CSE 351 Computer Networks

Fall 2024/2025

Group 11

Submitted to:

Dr. Ayman Bahaa

Eng. Noha Wahdan

Submitted by:

Mohamed Mostafa Mamdouh 21P0244

Gaser Zaghloul Hassan 21P0052

Ali Tarek 21P0123

Abdelrahman Sherif 21P0098

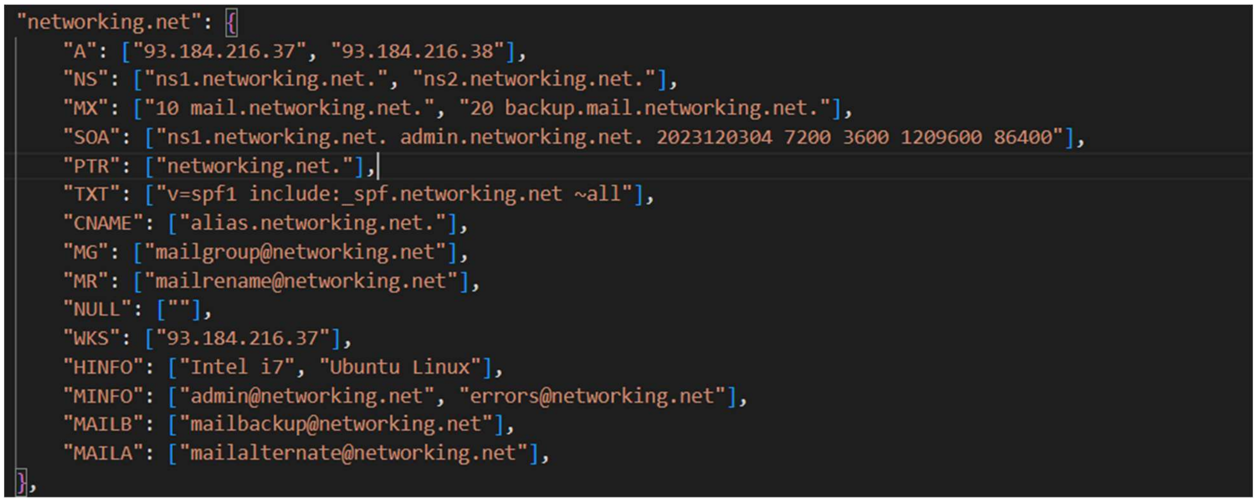
Contents

Introduction

The objective of this phase is to extend the server agent with advanced features as detailed in the relevant RFC. This phase demonstrates the server's capability to support required features, handle error codes, and interact with client systems under realistic scenarios. The implementation ensures compliance with the specified protocol, enabling robust and scalable communication. Note that networking.net was used to showcase the functionality of this phase, but other domains have also been tested.

Objective

- Extend the previously developed server agent to support protocol-specific features.
- Demonstrate error handling and protocol adherence through detailed test cases.
- Verify the functionality using different use cases and validate outputs.



```
"networking.net": {
    "A": ["93.184.216.37", "93.184.216.38"],
    "NS": ["ns1.networking.net.", "ns2.networking.net."],
    "MX": ["10 mail.networking.net.", "20 backup.mail.networking.net."],
    "SOA": ["ns1.networking.net. admin.networking.net. 2023120304 7200 3600 1209600 86400"],
    "PTR": ["networking.net."],
    "TXT": ["v=spf1 include:_spf.networking.net ~all"],
    "CNAME": ["alias.networking.net."],
    "MG": ["mailgroup@networking.net"],
    "MR": ["mailrename@networking.net"],
    "NULL": [""],
    "WKS": ["93.184.216.37"],
    "HINFO": ["Intel i7", "Ubuntu Linux"],
    "MINFO": ["admin@networking.net", "errors@networking.net"],
    "MAILB": ["mailbackup@networking.net"],
    "MAILA": ["mailalternate@networking.net"],
},
```

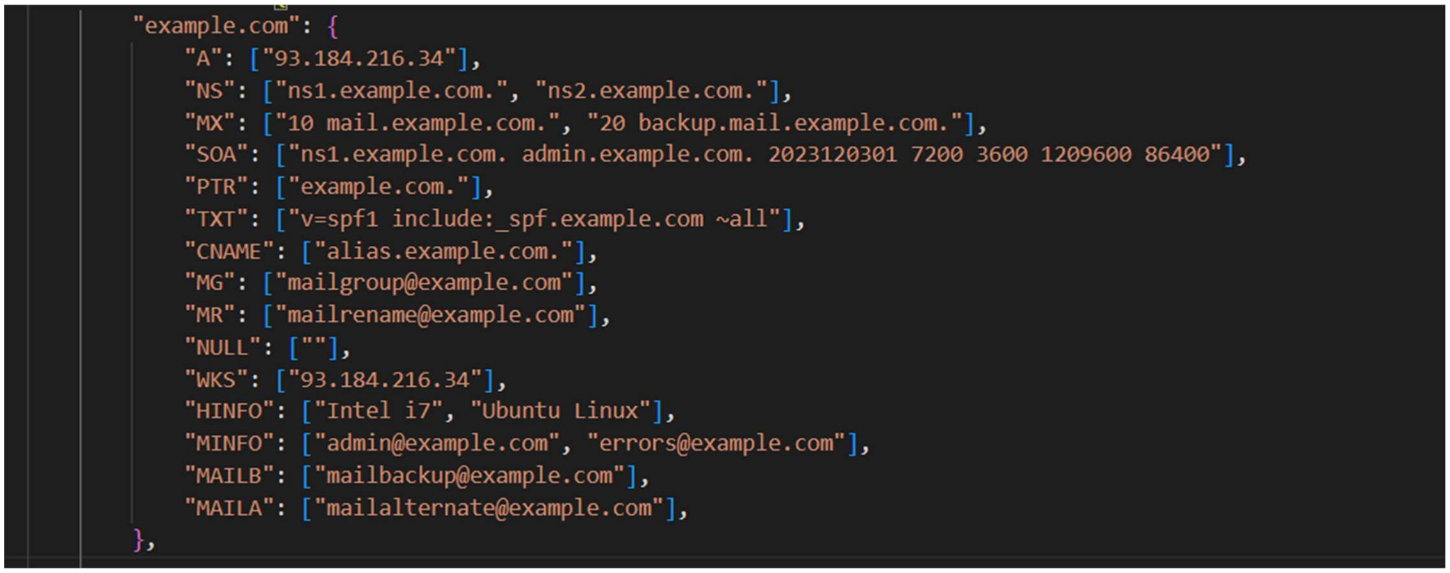
The provided section outlines the DNS zone configuration for the domain **networking.net**, showcasing various record types essential for domain management and functionality. Here's a summary of the key entries:

- **A (Address)**: Links the domain to its IP addresses (93.184.216.37, 93.184.216.38).
- **NS (Name Server)**: Lists the authoritative servers responsible for DNS queries (ns1.networking.net., ns2.networking.net.).
- **MX (Mail Exchange)**: Defines mail servers for email routing with priorities (10 mail.networking.net, 20 backup.mail.networking.net).
- **SOA (Start of Authority)**: Contains metadata about the zone, including the primary nameserver, admin email, serial number, and update intervals.
- **PTR (Pointer)**: Maps an IP address back to the domain name for reverse DNS lookups.
- **TXT (Text)**: Provides additional information, such as SPF (Sender Policy Framework) records for email validation.
- **CNAME (Canonical Name)**: Creates an alias for alias.networking.net..

- **Other Records**: Includes entries like HINFO (host information), WKS (well-known services), MINFO (mailbox info), and more, offering a complete setup for domain services.

This detailed configuration highlights the robust and efficient setup of DNS records, ensuring reliable domain operation and service management.

Other resource records tested include:



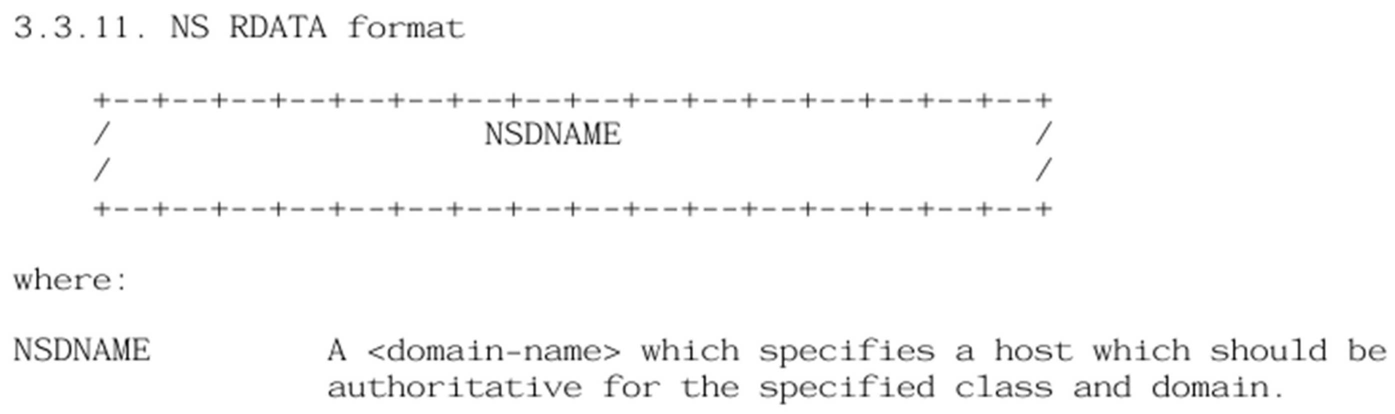
```
"example.com": {
    "A": ["93.184.216.34"],
    "NS": ["ns1.example.com.", "ns2.example.com."],
    "MX": ["10 mail.example.com.", "20 backup.mail.example.com."],
    "SOA": ["ns1.example.com. admin.example.com. 2023120301 7200 3600 1209600 86400"],
    "PTR": ["example.com."],
    "TXT": ["v=spf1 include:_spf.example.com ~all"],
    "CNAME": ["alias.example.com."],
    "MG": ["mailgroup@example.com"],
    "MR": ["mailrename@example.com"],
    "NULL": [""],
    "WKS": ["93.184.216.34"],
    "HINFO": ["Intel i7", "Ubuntu Linux"],
    "MINFO": ["admin@example.com", "errors@example.com"],
    "MAILB": ["mailbackup@example.com"],
    "MAILA": ["mailalternate@example.com"],
},
```

Functionality added as part of phase 3

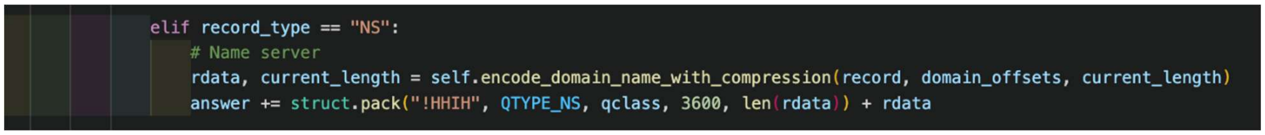
Query Types in action

NS (Name Server)

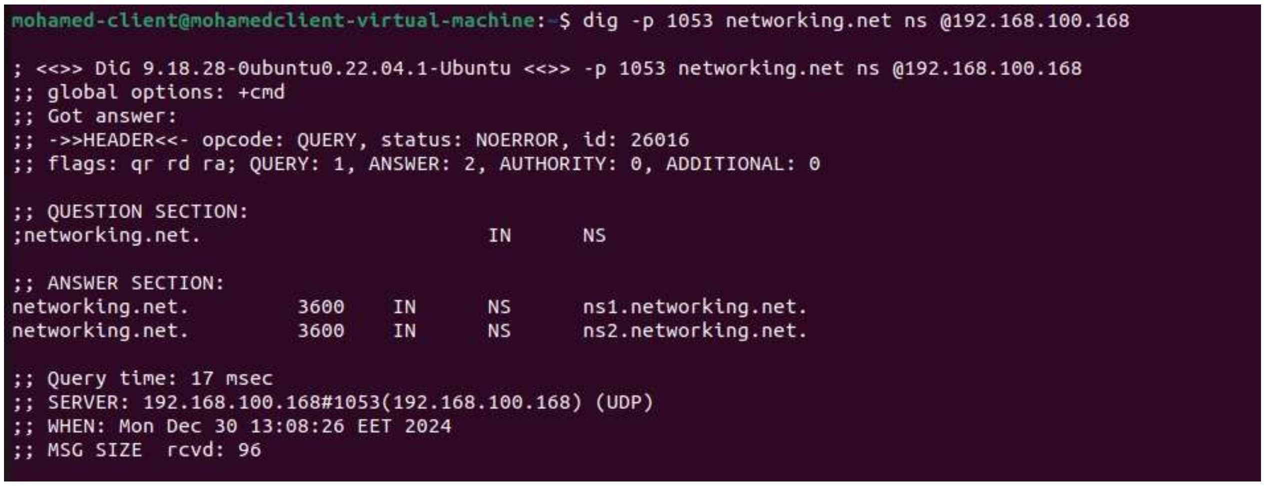
The NS records shown here define the authoritative name servers for networking.net (ns1.networking.net. and ns2.networking.net.). These servers handle DNS queries and ensure proper resolution of the domain.
In the response, each NS record is processed by encoding the name server domain using compression techniques, then adding it to the answer section with its type (QTYPE_NS), class, and TTL.



```
3.3.11. NS RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   NSDNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

NSDNAME         A <domain-name> which specifies a host which should be
                authoritative for the specified class and domain.
```



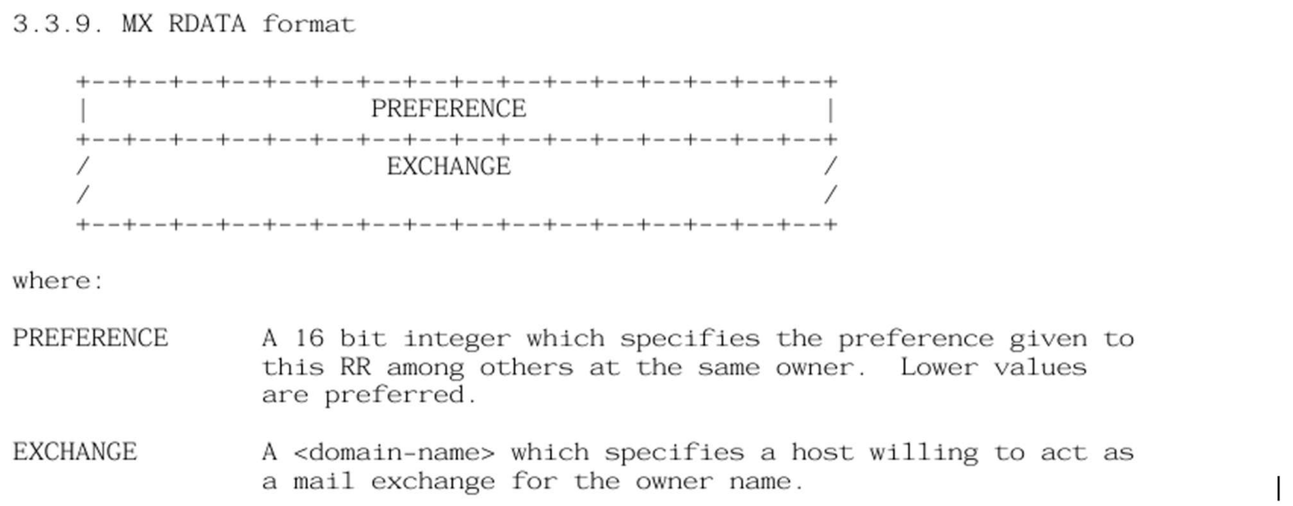
```
elif record_type == "NS":
    # Name server
    rdata, current_length = self.encode_domain_name_with_compression(record, domain_offsets, current_length)
    answer += struct.pack("!HHIH", QTYPE_NS, qclass, 3600, len(rdata)) + rdata
```



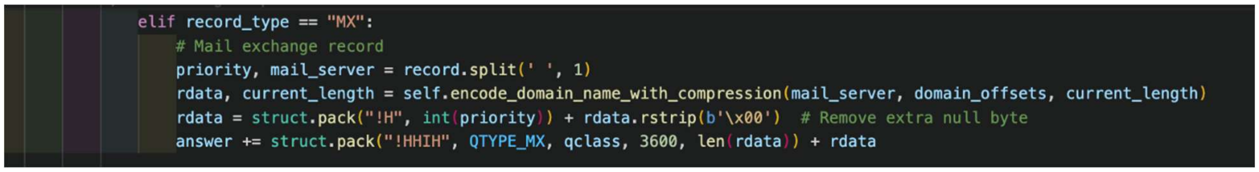
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net ns @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net ns @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26016
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                        IN      NS

;; ANSWER SECTION:
networking.net.         3600    IN      NS      ns1.networking.net.
networking.net.         3600    IN      NS      ns2.networking.net.

;; Query time: 17 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:08:26 EET 2024
;; MSG SIZE  rcvd: 96
```

MX (Mail Exchange) Record

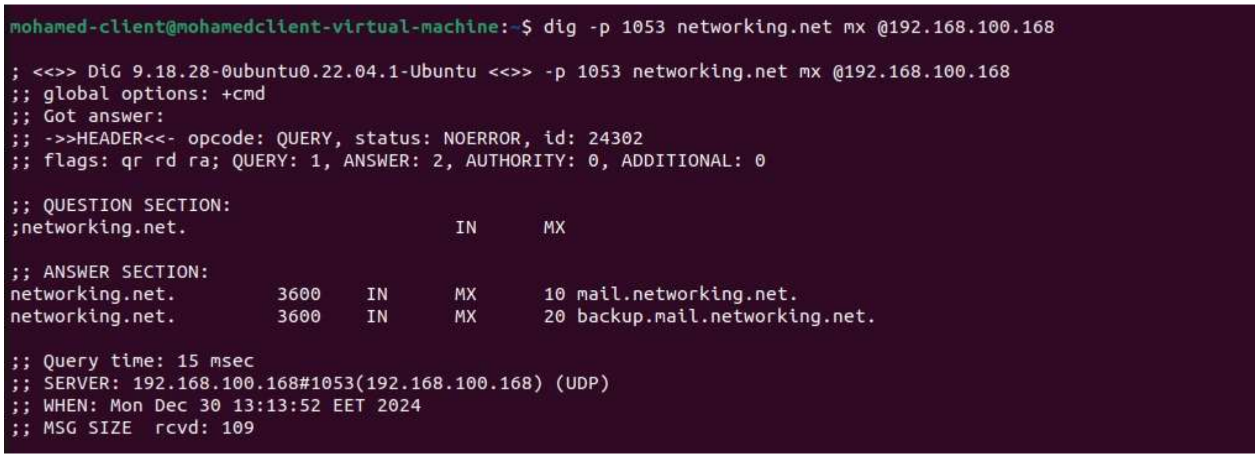
The MX records for networking.net, specifying the mail servers responsible for handling email delivery. The primary server has a priority of 10 (mail.networking.net), while the backup server has a priority of 20 (backup.mail.networking.net). To build the response, the server encodes the mail server domain and includes the priority as part of the record data (rdata), adding it to the DNS answer with type (QTYPE_MX), class, and TTL.



```
3.3.9. MX RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                  PREFERENCE                   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   EXCHANGE                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

PREFERENCE      A 16 bit integer which specifies the preference given to
                this RR among others at the same owner.  Lower values
                are preferred.

EXCHANGE        A <domain-name> which specifies a host willing to act as
                a mail exchange for the owner name.
```



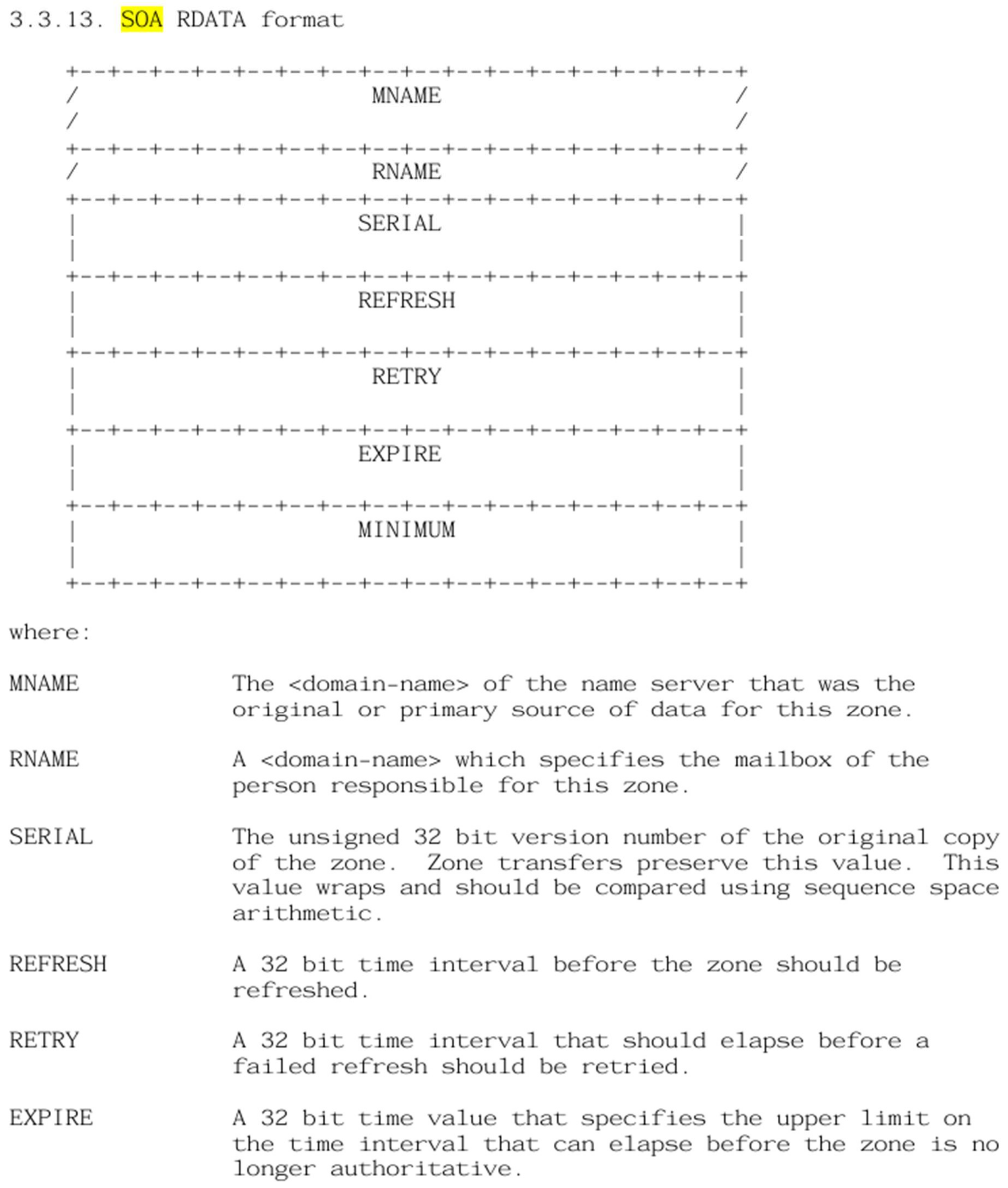
```
        elif record_type == "MX":
            # Mail exchange record
            priority, mail_server = record.split(' ', 1)
            rdata, current_length = self.encode_domain_name_with_compression(mail_server, domain_offsets, current_length)
            rdata = struct.pack("!H", int(priority)) + rdata.rstrip(b'\x00')  # Remove extra null byte
            answer += struct.pack("!HHIH", QTYPE_MX, qclass, 3600, len(rdata)) + rdata
```



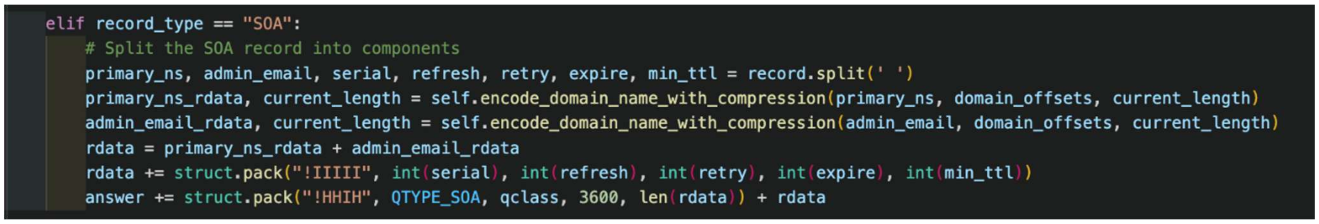
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net mx @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net mx @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24302
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                        IN      MX

;; ANSWER SECTION:
networking.net.         3600    IN      MX      10 mail.networking.net.
networking.net.         3600    IN      MX      20 backup.mail.networking.net.

;; Query time: 15 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:13:52 EET 2024
;; MSG SIZE  rcvd: 109
```

SOA (Start of Authority)

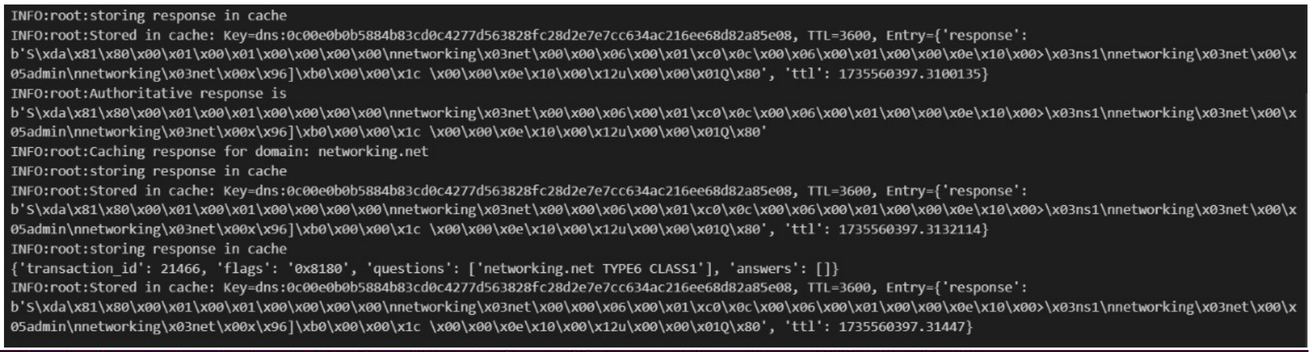
The SOA record includes administrative information for networking.net, such as the primary nameserver (ns1.networking.net), administrator email (admin.networking.net), and key timing values like refresh and expiry intervals. When constructing the response, the SOA record is split into components, such as primary nameserver and admin email, which are compressed and appended along with serialized integers for the timing parameters. This is added to the answer with type (QTYPE_SOA), class, and TTL.



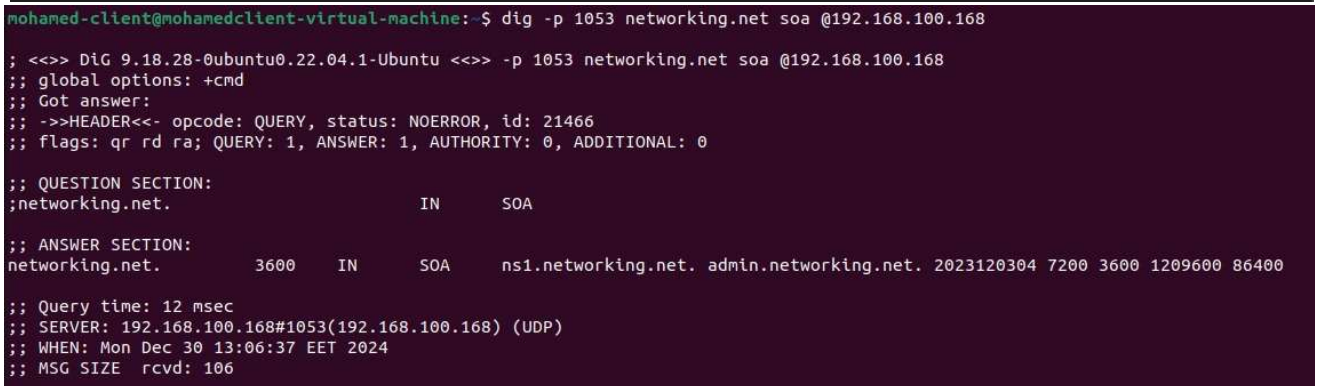
```
3.3.13. SOA RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                     MNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                     RNAME                     /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    SERIAL                     |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    REFRESH                    |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                     RETRY                     |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    EXPIRE                     |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    MINIMUM                    |
    |                                               |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

MNAME           The <domain-name> of the name server that was the
                original or primary source of data for this zone.

RNAME           A <domain-name> which specifies the mailbox of the
                person responsible for this zone.

SERIAL          The unsigned 32 bit version number of the original copy
                of the zone.  Zone transfers preserve this value.  This
                value wraps and should be compared using sequence space
                arithmetic.

REFRESH         A 32 bit time interval before the zone should be
                refreshed.

RETRY           A 32 bit time interval that should elapse before a
                failed refresh should be retried.

EXPIRE          A 32 bit time value that specifies the upper limit on
                the time interval that can elapse before the zone is no
                longer authoritative.
```



```python
elif record_type == "SOA":
    # Split the SOA record into components
    primary_ns, admin_email, serial, refresh, retry, expire, min_ttl = record.split(' ')
    primary_ns_rdata, current_length = self.encode_domain_name_with_compression(primary_ns, domain_offsets, current_length)
    admin_email_rdata, current_length = self.encode_domain_name_with_compression(admin_email, domain_offsets, current_length)
    rdata = primary_ns_rdata + admin_email_rdata
    rdata += struct.pack("!IIIII", int(serial), int(refresh), int(retry), int(expire), int(min_ttl))
    answer += struct.pack("!HHIH", QTYPE_SOA, qclass, 3600, len(rdata)) + rdata
```



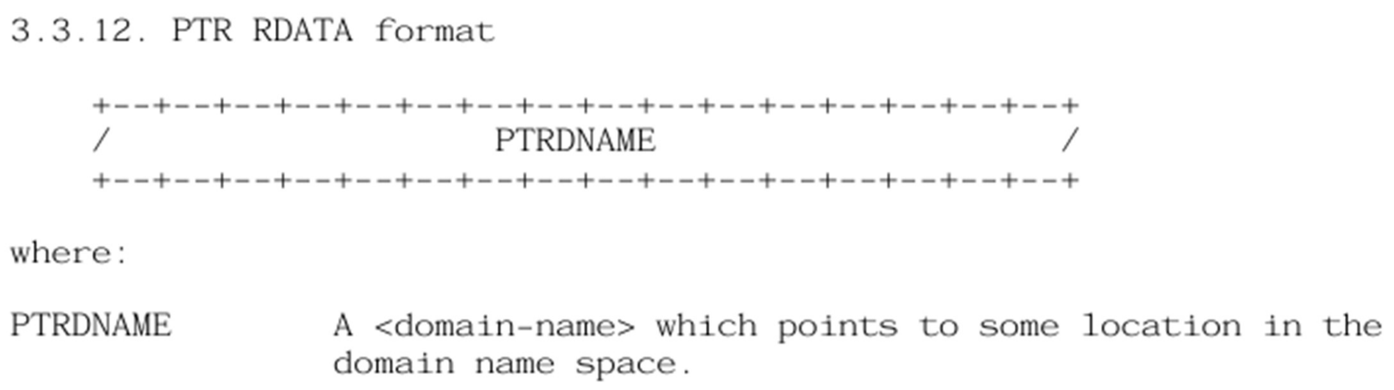
```
INFO:root:storing response in cache
INFO:root:Stored in cache: Key=dns:0c00e0b0b5884b83cd0c4277d563828fc28d2e7e7cc634ac216ee68d82a85e08, TTL=3600, Entry={'response':
b'S\xda\x81\x80\x00\x01\x00\x01\x00\x00\x00\x00\nnetworking\x03net\x00\x00\x06\x00\x01\xc0\x0c\x00\x06\x00\x01\x00\x00\x0e\x10\x00>\x03ns1\nnetworking\x03net\x00\x
05admin\nnetworking\x03net\x00x\x96]\xb0\x00\x00\x1c \x00\x00\x0e\x10\x00\x12u\x00\x00\x01Q\x80', 'ttl': 1735560397.3100135}
INFO:root:Authoritative response is
b'S\xda\x81\x80\x00\x01\x00\x01\x00\x00\x00\x00\nnetworking\x03net\x00\x00\x06\x00\x01\xc0\x0c\x00\x06\x00\x01\x00\x00\x0e\x10\x00>\x03ns1\nnetworking\x03net\x00\x
05admin\nnetworking\x03net\x00x\x96]\xb0\x00\x00\x1c \x00\x00\x0e\x10\x00\x12u\x00\x00\x01Q\x80'
INFO:root:Caching response for domain: networking.net
INFO:root:storing response in cache
INFO:root:Stored in cache: Key=dns:0c00e0b0b5884b83cd0c4277d563828fc28d2e7e7cc634ac216ee68d82a85e08, TTL=3600, Entry={'response':
b'S\xda\x81\x80\x00\x01\x00\x01\x00\x00\x00\x00\nnetworking\x03net\x00\x00\x06\x00\x01\xc0\x0c\x00\x06\x00\x01\x00\x00\x0e\x10\x00>\x03ns1\nnetworking\x03net\x00\x
05admin\nnetworking\x03net\x00x\x96]\xb0\x00\x00\x1c \x00\x00\x0e\x10\x00\x12u\x00\x00\x01Q\x80', 'ttl': 1735560397.3132114}
INFO:root:storing response in cache
{'transaction_id': 21466, 'flags': '0x8180', 'questions': ['networking.net TYPE6 CLASS1'], 'answers': []}
INFO:root:Stored in cache: Key=dns:0c00e0b0b5884b83cd0c4277d563828fc28d2e7e7cc634ac216ee68d82a85e08, TTL=3600, Entry={'response':
b'S\xda\x81\x80\x00\x01\x00\x01\x00\x00\x00\x00\nnetworking\x03net\x00\x00\x06\x00\x01\xc0\x0c\x00\x06\x00\x01\x00\x00\x0e\x10\x00>\x03ns1\nnetworking\x03net\x00\x
05admin\nnetworking\x03net\x00x\x96]\xb0\x00\x00\x1c \x00\x00\x0e\x10\x00\x12u\x00\x00\x01Q\x80', 'ttl': 1735560397.31447}
```



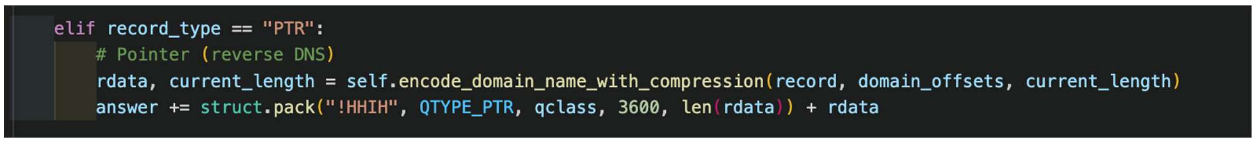
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net soa @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net soa @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21466
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                IN      SOA

;; ANSWER SECTION:
networking.net.         3600    IN      SOA     ns1.networking.net. admin.networking.net. 2023120304 7200 3600 1209600 86400

;; Query time: 12 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:06:37 EET 2024
;; MSG SIZE  rcvd: 106
```

PTR (Pointer)

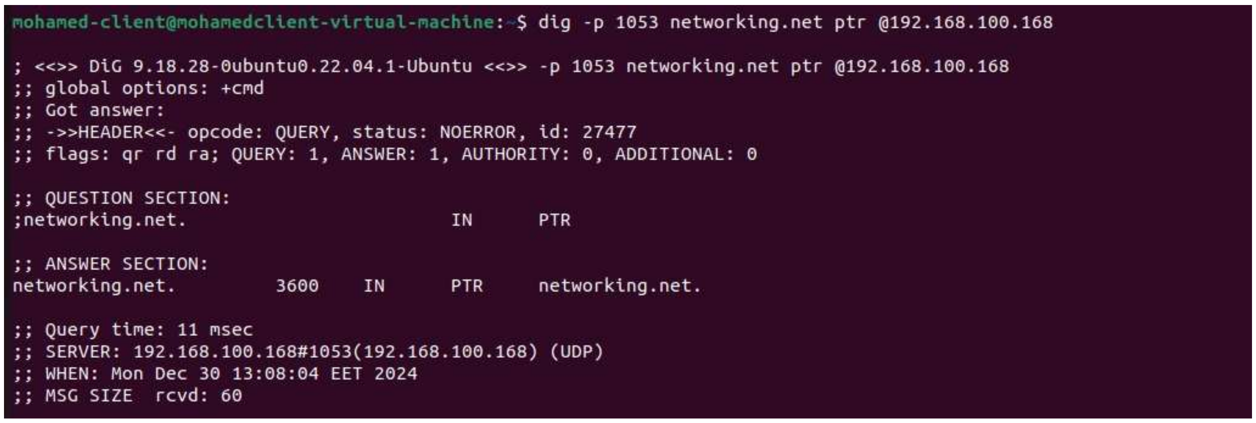
The PTR record links the IP address back to the domain name networking.net for reverse DNS lookups. This is crucial for validating the domain and ensuring accurate IP-to-hostname mapping. In the response, the pointer domain is encoded using compression and added to the answer section with its type (QTYPE_PTR), class, and TTL.



```
3.3.12. PTR RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   PTRDNAME                    /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

PTRDNAME        A <domain-name> which points to some location in the
                domain name space.
```



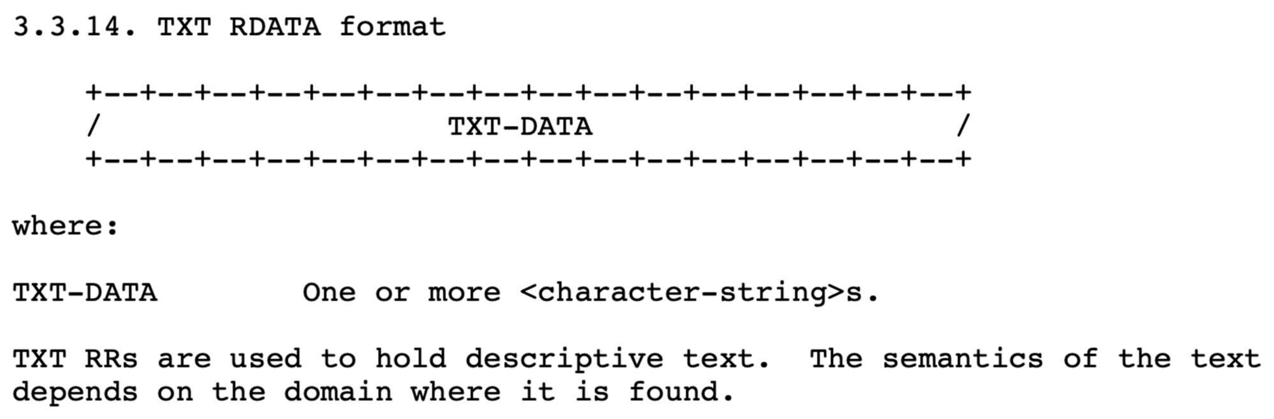
```
elif record_type == "PTR":
    # Pointer (reverse DNS)
    rdata, current_length = self.encode_domain_name_with_compression(record, domain_offsets, current_length)
    answer += struct.pack("!HHIH", QTYPE_PTR, qclass, 3600, len(rdata)) + rdata
```



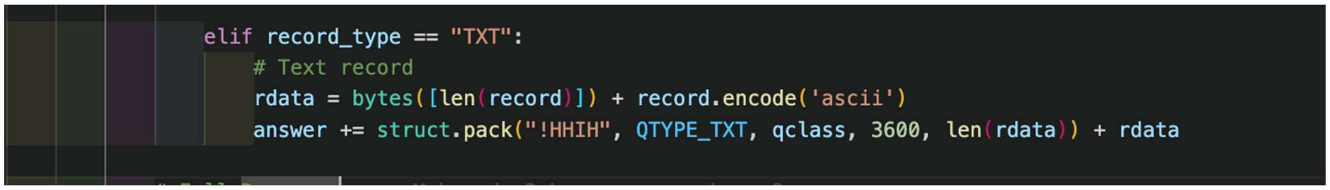
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net ptr @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net ptr @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27477
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                        IN      PTR

;; ANSWER SECTION:
networking.net.         3600    IN      PTR     networking.net.

;; Query time: 11 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:08:04 EET 2024
;; MSG SIZE  rcvd: 60
```

TXT (Text)

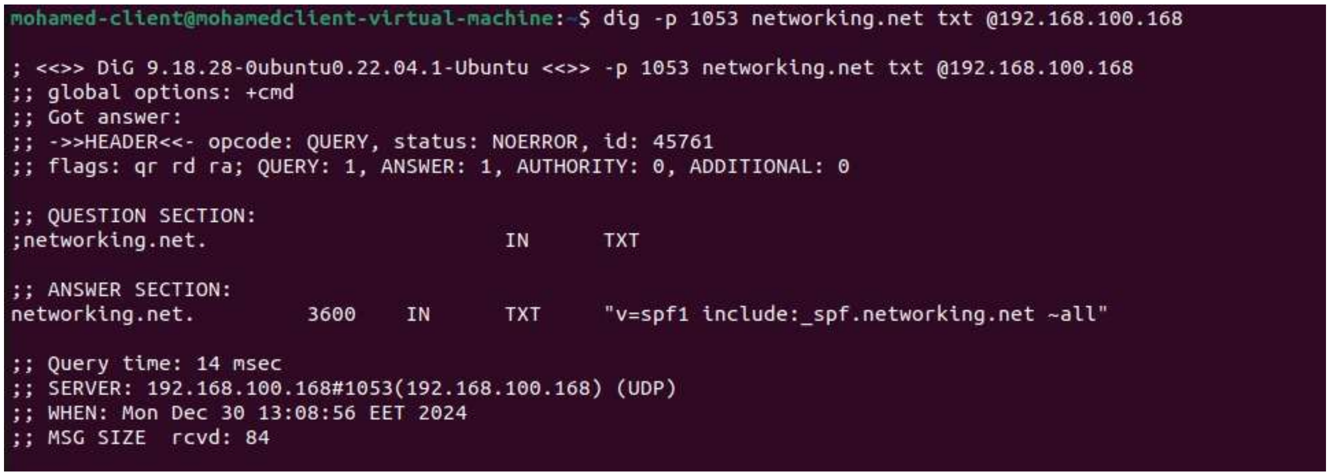
The TXT record stores additional information for the domain, including the SPF (Sender Policy Framework) entry (v=spf1 include:_spf.networking.net ~all) to prevent unauthorized email spoofing. To construct the response, the text data is encoded with a length prefix and added to the answer section with its type (QTYPE_TXT), class, and TTL.



```
3.3.14. TXT RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   TXT-DATA                     /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

TXT-DATA        One or more <character-string>s.

TXT RRs are used to hold descriptive text.  The semantics of the text
depends on the domain where it is found.
```



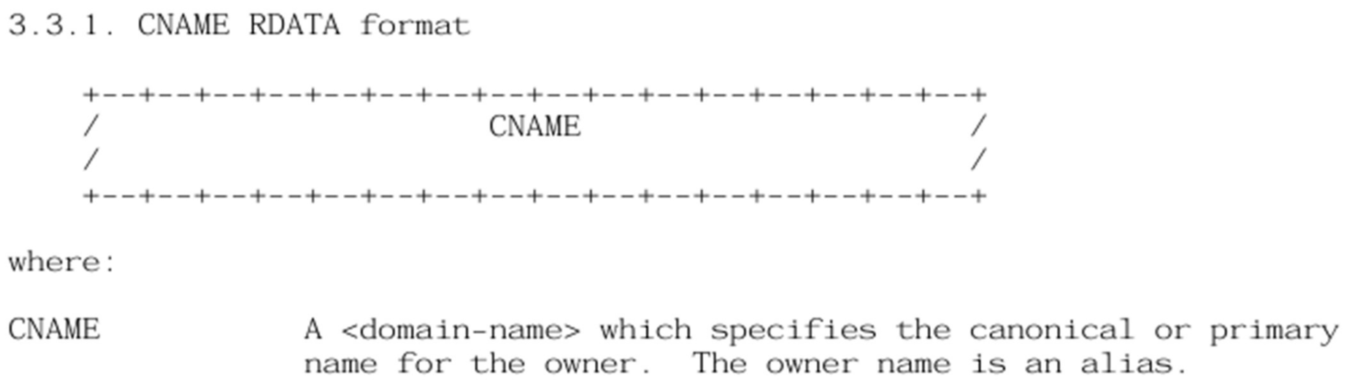
```
elif record_type == "TXT":
    # Text record
    rdata = bytes([len(record)]) + record.encode('ascii')
    answer += struct.pack("!HHIH", QTYPE_TXT, qclass, 3600, len(rdata)) + rdata
```



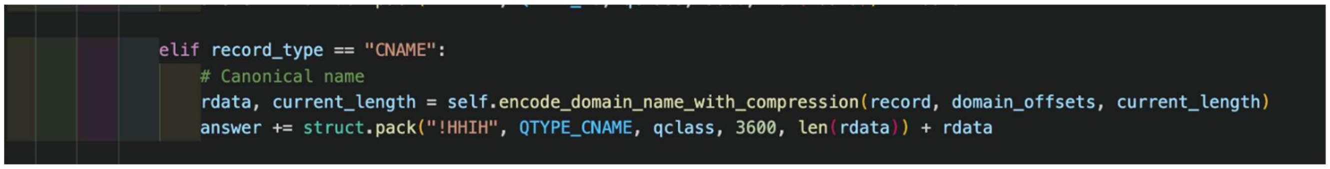
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net txt @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net txt @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45761
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                        IN      TXT

;; ANSWER SECTION:
networking.net.         3600    IN      TXT     "v=spf1 include:_spf.networking.net ~all"

;; Query time: 14 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:08:56 EET 2024
;; MSG SIZE  rcvd: 84
```

CNAME (Canonical Name)

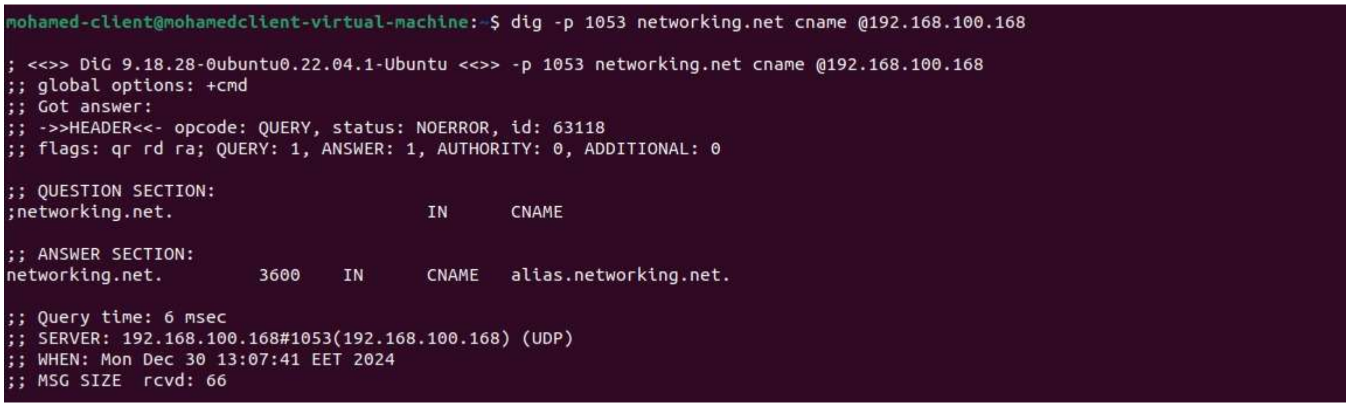
This section displays the CNAME record, which creates an alias for alias.networking.net., allowing the domain to redirect traffic to another domain or subdomain. The response encodes the alias using compression and includes it in the answer section with type (QTYPE_CNAME), class, and TTL.



```
3.3.1. CNAME RDATA format

    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                     CNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

where:

CNAME           A <domain-name> which specifies the canonical or primary
                name for the owner.  The owner name is an alias.
```



```
elif record_type == "CNAME":
    # Canonical name
    rdata, current_length = self.encode_domain_name_with_compression(record, domain_offsets, current_length)
    answer += struct.pack("!HHIH", QTYPE_CNAME, qclass, 3600, len(rdata)) + rdata
```



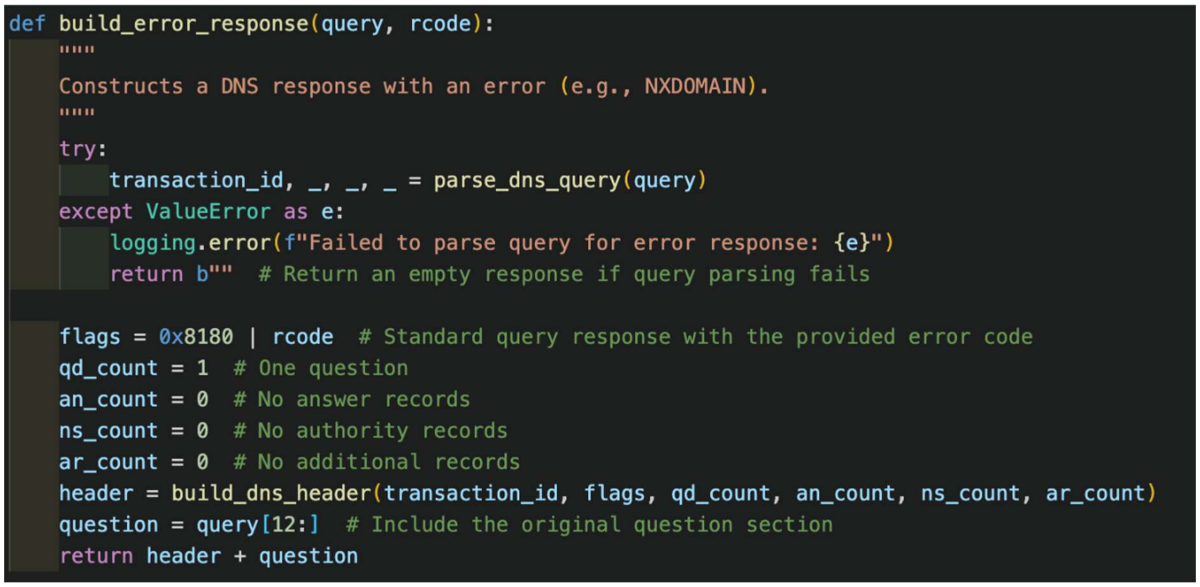
```
mohamed-client@mohamedclient-virtual-machine:~$ dig -p 1053 networking.net cname @192.168.100.168

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> -p 1053 networking.net cname @192.168.100.168
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63118
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;networking.net.                        IN      CNAME

;; ANSWER SECTION:
networking.net.         3600    IN      CNAME   alias.networking.net.

;; Query time: 6 msec
;; SERVER: 192.168.100.168#1053(192.168.100.168) (UDP)
;; WHEN: Mon Dec 30 13:07:41 EET 2024
;; MSG SIZE  rcvd: 66
```

DNS Error Response Construction

This part of the system handles the creation of DNS error responses, ensuring compliance with the protocol while providing meaningful feedback to clients. It generates a response message containing an appropriate error code, such as NXDOMAIN, while retaining critical details from the original query.

Key Details:

- **Purpose**:
  To construct a standardized DNS error response using the original query's transaction ID and specified error code (rcode).
- **Process**:
  1. The transaction ID is extracted from the incoming query. If the query cannot be parsed, the system logs an error and returns an empty response to prevent failure propagation.
  2. The DNS flags are configured to indicate an error response, combining standard query response flags (0x8180) with the provided error code.
  3. The DNS header is constructed, including:
     - One question (qd_count = 1).
     - No answers, authorities, or additional records (an_count, ns_count, ar_count = 0).
  4. The question section of the query is preserved and included in the response to ensure consistency with the client's original request.



```python
def build_error_response(query, rcode):
    """
    Constructs a DNS response with an error (e.g., NXDOMAIN).
    """
    try:
        transaction_id, _, _, _ = parse_dns_query(query)
    except ValueError as e:
        logging.error(f"Failed to parse query for error response: {e}")
        return b""  # Return an empty response if query parsing fails

    flags = 0x8180 | rcode  # Standard query response with the provided error code
    qd_count = 1  # One question
    an_count = 0  # No answer records
    ns_count = 0  # No authority records
    ar_count = 0  # No additional records
    header = build_dns_header(transaction_id, flags, qd_count, an_count, ns_count, ar_count)
    question = query[12:]  # Include the original question section
    return header + question
```

Error Handling

If the query is invalid or cannot be processed, the system logs the issue and safely returns an empty response. This ensures the system remains robust and operational in the face of malformed or unexpected inputs.

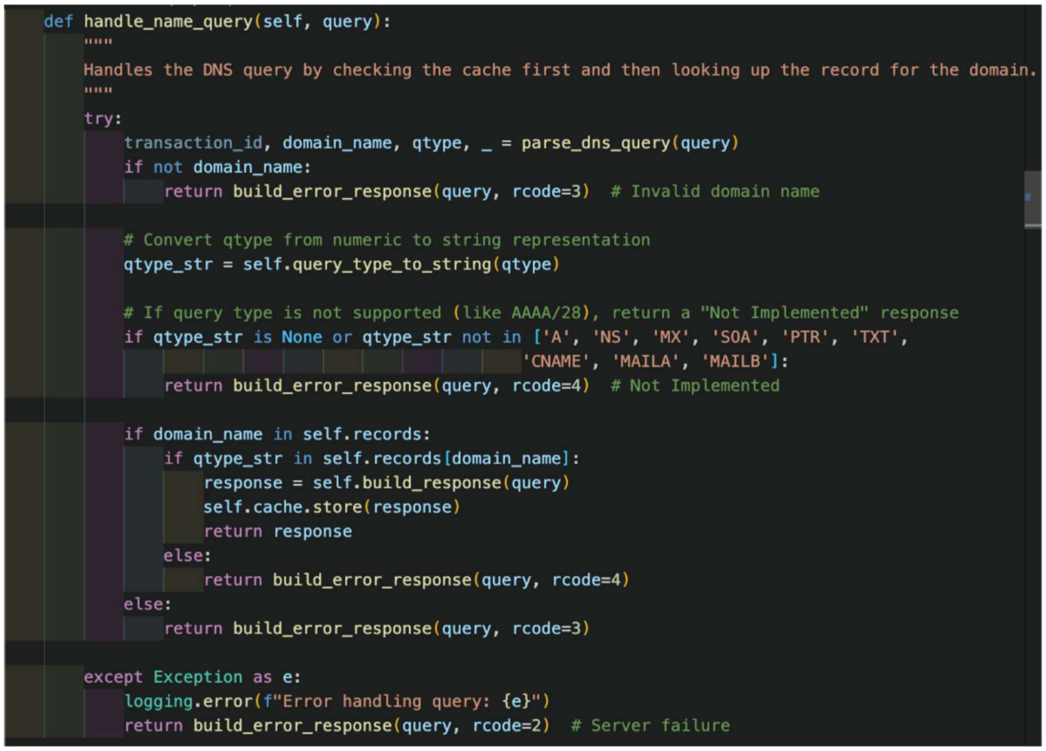
This mechanism ensures that the DNS server provides clear, RFC-compliant error responses, maintaining reliable communication even in error scenarios.

DNS Query Handling with Error Responses

This section describes the process for handling DNS queries within the authoritative server, focusing on the integration of error response construction to ensure protocol compliance and robust error handling.

Query Handling Logic:

- **Initial Parsing**:
  The incoming DNS query is parsed to extract the transaction ID, domain name, query type (qtype), and other details. If the domain name is invalid or missing, an error response with the NXDOMAIN code (rcode=3) is generated.
- **Query Type Validation**:
  The query type (qtype) is converted from numeric to its string representation (e.g., A, NS, MX). If the type is unsupported or unimplemented (e.g., AAAA or an unknown type), an error response with Not Implemented (rcode=4) is returned.
- **Record Lookup**:
  The server checks whether the requested domain name exists in its authoritative records:
  - If the domain is found and the query type is valid, the server constructs a response and stores it in the cache before returning it.
  - If the domain exists but lacks the requested query type, an error response with Not Implemented is generated.
  - If the domain does not exist, the server returns an NXDOMAIN error response.
  - **Error Handling**:
    Any unexpected errors during query processing are logged, and a server failure response (rcode=2) is sent to the client.



```python
def handle_name_query(self, query):
    """
    Handles the DNS query by checking the cache first and then looking up the record for the domain.
    """
    try:
        transaction_id, domain_name, qtype, _ = parse_dns_query(query)
        if not domain_name:
            return build_error_response(query, rcode=3)  # Invalid domain name

        # Convert qtype from numeric to string representation
        qtype_str = self.query_type_to_string(qtype)

        # If query type is not supported (like AAAA/28), return a "Not Implemented" response
        if qtype_str is None or qtype_str not in ['A', 'NS', 'MX', 'SOA', 'PTR', 'TXT',
                                                  'CNAME', 'MAILA', 'MAILB']:
            return build_error_response(query, rcode=4)  # Not Implemented

        if domain_name in self.records:
            if qtype_str in self.records[domain_name]:
                response = self.build_response(query)
                self.cache.store(response)
                return response
            else:
                return build_error_response(query, rcode=4)
        else:
            return build_error_response(query, rcode=3)

    except Exception as e:
        logging.error(f"Error handling query: {e}")
        return build_error_response(query, rcode=2)  # Server failure
```

Integration with build_error_response:

The build_error_response method is invoked to handle errors effectively:

- Invalid domain name or unsupported query types.
- Missing records for valid domains.
- General server errors or exceptions.

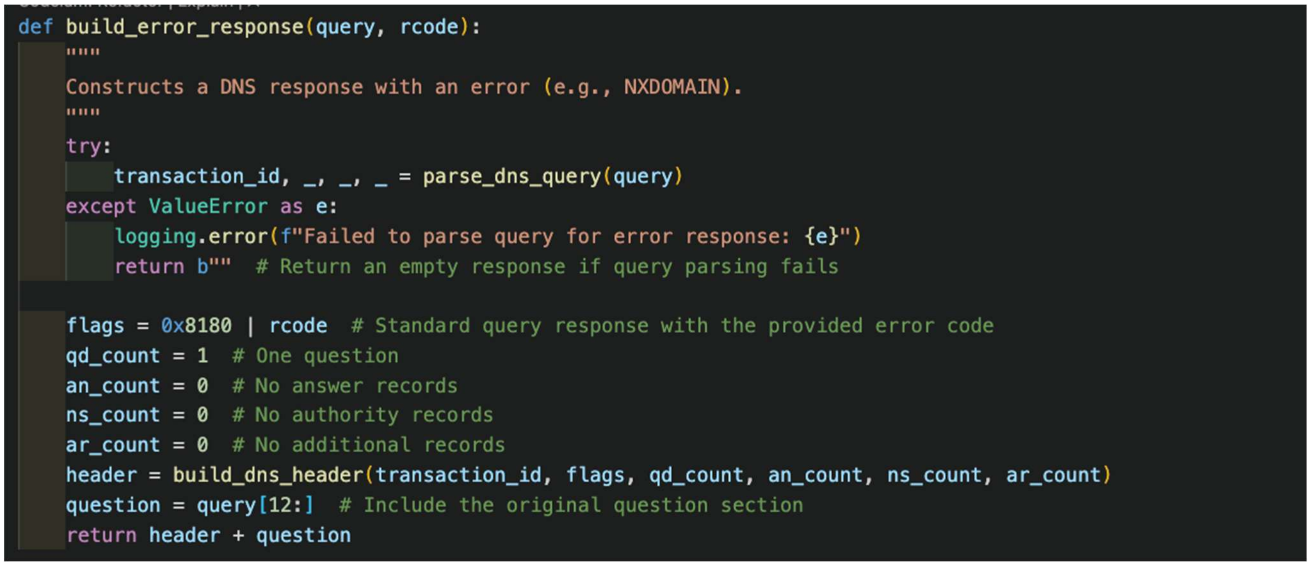
This mechanism ensures that the server remains robust, providing clear and meaningful responses for all scenarios while adhering to DNS protocol standards.

Error Handling in DNS Queries

Handling errors effectively is crucial for a DNS server to ensure clients receive meaningful feedback when issues occur. The build_error_response function creates appropriate error responses based on the provided RCODE values. These RCODEs indicate the type of error encountered :

- **Format Error (RCODE 1):** The query was not correctly formatted.
- **Server Failure (RCODE 2):** The server encountered an issue while processing the query.
- **Name Error (RCODE 3):** The queried domain name does not exist (NXDOMAIN).
- **Not Implemented (RCODE 4):** The requested query type is not supported.
- **Refused (RCODE 5):** The requested query was refused.

  The build_error_response function is responsible for constructing a DNS error response when the server encounters an issue. It accepts the original query and an RCODE to indicate the type of error. The response contains the same transaction ID as the query and includes the error code in the flags section of the DNS header.



```python
def build_error_response(query, rcode):
    """
    Constructs a DNS response with an error (e.g., NXDOMAIN).
    """
    try:
        transaction_id, _, _, _ = parse_dns_query(query)
    except ValueError as e:
        logging.error(f"Failed to parse query for error response: {e}")
        return b""  # Return an empty response if query parsing fails

    flags = 0x8180 | rcode  # Standard query response with the provided error code
    qd_count = 1  # One question
    an_count = 0  # No answer records
    ns_count = 0  # No authority records
    ar_count = 0  # No additional records
    header = build_dns_header(transaction_id, flags, qd_count, an_count, ns_count, ar_count)
    question = query[12:]  # Include the original question section
    return header + question
```

Parameters:

- query: The original DNS query received from the client.
- rcode: The error code indicating the type of issue (e.g., 1 for Format Error, 3 for NXDOMAIN).

Transaction ID:

- The transaction ID is extracted from the query to ensure the response matches the client's request.
- If parsing fails, an empty response is returned.

Flags:

The flags field in the DNS header is constructed to indicate:

- It's a standard query response (0x8180).
- The specific error type by bitwise OR-ing with the provided rcode.

Counts:

- qd_count = 1: The response includes one question section, copied from the query.
- an_count = 0: No answer records are included since it's an error response.
- ns_count = 0, ar_count = 0: No authority or additional records are included.

Header Construction:

- The build_dns_header function constructs the DNS header using the transaction ID, flags, and count values.

Question Section:

- The original question section of the query is appended to the header to complete the response.

Return Value:

- The constructed response is returned as a byte string.

Example Error Responses:

Invalid Domain Name (RCODE 3):

- If the parsed domain_name is empty or invalid, the function responds with RCODE 3 to indicate a name error

Unsupported Query Type (RCODE 4):

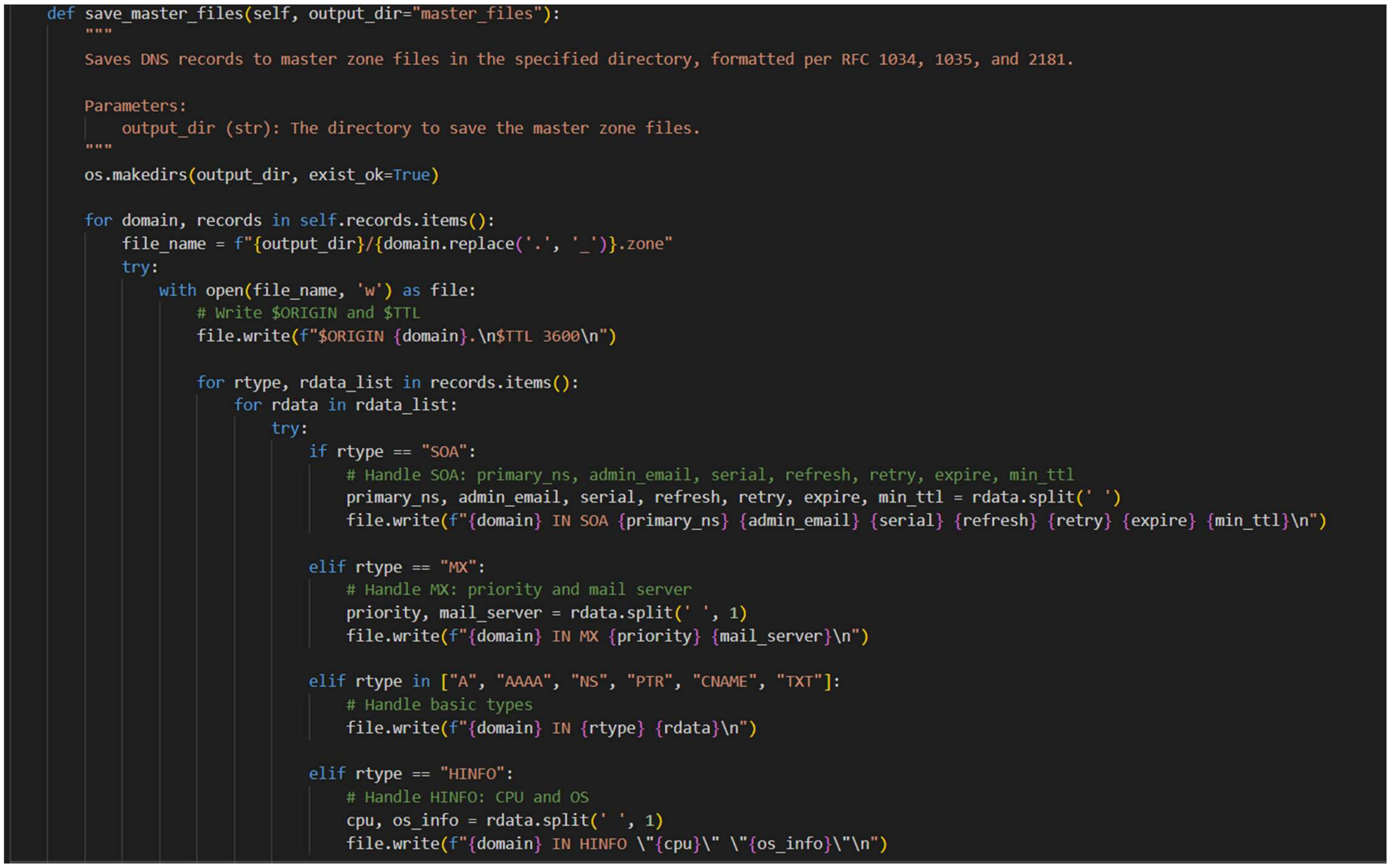
- If the query type is not recognized or not in the supported types (like AAAA), the function responds with RCODE 4.

Successful Query (RCODE 0):

- If the domain name exists and the query type is supported, a response is built, cached, and returned.

  Unhandled Exceptions (RCODE 2):

- If any unexpected error occurs, the function logs the error and returns a Server Failure (RCODE 2) response.



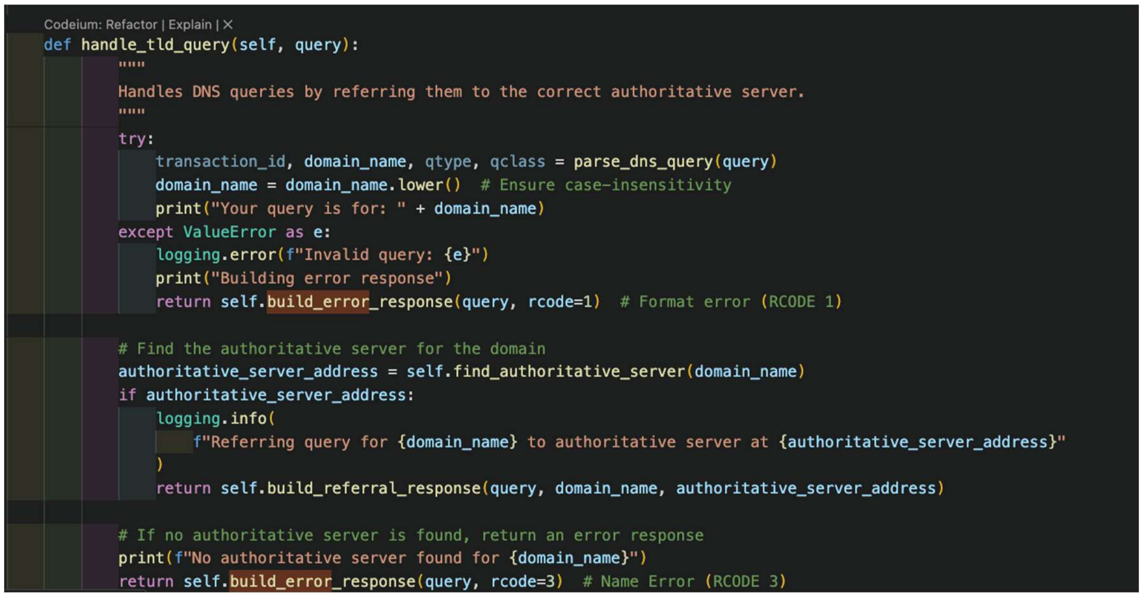
```python
def save_master_files(self, output_dir="master_files"):
    """
    Saves DNS records to master zone files in the specified directory, formatted per RFC 1034, 1035, and 2181.

    Parameters:
        output_dir (str): The directory to save the master zone files.
    """
    os.makedirs(output_dir, exist_ok=True)

    for domain, records in self.records.items():
        file_name = f"{output_dir}/{domain.replace('.', '_')}.zone"
        try:
            with open(file_name, 'w') as file:
                # Write $ORIGIN and $TTL
                file.write(f"$ORIGIN {domain}.\n$TTL 3600\n")

                for rtype, rdata_list in records.items():
                    for rdata in rdata_list:
                        try:
                            if rtype == "SOA":
                                # Handle SOA: primary_ns, admin_email, serial, refresh, retry, expire, min_ttl
                                primary_ns, admin_email, serial, refresh, retry, expire, min_ttl = rdata.split(' ')
                                file.write(f"{domain} IN SOA {primary_ns} {admin_email} {serial} {refresh} {retry} {expire} {min_ttl}\n")

                            elif rtype == "MX":
                                # Handle MX: priority and mail server
                                priority, mail_server = rdata.split(' ', 1)
                                file.write(f"{domain} IN MX {priority} {mail_server}\n")

                            elif rtype in ["A", "AAAA", "NS", "PTR", "CNAME", "TXT"]:
                                # Handle basic types
                                file.write(f"{domain} IN {rtype} {rdata}\n")

                            elif rtype == "HINFO":
                                # Handle HINFO: CPU and OS
                                cpu, os_info = rdata.split(' ', 1)
                                file.write(f"{domain} IN HINFO \"{cpu}\" \"{os_info}\"\n")
```

Format Error (RCODE 1):

- If the query parsing fails due to a malformed request, the function returns RCODE 1

Name Error (RCODE 3):

- If no authoritative server is found for the requested domain, the function returns RCODE 3 to indicate a name error.



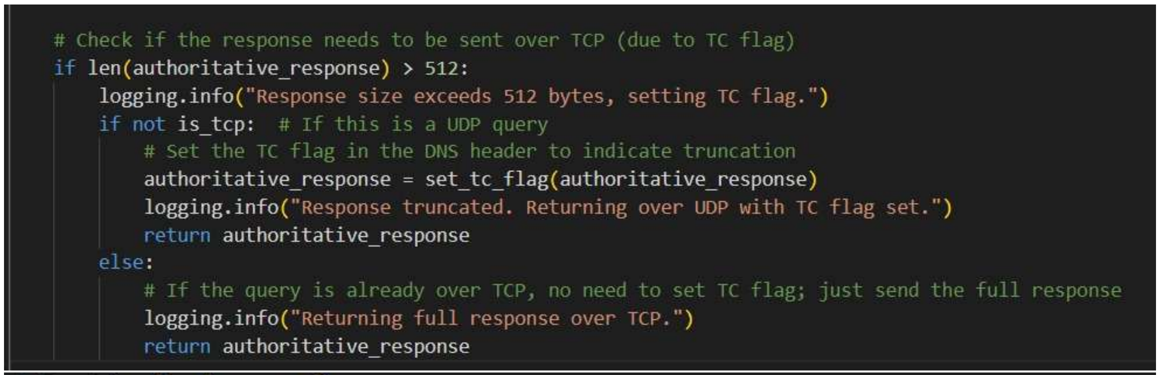
```python
def handle_tld_query(self, query):
    """
    Handles DNS queries by referring them to the correct authoritative server.
    """
    try:
        transaction_id, domain_name, qtype, qclass = parse_dns_query(query)
        domain_name = domain_name.lower()  # Ensure case-insensitivity
        print("Your query is for: " + domain_name)
    except ValueError as e:
        logging.error(f"Invalid query: {e}")
        print("Building error response")
        return self.build_error_response(query, rcode=1)  # Format error (RCODE 1)

    # Find the authoritative server for the domain
    authoritative_server_address = self.find_authoritative_server(domain_name)
    if authoritative_server_address:
        logging.info(
            f"Referring query for {domain_name} to authoritative server at {authoritative_server_address}"
        )
        return self.build_referral_response(query, domain_name, authoritative_server_address)

    # If no authoritative server is found, return an error response
    print(f"No authoritative server found for {domain_name}")
    return self.build_error_response(query, rcode=3)  # Name Error (RCODE 3)
```

DNS Response Handling with TCP and UDP

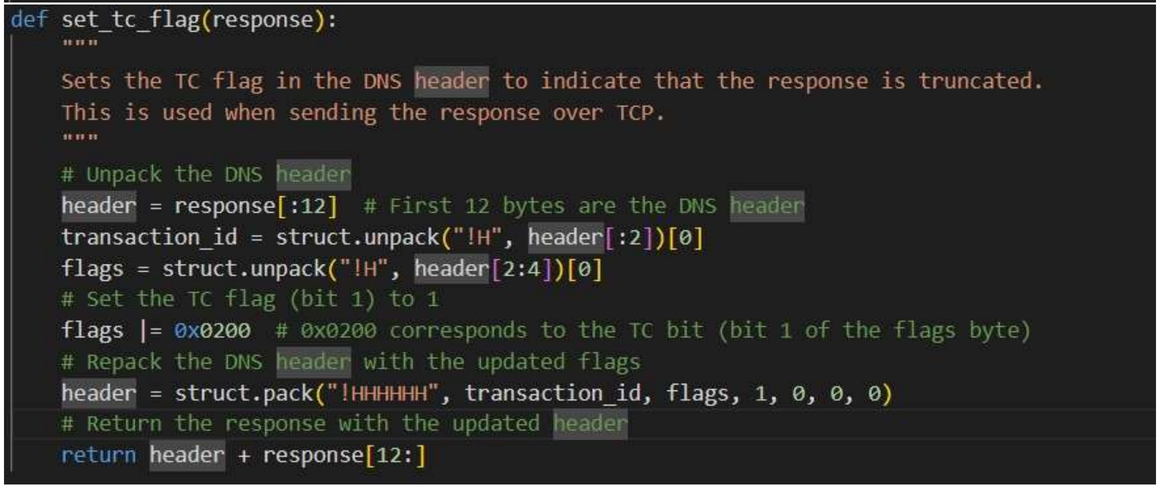
This code snippet handles DNS responses, ensuring that large responses (over 512 bytes) are sent over TCP, while smaller responses can still be sent over UDP. It sets the **TC flag** (Truncated) in the DNS header if the response exceeds the UDP size limit and needs to be sent over TCP.

Key Logic:

- If the response size exceeds 512 bytes (the typical UDP limit for DNS), the system checks if the query is using UDP. If so, it sets the TC flag and forces the response to be sent over TCP.
- If the query is already over TCP, the full response is returned.



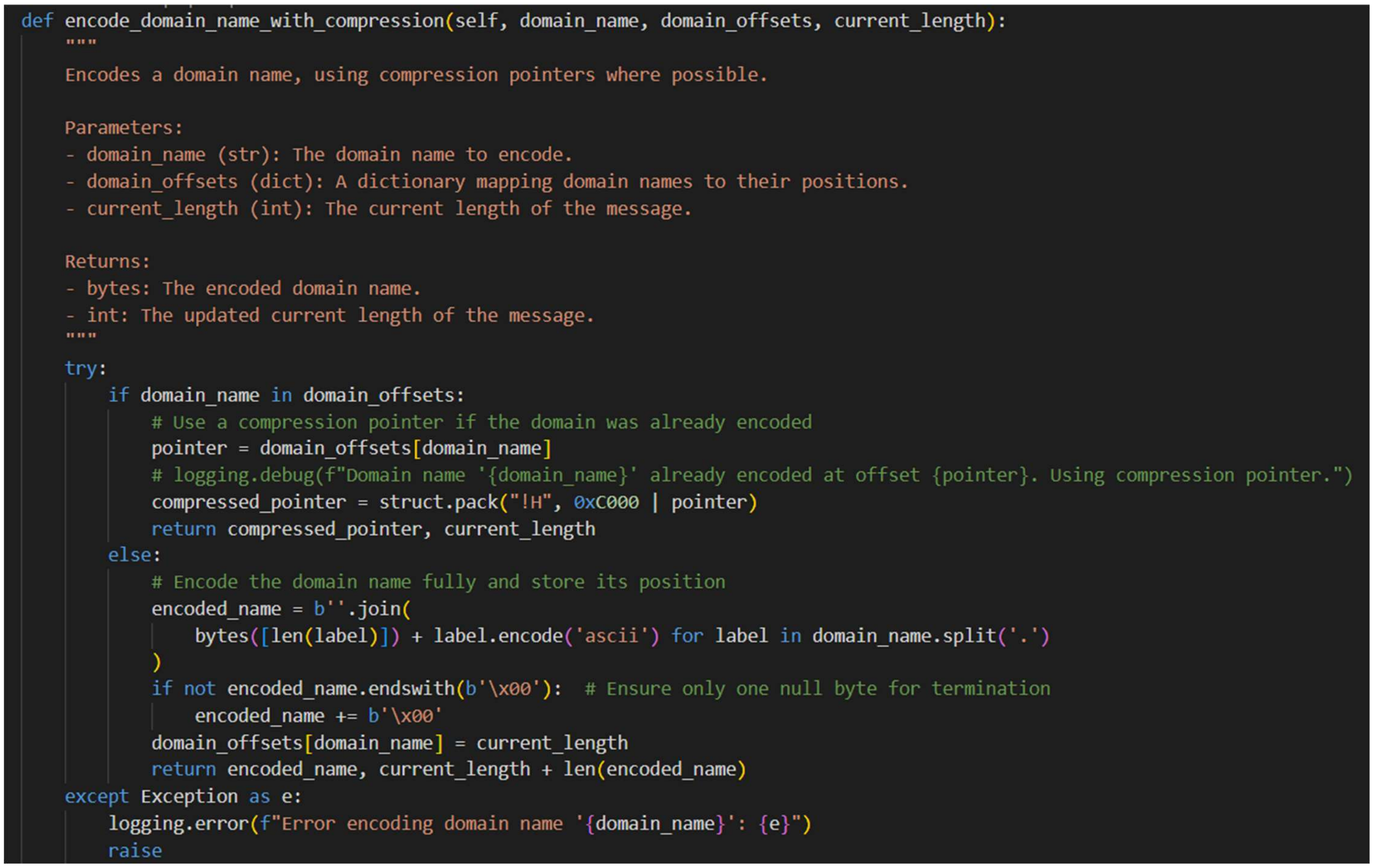
```python
# Check if the response needs to be sent over TCP (due to TC flag)
if len(authoritative_response) > 512:
    logging.info("Response size exceeds 512 bytes, setting TC flag.")
    if not is_tcp:  # If this is a UDP query
        # Set the TC flag in the DNS header to indicate truncation
        authoritative_response = set_tc_flag(authoritative_response)
        logging.info("Response truncated. Returning over UDP with TC flag set.")
        return authoritative_response
    else:
        # If the query is already over TCP, no need to set TC flag; just send the full response
        logging.info("Returning full response over TCP.")
        return authoritative_response
```



```python
def set_tc_flag(response):
    """
    Sets the TC flag in the DNS header to indicate that the response is truncated.
    This is used when sending the response over TCP.
    """
    # Unpack the DNS header
    header = response[:12]  # First 12 bytes are the DNS header
    transaction_id = struct.unpack("!H", header[:2])[0]
    flags = struct.unpack("!H", header[2:4])[0]
    # Set the TC flag (bit 1) to 1
    flags |= 0x0200  # 0x0200 corresponds to the TC bit (bit 1 of the flags byte)
    # Repack the DNS header with the updated flags
    header = struct.pack("!HHHHHH", transaction_id, flags, 1, 0, 0, 0)
    # Return the response with the updated header
    return header + response[12:]
```

- **len(authoritative_response) > 512**: The system checks if the DNS response exceeds 512 bytes.
- **set_tc_flag(response)**: A helper function that modifies the DNS header by setting the TC flag, indicating truncation, and ensures the response can be sent over TCP.
- **Logging**: Information is logged to track whether the response is truncated or sent fully.

Use of compression pointer



```python
def encode_domain_name_with_compression(self, domain_name, domain_offsets, current_length):
    """
    Encodes a domain name, using compression pointers where possible.

    Parameters:
    - domain_name (str): The domain name to encode.
    - domain_offsets (dict): A dictionary mapping domain names to their positions.
    - current_length (int): The current length of the message.

    Returns:
    - bytes: The encoded domain name.
    - int: The updated current length of the message.
    """
    try:
        if domain_name in domain_offsets:
            # Use a compression pointer if the domain was already encoded
            pointer = domain_offsets[domain_name]
            # logging.debug(f"Domain name '{domain_name}' already encoded at offset {pointer}. Using compression pointer.")
            compressed_pointer = struct.pack("!H", 0xC000 | pointer)
            return compressed_pointer, current_length
        else:
            # Encode the domain name fully and store its position
            encoded_name = b''.join(
                bytes([len(label)]) + label.encode('ascii') for label in domain_name.split('.')
            )
            if not encoded_name.endswith(b'\x00'):  # Ensure only one null byte for termination
                encoded_name += b'\x00'
            domain_offsets[domain_name] = current_length
            return encoded_name, current_length + len(encoded_name)
    except Exception as e:
        logging.error(f"Error encoding domain name '{domain_name}': {e}")
        raise
```

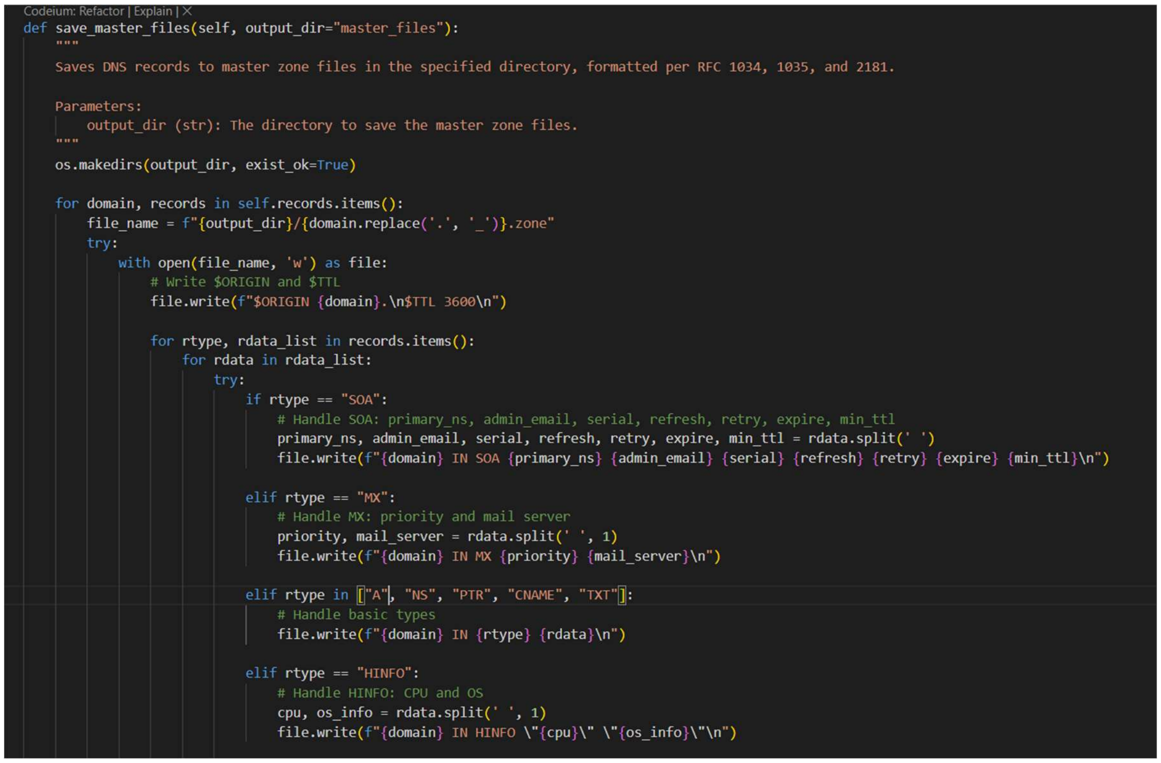
This Python function, **encode_domain_name_with_compression**, encodes a domain name into the format used in DNS (Domain Name System) messages, supporting DNS name compression.

If the domain name has been encoded previously (tracked via the domain_offsets dictionary), the function uses a compression pointer to refer back to its earlier occurrence, which saves space. The pointer is created by combining a special identifier (0xC000) with the offset where the domain name was first encoded.

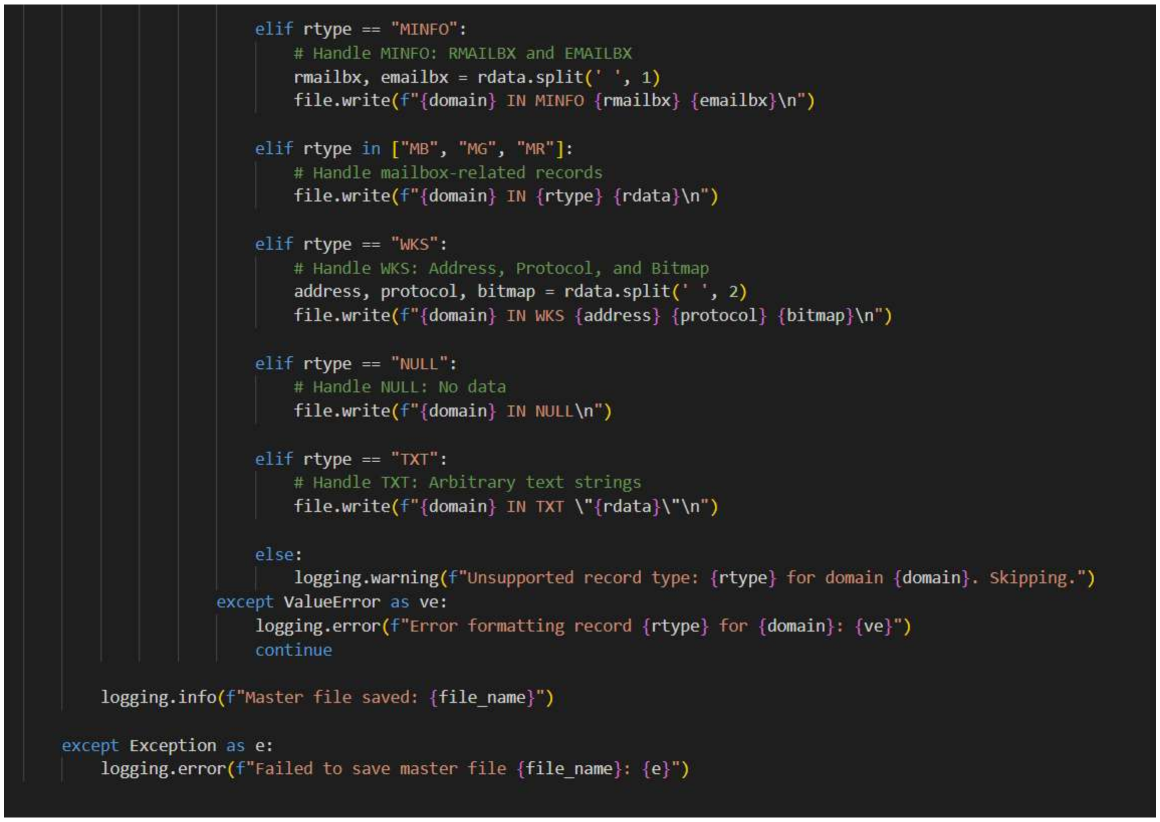
If the domain name has not been previously encoded, the function encodes it fully by splitting the name into labels (parts separated by dots, e.g., "example.com" into ["example", "com"]). Each label is prefixed by its length in bytes and combined into a single byte string. The function also ensures the encoded name ends with a null byte (\x00), indicating termination. It updates the domain_offsets dictionary with the domain's position and returns the encoded name alongside the updated message length.

This approach optimizes the DNS message size by reusing encoded domain names whenever possible.

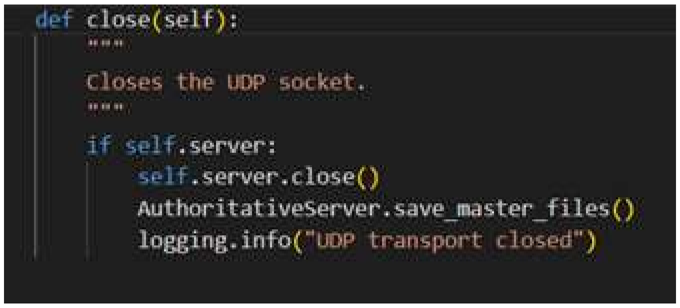
Master file



```python
def save_master_files(self, output_dir="master_files"):
    """
    Saves DNS records to master zone files in the specified directory, formatted per RFC 1034, 1035, and 2181.

    Parameters:
        output_dir (str): The directory to save the master zone files.
    """
    os.makedirs(output_dir, exist_ok=True)

    for domain, records in self.records.items():
        file_name = f"{output_dir}/{domain.replace('.', '_')}.zone"
        try:
            with open(file_name, 'w') as file:
                # Write $ORIGIN and $TTL
                file.write(f"$ORIGIN {domain}.\n$TTL 3600\n")

                for rtype, rdata_list in records.items():
                    for rdata in rdata_list:
                        try:
                            if rtype == "SOA":
                                # Handle SOA: primary_ns, admin_email, serial, refresh, retry, expire, min_ttl
                                primary_ns, admin_email, serial, refresh, retry, expire, min_ttl = rdata.split(' ')
                                file.write(f"{domain} IN SOA {primary_ns} {admin_email} {serial} {refresh} {retry} {expire} {min_ttl}\n")

                            elif rtype == "MX":
                                # Handle MX: priority and mail server
                                priority, mail_server = rdata.split(' ', 1)
                                file.write(f"{domain} IN MX {priority} {mail_server}\n")

                            elif rtype in ["A", "NS", "PTR", "CNAME", "TXT"]:
                                # Handle basic types
                                file.write(f"{domain} IN {rtype} {rdata}\n")

                            elif rtype == "HINFO":
                                # Handle HINFO: CPU and OS
                                cpu, os_info = rdata.split(' ', 1)
                                file.write(f"{domain} IN HINFO \"{cpu}\" \"{os_info}\"\n")
```



```python
                            elif rtype == "MINFO":
                                # Handle MINFO: RMAILBX and EMAILBX
                                rmailbx, emailbx = rdata.split(' ', 1)
                                file.write(f"{domain} IN MINFO {rmailbx} {emailbx}\n")

                            elif rtype in ["MB", "MG", "MR"]:
                                # Handle mailbox-related records
                                file.write(f"{domain} IN {rtype} {rdata}\n")

                            elif rtype == "WKS":
                                # Handle WKS: Address, Protocol, and Bitmap
                                address, protocol, bitmap = rdata.split(' ', 2)
                                file.write(f"{domain} IN WKS {address} {protocol} {bitmap}\n")

                            elif rtype == "NULL":
                                # Handle NULL: No data
                                file.write(f"{domain} IN NULL\n")

                            elif rtype == "TXT":
                                # Handle TXT: Arbitrary text strings
                                file.write(f"{domain} IN TXT \"{rdata}\"\n")

                            else:
                                logging.warning(f"Unsupported record type: {rtype} for domain {domain}. Skipping.")
                        except ValueError as ve:
                            logging.error(f"Error formatting record {rtype} for {domain}: {ve}")
                            continue

            logging.info(f"Master file saved: {file_name}")

        except Exception as e:
            logging.error(f"Failed to save master file {file_name}: {e}")
```



```
def close(self):
    """
    Closes the UDP socket.
    """
    if self.server:
        self.server.close()
        AuthoritativeServer.save_master_files()
        logging.info("UDP transport closed")
```

The **save_master_files** function is responsible for generating zone files that define DNS records for a given domain. These files are saved in a format compliant with RFC 1034, 1035, and 2181. The function iterates over the records of each domain, writing them to individual zone files in the specified directory. For each record, it properly formats the DNS Resource Record (RR) fields, such as the record type, name, class (IN), and data (e.g., IP address, mail server, or hostname).

The function supports a wide range of record types, including SOA, MX, A, AAAA, NS, PTR, CNAME, TXT, HINFO, MINFO, MB, MG, MR, WKS, and NULL. It ensures each record type is formatted with the required parameters, such as TTL, priority, and additional metadata, where applicable. Errors in individual records are handled gracefully, with invalid entries being skipped and logged. By producing zone files in a standardized format, this function ensures interoperability with DNS servers and compliance with DNS protocol standards.