PI-hole in P4

Készítette: Czeglédi Gábor, Fonyódi Balázs, Háfinec Szergej Miklós

What is the PI-hole?

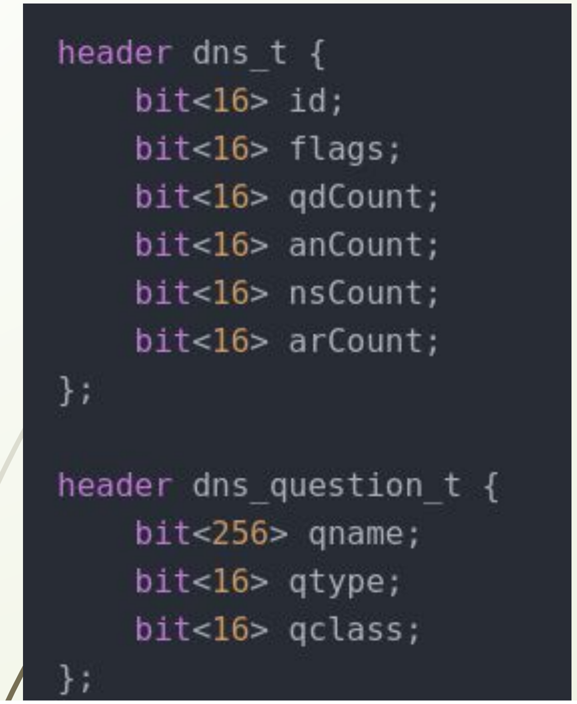
Pi-hole is a free and open-source software application designed to block advertisements and tracking domains at the network level. It acts as a DNS sinkhole, intercepting DNS requests from devices on a network and blocking requests to domains associated with ads and trackers. Pi, and can be configured to block ads on all devices connected to a network, including smartphones, tablets, and computers.

By blocking ads at the network level, Pi-hole can improve browsing speeds, reduce data usage, and protect user privacy by preventing devices from sending data to ad networks and trackers.

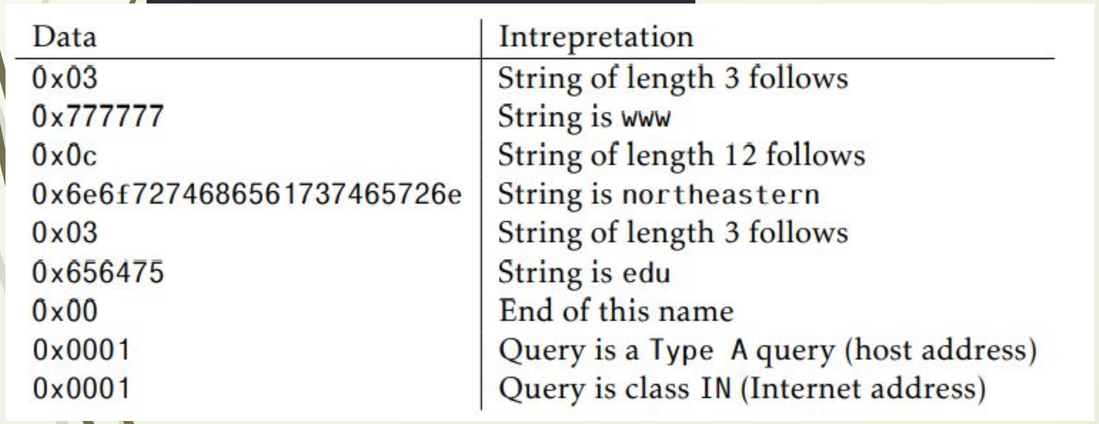
How to implement Pi-hole in P4

- Define a P4 program to capture DNS packets: The P4 program must include a DNS header.
- First, you need to add the DNS header you want to filter on.
- Then, you need to filter to the address in the control block.
- DNS header filtering can be applied in some ways. We will use table matching for fix addresses and if there is a match, it will block it.
- As we want to test the Pi-hole implementation to make sure it works as expected, we will create a script whose task will be to simulate the sending of packets and thus test the correctness of the program.

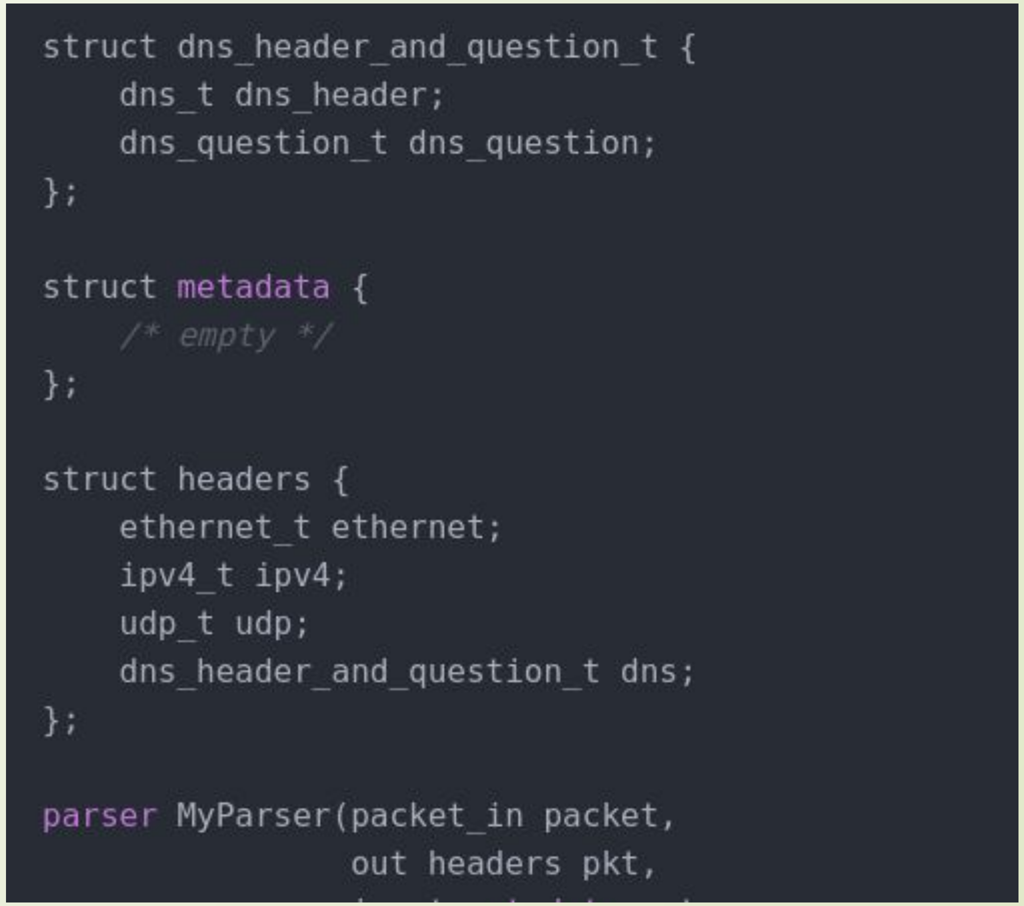
Screenshoots of the p4 file



```
header dns_t {
    bit<16> id;
    bit<16> flags;
    bit<16> qdCount;
    bit<16> anCount;
    bit<16> nsCount;
    bit<16> arCount;
};

header dns_question_t {
    bit<256> qname;
    bit<16> qtype;
    bit<16> qclass;
};
```



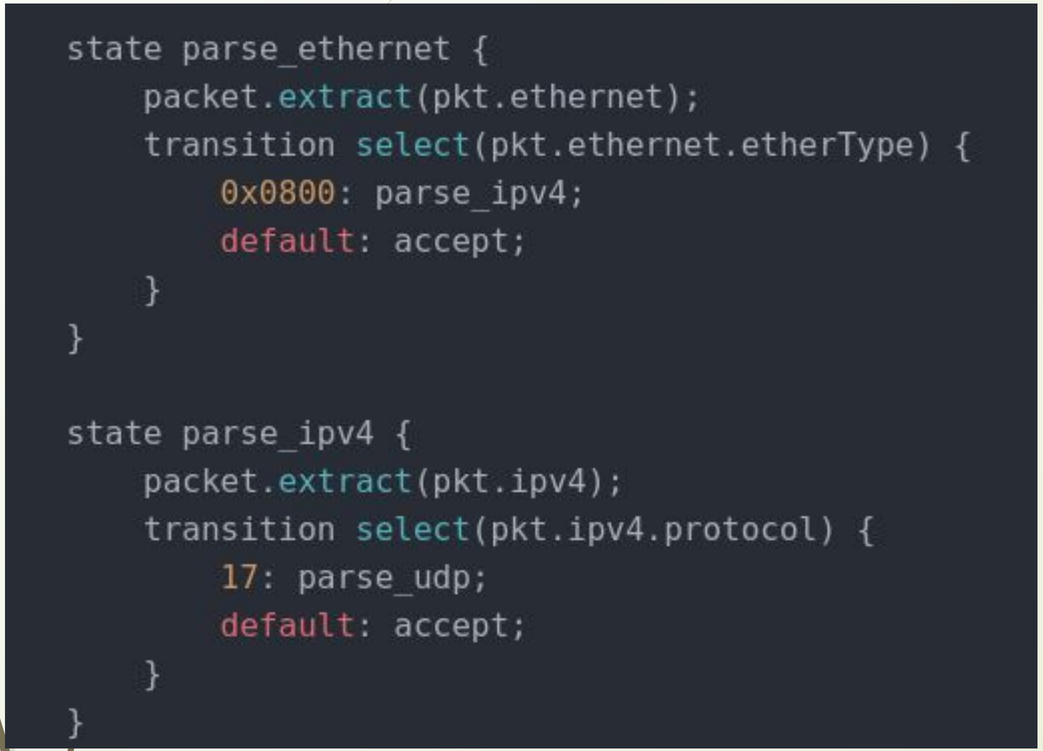
Data	Intrepretation
0x03	String of length 3 follows
0x777777	String is www
0x0c	String of length 12 follows
0x6e6f7274686561737465726e	String is northeastern
0x03	String of length 3 follows
0x656475	String is edu
0x00	End of this name
0x0001	Query is a Type A query (host address)
0x0001	Query is class IN (Internet address)



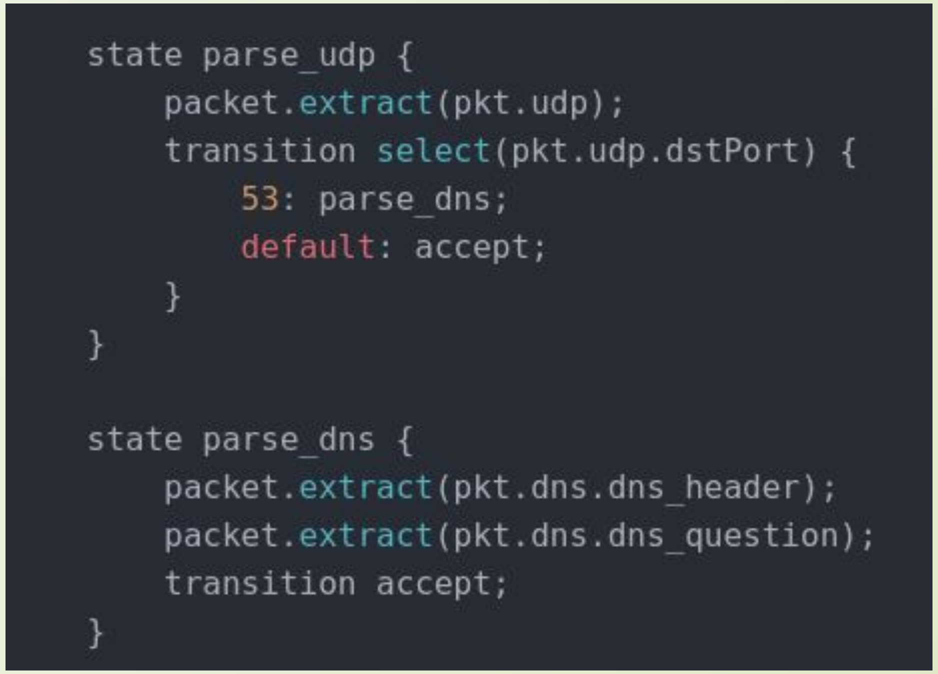
```
struct dns_header_and_question_t {
    dns_t dns_header;
    dns_question_t dns_question;
};

struct metadata {
    /* empty */
};

struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
    udp_t udp;
    dns_header_and_question_t dns;
};

parser MyParser(packet_in packet,
                out headers pkt,
```

pl www.google.com -> 3www6google3com

Screenshoots of the p4 file

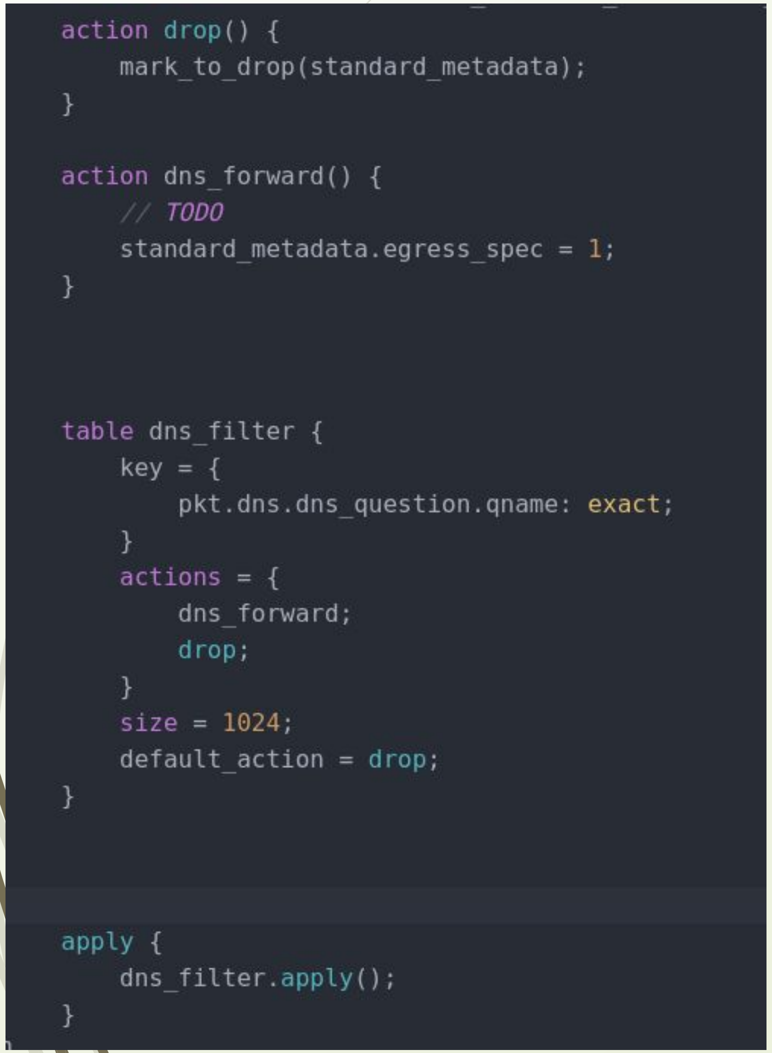


```
state parse_ethernet {
    packet.extract(pkt.ethernet);
    transition select(pkt.ethernet.etherType) {
        0x0800: parse_ipv4;
        default: accept;
    }
}

state parse_ipv4 {
    packet.extract(pkt.ipv4);
    transition select(pkt.ipv4.protocol) {
        17: parse_udp;
        default: accept;
    }
}
```

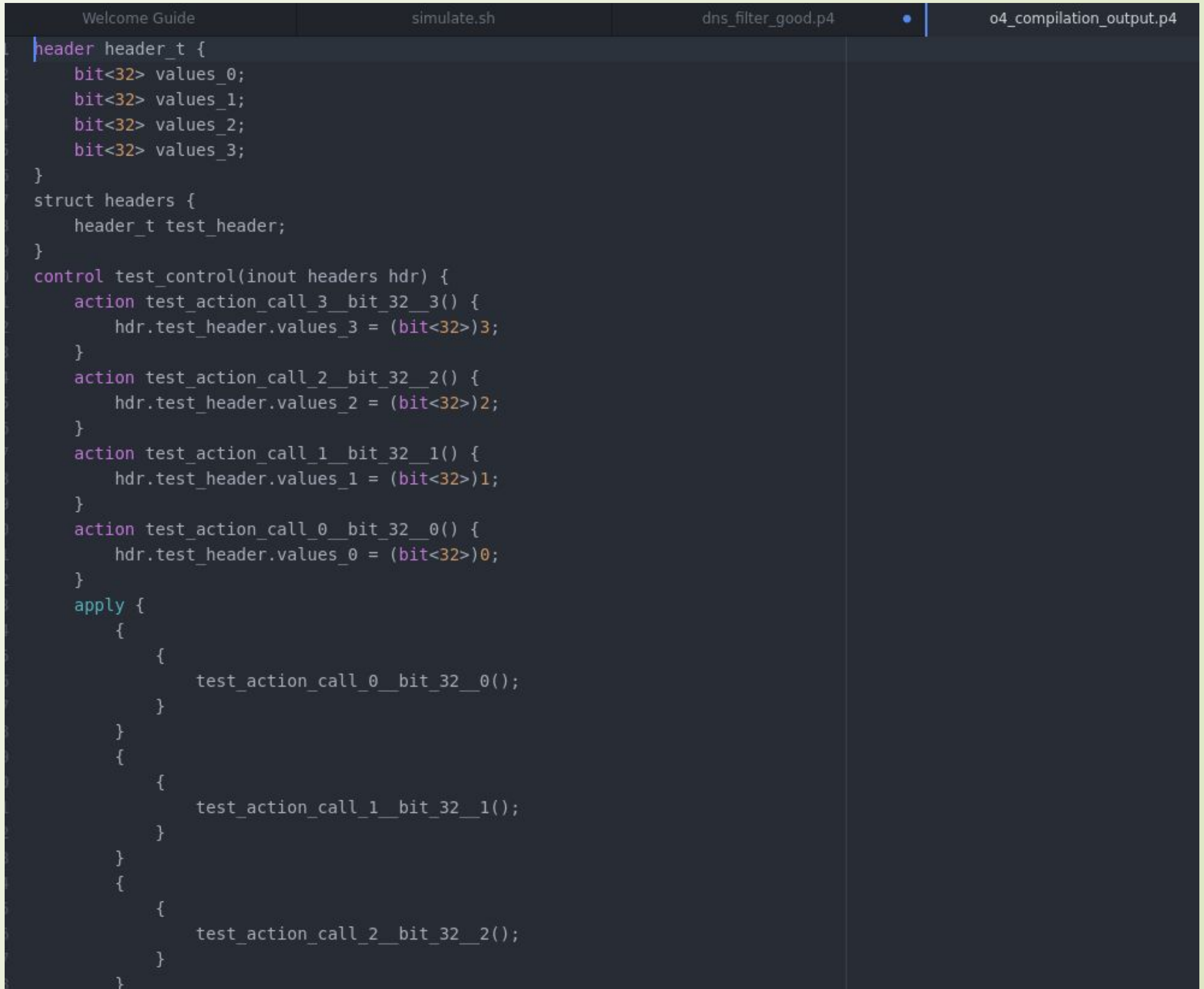


```
state parse_udp {
    packet.extract(pkt.udp);
    transition select(pkt.udp.dstPort) {
        53: parse_dns;
        default: accept;
    }
}

state parse_dns {
    packet.extract(pkt.dns.dns_header);
    packet.extract(pkt.dns.dns_question);
    transition accept;
}
```

Questions, TODO

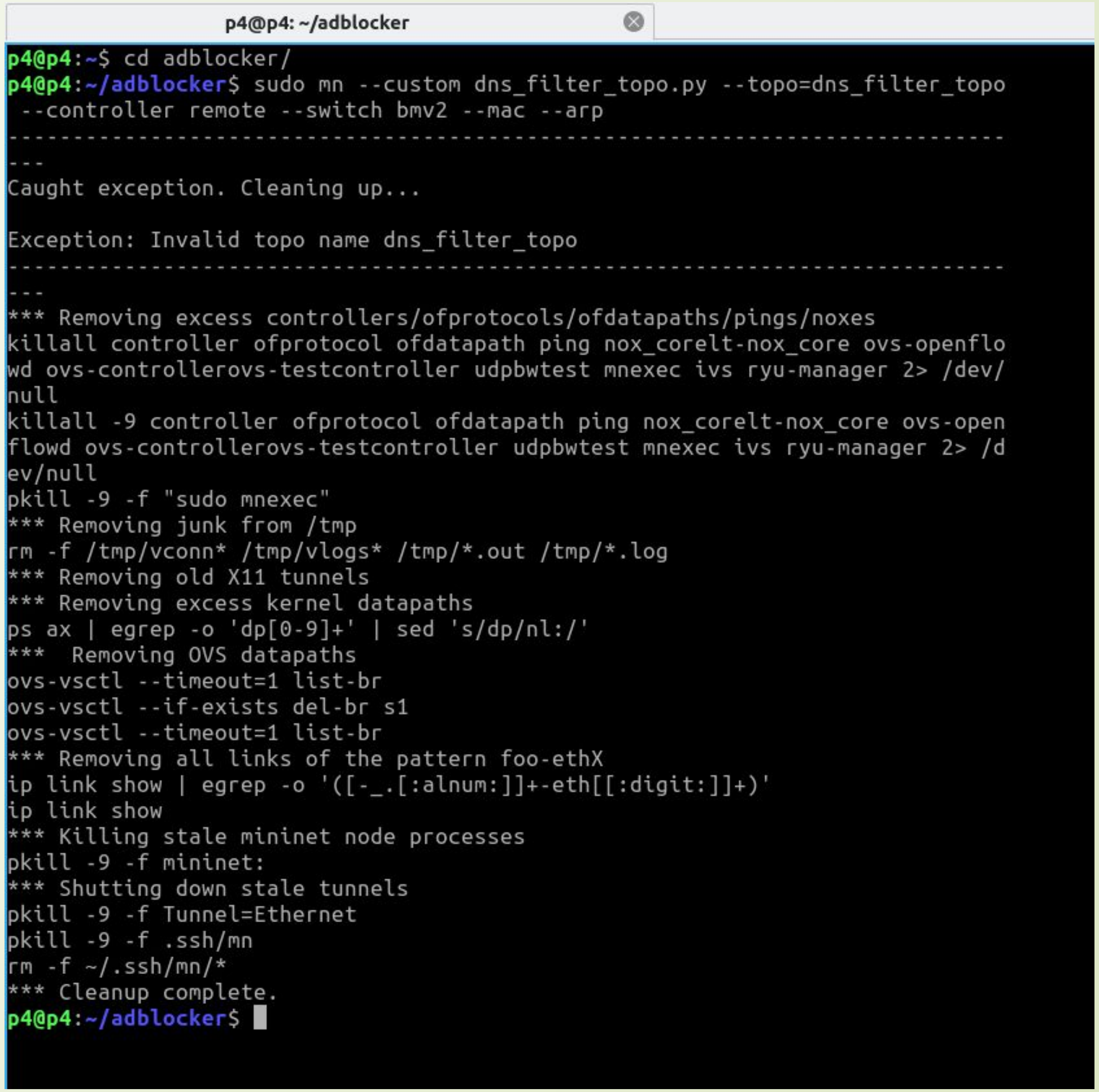


```
action drop() {
    mark_to_drop(standard_metadata);
}

action dns_forward() {
    // TODO
    standard_metadata.egress_spec = 1;
}



table dns_filter {
    key = {
        pkt.dns.dns_question.qname: exact;
    }
    actions = {
        dns_forward;
        drop;
    }
    size = 1024;
    default_action = drop;
}



apply {
    dns_filter.apply();
}
```



```
header header_t {
    bit<32> values_0;
    bit<32> values_1;
    bit<32> values_2;
    bit<32> values_3;
}
struct headers {
    header_t test_header;
}
control test_control(inout headers hdr) {
    action test_action_call_3__bit_32__3() {
        hdr.test_header.values_3 = (bit<32>)3;
    }
    action test_action_call_2__bit_32__2() {
        hdr.test_header.values_2 = (bit<32>)2;
    }
    action test_action_call_1__bit_32__1() {
        hdr.test_header.values_1 = (bit<32>)1;
    }
    action test_action_call_0__bit_32__0() {
        hdr.test_header.values_0 = (bit<32>)0;
    }
    apply {
        {
            {
                test_action_call_0__bit_32__0();
            }
        }
        {
            {
                test_action_call_1__bit_32__1();
            }
        }
        {
            {
                test_action_call_2__bit_32__2();
            }
        }
```

Questions, TODO


p4@p4: ~/adblocker

```
p4@p4:~$ cd adblocker/
p4@p4:~/adblocker$ sudo mn --custom dns_filter_topo.py --topo=dns_filter_topo
 --controller remote --switch bmv2 --mac --arp
----------------------------------------------------------------------------
---
Caught exception. Cleaning up...

Exception: Invalid topo name dns_filter_topo
----------------------------------------------------------------------------
---
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflo
wd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/
null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-open
flowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /d
ev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
p4@p4:~/adblocker$
```