Travaux Pratique n°6 :

Transfert d'une vidéo via le protocole UDP

Cédric Cognard
cedric.cognard@etu.univ-tours.fr

Paul Van-Elsue
paul.vanelsue@etu.univ-tours.fr

Dans ce TP nous allons streamer une vidéo via le protocole UDP à plusieurs client en utilisant des frame rates différents.

1. Préparation

Pour être en mesure de réaliser ce TP nous avons installé les librairies nécessaires sur nos PCs. Nous avons surtout dû mettre en place la connection Internet sous ArchLinux car le manager de connections ne s'en occupait pas automatiquement.
Nous avons utilisé la commande suivante :

```
sudo ip addr add 192.168.1.1/24 dev enp3s0
```

2. Serveur vidéo

Dans un premier temps nous avons mis en place un serveur simple.

Ce dernier initialise les variables qui lui seront utiles, la caméra puis se met en écoute sur le port qu'on lui indique.

Une fois l'initialisation effectuée il rentre dans une boucle infinie dans laquelle se déroulera sa routine. Cette routine consiste à prendre la dernière image capturée dans le pool d'attente, calculer sa taille et envoyer sa taille au client qui s'est connecté. Il est important que cette valeur soit connue du client car c'est ce qui lui permet par la suite de savoir quelle quantité de donnés attendre à la réception.

Ensuite c'est l'image elle même qui est envoyée, avec une petite conversion pour la rendre en noir et blanc et plus petite. Cette étape est très important pour réduire la taille de l'image et avoir des envois courts.
Si nous prenons trop de temps à envoyer nos images, un décalage entre la capture et l'affichage sur le client se créerait.

Voici le code qui en découle :

```
# Import the necessary packages.
from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2
import time
import socket
```

```
def int2bytes(n):
    result = bytearray()
    while n:
        result.append(n & 0xff)
        n = n >> 8
    return result[::-1]


# Initialize variables.
SERVER_ADDR = "192.168.1.253"
PORT = 5006
WIDTH = 320
HEIGHT = 180


# Initialize the camera and grab a reference to the raw camera capture.
camera = PiCamera()
camera.vflip = True
camera.hflip = True
camera.resolution = (WIDTH, HEIGHT)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(WIDTH, HEIGHT))
time.sleep(0.1)  # Allow the camera to warm up.


# Initialize the socket.
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Internet with UDP.
sock.bind((SERVER_ADDR, PORT))

print("Waiting on clients...")


# Capture frames from the camera.
while(True):
    for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
        # Grab the raw NumPy array representing the image, then initialize the timestamp and
occupied/unoccupied text.
        image = frame.array
        image.flags.writeable = True

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray = cv2.resize(gray, (WIDTH, HEIGHT), interpolation=cv2.INTER_LINEAR)

        # Send the image size to the client.
        data, address = sock.recvfrom(8)
        sock.sendto(int2bytes(gray.size), address)

        # Send the image to the client.
        sock.sendto(gray, address)
```

```
        # Clear the stream in preparation for the next frame.
        rawCapture.truncate(0)

        # If the 'q' key was pressed, break from the loop.
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

sock.close()
camera.close()
```

3. Client vidéo

Une fois le serveur fonctionnel il fallait un client pour récupérer et afficher les images reçues.

Comme pour le serveur, il initialise ses variables, se connecte puis rentre dans sa routine de fonctionnement. L'adresse et le port de connection sont données en paramètres au programme.

La routine consiste tout simplement à recevoir une taille d'image, recevoir un tableau de données de la taille de ce qui a été indiqué précédemment, le convertir en image et l'afficher.

Voici le code qui en découle :

```
# Import the necessary packages.
import matplotlib.pyplot as plt
import numpy as np
import cv2
import socket
import sys


# Initialize variables.
UDP_IP = str(sys.argv[1])
UDP_PORT = int(sys.argv[2])
WIDTH = 320
HEIGHT = 180


while(True):
    # Initialize the socket.
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Internet with UDP.
```

```
    # Receive the data.
    sock.sendto(bytes(0), (UDP_IP, UDP_PORT))

    # Receive the image size.
    img_size, server = sock.recvfrom(8)
    img_size = int.from_bytes(img_size, byteorder='big')
    print(img_size)

    # Receive the image.
    raw_img, server = sock.recvfrom(img_size)
    print(len(raw_img))

    # Convert the data in OpenCV img.
    im = np.frombuffer(raw_img, dtype='int8')  # Convert bytes to numpy format.
    im = np.uint8(im)  # Convert bytes in uint8.
    im = np.reshape(im, (HEIGHT, WIDTH))  # Convert 1D raw into matrix.

    # Print the image to the screen.
    cv2.imshow("Frame", im)

    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break

# Close the socket.
sock.close()
```

Nous avons tout de même rencontré un gros problème lors de la mise en place de ce système qui nous a fait perdre un temps considérable (au moins trois heures). En effet, nous avions déjà fait un système de client/serveur en C++ et pour l'envoie d'images, il fallait envoyer les données morceau par morceau selon les tailles de buffer qui elles mêmes dépendent des systèmes. C'est ce que nous avons essayé de faire ici, envoyer d'abord 4096 bits de données, puis les 4096 suivants, et ainsi de suite jusqu'à ce que l'image soit entièrement envoyée. Bien sûr cette taille de 4096 était variable et nous avons fait des tests avec plusieurs tailles de buffer différentes.

Cependant nous nous sommes complexifié la vie pour rien, il s'avère que lorsque l'on envoie une image en Python nous pouvons tout envoyer d'un coup. Si nous avions su plus tôt nous aurions pu gagner beaucoup de temps.

4. Multi-threading

Vient ensuite la partie du multi-threading. Le but cette fois est de pouvoir envoyer le streaming vidéo vers plusieurs clients, en ouvrant un thread par client sur le serveur.

Le premier problème que l'on rencontre avec ce système est que chaque thread cherche à capturer les images de la caméra et se bloquent les uns aux autres. Notre solution était de

faire un thread supplémentaire qui récupère l'image actuelle de la caméra et la met dans une variable globale. Les threads des clients eux n'ont qu'à récupérer cette image globale en lecture seule et l'envoyer aux clients.

Cependant nous avons rencontré plusieurs erreurs sur ce programme et nous n'avons pas réussi à le terminer. Nous avons choisi de passer à la suite pour ne pas perdre plus de temps que ce que nous avions déjà perdu précédemment.