

Virtual Labs on SDN and P4 Programmable Switches

Jorge Crichigno, Elie Kfoury, Jose Gomez, Ali AlSabeih
University of South Carolina

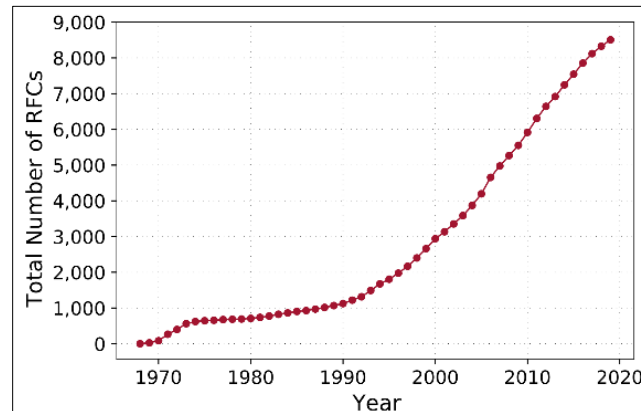
2022 Winter ICT Educators Conference
January 6-7, 2021
Online

Agenda

- Motivation
- Software-Defined Networking (SDN) motivation
- Lab environment
- SDN lab series
- P4 motivation
- P4 lab series

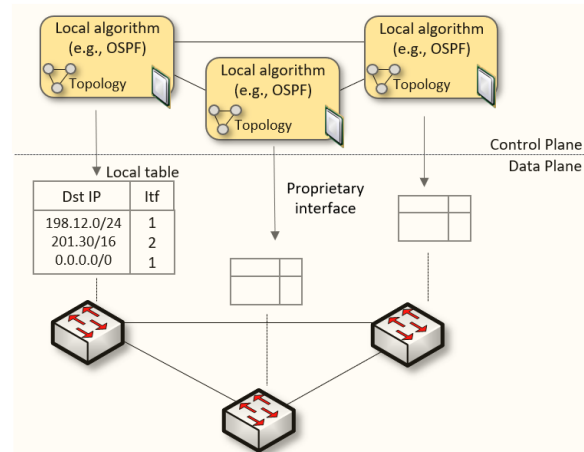
Motivation

- Since the explosive growth of the Internet in the 1990s, the networking industry has been dominated by closed and proprietary hardware and software
- There has been a lack of flexibility to design protocols
 - Standardized requirements cannot be easily removed to enable changes, leading to a *protocol ossification*



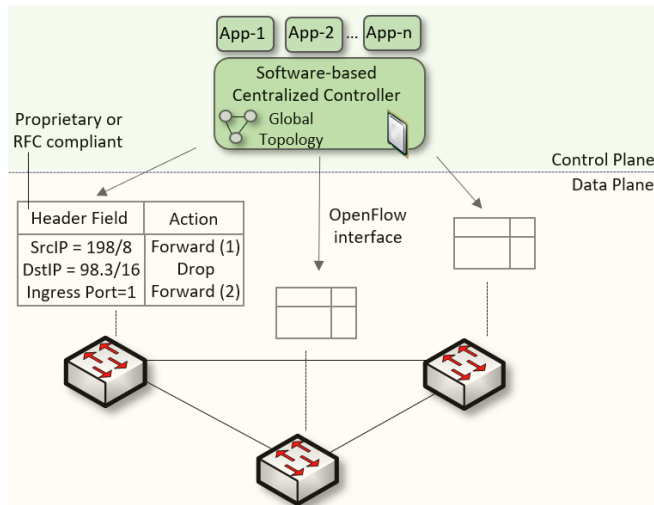
Traditional (Legacy) Networking

- The interface between the control plane and data plane has been historically proprietary
- A router is a monolithic unit built and internally accessed by the manufacturer only
- There is a vendor dependence: slow product cycles of vendor equipment, standardization, no room for innovation from network owners



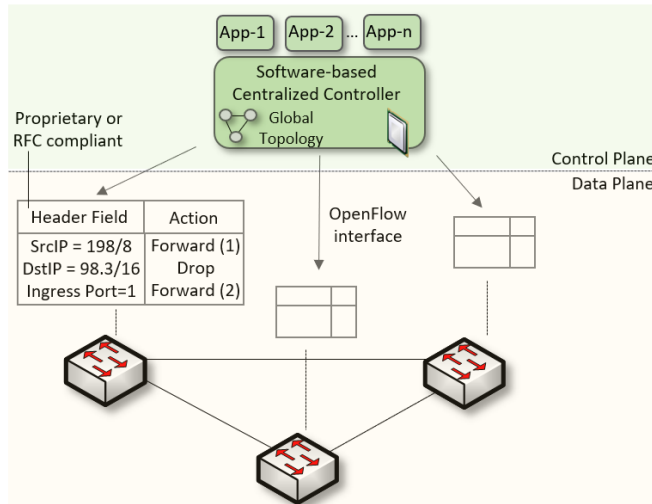
SDN

- Protocol ossification has been challenged first by SDN
- SDN explicitly separates the control and data planes, and implements the control plane intelligence as a software outside the switches



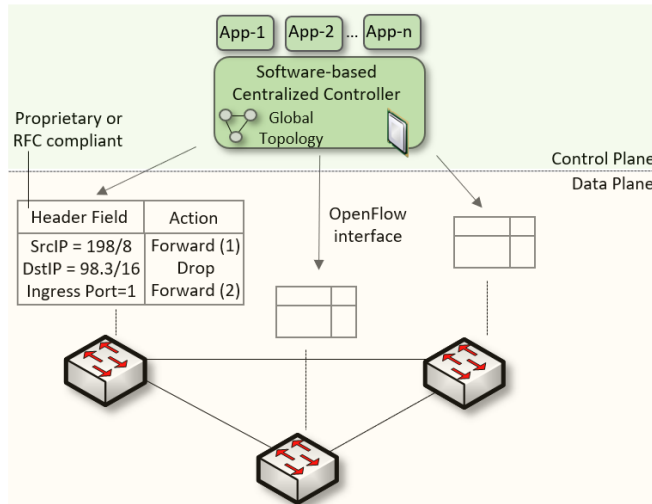
SDN

- The function of populating the forwarding table is now performed by the controller
- The controller is responsible for programming packet-matching and forwarding rules



SDN

- SDN also provides a framework for a more general way to forward packets
 - “**match plus action**” abstraction: match bits in arriving packet header(s) in any layers, then take action
 - local actions: drop, forward, modify, or send matched packet to controller
 - Possibility of experimentation and innovation (custom policies, apps can be deployed)
 - Packets can be forwarded based on other fields, such as TCP port number



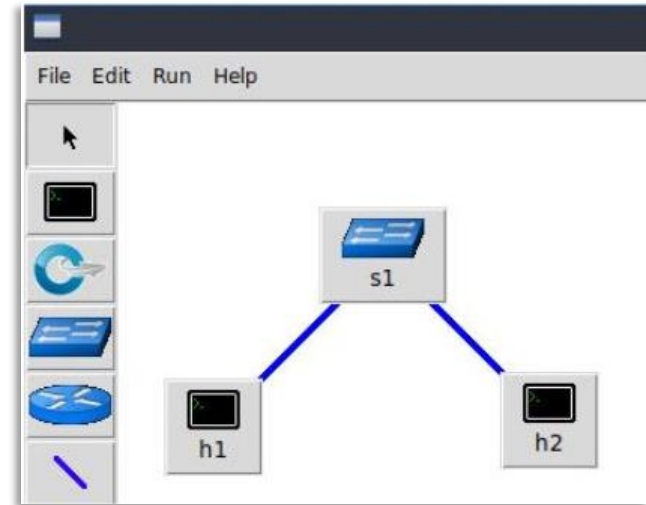
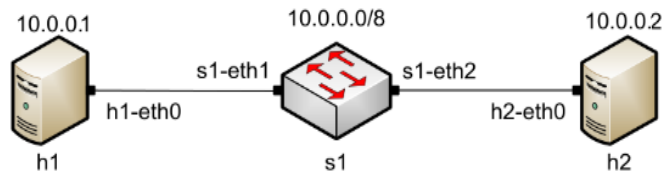
Environment: Mininet

Mininet

- Mininet is a virtual testbed for developing and testing network tools and protocols
- Nodes are sometimes called containers, or more accurately, *network namespaces*
- Features
 - Fast prototyping for new protocols
 - Simplified testing for complex topologies without the need of buying expensive hardware
 - It runs real code on Unix/Linux kernels (realistic emulation)
 - Open source
 - Containers consume few resources; complex networks can be created (100s or 1,000s of nodes)

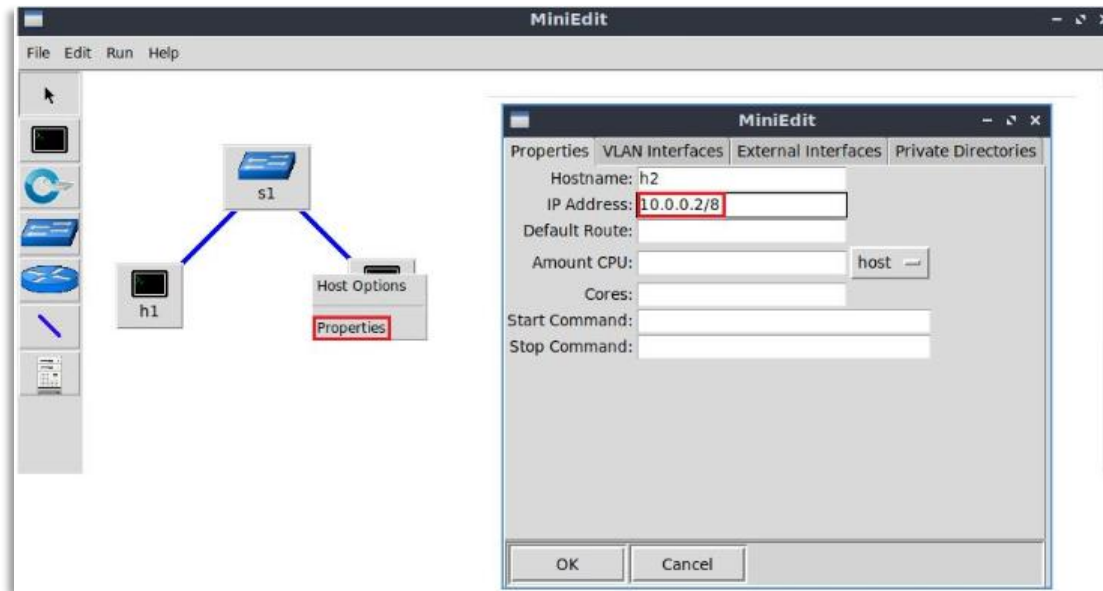
MiniEdit

- To build a topology, we use MiniEdit
- MiniEdit is a simple GUI editor for Mininet
- Example:



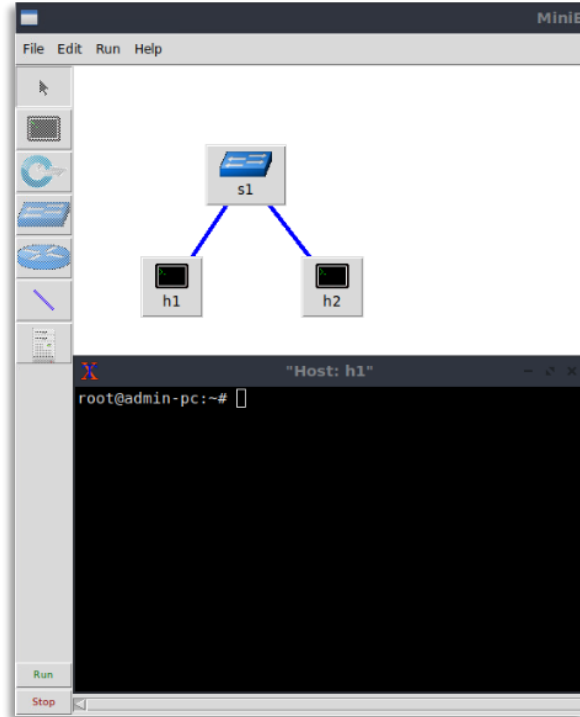
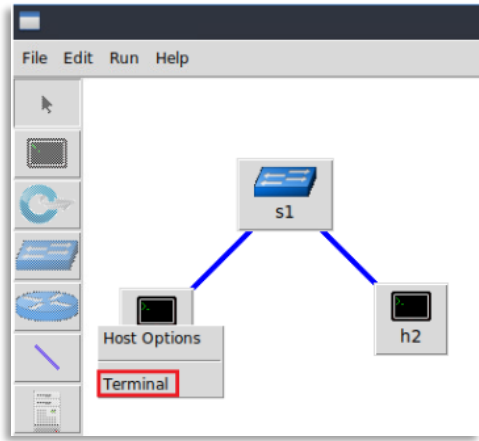
Host Configuration

- Configure the IP addresses at host h1 and host h2
- A host can be configured by holding the right click and selecting properties on the device



Executing Commands on Hosts

- Open a terminal on host by holding the right click and selecting *Terminal*



```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.541 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.042 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 110ms
rtt min/avg/max/mdev = 0.033/0.125/0.541/0.186 ms
root@admin-pc:~#
```

SDN Lab Series

SDN Lab Series

The labs provide learning experiences on essential SDN topics

- Legacy networks, Border Gateway Protocol (BGP)
- MPLS and FRR (an open-source router)
- SDN fundamentals – controllers, switches
 - ONOS controller
 - Open vSwitch (OVS)
- Traffic isolation with VXLAN
- OpenFlow
- Interconnection between SDN and legacy networks

SDN Lab Series

The labs provide learning experiences on essential SDN topics

Lab 1: Introduction to Mininet

Lab 2: Legacy Networks: BGP Example as a distributed system and autonomous forwarding decisions

Lab 3: Early efforts of SDN: MPLS example of a control plane that establishes semi-static forwarding paths

Lab 4: Introduction to SDN

Lab 5: Configuring VXLAN to provide network traffic isolation

Lab 6: Introduction to OpenFlow

Lab 7: SDN-routing within an SDN network

Lab 8: Interconnection between legacy networks and SDN networks

Lab 9: Configuring Virtual Private LAN Services (VPLS) with SDN networks

Lab 10: Applying Equal-Cost Multi-Path (ECMP) within SDN networks

Organization of Lab Manuals

Each lab starts with a section *Overview*

- Objectives
- Lab settings: passwords, device names
- Roadmap: organization of the lab

Section 1

- Background information (theory) of the topic being covered (e.g., fundamentals of SDN)
- Section 1 is optional (i.e., the reader can skip this section and move to lab directions)

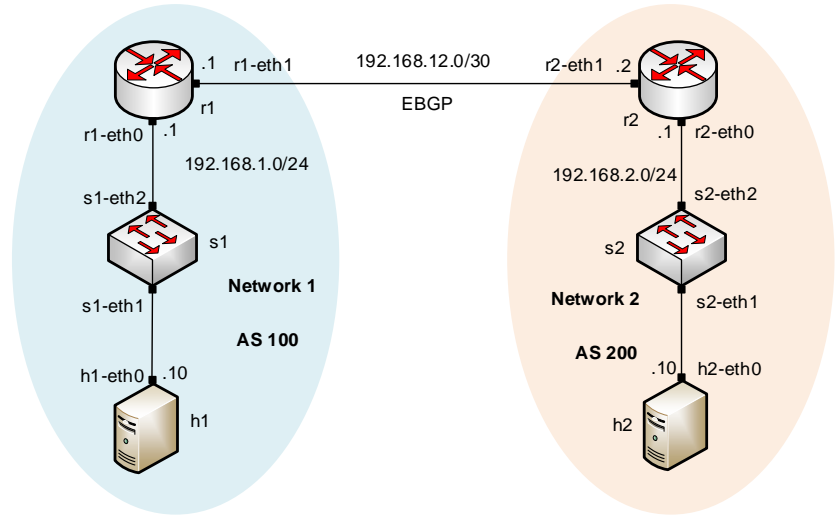
Section 2... n

- Step-by-step directions

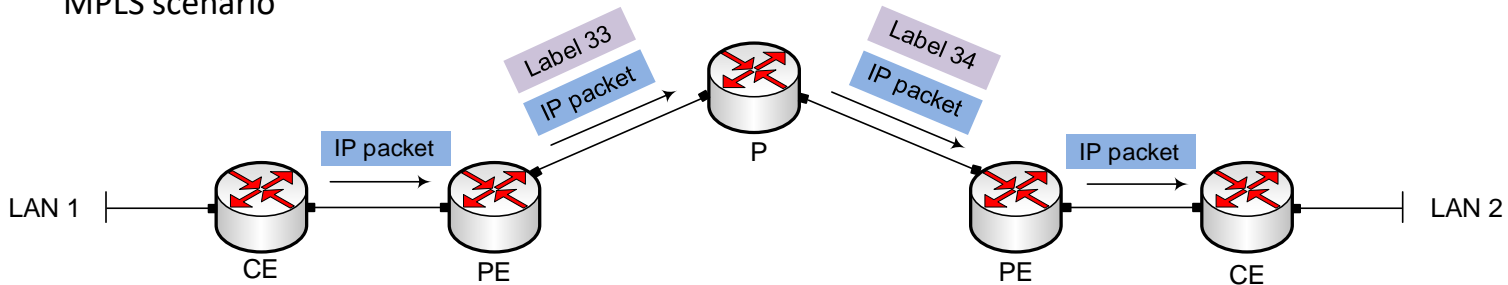
Examples

Legacy networks

BGP scenario

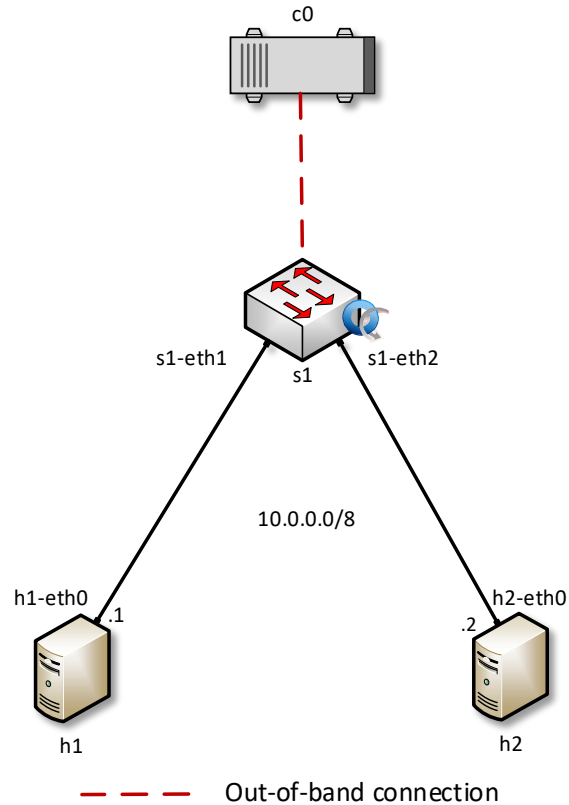


MPLS scenario



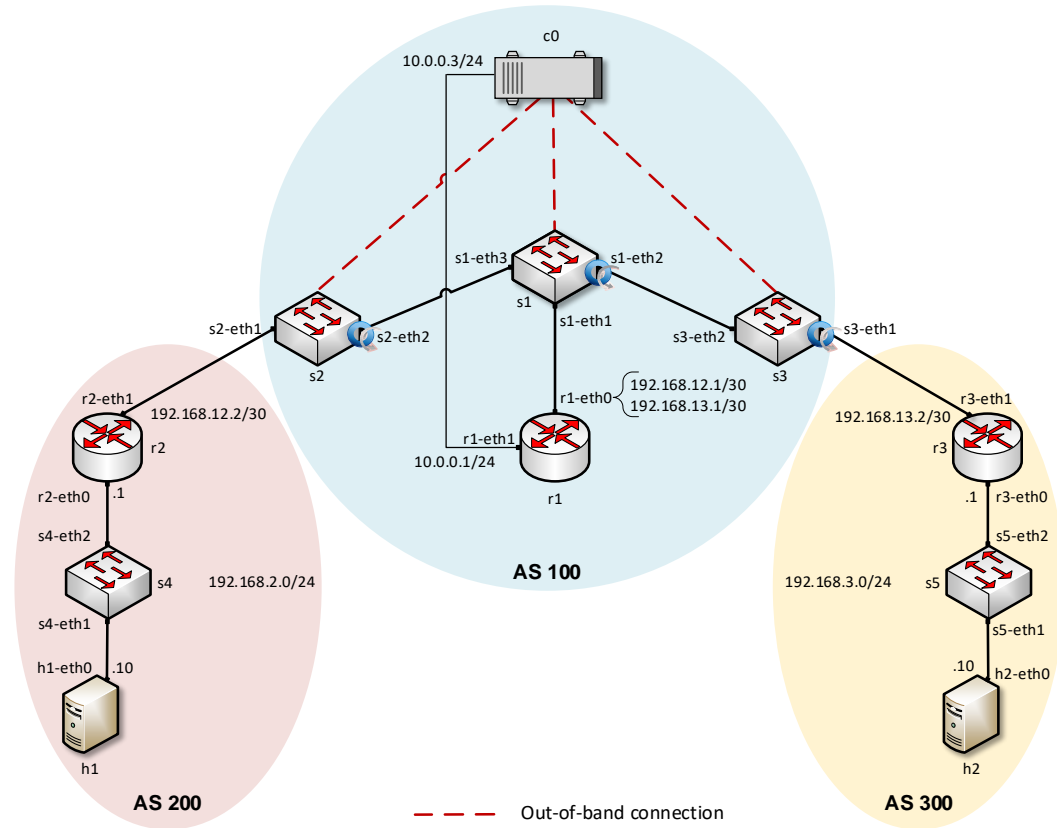
Examples

SDN networks



Examples

Interconnection of SDN and legacy networks



Overview SDN Exercises

SDN Exercises

Exercise set

Exercise 1: SDN Network Configuration

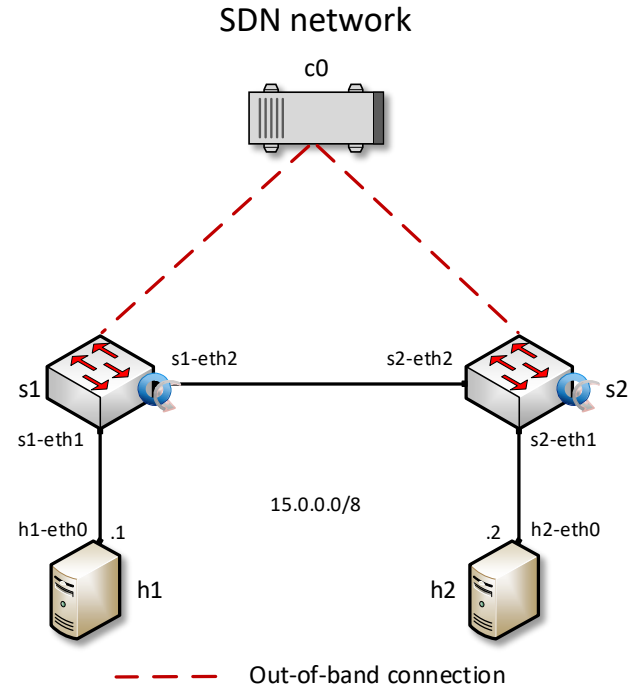
Exercise 2: Configuring VXLAN

Exercise 3: OpenFlow Protocol Management

Exercise 4: Incremental Deployment of SDN Networks within Legacy Networks

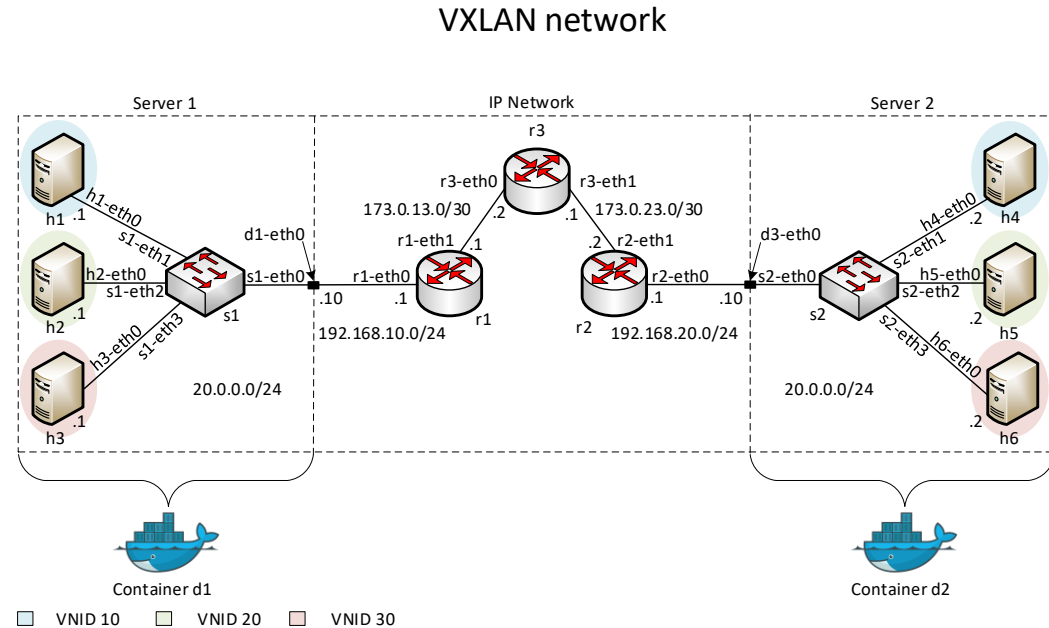
SDN Exercises

- Configure the SDN network
- Manage the OpenFlow switches using the ONOS controller
- Navigate through the ONOS terminal to enable applications, inspect links, devices, flow tables, etc.
- Establish connectivity between the two hosts



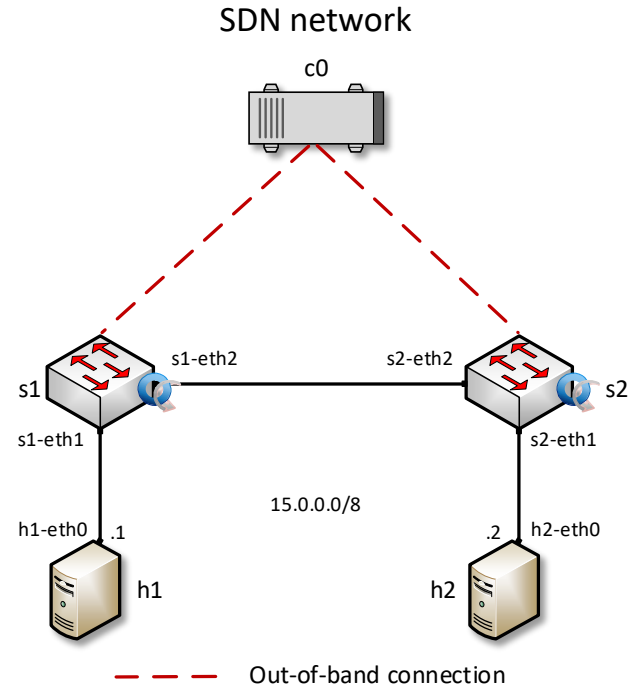
SDN Exercises

- Configure OSPF within the IP network
- Isolate the traffic in each server
- Provide an end-to-end connectivity between hosts with the same VXLAN identifier (VNID)



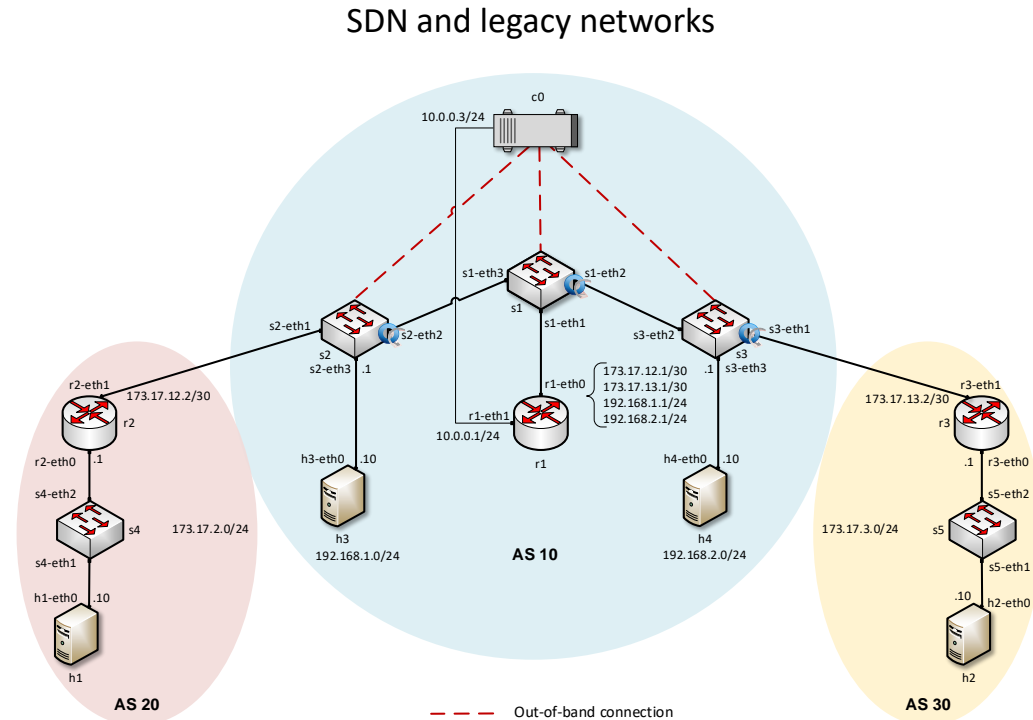
SDN Exercises

- Configure the SDN network
- Manage the switches manually using the OpenFlow protocol
- Manage the switches using the ONOS controller
- Inspect the OpenFlow messages exchanged between the control plane and the data plane
- Inspect the flow rules on the switches that forward traffic between the hosts



SDN Exercises

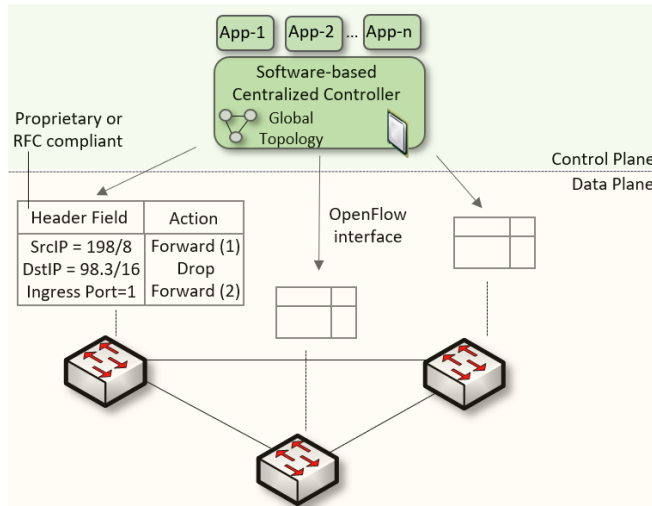
- Configure BGP within the legacy routers
- Configure the SDN switches to interconnect with the legacy networks
- Emulate virtual gateways and routing within the SDN network
- Establish connectivity between hosts in different legacy networks, as well as between hosts within the SDN network



Overview P4 Labs

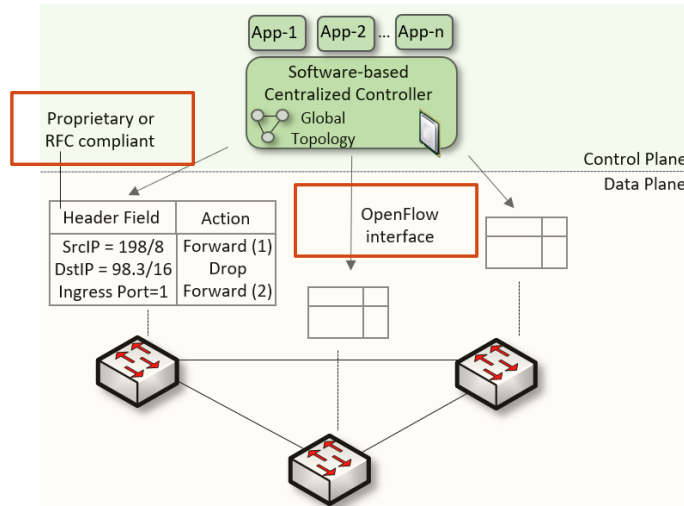
SDN Limitation

- SDN does not allow the programmer to create a new protocol and parse the protocol header in the data plane
 - SDN is limited to the OpenFlow specifications and the fixed-function data plane



SDN Limitation

- SDN does not allow the programmer to create a new protocol and parse the protocol header in the data plane
 - SDN is limited to the OpenFlow specifications and the fixed-function data plane



P4 Programmable Switches

- The programmable forwarding can be viewed as a natural evolution of SDN
- P4 programmable switches permit a programmer to program the data plane
 - Defining and parsing new protocols
 - Customizing packet processing functions
 - Measuring events occurring in the data plane at nanosecond resolution
 - Inspecting and analyzing each packet (per-packet analysis)
- P4 stands for Programming Protocol-independent Packet Processors

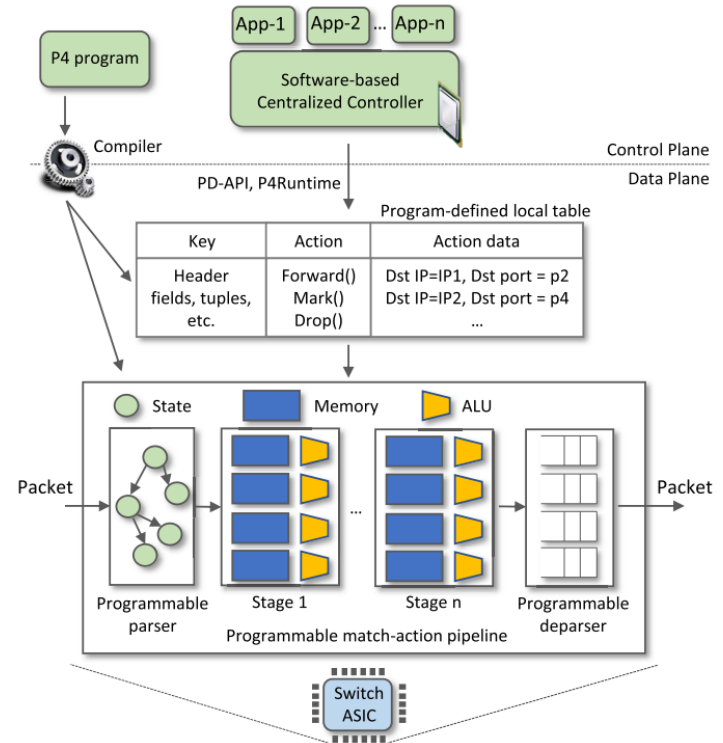
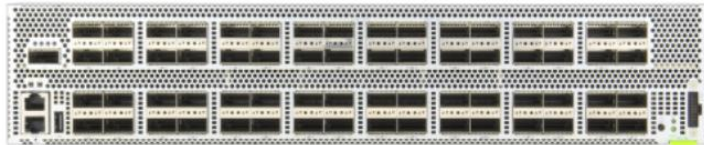
P4 Programmable Switches

- Analogy between networks and other computing domains

| Domain | Year | Processing Unit | Main Language/s |
|-------------------|------|---|-------------------------|
| General computing | 1971 | Central Processing Unit (CPU) | C, Java, Python, etc. |
| Signal processing | 1979 | Digital Signal Processor (DSP) | Matlab |
| Graphics | 1994 | Graphics Processing Unit (GPU) | Open Computing Language |
| Machine learning | 2015 | Tensor Processing Unit (TPU) | Tensor Flow |
| Computer networks | 2016 | Protocol Independent Switch Architecture (PISA) | P4 |

P4 Programmable Switches

- Programmable chip
 - Parser parses header fields, written by the programmer
 - Stages contain memory and Arithmetic Logic Units (ALUs)
 - Memory are used for tables, **match** bits
 - ALUs are simple, suitable for header field operations, **actions**
 - Stages are sequentially arranged (1, 2, ..., n), for sequential computation
 - Deparser assembles packet headers back



Examples of P4 Programmable Switches

- Behavioral Model Version 2 (BMv2)
 - Open source
 - Software switch used for teaching, researching ideas
 - Good to validate ideas
- Commercial physical devices
 - E.g., Edgecore Wedge 100BF-65X (based on Intel's Tofino chip)
 - 65x100G switch ports
 - Used in production networks and research



Introduction to P4 and BMv2 Lab Series

Lab experiments

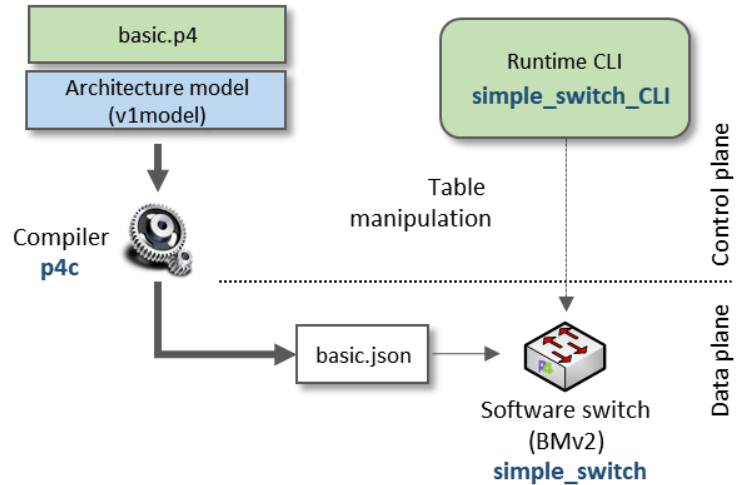
- Lab 1: Introduction to Mininet
- Lab 2: Introduction to P4 and BMv2
- Lab 3: P4 Program Building Blocks
- Lab 4: Parser Implementation
- Lab 5: Introduction to Match-action Tables (Part 1)
- Lab 6: Introduction to Match-action Tables (Part 2)
- Lab 7: Populating and Managing Match-action Tables
- Lab 8: Checksum Recalculation and Packet Deparsing

Exercises

- Exercise 1: Building a Basic Topology
- Exercise 2: Compiling and Testing a P4 Program
- Exercise 3: Parsing UDP and RTP
- Exercise 4: Building a Simplified NAT
- Exercise 5: Configuring Tables at Runtime
- Exercise 6: Building a Packet Reflector

Workflow of a P4 Program

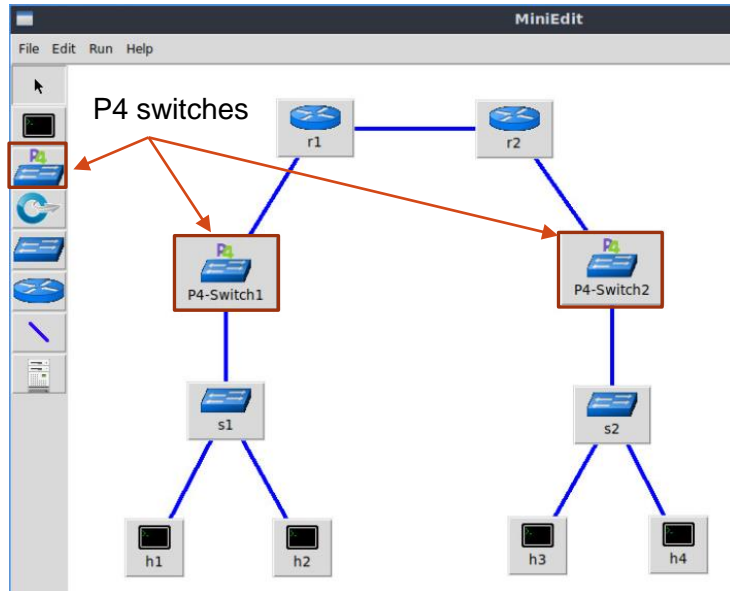
- Workflow used to program the BMv2 switch



Workflow used in the lab series

Development Environment

- Topology constructed with a modified version of the MiniEdit editor
- P4 software switches (BMv2) running inside Docker containers (through Containernet)
- Code written in Visual Studio Code with P4 syntax highlighting and a built-in terminal



The Visual Studio Code interface shows the 'basic.p4' file being edited. The code includes headers and defines types for IPv4 addresses and Ethernet frames. A terminal window at the bottom shows the command prompt 'admin@Lubuntu-vm: ~/P4_Labs/Lab2\$'.

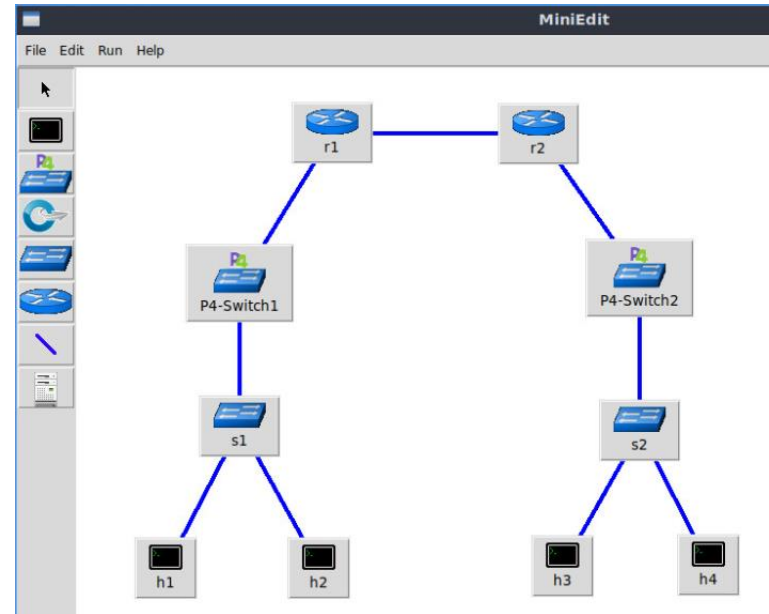
```
1 /* -*- P4_16 -*- */
2 #include <core.p4>
3 #include <v1model.p4>
4
5 const bit<16> TYPE_IPV4 = 0x800;
6
7 /*
8 ***** HEADERS *****
9 */
10
11 typedef bit<9> egressSpec_t;
12 typedef bit<48> macAddr_t;
13 typedef bit<32> ip4Addr_t;
14
15 header ethernet_t {
16     macAddr_t dstAddr;
17     macAddr_t srcAddr;
18     bit<16> etherType;
19 }
```

← Editor

← Terminal

Development Environment

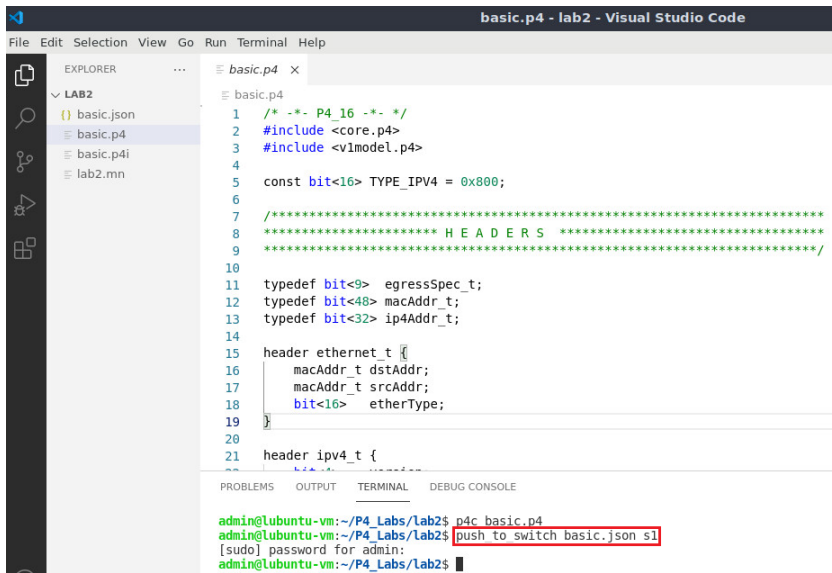
- Programmer has the flexibility of designing complex networks
- P4 programmable switches use BMv2
- Legacy/OpenFlow switches are Open vSwitch (OVS)
- Routers use a real routing stack (FRR)
- Hosts use Linux's network stack



Overview P4 Labs

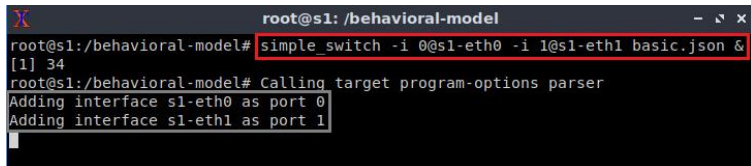
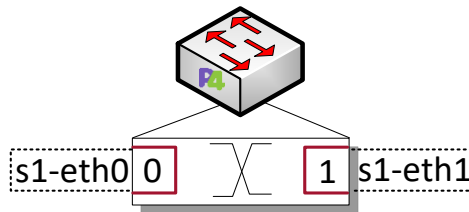
Examples

- Compiling a P4 program and pushing the output to the data plane
- Starting the switch daemon and allocating interfaces



```
basic.p4 - lab2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
LAB2
basic.json
basic.p4
basic.p4i
lab2.mn
basic.p4
1 /* -*- P4_16 -*- */
2 #include <core.p4>
3 #include <v1model.p4>
4
5 const bit<16> TYPE_IPV4 = 0x800;
6
7
8 ***** HEADERS *****
9
10
11 typedef bit<9> egressSpec_t;
12 typedef bit<48> macAddr_t;
13 typedef bit<32> ip4Addr_t;
14
15 header ethernet_t {
16     macAddr_t dstAddr;
17     macAddr_t srcAddr;
18     bit<16> etherType;
19 }
20
21 header ipv4_t {
```

```
admin@lubuntu-vm:~/P4_Labs/Lab2$ p4c basic.p4
admin@lubuntu-vm:~/P4_Labs/Lab2$ push to switch basic.json s1
[sudo] password for admin:
admin@lubuntu-vm:~/P4_Labs/Lab2$
```



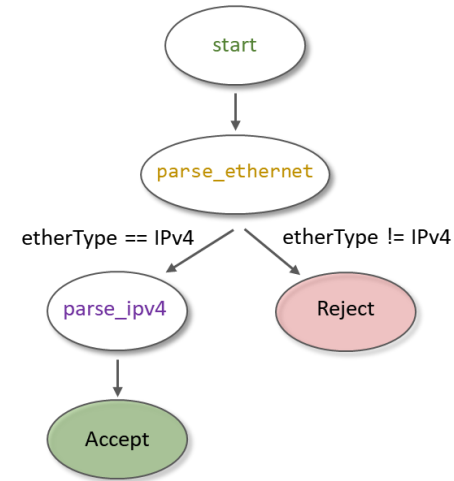
```
root@s1: /behavioral-model
root@s1:/behavioral-model# simple_switch -i 0@s1-eth0 -i 1@s1-eth1 basic.json &
root@s1:/behavioral-model# Calling target program-options parser
Adding interface s1-eth0 as port 0
Adding interface s1-eth1 as port 1
```

Examples

- Defining headers and programming a parser for Ethernet, IPv4, and IPv6

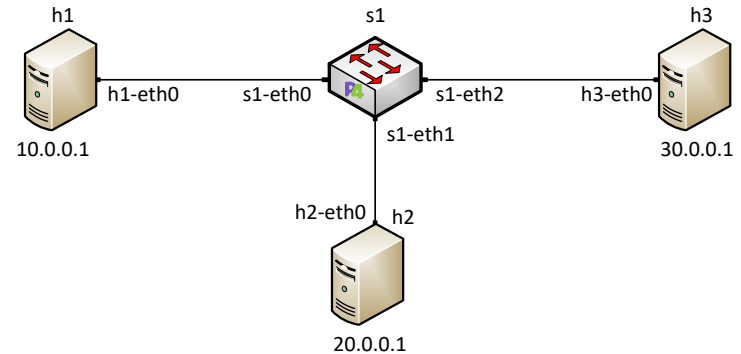
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------------------------|---|---|---|-----|---|---|---|----------|---|----|----|-------|----|----|----|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | Version | | | | IHL | | | | DSCP | | | | ECN | | | | Total Length | | | | | | | | | | | | | | | |
| 32 | Identifier | | | | | | | | | | | | Flags | | | | Fragment Offset | | | | | | | | | | | | | | | |
| 64 | Time To Live | | | | | | | | Protocol | | | | | | | | Header Checksum | | | | | | | | | | | | | | | |
| 96 | Source IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 128 | Destination IP Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 160 | Options (if IHL > 5) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

```
header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totallen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```



Examples

- Programming match-action tables:
 - Exact
 - Longest Prefix Matching (LPM)
- Forwarding using port information:
 - Packets arriving at port 0 are sent through port 1
 - Packets arriving at port 1 are sent through port 0
- Routing using layer-3 information:
 - Matching on the destination IP address
 - Modifying the source and destination MACs
 - Decrementing the Time-to-live (TTL)
 - Assigning the output port



Examples

- Populating and managing match-action tables
- Dumping table entries
- Adding/removing/modifying table entries
- Obtaining switch information
- Checking tables

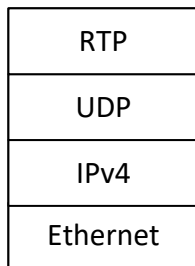
```
root@s1: /behavioral-model
RuntimeCmd: table_add MyIngress.ipv4_host MyIngress.forward 30.0.0.1 => 00:00:00
:00:00:02 2
Adding entry to exact match table MyIngress.ipv4_host
match key:          EXACT-1e:00:00:01
action:             MyIngress.forward
runtime data:       00:00:00:00:00:02  00:02
Entry has been added with handle 0
RuntimeCmd: █
```

Overview P4 Exercises

Exercises

- Parse UDP and Real-time Transport Protocol (RTP)
- UDP is identified by the “protocol field = 17,” in the IPv4 header
- Within UDP, if the destination port = 5004, then the packet is RTP

Packet headers



UDP header

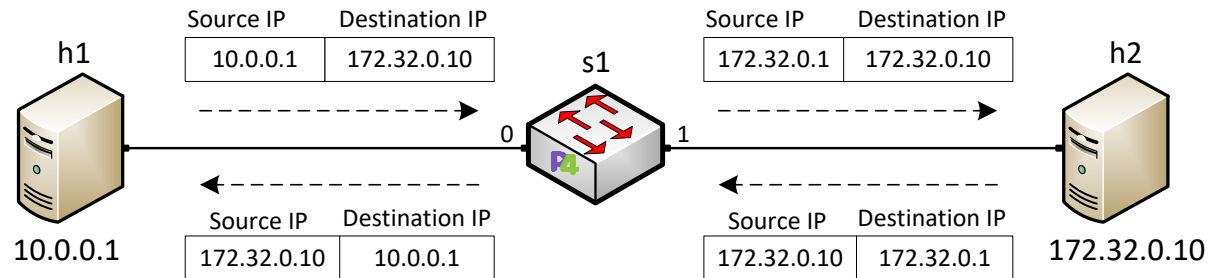
| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---------|-------|-------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Source port | | | | | | | | | | | | | | | | Destination port | | | | | | | | | | | | | | | |
| 4 | 32 | Length | | | | | | | | | | | | | | | | Checksum | | | | | | | | | | | | | | | |

RTP header

| Offsets | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---------|--------------------|-----------------|---|---|----|---|---|---|----|---|---|----|----|----|----|----|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Octet | Bit ^[a] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Version | P | X | CC | | | M | PT | | | | | | | | Sequence number | | | | | | | | | | | | | | | | |
| 4 | 32 | Timestamp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 64 | SSRC identifier | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

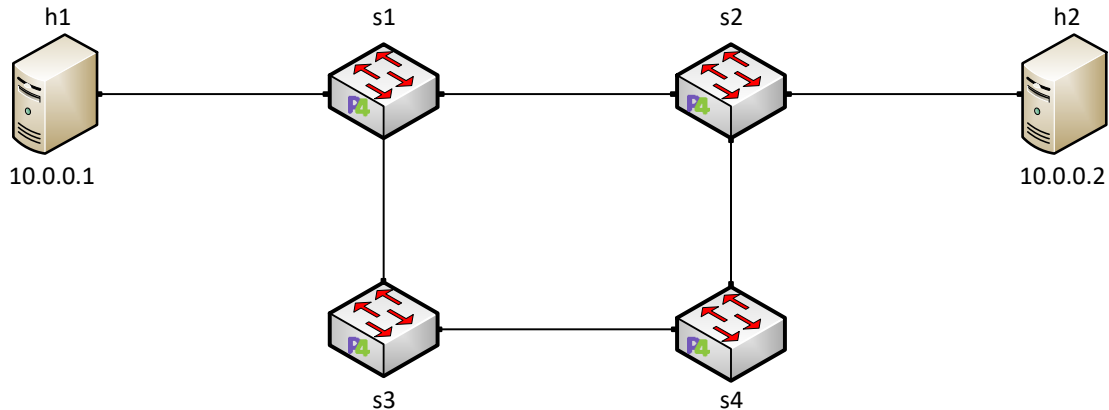
Exercises

- Implement a simplified version of the source and destination Network Address Translation (NAT)
- Modify the source IP address of the packet when leaving the network
- Modify the destination IP address of the packet when entering the network



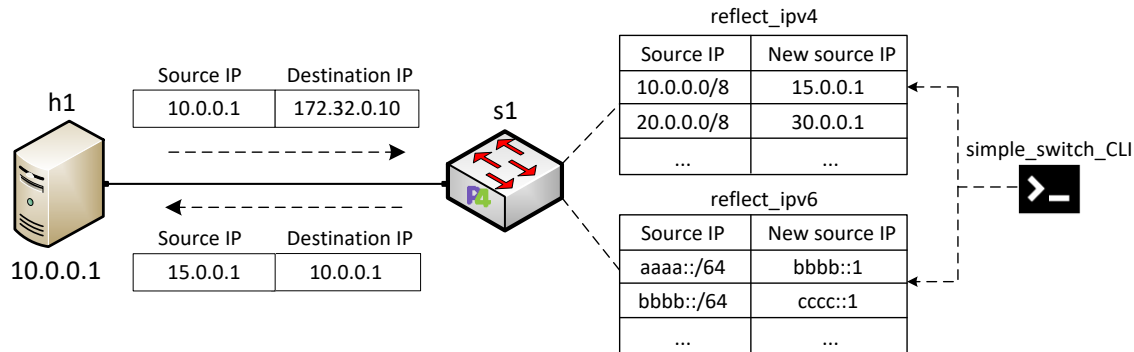
Exercises

- Push the table entries to the switches so that a packet sent from h1 to h2 traverses switches s1-s2
- Modify the path so that the packet traverses the switches s1-s3-s4-s2
- Write the rules that create a loop in the switches s1-s2-s4-s3-s1-s2-s4-s3...



Exercises

- Combining all concepts into a single program
- Define headers and parsing IPv4, IPv6
- Implement tables for reflecting IPv4 and IPv6 packets
- Populate the tables from the control plane
- Update the checksum of the IPv4 header



Additional Information

- Jorge Crichigno:
 - jcrichigno@cec.sc.edu
- Cyberinfrastructure lab at the University of South Carolina:
 - <http://ce.sc.edu/cyberinfra/>