

Programming Assignment 2 — Simple Name Service

*Assigned: February 13**Due: February 23, 23:59:59*

1 Introduction

In this project, you will write a simple name server and a resolver. The real domain name service in the Internet is used to resolve the mapping between a hostname and an IP address.

As usual, the server will run on a machine at a well-known port number. The clients, called resolvers, will contact the server with queries about hostnames and IP addresses. The server has a resident database of hostname to IP address mappings. The database and routines to access it will be provided to you. On receiving a name service query, the server will lookup the query in the database and respond with an appropriate answer.

The server and the client (resolver) will use UDP to communicate with each other. UDP is an unreliable protocol. Thus, for various reasons, UDP packets may be lost in the network.

In this assignment, you will need to implement two new mechanisms:

- **Re-transmissions from the client:** Since data transfer in UDP is not reliable, you will need to create a re-transmission mechanism in the client for queries that do not lead to a response from the server.
- **Checksum computation:** To ensure that the data received at either end has not been corrupted in the network, on receiving a packet from the network, you should first compute the checksum to see if the data is correct. The server and the resolvers will silently ignore all corrupted packets.

2 Protocol Specification

The protocol works as shown in Figure 1. The client will construct queries of different types and send them to the server. The server will query its internal database for each client query and send back a response. The client is responsible for re-transmitting any query that gets lost or corrupted in the network. Details of the protocol follow.

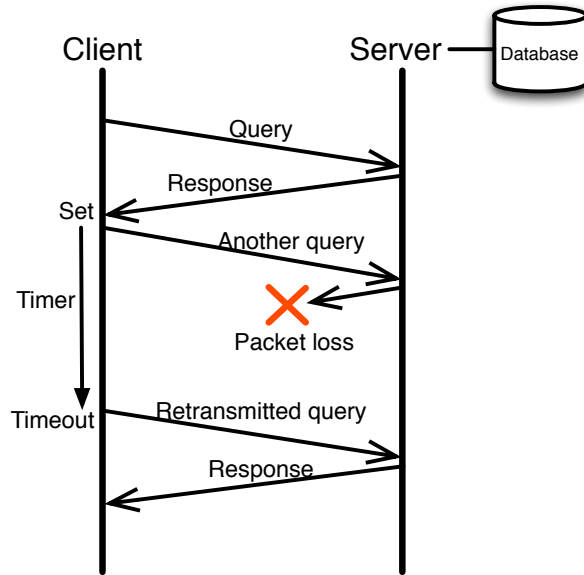
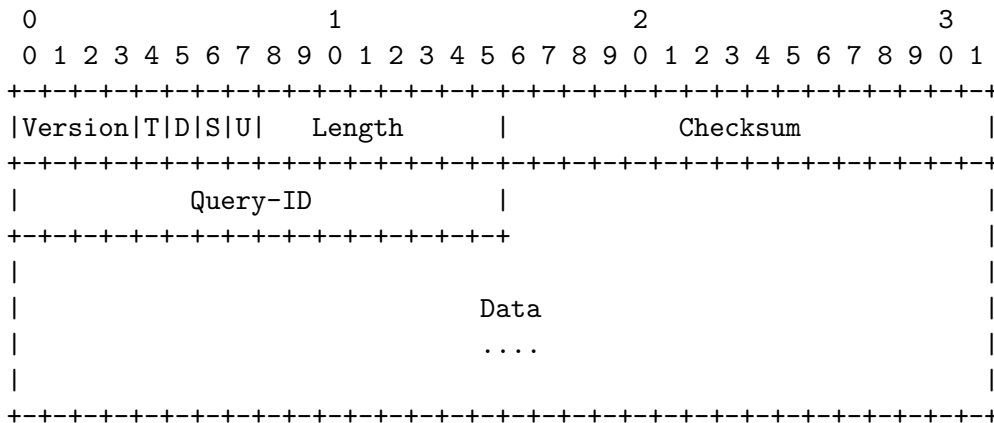


Figure 1: protocol outline

2.1 Packet Format

Both query and response packets have the same packet structure shown below. The maximum size of each packet is 256 bytes.



Version (4 bits) The version field for the protocol should be set to 1111.

Packet-type (T) (1 bit) Set to 0 for query packet, and 1 for response packets

Data-type (D) (1 bit) The data bit is set to 0 if the data field is an IP address, and 1 if it is a hostname.

Status (S) (1 bit) The status bit is set to 1 in query packets. In response packets, the status bit is set to 1 if a match is found in the database, and set to 0 otherwise.

Unused (U) (1 bit) Always set to 0

Length (8 bits) Contains the length of the **Data** field in number of octets (bytes), including any padding.

Checksum (16 bits) The checksum is computed to ensure the integrity of the packet. The details about checksum computation is described below.

Query-Id (16 bit) This is an unsigned 16 bit query ID randomly assigned by the resolver to the query. The name server copies the query ID from the query packet into the response packet.

Data (Variable length) This field contains the hostname or the IP address as required. For example, a client trying to resolve a hostname must fill up the **Data** field with the hostname in question, and the server responding to that query must fill it up with the IP address. Note that the **D**-bit is an indicator of whether a hostname or an IP address is stored in the **Data** field.

An IP address is stored in the network format as a 32-bit integer (see `inet_aton(3)` and `inet_ntoa(3)` for conversion between dotted decimal format and integer format of the addresses).

A hostname is stored as a ASCII character sequence, where each character occupies an octet (byte). The hostname is not necessarily terminated by a NULL character, since you can determine the length of it by referencing the **Length** field. If the size of hostname is not a multiple of 16 bits, pad it to the nearest 16-bit boundary with 0x00.

The server will receive packets on a well-known port. It will ignore any packet whose type it does not know about. It will also ignore any packet that contains a bad checksum. In general, both the server and the resolver will ignore ill-formed packets. Upon receiving a valid query, the server will query its internal database for the information requested and will return what it finds in the database.

Note: the server will (re-)construct a new packet with its answer. For queries that do not have an entry in the database, the data part is 0 bytes and the **S**-bit is set to 0. For queries that do have an entry in the database, the answer to the query will be sent back. The client is responsible for mapping the answer to the original query using a **Query-Id**.

2.2 Retransmission and checksum

If the server does not reply within a pre-defined time period (i.e., the retransmission timer expires before receiving responses from the server), the client will re-try its query. The server is completely stateless, i.e., it does not *remember* which queries it has served, from whom, and when. If it receives the same query from the same resolver, the server will answer as appropriate.

Note: Clients *must* be able to disambiguate stale responses from the server. Thus, if a client gets a response which does not match the latest `Query-Id`, it should ignore this response and *not* re-set the retransmission timer.

The checksum is one's complement of the one's complement sum of all 16-bit words in the packet. Compute the checksum in the following manner:

- When sending a packet:
 - Initialize the checksum field to 0.
 - Compute the one's complement sum of all 16-bit words in the packet.
 - Take one's complement on the sum obtained above and fill it into the checksum field.
- When receiving a packet:
 - Compute the one's complement sum of all 16-bit words in the packet.
 - Take one's complement on the sum obtained above.
 - Accept the packet iff the checksum is 0

3 Requirements

You will have to write both the server and the client for this assignment. No TCP connections are allowed. We will have a server up and running for you to test and debug your client. The server we provide, however, will not drop, delay, or damage the packets it sends. You may modify your own server to simulate poor network conditions if you want to test your client more intensively, but the server you turn in should operate normally (i.e., no dropping, delaying, or damaging the packets).

The name server and the client executables will accept the following command line arguments:

- Server:
`server [-p <port>] [-d <database-file-name>]`

- (Optional) port : Port of the host at which the server will run. Note that the ports you can use are the same as the previous assignment. Default value: `SERVER_PORT`.
- (Optional) database-file-name : Name of the database file name to use for mapping the IP addresses to hostnames. Default value: `DEFAULT_DB_NAME`.

- Resolver:

```
client [-h <hostname>] [-p <port>] \
      [-t <timeout>] [-i <max-retries>] [-r] <data>
```

- (Optional) hostname: Hostname of the host at which the server is running. Default value: `localhost`
- (Optional) port : Port of the host at which the server is running. Default value: `SERVER_PORT`.
- (Optional) timeout: Indicates in seconds, how long to wait before regenerating an unanswered query. Default value: 5.
- (Optional) max-retries: Indicates the number of times the resolver will re-generate the query, before it quits, if no response is received from the server. Default value: 3.
- (Optional) r flag: Use the `-r` flag to indicate a reverse DNS lookup, i.e., the resolver queries with an IP address and expects to receive the corresponding hostname from the server. If this flag is not provided, the resolver will query with a hostname.
- data: This is a string that will form the data portion of the query. Hostname queries *must* specify a dotted quad here, while IP address queries will specify a hostname. Note that the length of the hostname should be able to fit in a single packet.

4 Database Interface Functions

The server will query entries in the database using the following interface routines :

- `int db_open(char *file_name);`
opens and initializes database, where the actual data is stored in the file `file_name`.
- `int db_lookup_by_hostname(char *result, int result_size, char *query);`
queries the IP address associated with a hostname. The hostname is passed in as a string in `query`, and the result of the query will be stored as a string in `result`, where `result_size` specifies the size of the `result` buffer. The calling function is responsible for allocating storage.

The returned IP address will be in dotted decimal format. A negative value is returned if no match is found.

- `int db_lookup_by_ipaddr(char *result, int result_size, char *query);`
queries the hostname associated with an IP address. The IP address is passed in as a string in `query` (in dotted decimal format), and the result of the query will be stored as a string in `result`, where `result_size` specifies the size of the `result` buffer. The calling function is responsible for allocating storage. A negative value is returned if no match is found.
- `void db_close(void);`
closes the database.

The functions described above will return a negative value on error.

5 Submission

- Please submit your code as described in the forum.
- What to turn in: All of your source code along with a Makefile. Your Makefile must produce executables named `client` and `server` after a single `make` command.

6 Hints and notes

- You will need `select()` function to implement the retransmission timer mechanism.
- You will need `sendto()` and `recvfrom()` to send and receive UDP packets.
- Look at `data.txt` to see what queries you can make, and what the correct response should be.
- Get a simple client working first, then the server, then do the retransmission mechanism. Tweak your server to test the retransmit-enabled client (but submit a working server).