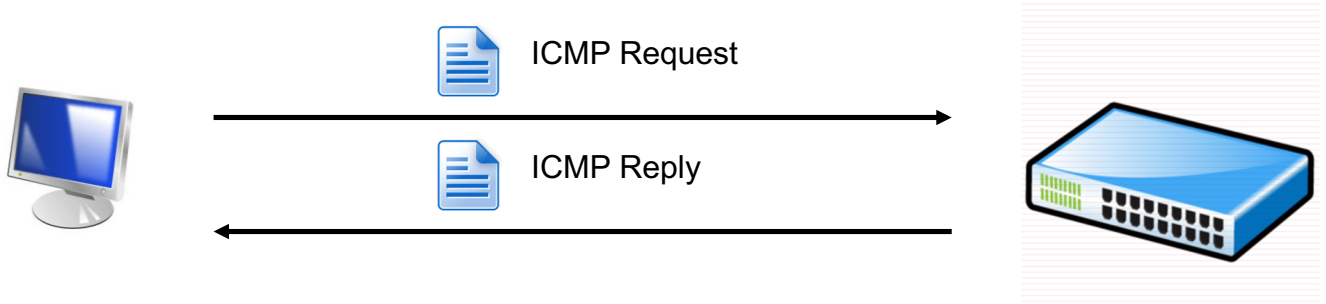
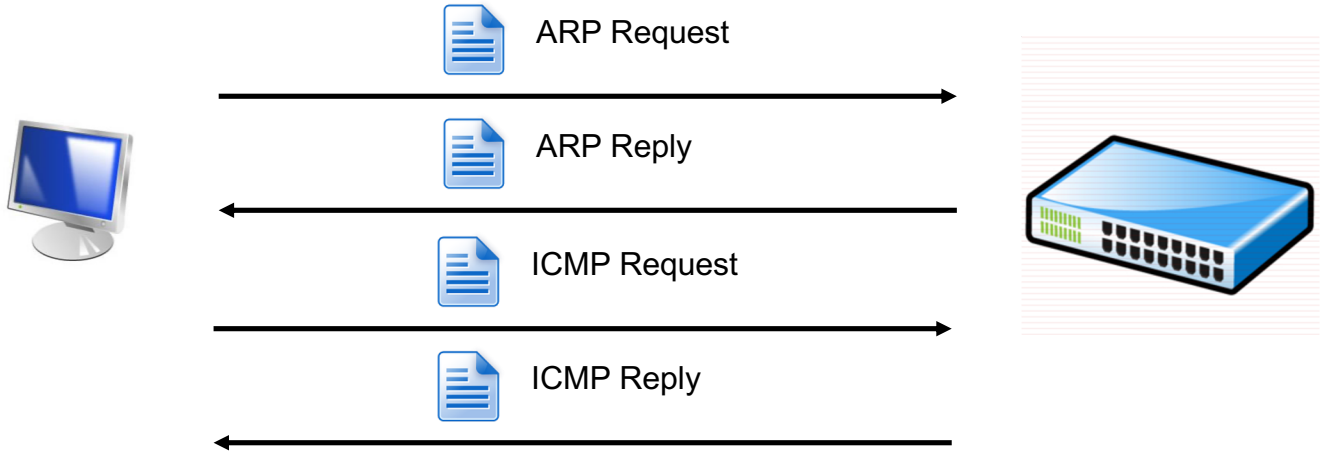
Virtual address

- Create a module that mimics the existence of Virtual hosts processing some traffic directed to a *virtual IPv4 address*.
- The controller has to process ICMP ping requests to a specific IP address (that does not physically exist) generating ICMP replies



Virtual address

- No new rules are installed
- To this aim, the controller has also to handle ARP requests. The controller has to reply to ARP requests aimed at the virtual IPv4 address



Mininet

- If the virtual address is an address outside the local network of hosts emulated in mininet an extra configuration on them is required (they do not have a gateway configured)
- Each host has to be configured to send out traffic for external networks on its NIC card as it is
  - h1 route add -net 0.0.0.0/32 dev h1-eth0

Virtual address

- Inside the receiver function the controller has to process ARP requests and ICMP echo requests for the virtual IPv4:

```
// Cast packet
OFPacketIn pi = (OFPacketIn) msg;

Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
                IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

// Dissect Packet included in Packet-In
IPacket pkt = eth.getPayload();

if (eth.isBroadcast() || eth.isMulticast()) {
     if (pkt instanceof ARP) {
     handleARPRequest(…); // Handle ARP requests for the virtual IP
     return Command.STOP;
     }
}
else {
    // We only care about packets which are sent to the virtual IP address
    IPv4 ip_pkt = (IPv4) pkt;

    if(ip_pkt.getDestinationAddress().compareTo(VIRTUAL_IP) == 0){
     handleIPPacket(…); // Handle IP packets towards the Virtual IP
     return Command.STOP;
    }
}
// Allow the next module to also process this OpenFlow message
return Command.CONTINUE;
```

ARP

IPv4

Virtual address - handleARPRequest

- In case the ARP request is directed to the virtual IP an ARP reply with a virtual MAC address has to be generated

```
// Double check that the payload is ARP
Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
     IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
if (! (eth.getPayload() instanceof ARP))
     return;
// Cast the ARP request
ARP arpRequest = (ARP) eth.getPayload();
// Generate ARP reply
IPacket arpReply = new Ethernet()
     .setSourceMACAddress(VIRTUAL_MAC)
     .setDestinationMACAddress(eth.getSourceMACAddress())
     .setEtherType(EthType.ARP)
     .setPriorityCode(eth.getPriorityCode())
     .setPayload(
          new ARP()
          .setHardwareType(ARP.HW_TYPE_ETHERNET)
          .setProtocolType(ARP.PROTO_TYPE_IP)
          .setHardwareAddressLength((byte) 6)
          .setProtocolAddressLength((byte) 4)
          .setOpCode(ARP.OP_REPLY)
          .setSenderHardwareAddress(VIRTUAL_MAC) // Set my MAC address
          .setSenderProtocolAddress(VIRTUAL_IP) // Set my IP address
          .setTargetHardwareAddress(arpRequest.getSenderHardwareAddress())
          .setTargetProtocolAddress(arpRequest.getSenderProtocolAddress())
     );
```

Virtual address - handleARPRequest

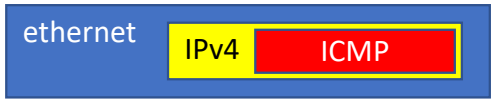
- Create Packet-Out
- Create list of actions
- Set the ARP reply as payload of Packet-Out
- Send it out

```
// Initialize a packet out
OFPacketOut.Builder pob = sw.getOFFactory().buildPacketOut();
pob.setBufferId(OFBufferId.NO_BUFFER);
pob.setInPort(OFPort.ANY);

// Set the output action
OFActionOutput.Builder actionBuilder = sw.getOFFactory().actions().buildOutput();
OFPort inPort = pi.getMatch().get(MatchField.IN_PORT);
actionBuilder.setPort(inPort);
pob.setActions(Collections.singletonList((OFAction) actionBuilder.build()));

// Set the ARP reply as packet data
byte[] packetData = arpReply.serialize();
pob.setData(packetData);

// Send packet
sw.write(pob.build());
```

Virtual address - handleIPPacket

- Check Packet-In
- Create the ICMP reply

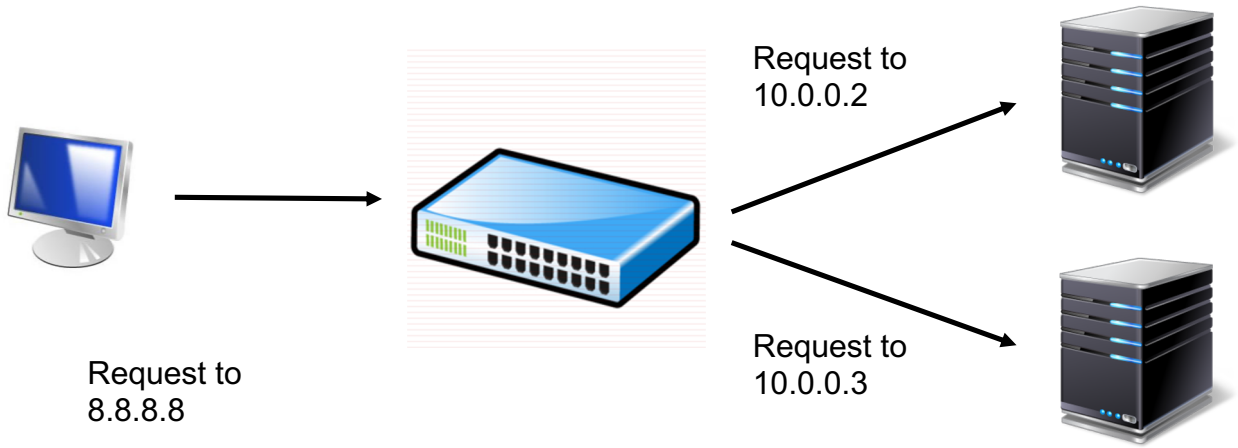
```
// Cast the IP packet
IPv4 ipv4 = (IPv4) eth.getPayload();

// Check that the IP is actually an ICMP request
if (! (ipv4.getPayload() instanceof ICMP))
      return;

// Cast to ICMP packet
ICMP icmpRequest = (ICMP) ipv4.getPayload();
// Generate ICMP reply
IPacket icmpReply = new Ethernet()
      .setSourceMACAddress(VIRTUAL_MAC)
      .setDestinationMACAddress(eth.getSourceMACAddress())
      .setEtherType(EthType.IPv4)
      .setPriorityCode(eth.getPriorityCode())
      .setPayload(
            new IPv4()
            .setProtocol(IpProtocol.ICMP)
            .setDestinationAddress(ipv4.getSourceAddress())
            .setSourceAddress(VIRTUAL_IP)
            .setTtl((byte)64)
            .setProtocol(IpProtocol.IPv4)
            // Set the same payload included in the request
            .setPayload(
                  new ICMP()
                  .setIcmpType(ICMP.ECHO_REPLY)
                  .setIcmpCode(icmpRequest.getIcmpCode())
                              .setPayload(icmpRequest.getPayload())
            )
      );
```



Send response…

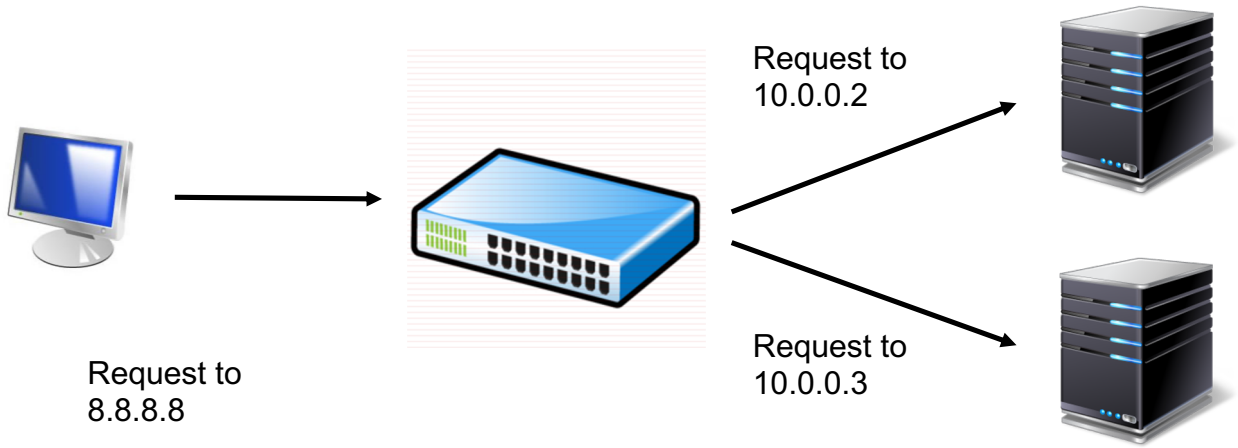
Load Balancer

- Create a module that acts as a load balancer for a pool of servers
- A virtual IP address (e.g. 8.8.8.8) is set as the public IP address of a group of servers offering the same service (e.g. a web server with the same page)
- The module has to control the switch to dispatch the requests directed to the virtual IP to one of the IP of a real server from the pool.
- The real server from the pool selected to provide the service must change every 20 seconds in order to balance the load



Load Balancer

- ***The switch has to handle ARP requests.***
- ***This operation has to be performed transparently***, i.e. changing the destination IP and MAC addresses on the requests (with the addresses of the real server selected) and the source IP and MAC addresses on the responses (with the addresses of the virtual IP and MAC)
- ***New rules have to be installed on the switch***



TODO list

Load Balancer

```
// Create a flow table modification message to add a rule
        OFFlowMod.Builder fmb = sw.getOFFactory().buildFlowAdd();

        fmb.setIdleTimeout(IDLE_TIMEOUT);
        fmb.setHardTimeout(HARD_TIMEOUT); // Set as hard timeout the period to change the current server
        fmb.setBufferId(OFBufferId.NO_BUFFER);
        fmb.setOutPort(OFPort.CONTROLLER);
        fmb.setCookie(U64.of(0));
        fmb.setPriority(FlowModUtils.PRIORITY_MAX);

// Create the match for the new rule (incoming IPv4 traffic directed to the load balancer)
        Match.Builder mb = sw.getOFFactory().buildMatch();
        mb.setExact(MatchField.ETH_TYPE, EthType.IPv4)
        .setExact(MatchField.IPV4_DST, LOAD_BALANCER_IP)
        .setExact(MatchField.ETH_DST, LOAD_BALANCER_MAC);
```

Load Balancer

```
// Create the list of actions associated with a match
        OFActions actions = sw.getOFFactory().actions();
        ArrayList<OFAction> actionList = new ArrayList<OFAction>();
        OFOxms oxms = sw.getOFFactory().oxms();
        // Set as new MAC destination the MAC address of the current server
        OFActionSetField setDlDst = actions.buildSetField()
                .setField(
                    oxms.buildEthDst()
                    .setValue(MacAddress.of(SERVERS_MAC[last_server]))
                    .build()
                )
                .build();
        actionList.add(setDlDst);
        // Set as new IP destination the IP address of the current server
        OFActionSetField setNwDst = actions.buildSetField()
                .setField(
                    oxms.buildIpv4Dst()
                    .setValue(IPv4Address.of(SERVERS_IP[last_server]))
                    .build()
                ).build();
        actionList.add(setNwDst);
        // Set as output port the port of the current server
        OFActionOutput output = actions.buildOutput()
                .setMaxLen(0xFFffFFff)
                .setPort(OFPort.of(SERVERS_PORT[last_server]))
                .build();
        actionList.add(output);

// Send out the mod message
        fmb.setActions(actionList);
        fmb.setMatch(mb.build());

        sw.write(fmb.build());
```

Complex networks