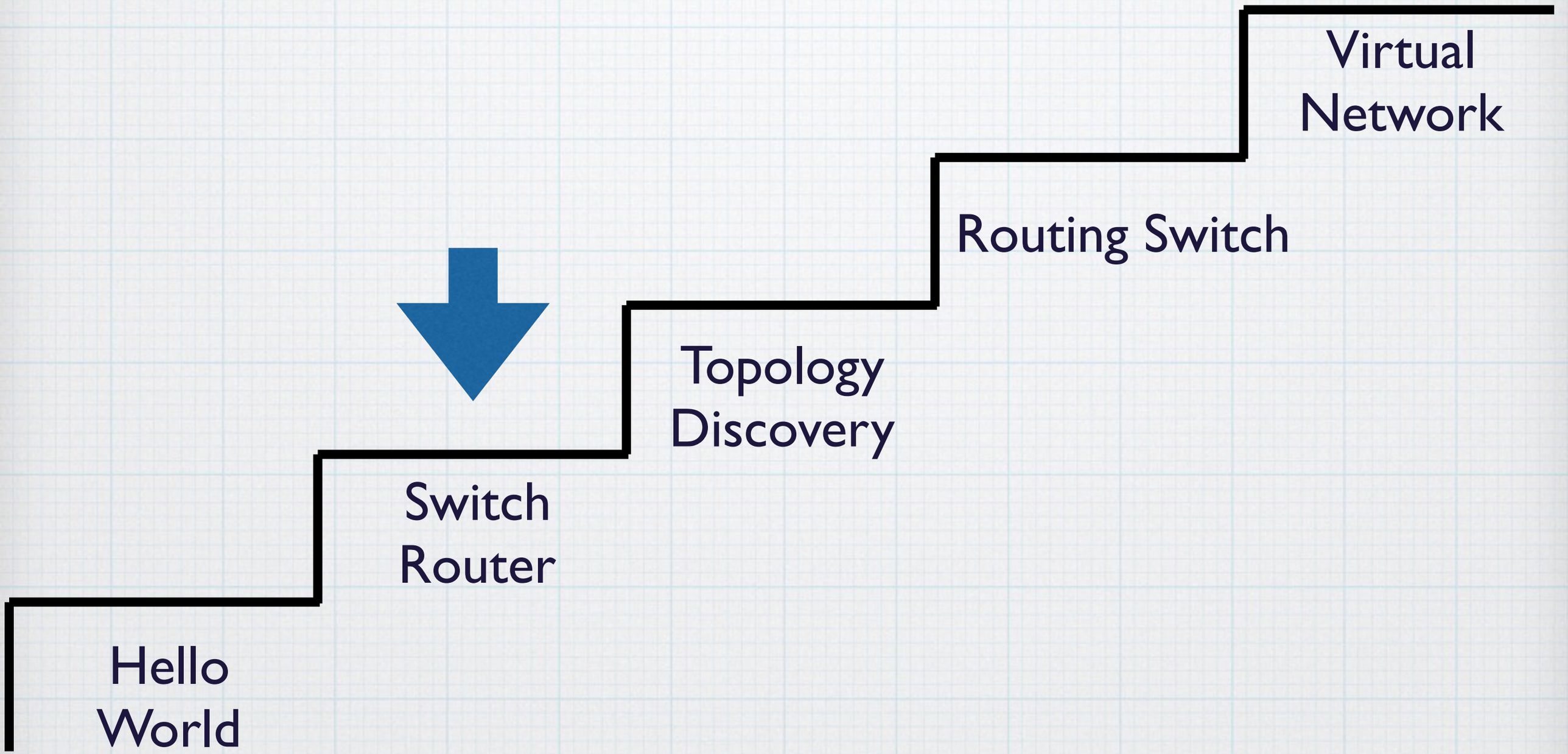
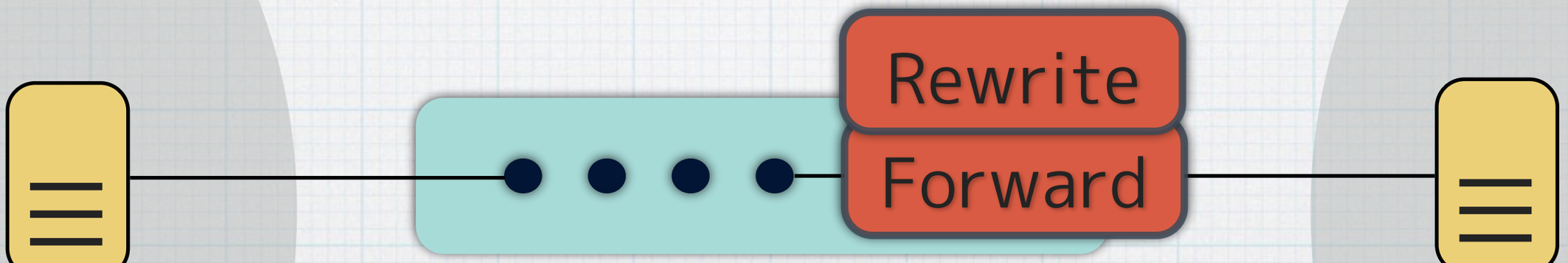
Router with OpenFlow Part 1

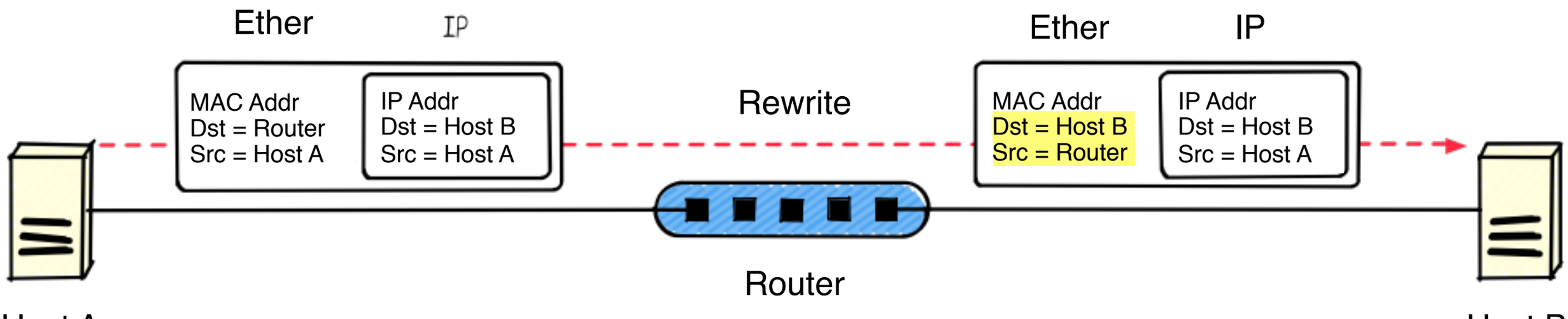

Virtual
Network
Routing Switch
Topology
Discovery
Switch
Router
Hello
World

Two Main Functions of a Router



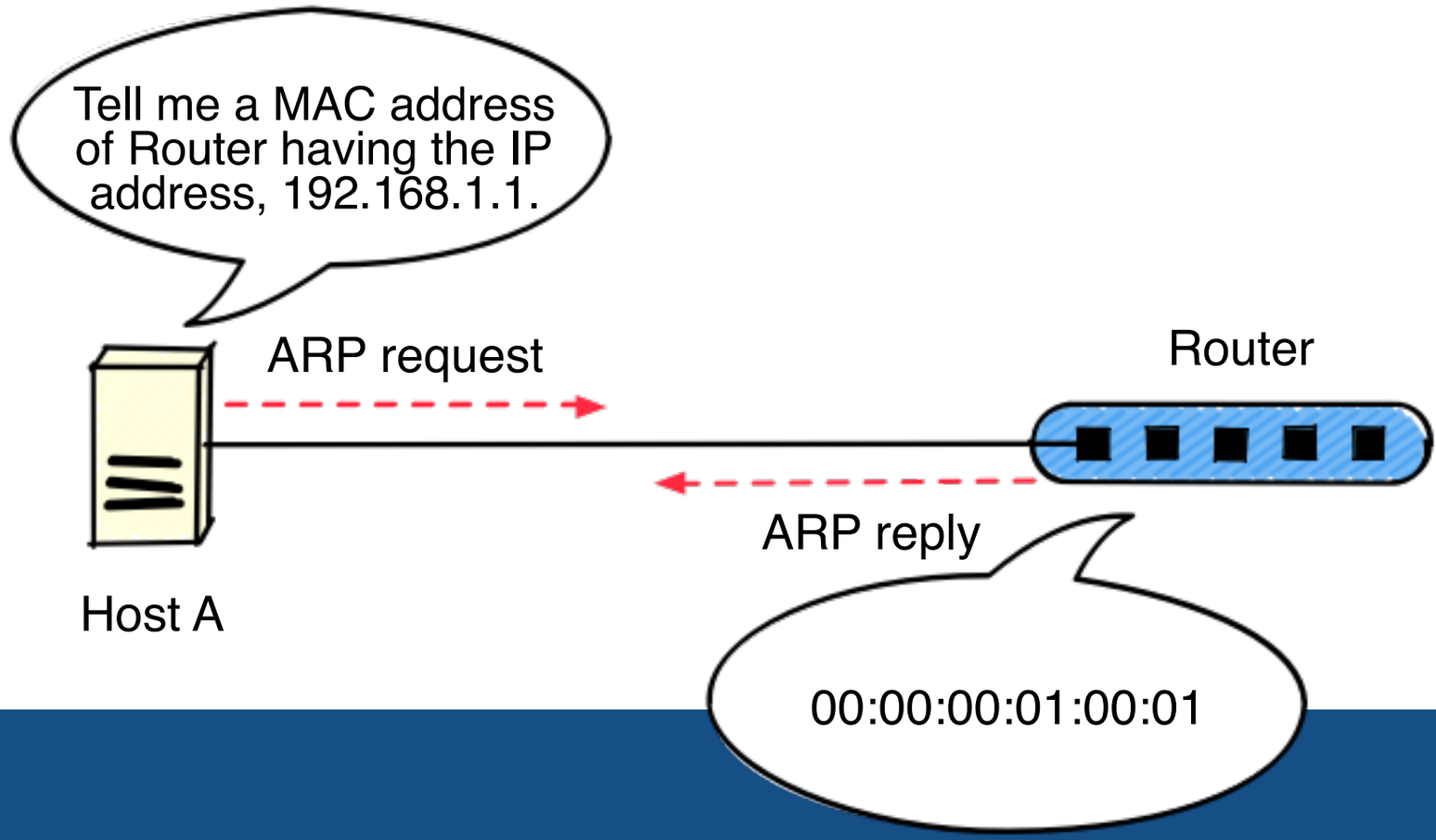
1. Rewrite source and destination mac address of a packet
2. Forward it

Rewriting and Forwarding Packets



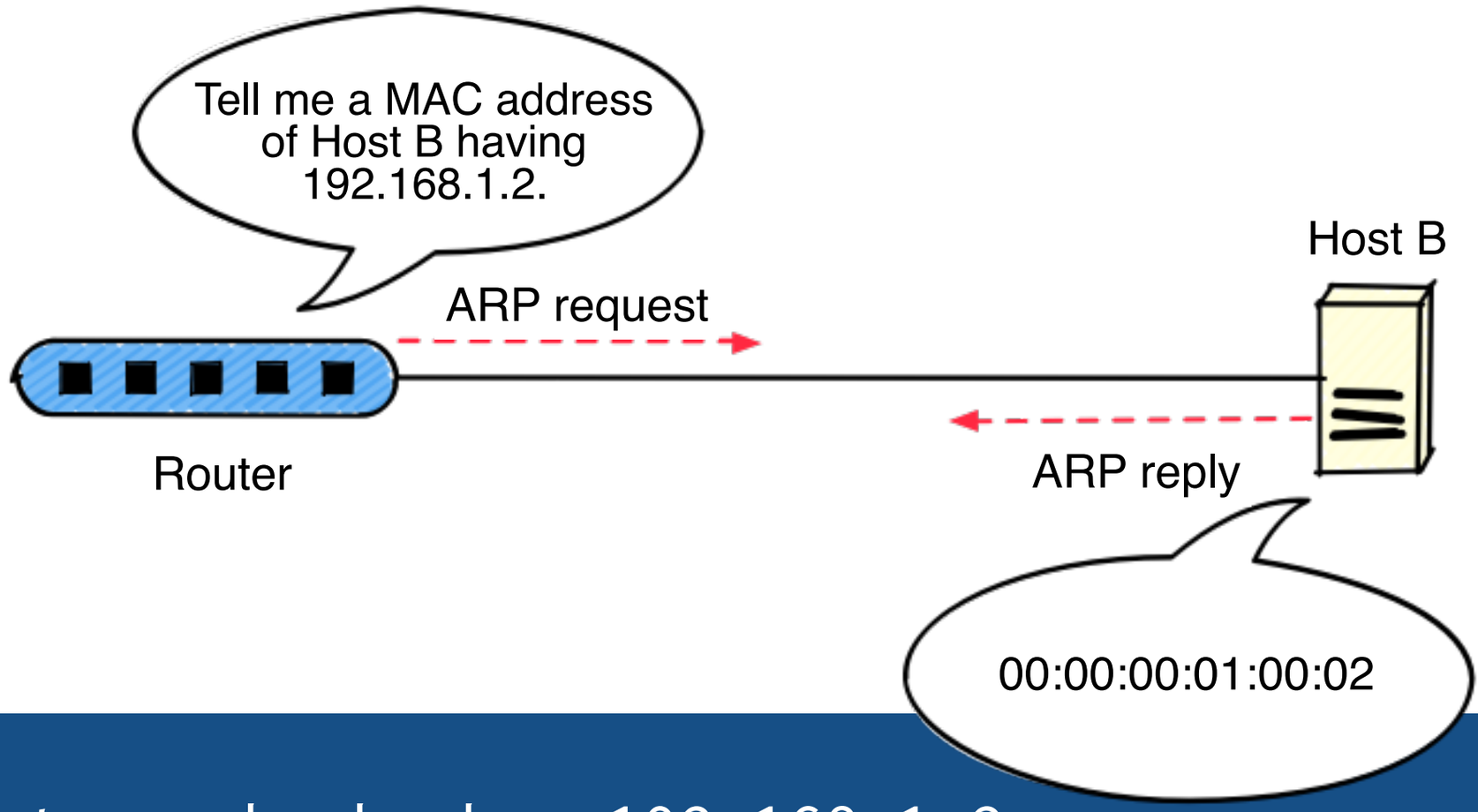
- Router rewrites the ethernet header of received packets as follows:
  - Set the destination MAC address to Host B
  - Set the source MAC address to Router
- Then, Router forwards them.

Address Resolution



- Host A asks who has the IP address 192.168.1.1.
- Router replies to the request with its MAC address so that Host A can send packets to Router
- ARP (Address Resolution Protocol)

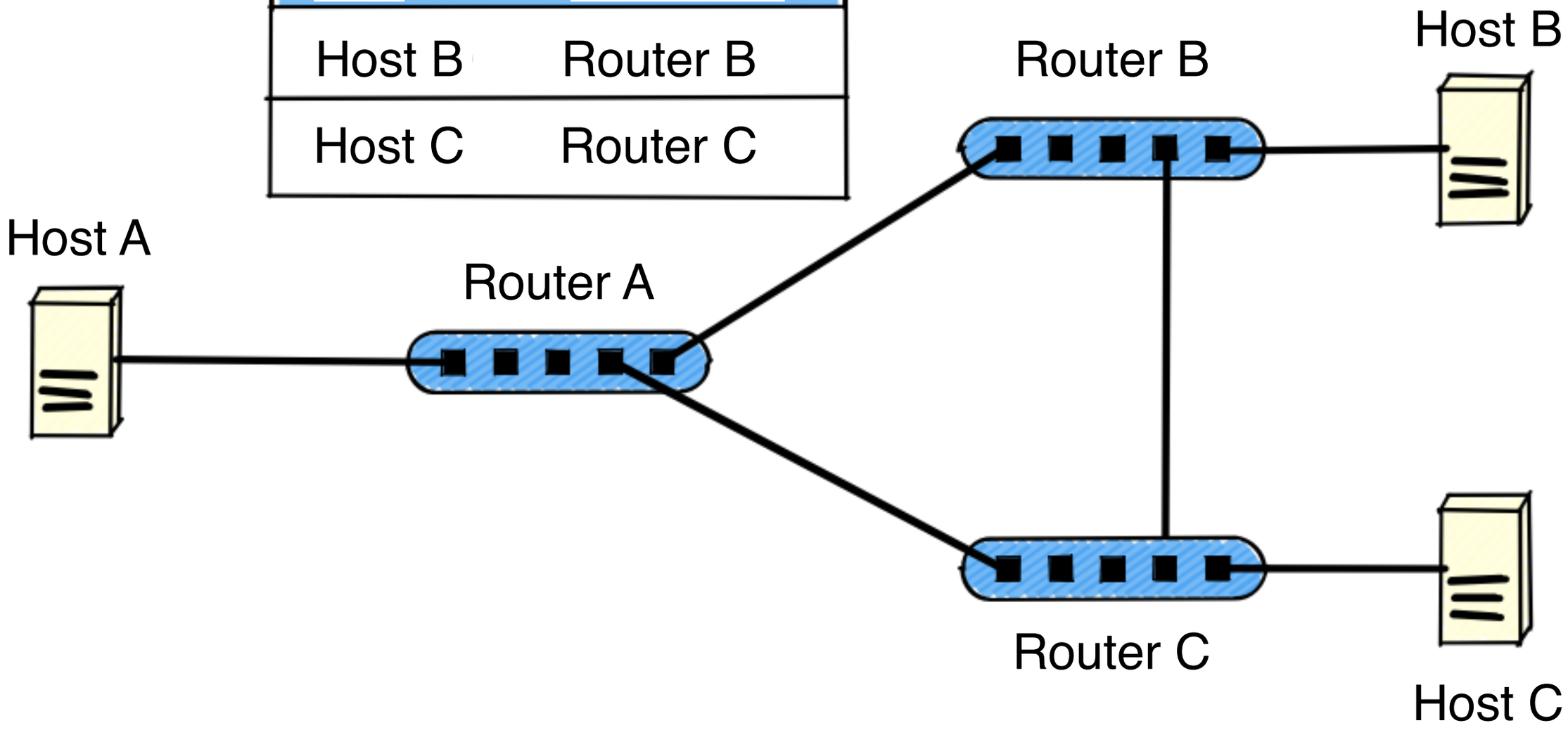
Address Resolution



- Router ask who has 192.168.1.2.
- Host B replies with its MAC address so that Router can send packets to Host B
- Router stores the resolved MAC and IP addresses to the ARP table on Router

Forwarding Packets via Routers

Dst	Next hop
Host B	Router B
Host C	Router C



PacketIn Handler

PacketIn Handler

```
def packet_in(dpid, packet_in)
  return unless sent_to_router?(packet_in)

  case packet_in.data
  when Arp::Request
    packet_in_arp_request dpid, packet_in.in_port, packet_in.data
  when Arp::Reply
    packet_in_arp_reply dpid, packet_in
  when Parser::IPv4Packet
    packet_in_ipv4 dpid, packet_in
  else
    logger.debug "Dropping unsupported packet type: #{packet_in.data.inspect}"
  end
end
```

- Check whether destination of a received packet is a router
- Check its type

Packet Destination Check

```
def sent_to_router?(packet_in)
  return true if packet_in.destination_mac.broadcast?
  interface = Interface.find_by(port_number: packet_in.in_port)
  interface && interface.mac_address == packet_in.destination_mac
end
```

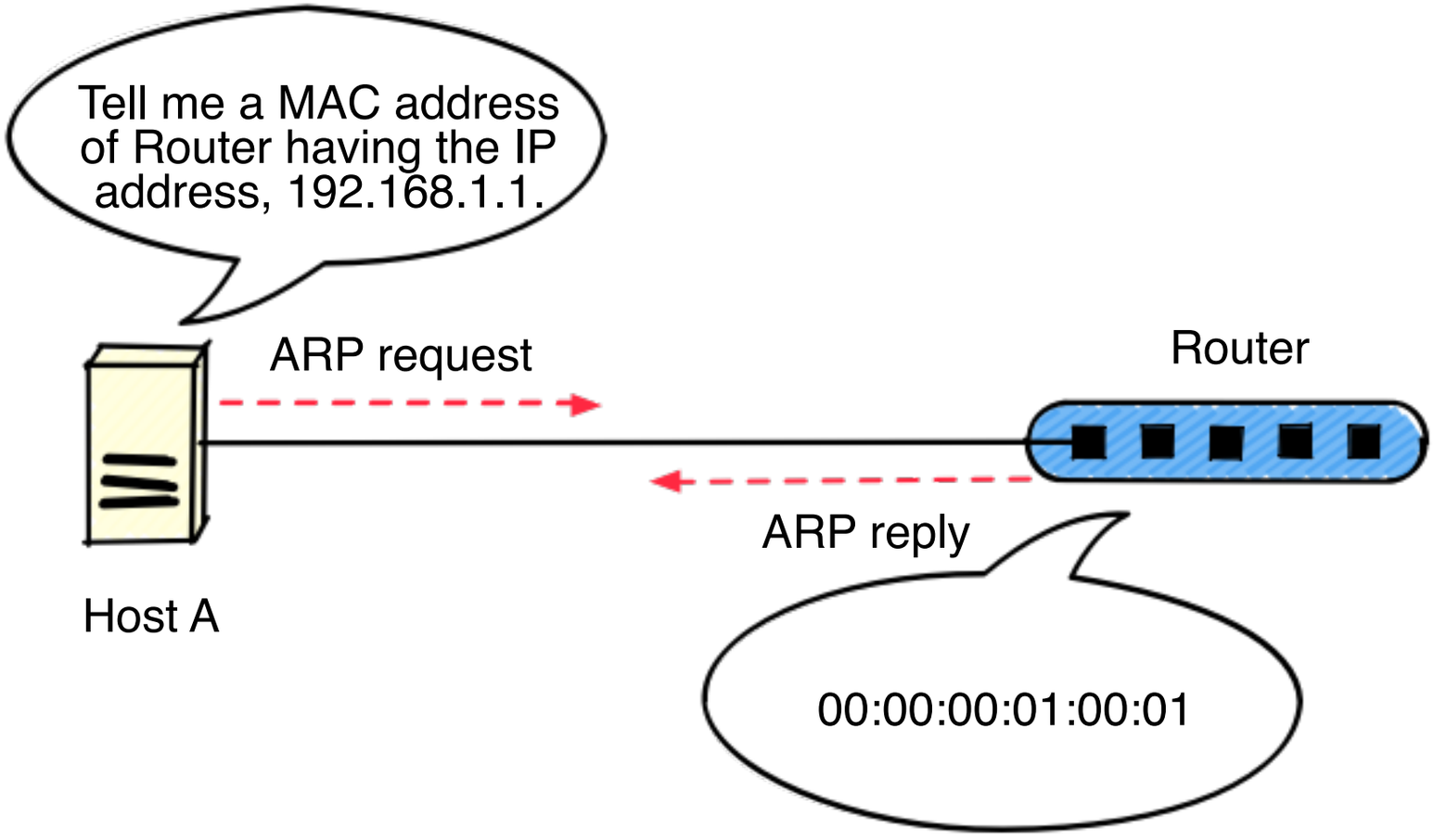
- Return true if the destination MAC address of a received packet is the broadcast address
- Return true if its destination MAC address is that of the router

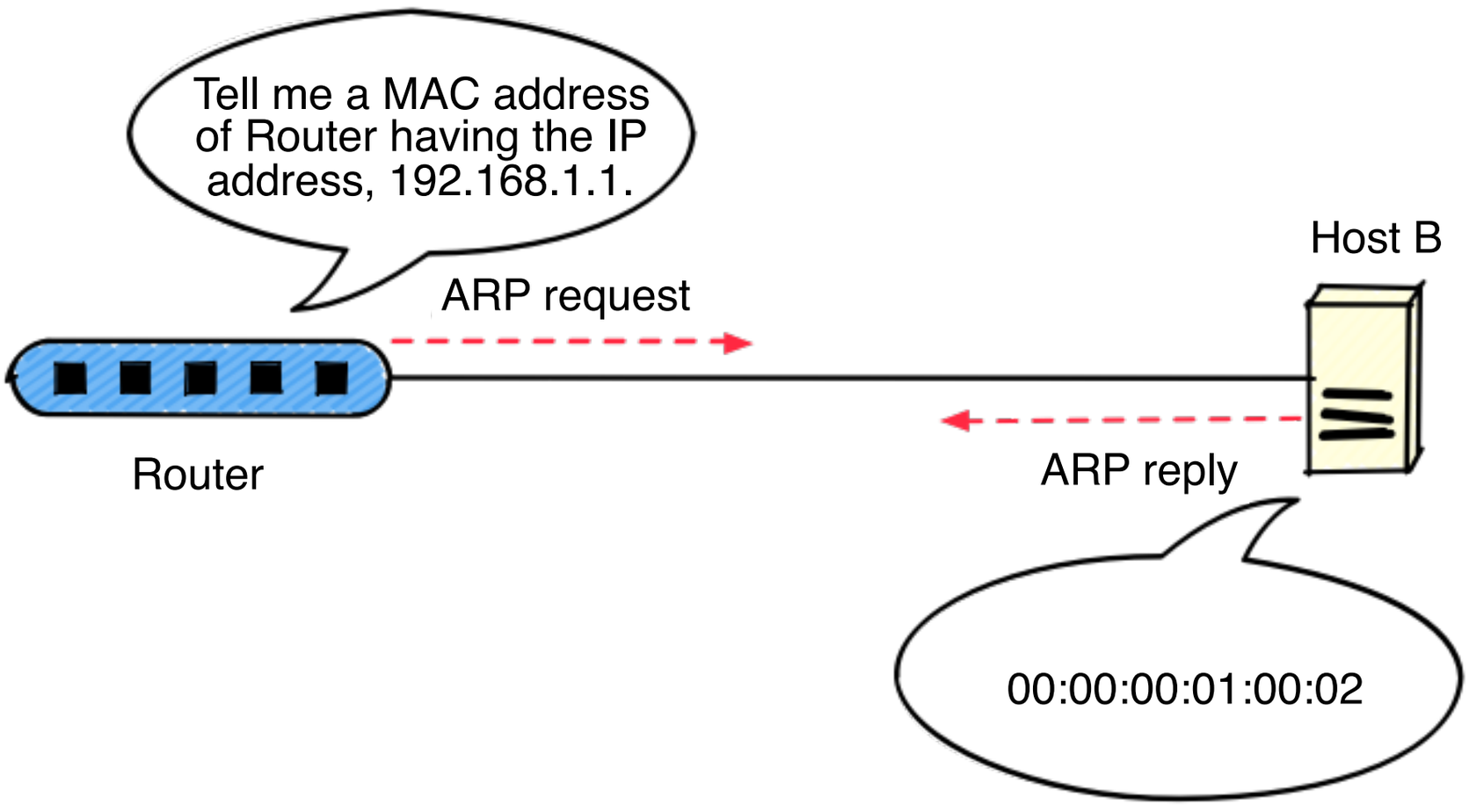
Packet Type Check

```
  case packet_in.data
  when Arp::Request
    packet_in_arp_request dpid, packet_in.in_port, packet_in.data
  when Arp::Reply
    packet_in_arp_reply dpid, packet_in
  when Parser::IPv4Packet
    packet_in_ipv4 dpid, packet_in
  else
    logger.debug "Dropping unsupported packet type: #{packet_in.data.inspect}"
  end
end
```

1. ARP request packet
2. ARP reply packet
3. IPv4 packet

PacketIn Handler
→Procedures for ARP


Tell me a MAC address of Router having the IP address, 192.168.1.1.
ARP request
Router
ARP reply
Host A
00:00:00:01:00:01


Tell me a MAC address of Router having the IP address, 192.168.1.1.
Host B
ARP request
Router
ARP reply
00:00:00:01:00:02

Procedure when the router receives an APR request packet

```
def packet_in_arp_request(dpid, in_port, arp_request)
  interface =
    Interface.find_by(port_number: in_port,
                      ip_address: arp_request.target_protocol_address)
  return unless interface
  send_packet_out(
    dpid,
    raw_data: Arp::Reply.new(
      destination_mac: arp_request.source_mac,
      source_mac: interface.mac_address,
      sender_protocol_address: arp_request.target_protocol_address,
      target_protocol_address: arp_request.sender_protocol_address
    ).to_binary,
    actions: SendOutPort.new(in_port))
end
```

- The router checks whether the ARP request asks one of IP addresses of the router
- If so, it sends an ARP reply

Procedure when the router receives an ARP reply packet

```
def packet_in_arp_reply(dpid, packet_in)
  @arp_table.update(packet_in.in_port,
                    packet_in.sender_protocol_address,
                    packet_in.source_mac)
  flush_unsent_packets(dpid,
                       packet_in.data,
                       Interface.find_by(port_number: packet_in.in_port))
end
```

- The router updates its ARP table according to the ARP reply packet
- The router flushes unsent packets, whose destination IP address has just resolved with this ARP reply packet

PacketIn Handler
→Procedures for IP

Procedure when the router receives an IP packet

```
def packet_in_ipv4(dpid, packet_in)
  if forward?(packet_in)
    forward(dpid, packet_in)
  elsif packet_in.ip_protocol == 1
    icmp = Icmp.read(packet_in.raw_data)
    packet_in_icmpv4_echo_request(dpid, packet_in) if icmp.icmp_type == 8
  else
    logger.debug "Dropping unsupported IPv4 packet: #{packet_in.data}"
  end
end
```

The router does the several procedures depending on the following three cases:

1. The case that it has to forward the packet
2. The case that the destination of the packet is the router and the packet is an ICMP echo message
3. Otherwise it discards the packet

How to check whether the router has to forward the packet

```
def forward?(packet_in)
  !Interface.find_by(ip_address: packet_in.destination_ip_address)
end
```

Not (Destination IP address ==
Any of IP addresses of the router)

How to reply to the echo message

```
def packet_in_icmpv4_echo_request(dpid, packet_in)
  icmp_request = Icmp.read(packet_in.raw_data)
  if @arp_table.lookup(packet_in.source_ip_address)
    send_packet_out(dpid,
                    raw_data: create_icmp_reply(icmp_request).to_binary,
                    actions: SendOutPort.new(packet_in.in_port))
  else
    send_later(dpid,
               interface: Interface.find_by(port_number: packet_in.in_port),
               destination_ip: packet_in.source_ip_address,
               data: create_icmp_reply(icmp_request))
  end
end
```

1. Read content of an ICMP request
2. Send an ICMP reply if the target MAC address is already in the ARP table
3. Otherwise, the ICMP reply is stored to the “send later” queue

PacketIn Handler

→Procedures for IP

→Rewriting and Forwarding

Rewriting and Forwarding Packets

```
def forward(dpid, packet_in)
  next_hop = resolve_next_hop(packet_in.destination_ip_address)

  interface = Interface.find_by_prefix(next_hop)
  return if !interface || (interface.port_number == packet_in.in_port)

  arp_entry = @arp_table.lookup(next_hop)
  if arp_entry
    actions = [SetSourceMacAddress.new(interface.mac_address),
               SetDestinationMacAddress.new(arp_entry.mac_address),
               SendOutPort.new(interface.port_number)]
    send_flow_mod_add(dpid,
                      match: ExactMatch.new(packet_in), actions: actions)
    send_packet_out(dpid, raw_data: packet_in.raw_data, actions: actions)
  else
    send_later(dpid,
               interface: interface,
               destination_ip: next_hop,
               data: packet_in.data)
  end
end
```

Rewriting and Forwarding Packets

```
actions = [SetSourceMacAddress.new(interface.mac_address),
           SetDestinationMacAddress.new(arp_entry.mac_address),
           SendOutPort.new(interface.port_number)]
send_flow_mod_add(dpid,
                  match: ExactMatch.new(packet_in), actions: actions)
send_packet_out(dpid, raw_data: packet_in.raw_data, actions: actions)
```

- Create an action to rewrite an MAC address, and then invoke FlowMod and PacketOut

Queued ICMP replies are sent after their destination MAC addresses are resolved

```
def packet_in_icmpv4_echo_request(dpid, packet_in)
  icmp_request = Icmp.read(packet_in.raw_data)
  if @arp_table.lookup(packet_in.source_ip_address)
    send_packet_out(dpid,
                    raw_data: create_icmp_reply(icmp_request).to_binary,
                    actions: SendOutPort.new(packet_in.in_port))
  else
    send_later(dpid,
               interface: Interface.find_by(port_number: packet_in.in_port),
               destination_ip: packet_in.source_ip_address,
               data: create_icmp_reply(icmp_request))
  end
end
```

How is this action, sending packets later, realized?

Queued ICMP replies are sent after their destination MAC addresses are resolved

```
def send_later(dpid, options)
  destination_ip = options.fetch(:destination_ip)
  @unresolved_packet_queue[destination_ip] += [options.fetch(:data)]
  send_arp_request(dpid, destination_ip, options.fetch(:interface))
end
```

1. Prepare queues for each destination IP address and store packets to the queue
2. Send ARP requests to resolve destination MAC addresses

When ARP resolution is completed,

```
def flush_unsent_packets(dpid, arp_reply, interface)
  destination_ip = arp_reply.sender_protocol_address
  @unresolved_packet_queue[destination_ip].each do |each|
    rewrite_mac =
      [SetDestinationMacAddress.new(arp_reply.sender_hardware_address),
       SetSourceMacAddress.new(interface.mac_address),
       SendOutPort.new(interface.port_number)]
    send_packet_out(dpid, raw_data: each.to_binary_s, actions: rewrite_mac)
  end
  @unresolved_packet_queue[destination_ip] = []
end
```

- #packet_in → #packet_in_arp_reply → #flush_unsent_packets
- Rewrite a destination MAC address with resolved one

Conclusion

- Fundamentals of routers
  - Rewriting MAC address
  - ARP request/reply
- Routing table (See the next slide for the detailed instructions)