Web — Master 1 IFI
Lab Session #1: HTTP

Andrea G. B. Tettamanzi
Université côte d'Azur
andrea.tettamanzi@univ-cotedazur.fr

Academic Year 2019/2020

Abstract

We will build a simple HTTP client in Python, to test various requests and correctly receive and print the response messages on the console.

1 Introduction

HTTP is an application protocol, which builds on top of the TCP/IP stack of protocols. This means that its messages are exchanged between clients and servers through TCP sockets.

As a consequence, we will have to use the `socket` module of Python, whose documentation can be found at `https://docs.python.org/3/library/socket.html`. This module provides access to the BSD socket interface, which is a low-level networking interface. Higher-level networking APIs exist, but it is a good idea to get started using this one because of its simplicity and of the complete control it leaves to the developer.

You are assumed to be already familiar with socket programming, possibly in another programming language; however, socket programming in Python does not differ too much from socket programming in C or any other language.

2 Requirements

1. We are going to build a simple HTTP client. An HTTP client is also a socket client. Therefore, the first step is to implement a simple TCP socket client. Here is the Python code of an embryonic client, from which you can start:

```
#!/usr/bin/env python3

import socket

HOST = 'www.w3c.org'  # The server's IP address
PORT = 80             # By default, HTTP uses port 80

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'GET /index.html HTTP/1.1\r\nHost: '
      + HOST.encode("utf-8") + b'\r\n\r\n')
    response = s.recv(1024)

print('Received', repr(response))
```

2. Our client should accept a command-line argument specifying the URI of the resource for which we are to issue an HTTP request.

   (a) The client will check that the protocol is `http` (or nothing, which will be interpreted as the default protocol, namely `http`) and issue an error message otherwise.

   (b) The URI will be parsed and the `HOST` and `PORT` changed accordingly, if applicable.

   (c) Any syntax error in the URI will be reported by the client, via an informative error message.

3. The client should make it possible to experiment with all the methods supported by HTTP. Therefore, it will accept another command-line argument specifying the method (`GET`, `HEAD`, `POST`, etc.).

   (a) Implement the nine methods discussed during the lesson.

   (b) If an unsupported method is specified on the command-line, show an error message.

   (c) If the request for the specified method provides for a body, expect another command-line argument specifying a file containing the message body.

   (d) If the body file is not specified on the command line, show a prompt to the user, e.g.,

   ```
   Please type the message body (end with ctrl-D):
   ```

   then read the message body from the console.

4. The client will process the response according to its status code:

   (a) Informational (1xx) and successful (2xx) responses will just be copied to the console.

   (b) Redirection (3xx) responses will be reported with an informative warning message and the request will be forwarded to the new URI provided by the response, unless the protocol is other than `http`.

   (c) Client and server error (4xx and 5xx) responses will be reported with an informative error message.