IMAGE STREAMING USING A REALTIME PROTOCOL

A MINI PROJECT REPORT

By

AJAY KUMAR (RA2111030010084)
NIVETHA G (RA2111030010112)
SUDIKSHAN S (RA2111030010116)
VIIGNESH S (RA2111030010110)
ESWAR A (RA2111030010086)
SMARAN V (RA211103001067)

Under the guidance of

D.Saveetha

In partial fulfilment for the Course

of

18CSC302J-COMPUTER NETWORKS

FACULTY OF ENGINEERING AND TECHNOLOGY,
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
Kattankulathur, Chenpalpattu District

OCTOBER 2023

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report about **Image Streaming Using Real Time Protocol** is the bonafide work of **AJAY KUMAR (RA2111030010084), NIVETHA G (RA2111030010112), SUDIKSHAN S (RA2111030010116), VIIGNESH S (RA2111030010110), ESWAR A (RA2111030010086) & SMARAN V (RA211103001067)** whom carried out the project work under my supervision.

SIGNATURE

Saveetha D,

Assistant Professor,

NWC Department,

SRM Institute of Science and Technology.

TABLE OF CONTENTS

(CHAPTER 1)

Abstract

Image streaming has gained significant attention in recent years due to its diverse applications in fields such as surveillance, remote sensing, healthcare, and entertainment. This paper introduces the concept of image streaming using real-time protocols, focusing on the efficient and seamless transmission of visual data over networks. Real-time protocols play a critical role in ensuring the timely and accurate delivery of images, enabling applications that demand instant access to visual information.

This paper begins by presenting an overview of image streaming and its significance in various domains. It then delves into the characteristics and challenges associated with real-time protocols for image streaming. The discussion includes considerations for low latency, high throughput, and reliable delivery, which are paramount for maintaining the integrity of the streamed images. Various existing real-time protocols, such as Real-time Transport Protocol (RTP) and Web Real-Time Communication (WebRTC), are analyzed in terms of their suitability for image streaming applications.

(CHAPTER 2)

Introduction

This project aims at designing and assessing the functioning of an enhanced protocol for image transmission over loss-prone crowded or wireless networks. The usual approach to transporting images uses TCP. The main motivation to undertake this project starts with stressing on the disadvantages of using TCP. The main drawback of using TCP for image downloads is that its in-order delivery model interferes with interactivity. TCP provides a general reliable, in-order byte- stream abstraction, but which is excessively restrictive for image data. TCP is not suitable for real-time applications as the retransmissions can lead to high delay and cause delay jitter, which significantly degrades the quality.

On the other hand , UDP is a connectionless protocol and is not dedicated to end-to- end communications, nor does it check the readiness of the receiver (requiring fewer overheads and taking up less space). Hence, UDP is a much faster, simpler, and efficient protocol for streaming processes, but it does come with a concerning drawback, prone to packet losses. Packet losses can lead to corrupt images, which is something we want to avoid. Therefore this project is a step taken at tackling that very issue. Although it's built on UDP, it also uses a small programmatically written optimizer that can also be considered as a protocol running on top of UDP. Hence, enhancing and optimizing its features according to our objective. This project enables high quality image streaming without exhausting the buffer , therefore giving us a higher chance of receiving images without corrupting them.

To validate the effectiveness of real-time protocols for image streaming, the paper presents experimental results that measure latency, image quality, and overall performance. A comparison of different protocols under varying network conditions provides insights into their real-world applicability.

(CHAPTER 3)

Requirement Analysis

The following are the requirements for the chat application:

Environment Software : Anaconda

IDE: Jupyter Notebook

Language : Python

Libraries:

- Imported socket for socket programming eg: creation of sockets, etc.
- Imported PIL that is PYTHON IMAGE LIBRARY for opening and closing images via path
- Imported tkinter for making the client login, popup and server login GUI.
- Imported sys for accessing system specific functions.
- Imported cv2, the OpenCV library which in our project is used for handling the images such as encoding, decoding it and streaming it through a netwrok.
- Imported numpy for creating multi-dimensional arrays
- Imported struct for packing data into structures.

(CHAPTER 4)

Architecture & Design

The chat application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The server is responsible for managing the communication between the clients. We have used Python's Tkinter module to create the GUI and socket programming and threading to implement the network communication.

The GUI of the application consists of two main parts:

Chat window: This window displays the messages sent and received by the user.

Input box: This box allows the user to type and send messages to other users.

The server creates a new thread for each client that connects to it. This thread handles the communication between the server and the client. When a client sends a message, the server receives the message and broadcasts it to all other connected clients.

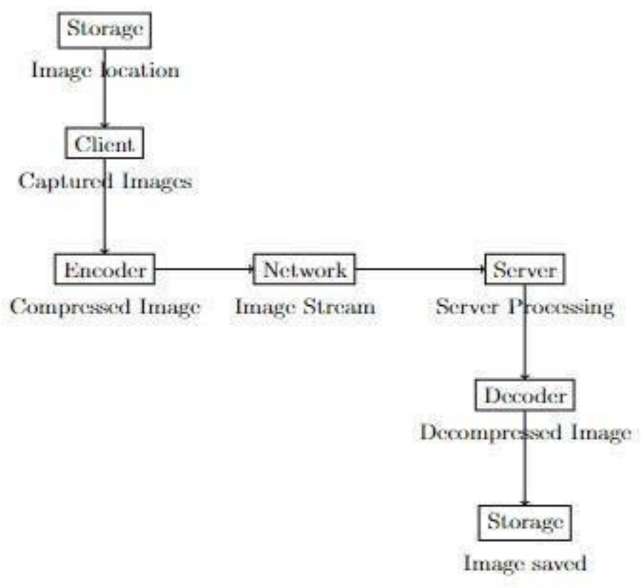


Figure 1: Image Streaming System Architecture

(CHAPTER 5)

Implementation

There's Two GUIs for Client and Server Login. Once the Client and Server is successfully logged in, the server starts listening and the client accepts the connection respectively. The client then asks for the user's input for an Image through a file dialog. Once the image is received the client prints out the location path and the image type. Then the image is compressed and converted into a numpy array using the Open CV Library. The numpy array is a three-dimensional array in height, width and channel format.

The client starts streaming the data in batch of size 1024 bytes (The buffer size) it also at the end of the transmission checks for any remaining data less than 1024 Bytes and if found streams them as well. The server prints out each stream it receives with the particular size of data received along with the total amount of data received till then. The server also outputs in the end an confirmation message, along with the total streams received, the amount of lost packets, loss percentage and Success percentage. The Client and Server Primarily run on UDP but since we have defined how the data should be transferred this can also be considered as a small protocol that sits on top of UDP.

By Using UDP for streaming rather than TCP/IP as it supports 3 Way handshake which could lead to long idle times while also writing a optimization in the streaming process to stream images in a cyclic way instead of directly exhausting the buffer by trying to transfer it in a single go, Our project has a higher rate of success for image streaming and transferring

(CHAPTER 6)

Code

Server code:

```
{
"cells": [
 {
  "cell_type":
"code",

"execution_count":
null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [
   {
    "name":
"stdout",
    "output_type":
"stream",
    "text": [
      "Server  is listening for
  connections...\n",
      "Connected   to
  ('127.0.0.1',
  64434)\n",
      "Image     received
  and saved  as
  C:\\Users\\ajayc\\D
  esktop\\Image
 streaming        Mini
  Project\\received_i
  mage_127.0.0.1.jpg
  \n"
     ]
    }
   ],

   "source": [
    "import
  socket\n",
    "import os\n",
    "\n",
    "def
  save_image(image_
  data,
  save_path):\n",
    "                with
```

```
open(save_path,
'wb')                    as
image_file:\n",
   "
image_file.write(i
m age_data)\n",
   "\n",
   "server_address =
('127.0.0.1',
12345)\n",
   "save_directory =
r'C:\\Users\\ajayc\
\   Desktop\\Image
streaming      Mini
Project'    #    Your
specified
directory\n",
   "\n",
   "server_socket                    =
socket.socket(socke
t.AF_INET,
socket.SOCK_STR
EAM)\n",


"server_socket.bind
(server_address)\n"
```

```
,
"server_socket.liste
n(1)\n",
   "\n",
   "print(\"Server
is  listening  for
connections...\")\n
",
   "\n",
   "while True:\n",
   "  client_socket,
client_address   =
server_socket.acc
ep t()\n",
   "
print(f\"Connected
to
{client_address}\")
\ n",
   "\n",
   "  image_data  =
b"

#  Initialize  an
empty byte string
```

```
to hold the image
data\n",


   "              while
True:\n",
   "    chunk = client_socket.recv(
1024)\n",
   "              if not
chunk:\n",
   "          break\n",
   "      image_data
+= chunk\n",
   "\n",
   "    filename =
f\"received_image_
{client_address[0]}
.jpg\"      #      Use
client's IP address
in the filename\n",
   "     save_path =
os.path.join(save_d
irectory,
filename)\n",
   "
save_image(image_
data, save_path)\n",
   "\n",
   "   print(\"Image
received and saved
as\",
save_path)\n",
   "\n",
   "
client_socket.close(
)\n",
   "\n",

"server_socket.clos
e()\n"
  ]
  },
  {
  "cell_type":
"code",


"execution_count":
null,
  "metadata": {},
  "outputs": [],
  "source": []

  }
],
 "metadata": {
 "kernelspec": {
 "display_name":
"Python                3
 (ipykernel)",
   "language":
 "python",
   "name":
 "python3"

  },
  "language_info": {

 "codemirror_mode"
 : {

    "name":
 "ipython",
    "version": 3

   },
   "file_extension":
 ".py",
   "mimetype":
 "text/x-python",
   "name": "python",
"nbconvert_exporte
r": "python",

"pygments_lexer":
"ipython3",
  "version":
"3.9.13"
 }
},

"nbformat": 4,
"nbformat_minor":
4
}
```

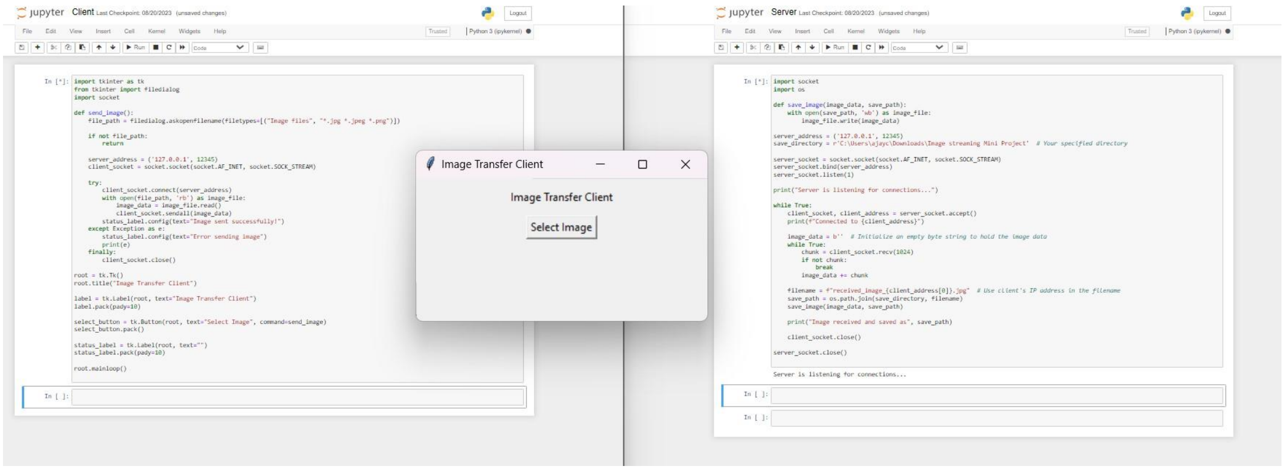
Client code:

```
{
"cells": [
 {
  "cell_type": "code", "execution_count": null,
  "metadata": {
  "scrolled": false
  },

  "outputs": [],
  "source": [
   "import tkinter as tk\n",
"from tkinter import filedialog\n",
"import socket\n",
"\n",
"def send_image():\n",
" file_path = filedialog.askopenfilename(filetypes=[(\"Image files\", \"*.jpg *.jpeg
*.png\")])\n",
"   \n",
" if not file_path:\n",
"  return\n",
"   \n",
" server_address = ('127.0.0.1', 12345)\n",
" client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)\n",
" \n",
" try:\n",
"    client_socket.connect(server_address)\n",
"    with open(file_path, 'rb') as image_file:\n",
"        image_data = image_file.read()\n",
"        client_socket.sendall(image_data)\n",
"  status_label.config(text=\"Image sent successfully!\")\n",


" except Exception as e:\n",
"    status_label.config(text=\"Error sending image\")\n",
"      print(e)\n",
" finally:\n",
"     client_socket.close()\n",
"\n",
"root = tk.Tk()\n",
"root.title(\"Image Transfer Client\")\n",
"\n",
   "label = tk.Label(root, text=\"Image Transfer Client\")\n",
   "label.pack(pady=10)\n",
   "\n",
   "select_button = tk.Button(root, text=\"Select Image\", command=send_image)\n",
   "select_button.pack()\n",
   "\n",
   "status_label = tk.Label(root, text=\"\")\n",
   "status_label.pack(pady=10)\n",
   "\n",
   "root.mainloop()\n"
  ]
 },

 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": []
 },

 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": []
 }
```

```
],
"metadata": {
 "kernelspec": {
  "display_name": "Python 3 (ipykernel)",
  "language": "python",
  "name": "python3"
 },

 "language_info": {
  "codemirror_mode": {
  "name": "ipython",
  "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.9.13"
 }
},

"nbformat": 4,
"nbformat_minor": 4
}
```

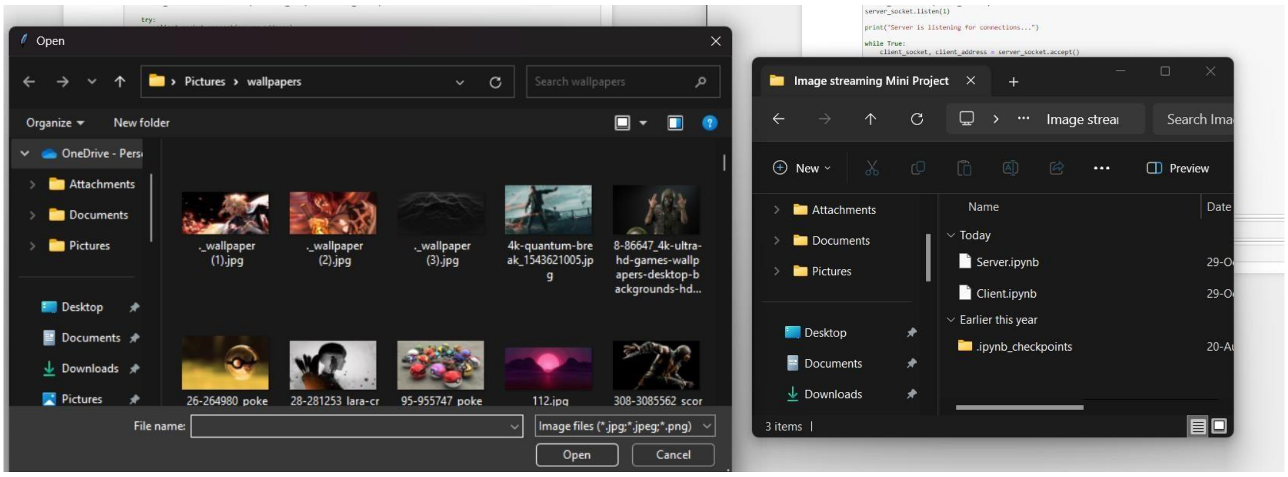
(CHAPTER 7)

Experiment Result & Outputs


jupyter Client Last Checkpoint: 08/20/2023 (unsaved changes)
Image Transfer Client
Select Image
jupyter Server Last Checkpoint: 08/20/2023 (unsaved changes)

```
In [*]: import tkinter as tk
        from tkinter import filedialog
        import socket

        def send_image():
            file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg *.jpeg *.png")])

            if not file_path:
                return

            server_address = ('127.0.0.1', 12345)
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

            try:
                client_socket.connect(server_address)
                with open(file_path, 'rb') as image_file:
                    image_data = image_file.read()
                    client_socket.sendall(image_data)
                status_label.config(text="Image sent successfully!")
            except Exception as e:
                status_label.config(text="Error sending image")
                print(e)
            finally:
                client_socket.close()

        root = tk.Tk()
        root.title("Image Transfer Client")

        label = tk.Label(root, text="Image Transfer Client")
        label.pack(pady=10)

        select_button = tk.Button(root, text="Select Image", command=send_image)
        select_button.pack()

        status_label = tk.Label(root, text="")
        status_label.pack(pady=10)

        root.mainloop()
```

```
In [*]: import socket
        import os

        def save_image(image_data, save_path):
            with open(save_path, 'wb') as image_file:
                image_file.write(image_data)

        server_address = ('127.0.0.1', 12345)
        save_directory = r'C:\Users\ajayc\Downloads\Image streaming Mini Project'  # Your specified directory

        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind(server_address)
        server_socket.listen(1)

        print("Server is listening for connections...")

        while True:
            client_socket, client_address = server_socket.accept()
            print(f"Connected to {client_address}")

            image_data = b''  # Initialize an empty byte string to hold the image data
            while True:
                chunk = client_socket.recv(1024)
                if not chunk:
                    break
                image_data += chunk

            filename = f"received_image_{client_address[0]}.jpg"  # Use client's IP address in the filename
            save_path = os.path.join(save_directory, filename)
            save_image(image_data, save_path)

            print("Image received and saved as", save_path)

            client_socket.close()

        server_socket.close()

        Server is listening for connections...
```



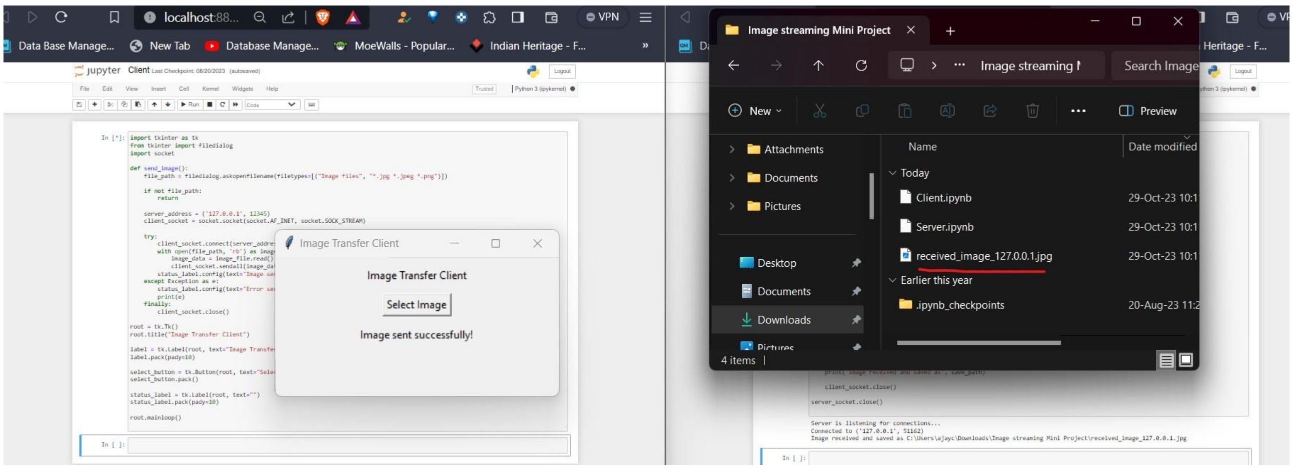


Image sent successfully!

```
Server is listening for connections...
Connected to ('127.0.0.1', 51162)
Image received and saved as C:\Users\ajayc\Downloads\Image streaming Mini Project\received_image_127.0.0.1.jpg
```

(CHAPTER 8)

Conclusion & Future Enhancement

In this project, we have successfully developed a image streaming application using Python's Tkinter module. The application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The application has a graphical user interface (GUI) that allows users to send and receive messages. We have used socket programming and threading to implement the network communication between the server and clients. The application has been tested and is functional for different types of messages and multiple clients simultaneously.

In the future, we can enhance the chat application in the following ways:

- Add support for video and voice calls.
- Add the ability to create chat rooms.
- Implement end-to-end encryption to ensure the security of user data.
- Add the ability to send and receive notifications.
- Add the ability to delete messages.
- Implement a search feature to search for messages or users.

References

"Python Socket Programming Tutorial."-realpython.com/python-sockets/

"Python Tkinter Tutorial - Python GUI Programming." -realpython.com/python-gui-tkinter/

"Python Threading Tutorial: Run Code Concurrently Using the Threading Module."realpython.com/intro-to-python-threading/

"Creating a Chat Application Using Python and Flask." -codeburst.io/creating-achatapplication- using-python-and-flask-7a8f547bcc8f