Computer Networks: Assignment 2

Nimitt (22110169) & Tapananshu Gandhi (22110270)

A. Connectivity Check

For Part A, we built the specified network topology using Mininet's Python API. We created a custom `Topo` class that adds all nine nodes: the four hosts (H1-H4), the four switches (S1-S4), and the dedicated `dns` host (`10.0.0.5`).

The most critical part was configuring the links. We used Mininet's `TCLink` interface, which allowed us to set the **bandwidth (`bw`)** and **delay** for each link precisely as shown in the assignment diagram (e.g., `bw=100`, `delay='5ms'`).

After starting the network, we ran `net.pingAll()` to confirm full connectivity between all nodes, which passed successfully.

```
*** Ping: testing ping reachability
dns → h1 h2 h3 h4 nat0
h1 → dns h2 h3 h4 nat0
h2 → dns h1 h3 h4 nat0
h3 → dns h1 h2 h4 nat0
h4 → dns h1 h2 h3 nat0
nat0 → dns h1 h2 h3 h4
*** Results: 0% dropped (30/30 received)
```

B. Default Resolver

1. Methodology:

First, we wrote a separate Scapy script to parse the large PCAP files. This script filtered for all `DNSQR` packets, extracted the unique domain names, and saved them to simple `.txt` files. This was much more efficient than parsing the PCAP file on every run.

Next, we had to provide internet access to the hosts. We added a `net.addNAT()` node to our script. We also had to fix the VM's `eth1` (NAT) adapter, which was `DOWN` by default, by enabling it with `sudo dhclient eth1` and making the change

permanent in `/etc/netplan/`. Here all the hosts are requesting one by one and not parallelly at the same time.

2. Resolver Selection:

- **Initial Check (`/etc/resolv.conf`):** We first checked the hosts' `resolv.conf` file, which showed the default nameserver was `127.0.0.53`. This is an internal `systemd-resolved` stub resolver. It is inaccessible from within Mininet's isolated host namespaces, so it could not be used.
- **Upstream Resolver (`resolvectl status`):** We then used `resolvectl status` to find the VM's "true" upstream resolver, which was `10.0.136.7`. However, all `dig` queries to this IP from `h1` timed out. We concluded this is a private resolver (likely for our university/organization) that is firewalled and does not accept queries from our VirtualBox NAT adapter's IP (`10.0.2.15`).
- **Gateway Resolver (`ip route`):** We also identified the VirtualBox NAT gateway (`10.0.2.2`) as a potential resolver. This also did not respond.

To get a meaningful and realistic baseline for public internet DNS performance, we selected Google's public DNS server (`8.8.8.8`)

3. Data Collection:

Our script iterated through each host's domain list and ran `host.cmd(f'dig {domain} @8.8.8.8')`. We then used Python's `re` module to parse the `dig` output for two key fields:

- **Latency:** The `Query time: X msec` value.
- **Success:** The `status: NOERROR` flag.

For "average throughput," we measured **Queries Per Second (QPS)** using the formula `success_count / total_time`. We chose QPS because bandwidth is not the bottleneck for small DNS packets; transaction speed is the correct metric.

```
Starting DNS lookups for 100 domains...
Results for h1:
   - Successfully Resolved: 76
   - Failed Resolutions:    24
   - Average Lookup Latency: 290.26 ms
   - Average Throughput:    2.76 queries/sec

Starting DNS lookups for 100 domains...
```

```
Results for h2:
   - Successfully Resolved: 72
   - Failed Resolutions:    28
   - Average Lookup Latency: 281.50 ms
   - Average Throughput:    2.63 queries/sec

Starting DNS lookups for 100 domains...
Results for h3:
   - Successfully Resolved: 71
   - Failed Resolutions:    29
   - Average Lookup Latency: 238.83 ms
   - Average Throughput:    2.96 queries/sec

Starting DNS lookups for 100 domains...
Results for h4:
   - Successfully Resolved: 77
   - Failed Resolutions:    23
   - Average Lookup Latency: 270.12 ms
   - Average Throughput:    3.10 queries/sec
```

Obersevations: The reason for failed resolutions: We found that, the URL domains of failed domains did not exist, or their server was down as upon trying to query/open those websites through browser they couldn't be reached. We also checked this by trying to run all domains using python.

C. DNS Configuration of Mininet Hosts

We need to update DNS resolver configuration files for each host as shown below:

```
# Update resolv.conf for hosts
      for h in ['h1', 'h2', 'h3', 'h4']:
         host = net.get(h)
         host.cmd('echo "nameserver 10.0.0.5" > /etc/resolv.conf')
```

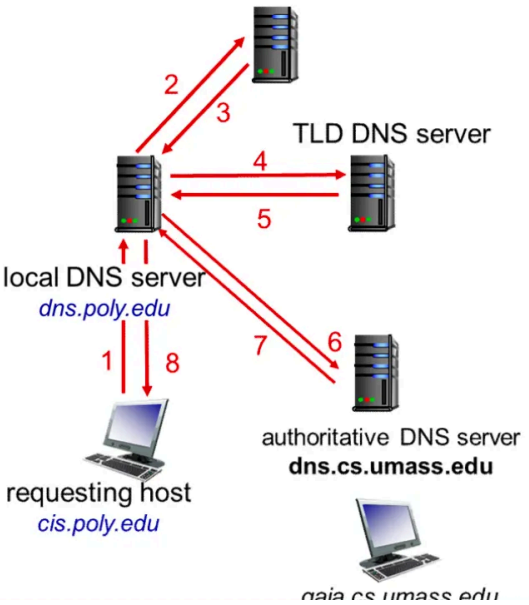
Updated configuration files for each host:

```
mininet> h1 cat /etc/resolv.conf
nameserver 10.0.0.5
mininet> h2 cat /etc/resolv.conf
nameserver 10.0.0.5
mininet> h3 cat/etc/resolv.conf
bash: cat/etc/resolv.conf: No such file or directory
mininet> h3 cat etc/resolv.conf
cat: etc/resolv.conf: No such file or directory
mininet> h3 cat /etc/resolv.conf
nameserver 10.0.0.5
mininet> h4 cat /etc/resolv.conf
nameserver 10.0.0.5
```

D. Custom DNS Resolver

Implementation

The implementation involves querying root servers, TLD servers and Authorative servers iteratively to get IP Address of lower-level DNS resolver till the final address is resolved.



Step Wise Execution:

1. Check Cache
2. Reach Root Server for TLD → TLD IP
3. Reach TLD for Authorative → Authorative IP
4. Reach Authorative for NS → NS IP/ NS URL
5. [If not glue]: Resolve NS URL
6. Reach NS → Final Address

Log timestamp, trace, and resolution data.

Execution

Step 1> Running the resolver `@127.0.0.1 port=53534`

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   JUPYTER

nimitt@Nimitts-MacBook-Air-10 A2 % python3 resolver.py
Starting custom DNS resolver with caching: True
Listening on 0.0.0.0:53534 with 50 workers...
```

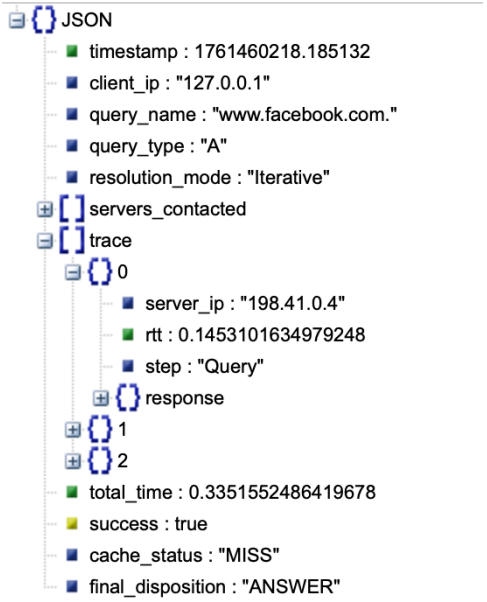
Step 2> Querying the resolver `www.facebook.com` with iterative mode. The resolver resolves the query and sends back the response and logs the corresponding resolution data.

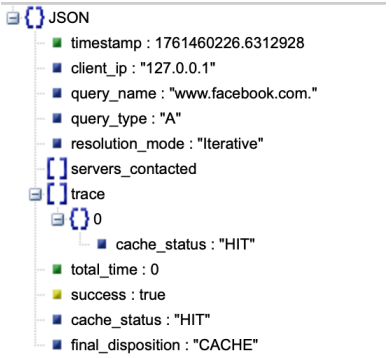
```
nimitt@Nimitts-MacBook-Air-10 A2 % dig @127.0.0.1 -p 53534 www.facebook.com +norecurse

; <<>> DiG 9.10.6 <<>> @127.0.0.1 -p 53534 www.facebook.com +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44199
;; flags: qr; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 8192
;; QUESTION SECTION:
;www.facebook.com.              IN      A

;; ANSWER SECTION:
www.facebook.com.       3600    IN      CNAME   star-mini.c10r.facebook.com.
star-mini.c10r.facebook.com. 60 IN      A       57.144.242.1

;; Query time: 339 msec
;; SERVER: 127.0.0.1#53534(127.0.0.1)
;; WHEN: Sun Oct 26 12:00:18 IST 2025
;; MSG SIZE  rcvd: 90
```

Step 3> **CACHING:** When queried again with `www.facebook.com` resolver finds it in cache and sends cached resolution as seen in query time and logs.

```
nimitt@Nimitts-MacBook-Air-10 A2 % dig @127.0.0.1 -p 53534 www.facebook.com +norecurse

; <<>> DiG 9.10.6 <<>> @127.0.0.1 -p 53534 www.facebook.com +norecurse
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2893
;; flags: qr; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 8192
;; QUESTION SECTION:
;www.facebook.com.              IN      A

;; ANSWER SECTION:
www.facebook.com.       3600    IN      CNAME   star-mini.c10r.facebook.com.
star-mini.c10r.facebook.com. 60 IN      A       57.144.242.1

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53534(127.0.0.1)
;; WHEN: Sun Oct 26 12:00:26 IST 2025
;; MSG SIZE  rcvd: 90
```

LOGGING: Resolver logs the required resolution details as JSON object for each query. Resolver's log for 1st and 2nd (Cached) query.





Execution with DNS in Mininet as our resolver

Now we run our custom resolver from the dns node in mininet topology to resolver `H{i}` 's queries with `CACHE = FALSE` .

Step 1> Configure hosts (Part C).

Step 2> Run `resolver.py` from dns node and start querying the dns sequentially for each host (Assumption).

Resolution Details:

Link to Resolver's logs: Link

```
Starting custom DNS resolver on dns node...

Starting DNS lookups for 100 do main
Results for h1:
  - Successfully Resolved: 69
  - Failed Resolutions:    31
  - Average Lookup Latency: 827
```

```
Starting DNS lookups for 100 do main
Results for h3:
  - Successfully Resolved: 71
  - Failed Resolutions:    29
  - Avg Lookup Latency: 1013.65 ms
  - Average Throughput:    0.73 q/s
```

ms
 - Average Throughput: 0.79 q/s

Starting DNS lookups for 100 domain
Results for h2:
 - Successfully Resolved: 66
 - Failed Resolutions: 34
 - Average Lookup Latency: 839.4 m
 - Average Throughput: 0.87 q/s

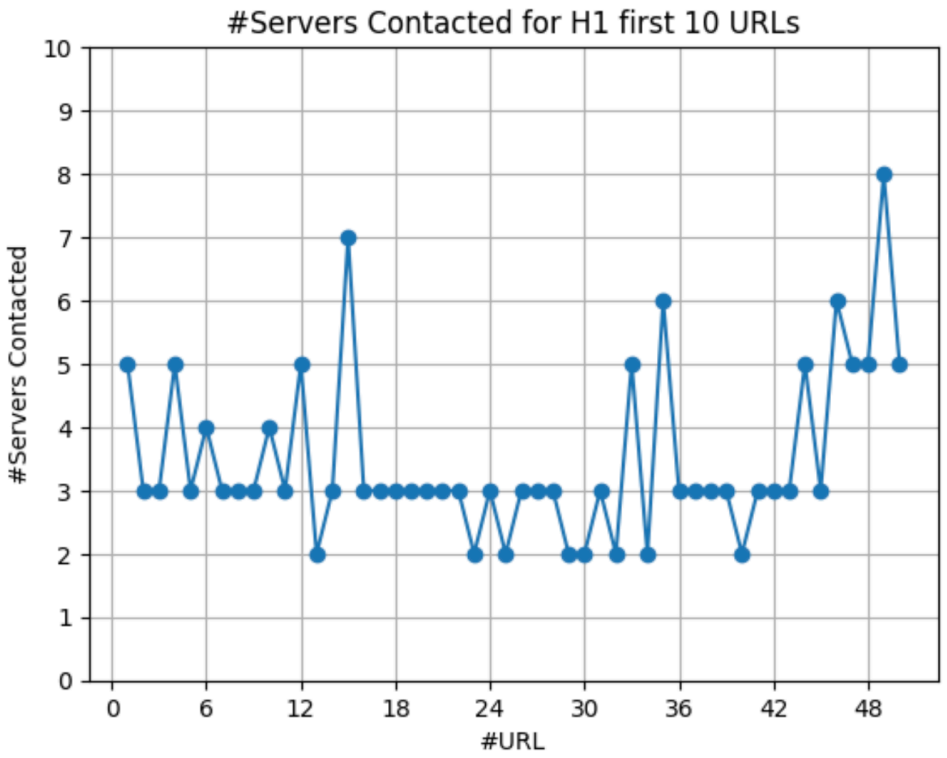
Starting DNS lookups for 100 domain
Results for h4:
 - Successfully Resolved: 71
 - Failed Resolutions: 29
 - Avg Lookup Latency: 1000.14 ms
 - Average Throughput: 0.71 q/s
Stopping network...

(q/s: queries per second)

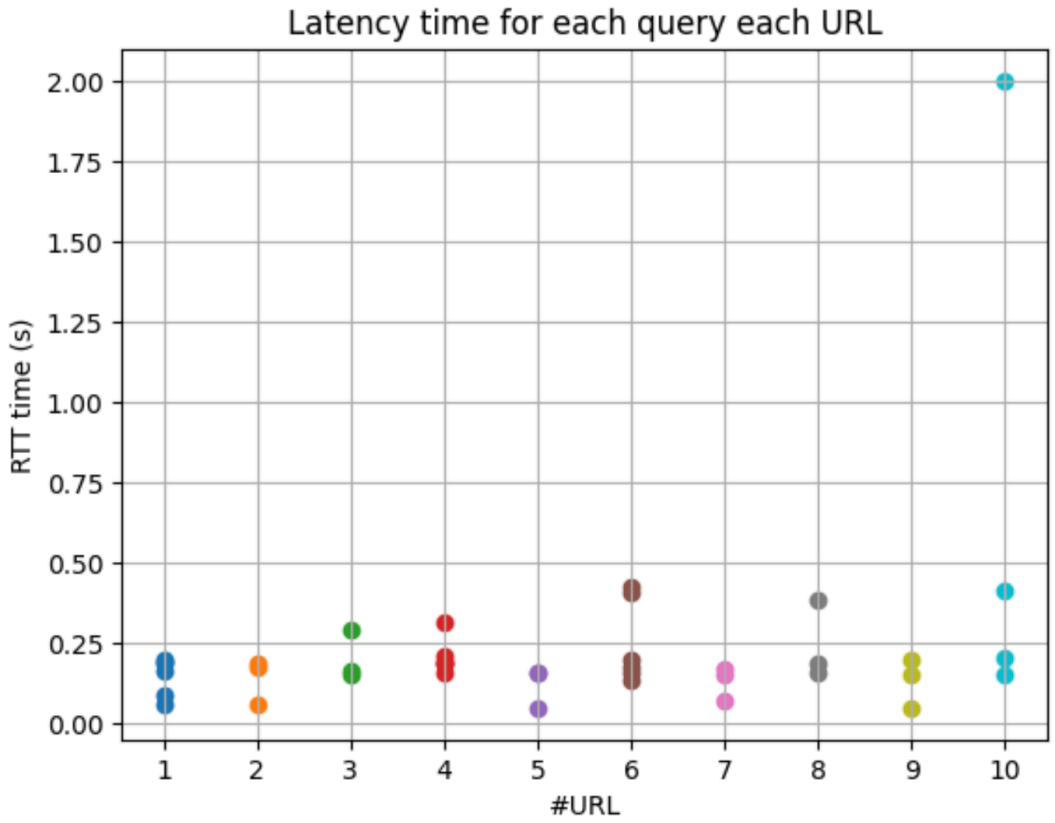
Visualization

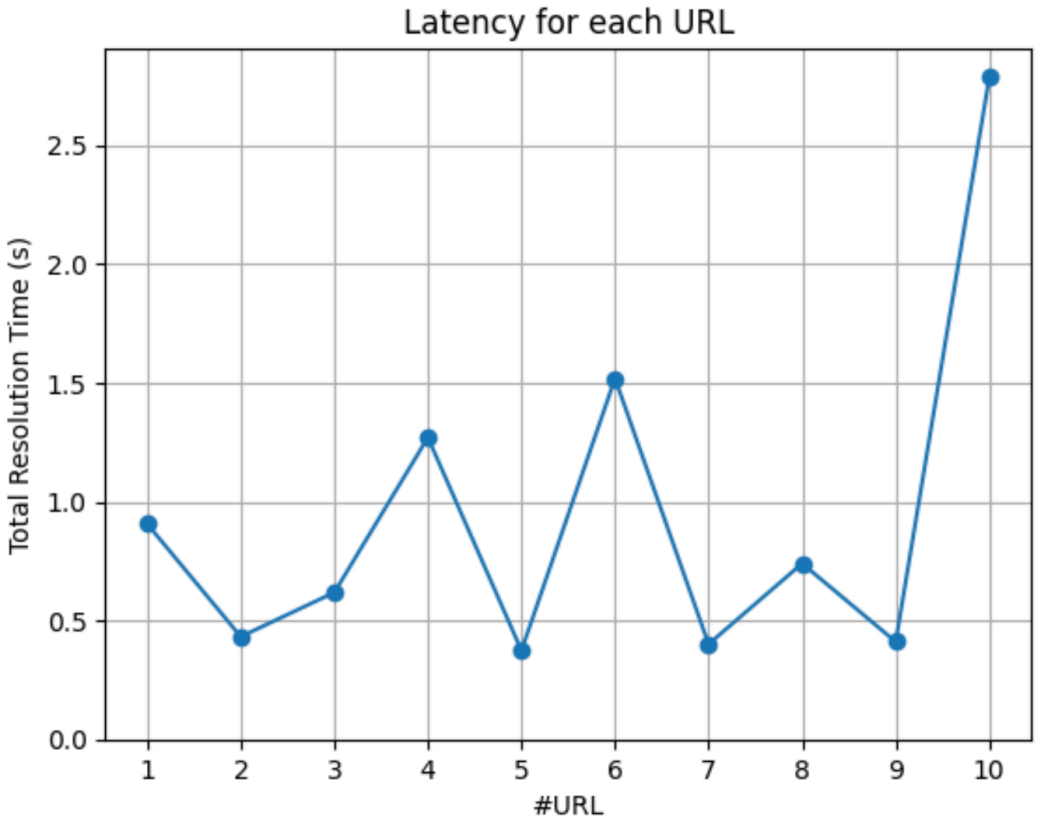
Number of Servers Contacted

Average #Servers Contacted for all URLs = 3.53



Latency per query


Latency time for each query each URL
RTT time (s)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7
8
9
10
#URL


Latency for each URL
Total Resolution Time (s)
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
8
9
10
#URL

Comparison with Default Resolver

Our custom resolver is slower than the default resolver (as observed in Part B). The reasons for this is:

\> The custom resolver does not have cache unlike default resolver.

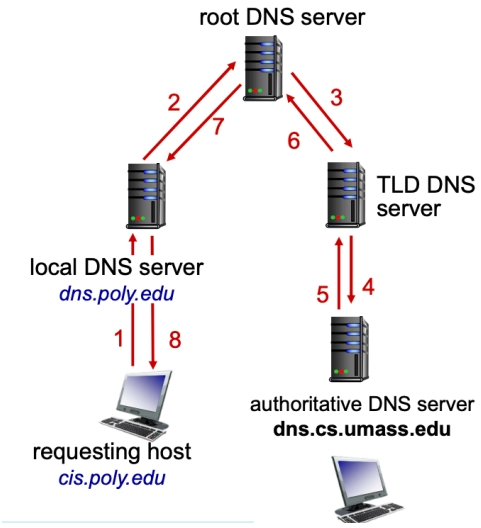
\> The custom resolver has to log all the resolution data for each query. This involves I/O and leads to latency.

\> The custom resolver does not have any optimisations to avoid dead and far DNS servers.

E. Recursive Resolver

Recursive resolver could not be implemented because any the root servers are not responding to recursive query.

Source: Lecture slides. Recursive resolver could not be implemented as recursion is not available in root servers.




```
nimitt@Nimitts-MacBook-Air-10 A2 % dig @202.12.27.33 -p 53 www.google.com

; <<>> DiG 9.10.6 <<>> @202.12.27.33 -p 53 www.google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50593
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
;; WARNING: recursion requested but not available
```

F. Cache

Resolution Details

Link to Resolver's logs: Link

```
Starting custom DNS resolver on dns node...

Starting DNS lookups for 100 do main
Results for h1:
- Successfully Resolved: 70
- Failed Resolutions:    30
- Average Lookup Latency: 800.1 3 ms
- Average Throughput:    0.86 q/s

Starting DNS lookups for 100 do main
Results for h2:
- Successfully Resolved: 67
- Failed Resolutions:    33
- Average Lookup Latency: 743.1 8 ms
- Average Throughput:    1.01 q/s

Starting DNS lookups for 100 do main
Results for h3:
- Successfully Resolved: 69
- Failed Resolutions:    31
- Average Lookup Latency: 777.3 5 ms
- Average Throughput:    0.89 q/s

Starting DNS lookups for 100 do main
Results for h4:
- Successfully Resolved: 72
- Failed Resolutions:    28
- Average Lookup Latency: 804.2 1 ms
- Average Throughput:    0.90 q/s

 Stopping network...
```

Cache Hit Rate

Total Number of Times Cache Searched = `400` URLs `+ 142` NS Resolutions = `542`

Number of Times Cache Hit = `21`

Cache Hit Rate = `21/542` = `3.9 %`

Observations

\> Caching gives higher average througput as we are able to resolve already queried servers much quickly.