CMSC 417 Computer Networks **Spring 2010**

Programming Assignment 2 — Triple Query Service

Assigned: Feb. 16, 2010 *Due: Feb. 25. 2010, 11:59:59 PM*

1 Introduction

In this assignment you will write a server and a client. We will continue to use the paradigm of the Pythagorean triple but in a slightly different way. The server will run on a well known port number. The clients will read Pythagorean triples from an input file and they will send these triples to the server. The server will verify the triples and will respond with the validity of the triples.

The server and the client will communicate using UDP (User Datagram Protocol) . UDP is a best effort protocol and does not guarantee delivery of messages. Your programs will implementing two new mechanisms

- **Re-transmissions** 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 response from the server.
- **Checksum computation:** Checksums ensure that has not been corrupted in the network. Upon receiving a packet from the network, you should first compute a checksum to see if the data is correct. The server will silently ignore all corrupted packets.

2 Protocol Specification

The protocol works as shown in Figure 1. The client constructs messages of different types and sends them to server. The server validates each of the triples that the client sends and responds with the results. The client is responsible for re-transmitting any query that gets lost in the network. Details of the protocol follow.

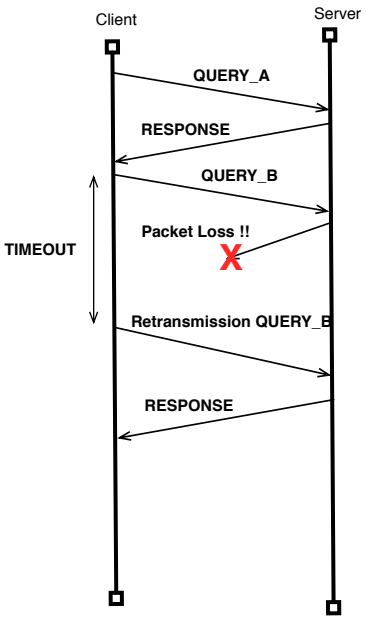

Figure 1: Assignment's Protocol

2.1 Packet Header

Both query and response packets have the header structure shown below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|T|F|S|U|   Length      |           Query-Id            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Checksum             |           Data                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               Data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- `Version` (4 bits) The version field is set to 0010 for this project.
- `Type` (1 bit) Set to 0 for query packet, and 1 for a response packet.
- `First Triple Status` (1 bit) In query messages, the first triple status will be always 0. In response messages server will set 0 for invalid and 1 for valid triples.
- `Second Triple Status` (1 bit) In query messages, the second triple status will be always 0. In response messages server will set 0 for invalid and 1 for valid triples.
- `Unused` (1 bit) Always 0.
- `Length` (8 bits) Contains the length of the packet in number of bytes.

- `Query-Id` (16 bits) This is a randomly generated 16 bit number that the client uses to map server responses to outstanding requests. (This is described in a bit more detail later). The server should copy the `Query-Id` from the query packet into the response packet.
- `Checksum` (16 bits) This is a ones-complement checksum of the entire packet, including header and data. The checksum computation is also explained below.
- `Data` The format of the data field changes depending on the type of the packet.
- For the `query` packet the data consists of 6 unsigned 16 bit numbers that represent each one of the numbers of the two triples. Each number is an unsigned short integer (0...65535). The input does not contain numbers larger than 65535. The packet should be 4-byte aligned, that means if the total length of the packet is not a multiple of 4, there should be an appropriate number of NULL characters to pad the data field so that the total length is a multiple of 4. One such packet might look like this:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Checksum             |               a1              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               b1              |               c1              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               a2              |               b2              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               c2              |                ...            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- For the `response` packet, we don't need to use the data portion. Server's response is included in the two status fields of the header. As we mentioned before the packets should be 4-byte aligned, thus you should pad appropriately with NULL bytes if need be for the response packet.
- All multi-byte numeric data, should be transmitted in NETWORK byte order.

The server will receive packets on a well-known port. It will ignore any packet whose type it not known. It will also ignore any packet that contains a bad checksum. In general, both the server and the client will ignore ill-formed packets. Upon receiving a valid query, server with some probability decides to drop or to answer the query. The client is responsible for mapping the answer to the original query using a `Query Id` (defined in the next section).

2.2 Retransmission and Checksums

If the server does not reply within a pre-defined time period, 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, when. If it receives the same query from the same client, the server will answer appropriately.

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

the retransmission timer. Likewise, a malformed packet should be dropped by the client and not re-set the timer.

The checksum contained in both the query and the response is a ones-complement sum of all the contents in the packet. Compute the checksum in the following manner:

- When sending a packet:
  - Initialize the checksum field to 0x0
  - Compute the ones-complement sum of the entire packet and put the result in the checksum field
- When receiving a packet:
  - Compute the ones-complement sum of the entire packet, including the checksum
  - Accept the packet *if and only if* the sum is zero.

3 Deliverables

You will have to write both the server and the client for this assignment. Your client should issue queries for all the triples included to the input file.

- Server:
  `server -p port -d drop`
  - port: Port of the host at which the server will run. Note that you cannot bind to a port below 1024 without being superuser. Given that your class account login id is `cs4170xx`, the ports you should use are `10xxn` where `n` ranges between 0 and 9 (inclusive). Thus, if your login id is `cs417060`, you will use port range 1060[0-9]. Your final assignment must work with *any* port, but when *you* are testing your project you should only use the ports you have been allocated to avoid collision with others.
  - drop: drop probability that your server uses to drop valid queries
- Client:
  `client -f file -p port -t timeout -r retries`
  - file: the input triples file.
  - port: Port of the host at which the server is running.
  - timeout: Indicates in seconds how long to wait before regenerating an un-answered query. (a reasonable value is 20 seconds.)
  - retries: Indicates the number of times the client will re-generate the query before it quits if no response is received from the server.

    This program should simply print the value that is returned by the server, along with with the appropriate triple "a,b,c: 1" if the triple is valid, or "a,b,c: 0" if the triple is not valid if there is no correct response from the server `max-retries` attempts. If the query is left not answered the output will be "a,b,c: -1".

4 Hints and Notes

- In order to implement the command line parsing, consider using `getopt`.
- Get a simple client working first, then the server, then do the retransmit mechanism. Tweak your server to test the retransmit-enabled client (but submit a working server).
- Please check the forum frequently.
- Please don't submit code that generate warnings. Check your code with the `-Wall -pedantic`
- Provide a Makefile.