TP6 : Envoie d'une vidéo via le protocole UDP

Objectif :

Etre en mesure d'envoyer une vidéo « live » via le protocole UDP à plusieurs clients et à plusieurs frame rate différents.

1°) Installation des outils logiciels

Sur votre machine fonctionnant sous Ubuntu 16, veuillez installer les éléments suivants :

L'installation peut prendre quelques minutes suivant la connexion réseau disponible. Pendant l'installation, commencez à lire la section 2.

A°) L'outil de développement python IDLE3 :

Taper la commande dans le terminal :

sudo apt-get install idle3

B°) Installer OpenCv pour voir les images sur votre machine :

Taper les commandes dans le terminal :

sudo apt install python3-pip

pip3 install opencv-python

2°) Les sockets sous Python

Le module socket :

```
import socket
```

Création de la socket coté serveur :

```
sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
```

Binding de la socket :
Cette étape permet de dire au système d'exploitation qu'un port est utilisé pour recevoir des données. Le binding est nécessaire pour recevoir de données.

```
sock.bind((UDP_IP, UDP_PORT))
```

Envoi de données :
```
sock.sendto(data, (Client_UDP_IP, UDP_PORT))
```

Réception de données : Il faut connaître le nombre d'octet à recevoir.

```
data, addr = sock.recvfrom(nbbyte)
```

Rappel : la quantité de données que l'on peut envoyer via une socket du maximum transmission unit (MTU). MTU est la taille maximale d'un paquet pouvant être transmis en une seule fois (sans fragmentation) sur une interface.
Concrètement si l'on veut envoyer une image en une fois sur de l'Ethernet, il faut que cette dernière soit au maximum d'une taille 320x180 en niveau de gris.

3°) Réalisation : Serveur Vidéo

Créer un serveur UDP sur votre Raspberry PI qui enverra le flux de la camera (PiCam) vers les 3 adresses et ports suivants :

Adresse : 192.168.1.252* port : 5006

Adresse : 192.168.1.252* port : 5007

Adresse : 192.168.1.252* port : 5008

*Remplacer cette adresse par l'adresse IP de votre machine.

Pour lire le flux de la camera vous vous appuierez sur le TP1.

Chaque image devra être convertie en niveau de gris et redimensionner à la taille de l'image en 320x180.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray=cv2.resize(gray,(width, height), interpolation = cv2.INTER_LINEAR)
```

4°) Réalisation : Client Vidéo

Créer un client UDP qui recevra image par image les images envoyées par le serveur vidéo.

Le port d'écoute du serveur devra être un paramètre de votre programme python.

```
UDP_PORT = int(sys.argv[1])
```

Pour convertir le tableau d'octet reçu par la socket (recvfrom) en image opencv vous pouvez procéder de la sorte :

```
im=np.frombuffer(data,dtype='int8') #conversion des octets au format numpy.
im=np.uint8(im) #conversion des octets en uint8
im=np.reshape(im,(height,width)) #conversion du tableau 1D en matrice
```

Une fois le programme codé, lancer 3 exécutions de votre programme avec les ports 5006, 5007, 5008

Vous pouvez aussi tester votre programme avec la RPI d'un autre camarade.

5°) Amélioration du serveur vidéo

Chaque client aura un thread dédié et l'émission d'une image se fera dans un thread différent pour chaque client.

Réaliser un programme capable de diffuser le flux de la caméra à un nombre de frames par seconde donné.

Pour ce faire vous pourrez soit utiliser les thread python.

Voici un exemple d'utilisation des thread en python :

```
import numpy
import cv2
from threading import *
import time

class videoprocess(Thread):

    """Thread chargé simplement d'afficher une lettre dans la console."""

    def __init__(self ,fps,id,ipclient,portclient):
        Thread.__init__(self)
       self.image=np.zeros((20,20,3),np.uint8)
        self.fps = fps
        self.id=id
        print('thread=live at fps='+str(fps))
     def setimage(self,image):
          self.image = image

    def run(self):
        """Code à exécuter pendant l'exécution du thread."""
        print('run thread='+self.id) #ne pas décommenter cette ligne
         attente = 1/float(self.fps)
         time.sleep(attente)
```

```
# Création des threads
thread_1 = videoprocess(image,1,1, « 192.168.1.22 »,5006)

# Lancement des threads
thread_1.start()
time.sleep(1) #warmup du thread
# Attend que les threads se terminent
thread_1.join()

cv2.destroyAllWindows()
```

A faire :

Associer un thread à chaque couple @ipclient et portclient.

Le nombre de frame par seconde sera choisi au lancement du thread (exemple : 20 images par seconde).

Afficher le nombre de frame par seconde sur l'image.

Visualiser le résultat sur les clients.

Pour un client (1 thread) quelle est la valeur max du nombre de frame par seconde ?

Faites varier le nombre de clients et ? Quel est l'impact sur le nombre de frame par seconde ?