FORTH
Foundation for Research & Technology - Hellas

Assignment 3: Simple Load Balancer on a P4 Programmable Switch

Dr. Vasileios Kotronis
vkotronis[at]ics[dot]forth[dot]gr

HY-436 – Software Defined Networks (SDN)
Autumn-Winter 2020

Open-source tools you will need

- On VM:
  - `Bmv2`: a P4 software switch (P4 target) [1]
  - `p4c`: the reference P4 compiler [2]
  - `PI`: an implementation framework for a P4Runtime server [3]
  - `mininet`: a lightweight network emulation environment [4]

- On host machine:
  - Host virtualization software (e.g., `Virtualbox`) [5]
  - `ssh`, `scp`, etc. (e.g., via Putty [6], Filezilla [7], etc.)
  - Your favorite editor (e.g., Sublime Text [8])

[1] https://github.com/p4lang/behavioral-model
[2] https://github.com/p4lang/p4c
[3] https://github.com/p4lang/PI
[4] http://mininet.org
[5] https://www.virtualbox.org
[6] https://www.putty.org/
[7] https://filezilla-project.org/
[8] https://www.sublimetext.com/

The VM: Download and import in VBox

https://tinyurl.com/hy436P4vm

The VM: Specs

Name	P4 Tutorial 2019-04-25_1
# CPU cores	2
RAM	2GB
HDD	16GB (expandable to 50GB)
OS	Ubuntu Server 16.04 with lightweight GUI (64-bit)
Network adapters	1 NAT (communication with Internet)
Installed packages	P4 dependencies (models, architectures), Mininet, Wireshark
Credentials (user/pass)	p4/p4

The VM: Useful commands

- **Find the VM's host-only IP address (sth like 192.168.56.X):**
  `vm> sudo ifconfig`
  (if no IP address on host-only interface, run):
  `vm> sudo dhclient <host_only_interface>`

- **Connect via SSH to the VM:**
  `your_machine> ssh (-X) p4@<vm_ip>`

- **Copy a file from your host to the VM:**
  `your_machine> scp (-r) <your_file>`
  `p4@<vm_ip>:/home/p4/<dest_file_path>`

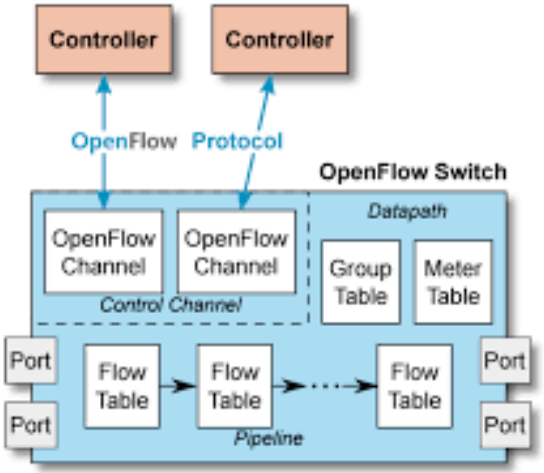
Note: to talk to the VM from your host OS:

- Settings → Network → Adapter 2
- Add a host-only network adapter
- Enable both the option "Enable Network Adapter" and "Cable Connected"

P4 and programmable data planes: Recap

- P4: *program the switch on the fly*, not just its flow tables
  - Control, customize, introduce new features, manage resources efficiently, ...
- Define the API/protocol between the controller and your switch(es)
- Apply software engineering practices in switch hardware manipulation
- Reconfigure on the field, even during line-rate forwarding
- Achieve protocol independence: packet (headers) are just bit sequences
  - You can edit, cut, append subsequences
  - Treat protocols as bitstring manipulations!
- Achieve target independence: whatever the hardware, your logic is portable
  - Compiler deals with the details of the generated/loaded binary

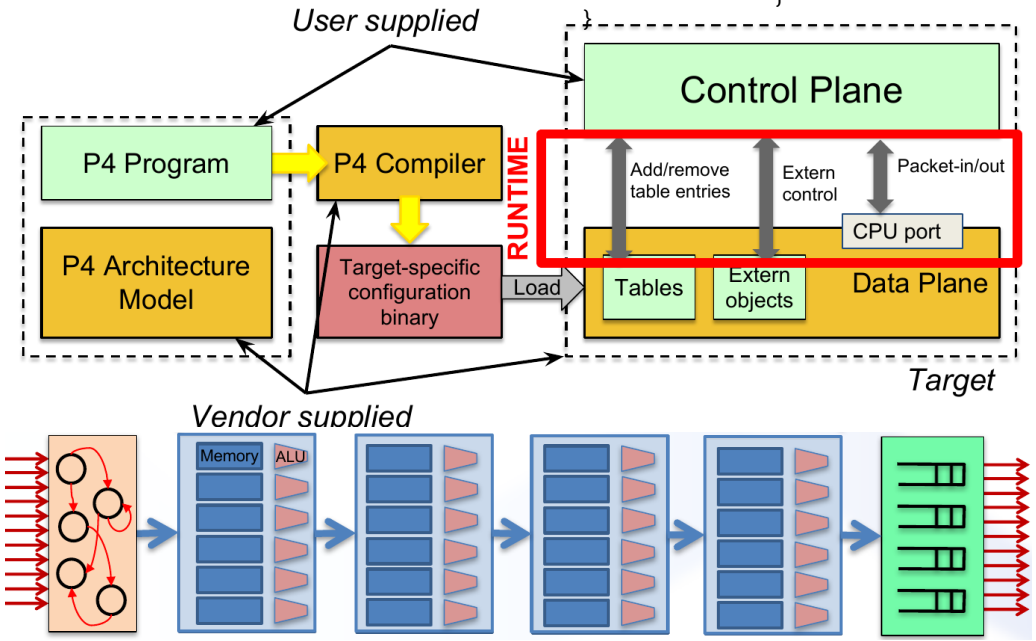
P4 and programmable data planes: Recap



- Protocol independence
  - Specify how switch processes pkts
- Target independence
  - On different HW/SW switches
- Field reconfigurability
  - Even after deployment!

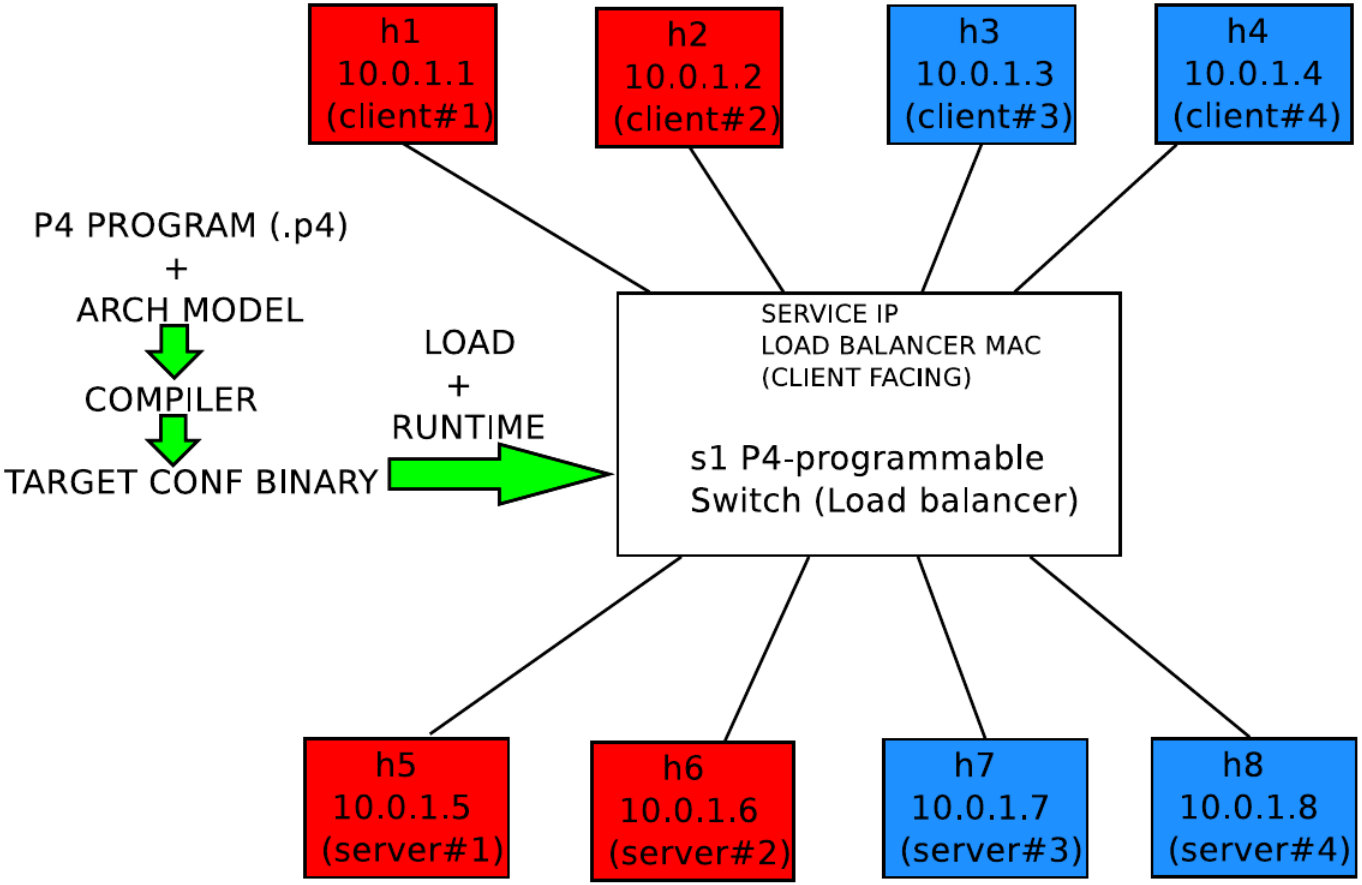
```
table routing {
                key = { ipv4.dstAddr : lpm;
}
                actions = { drop; route; }
                size : 2048;
}
control ingress() {
                apply {

                routing.apply();
                }
}
```

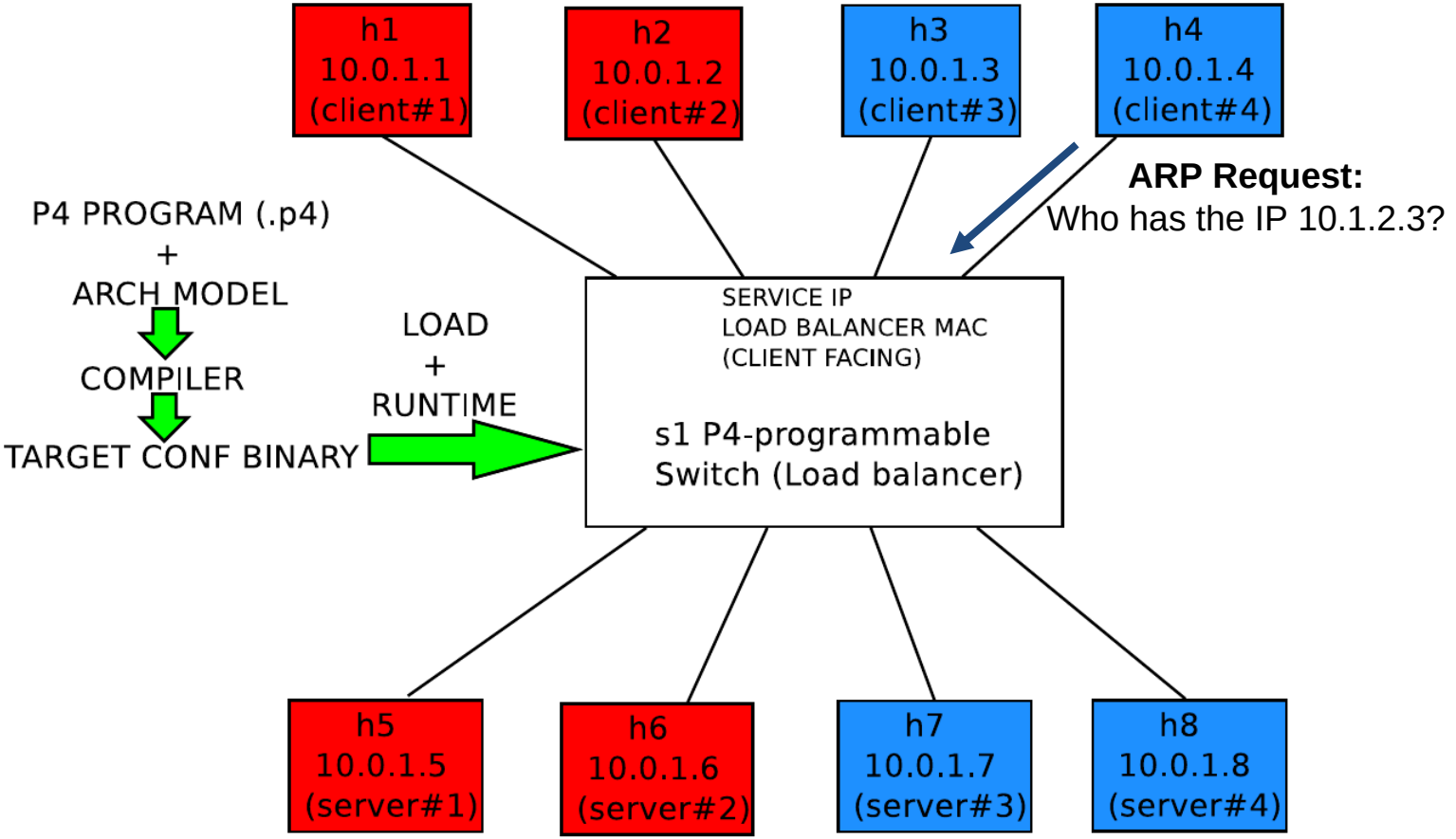


https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf

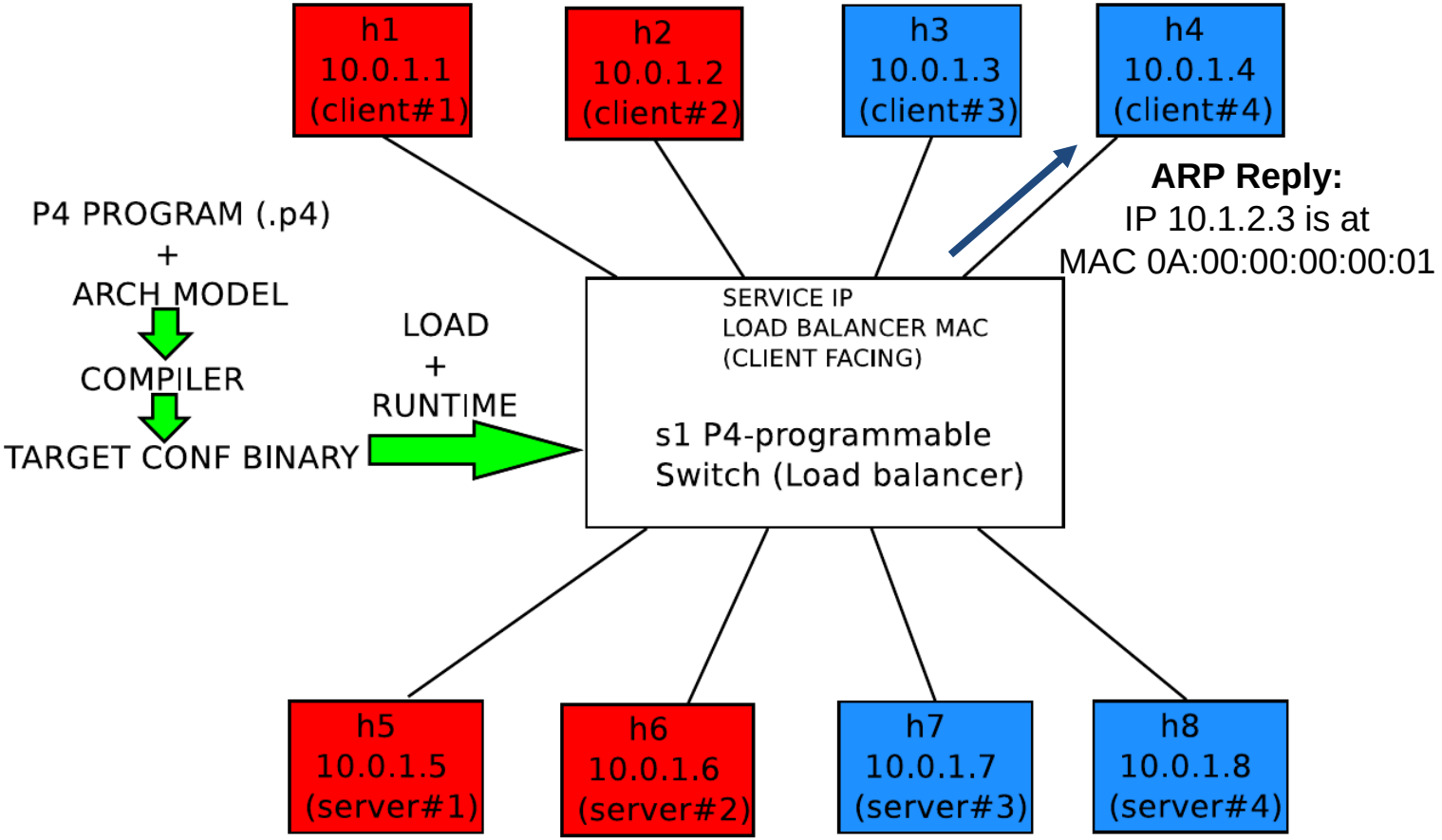
Setup


h1
10.0.1.1
(client#1)
h2
10.0.1.2
(client#2)
h3
10.0.1.3
(client#3)
h4
10.0.1.4
(client#4)
P4 PROGRAM (.p4)
+
ARCH MODEL
COMPILER
TARGET CONF BINARY
LOAD
+
RUNTIME
SERVICE IP
LOAD BALANCER MAC
(CLIENT FACING)
s1 P4-programmable
Switch (Load balancer)
h5
10.0.1.5
(server#1)
h6
10.0.1.6
(server#2)
h7
10.0.1.7
(server#3)
h8
10.0.1.8
(server#4)

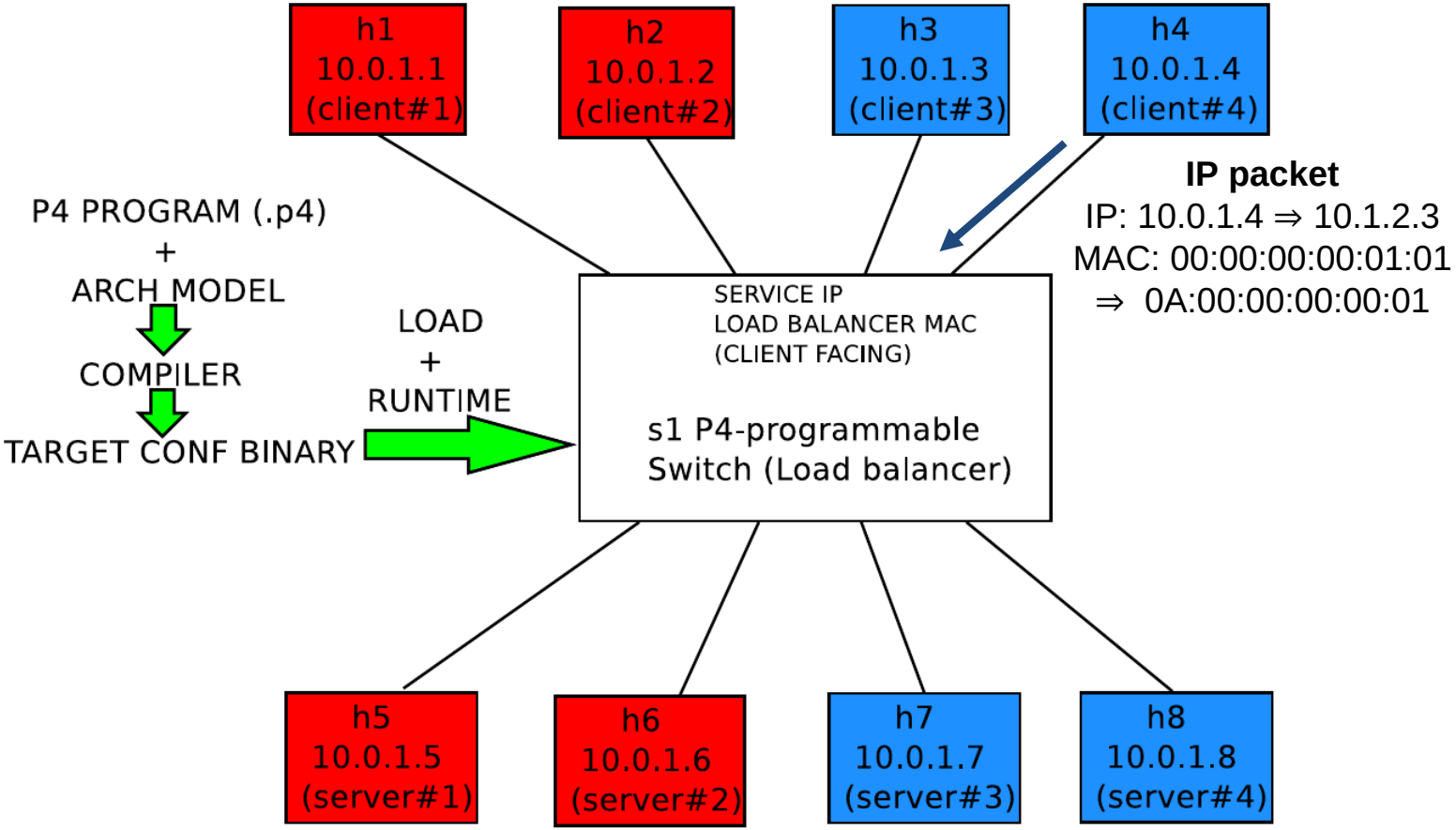
ARP requests: clients to LB



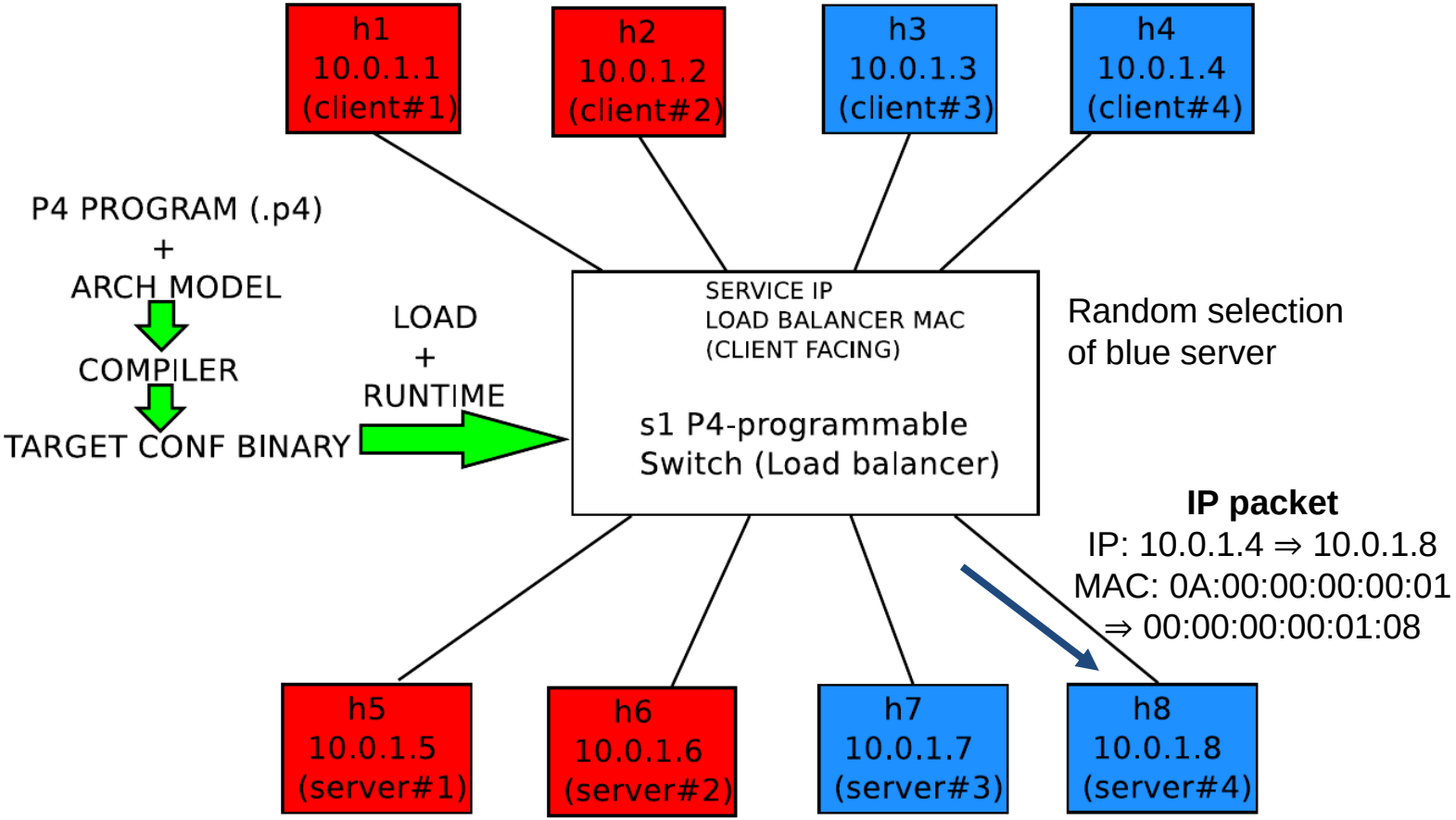
ARP replies: LB to clients



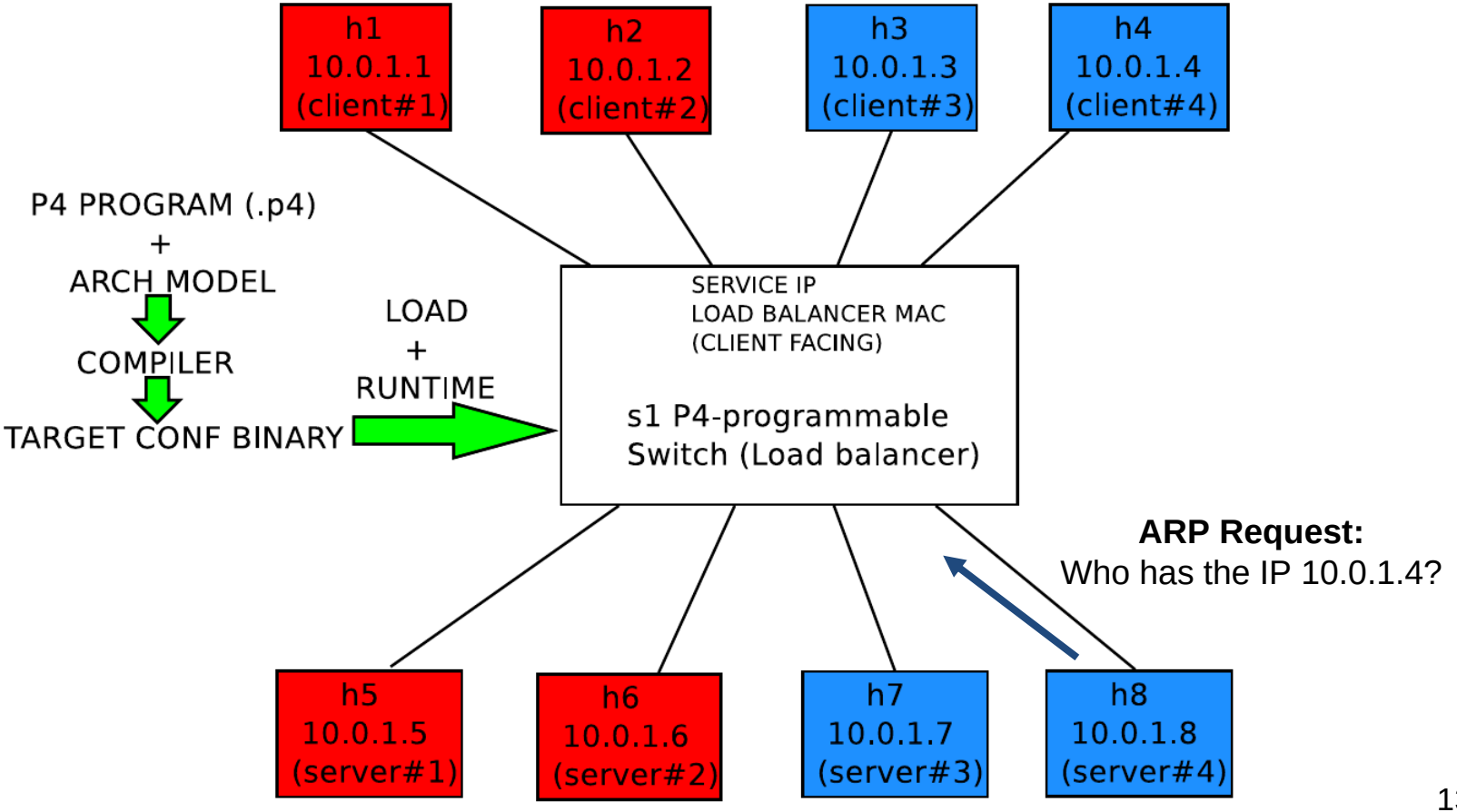
IP traffic: client to server request



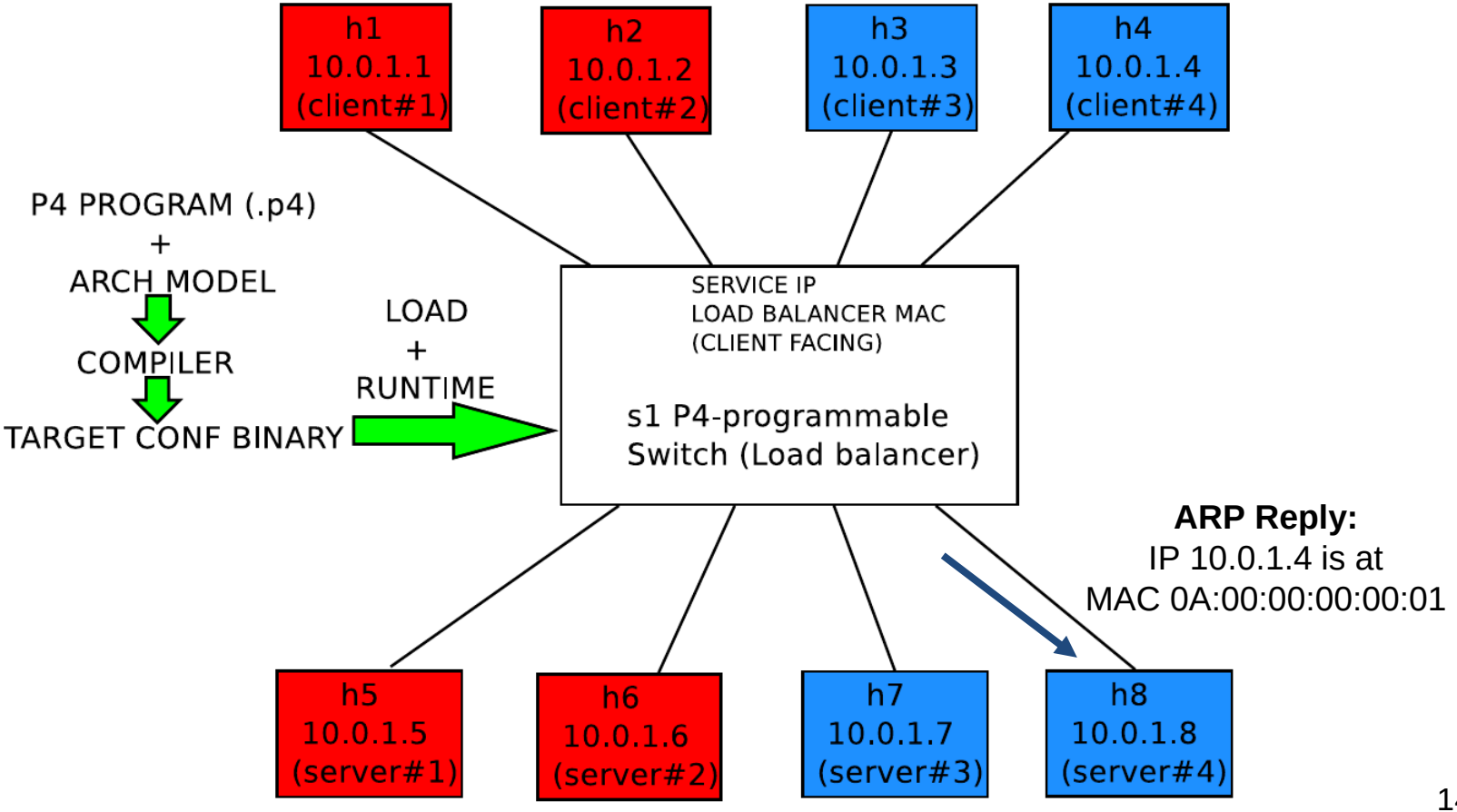
IP traffic: client to server request



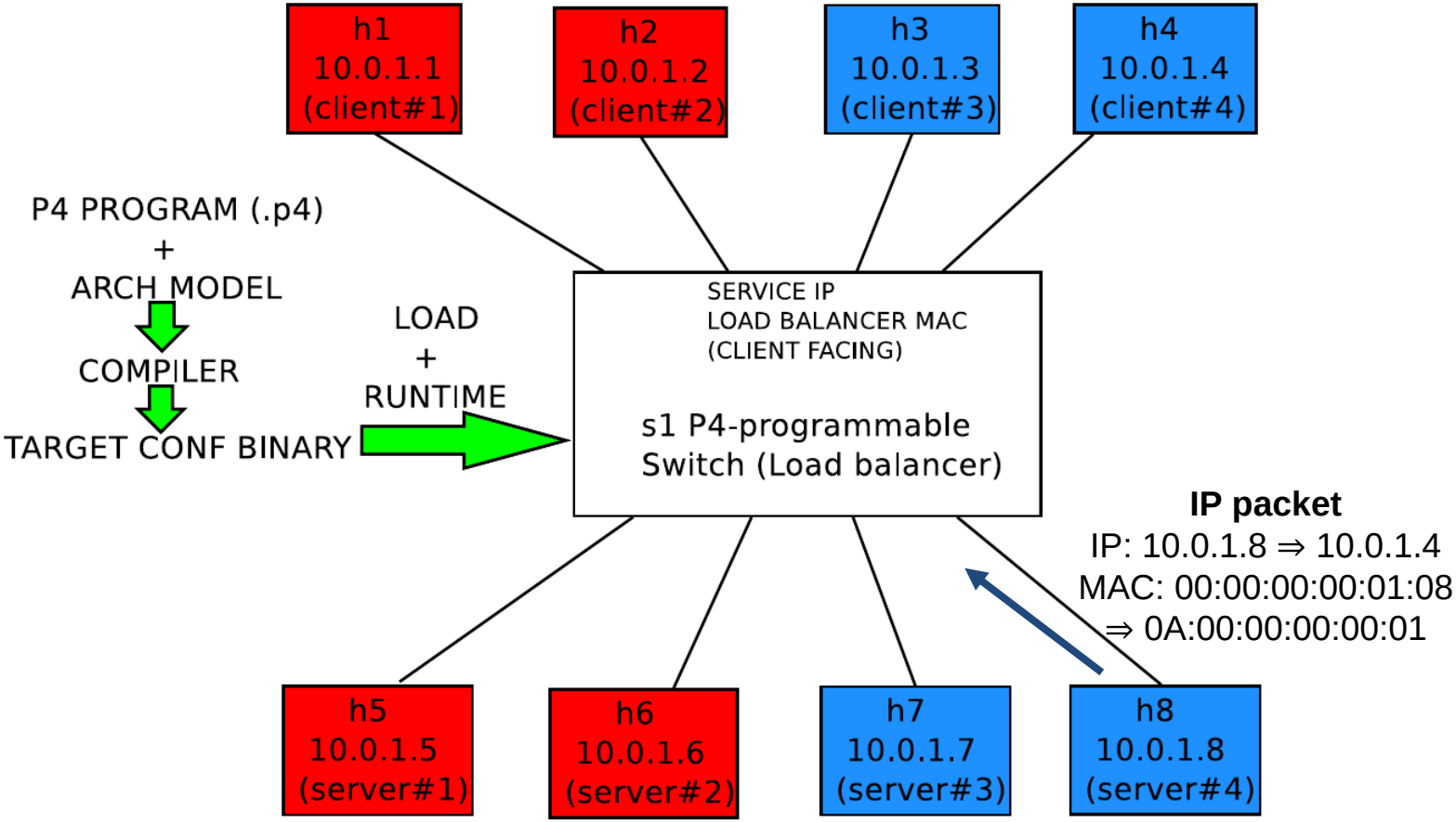
ARP requests: servers to LB



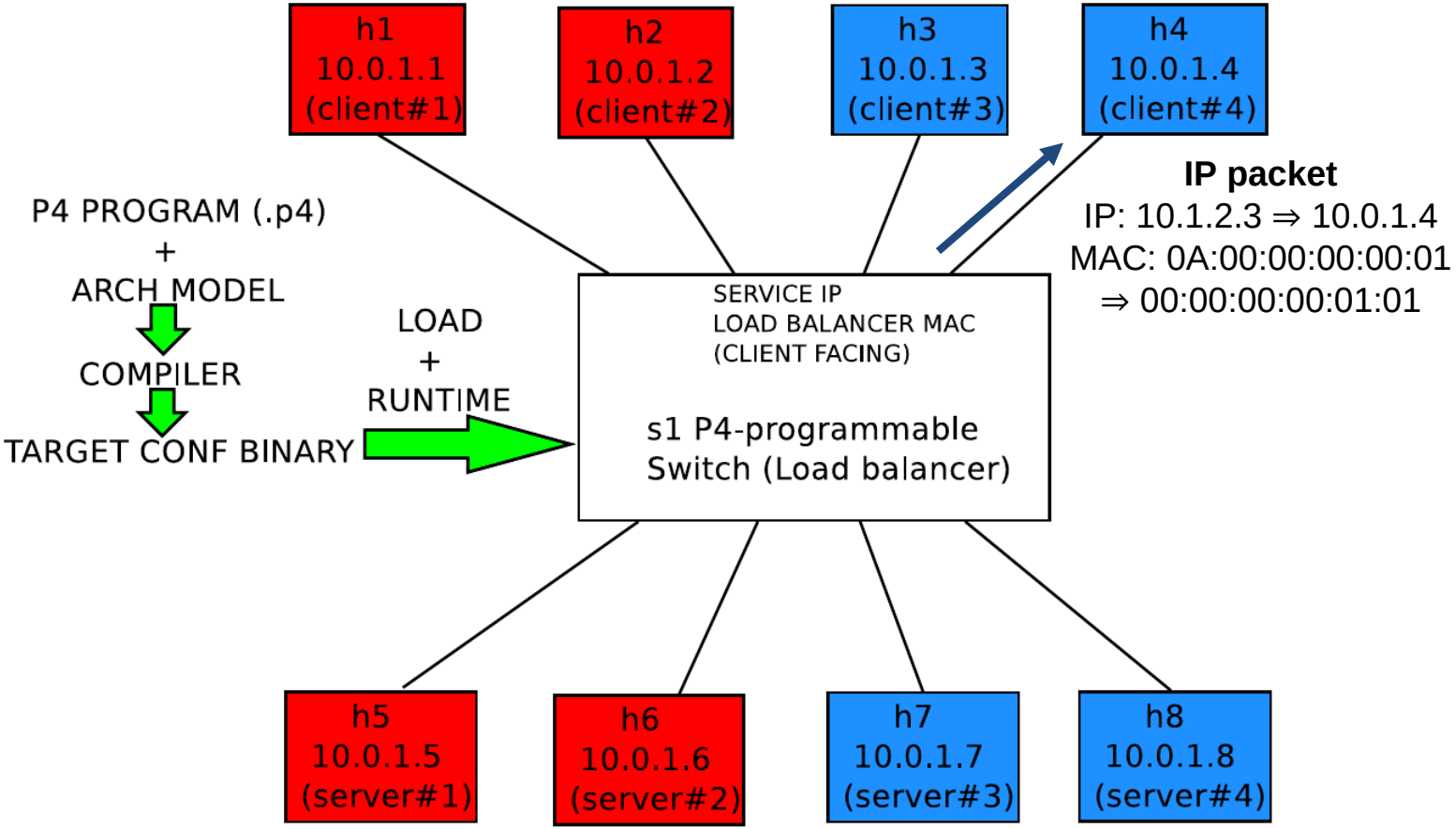
ARP replies: LB to servers



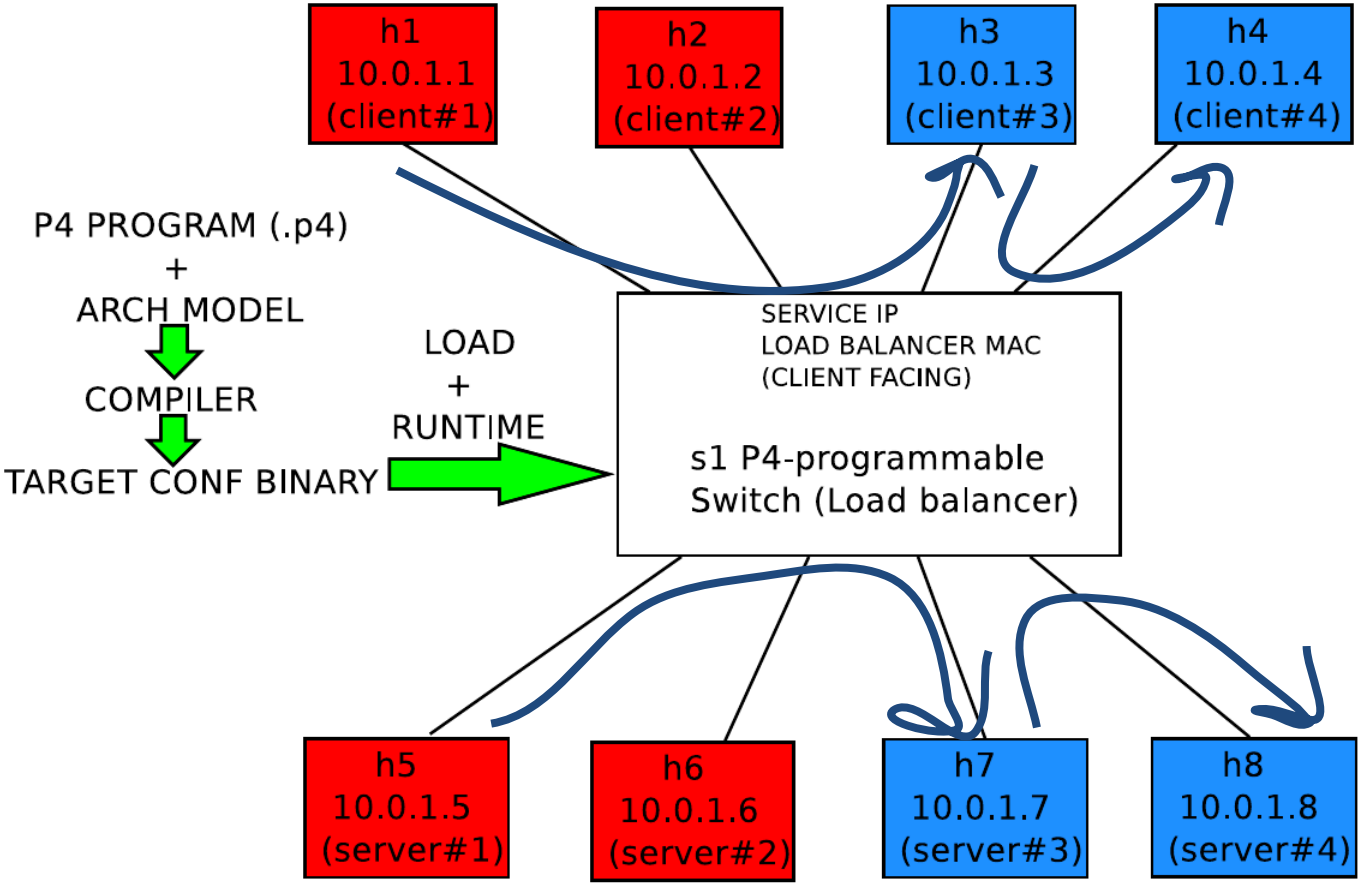
IP traffic: server to client reply



IP traffic: server to client reply



Traffic you do NOT need to deal with



Writing your application

1. Download + unzip `hy436as3` source code on your local machine

2. Check included files that require no further edits:
   a. `hy436as3/Makefile`: makefile for compiling the P4 application.
   b. `hy436as3/README.md`: README for explaining how to set up.
   c. `hy436as3/receive.py`: script to receive IP packets.
   d. `hy436as3/send.py`: script to send IP packets.
   e. `hy436as3/topology.json`: configuration of Mininet topology.
   f. `hy436as3/slb-runtime.json`: the run-time configuration of the simple load balancer.
   g. `hy436as3/reconf_lb_groups_runtime.py`: script to reconfigure the switch at run-time with new rules (defined in `slb-runtime.json`)

3. Edit `hy436as3/simple_load_balancer.p4`
   a. Check provided code skeleton
   b. Look for “WRITE YOUR CODE HERE” hints
   c. Check in-code comments
   d. Study carefully `hy436as3/slb-runtime.json`

Imports & Constants

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_IPV4       = 0x800;
const bit<16> TYPE_ARP        = 0x806;
const bit<32> serviceIP       = 0xa010203;
const bit<48> lbMAC           = 0xa0000000001;
```

Headers

```
typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

/* ethernet frame header */
header ethernet_t {
          macAddr_t   dstAddr;
          macAddr_t   srcAddr;
          bit<16>   etherType;
}

/* ARP packet header */
header arp_t {
          ...
          macAddr_t   hwSrcAddr;
          ip4Addr_t   protoSrcAddr;
          macAddr_t   hwDstAddr;
          ip4Addr_t   protoDstAddr;
}

/* IP packet header */
header ipv4_t {
          bit<4>    version;
          ...
          ip4Addr_t   srcAddr;
          ip4Addr_t   dstAddr;
}
```

```
/* metadata carried by the packet through
the processing pipelines */
struct metadata {
          macAddr_t           dstMAC;

          bit<1>              isClient;

          bit<1>              isServer;

          bit<8>              srcGroup;
          bit<8>              dstGroup;
}

struct headers {
          ethernet_t          ethernet;
          arp_t               arp;
          ipv4_t              ipv4;
}
```

Parser & Checksum Verification

```
parser SLBParser(packet_in packet,
                      out headers hdr,
                      inout metadata meta,
                      inout
standard_metadata_t
                  standard_metadata) {

          state start {
                     transition
parse_ethernet;
          }

          state parse_ethernet {
                     /* WRITE YOUR CODE HERE
*/
          }

          state parse_arp {

          packet.extract(hdr.arp);
                     transition accept;

     }

          state parse_ipv4 {

          packet.extract(hdr.ipv4);
```

```
control SLBVerifyChecksum(inout headers hdr,

            inout metadata meta) {
           apply {  }
}
```

Ingress processing (I)

```
control SLBIngress(inout headers hdr,
                   inout metadata meta,
                   inout
                   standard_metadata_t
                   standard_metadata) {

        action drop() {

        mark_to_drop(standard_metadata);
        }

        action arp_request_to_reply(
        macAddr_t srcMAC,
        macAddr_t dstMAC,
        ip4Addr_t srcIP,
        ip4Addr_t dstIP) {
                  /* WRITE YOUR CODE HERE */

        standard_metadata.egress_spec =

        standard_metadata.ingress_port;

    }

        action set_egress_metadata(
        macAddr_t dstMAC,
        egressSpec_t port) {
```

```
action set_client_metadata(
     ip4Addr_t firstAllowedReplica,
     ip4Addr_t lastAllowedReplica) {
               meta.isClient = 1;
               meta.isServer = 0;
               /* WRITE YOUR CODE HERE */
     }

action unset_client_metadata() {
               meta.isClient = 0;
     }

action set_server_metadata() {
               meta.isServer = 1;
               meta.isClient = 0;
}

     action unset_server_metadata() {
               meta.isServer = 0;
     }

action set_src_membership(bit<8> group) {
               meta.srcGroup = group;
     }

     action set_dst_membership(bit<8>
group) {
               meta.dstGroup = group;
```

Ingress processing (II)

```
        table arpmap {
                /* WRITE YOUR CODE HERE
*/
        }

        table ipv4_clients {
                key = {
                hdr.ipv4.srcAddr: lpm;
                }
                actions = {
                set_client_metadata;
                unset_client_metadata;
                }
        }

        table ipv4_servers {
                /* WRITE YOUR CODE HERE
*/
        }

        table src_group_membership {
                /* WRITE YOUR CODE HERE
*/
        }

        table dst_group_membership {
        apply {
                if (!(hdr.arp.isValid()
||
        hdr.ipv4.isValid())) {
                drop();
                }
        else if (hdr.arp.isValid() &&
        hdr.arp.opCode == 1) {
                /* WRITE YOUR CODE HERE
*/
                }
        else if (hdr.ipv4.isValid()) {
                /* WRITE YOUR CODE HERE
*/
                if (!((meta.isClient ==
1)
                                        ||
meta.isServer == 1)
                                        ||
meta.srcGroup
                                        !=
meta.dstGroup) {
                                drop();
                }
```

Egress processing

```
control SLBEgress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {

        action rewrite_client_to_server() {
                /* WRITE YOUR CODE HERE */
        }

        action rewrite_server_to_client() {
                /* WRITE YOUR CODE HERE */
        }

        apply {
                if (hdr.ipv4.isValid()) {
                /* WRITE YOUR CODE HERE */
                }
        }
}
```

Checksum Computation, Deparser and Switch

```
control SLBComputeChecksum(
     inout headers  hdr,
     inout metadata meta) {
                    apply {

          update_checksum(

          hdr.ipv4.isValid(),
                                {
                         hdr.ipv4.version,
                                        …

          hdr.ipv4.srcAddr,

          hdr.ipv4.dstAddr
                    },

          hdr.ipv4.hdrChecksum,

          HashAlgorithm.csum16
               );
                    }
}

control SLBDeparser(
     packet_out packet, in headers hdr) {
                    apply {
```

```
V1Switch(
          SLBParser(),

          SLBVerifyChecksum(),
          SLBIngress(),
          SLBEgress(),
          SLBComputeChecksum(),
          SLBDeparser()
) main;
```

Runtime entries: SLBIngress.ipv4_clients

```
...
{
     "table": "SLBIngress.ipv4_clients",
          "default_action": true,
          "action_name": "SLBIngress.unset_client_metadata",
          "action_params": {}
},
{
          "table": "SLBIngress.ipv4_clients",
          "match": {
                    "hdr.ipv4.srcAddr": [
                    "10.0.1.1",
                    32
                    ]
          },
          "action_name": "SLBIngress.set_client_metadata",
          "action_params": {
          "firstAllowedReplica": 167772421,
          "lastAllowedReplica": 167772422
          }
},
...
```

Runtime entries: SLBIngress.ipv4_servers

```
...
{
        "table": "SLBIngress.ipv4_servers",
        "default_action": true,
        "action_name": "SLBIngress.unset_server_metadata",
        "action_params": {}
},
{
        "table": "SLBIngress.ipv4_servers",
        "match": {
                  "hdr.ipv4.srcAddr": [
                  "10.0.1.5",
                  32
                  ]
        },
        "action_name": "SLBIngress.set_server_metadata",
        "action_params": {}
},
...
```

Runtime entries: SLBIngress.arpmap

```
...
{
        "table": "SLBIngress.arpmap",
        "match": {
                  "hdr.ipv4.dstAddr": [
                  "10.0.1.1",
                  32
                  ]
        },
        "action_name": "SLBIngress.set_egress_metadata",
        "action_params": {
                  "dstMAC": "00:00:00:00:01:01",
                  "port": 1
        }
},
...
```

Runtime entries: SLBIngress.src_group_membership

```
...
{
        "table": "SLBIngress.src_group_membership",
        "match": {
                  "hdr.ipv4.srcAddr": [
                  "10.0.1.1",
                  32
                  ]
        },
        "action_name": "SLBIngress.set_src_membership",
        "action_params": {
        "group": 1
        }
},
...
```

Runtime entries: SLBIngress.dst_group_membership

```
...
{
        "table": "SLBIngress.dst_group_membership",
        "match": {
                  "hdr.ipv4.dstAddr": [
                  "10.0.1.5",
                  32
                  ]
        },
        "action_name": "SLBIngress.set_dst_membership",
        "action_params": {
                  "group": 1
        }
},
...
```

(Compiling and) running your application (I)

1. On VM: `mkdir /home/p4/tutorials/exercises/simple_load_balancer`
2. copy all files from local hy436 folder to the VM at: `/home/p4/tutorials/exercises/simple_load_balancer`
3. `cd /home/p4/tutorials/exercises/simple_load_balancer`
4. `make run`

```
...

Reading topology file.
Building mininet topology.
Switch port mapping:
s1:  1:h1    2:h2    3:h3    4:h4    5:h5    6:h6    7:h7    8:h8
Configuring switch s1 using P4Runtime with file slb-runtime.json
- Using P4Info file build/simple_load_balancer.p4.p4info.txt…
- Connecting to P4Runtime server on 127.0.0.1:50051 (bmv2)...
- Setting pipeline config (build/simple_load_balancer.json)...
- Inserting 34 table entries…
- SLBIngress.ipv4_clients: (default action) => SLBIngress.unset_client_metadata()
…
s1 -> gRPC port: 50051
**********
H1
default interface: h1-eth0    10.0.1.1    00:00:00:00:01:01
**********
...
```

(Compiling and) running your application (II)

```
...
Starting mininet CLI
================================================================================
Welcome to the BMV2 Mininet CLI!
================================================================================
Your P4 program is installed into the BMV2 software switch
and your initial runtime configuration is loaded. You can interact
with the network using the mininet CLI below.

To view a switch log, run this command from your host OS:
tail -f /home/p4/tutorials/exercises/simple_load_balancer/logs/<switchname>.log

To view the switch output pcap, check the pcap files in
/home/p4/tutorials/exercises/simple_load_balancer/pcaps:
for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

To view the P4Runtime requests sent to the switch, check the
corresponding txt file in /home/p4/tutorials/exercises/simple_load_balancer/logs:
for example run:  cat /home/p4/tutorials/exercises/simple_load_balancer/logs/s1-
p4runtime-requests.txt

mininet>
```

⇒ You now have a mininet terminal that you can use for testing!

Testing your application

1. `mininet> xterm h1 h3 h5 h6 h7 h8`

2. `host_xterm# sudo tcpdump -ni <server_hostname>-eth0 icmp`
   (`server_hostname` in {`h5, …, h8`}

3. Ping service IP (10.1.2.3) from different clients (e.g., h1, h3)
   a. Where do the ICMP requests go? Who replies?
   b. What happens if you use TCP sessions (`send.py, receive.py`)?

4. Edit `slb-runtime.json` and apply new runtime (at run-time :)):
   a. `sudo python reconf_lb_groups_runtime.py`
   b. Example change: point h1 client to servers h7, h8 instead of h5, h6
      i. Change allowed replicas in `SLBIngress.ipv4_clients` table
      ii. Change src group membership for h1 in `SLBIngress.src_group_membership` table
      iii. Change dst group membership for h1 in `SLBIngress.dst_group_membership` table

Debugging your application

- `simple_load_balancer/logs/s1.log`
  - Logs of how the switch has processed received packets
- `simple_load_balancer/logs/s1-p4runtime-requests.txt`
  - Logs of requests from control plane to update switch tables (P4Runtime)
- `simple_load_balancer/logs/pcaps/*.pcap`
  - Captured packets on switch interfaces (in/out), to be read by `Wireshark`
- `simple_load_balancer/build/simple_load_balancer.p4.p4info.txt`
  - P4 run-time attributes
  - table/action/param IDs, …
  - table structure, action params, …
- `simple_load_balancer/build/simple_load_balancer.json`
  - Compiled `.p4` program in `.json` format

Shutting down and re-testing your application

1. On Mininet CLI, run:
   ```
   mininet> exit
   ```

2. After Mininet exits, run:
   ```
   vm_terminal$ make stop && make clean
   ```

ATTENTION: make clean deletes all log/debug files (see debugging slide), so examine them either while Mininet is running or before cleaning everything up!

Submitting your application

1. Log-in to one of the CS department's systems.

2. Create a folder named ask3.

3. ask3 folder should contain the following files:
   a. `simple load balancer.p4`
   b. any additional version of `slb-runtime.json` you used to test your code named e.g., `slb-runtime-vX.json`, where X∈ {2, 3, · · · }

4. Use `cd` to make the directory one level above ask3 your current working directory.

5. Issue the following command:
   `turnin assignment3@hy436 ask3`

6. Deadline for submission is: December 4, 23:59

DEMO

Online resources

- Link for downloading official tutorial P4 VM: https://tinyurl.com/hy436P4vm
- Mininet: http://mininet.org
- VirtualBox: https://www.virtualbox.org
- Putty: https://www.putty.org/
- Filezilla: https://filezilla-project.org/
- Sublime Text: https://www.sublimetext.com/
- P4 general portal (blog, events, specifications, code links, community, …):
  - https://p4.org/
- P4 code portal (tutorials, examples, compiler, BMv2, P4Runtime, …):
  - https://p4.org/code/
- P4 tutorials:
  - https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf
  - https://github.com/p4lang/tutorials
- P4 GitHub repositories:
  - P4 Compiler: https://github.com/p4lang/p4c
  - BMv2: https://github.com/p4lang/behavioral-model
  - P4Runtime: https://github.com/p4lang/p4runtime
  - Packet Test Framework: https://github.com/p4lang/ptf
  - P4 Container: https://github.com/p4lang/p4app
- Publications:
  - https://p4.org/publications/