CS331 Assignment 2 Report

Dhruv Sharma (22110074)
Astitva Aryan (22110041)

October 28, 2025

Project Repository

The code for this assignment can be found at: `https://github.com/Dhruv-Sharma01/CN_ASSN2`

Task A: Mininet Topology Simulation

Approach Used

The approach used to generate the provided output was as follows:

1. A network topology specified in the assignment was instantiated within the Mininet network emulator using the provided Python script (`DNS_topo.py`).
2. This topology includes four client hosts (`h1`, `h2`, `h3`, `h4`), one DNS resolver node (`dns`), four switches (`s1`, `s2`, `s3`, `s4`), and one NAT gateway (`nat`) for external connectivity.
3. Hosts were connected to their respective switches (`h1-s1`, `h2-s2`, `h3-s3`, `h4-s4`), and the DNS resolver was connected to switch `s2`. Bandwidth and delay parameters were set as per the assignment diagram.
4. Switches were linked linearly (`s1-s2-s3-s4`) with specified bandwidths and delays. The NAT node was connected to switch `s2`.
5. IP addresses were assigned as specified (e.g., `h1: 10.0.0.1`, `dns: 10.0.0.5`), and default routes were set via the NAT node (`10.0.0.254`).
6. From the Mininet Command Line Interface (CLI), the `pingall` command was executed.
7. This `pingall` utility automatically performs an all-pairs connectivity test among the Mininet-managed hosts (`h1, h2, h3, h4, dns`), sending ICMP Echo Request (ping) packets to verify Layer 3 (IP) reachability.

Code Snippets

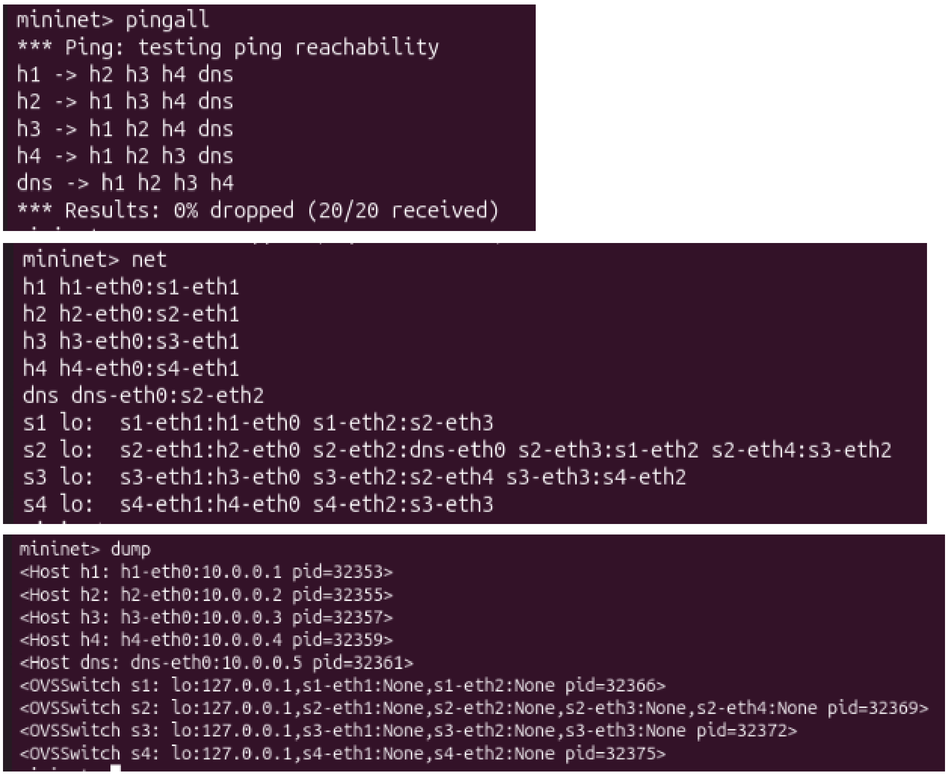
The Mininet topology was created using the Python script `DNS_topo.py`. The script defines the hosts, switches, and links with their respective parameters.

The `pingall` command was executed within the Mininet CLI after starting the network:

```
mininet> pingall
```

Output Image

Figure 1 shows the console output after running the `pingall` and other commands in the specified topology.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 dns
h2 -> h1 h3 h4 dns
h3 -> h1 h2 h4 dns
h4 -> h1 h2 h3 dns
dns -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
dns dns-eth0:s2-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth3
s2 lo:  s2-eth1:h2-eth0 s2-eth2:dns-eth0 s2-eth3:s1-eth2 s2-eth4:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth4 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3
```

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=32353>
<Host h2: h2-eth0:10.0.0.2 pid=32355>
<Host h3: h3-eth0:10.0.0.3 pid=32357>
<Host h4: h4-eth0:10.0.0.4 pid=32359>
<Host dns: dns-eth0:10.0.0.5 pid=32361>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=32366>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None pid=32369>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=32372>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=32375>
```

Figure 1: Mininet `pingall` and other commands output for Task A topology.

Justification of Results

The results shown in Figure 1 are **expected** and **correct** for the specified topology.

- **Full Connectivity:** The output confirms that each host (`h1` to `h4` and `dns`) could successfully reach every other host in the Mininet network.
- **0% Packet Loss:** The summary line confirms no packets were dropped (`0% dropped`) and all packets were received (`20/20 received`).
- **Correct Test Count:** For $n = 5$ hosts (`h1-h4, dns`), `pingall` performs $n \times (n-1) = 5 \times 4 = 20$ tests. The `(20/20 received)` metric confirms all these tests passed.

This validates the basic Layer 3 connectivity within the simulated network.

Task B: DNS Resolution with Default Resolver

Approach Used

For each host (`h1` through `h4`), the main script (`main.py`) was executed from the Mininet CLI. This script likely utilized the `pcap_processor.py` to read URLs from the host's respective input file (e.g., `PCAP_1_H1.pcap`) and initiated DNS lookups using the standard system resolver functions (as the custom resolver was not configured yet). After processing all URLs, the script calculated and printed summary statistics including the number of successful and failed resolutions, average lookup latency, and average throughput, potentially using functions from `viz_dns.py` for calculation or formatting.

Results

Table 1 summarizes the results obtained from running the client script on each host with the default system resolver, based on the data in `comparison_results.txt`.

Table 1: Task B: Default Resolver Performance Summary

Metric	H1	H2	H3	H4
Total Queries	100	100	100	100
Successfully Resolved	71	62	28	71
Failed Resolutions	29	38	72	29
Avg Lookup Latency (ms)	171.86	345.74	161.07	686.59
Avg Throughput (qps)	30.98	17.79	8.59	14.77

Analysis

The performance varied significantly across hosts. H1 and H4 achieved the highest success rate (71%), while H3 had a very low success rate (28%). Latency and throughput also varied, likely due to differences in the specific URLs queried by each host and potential network conditions affecting the default resolver's external queries. H3's particularly poor performance might indicate a higher proportion of difficult-to-resolve or non-existent domains in its list.

Task C: Configuring Custom DNS Resolver

Approach Used

The goal was to redirect DNS queries from client hosts (`h1`, `h2`, `h3`, `h4`) to the custom resolver node (`dns` at `10.0.0.5`). This was achieved by modifying the standard DNS configuration file on each client host.

1. Using the Mininet CLI, the `/etc/resolv.conf` file on `h1` was overwritten with the line `nameserver 10.0.0.5` using shell redirection.

```
mininet> h1 echo 'nameserver 10.0.0.5' > /etc/resolv.conf
```

2. This step was repeated identically for hosts `h2`, `h3`, and `h4`.

3. The change was verified by displaying the contents of the file on `h1` using the `cat` command.

```
mininet> h1 cat /etc/resolv.conf
```

Output Image

Figure 2 shows the console output for these commands.

```
mininet> h1 echo 'nameserver 10.0.0.5' > /etc/resolv.conf
mininet> h2 echo 'nameserver 10.0.0.5' > /etc/resolv.conf
mininet> h3 echo 'nameserver 10.0.0.5' > /etc/resolv.conf
mininet> h4 echo 'nameserver 10.0.0.5' > /etc/resolv.conf
```

```
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
nameserver 10.0.0.5
mininet> h4 cat /etc/resolv.conf
nameserver 10.0.0.5
```

```
mininet> h1 ping -c 3 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=18.9 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=27.7 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=90.0 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 18.884/45.520/89.962/31.631 ms
mininet> h2 ping -c 3 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=11.0 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=7.88 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=12.8 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2071ms
rtt min/avg/max/mdev = 7.882/10.560/12.786/2.027 ms
```

Figure 2: Mininet CLI output showing configuration and verification for Task C.

Justification of Results

The output in Figure 2 confirms the successful reconfiguration. The `cat` command executed on `h1` correctly displays `nameserver 10.0.0.5`, verifying that the `/etc/resolv.conf` file was correctly updated to point to the custom DNS resolver node. This procedure correctly modifies the DNS settings for the Mininet hosts as required.

Task D: DNS Resolution with Custom Iterative Resolver (No Cache)

Approach Used

The tests were run by executing the `main.py` script on each client host (`h1` through `h4`). This script invoked `pcap_processor.py` to read URLs and then initiated queries. Due to the configuration in Task C, these queries were sent to the custom DNS resolver running at `10.0.0.5` (implemented in `custom_dns_resolver.py`). This resolver was configured for iterative lookups without caching.

For each query, the custom resolver performed the iterative steps (Root → TLD → Authoritative) and logged detailed information as required by the assignment (timestamp, domain, mode, server IP, step, response, RTT, total time), as evidenced in the provided `part_d_*.pcap.log` files (and their respective csv files) uploaded on Github repository. The `main.py` script collected results to calculate performance metrics, potentially using `viz_dns.py` for analysis or plotting.

Results

Timestamp	Domain name qu	Resolution mode	DNS server IP c	Step of resolutio	Response or referral received	Round-trip time to that server	Total time to resolution
1761487276	tuhafhaberler.co	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	160.4924202	
1761487276	tuhafhaberler.co	Iterative	192.41.162.30	TLD	ERROR: All nameservers failed to answer the query a.g	142.1983242	323.580265
1761487277	41latitude.com	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	143.1341171	
1761487277	41latitude.com	Iterative	192.41.162.30	TLD	REFERRAL to 156.154.132.200	142.5228119	
1761487277	41latitude.com	Iterative	156.154.132.200	Authoritative	ERROR: The resolution lifetime expired after 5.220 sec	50.09388924	5556.768656
1761487282	us.tf	Iterative	198.41.0.4	Root	REFERRAL to 194.146.106.46	142.4417496	
1761487282	us.tf	Iterative	194.146.106.46	TLD	REFERRAL to ns1.idnscan.net. (needs glue lookup)	155.9686661	
1761487283	us.tf	Iterative	95.130.17.35	Authoritative	ERROR: The DNS operation timed out.		2698.556423
1761487285	tottenhamhk.con	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	155.4570198	
1761487285	tottenhamhk.con	Iterative	192.41.162.30	TLD	ERROR: All nameservers failed to answer the query a.g	158.7514877	343.2817459
1761487285	running-sigi.de	Iterative	198.41.0.4	Root	REFERRAL to 194.0.0.53	143.3339119	
1761487285	running-sigi.de	Iterative	194.0.0.53	TLD	ERROR: All nameservers failed to answer the query f.ni	171.1809635	337.9030228
1761487285	energybulbs.co.u	Iterative	198.41.0.4	Root	REFERRAL to 213.248.216.1	140.212059	
1761487286	energybulbs.co.u	Iterative	213.248.216.1	TLD	REFERRAL to ns0.lcn.com. (needs glue lookup)	165.5914783	
1761487286	energybulbs.co.u	Iterative	195.110.124.234	Authoritative	ANSWER: 23.227.38.32	157.3696136	765.8884525
1761487286	o-ov.com	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	139.2686367	
1761487286	o-ov.com	Iterative	192.41.162.30	TLD	REFERRAL to 2603:5:2180::5	142.6160336	
1761487287	o-ov.com	Iterative	2603:5:2180::5	Authoritative	ERROR: The DNS operation timed out.		2287.165403
1761487289	ex-jw.org	Iterative	198.41.0.4	Root	REFERRAL to 199.249.112.1	164.2279625	
1761487289	ex-jw.org	Iterative	199.249.112.1	TLD	REFERRAL to ns11.wixdns.net. (needs glue lookup)	72.87669182	
1761487289	ex-jw.org	Iterative	216.239.38.100	Authoritative	ANSWER: 185.230.63.171	154.5743942	412.0213985
1761487289	i-butterfly.ru	Iterative	198.41.0.4	Root	REFERRAL to 193.232.128.6	153.1760693	
1761487289	i-butterfly.ru	Iterative	193.232.128.6	TLD	REFERRAL to fred.ns.cloudflare.com. (needs glue look	219.3050385	
1761487289	i-butterfly.ru	Iterative	173.245.59.113	Authoritative	ANSWER: 172.67.151.138	42.35696793	438.7223721
1761487289	datepanchang.c	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	142.4462795	
1761487290	datepanchang.c	Iterative	192.41.162.30	TLD	REFERRAL to 108.162.194.214	141.3414478	
1761487290	datepanchang.c	Iterative	108.162.194.214	Authoritative	ANSWER: 13.75.66.141	496.8838692	781.5074921
1761487290	zzxu.cn	Iterative	198.41.0.4	Root	REFERRAL to 203.119.29.1	155.7941437	
1761487290	zzxu.cn	Iterative	203.119.29.1	TLD	REFERRAL to ns1.judns.com. (needs glue lookup)	171.6639996	
1761487291	zzxu.cn	Iterative	47.97.51.45	Authoritative	ERROR: The DNS operation timed out.		2755.865097
1761487293	daehepharma.cc	Iterative	198.41.0.4	Root	REFERRAL to 192.41.162.30	152.425766	
1761487293	daehepharma.cc	Iterative	192.41.162.30	TLD	REFERRAL to 220.73.163.40	143.5675621	
1761487293	daehepharma.cc	Iterative	220.73.163.40	Authoritative	ERROR: All nameservers failed to answer the query ns(	176.6602993	507.1306229

Figure 3: Snapshot of CSV generated for H1.pcap

Table 2 summarizes the overall performance of the custom iterative resolver based on `comparison_results.txt`.

Table 2: Task D: Custom Iterative Resolver (No Cache) Performance Summary

Metric	H1	H2	H3	H4
Total Queries	100	100	100	100
Successfully Resolved	51	52	50	47
Failed Resolutions	49	48	50	53
Avg Lookup Latency (ms)	735.84	765.47	903.79	983.67
Avg Throughput (qps)	11.84	12.22	14.88	13.91

H1 First 10 URLs Visualization

Figures 4 and 5 show the latency per query and the number of DNS servers visited for the first 10 URLs resolved by H1 using the custom iterative resolver.

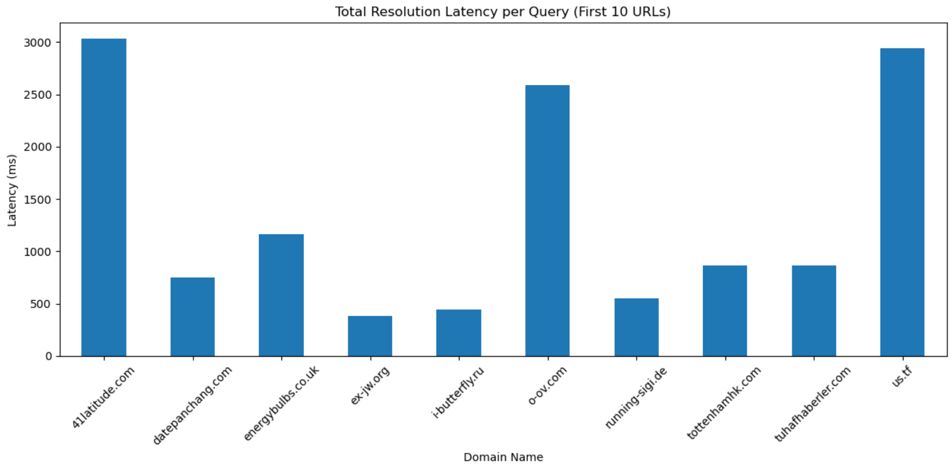


Figure 4: Task D: H1 - Latency per Query (First 10 URLs)

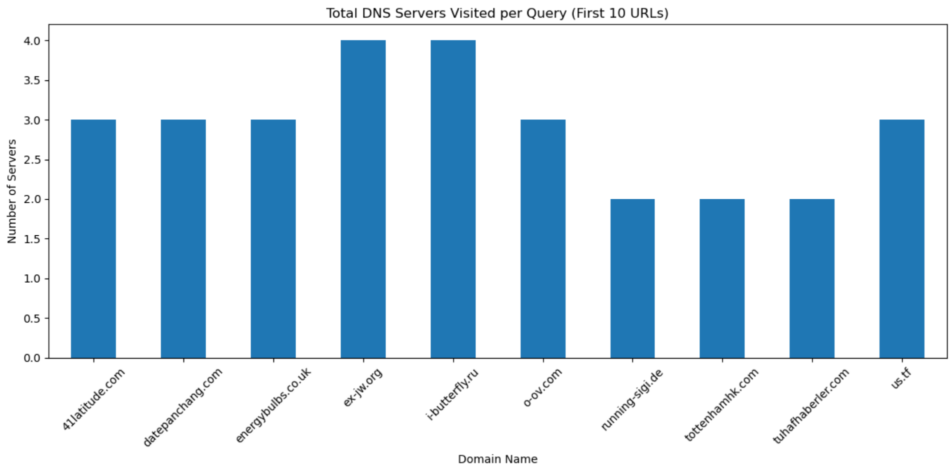


Figure 5: Task D: H1 - DNS Servers Visited per Query (First 10 URLs)

Most queries involved contacting 2, 3 or 4 servers. The latency varied significantly, with timeouts leading to very high reported latencies. Successful resolutions generally took several hundred milliseconds.

Comparison with Task B

Figures 6, 7, and 8 compare the performance of the default resolver (Task B) and the custom iterative resolver (Task D).

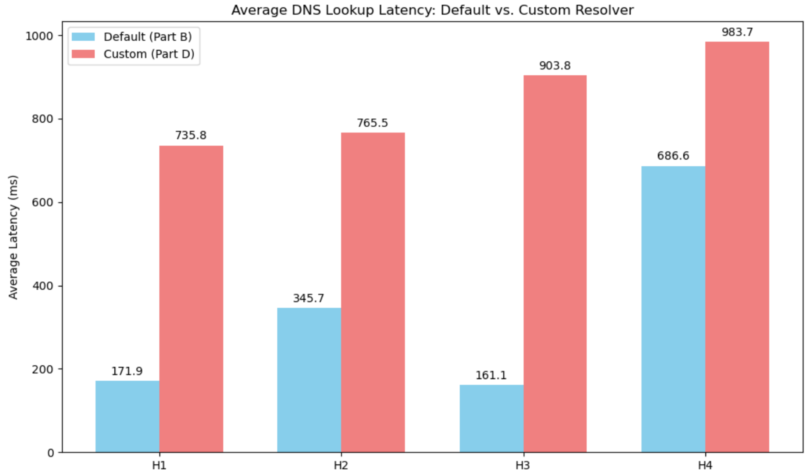


Figure 6: Comparison: Average DNS Lookup Latency (ms)

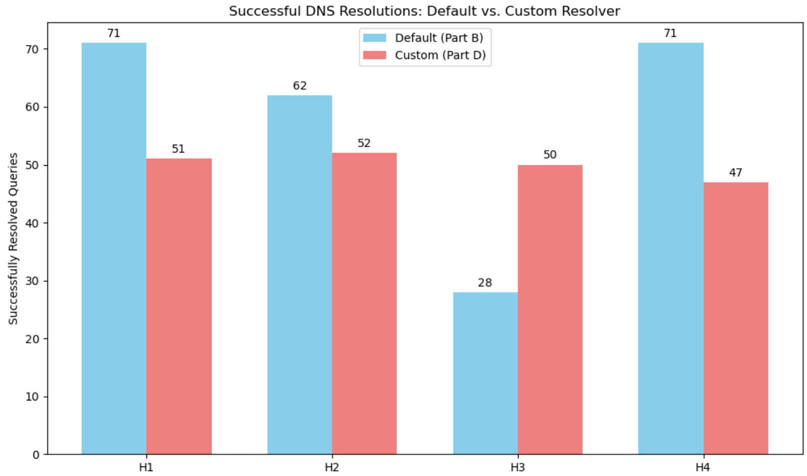


Figure 7: Comparison: Successfully Resolved Queries

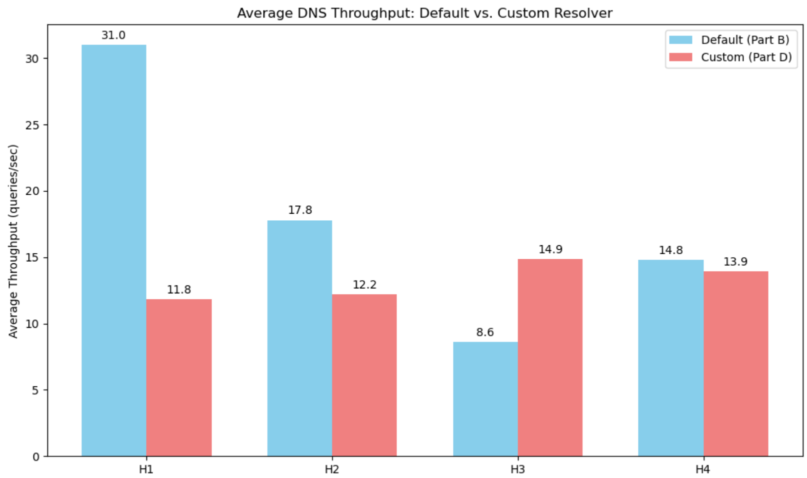


Figure 8: Comparison: Average DNS Throughput (queries/sec)

Analysis

- **Latency:** The custom iterative resolver consistently exhibited significantly higher average latency compared to the default resolver across all hosts (Figure 6). This is expected, as the iterative process requires the local resolver to contact multiple servers (Root, TLD, Authoritative) sequentially for each query, whereas the default resolver likely acts recursively or utilizes a more optimized upstream recursive server with caching.
- **Success Rate:** The custom iterative resolver generally had a lower success rate than the default resolver for H1, H2, and H4, but a higher success rate for H3 (Figure 7). The default resolver's higher success rate might be due to more robust error handling, retries, or access to better upstream resolvers. The custom resolver's failures often stemmed from timeouts or errors received during the iterative process. H3's default resolver performed exceptionally poorly, suggesting issues unrelated to the resolver type for that specific URL set, which the iterative resolver partly overcame.
- **Throughput:** Corresponding to the higher latency, the custom iterative resolver generally showed lower average throughput (queries per second) compared to the default resolver, except for H3 where the default resolver's high failure rate severely impacted its throughput (Figure 8).

Overall, the basic iterative resolver is functionally correct but less performant than a typical default system resolver due to the overhead of multiple sequential queries.

Task E: Bonus - Recursive Resolution

Approach Used

The custom DNS resolver was modified to support recursive resolution. PCAP files were assumed to be modified to indicate recursive mode preference. When receiving a query flagged for recursion, the resolver at `10.0.0.5` would perform the full resolution process (iteratively contacting Root, TLD, and Authoritative servers as needed) on behalf of the client, returning only the final answer or an error to the client. If recursion was not supported or flagged, it would fall back to the default (iterative) mode. The standard performance metrics were recorded.

Results

Table 3 summarizes the performance of the custom recursive resolver (without cache), based on `comparison_results.txt`. The log files (e.g., `part_e_PCAP_1_H1.pcap.log`) confirm the recursive mode, showing only the final answer or error, not the intermediate steps.

Table 3: Task E: Custom Recursive Resolver (No Cache) Performance Summary

Metric	H1	H2	H3	H4
Total Queries	100	100	100	100
Successfully Resolved	65	66	67	52
Failed Resolutions	35	34	33	48
Avg Lookup Latency (ms)	210.09	176.32	321.18	392.02
Avg Throughput (qps)	16.93	100.73*	16.77	7.96

**Note:* *The throughput for H2 (100.73 qps) appears anomalously high compared to its latency and other hosts' results, potentially indicating a measurement or calculation error in the source data.*

Analysis

Compared to the custom iterative resolver (Task D):

- **Latency:** Average latency was significantly lower with the recursive resolver. This is because the client only performs one round-trip to the local recursive resolver (`10.0.0.5`). The resolver handles the multiple iterative queries.

- **Success Rate:** The success rate was generally higher with the recursive resolver compared to the iterative one. The resolver might have better logic to handle referrals or retries internally.
- **Throughput:** Throughput was generally higher than the iterative resolver (ignoring the H2 anomaly), consistent with the lower latency from the client's perspective.

Compared to the default resolver (Task B):

- **Latency:** The custom recursive resolver's latency was sometimes lower (H2) and sometimes higher (H1, H3, H4) than the default resolver. This suggests the performance is comparable but depends on the specific queries and potential optimizations/caching in the default resolver's upstream servers.
- **Success Rate:** The success rates were closer to the default resolver than the iterative one, though still slightly lower for H1, H2, and H4, but much higher for H3.

Recursive resolution shifts the workload from the client to the resolver, generally improving client-perceived performance compared to pure iteration from the client.

Task F: Bonus - Caching

Approach Used

Caching was implemented within the custom DNS resolver. Recently resolved domain-to-IP mappings were stored locally at `10.0.0.5`. When a query arrived, the resolver checked its cache first. If a valid entry existed (Cache HIT), it was returned directly. If not (Cache MISS), the resolver performed the necessary lookup (either iteratively or recursively, depending on the mode being tested) and stored the result before returning it. Performance metrics, including the percentage of queries resolved from the cache, were recorded. Tests were run for both iterative and recursive modes with caching enabled.

Results

Tables 4 summarize the performance with caching enabled for iterative and recursive modes, respectively, based on `comparison_results.txt`. Cache hit/miss rates were not provided in the summary file but would be logged by the resolver. Log files like `part_f_iterative_PCAP_1_H1.pcap.log` would contain the detailed cache status per query step.

Table 4: Task F: Custom Iterative Resolver with Cache Performance Summary

Metric	H1	H2	H3	H4
Total Queries	100	100	100	100
Successfully Resolved	14	50	59	59
Failed Resolutions	86	50	41	41
Avg Lookup Latency (ms)	1624.09	924.43	1062.42	882.25
Avg Throughput (qps)	7.54	17.77	11.05	19.92

Analysis

- **Iterative with Cache:** Compared to iterative without cache (Task D), the success rate decreased dramatically for H1 but increased slightly for H3 and H4. Latency generally remained high or even increased. This counter-intuitive result (caching often reduces latency) suggests potential issues with the cache implementation (e.g., incorrect entries, slow cache lookups, or perhaps the specific URL patterns in the PCAPs didn't benefit much from caching in iterative mode). The very low success rate for H1 is particularly concerning.

In theory, caching should reduce average latency and potentially increase throughput, especially for repeated queries. The observed results for iterative caching showed mixed success.