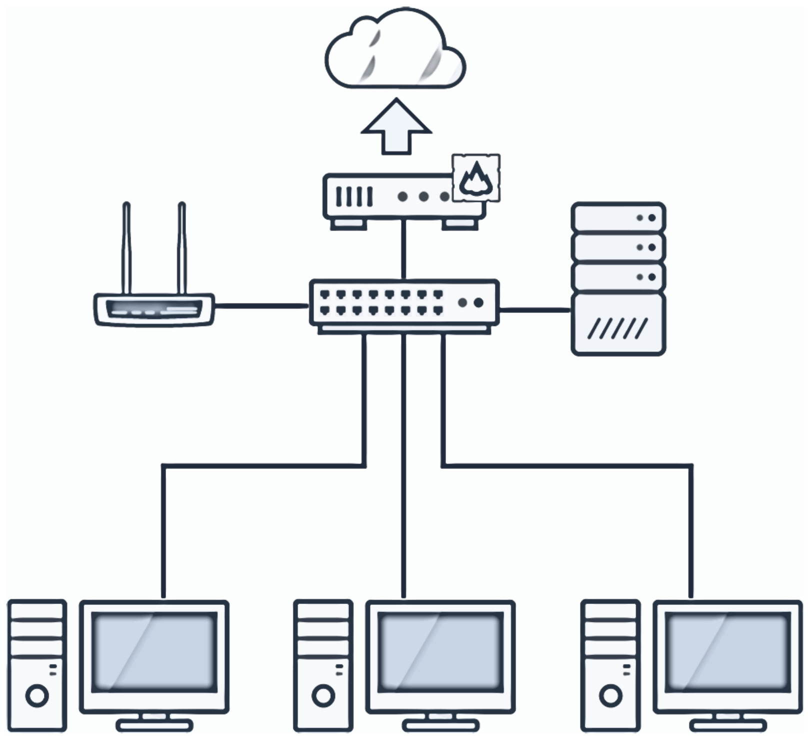
Asynchronous Message Delivery System

Biagio Cornacchia
Matteo Abaterusso

Overview

- A server that **subscribes** to the system obtains a **virtual IP** and **virtual MAC**
- When the server is online, all client requests to that virtual IP are delivered **synchronously**
- When the server is offline, the system **stores** all requests destinated to it
- When the server **comes back online**, packets stored inside the controller are immediately sent out to it
- A server can subscribe/unsubscribe or change its status using the **Restful APIs** exposed by the system



Design

Restful APIs

Subscription

http://<CONTROLLER_IP>:8080/amd/server/subscription/json

- POST
- DELETE

Status

http://<CONTROLLER_IP>:8080/amd/server/status/json

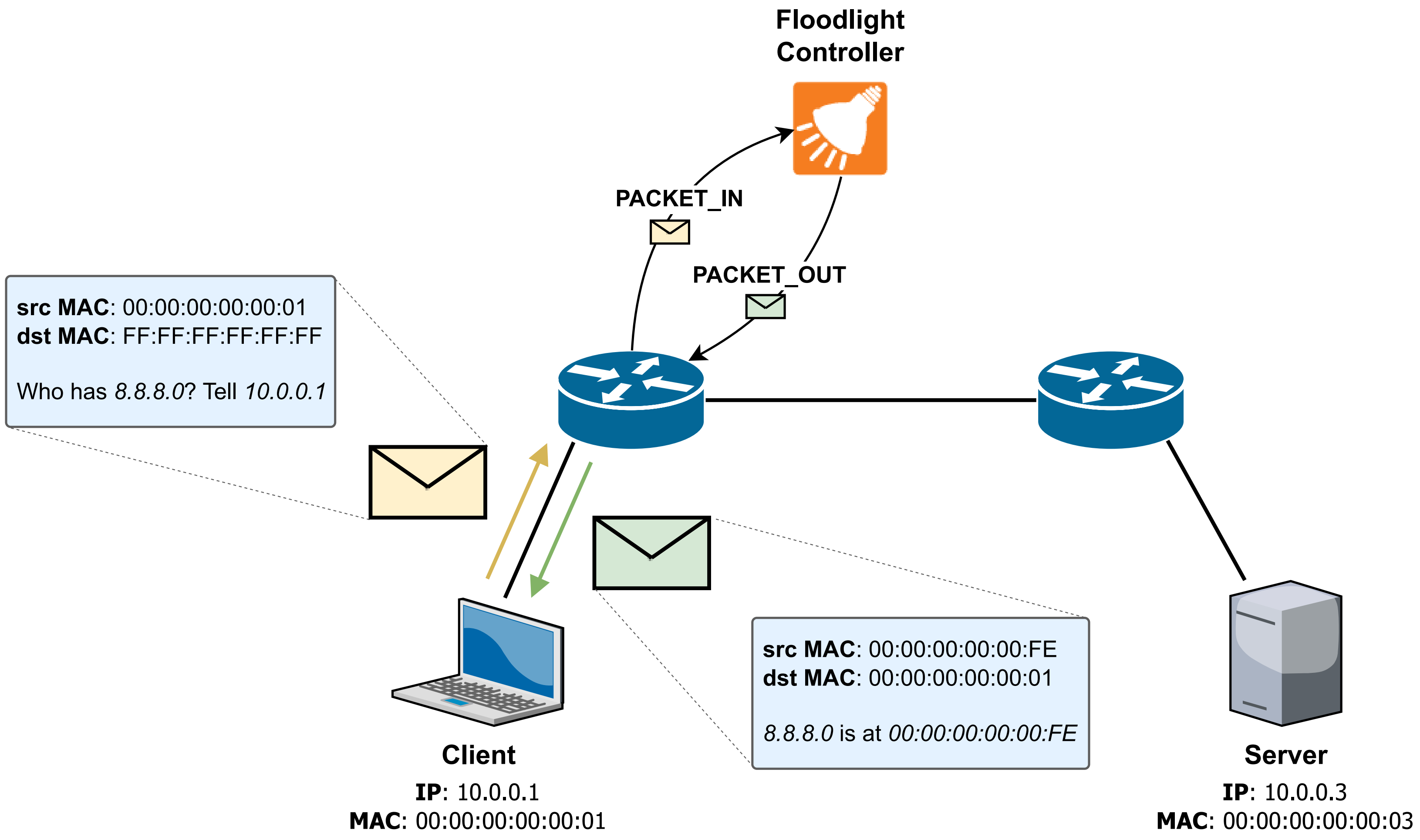
- PUT

Info

http://<CONTROLLER_IP>:8080/amd/server/info/json

- GET

ARP Requests



src MAC: 00:00:00:00:00:01
dst MAC: FF:FF:FF:FF:FF:FF

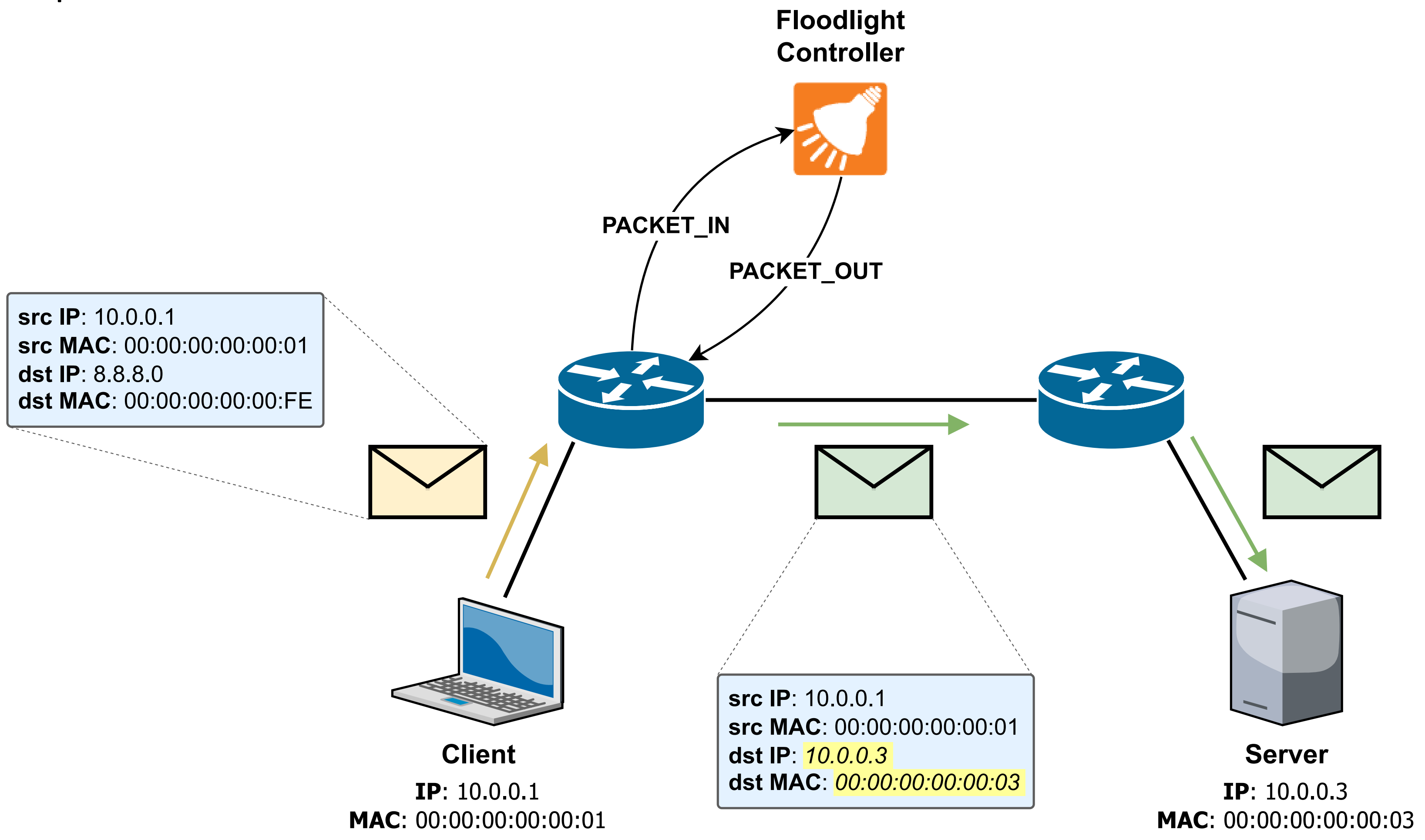
Who has *8.8.8.0*? Tell *10.0.0.1*

src MAC: 00:00:00:00:00:FE
dst MAC: 00:00:00:00:00:01

8.8.8.0 is at *00:00:00:00:00:FE*

IPv4 Requests (Server online)

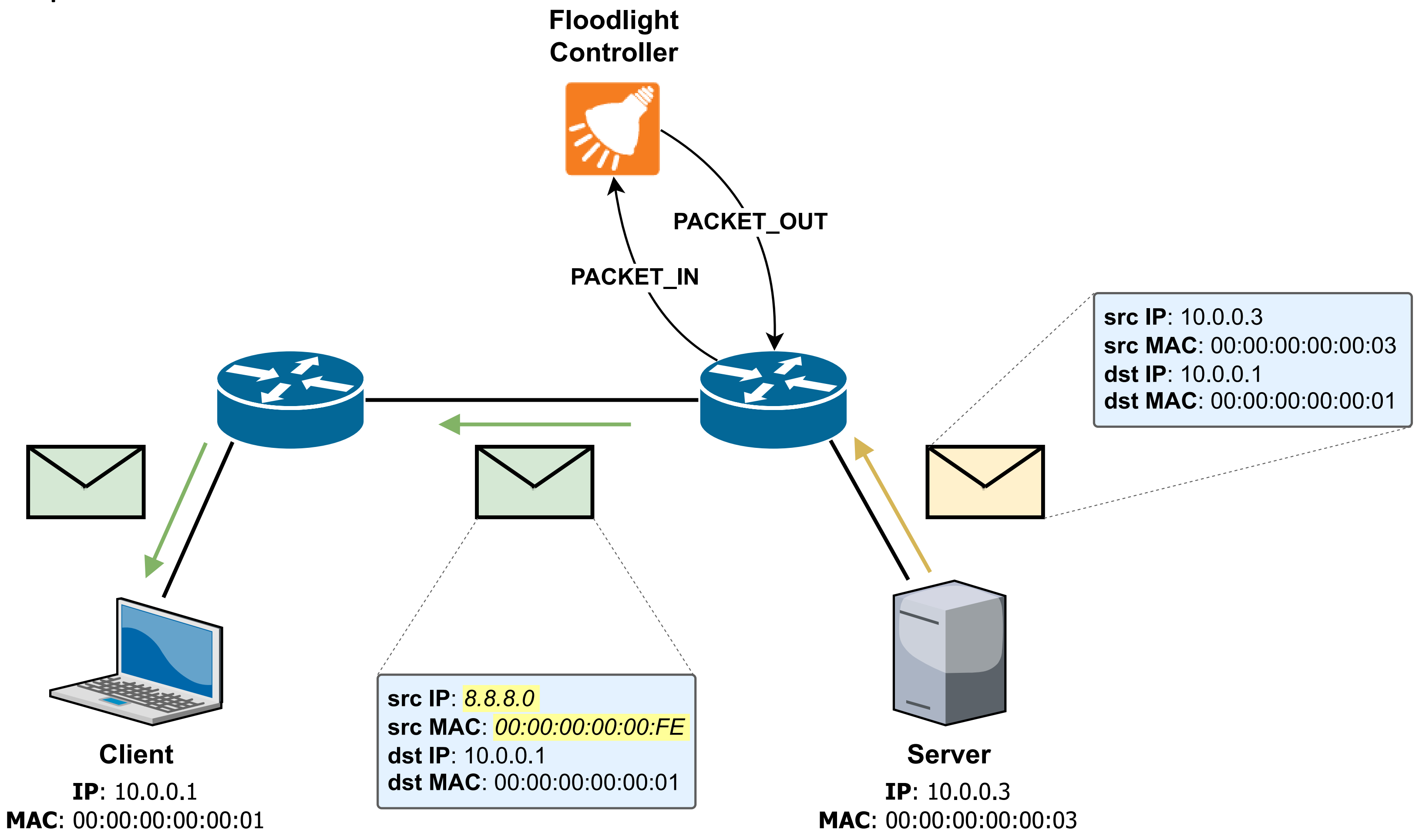
- **Client** to **Server** path



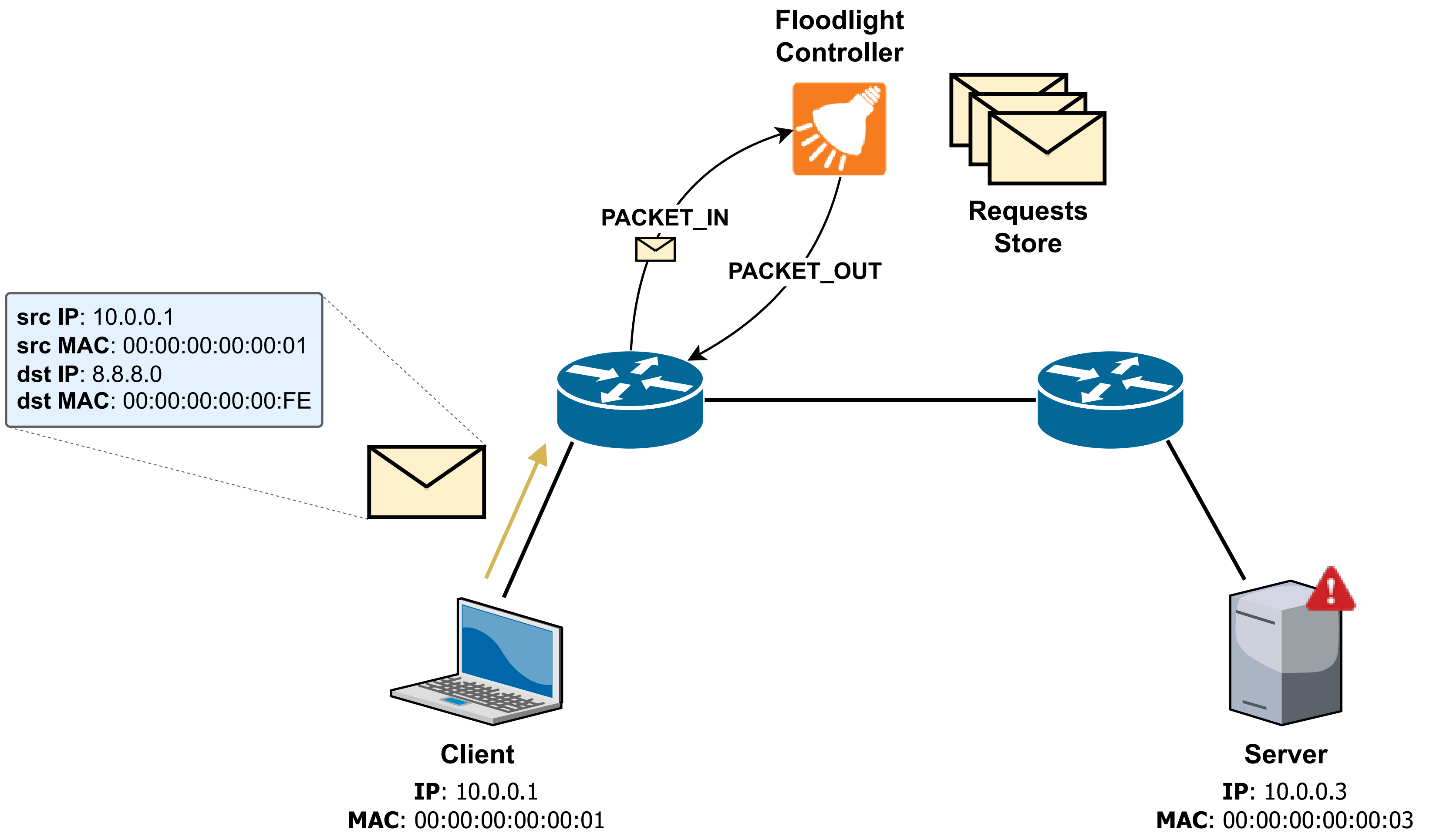
Floodlight Controller
PACKET_IN
PACKET_OUT
src IP: 10.0.0.1
src MAC: 00:00:00:00:00:01
dst IP: 8.8.8.0
dst MAC: 00:00:00:00:00:FE
src IP: 10.0.0.1
src MAC: 00:00:00:00:00:01
dst IP: *10.0.0.3*
dst MAC: *00:00:00:00:00:03*
Client
IP: 10.0.0.1
MAC: 00:00:00:00:00:01
Server
IP: 10.0.0.3
MAC: 00:00:00:00:00:03

IPv4 Requests (Server online)

- **Server** to **Client** path

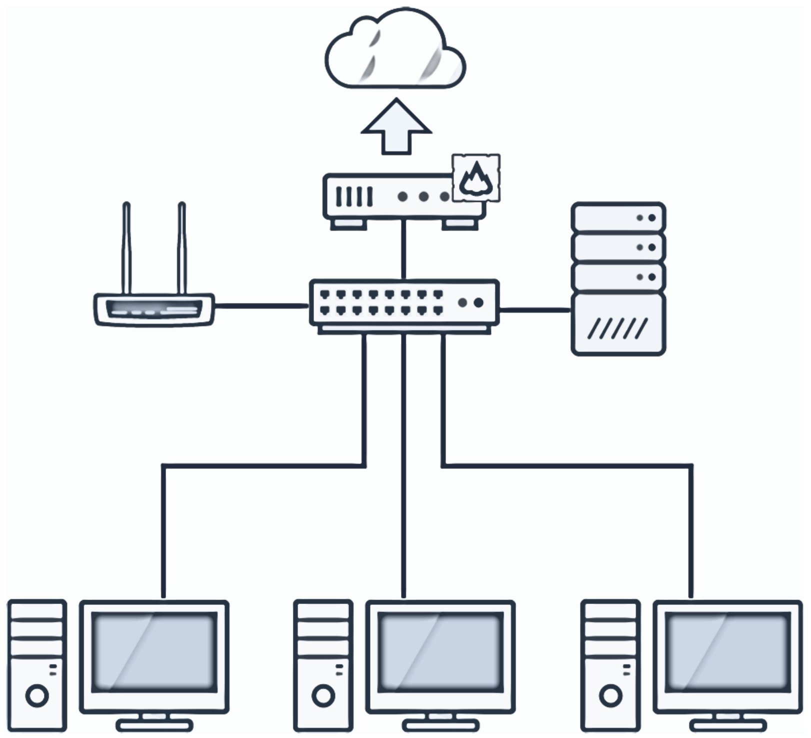

Floodlight Controller
PACKET_OUT
PACKET_IN
src IP: 10.0.0.3
src MAC: 00:00:00:00:00:03
dst IP: 10.0.0.1
dst MAC: 00:00:00:00:00:01
src IP: *8.8.8.0*
src MAC: *00:00:00:00:00:FE*
dst IP: 10.0.0.1
dst MAC: 00:00:00:00:00:01
Client
IP: 10.0.0.1
MAC: 00:00:00:00:00:01
Server
IP: 10.0.0.3
MAC: 00:00:00:00:00:03

IPv4 Requests (Server offline)



Flow Entries

- The **soft timeout** and **hard timeout** establish the lifetime of the flow entries on a switch.
- This will cause a **transition time** during which packets will be lost. So, there is a **trade-off** on the timeout choice.
- The chosen approach is to **actively intervene** when the server status changes by removing the flow entries related to that server.
- The fact that translation flow entries are installed only on **access switches** is exploited to interact with a reasonable number of switches.



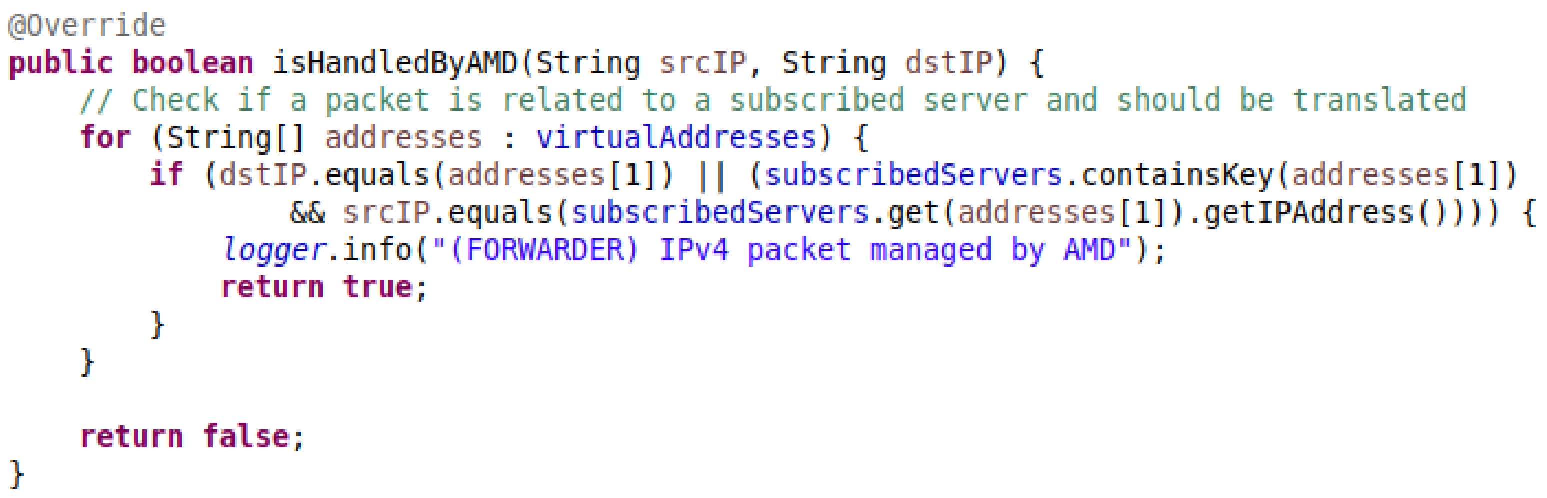
Implementation

Forwarding Module Problem

- In the Floodlight pipeline the **Forwarding** module is located before the **AMD** module.
- The **default behavior** of the Forwarding module is that if the destination address of the packet is **unknown** then a **flooding** action is applied, otherwise the packet is **directly forwarded**.
- This results in having **duplicated packets** in the network.

Forwarding Module Solution

- In the AMD module

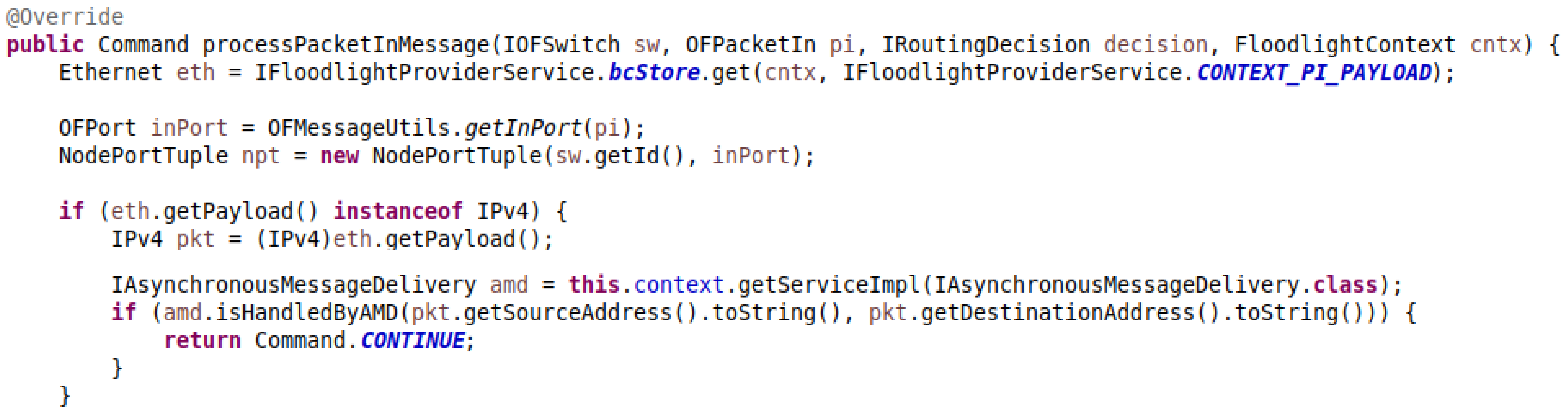


```java
@Override
public boolean isHandledByAMD(String srcIP, String dstIP) {
    // Check if a packet is related to a subscribed server and should be translated
    for (String[] addresses : virtualAddresses) {
        if (dstIP.equals(addresses[1]) || (subscribedServers.containsKey(addresses[1])
                && srcIP.equals(subscribedServers.get(addresses[1]).getIPAddress()))) {
            logger.info("(FORWARDER) IPv4 packet managed by AMD");
            return true;
        }
    }

    return false;
}
```

Forwarding Module Solution

- In the processPacketInMessage method of the Forwarding class



```java
@Override
public Command processPacketInMessage(IOFSwitch sw, OFPacketIn pi, IRoutingDecision decision, FloodlightContext cntx) {
    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx, IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

    OFPort inPort = OFMessageUtils.getInPort(pi);
    NodePortTuple npt = new NodePortTuple(sw.getId(), inPort);

    if (eth.getPayload() instanceof IPv4) {
        IPv4 pkt = (IPv4)eth.getPayload();

        IAsynchronousMessageDelivery amd = this.context.getServiceImpl(IAsynchronousMessageDelivery.class);
        if (amd.isHandledByAMD(pkt.getSourceAddress().toString(), pkt.getDestinationAddress().toString())) {
            return Command.CONTINUE;
        }
    }
```

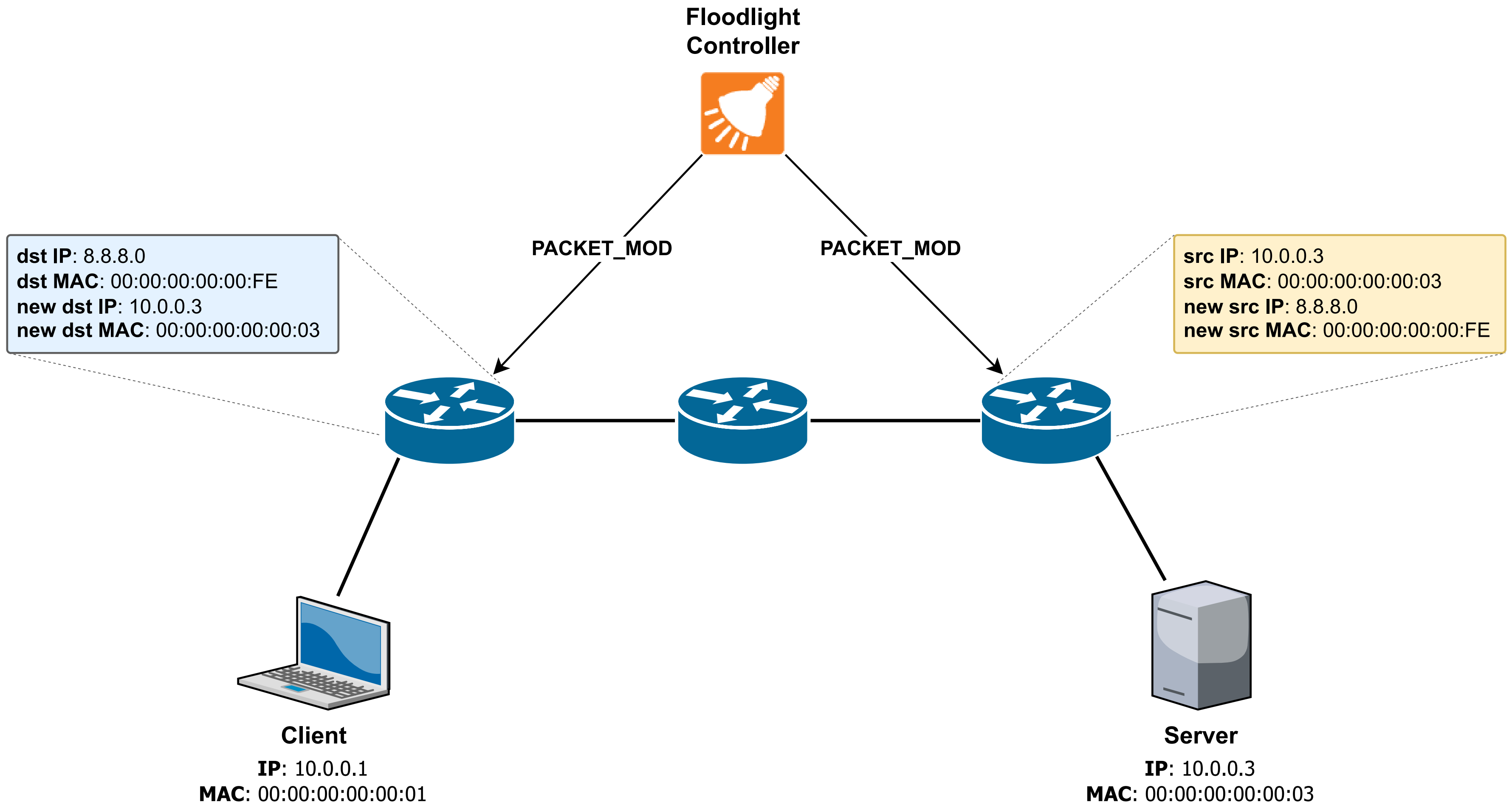
OVS Security Policy Problem

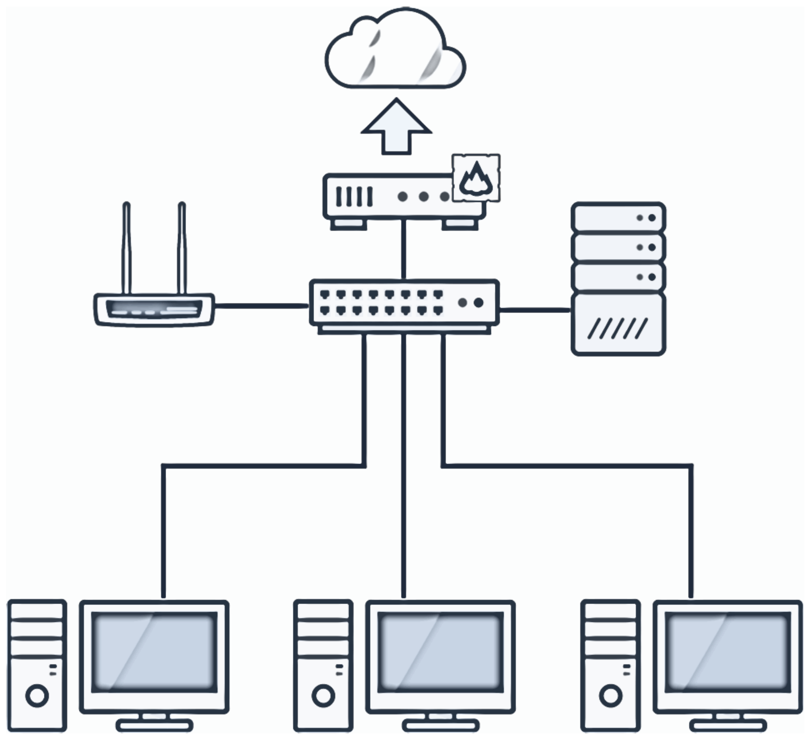
By default, the **Open vSwitch** implements a security policy that **drops** packets with an **external source MAC address**.

The parameters specified in the flow mod are the following:

- **priority** = 1
- **dl src** = VIRTUAL MAC
- **action** = normal

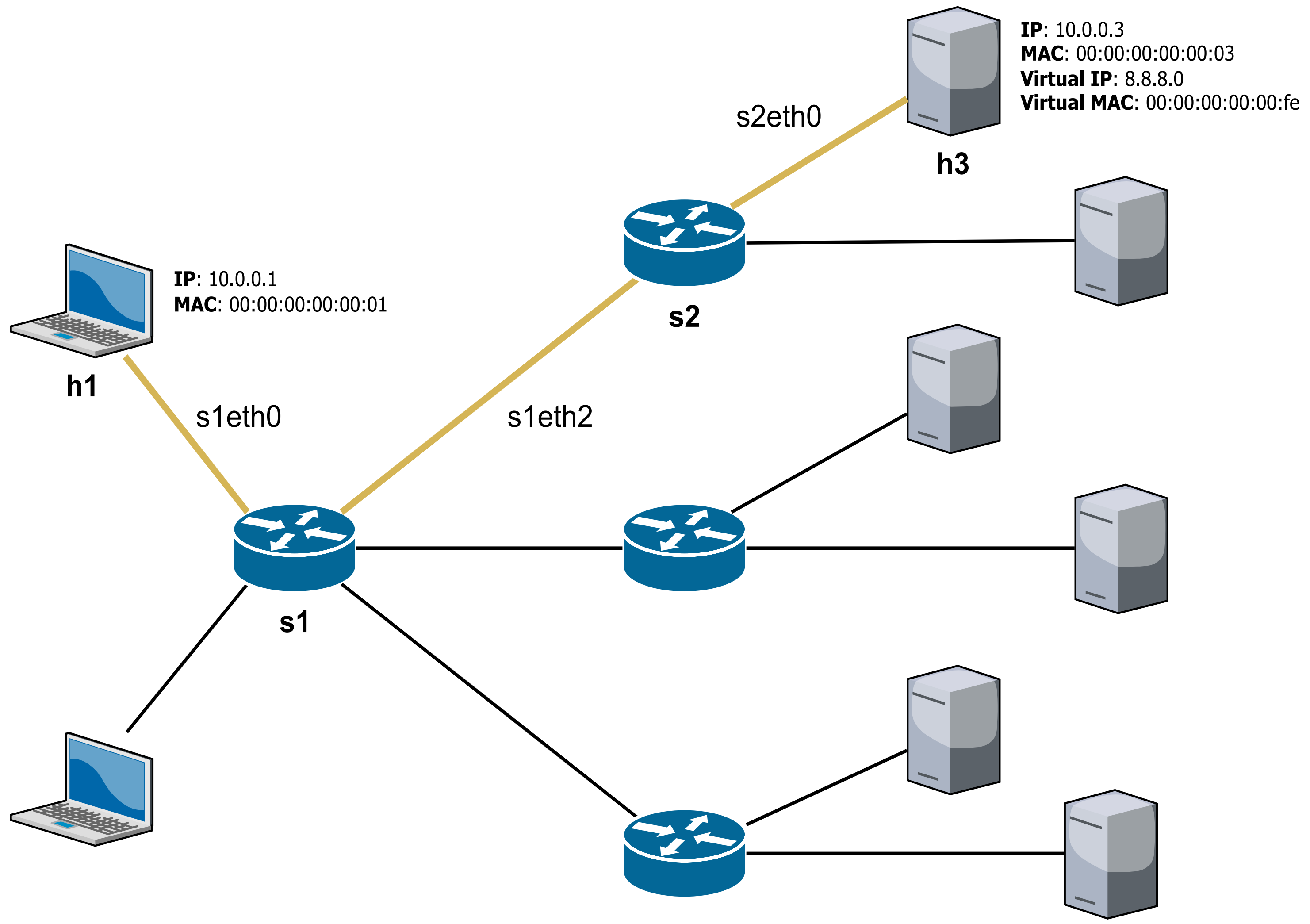
Flow Mod Optimization


Floodlight Controller
PACKET_MOD
PACKET_MOD
dst IP: 8.8.8.0
dst MAC: 00:00:00:00:00:FE
new dst IP: 10.0.0.3
new dst MAC: 00:00:00:00:00:03
src IP: 10.0.0.3
src MAC: 00:00:00:00:00:03
new src IP: 8.8.8.0
new src MAC: 00:00:00:00:00:FE
Client
IP: 10.0.0.1
MAC: 00:00:00:00:00:01
Server
IP: 10.0.0.3
MAC: 00:00:00:00:00:03

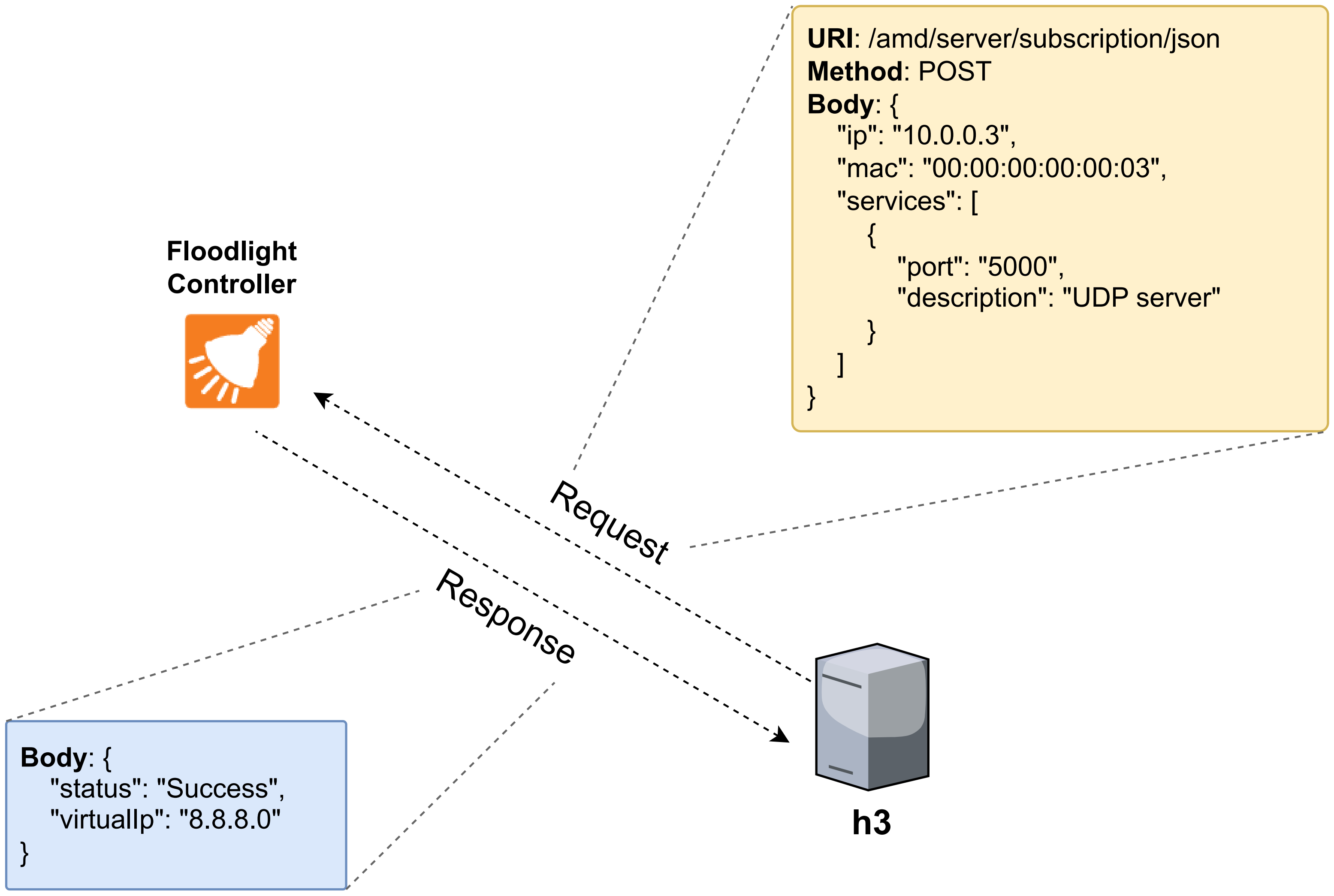


Testing

Testing Network



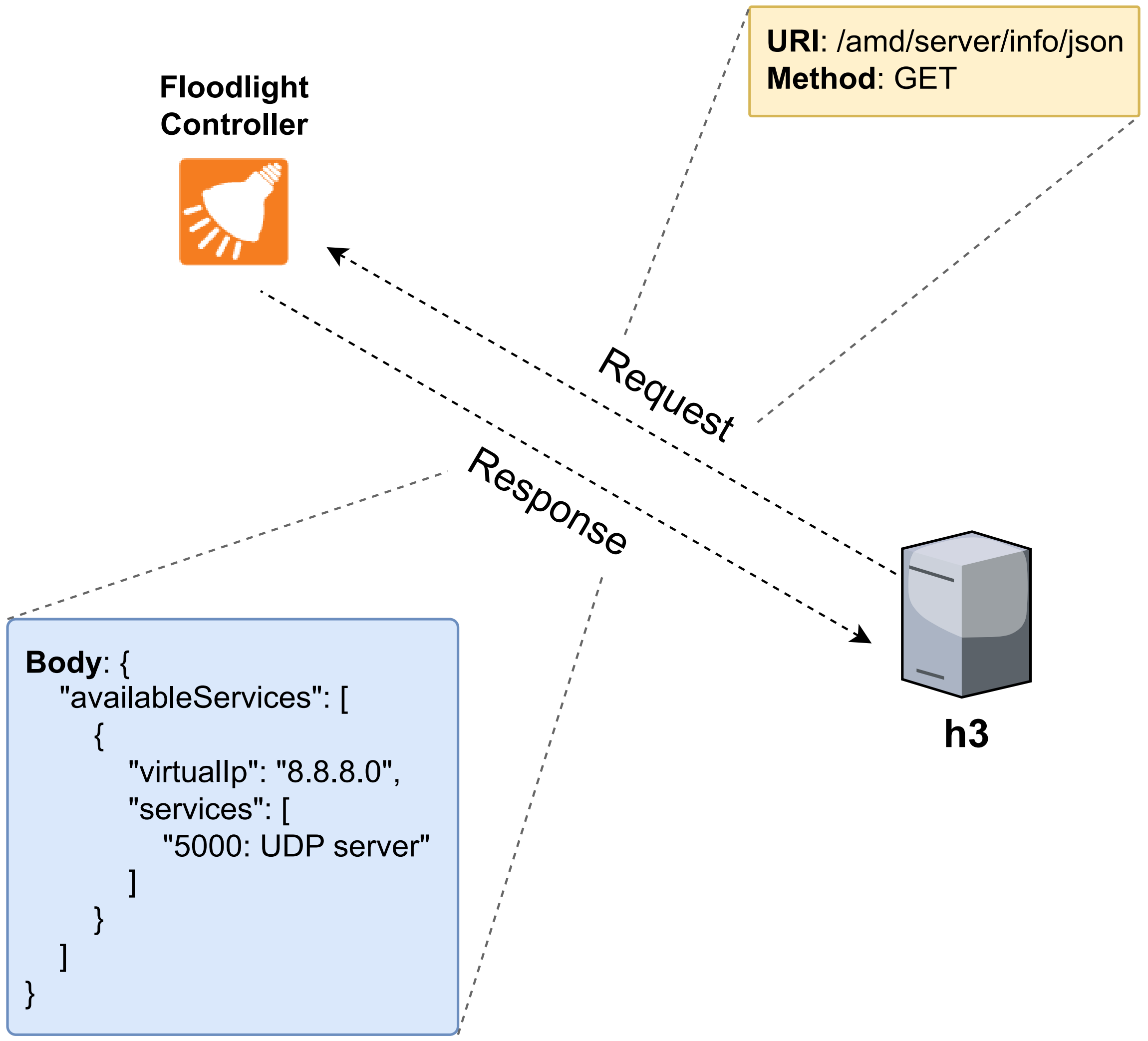
Server Subscription



URI: /amd/server/subscription/json
Method: POST
Body:
```
{
    "ip": "10.0.0.3",
    "mac": "00:00:00:00:00:03",
    "services": [
      {
        "port": "5000",
        "description": "UDP server"
      }
    ]
}
```

Body:
```
{
    "status": "Success",
    "virtualIp": "8.8.8.0"
}
```

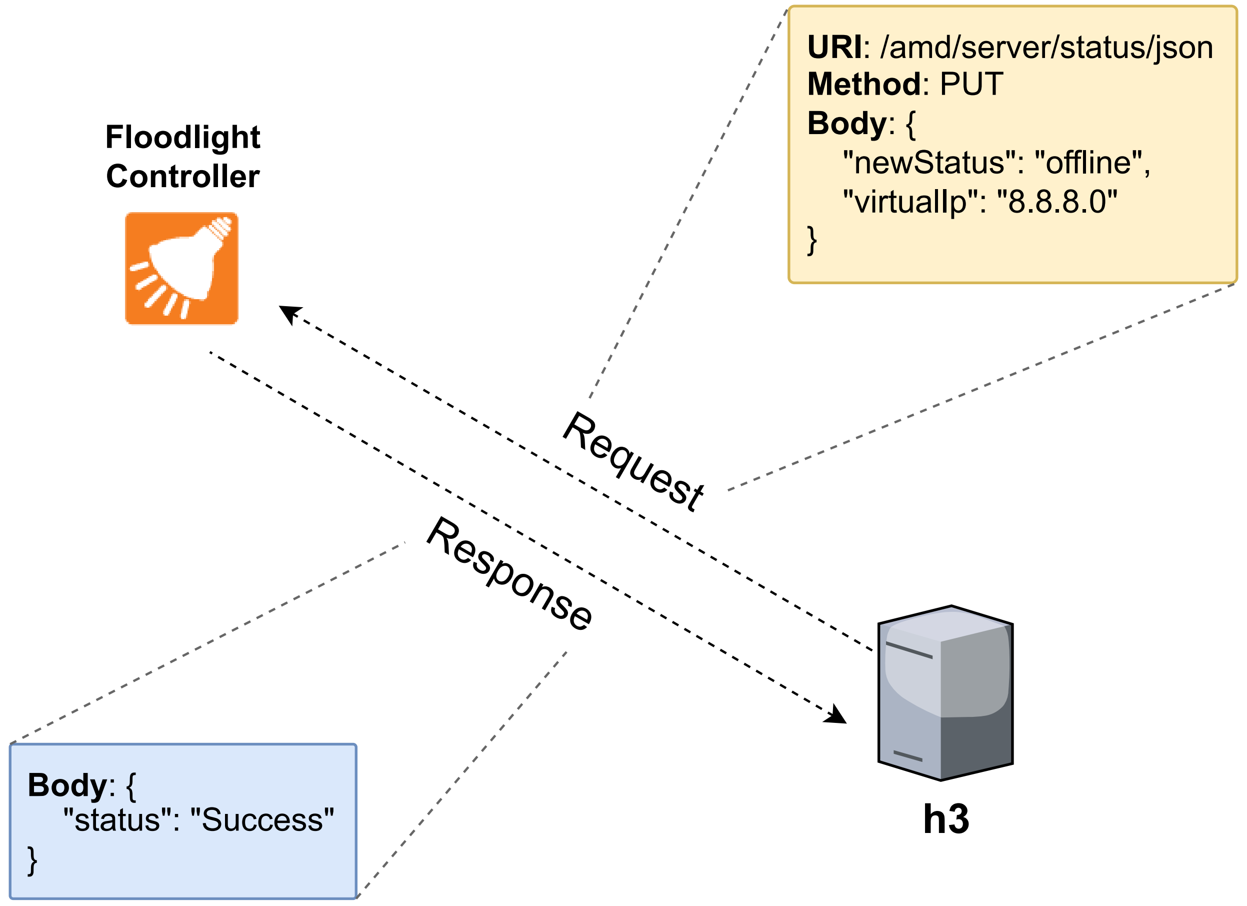
Server Info



URI: /amd/server/info/json
Method: GET

```
Body: {
    "availableServices": [
        {
            "virtualIp": "8.8.8.0",
            "services": [
                "5000: UDP server"
            ]
        }
    ]
}
```

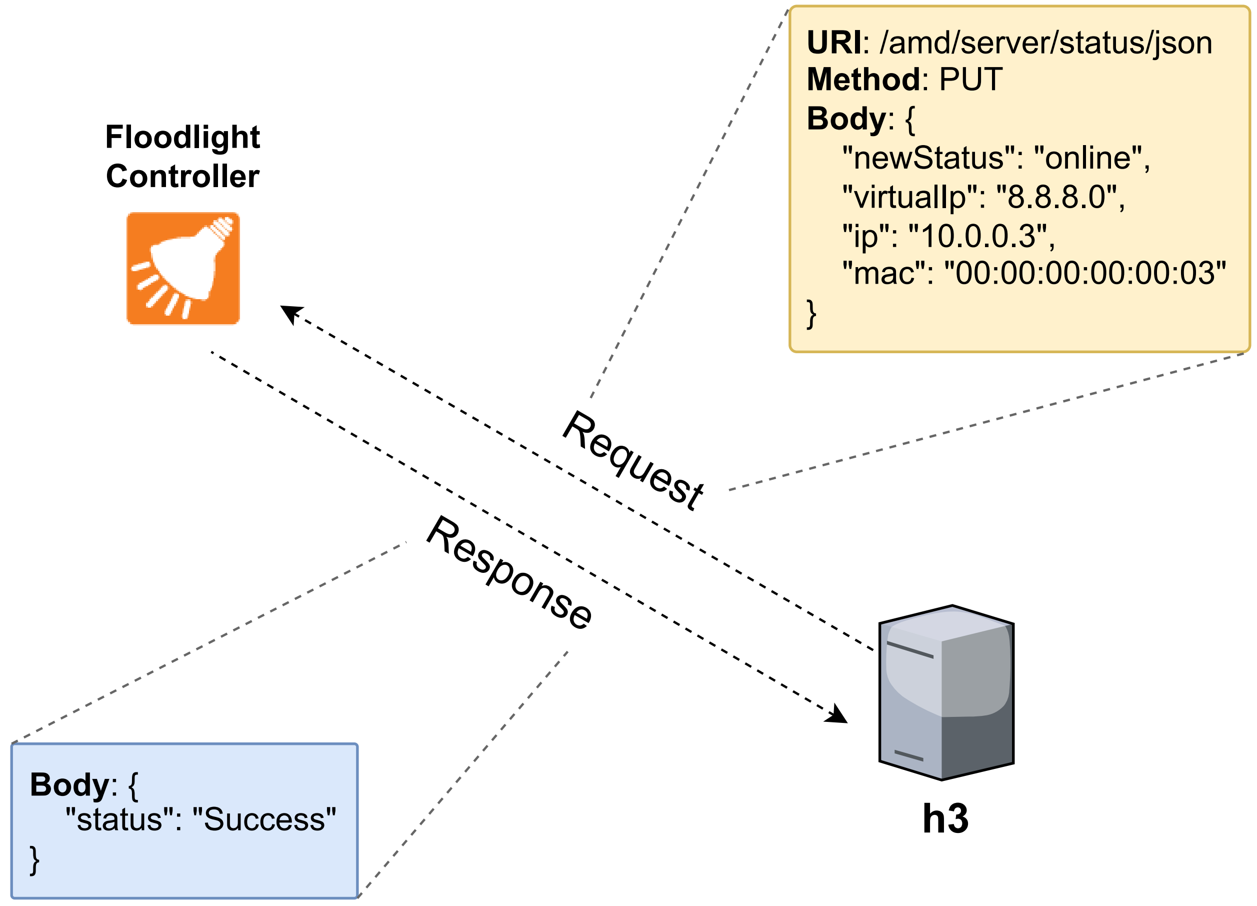
Server Status Update



URI: /amd/server/status/json
Method: PUT
Body: {
    "newStatus": "offline",
    "virtualIp": "8.8.8.0"
}

Body: {
    "status": "Success"
}

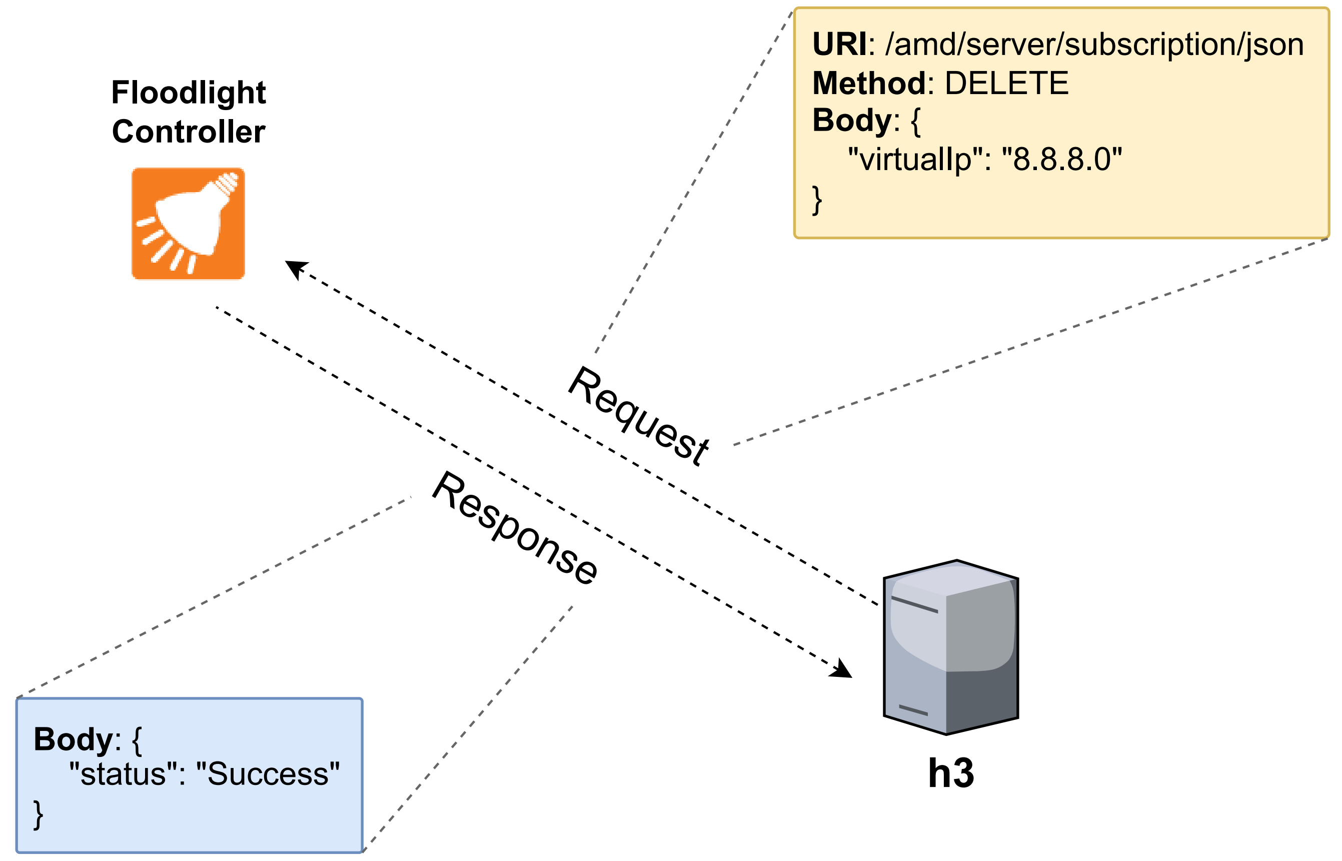
Server Status Update



URI: /amd/server/status/json
Method: PUT
Body: {
    "newStatus": "online",
    "virtualIp": "8.8.8.0",
    "ip": "10.0.0.3",
    "mac": "00:00:00:00:00:03"
}

Body: {
    "status": "Success"
}

Server Unsubscription



URI: /amd/server/subscription/json
Method: DELETE
Body: {
    "virtualIp": "8.8.8.0"
}

Body: {
    "status": "Success"
}