DNS FlowNet : A flownet for tracking DNS requests/responses

For partial fulfillment of CS-F366 under Dr. Vishal Gupta

Submitted By: Divesh Uttamchandani (2016A7TS0045P)

Introduction

Accountability is very important in computer networks. It helps us in tracing events to its cause. It answers questions like “Who did what”. Answering these questions in a system without accountability is difficult and falls under the domain of forensics. Accountable systems make forensics trivial.

Our aim through this work is to create an accountable system of DNS packets which can help us take actions in case a DNS attack is detected and hold an entity responsible

Yang Xiao’s work on FlowNet

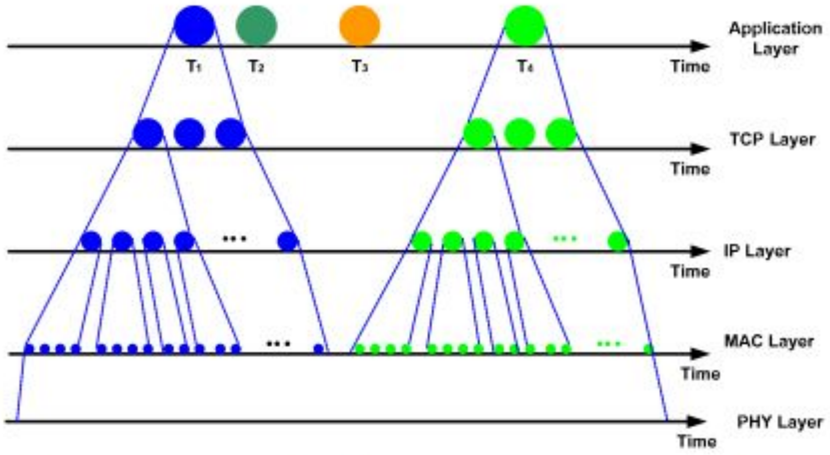


Figure 3. Multi-level and multi-resolution (layers). IP, Internet protocol; MAC, media access control; PHY, physical; TCP, transmission control protocol.

Paper ref: https://ieeexplore.ieee.org/document/5680278

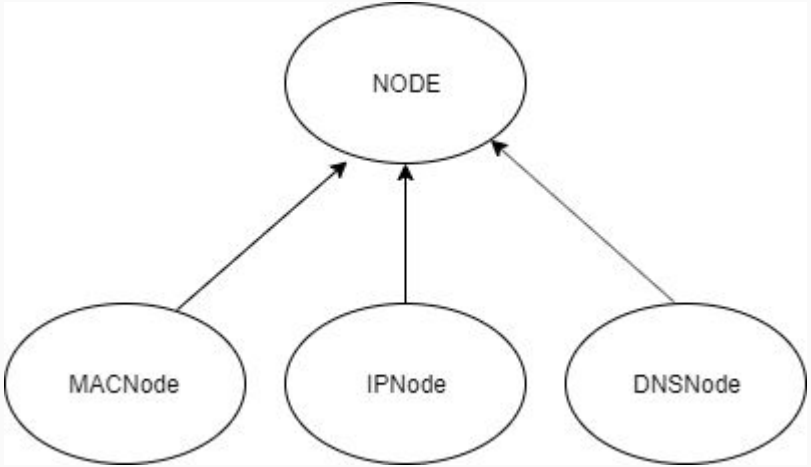
Problems with Yang Xiao’s FlowNet

- The FlowNet implemented by the above paper works only on wireless networks where a machine can listen to all the packets in transit.
- Implementing the above strategy on a wired network will be very difficult and resource consuming as we will have to mirror all the packets to the machine creating FlowNet
- Also the FlowNet proposed always keeps on increasing which may not be practically feasible

How we use/extend his work

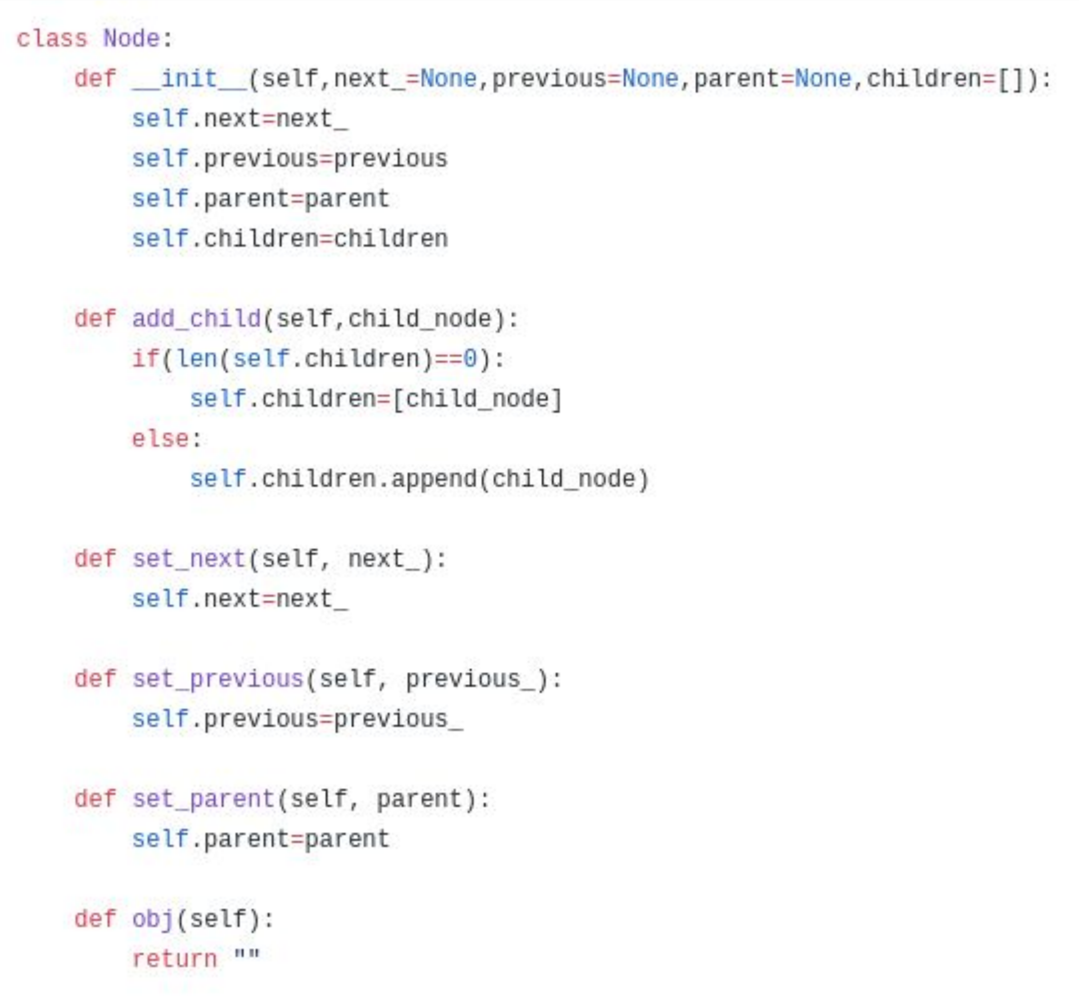
- We want to create a FlowNet of DNS flows
- We use software defined network as a test-bed which makes it very easy to deploy various applications on it
- We create an accountability application for DNS flows on SDN
- We also evict flows which have reached correctly*

DNS FlowNet: Classes



Class Node

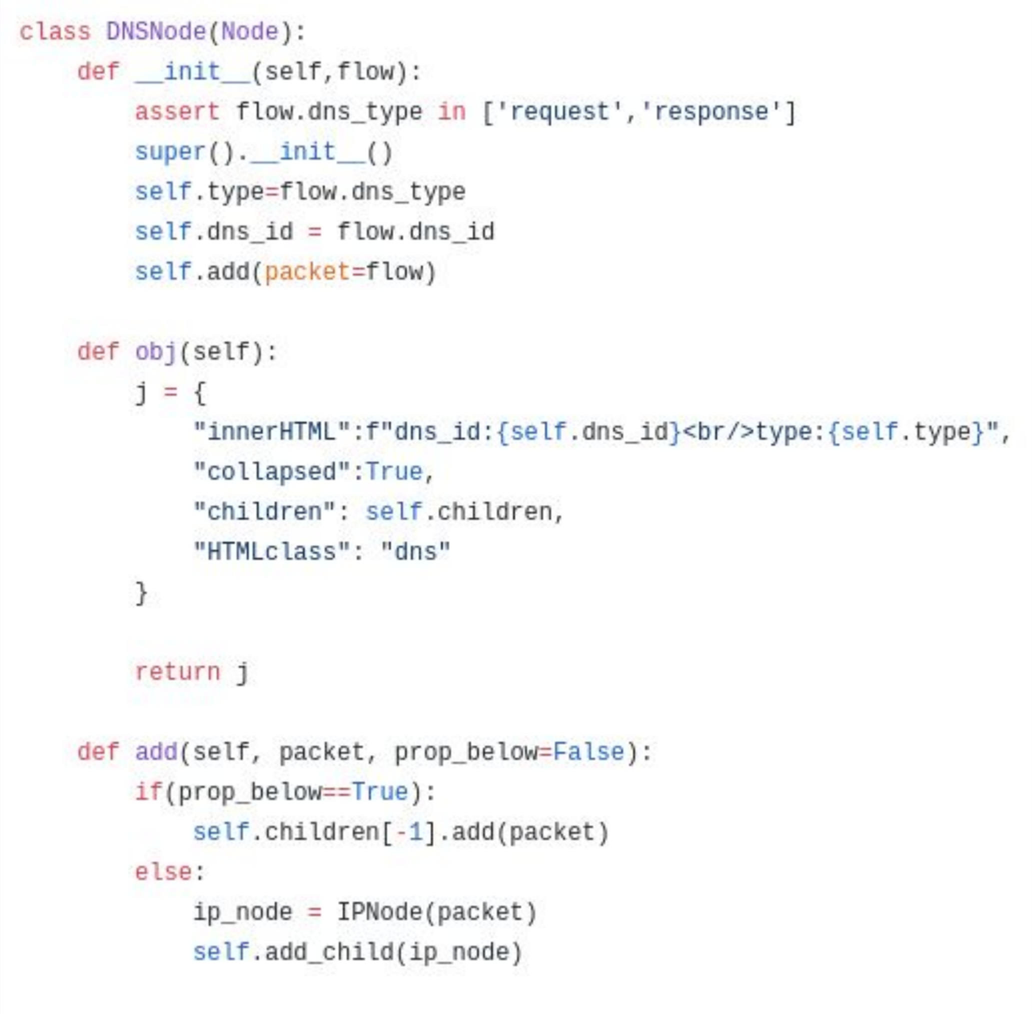
Generic Class for tree (FlowNet) based functionality



```python
class Node:
    def __init__(self,next_=None,previous=None,parent=None,children=[]):
        self.next=next_
        self.previous=previous
        self.parent=parent
        self.children=children

    def add_child(self,child_node):
        if(len(self.children)==0):
            self.children=[child_node]
        else:
            self.children.append(child_node)

    def set_next(self, next_):
        self.next=next_

    def set_previous(self, previous_):
        self.previous=previous_

    def set_parent(self, parent):
        self.parent=parent

    def obj(self):
        return ""
```

Class DNSNode

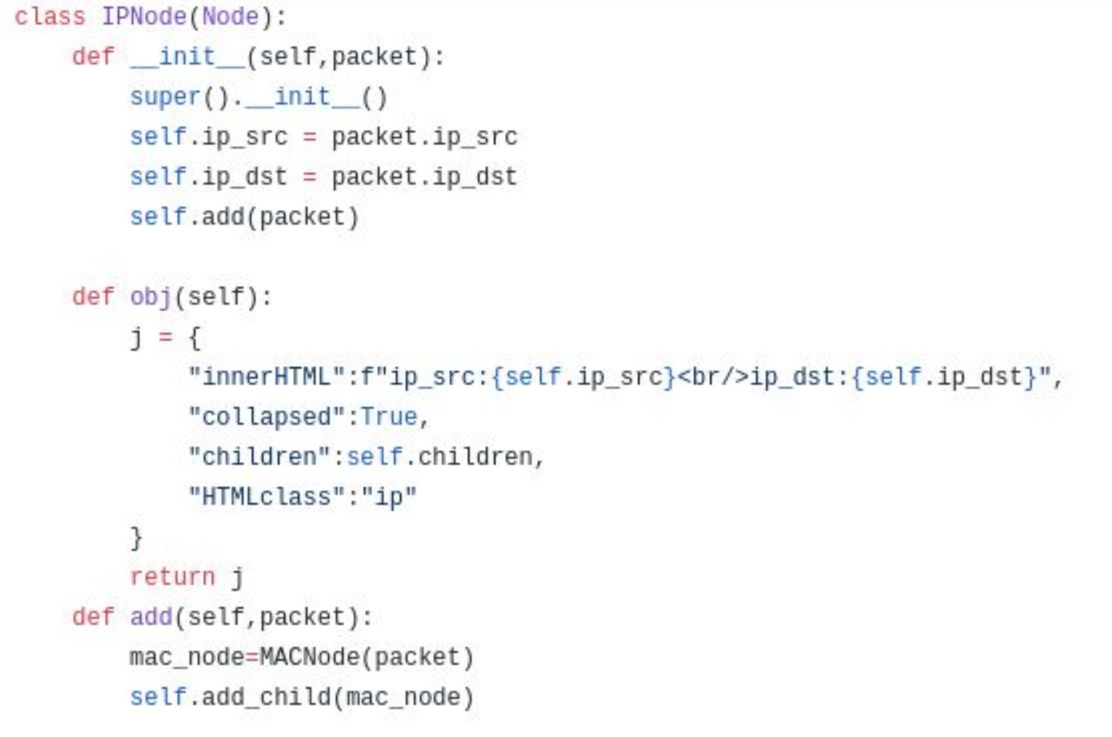
- DNSNode in the FlowNet
- Will always have IPNode as child



```python
class DNSNode(Node):
    def __init__(self,flow):
        assert flow.dns_type in ['request','response']
        super().__init__()
        self.type=flow.dns_type
        self.dns_id = flow.dns_id
        self.add(packet=flow)

    def obj(self):
        j = {
            "innerHTML":f"dns_id:{self.dns_id}<br/>type:{self.type}",
            "collapsed":True,
            "children": self.children,
            "HTMLclass": "dns"
        }

        return j

    def add(self, packet, prop_below=False):
        if(prop_below==True):
            self.children[-1].add(packet)
        else:
            ip_node = IPNode(packet)
            self.add_child(ip_node)
```

IPNode Class

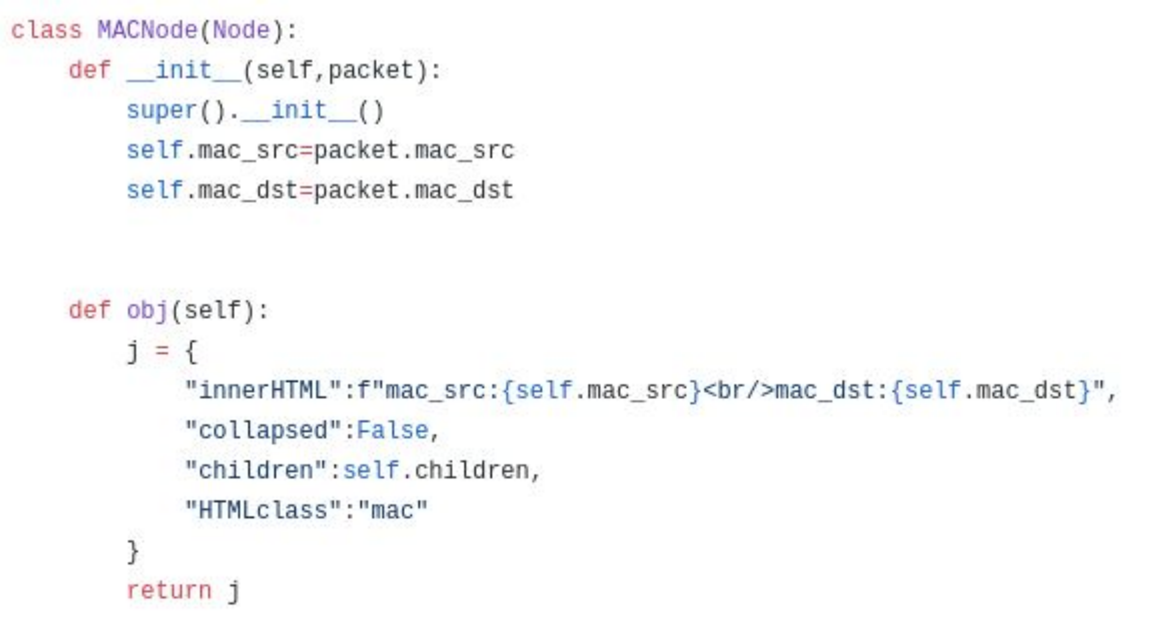
- IPNode in the FlowNet
- Will always have MAC Node as child



```
class IPNode(Node):
    def __init__(self,packet):
        super().__init__()
        self.ip_src = packet.ip_src
        self.ip_dst = packet.ip_dst
        self.add(packet)

    def obj(self):
        j = {
            "innerHTML":f"ip_src:{self.ip_src}<br/>ip_dst:{self.ip_dst}",
            "collapsed":True,
            "children":self.children,
            "HTMLclass":"ip"
        }
        return j
    def add(self,packet):
        mac_node=MACNode(packet)
        self.add_child(mac_node)
```

MACNode

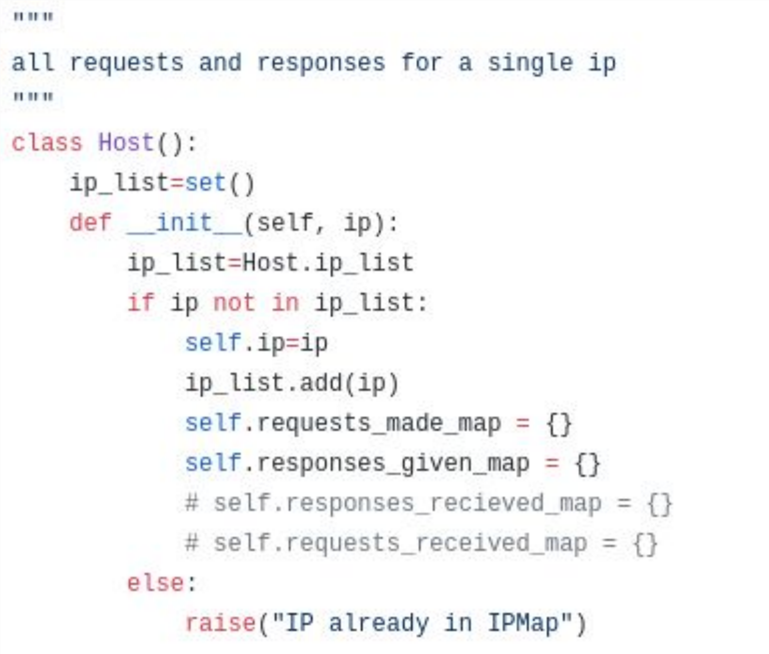
- MAC Node in the FlowNet
- Will always have no children



```
class MACNode(Node):
    def __init__(self,packet):
        super().__init__()
        self.mac_src=packet.mac_src
        self.mac_dst=packet.mac_dst


    def obj(self):
        j = {
            "innerHTML":f"mac_src:{self.mac_src}<br/>mac_dst:{self.mac_dst}",
            "collapsed":False,
            "children":self.children,
            "HTMLclass":"mac"
        }
        return j
```

Class Host

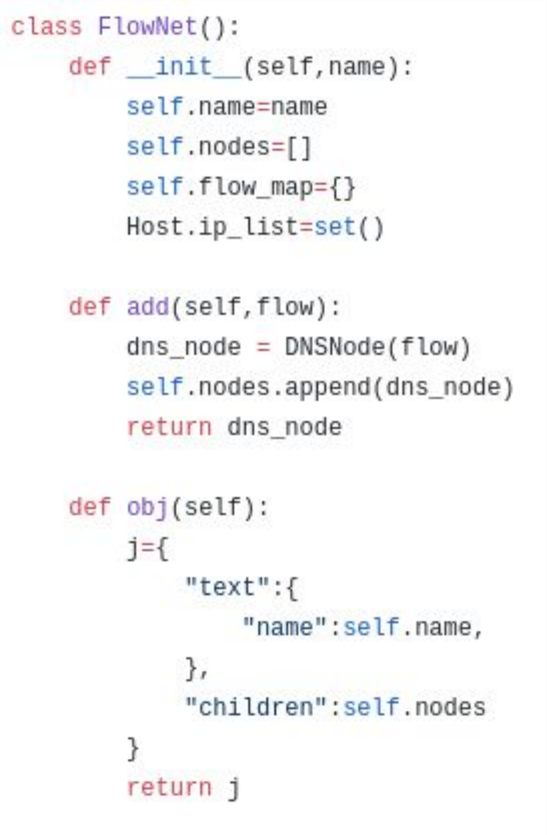
Host has all requests and responses for a single IP



```
"""
all requests and responses for a single ip
"""
class Host():
    ip_list=set()
    def __init__(self, ip):
        ip_list=Host.ip_list
        if ip not in ip_list:
            self.ip=ip
            ip_list.add(ip)
            self.requests_made_map = {}
            self.responses_given_map = {}
            # self.responses_recieved_map = {}
            # self.requests_received_map = {}
        else:
            raise("IP already in IPMap")
```

Class FlowNet

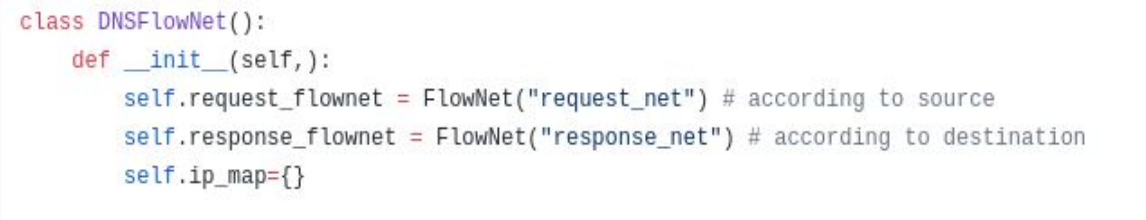
Represents a single FlowNet



```
class FlowNet():
    def __init__(self,name):
        self.name=name
        self.nodes=[]
        self.flow_map={}
        Host.ip_list=set()

    def add(self,flow):
        dns_node = DNSNode(flow)
        self.nodes.append(dns_node)
        return dns_node

    def obj(self):
        j={
            "text":{
                "name":self.name,
            },
            "children":self.nodes
        }
        return j
```

Class DNS FlowNet

ip_map maps ip addresses to Host objects

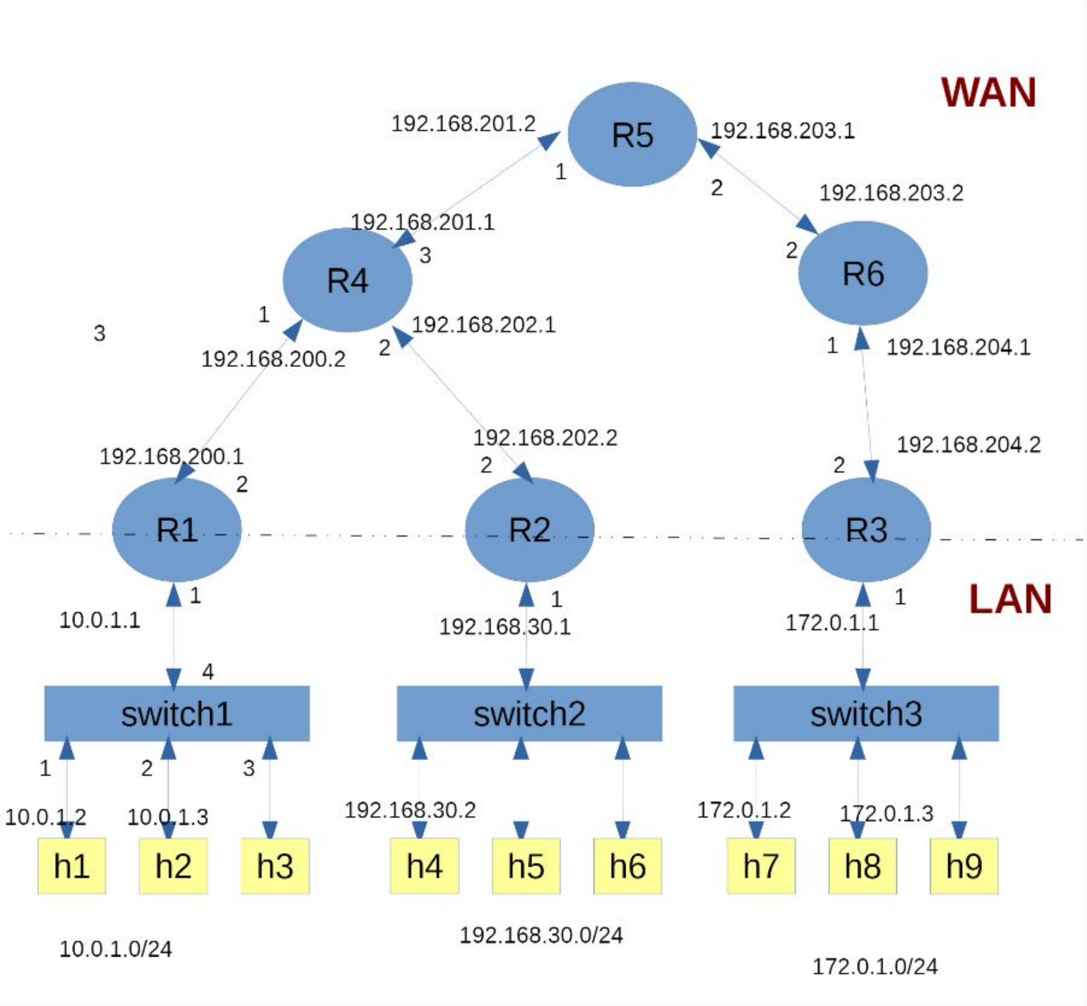


```
class DNSFlowNet():
    def __init__(self,):
        self.request_flownet = FlowNet("request_net") # according to source
        self.response_flownet = FlowNet("response_net") # according to destination
        self.ip_map={}
```

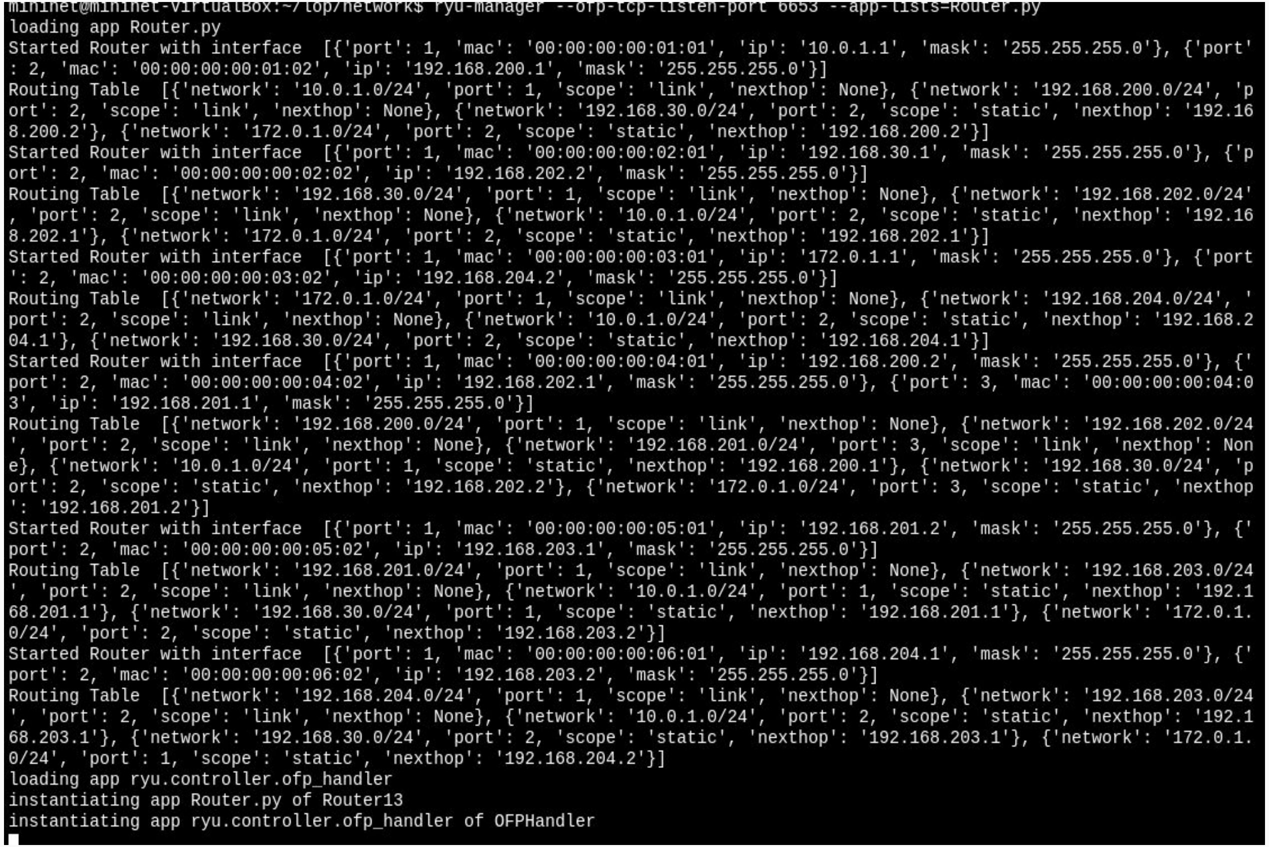
TestBed

We use this WAN testbed.

- DNS Server is at h9
- We test 3 requests in flownet shown later
    - h1 - h9
    - h4 - h9
    - h7 - h9

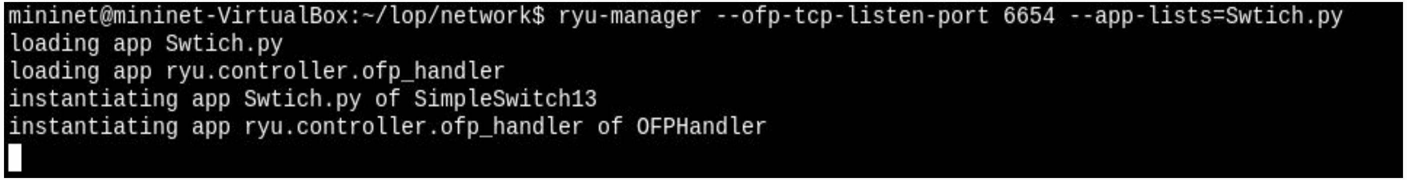


Router controller runs on port 6653



```
mininet@mininet-VirtualBox:~/top/network$ ryu-manager --ofp-tcp-listen-port 6653 --app-lists=Router.py
loading app Router.py
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:01:01', 'ip': '10.0.1.1', 'mask': '255.255.255.0'}, {'port'
: 2, 'mac': '00:00:00:00:01:02', 'ip': '192.168.200.1', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '10.0.1.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.200.0/24', 'p
ort': 2, 'scope': 'link', 'nexthop': None}, {'network': '192.168.30.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.16
8.200.2'}, {'network': '172.0.1.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.200.2'}]
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:02:01', 'ip': '192.168.30.1', 'mask': '255.255.255.0'}, {'p
ort': 2, 'mac': '00:00:00:00:02:02', 'ip': '192.168.202.2', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '192.168.30.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.202.0/24'
, 'port': 2, 'scope': 'link', 'nexthop': None}, {'network': '10.0.1.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.16
8.202.1'}, {'network': '172.0.1.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.202.1'}]
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:03:01', 'ip': '172.0.1.1', 'mask': '255.255.255.0'}, {'port
': 2, 'mac': '00:00:00:00:03:02', 'ip': '192.168.204.2', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '172.0.1.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.204.0/24', '
port': 2, 'scope': 'link', 'nexthop': None}, {'network': '10.0.1.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.2
04.1'}, {'network': '192.168.30.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.204.1'}]
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:04:01', 'ip': '192.168.200.2', 'mask': '255.255.255.0'}, {'
port': 2, 'mac': '00:00:00:00:04:02', 'ip': '192.168.202.1', 'mask': '255.255.255.0'}, {'port': 3, 'mac': '00:00:00:00:04:0
3', 'ip': '192.168.201.1', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '192.168.200.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.202.0/24
', 'port': 2, 'scope': 'link', 'nexthop': None}, {'network': '192.168.201.0/24', 'port': 3, 'scope': 'link', 'nexthop': Non
e}, {'network': '10.0.1.0/24', 'port': 1, 'scope': 'static', 'nexthop': '192.168.200.1'}, {'network': '192.168.30.0/24', 'p
ort': 2, 'scope': 'static', 'nexthop': '192.168.202.2'}, {'network': '172.0.1.0/24', 'port': 3, 'scope': 'static', 'nexthop
': '192.168.201.2'}]
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:05:01', 'ip': '192.168.201.2', 'mask': '255.255.255.0'}, {'
port': 2, 'mac': '00:00:00:00:05:02', 'ip': '192.168.203.1', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '192.168.201.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.203.0/24
', 'port': 2, 'scope': 'link', 'nexthop': None}, {'network': '10.0.1.0/24', 'port': 1, 'scope': 'static', 'nexthop': '192.1
68.201.1'}, {'network': '192.168.30.0/24', 'port': 1, 'scope': 'static', 'nexthop': '192.168.201.1'}, {'network': '172.0.1.
0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.203.2'}]
Started Router with interface  [{'port': 1, 'mac': '00:00:00:00:06:01', 'ip': '192.168.204.1', 'mask': '255.255.255.0'}, {'
port': 2, 'mac': '00:00:00:00:06:02', 'ip': '192.168.203.2', 'mask': '255.255.255.0'}]
Routing Table  [{'network': '192.168.204.0/24', 'port': 1, 'scope': 'link', 'nexthop': None}, {'network': '192.168.203.0/24
', 'port': 2, 'scope': 'link', 'nexthop': None}, {'network': '10.0.1.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.1
68.203.1'}, {'network': '192.168.30.0/24', 'port': 2, 'scope': 'static', 'nexthop': '192.168.203.1'}, {'network': '172.0.1.
0/24', 'port': 1, 'scope': 'static', 'nexthop': '192.168.204.2'}]
loading app ryu.controller.ofp_handler
instantiating app Router.py of Router13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

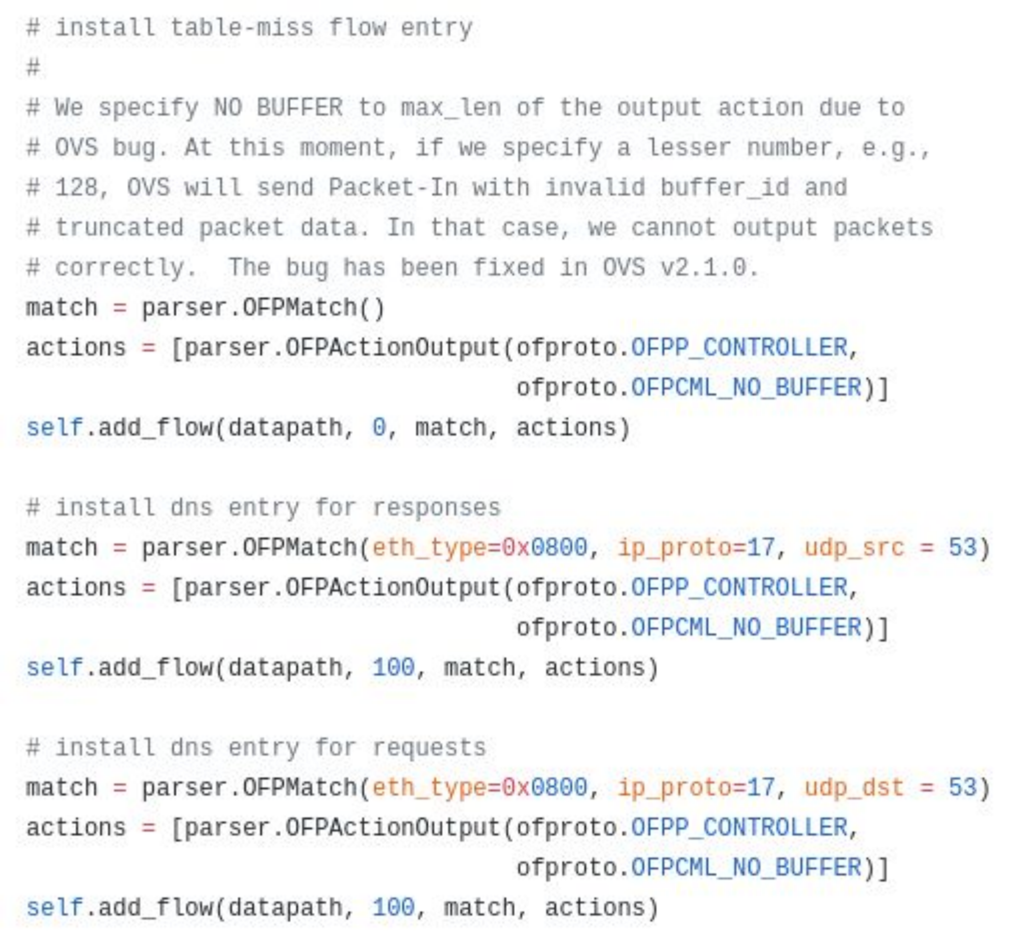
Switch controller runs on port 6654



```
mininet@mininet-VirtualBox:~/lop/network$ ryu-manager --ofp-tcp-listen-port 6654 --app-lists=Swtich.py
loading app Swtich.py
loading app ryu.controller.ofp_handler
instantiating app Swtich.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
█
```

Router/Switch Controller

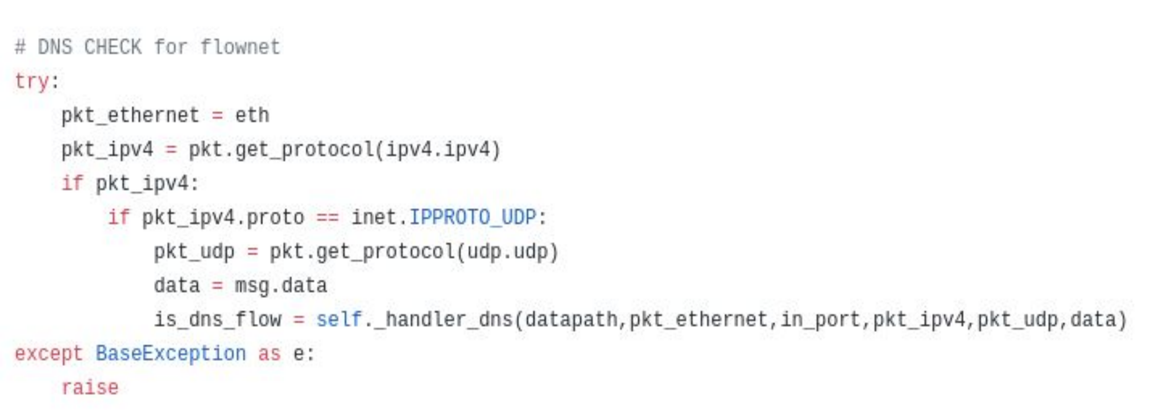
- Default entry for all DNS request, responses with high priority of 100 to forward to controller
- Matching protocol 17 for udp
- Matching eth_type 0x0800 for ipv4



```
# install table-miss flow entry
#
# We specify NO BUFFER to max_len of the output action due to
# OVS bug. At this moment, if we specify a lesser number, e.g.,
# 128, OVS will send Packet-In with invalid buffer_id and
# truncated packet data. In that case, we cannot output packets
# correctly.  The bug has been fixed in OVS v2.1.0.
match = parser.OFPMatch()
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                  ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

# install dns entry for responses
match = parser.OFPMatch(eth_type=0x0800, ip_proto=17, udp_src = 53)
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                  ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 100, match, actions)

# install dns entry for requests
match = parser.OFPMatch(eth_type=0x0800, ip_proto=17, udp_dst = 53)
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                  ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 100, match, actions)
```

Router/Switch Controller

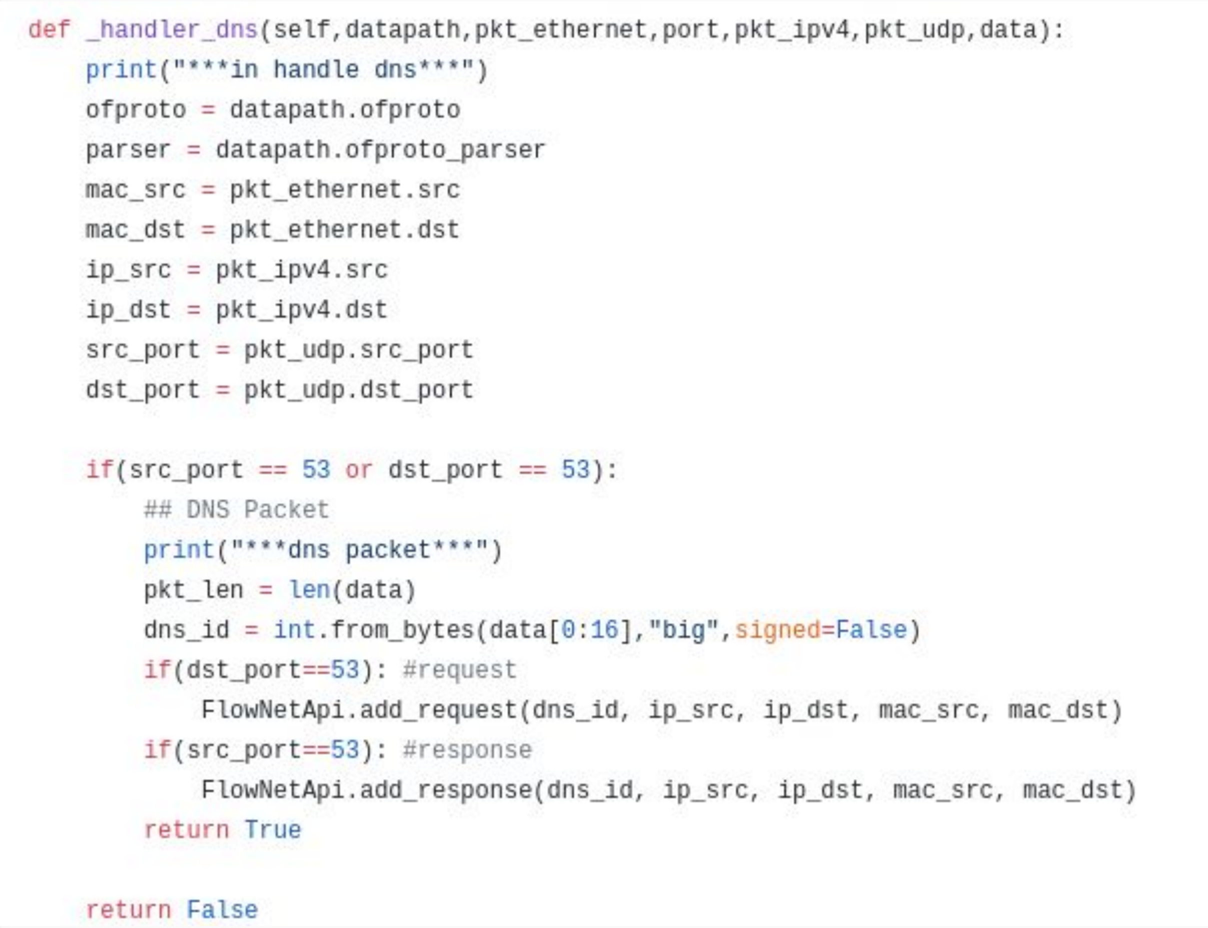
- inside packet in handler
- call _handle_dns() if it is a UDP packet
- handle dns returns true if it was a DNS packet



```
# DNS CHECK for flownet
try:
    pkt_ethernet = eth
    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    if pkt_ipv4:
        if pkt_ipv4.proto == inet.IPPROTO_UDP:
            pkt_udp = pkt.get_protocol(udp.udp)
            data = msg.data
            is_dns_flow = self._handler_dns(datapath,pkt_ethernet,in_port,pkt_ipv4,pkt_udp,data)
except BaseException as e:
    raise
```

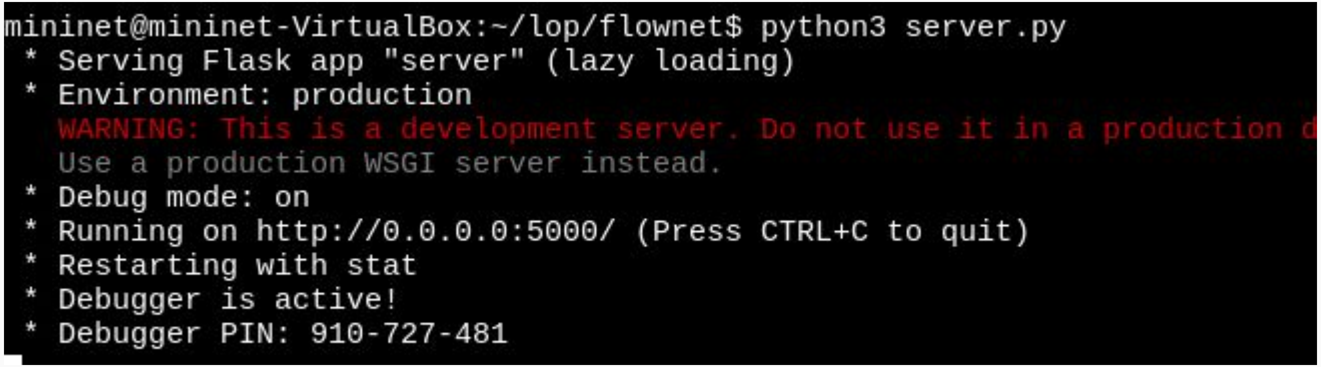
Router/Switch Controller

- Handle DNS on router/switch controller.
- All UDP Packets arriving at controller are forwarded to this function. It checks if it is a DNS packet and forwards details to FlowNet server



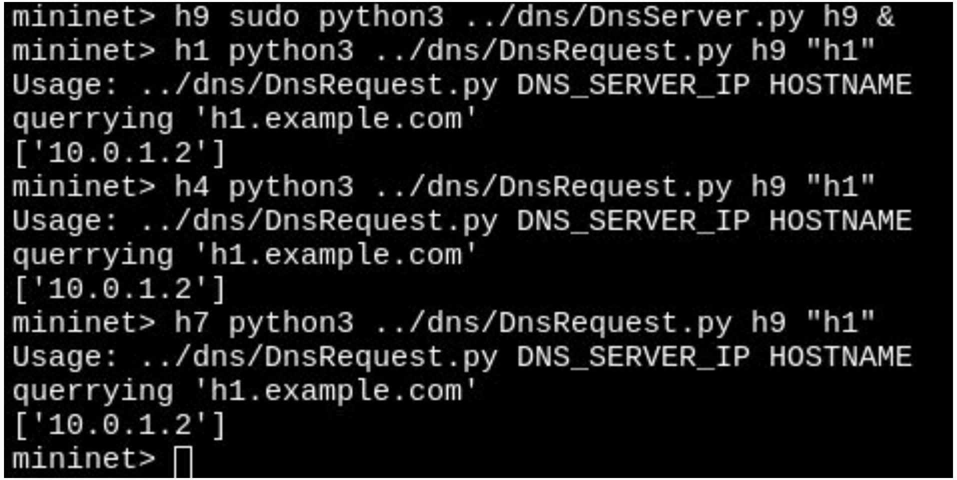
```
def _handler_dns(self,datapath,pkt_ethernet,port,pkt_ipv4,pkt_udp,data):
    print("***in handle dns***")
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    mac_src = pkt_ethernet.src
    mac_dst = pkt_ethernet.dst
    ip_src = pkt_ipv4.src
    ip_dst = pkt_ipv4.dst
    src_port = pkt_udp.src_port
    dst_port = pkt_udp.dst_port

    if(src_port == 53 or dst_port == 53):
        ## DNS Packet
        print("***dns packet***")
        pkt_len = len(data)
        dns_id = int.from_bytes(data[0:16],"big",signed=False)
        if(dst_port==53): #request
            FlowNetApi.add_request(dns_id, ip_src, ip_dst, mac_src, mac_dst)
        if(src_port==53): #response
            FlowNetApi.add_response(dns_id, ip_src, ip_dst, mac_src, mac_dst)
        return True

    return False
```

FlowNet Server



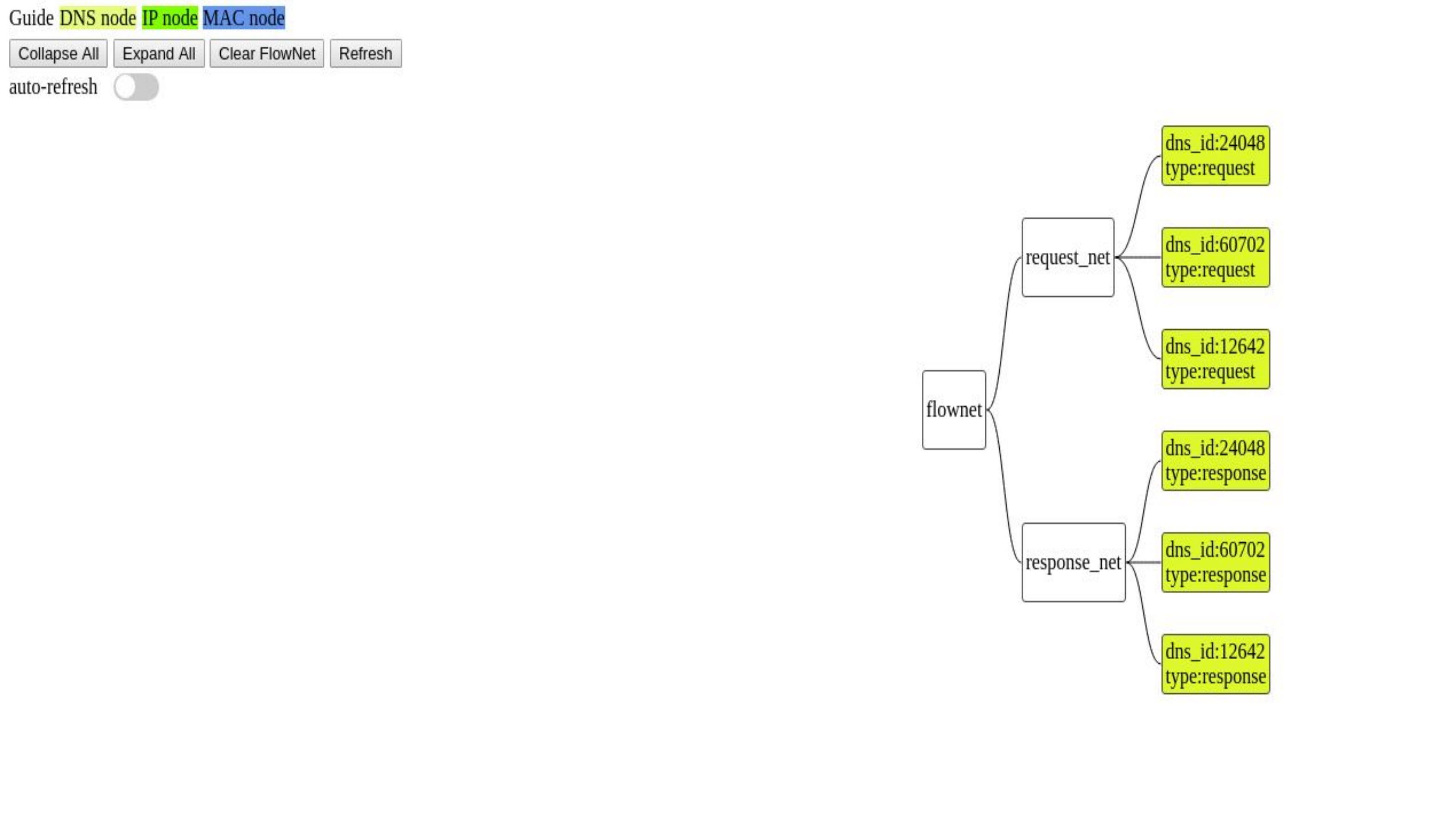
```
mininet@mininet-VirtualBox:~/lop/flownet$ python3 server.py
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production d
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 910-727-481
```

Mininet Shell



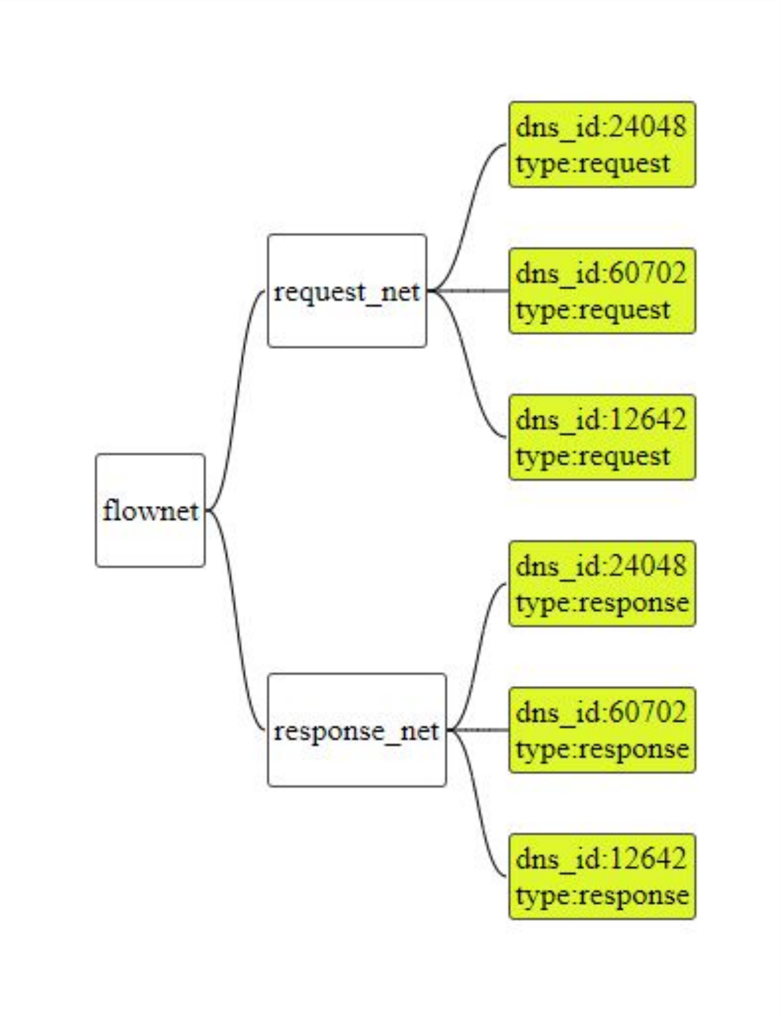
```
mininet> h9 sudo python3 ../dns/DnsServer.py h9 &
mininet> h1 python3 ../dns/DnsRequest.py h9 "h1"
Usage: ../dns/DnsRequest.py DNS_SERVER_IP HOSTNAME
querrying 'h1.example.com'
['10.0.1.2']
mininet> h4 python3 ../dns/DnsRequest.py h9 "h1"
Usage: ../dns/DnsRequest.py DNS_SERVER_IP HOSTNAME
querrying 'h1.example.com'
['10.0.1.2']
mininet> h7 python3 ../dns/DnsRequest.py h9 "h1"
Usage: ../dns/DnsRequest.py DNS_SERVER_IP HOSTNAME
querrying 'h1.example.com'
['10.0.1.2']
mininet> 
```

FlowNet Server UI



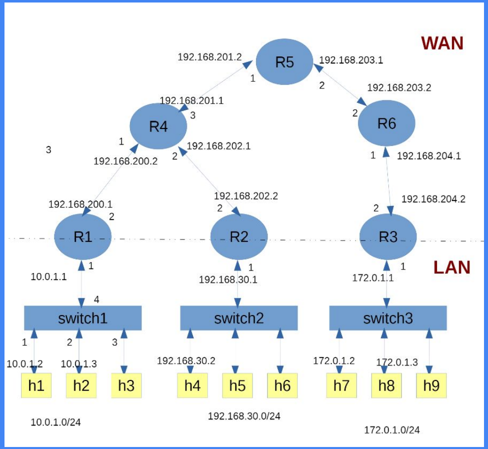
Output DNS

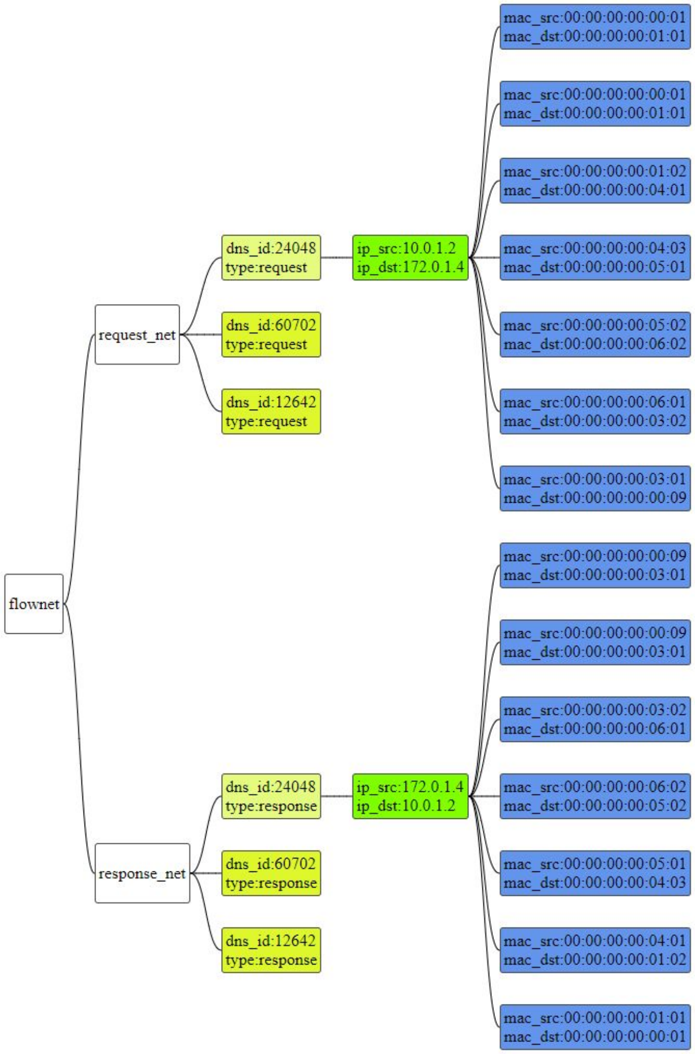
The 3 requests and responses created



Output h1-h9

Showing h1-h9 expanded




flownet

request_net

dns_id:24048
type:request

ip_src:10.0.1.2
ip_dst:172.0.1.4

mac_src:00:00:00:00:00:01
mac_dst:00:00:00:00:01:01

mac_src:00:00:00:00:00:01
mac_dst:00:00:00:00:01:01

mac_src:00:00:00:00:01:02
mac_dst:00:00:00:00:04:01

mac_src:00:00:00:00:04:03
mac_dst:00:00:00:00:05:01

mac_src:00:00:00:00:05:02
mac_dst:00:00:00:00:06:02

mac_src:00:00:00:00:06:01
mac_dst:00:00:00:00:03:02

mac_src:00:00:00:00:03:01
mac_dst:00:00:00:00:00:09

dns_id:60702
type:request

dns_id:12642
type:request

response_net

dns_id:24048
type:response

ip_src:172.0.1.4
ip_dst:10.0.1.2

mac_src:00:00:00:00:00:09
mac_dst:00:00:00:00:03:01

mac_src:00:00:00:00:00:09
mac_dst:00:00:00:00:03:01

mac_src:00:00:00:00:03:02
mac_dst:00:00:00:00:06:01

mac_src:00:00:00:00:06:02
mac_dst:00:00:00:00:05:02

mac_src:00:00:00:00:05:01
mac_dst:00:00:00:00:04:03

mac_src:00:00:00:00:04:01
mac_dst:00:00:00:00:01:02

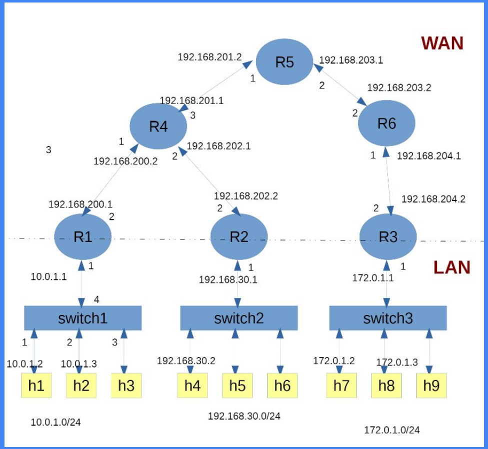
mac_src:00:00:00:00:01:01
mac_dst:00:00:00:00:00:01

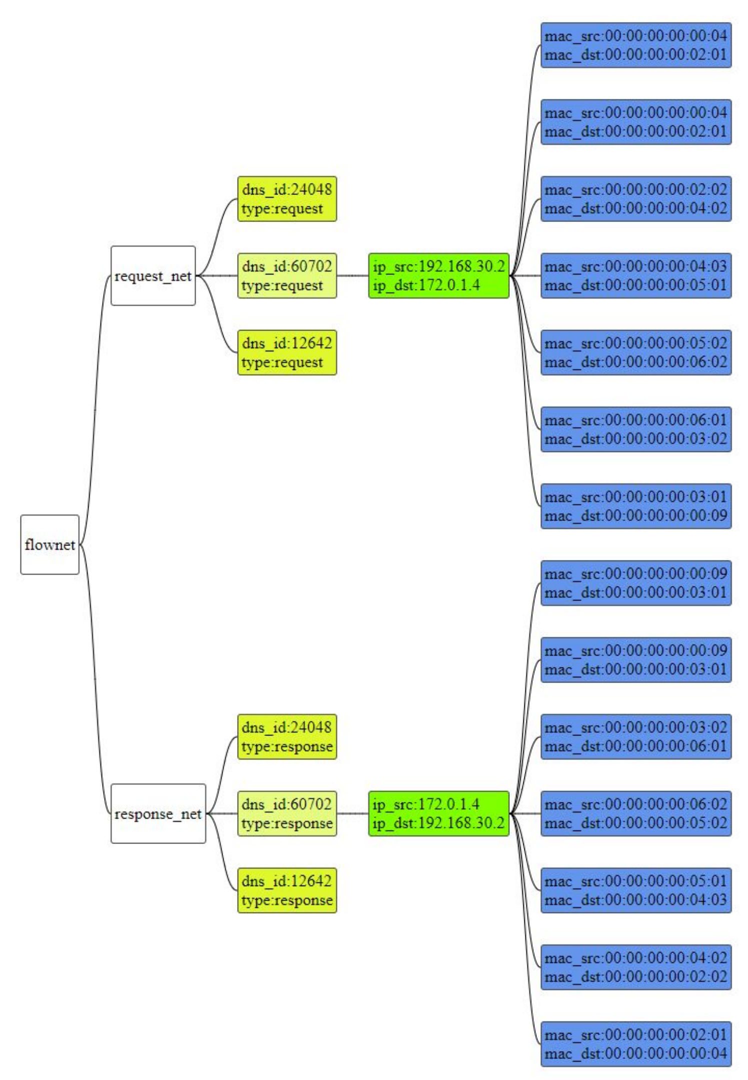
dns_id:60702
type:response

dns_id:12642
type:response

Output h4-h9

Showing h4-h9 expanded





flownet

request_net

dns_id:24048
type:request

dns_id:60702
type:request

dns_id:12642
type:request

ip_src:192.168.30.2
ip_dst:172.0.1.4

mac_src:00:00:00:00:00:04
mac_dst:00:00:00:00:02:01

mac_src:00:00:00:00:00:04
mac_dst:00:00:00:00:02:01

mac_src:00:00:00:00:02:02
mac_dst:00:00:00:00:04:02

mac_src:00:00:00:00:04:03
mac_dst:00:00:00:00:05:01

mac_src:00:00:00:00:05:02
mac_dst:00:00:00:00:06:02

mac_src:00:00:00:00:06:01
mac_dst:00:00:00:00:03:02

mac_src:00:00:00:00:03:01
mac_dst:00:00:00:00:00:09

response_net

dns_id:24048
type:response

dns_id:60702
type:response

dns_id:12642
type:response

ip_src:172.0.1.4
ip_dst:192.168.30.2

mac_src:00:00:00:00:00:09
mac_dst:00:00:00:00:03:01

mac_src:00:00:00:00:00:09
mac_dst:00:00:00:00:03:01

mac_src:00:00:00:00:03:02
mac_dst:00:00:00:00:06:01

mac_src:00:00:00:00:06:02
mac_dst:00:00:00:00:05:02

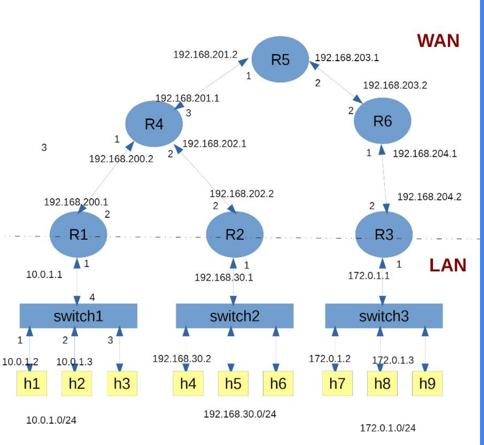
mac_src:00:00:00:00:05:01
mac_dst:00:00:00:00:04:03

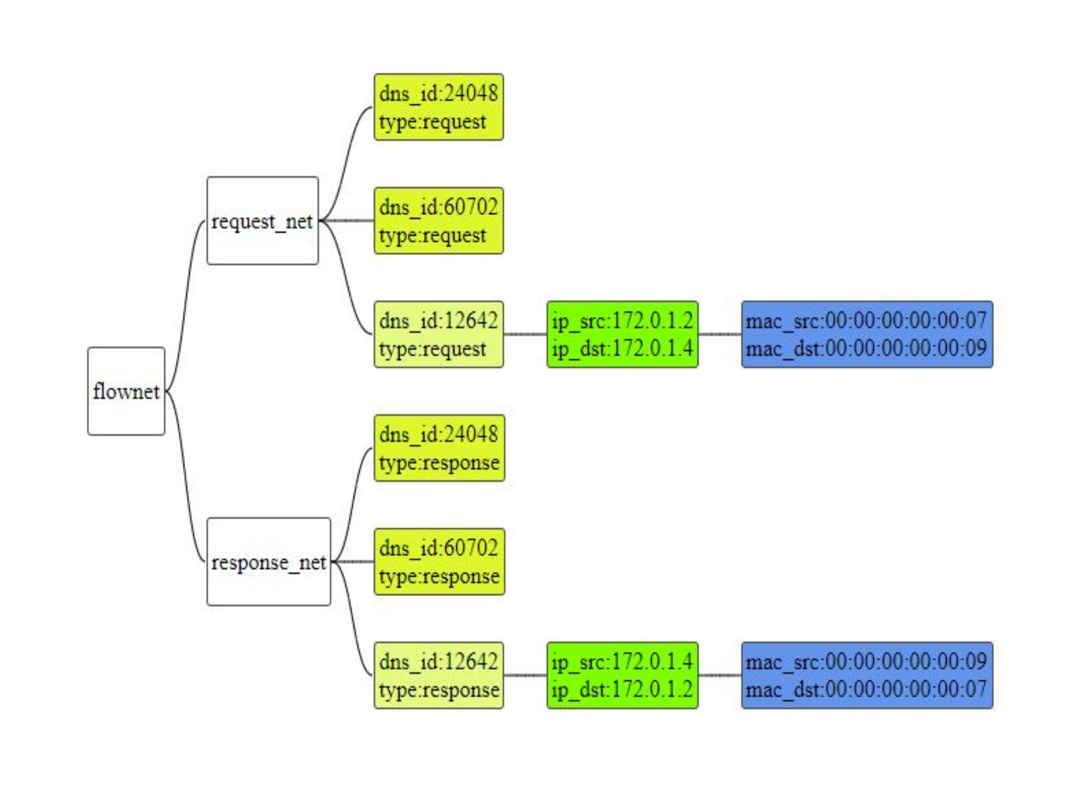
mac_src:00:00:00:00:04:02
mac_dst:00:00:00:00:02:02

mac_src:00:00:00:00:02:01
mac_dst:00:00:00:00:00:04

Output h7-h9

Showing h7-h9 expanded





flownet

request_net

dns_id:24048
type:request

dns_id:60702
type:request

dns_id:12642
type:request

ip_src:172.0.1.2
ip_dst:172.0.1.4

mac_src:00:00:00:00:00:07
mac_dst:00:00:00:00:00:09

response_net

dns_id:24048
type:response

dns_id:60702
type:response

dns_id:12642
type:response

ip_src:172.0.1.4
ip_dst:172.0.1.2

mac_src:00:00:00:00:00:09
mac_dst:00:00:00:00:00:07

Assumptions/Future Work

- We are only testing this on IPv4 network
- Single flow net machine is a bottleneck. Make this distributed
- Currently, flows are not removed from FlowNet. Though this
- DNS Standard specifies how DNS requests/responses can be broken into multiple packets. Current code ignores this problem
- We assume no NATs in the network
- We assume only UDP based DNS requests in network