Advanced Networks — Laboratory 7

Juliusz Chroboczek

8 April 2025

In this practical, we will use a UDP-based client-server request-response protocol, where all datagrams have the following syntax:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Id                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |            Length             | Body...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

The field *Id* contains an identifier that is used for matching requests to replies. It is chosen arbitrarily by the client, and copied from the request into the response by the server.

The field *Type* contains the type of the message; if it is between 0 and 127, then the message is a request; otherwise it is a reply. The field *Length* contains the length of the body in bytes. The field *Body* contains the body of the message, whose meaning depends on the value of the Type field.

If the datagram is longer than $4 + 1 + 2 + \textit{Length}$ bytes, then the extra data is ignored. If the datagram is shorter, then it is erroneous, and should be discarded.

The protocol is strictly request-response: to every request corresponds exactly one response. The following requests are defined:

- Type = 0 : *get-quotation*. Asks the server to return a random literary quotation. The body must be empty (of length 0).

The following replies are defined:

- Type = 128 : *quotation*. The body is a literary quotation encoded in UTF-8.
- Type = 129 : *error*. The body is an error message encoded in UTF-8.

Exercice 1.

1. Write a program that fetches and decodes the JSON object from the address `https://galene.org:8447/udp-addresses.json` to obtain the set of socket addresses of the server. What is the purpose of this step? What other protocol could I have used to publish this data?
2. Make your program send a request to the first IPv4 address of the server and display the body of the response. Take care to verify that the *Id* of the reply is equal to the *Id* of the request. What happens if the request is lost in the network? If the reply is lost?

3. Make your program wait for the reply for up to two seconds and send a new request if the reply is not received. (You may use the method `SetReadDeadline`). Should you be using the same *Id* for reemis Should you be using the same *Id* for reemission? What property of the protocol makes this irrelevant?
4. Modify your program to use the *exponential backoff* algorithm.

Exercice 2.

1. Modify the program from the previous exercice so that it runs in an infinite loop and displays a different quotation every 5 s.
2. Modify the program to estimate the RTT of every request. You will maintain two variables:

$$\begin{aligned}\text{RTT} &:= 2\,s \text{ initially}\\ \text{RTTvar} &:= 0\,s \text{ initially}\end{aligned}$$

   and at every new sample τ, you will do:

$$\begin{aligned}\delta &:= |\tau - \text{RTT}|\\ \text{RTT} &:= \alpha\,\text{RTT} + (1-\alpha)\tau\\ \text{RTTvar} &:= \beta\,\text{RTTvar} + (1-\beta)\delta\end{aligned}$$

   You may use $\alpha = 7/8$ et $\beta = 3/4$.
   If a request is resent, there is an ambiguity: the response may correspond to either the original request, or to the resent one. You may solve this issue in one of two ways:

   – by using a different *Id* for the resent request; or
   – by systematically considering that a reply corresponds to the original request, which will cause you to over-estimate the RTT (which is less serious than under-estimating it).

   Display the RTT and RTTvar values computes, and make sure they make sense.
3. Modify your program so that the initial delay of your exponential backoff algorithm is equal to

$$\text{RTO} = \text{RTT} + 4\,\text{RTTvar}.$$

Exercice 3. Your program only uses a single address of the server. Modify it to do something smart if the server has multiple addresses.