ルータを作ろう 前編

仮想NW

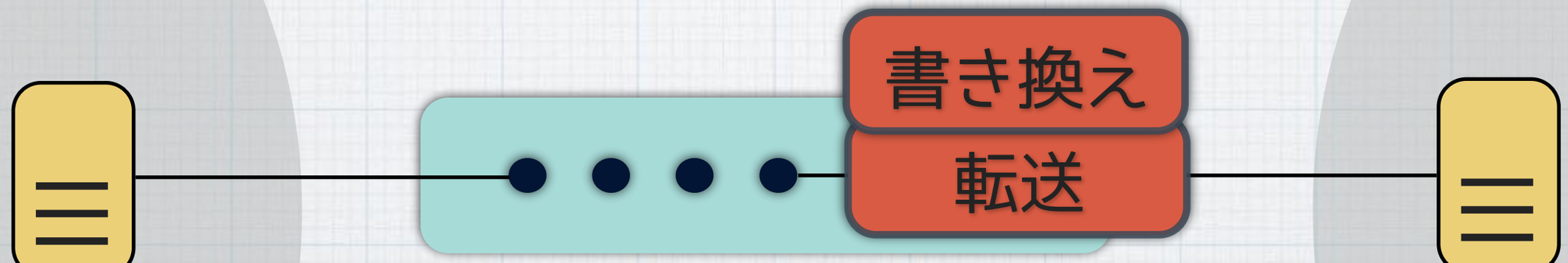
ルーティング
スイッチ

トポロジ
ディスカバリ

スイッチ
ルータ

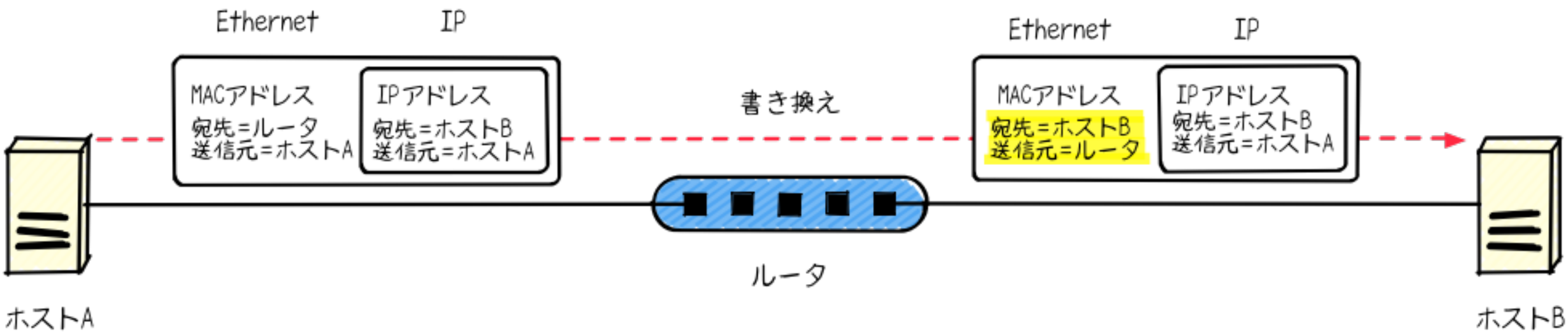
Hello
World

ルータの仕事は2つ



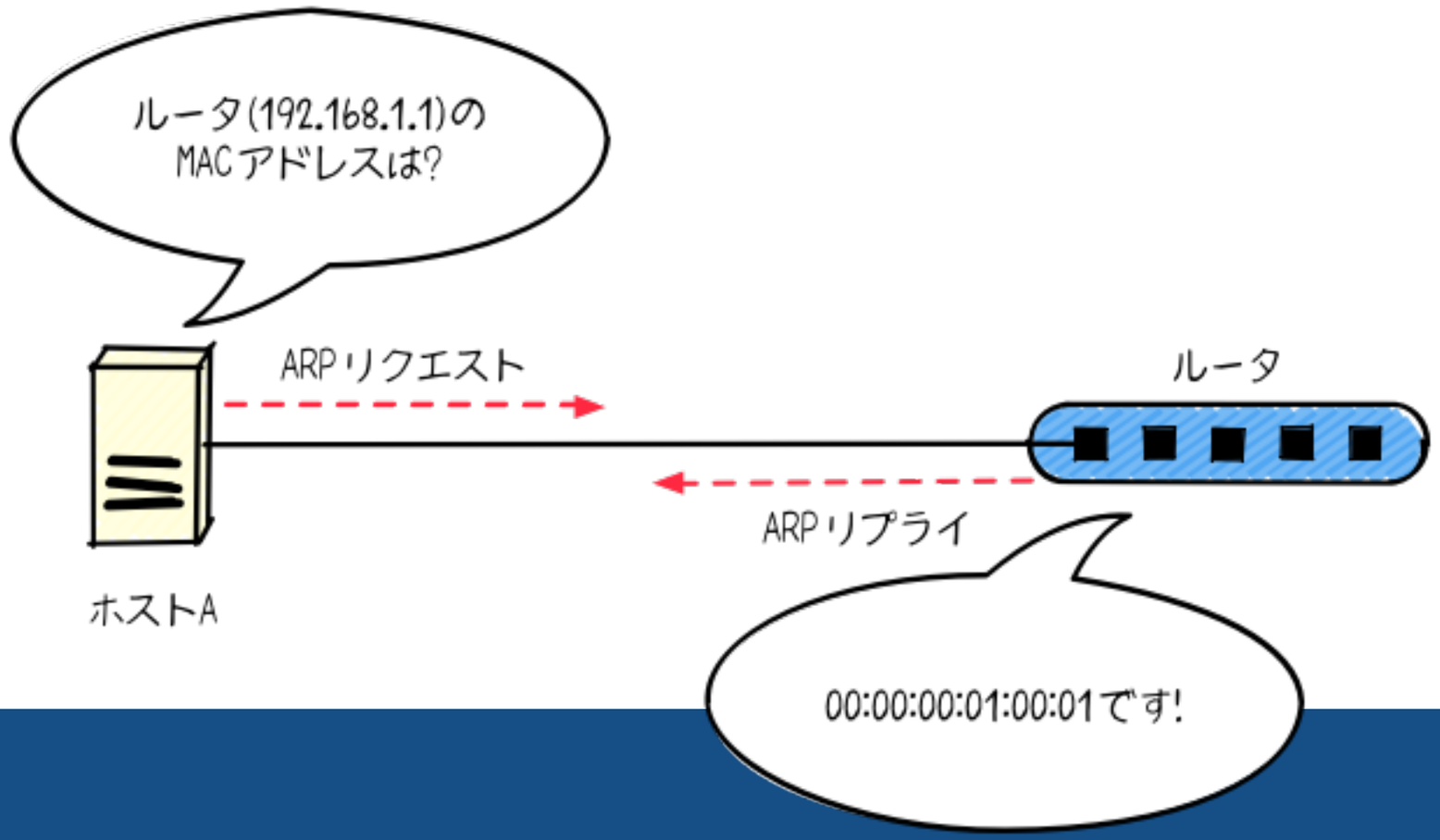
送信元・宛先
MACアドレスを
書き換えて転送

パケットの書換えと転送



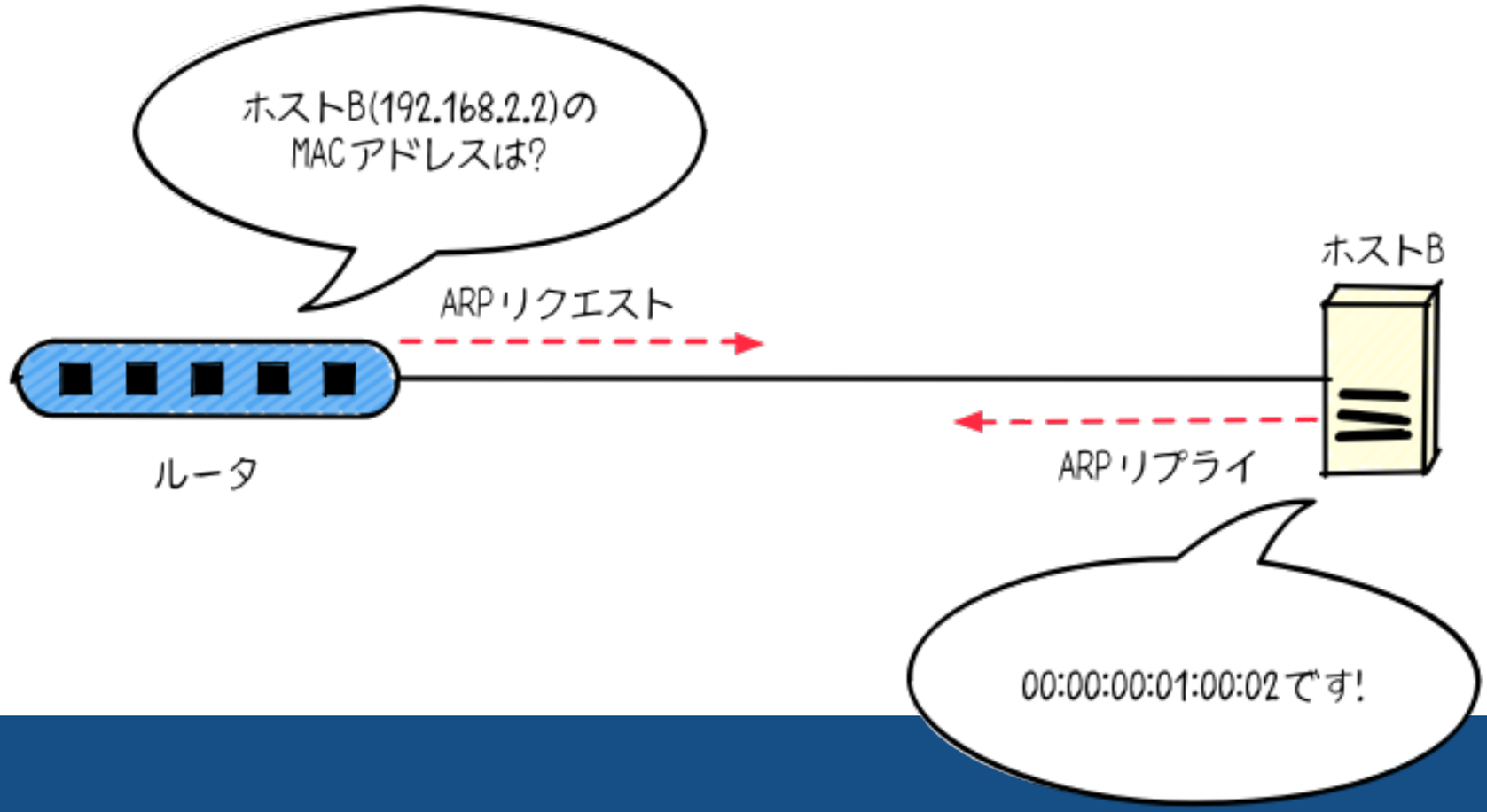
- パケットの宛先をホストBのMACアドレスに
- 送信元をルータのMACアドレスに書き換え

ルータのMACアドレス



- ルータがパケットを受け取るために、
  自分のMACアドレスを教える
- ARP (Address Resolution Protocol)

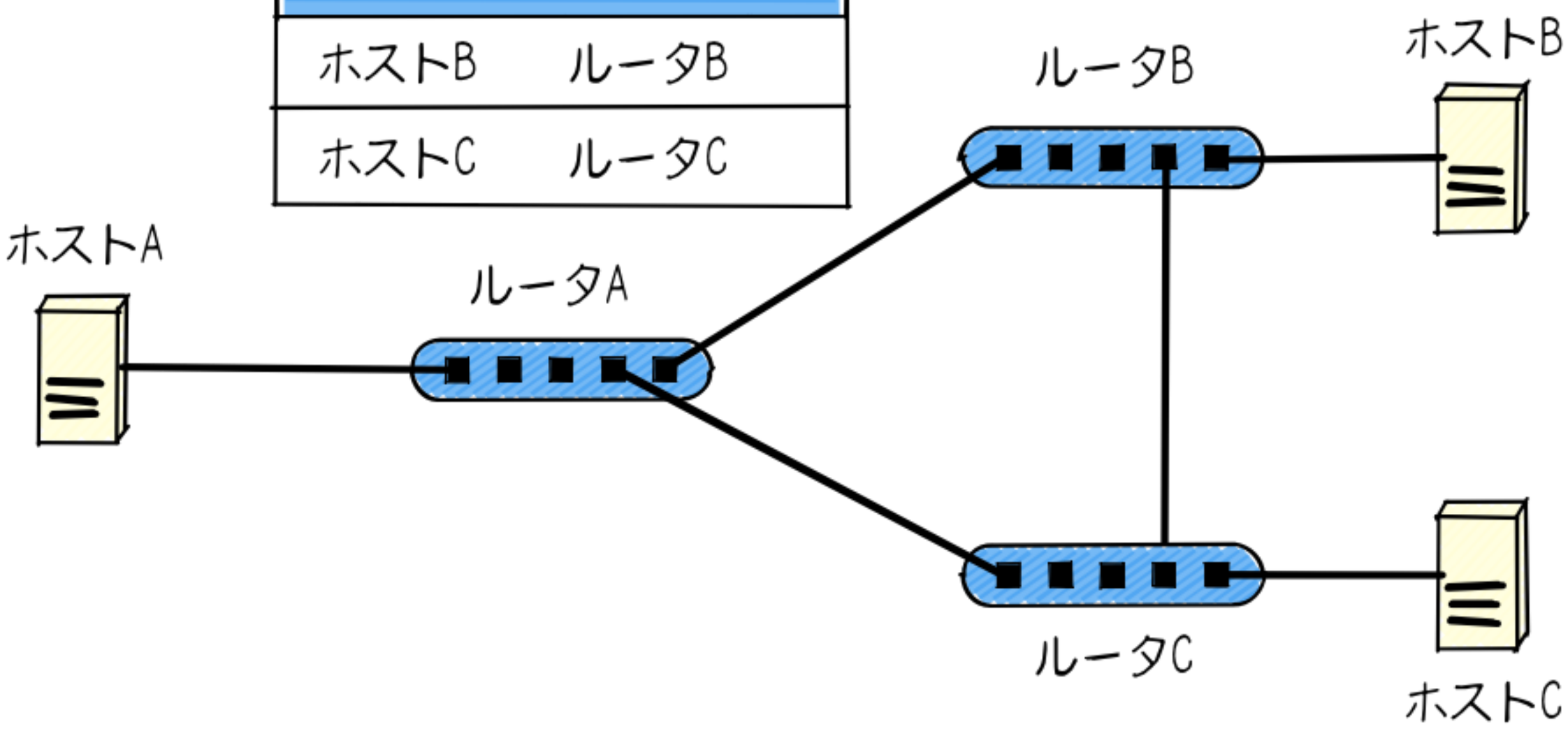
宛先のMACアドレス



- ルータがパケットを宛先に送るために、
  宛先ホストに ARP する
- 調べたMACアドレスはARPテーブルにキャッシュ

ルータを経由して転送

宛先	次の転送先
ホストB	ルータB
ホストC	ルータC



PacketInハンドラ

PacketIn ハンドラ

```
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
```

- ルータ宛のパケットか?
- どんなパケットか?

ルータ宛のパケットか?

```
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
```

- ブロードキャストだったら true
- パケットの宛先MACがルータインタフェースのMACアドレスだったら true

何がPacketInした?

```
  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 リクエスト
2. ARP リプライ
3. IPv4 パケット

PacketInハンドラ
→ARPの処理

ルータ(192.168.1.1)の
MACアドレスは?

ARPリクエスト

ルータ

ARPリプライ

ホストA

00:00:00:01:00:01です!

ホストB(192.168.2.2)の
MACアドレスは?

ホストB

ARPリクエスト

ルータ

ARPリプライ

00:00:00:01:00:02です!

ARP リクエストの処理

```
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
```

- ルータ宛のARPリクエスかどうか判定
- ルータ宛だったらARPリプライを送る

ARP リプライの処理

```
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
```

- 届いたARPリプライをARPテーブルへ
- 溜めておいたARP未解決パケットを送る

PacketInハンドラ
→IPパケットの処理

IPv4パケットの処理

```
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
```

1. パケットの転送が必要な場合
2. 宛先IPアドレスがルータでPingの場合
3. それ以外だった場合 (破棄)

転送するかどうか?

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

宛先IPアドレス !=
ルータインタフェースのIPアドレス

ping に応える

```
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. ICMPリクエストを読んで、
2. 宛先MACアドレスが分かる場合はICMPリプライを送る
3. 分からない場合は「後で送る」キューに溜める

PacketInハンドラ
→IPパケットの処理
→書換えと転送

パケット書換えと転送

```
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
```

パケット書換えと転送

```
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)
```

- MACアドレス書換えアクションを作って、普通にFlowMod & PacketOut

ARPが解決してから送る

```
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
```

「後で送る」の仕組みは?

ARPが解決してから送る

```
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. パケットを宛先IPアドレスごとのキューに溜める
2. 宛先のMACアドレスを解決するために
   ARPリクエストを送る

ARPが解決したとき

```
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
- 解決したMACアドレスで書換えて送る

まとめ

- ルータの基本的な仕組み
  - MACアドレスの書換え
  - ARPへの応答と解決
- ルーティングテーブル (後半で詳しく説明)