CS 352 Project 2: Domain Name Systems

Due 23 October 2025, Time 11:59pm ET

Project Overview

This project introduces you to the fundamentals of the Domain Name System (DNS) by progressively building a DNS client in Python. You will work with low-level UDP socket programming and packet parsing to understand how DNS queries and responses are structured and how DNS resolves the IP address for a given host name.

We will provide you with **helper functions** (`build_query`, `parse_name`, `parse_response`) that handle the low-level details of packet construction and parsing. Students will focus on implementing the resolver logic in multiple parts.

You will also use **Wireshark** to capture DNS traffic while running your resolver. This will help you visualize how queries and responses are exchanged and answer questions based on DNS messages sent and received.

1 DNS Packet Structure

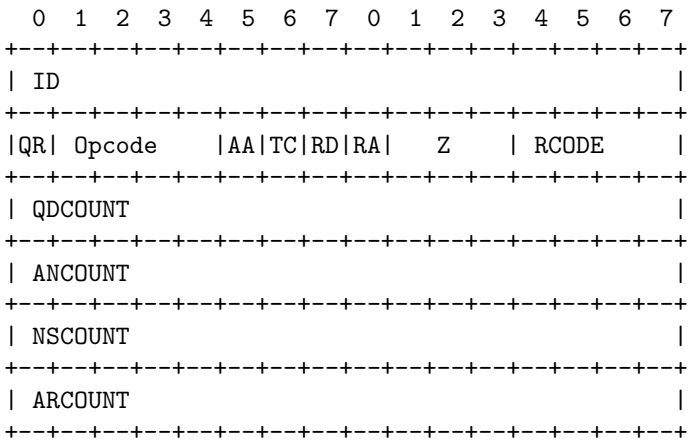
All DNS packets have a structure that is

```
+----------------------+
| Header               |
+----------------------+
| Question             | Question for the name server
+----------------------+
| Answer               | Answers to the question
+----------------------+
| Authority            | Name Servers for this query
+----------------------+
| Additional           | IP address of Name Servers if provided
+----------------------+
```

The header describes the type of packet and which fields are contained in the packet. Following the header are a number of questions, answers, authority records, and additional records. Note that a response for a single question may contain multiple answers, such as domain resolves to multiple IP addresses. Your client must process the entire answer section and add each one of these records.

2 DNS Packet Header

DNS packets have a header that is shown below. Note that requests and replies follow the same header format.



Where each of these fields is as described below:

ID A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied in the corresponding reply and can be used by the requester to match up replies to outstanding queries. You should always use 1337 for this field.

QR A one bit field that specifies whether this message is a query (0), or a response (1). Obviously, you should use 0 for your requests, and expect to see a 1 in the response you receive.

OPCODE A four bit field that specifies kind of query in this message. You should use 0, representing a standard query.

AA Authoritative Answer - this bit is only meaningful in responses, and specifies that the responding name server is an authority for the domain name in question section. You should use this bit to report whether or not the response you receive is authoritative.

TC TrunCation - specifies that this message was truncated. For this project, you must exit and return an error if you receive a response that is truncated.

RD Recursion Desired - this bit directs the name server to pursue the query recursively. You should use 1, representing that you desire recursion, for Part A and Part B. For Part C this should be set to 0.

RA Recursion Available - this bit is set or cleared in a response, and denotes whether recursive query support is available in the name server. Recursive query support is optional. You must exit and return an error if you receive a response that indicates the server does not support recursion for Part A and Part B.

Z Reserved for future use.

RCODE Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

    0 No error condition
    1 Format error - The name server was unable to interpret the query.
    2 Server failure - The name server was unable to process this query due to a problem with the name server.
    3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.
    4 Not Implemented - The name server does not support the requested kind of query.
    5 Refused - The name server refuses to perform the specified operation for policy reasons.

QDCOUNT an unsigned 16 bit integer specifying the number of entries in the question section.

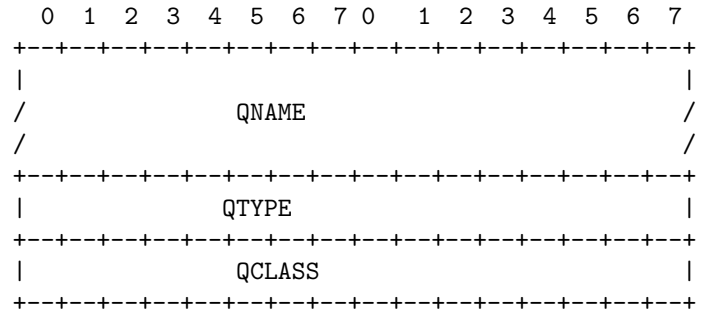
ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section.

3 DNS Questions

A DNS question has the format



Where each of these fields is as described below:

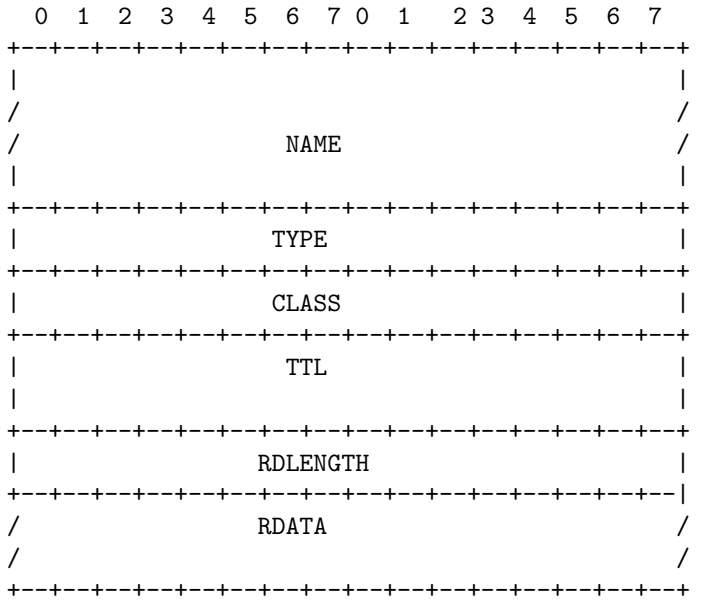
QNAME A domain name represented as a sequence of labels, where each label consists of a length byte followed by that number of bytes. The domain name terminates with the zero length octet for the null label of the root. See the DNS Example query below.

QTYPE A two byte code which specifies the type of the query. You should use 0x0001 (decimal 1) representing A records (host addresses), 0x001c (decimal 28) representing AAAA records (host addresses), and 0x0002 (decimal 2) for name servers (NS) records.

QCLASS A two byte code that specifies the class of the query. You should always use 0x0001 for this project, representing Internet addresses.

4 DNS Answers

A DNS answer has the format



Where each of these fields is as described below:

NAME The domain name that was queried, in the same format as the QNAME in the questions.

TYPE Two octets containing one of the type codes. This field specifies the meaning of the data in the RDATA field. Interpret type 0x0001 (A record) and type 0x001c (AAAA record), type 0x0002 (name servers).

CLASS Two bytes which specify the class of the data in the RDATA field. You should expect 0x0001 for this project, representing Internet addresses.

TTL The number of seconds the results can be cached.

RDLENGTH The length of the RDATA field.

RDATA The data of the response. The format is dependent on the TYPE field: if the TYPE is 0x0001 for A records, then this is the IPv4 address (4 bytes). If the type is 0x001c for AAAA, then this is the IPv6 address (16 bytes). If the type is 0x0002 for name servers, then this is the name of the server.

Part A: Recursive Resolver (A Records only)

- **Goal:** Implement a simple DNS client that queries a recursive resolver (e.g., Google DNS at `8.8.8.8`, port 53).
- **Task:**
  - Build a DNS query for an **A record** (IPv4 address) or **AAAA record** (IPv6 address).
  - Send the query to a public recursive resolver.
  - Parse the response to extract the IP address.
- **Wireshark Activity:**
  - Capture your query and response packets.
  - Identify the following fields in the DNS packet:
    - Transaction ID
    - Flags (standard query/response)
    - Question section (domain, type)
    - Answer section (A record)
- **Learning Outcome:** Understand the structure of a DNS query/response and how to interact with a DNS server using UDP.

Starter Code (Part A)

```
import socket
import json

# Provided helper functions: dns_query,build_query, parse_response
# Example query spec as JSON
dns_query_spec = {
    "id": random.randint(0, 65535),
    "qr": 0,       # query
    "opcode": 0,  # standard query
    "rd": 1,       # recursion desired
    "questions": [
        {
            "qname": "ilab1.cs.rutgers.edu",
            "qtype": 1,   # A record
            "qclass": 1   # IN
        }
    ]
}

your test file is a json array of records with two fields:
"questions": [
        {
            "qname": "ilab1.cs.rutgers.edu",
```

```
            "qtype": 28,
            }
        {
            "qname": "whale.stanford.edu",
            "qtype": 1,   # A record
            }
            ]

# Read questions from file input.txt
with open("input.txt", "r") as f:
  query_json = json.load(f)
# main loop
for q in query_json["questions"]:
     # TODO  create dns_query_spec  with all the fields and "qname" , "qtype" obtained from q
      response = dns_query(dns_query_spec)
      print(json.dumps(response, indent=2))
     # TODO append response to output.txt
```

Part B: Extend to NS Records

- **Goal:** Add support for resolving **NS (Name Server) records**.
- **Task:**
  - Extend the parser to handle NS records in the Authority and Additional sections.
  - Write logic to display the authoritative name servers for a given domain.
- **Wireshark Activity:**
  - Capture the response for an NS query.
  - Identify the Authority and Additional sections in the DNS packet.
  - Match NS hostnames with glue A records (if present).
- **Learning Outcome:** Learn that DNS responses can include multiple record types and understand delegation through NS records.

Starter Code (Part B)

```
import socket

# Provided helper functions: build_query, parse_response

def query_ns(domain):
    query = build_query(domain, qtype=2)  # 2 = NS record
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(query, ("8.8.8.8", 53))
    data, _ = sock.recvfrom(512)
    sock.close()
    response = parse_response(data)
    # TODO: Extract NS names from the response and return
    return response

if __name__ == "__main__":
    domain = "example.com"
    print(query_ns(domain))
```

Part C: Iterative Resolver

- **Goal:** Implement an **iterative resolver** from scratch.
- **Task:**
  - Start querying at the **root servers** (list of root IPs will be provided).
  - At each step:
    - Parse the referral response (authorities + glue records).
    - Select the **first available NS** and continue the query.
  - Stop once an answer is received or an authoritative server returns no further delegation.
- **Constraint:** Assume **glue records are always available**. If not, return an error.
- **Wireshark Activity:**
  - Capture your iterative resolution process.
  - Answer questions:
    - How many queries did your resolver send?
    - Which servers did it contact (list IPs)?
    - At which level did you finally receive the answer?
- **Learning Outcome:** Learn how DNS resolution works internally, mimicking the behavior of a recursive resolver.

Starter Code (Part C)

```python
import socket

# Provided helper functions: build_query, parse_response

ROOT_SERVERS = [
    "198.41.0.4",     # a.root-servers.net
    "199.9.14.201",   # b.root-servers.net
    "192.33.4.12",    # c.root-servers.net
]


def iterative_resolve(domain):
    server = ROOT_SERVERS[0]  # start with first root server

    while True:
        query = build_query(domain, qtype=1)  # request A record
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.sendto(query, (server, 53))
        data, _ = sock.recvfrom(512)
        sock.close()

        response = parse_response(data)
        # TODO: Check if response has answer A return it
        # TODO: Else, extract first NS glue record and update `server`
        # TODO: If no glue record found, raise error

    return None

if __name__ == "__main__":
    domain = "example.com"
    print(iterative_resolve(domain))
```

Instructions to capture traffic

1. Start the capture with tshark before running the dns program using the command

   tshark -i any -f "udp port 53" -w project2_partA.pcap

2. Capture the traffic with wireshark as follows

   1. Start Wireshark on rlab5
   2. Select interface 'any'
   3. Set capture filter: 'udp.port == 53'
   4. Start packet capture before running the dns program, stop the packet capture when the dns program finishes its execution
   5. Save as project2_partA.pcap

Deliverables

1. Python implementation for Parts A, B and C.
2. PCAP files for Parts A, B and C. Answers to wireshark questions in your report will be cross-checked with your packet capture. So don't forget to include PCAP files in submission.
3. A short report with answers to wireshark questions for each query in the test suite:

Part A -

1. What is the value of domain name found in the question section of the dns query? (Give the answer in the hexadecimal representation as observed in the wireshark)
2. What is the value of domain name found in the answer section of dns response? Explain (Give the answer in the hexadecimal representation as observed in the wireshark)
3. What is the value of the *rdlength* field in the dns response message? (Give the answer in the hexadecimal representation as observed in the wireshark)
4. What is the value of the *address* received in the dns response message? (Give the answer in the hexadecimal representation as observed in the wireshark)

Part B -

5. What is the value of QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT in the dns response message? (Give the answer in the hexadecimal representation as observed in the wireshark)
6. What is the value of the *rdlength* field in the dns response message? (If multiple resource records are present mention *rdlength* in each resource record. Give the answer in the hexadecimal representation as observed in the wireshark)
7. What are the *name server* names received in the response? (If multiple resource records are present mention *name server* present in each resource record. Give the answer in the hexadecimal representation as observed in the wireshark)

Part C -

8. What is the value of the *type* field in the first dns query among the iterative messages? (Give the answer in the hexadecimal representation as observed in the wireshark)
9. What is the value of the *type* field value in resource records in first dns response message among the iterative messages? (If multiple sections are present mention *type* field in resource records of each section along with the section name)

10. What is the destination IP of second dns query sent (Give the answer in the hexadecimal representation as observed in the wireshark)? Where is this ip found from?

4. Report should include the name and netid of all the members of the group.

Evaluation

- **Part A (25%)**: Working recursive resolver for A records + Wireshark answers.
- **Part B (25%)**: Support for NS records + Wireshark answers.
- **Part C (50%)**: Iterative resolver working under glue-available assumption + Wireshark analysis.
