Analyse d'une trame HTTP avec Wireshark

Dans cette manipulation, nous allons explorer les concepts qui vont être utiles pour la suite du cours. En particulier, nous allons analyser le protocole http. Nous allons comprendre les principes de base de son architecture. Certains protocoles de l'internet des objets utilisent les mêmes principes. Nous allons aussi utiliser Wireshark un outil très pratique pour l'analyse des messages qui circule sur le réseau et déboguer en cas d'erreur.

Finalement, nous allons développer en python un serveur web rudimentaire.

En résumé, voici d'autres points abordés dans ce TP :

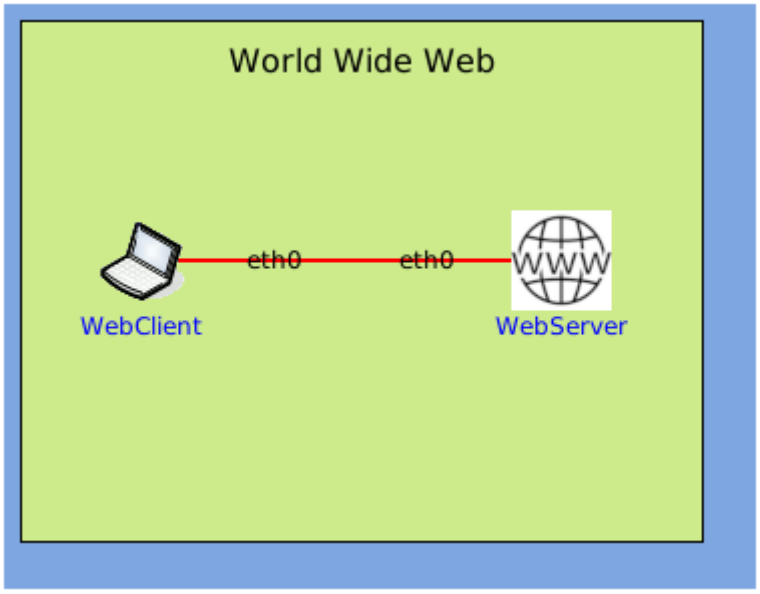
- Trouver les standards et trouver l'information utile,
- Revoir les concepts d'empilement protocolaire, concept essentiel dans les architectures réseau,
- Découvrir la notion de ressource utile dans l'IoT,

Http est un protocole très populaire pour transporter les informations entre les serveurs Web et votre navigateur.

Premièrement, nous allons lancer un émulateur réseau dans la machine virtuelle. Pour ce faire :

- Cliquez sur la deuxième icône en haut à droite de l'écran, juste en dessous de l'abeille,
- Sélectionnez File > Open et
- Choisissez le fichier /home/ilab/PLIDO/tp-wireshark/web.imn

L'écran suivant s'affiche :



Pour exécuter le programme, vous devez cliquer sur la flèche verte en haut à gauche.

Double-cliquez sur l'image de l'ordinateur de gauche.

Tapez dans la fenêtre :

- **export DISPLAY=:0.0**
- **firefox&**

Le caractère & permet de continuer à taper des commandes dans la fenêtre sans arrêter le programme démarré.

- Dans la barre d'adresses, tapez **http://10.10.10.1**

On va disséquer ce que l'on a reçu. Firefox dispose d'un outil d'analyse des données échangées.

- Dans le menu Firefox, allez sur Tools> Web Developer > Network
- Rechargez la page. Vous devez avoir quelque chose comme sur la figure d'en dessous. Nous allons nous intéresser à la fenêtre du bas.
- La partie basse de cette fenêtre donne les échanges qui ont eu lieu. Parfois, une seule ligne apparait. En cliquant dessus, le contenu s'affiche. Cela commence par la réponse suivie de la requête qui l'a initiée. En cliquant sur le bouton *Raw headers*, vous pouvez vous rapprocher encore plus des bits qui ont circulé sur le réseau.

  Question :

  - À l'aide du standard RFC 7231 décrivant le protocole HTTP utilisé par le navigateur, cherchez quelle est la raison du code 404 ?
  - Qui est responsable de l'erreur ? Le client ou le serveur ?

La version 1.1 du protocole HTTP (*Hyper Text Transmission Protocol*) est spécifiée par le RFC 7230. Nous allons regarder une petite description en anglais de l'architecture et des formats des messages.

```
2.  Architecture

  HTTP was created for the World Wide Web (WWW) architecture and has
  evolved over time to support the scalability needs of a worldwide
  hypertext system.  Much of that architecture is reflected in the
  terminology and syntax productions used to define HTTP.

2.1.  Client/Server Messaging

  HTTP is a stateless request/response protocol that operates by
  exchanging messages  across a reliable transport- or
  session-layer "connection".  An HTTP "client" is a
  program that establishes a connection to a server for the purpose of
  sending one or more HTTP requests.  An HTTP "server" is a program
  that accepts connections in order to service HTTP requests by sending
  HTTP responses.
```

Question :

- Quelle organisation de standardisation a publié ce document ?

Microsoft – ISO – IEEE – IETF

Le RFC indique ensuite :

```
The terms "client" and "server" refer only to the roles that these
programs perform for a particular connection.  The same program might
act as a client on some connections and a server on others. [...]

Most HTTP communication consists of a retrieval request (GET) for a
representation of some resource identified by a URI.  In the simplest
case, this might be accomplished via a single bidirectional
connection (===) between the user agent (UA) and the origin
server (O).

         request   >
    UA ======================================= O
                                <   response

A client sends an HTTP request to a server in the form of a request
message, beginning with a request-line that includes a method, URI,
and protocol version, followed by header fields
containing request modifiers, client information, and representation
metadata, an empty line to indicate the end of the
header section, and finally a message body containing the payload
body.

A server responds to a client's request by sending one or more HTTP
response messages, each beginning with a status line that includes
the protocol version, a success or error code, and textual reason
phrase possibly followed by header fields containing
server information, resource metadata, and representation metadata,
an empty line to indicate the end of the header
section, and finally a message body containing the payload body.

The following example illustrates a typical message exchange for a
GET request on the URI "http://www.example.com/hello.txt":

Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi

Server response:

  HTTP/1.1 200 OK
  Date: Mon, 27 Jul 2009 12:28:53 GMT
  Server: Apache
  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
  ETag: "34aa387-d-1568eb00"
  Accept-Ranges: bytes
  Content-Length: 51
  Vary: Accept-Encoding
  Content-Type: text/plain

  Hello World! My payload includes a trailing CRLF.
```

Question :

- Est-ce que les en-têtes HTTP ont une taille fixe (vous pouvez aller voir le RFC 7231 qui donne des indications sur le protocole) ?

- Comment sont construites les lignes optionnelles de l'en-tête ?

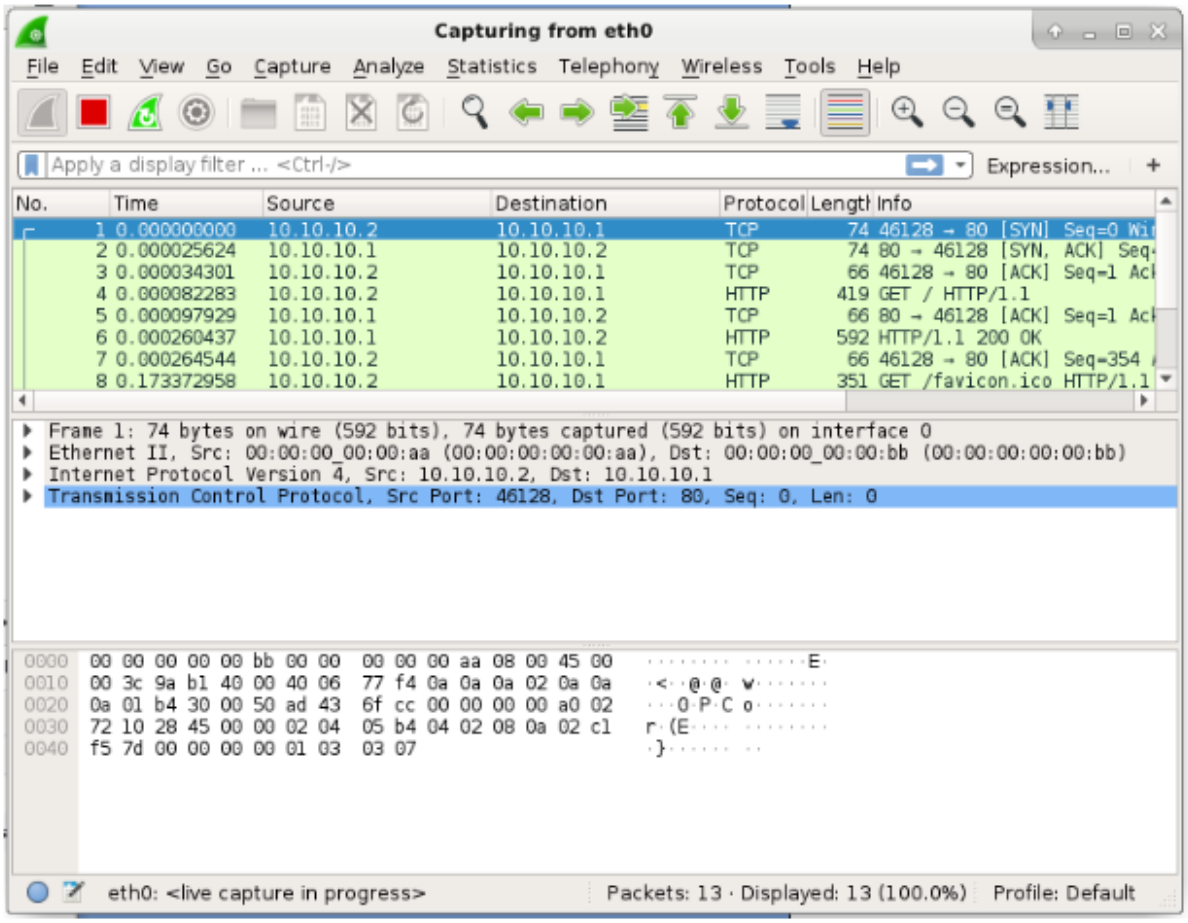
Si vous surfez avec Firefox, *Wireshark* sera votre compagnon sous-marin. Il sert à comprendre les flux d'informations et les décoder. *Wireshark* sert aussi à mesurer de nombreux paramètres du réseau.

On va utiliser l'analyseur de réseau *Wireshark* pour capturer le trafic échangé entre le client et le serveur.

- Sélectionnez la fenêtre de terminal et tapez maintenant **sudo wireshark &**
- Wireshark ouvre une fenêtre qui nous demande quelle interface nous voulons utiliser pour capturer le trafic.
- Double-cliquez sur eth0.

Une nouvelle fenêtre apparait qui montre le trafic capturé.

Sur Firefox, rechargez la page, **en maintenant la touche *shift* (majuscule) appuyée** et en cliquant sur la flèche circulaire.



L'écran de Wireshark se divise en 3 parties :

- En haut, principalement en vert, les trames qui ont circulé sur le réseau sont affichées synthétiquement sur une ligne. Chaque ligne contient :
    - Le numéro de trame capturée, il s'agit d'une information ajoutée par Wireshark,
    - L'heure de capture de la trame. Cette information est aussi ajoutée par Wireshark,
    - L'adresse IP de la machine à l'origine du paquet,
    - L'adresse IP de la machine destinataire du paquet,
    - Le protocole de plus haut niveau contenu dans la trame. Dans notre cas, cela peut être TCP si le message TCP ne contient pas de données, comme lors de l'ouverture de connexion, ou de certains acquittements. On voit également les messages HTTP qui sont bien entendu encapsulés dans TCP,
    - La taille en octets de la trame capturée par Wireshark,
    - Finalement Wireshark fourni un résumé du contenu de la trame, pour comprendre ce qui se passe sur le réseau. Dans la capture, on retrouve pour les messages HTTP, les requêtes GET ou les notifications ;
- Si une trame est sélectionnée dans la liste, elle apparait dans la zone du milieu avec l'empilement protocolaire. Le contenu de chacun de ces protocoles peut être détaillé en cliquant sur le petit triangle à gauche ;
- La fenêtre du bas donne l'équivalent en hexadécimal. Les parties surlignées correspondent aux champs sélectionnés dans la fenêtre du milieu. À noter que l'on retrouve l'information à la fois en hexadécimal et en caractère ASCII, ce qui aide à la lecture quand on cherche une valeur spécifique.

Question :

Dans la capture précédente, à quoi correspondent les colonnes de la zone verte ?

La première colonne:

Le numéro de la trame attribué par Wireshark à sa réception
Le numéro de la trame relevé directement dans la trame Ethernet

La deuxième colonne:

L'heure de réception par Wireshark
L'instant d'émission de la trame

La troisième et la quatrième colonne

Les adresses Ethernet des machines
Les adresses IPv6 dans la représentation compacte
Les adresses IPv4

La cinquième colonne:

Le protocole applicatif
Le dernier (de plus haut niveau) protocole reconnu
Le protocole de niveau 4

La sixième colonne:

La taille en bits de la trame
La taille en octets de la trame

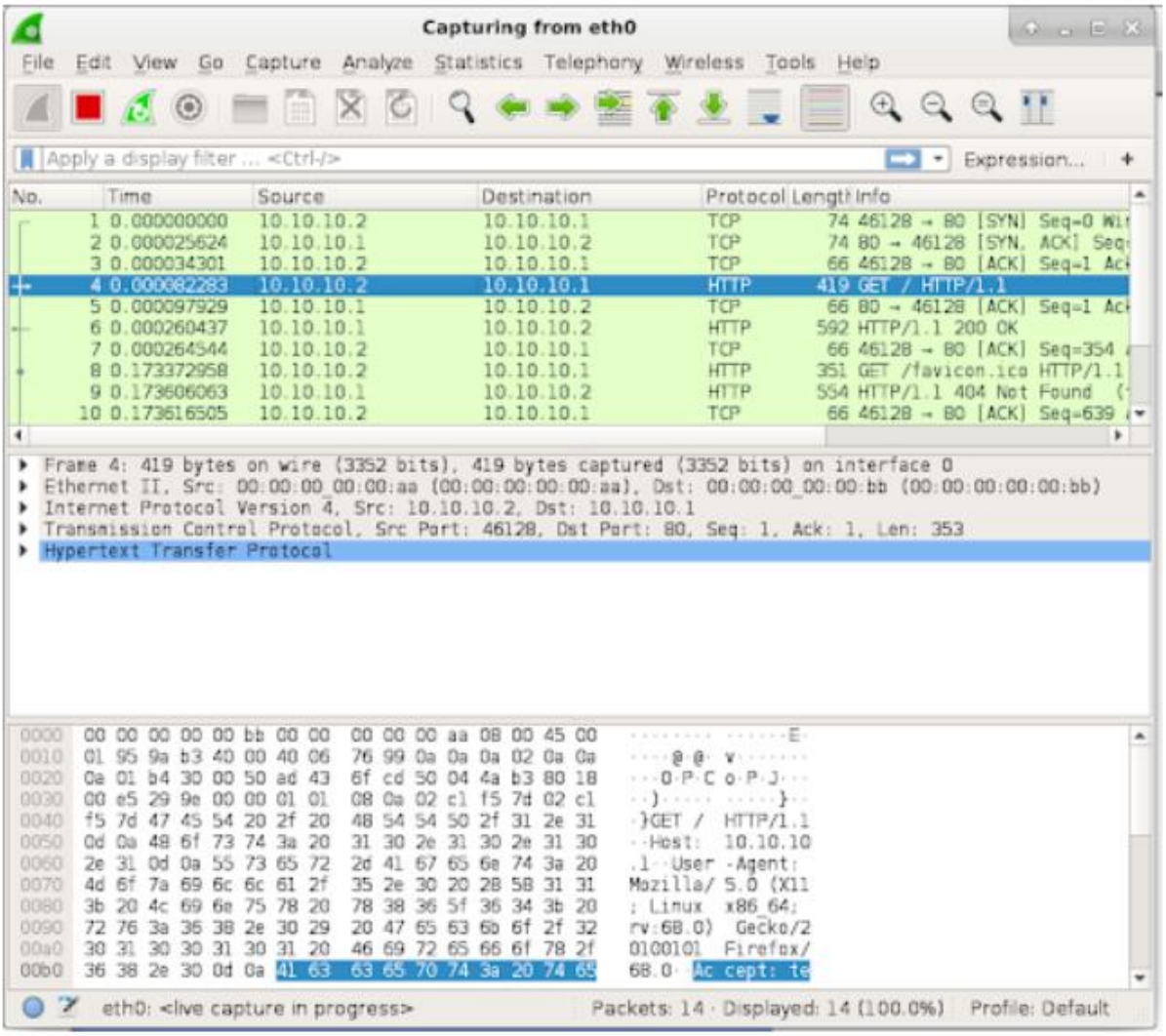
La septième colonne:

Un résumé des informations transportées par le protocole de plus haut niveau
Les options d'IPv4
Le contenu en ASCII du message de plus haut niveau

Les 7 couches :

Wireshark capture des trames Ethernet. À l'intérieur, il y a un en-tête et une partie données. La partie « données » contient les informations venant d'un autre protocole qui divise à son tour l'information en en-tête et données, etc. Le modèle de référence définit 7 couches mais dans la pratique ce nombre peut varier. Et Wireshark est là pour nous montrer cet empilement.

L'exemple ci-dessous montre, dans la fenêtre du milieu, l'empilement protocolaire à l'envers : la couche la plus basse se trouve au sommet. En cliquant sur les différents protocoles, vous pouvez voir où ils se situent dans la trame. En cliquant sur le petit triangle à gauche vous pouvez voir le contenu de l'unité protocolaire.

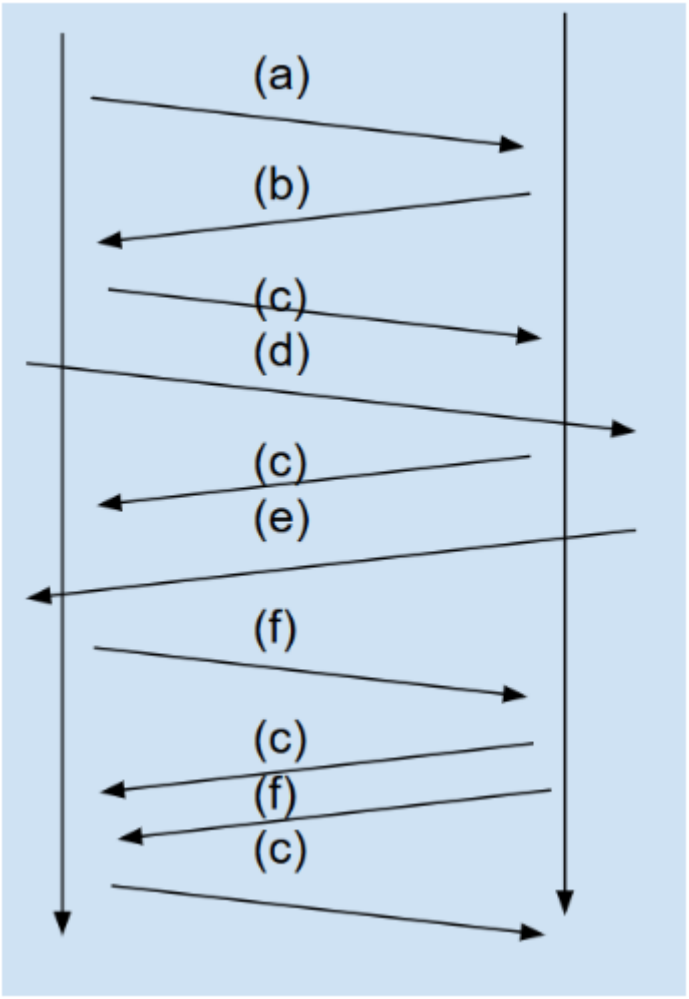


Nous voyons dans cet exemple que toutes les trames capturées dans cet échange contiennent un en-tête TCP. Remarquez la direction de la transmission pour chaque trame : en regardant l'adresse IP, rappelez-vous que celle du serveur distant est 10.10.10.1. Si cette adresse apparaît dans Wireshark comme adresse de destination, il s'agit d'un paquet envoyé par votre ordinateur local. Dans l'autre sens, cette adresse apparaitra comme adresse source.

Pour ouvrir une connexion (c'est-à-dire établir un contexte aux deux extrémités pour superviser l'échange de données), TCP utilise 3 messages : SYN pour indiquer à l'autre extrémité que la machine veut ouvrir la connexion ; l'autre extrémité répond avec un SYN ACK ; puis l'initiateur de la connexion répond à ce dernier message par un message ACK. Cet échange est appelé poignée de main à trois voies.

Une fois la connexion ouverte, le message TCP peut contenir des données.

Pour fermer une connexion (c'est-à-dire supprimer le contexte des deux extrémités utilisé pour superviser l'échange de données), TCP utilise 4 messages : une extrémité envoie un message FIN qui est reconnu par un message ACK. Ensuite, l'autre extrémité envoie un message FIN qui est également acquitté.



Complétez le diagramme temporel précédent représentant l'échange capturé par Wireshark.

REST

- Fermez Wireshark et Firefox
- Ouvrez un terminal sur la machine hébergeant **le serveur web** en double-cliquant sur l'icône.
- Allez dans le répertoire /home/ilab/PLIDO/tp-wireshark
- Dans le terminal, affichez le programme firstpage.py

```
1
2
3
4
5
6
7
```

```
from flask import Flask
app = Flask("My First Web Server")
@app.route('/')

def hello_world():
   return "Hello World"
```

Ce script nécessite quelques explications :

from flask **import** Flask
est utilisé pour importer l'objet Flask à partir du module flask.

app = Flask("My First Web Server")
crée une instance d'un objet Flask, c'est-à-dire un serveur web. Un nom lui est associé à des fins de débogage.

@app.route('/')
C'est la partie la plus délicate du script. @ est un décorateur qui est utilisé en python pour ajouter des propriétés à une méthode. Ici, nous associons un chemin à la fonction suivante. De cette façon, lorsque le serveur Flash recevra une requête sur ce chemin, il appellera la fonction. La fonction sera appelée pour '/'.

def hello world():
la fonction est définie. Aucun argument n'est donné.

   return "Hello World"
retourne ce que le navigateur va afficher.

Ce script peut être exécuté en mettant le nom du fichier dans une variable et en lançant flask.

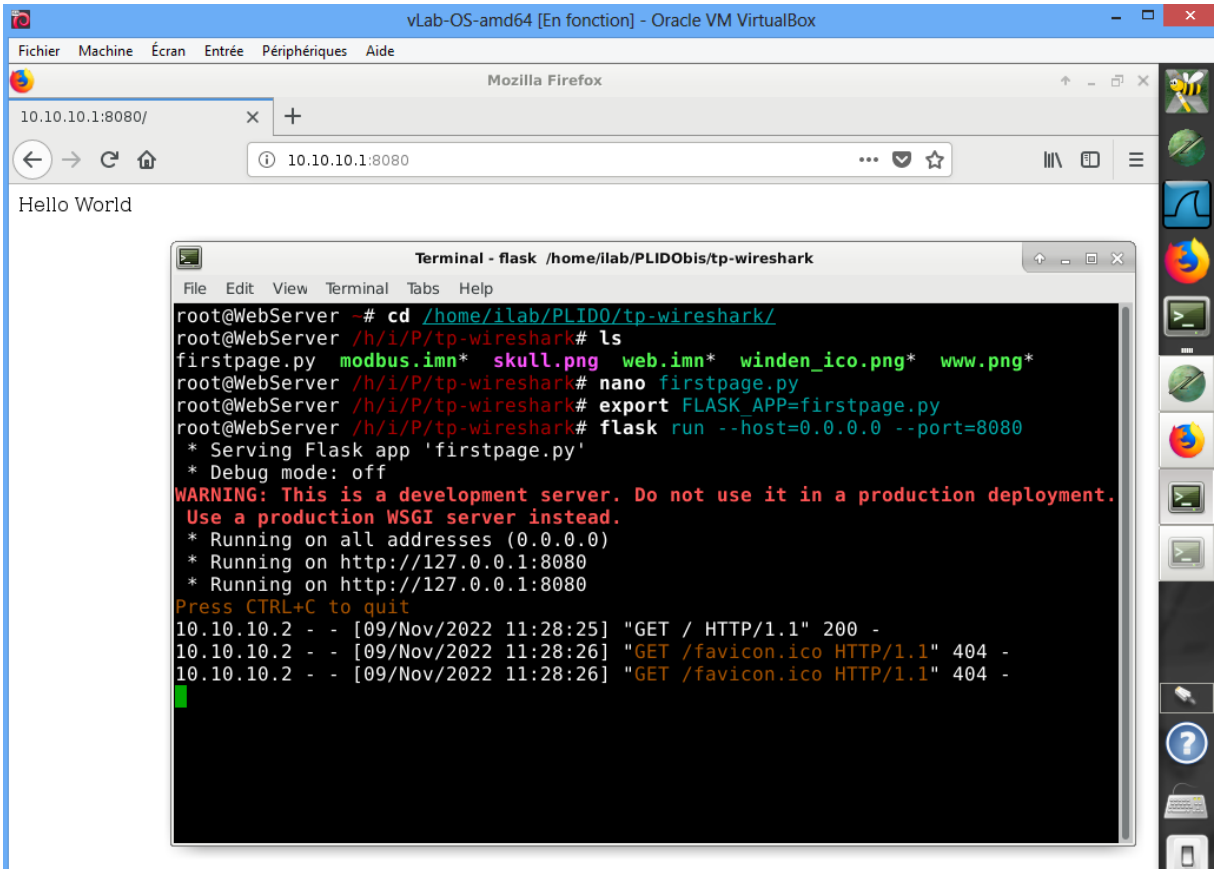
- Dans le Terminal, tapez ces commandes :

  $ **export FLASK_APP=firstpage.py**
  $ **flask run --host=0.0.0.0 --port=8080**

La première définit la variable FLASH_APP qui indique le fichier à utiliser par flask.

La seconde lance le serveur web flask ; host=0.0.0.0 indique au serveur d'accepter une connexion depuis n'importe quelle interface et dans notre cas depuis la machine WebClient.

- Dans le terminal de l'ordinateur, relancez Firefox.
- Allez chercher la ressource sur votre nouveau serveur web.



Question :

Instinctivement, on lit http://10.10.10.1:8080/ comme une URL, 10.10.10.1:8080 est l'adresse IPv4 du serveur et le numéro de port. Mais si on lit comme une URI, donnez : son schéma, son autorité, son chemin.

Fin.