Issue:** Need to convert L3 address to L2 address

- **Current solution:**
  - **Data structure:** each host has an ARP table mapping IP to L2
    - `arp -a`
    - IP -> MAC
  - **Protocol:** a broadcast protocol
    - Client broadcasts query frame, containing queried IP address
    - Node with queried IP receives ARP frame, replies its MAC address
ARP Format and Features

- **Query**: Layer 2 (Link layer) broadcast: destination ff:ff:ff:ff:ff:ff to be received by all hosts at the same local network
- **Response**: Host with the MAC returns its MAC if it has the query IP
- **Gratuitous ARP**: A host sends this message to update other devices if it changes MAC
**A -> B in Layer 2**

After A finds the link-layer (MAC) address of B, it constructs the L2 frame:

- A’s MAC addr
- B’s MAC addr
- A’s IP addr
- B’s IP addr
- IP payload
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
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
## Recap: Basic Network Device Data Structure and Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Data Structure</th>
<th>Function</th>
<th>How currently populated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 basic</td>
<td>switch table</td>
<td>MAC ( \rightarrow ) ports</td>
<td>flooding +self learning +spanning tree protocol</td>
</tr>
<tr>
<td>L2 VLAN</td>
<td>switch table</td>
<td>MAC+VLAN ( \rightarrow ) ports</td>
<td>Flooding +selflearning +config+spanning tree</td>
</tr>
<tr>
<td>L3 lookup</td>
<td>routing table (forwarding information base)</td>
<td>IP ( \rightarrow ) next hop router IP; longest prefix match</td>
<td>distributed routing protocols; command line</td>
</tr>
<tr>
<td>ARP</td>
<td>arp table</td>
<td>IP ( \rightarrow ) MAC</td>
<td>ARP protocol</td>
</tr>
<tr>
<td>NAT/LB</td>
<td>NAT/LB table</td>
<td>srcIP, srcPort, dstIP, dstPort ( \rightarrow ) srcIP, srcPort, dstIP, dstPort</td>
<td>NAT/LB controller through config</td>
</tr>
</tbody>
</table>
Exercise: Draw a Big Picture of Internet using (L2/L3) Tables
Exercise: Visualize a Real Network using Tables

Internet

Tier-1
- H1
- H2

Tier-2
- H3
- H4

Tier-3
- H5
- H6

CE

F1 (Firewall)

LB1 (Load balancer)

IPS1 (Intrusion prevention)

S1

S2

LB2

F2

S3

S4

S5

S6

R1

IPS3

IPS2

IPS3

Logger
Exercise: Visualize a Real Network using Tables
Basic ideas

- Multiple protocols
- Distributed protocols
- Vendor manual configuration
Recap: OpenFlow

<table>
<thead>
<tr>
<th>Match Fields</th>
<th>Priority</th>
<th>Counters</th>
<th>Instructions</th>
<th>Timeouts</th>
<th>Cookie</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Port</td>
<td>VLAN ID</td>
<td>VLAN pcp</td>
<td>MAC src</td>
<td>MAC dst</td>
<td>Eth type</td>
<td>IP Src</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP Dst</td>
<td>IP ToS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP Prot</td>
<td>L4 sport</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>L4 dport</td>
</tr>
</tbody>
</table>

+ mask what fields to match

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!
Example OF Test 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"
SDN Example: Google WAN

- 52 edge switches interconnecting google data centers
- ~143,000 OF rules
Exercise: What does this switch do?

<table>
<thead>
<tr>
<th>Priority</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>tcp_dst:22</td>
<td>drop</td>
</tr>
<tr>
<td>20</td>
<td>ip_dst: 101.22.0.0/16</td>
<td>port 2</td>
</tr>
<tr>
<td>1</td>
<td>in_port:1, mac_dst: 0xffffffff</td>
<td>ports 2,3,4</td>
</tr>
</tbody>
</table>
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.
Is the flow table computation model complete?

- Realize any function:
  (headers) -> output_ports

- Realize any function (mapping+transformation):
  (headers) -> [new headers, output_ports]
Flow Table Efficiency

Policy:
srcType = lookup ethSrc in hostTypeTable;
dstType = lookup ethDst in hostTypeTable;
Compute routing according to srcType, dstType

- Assume n hosts in hostTypeTable;
- Assume k types of hosts

<table>
<thead>
<tr>
<th>MAC</th>
<th>hostType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac1</td>
<td>Linux v1</td>
</tr>
<tr>
<td>Mac2</td>
<td>Linux v2</td>
</tr>
<tr>
<td>Mac3</td>
<td>IOS12</td>
</tr>
<tr>
<td>Mac4</td>
<td>IOS11</td>
</tr>
<tr>
<td>Mac5</td>
<td>Win10</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Flow Table Efficiency

<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
Outline

- Admin and recap
- SDN datapath model
  - Basic datapath model: flow tables
  - Improved datapath model: flow table pipelines
## Multi-Table (2 Tables) Design

### Table 1

<table>
<thead>
<tr>
<th>ethSrc</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$\text{reg}_{\text{srcType}} = y_1$ jump 2</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\text{reg}_{\text{srcType}} = y_2$ jump 2</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$\text{reg}_{\text{srcType}} = y_n$ jump 2</td>
</tr>
<tr>
<td>otherwise</td>
<td>drop</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>$\text{reg}_{\text{srcType}}$</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>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</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
# Multi-Table (3 Tables) Design

## Table 1

<table>
<thead>
<tr>
<th>ethSrc</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$\text{reg}_{\text{srcType}} = y_1$ jump 2</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\text{reg}_{\text{srcType}} = y_2$ jump 2</td>
</tr>
<tr>
<td>..</td>
<td>...</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$\text{reg}_{\text{srcType}} = y_n$ jump 2</td>
</tr>
<tr>
<td>otherwise</td>
<td>drop</td>
</tr>
</tbody>
</table>

## Table 2

<table>
<thead>
<tr>
<th>ethDst</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$\text{reg}_{\text{dstType}} = y_1$ jump 3</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\text{reg}_{\text{dstType}} = y_2$ jump 3</td>
</tr>
<tr>
<td>..</td>
<td>...</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$\text{reg}_{\text{dstType}} = y_n$ jump 3</td>
</tr>
<tr>
<td>otherwise</td>
<td>drop</td>
</tr>
</tbody>
</table>

## Table 3

<table>
<thead>
<tr>
<th>$\text{reg}_{\text{srcType}}$</th>
<th>$\text{reg}_{\text{dstType}}$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$y_1$</td>
<td>$p_{1,1}$</td>
</tr>
<tr>
<td>$y_1$</td>
<td>$y_2$</td>
<td>$p_{1,2}$</td>
</tr>
<tr>
<td>..</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$y_k$</td>
<td>$y_n$</td>
<td>$p_{k,k}$</td>
</tr>
<tr>
<td>otherwise</td>
<td></td>
<td>drop</td>
</tr>
</tbody>
</table>

2n + $k^2$ entries
**Comparison of 3 Designs**

Assume $n = 4000$, $k = 100$

<table>
<thead>
<tr>
<th>Design</th>
<th>#flow rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 table</td>
<td>$16,000,000 = 16M$</td>
</tr>
<tr>
<td>2 tables</td>
<td>$4000+400,000 = 404K$</td>
</tr>
<tr>
<td>3 tables</td>
<td>$8000+10,000 = 18K$</td>
</tr>
</tbody>
</table>
Example Pipeline: OF-DPA

Figure 1.2. OF-DPA Pipeline

https://www.broadcom.com/products/ethernet-connectivity/software/of-dpa#documentation
Outline

- Admin and recap
- SDN datapath model
  - Basic datapath model: flow tables
  - Improved datapath model: flow table pipelines
- SDN programming
Discussion

- Components of a programming environment to control programmable switches (switches with table pipelines)?
Extremely High-Level Programmable Network Arch

logically centralized data store

Network View

Service/Policy

NE Datapath

NE Datapath

Program
Two Basic Types of Programming

Reactive

- First packet of flow triggers controller to insert flow entries

Pros

- Efficient use of flow table

Cons

- Every flow incurs additional flow setup time
- If control connection lost, switch has limited utility

Proactive

- Controller pre-populates flow table in switch

Pros

- Zero additional flow setup time
- Loss of control connection does not disrupt traffic

Cons

- Need large flow table
- Many entries maybe cold
Outline

- Admin and recap
- Datapath model
  - Basic datapath model: flow tables
  - Improved datapath model: flow table pipelines
- SDN programming
  - Reactive, higher-level logic driven user programming (i.e., user generates all the flow rules)
An Example Higher-Level Logic

badPort = 22  // policy
hostTbl = {A:1,B:2,
           C:3,D:4}  // net view

def onPacketIn(p):
    if badPort == p.tcp_dst:
        drop
    else:
        forward([hostTbl(p.eth_dst)])

Logic:
- Assume only packets to A,B,C,D can appear
- Block traffic to SSH (port 22)
Higher-Level Logic with Datapath Generation

hostTbl = {A:1,B:2,C:3,D:4}

def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop

    installRule({'match':{'tcp_dst':22},
                'action':[[]]})

else:
    forward([hostTbl(p.eth_dst)])

    installRule({'match': {'eth_dst':p.eth_dst,
                     'tcp_dst':p.eth_dst,
                     'action':[hostTbl(p.eth_dst)]})

match does not support logic negation
Higher-Level Logic with Datapath Generation

hostTbl = {A:1,B:2,C:3,D:4}

def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop

    installRule({'priority':1,
                 'match': {'tcp_dst':22},
                 'action':[]})

    else:
        forward([hostTbl(p.eth_dst)])

    installRule({'priority':0,
                 'match': {'eth_dst':p.eth_dst},
                 'action':[hostTbl(p.eth_dst)]})
def onPacketIn(p):
    if 22 == p.tcp_dst:
        // drop
        installRule({'priority':1,'match':{'tcp_dst':22},'action':[]})
    else:
        installRule({'priority':0,'match':{'eth_dst':p.eth_dst},
                     'action':[hostTbl(p.eth_dst)]})
        // forward([hostTbl(p.eth_dst)])