Simple Router

EE323 Spring 2021

ee323@nmsl.kaist.ac.kr

Presentation template was originally adopted from SlidesCarnival.

Logistics

In lab sessions, we will give a brief introduction of upcoming projects.

- (4.5%) Lab session #1: Socket Programming
  - Open at 3/10, Due 3/18 (11:55 pm) - 1 week
- (4.5%) Lab session #2: HTTP Proxy
  - Open at 3/19, Due 3/30 (11:55 pm) - 1.5 week
- (6+6%) Lab session #3: Simple TCP in Reliable Transport Layer
  - Open at 3/31, Due 4/13, 5/11 (11:55 pm) - 2/4 weeks for each (including Midterm period)
- (9%) Lab session #4: Simple Router
  - Open at 5/12, Due 6/1 (11:55 pm) - 3 weeks

The Ultimate Guide

Primary Project Document

https://docs.google.com/document/d/1FvjZ3yYsJawWgalUDXL29v5gacAoPAZzep9CqiNjHmQ/edit?usp=sharing

This slide is based on the document above.

Please refer to this document first if there is any question.

Still ongoing project - we need your help and participation!

You can view and comment on the document directly, so please participate.

(hey, it’s a rich source of participation points!)

Introduction

- You are going to mimic a “Router”
  - Given a **static** network topology & routing table
  - No hardware router, but **software** one!

Introduction (2)

You will be able to understand

- How does a router use a **routing table** and **forward** received packets?
- How does a router handle **ARP** packets?
- When does a router send **ICMP** packets?

By emulating a **Simple Router** through **Virtual Network**.

High Level Requirements

You will build a simple virtual router with

1. Packet forwarding logic
   a. How to use longest-prefix-match on routing table
   b. Blacklist-based firewall
2. Link/Internet layer protocols
   a. Ethernet
   b. ARP
   c. IP
   d. ICMP

For details, check the document.

High Level Requirements (2)

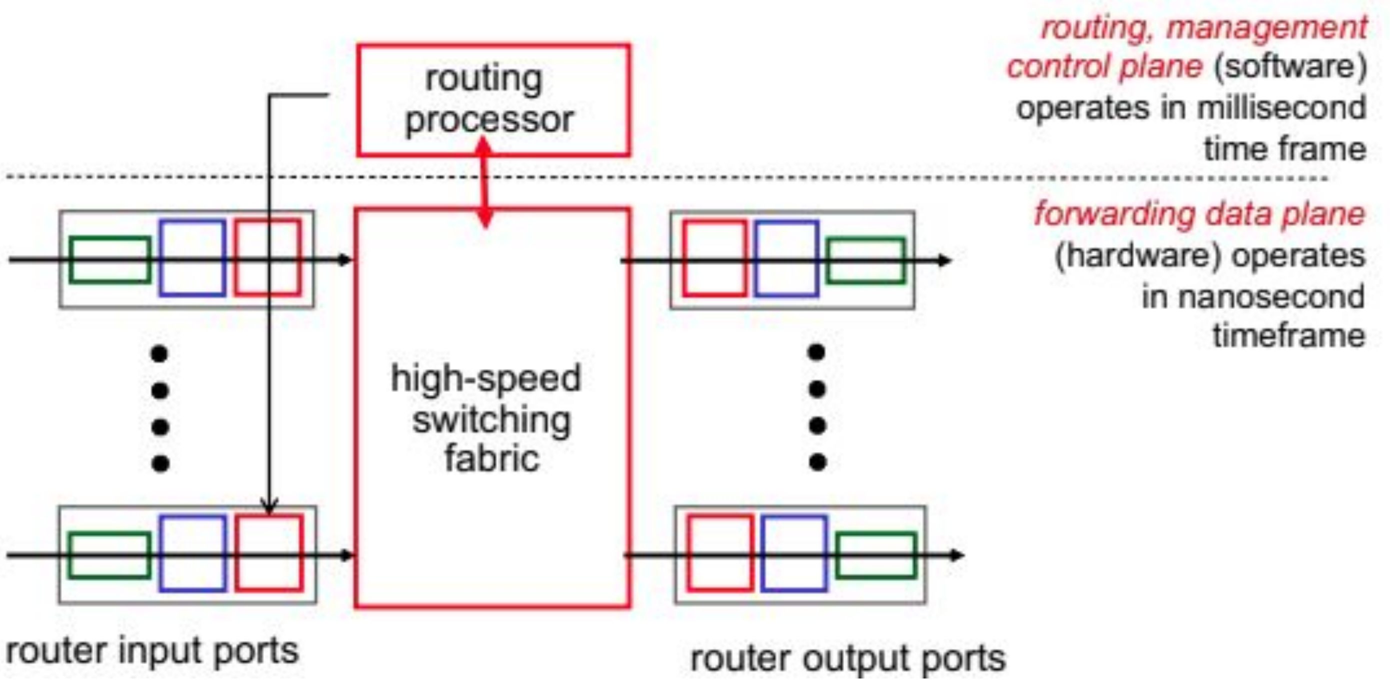
You should fill in the code at router directory:

- sr_router.c
  - `ip_back_list()`
  - `sr_handlepacket()`
- sr_arpcache.c
  - `sr_arpcache_handle_arpreq()`

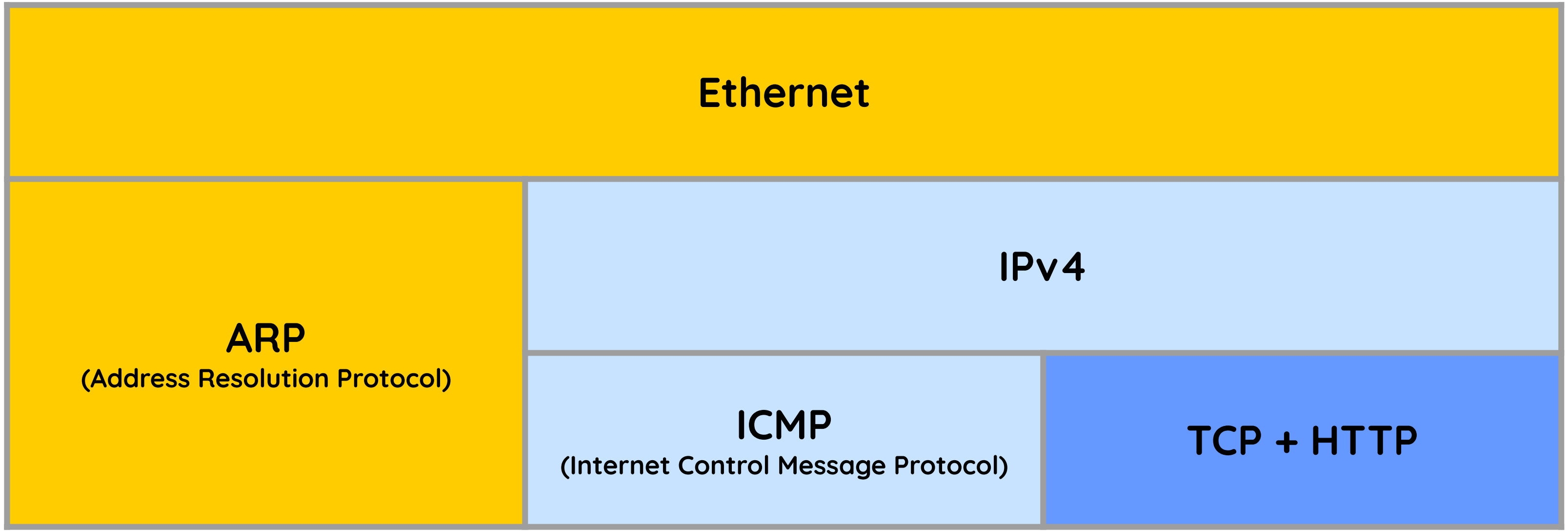
Do not modify the skeleton codes!

Router

- You will implement a ‘routing software’


routing processor
high-speed switching fabric
router input ports
router output ports
routing, management control plane (software) operates in millisecond time frame
forwarding data plane (hardware) operates in nanosecond timeframe

Protocols to Understand



Ethernet

ARP
(Address Resolution Protocol)

IPv4

ICMP
(Internet Control Message Protocol)

TCP + HTTP

You need to fully understand: Ethernet, ARP, IPv4, ICMP

Ethernet

Destination Address. 6 bytes.

- The address(es) are specified for a unicast, multicast (subgroup), or broadcast (an entire group).

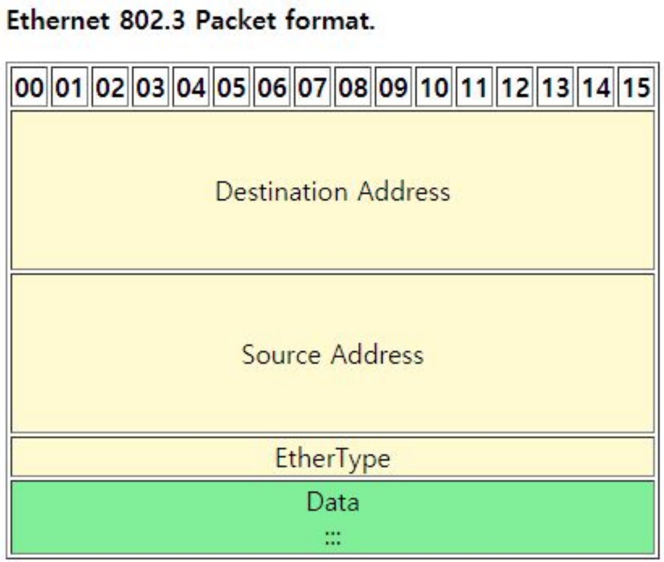
Source Address. 6 bytes.

- The address is for a unicast (single computer or device).

EtherType. 16 bits.

- Which upper layer protocol will utilized the Ethernet frame.

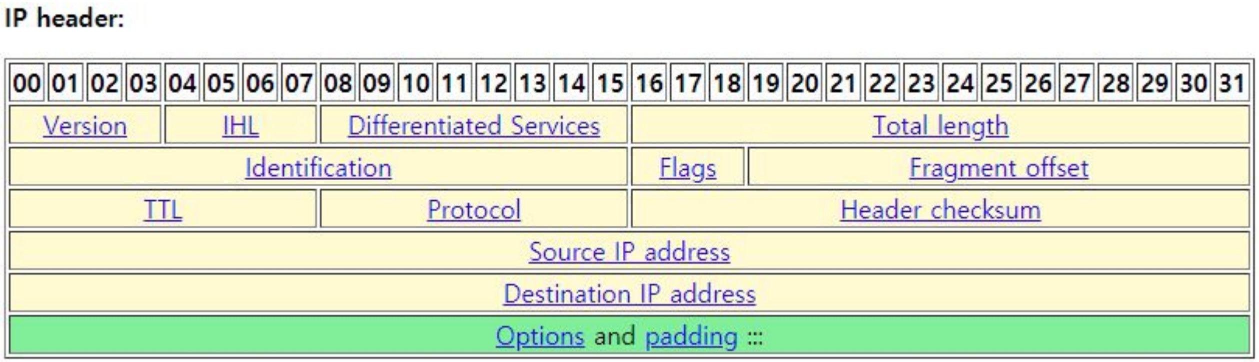
Data. variable, 46-1500 bytes.



Ethernet 802.3 Packet format.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
Destination Address															
Source Address															
EtherType															
Data :::															

IPv4



IP header:

<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="8">Differentiated Services</td><td colspan="16">Total length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment offset</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">Protocol</td><td colspan="16">Header checksum</td></tr>
<tr><td colspan="32">Source IP address</td></tr>
<tr><td colspan="32">Destination IP address</td></tr>
<tr><td colspan="32">Options and padding :::</td></tr>
</table>

Total Length. 16 bits.: Contains the length of the datagram.

TTL, Time to Live. 8 bits.: A timer field used to track the lifetime of the datagram. When the TTL field is decremented down to zero, the datagram is discarded.

Protocol. 8 bits.: This field specifies the next encapsulated protocol (ICMP / TCP).

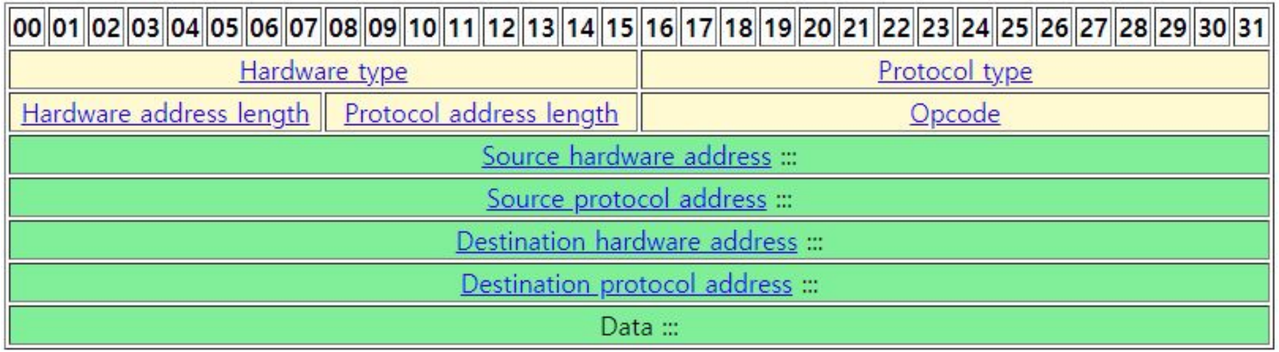
Header checksum. 16 bits.

Source IP address. 32 bits.

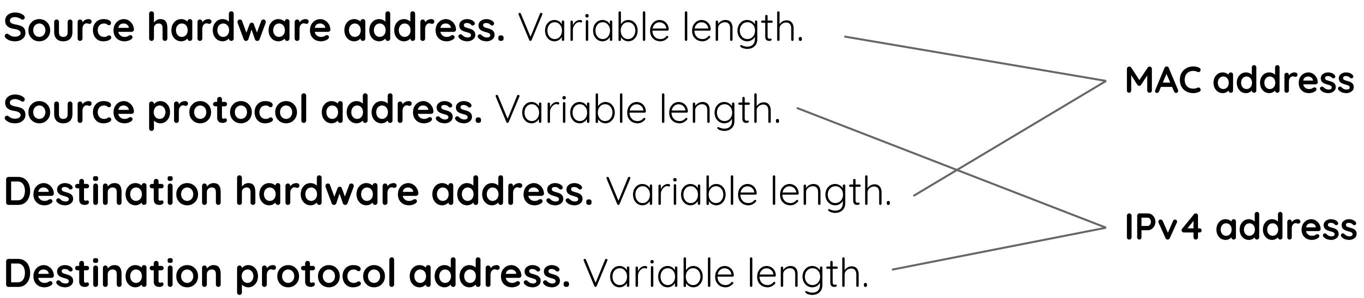
Destination IP address. 32 bits.

ARP

ARP header:



<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Hardware type</td><td colspan="16">Protocol type</td></tr>
<tr><td colspan="8">Hardware address length</td><td colspan="8">Protocol address length</td><td colspan="16">Opcode</td></tr>
<tr><td colspan="32">Source hardware address :::</td></tr>
<tr><td colspan="32">Source protocol address :::</td></tr>
<tr><td colspan="32">Destination hardware address :::</td></tr>
<tr><td colspan="32">Destination protocol address :::</td></tr>
<tr><td colspan="32">Data :::</td></tr>
</table>



Source hardware address. Variable length.

Source protocol address. Variable length.

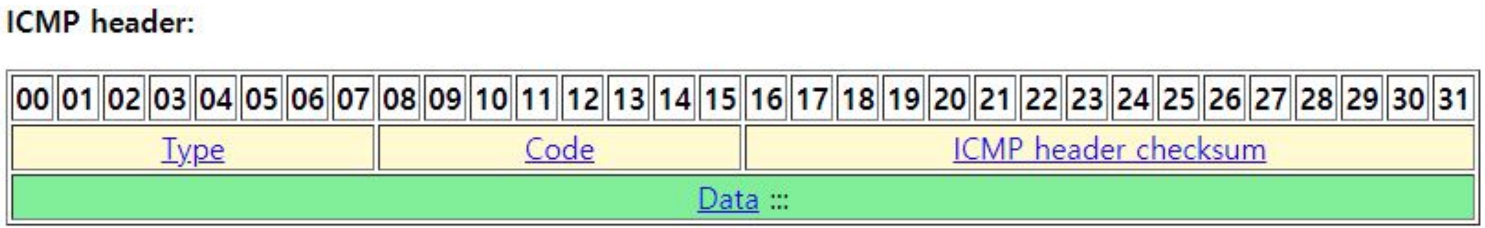
Destination hardware address. Variable length.

Destination protocol address. Variable length.

MAC address

IPv4 address

ICMP



ICMP header:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type								Code								ICMP header checksum															
Data :::																															

Type. 8 bits.

Code. 8 bits.

ICMP Header Checksum. 16 bits.

Data. Variable length.

Echo reply (Type: 0, Code: 0)

- Send as a response to an echo request (ping) to one of the router's interfaces.

Destination unreachable (Type: 3, Code: 0)

- Send if there is a non-existent route to the destination IP.

Destination host unreachable (Type: 3, Code: 1)

- Send if five ARP requests (sent to the next-hop IP) had no response.

Port unreachable (Type: 3, Code: 1)

- Send if IP packet containing TCP/UDP payload is sent to router's interface.

Time exceeded (Type: 11, Code: 0)

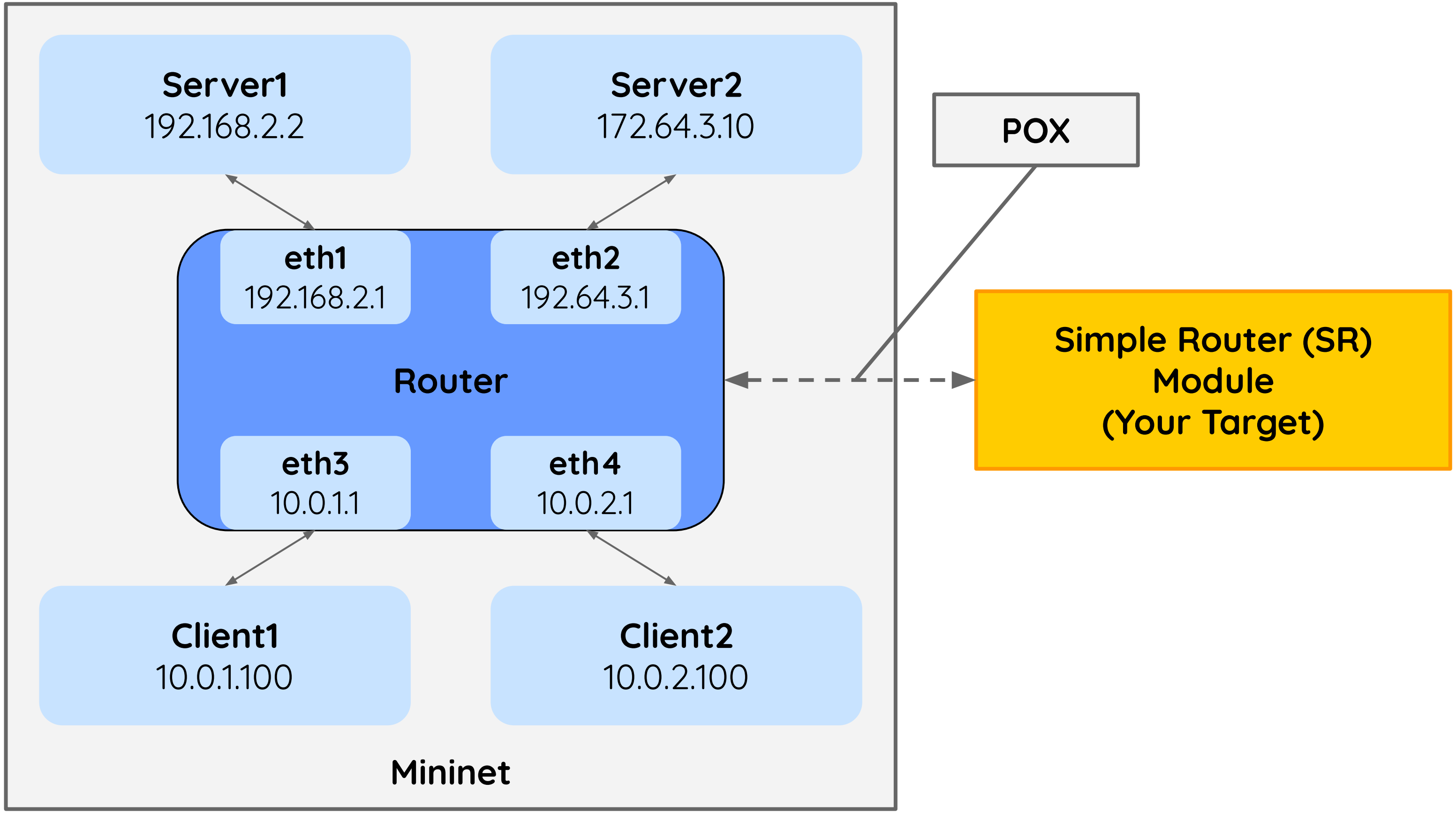
- Send if TTL expires (as zero).

Packet Forwarding Logic

Given an Ethernet frame which contains an IP packet

1. Sanity check the packet (length / checksum)
2. Check if the IP is on the **blacklist** (if yes → drop)
3. Decrement TTL (time-to-live) by 1, check if TTL expired (if yes → drop)
4. Find the proper interface by the longest prefix matching
5. Check the ARP cache for the next-hop MAC address corresponding to the IP address. If cache hit, send. Otherwise, queue the packet and request ARP.

Virtual Network Environment



How to Set the Environment?

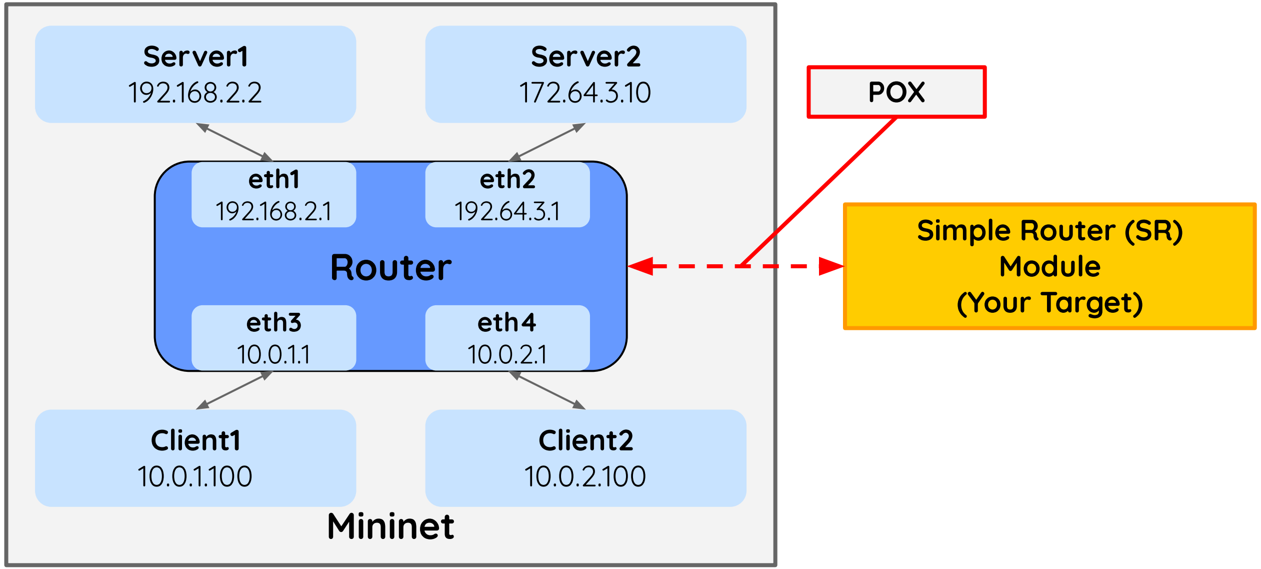
- We provide a Virtual Machine (VM) image that all required programs and environments are set.
  - Download VM image (link: TBD)
  - Install VirtualBox (https://www.virtualbox.org/wiki/Downloads)
  - Import image and execute
    - Ubuntu ID: ee323 / PW: ee323
    - Root PW: root
  - Do not upgrade your operating system!

How to Run the Solution Program

- We provide a solution program. To run,

1.
```
$ ./run_pox.sh
```

POX: Communication module between router in Mininet and your Simple Router

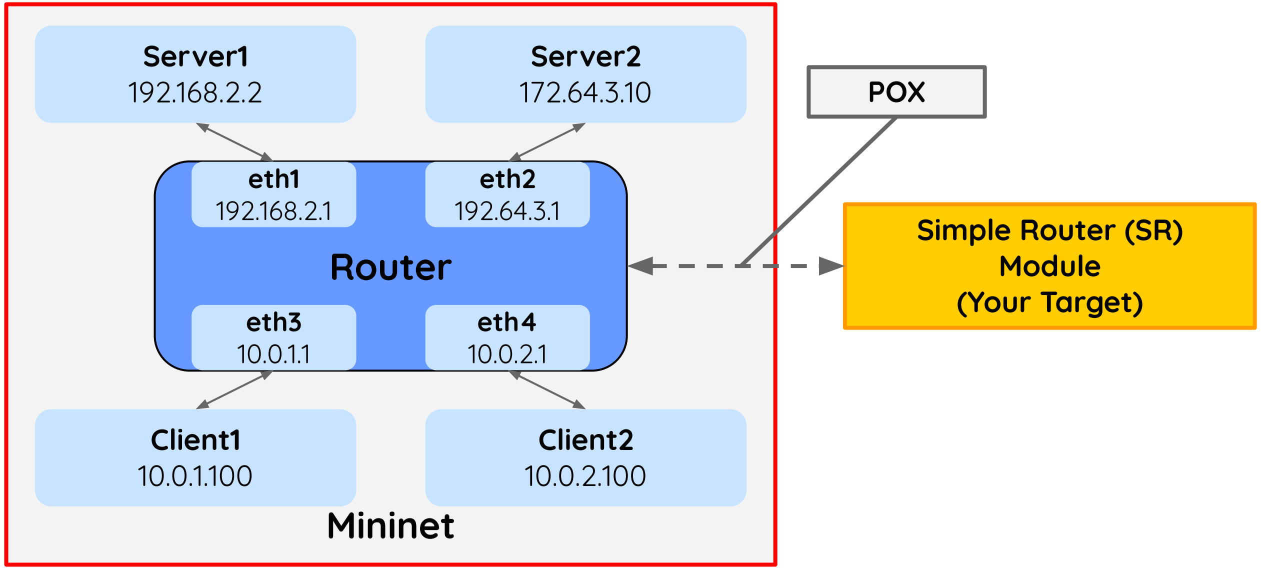


How to Run the Solution Program (2)

- We provide a solution program. To run,

2. `$ ./run_mininet.sh`

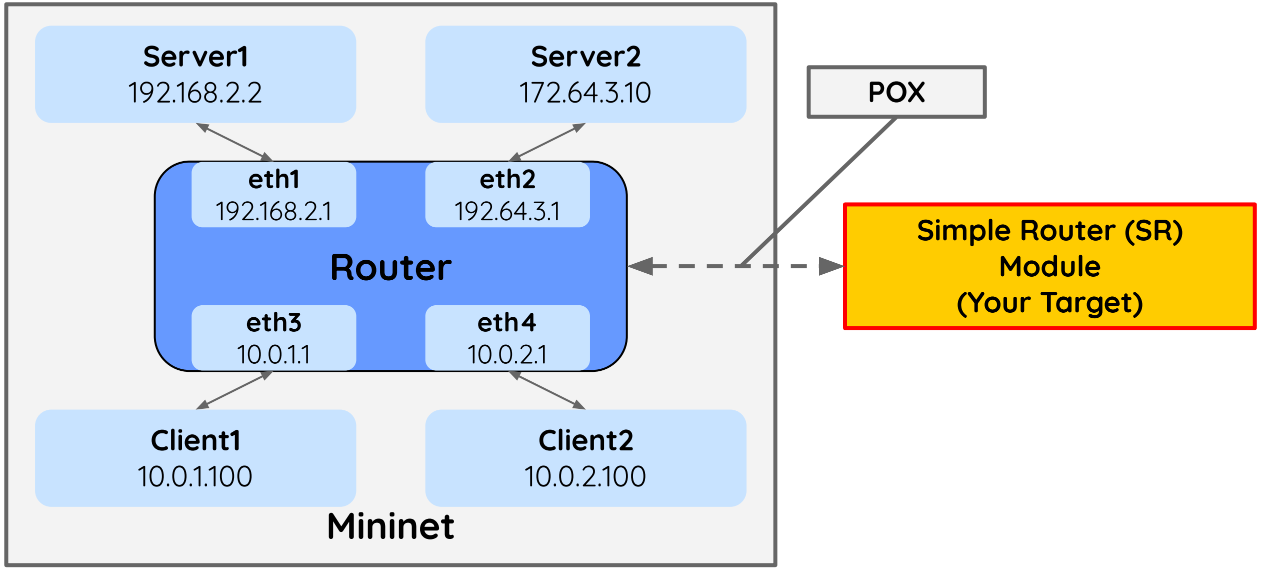
Mininet: Virtual Network Emulator



How to Run the Solution Program (3)

- We provide a solution program. To run,

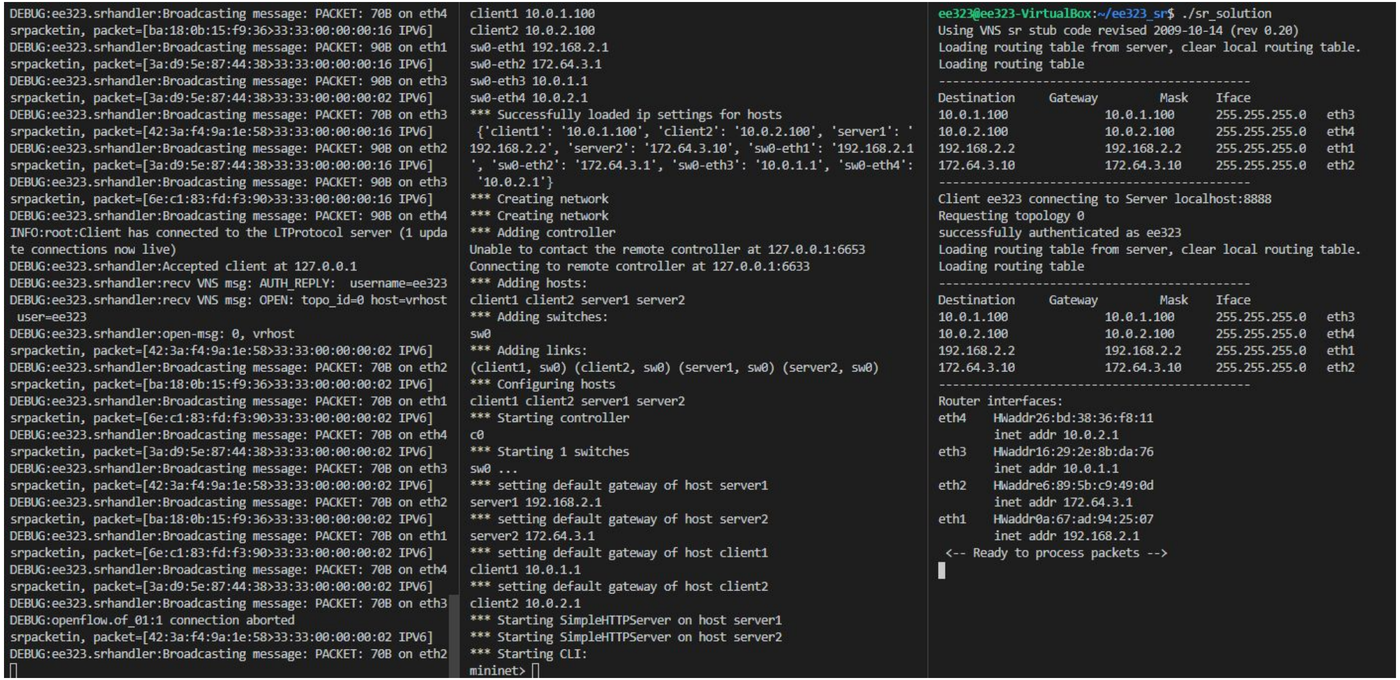
3. `$ ./sr_solution`



How to Run the Solution Program (4)

- We provide a solution program. To run,

3. `$ ./sr_solution`



```
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth4
srpacketin, packet=[ba:18:0b:15:f9:36>33:33:00:00:00:16 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 90B on eth1
srpacketin, packet=[3a:d9:5e:87:44:38>33:33:00:00:00:16 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 90B on eth3
srpacketin, packet=[3a:d9:5e:87:44:38>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth3
srpacketin, packet=[42:3a:f4:9a:1e:58>33:33:00:00:00:16 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 90B on eth2
srpacketin, packet=[3a:d9:5e:87:44:38>33:33:00:00:00:16 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 90B on eth3
srpacketin, packet=[6e:c1:83:fd:f3:90>33:33:00:00:00:16 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 90B on eth4
INFO:root:Client has connected to the LTProtocol server (1 upda
te connections now live)
DEBUG:ee323.srhandler:Accepted client at 127.0.0.1
DEBUG:ee323.srhandler:recv VNS msg: AUTH_REPLY:  username=ee323
DEBUG:ee323.srhandler:recv VNS msg: OPEN: topo_id=0 host=vrhost
 user=ee323
DEBUG:ee323.srhandler:open-msg: 0, vrhost
srpacketin, packet=[42:3a:f4:9a:1e:58>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth2
srpacketin, packet=[ba:18:0b:15:f9:36>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth1
srpacketin, packet=[6e:c1:83:fd:f3:90>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth4
srpacketin, packet=[3a:d9:5e:87:44:38>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth3
srpacketin, packet=[42:3a:f4:9a:1e:58>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth2
srpacketin, packet=[ba:18:0b:15:f9:36>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth1
srpacketin, packet=[6e:c1:83:fd:f3:90>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth4
srpacketin, packet=[3a:d9:5e:87:44:38>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth3
DEBUG:openflow.of_01:1 connection aborted
srpacketin, packet=[42:3a:f4:9a:1e:58>33:33:00:00:00:02 IPV6]
DEBUG:ee323.srhandler:Broadcasting message: PACKET: 70B on eth2
▯
```

```
client1 10.0.1.100
client2 10.0.2.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
sw0-eth4 10.0.2.1
*** Successfully loaded ip settings for hosts
 {'client1': '10.0.1.100', 'client2': '10.0.2.100', 'server1': '
192.168.2.2', 'server2': '172.64.3.10', 'sw0-eth1': '192.168.2.1
', 'sw0-eth2': '172.64.3.1', 'sw0-eth3': '10.0.1.1', 'sw0-eth4':
 '10.0.2.1'}
*** Creating network
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
client1 client2 server1 server2
*** Adding switches:
sw0
*** Adding links:
(client1, sw0) (client2, sw0) (server1, sw0) (server2, sw0)
*** Configuring hosts
client1 client2 server1 server2
*** Starting controller
c0
*** Starting 1 switches
sw0 ...
*** setting default gateway of host server1
server1 192.168.2.1
*** setting default gateway of host server2
server2 172.64.3.1
*** setting default gateway of host client1
client1 10.0.1.1
*** setting default gateway of host client2
client2 10.0.2.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
mininet> ▯
```

```
ee323@ee323-VirtualBox:~/ee323_sr$ ./sr_solution
Using VNS sr stub code revised 2009-10-14 (rev 0.20)
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100              10.0.1.100      255.255.255.0   eth3
10.0.2.100              10.0.2.100      255.255.255.0   eth4
192.168.2.2             192.168.2.2     255.255.255.0   eth1
172.64.3.10             172.64.3.10     255.255.255.0   eth2
---------------------------------------------
Client ee323 connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as ee323
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100              10.0.1.100      255.255.255.0   eth3
10.0.2.100              10.0.2.100      255.255.255.0   eth4
192.168.2.2             192.168.2.2     255.255.255.0   eth1
172.64.3.10             172.64.3.10     255.255.255.0   eth2
---------------------------------------------
Router interfaces:
eth4    HWaddr26:bd:38:36:f8:11
        inet addr 10.0.2.1
eth3    HWaddr16:29:2e:8b:da:76
        inet addr 10.0.1.1
eth2    HWaddre6:89:5b:c9:49:0d
        inet addr 172.64.3.1
eth1    HWaddr0a:67:ad:94:25:07
        inet addr 192.168.2.1
 <-- Ready to process packets -->
▮
```

Recap: High Level Requirements

You will build a simple virtual router with

1. Packet forwarding logic
   a. How to use longest-prefix-match on routing table
   b. Blacklist-based firewall
2. Link/Internet layer protocols
   a. Ethernet
   b. ARP
   c. IP
   d. ICMP

For details, check the document.

Recap: High Level Requirements (2)

You should fill in the code at router directory:

- sr_router.c
  - `ip_back_list()`
  - `sr_handlepacket()`
- sr_arpcache.c
  - `sr_arpcache_handle_arpreq()`

Do not modify the skeleton codes!

Submission

Due date: June 1st, Thursday, 23:55 PM

You need to submit

- sr_router.c
- sr_arpcache.c
- report.pdf

Compress above items into one zip file and rename to: {studentID}_{name(in English)}_project4.zip

(ex. 20219876_JohnDoe_project4.zip)

Tips & Caution

The most important thing is to understand the project

- Read the **document & resources** carefully

And then, understand the source codes

- Check several header/source files in **router** folder

Be careful on handling endianness

- **Network byte order** and **host byte order** is different
- Be familiar with related functions

Live Demo Session