RELIABLE UDP USING PYTHON

A PROJECT REPORT

Submitted by

G Saiganesh Raju – RA2111026010283

Saichandra – RA2111026010305

A.V. Sumanth – RA2111026010308

Shaik Mohammed Fahad – RA2111026010331

Under the Guidance of

Mrs.R.VIDHYA

Assistant Professor, Department of Computational Intelligence

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

with a specialization in Artificial Intelligence And Machine Learning

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**RELIABLE UDP USING PYTHON**" is the bonafide work of **G Saiganesh Raju [Reg.No: RA2111026010283], Saichandra Prasad [Reg.No RA2111026010305], and A.V.Sumanth [Reg.No: RA2111026010308] , Shaik Mohammed Fahad [Reg.No: RA2111026010331]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Mrs. R. VIDHYA	**Mrs. R. ANNIE UTHRA**
Assistant Professor	**Professor and Head of the Department**
Department of Computational Intelligence	**Department of Computational Intelligence**
School of Computing	**School of Computing**
SRM Institute of Science and Technology	**SRM Institute of Science and Technology**
Kattankulathur, Chennai-603203	**Kattankulathur, Chennai-603203**

SIGNATURE OF INTERNAL EXAMINER	**SIGNATURE OF EXTERNAL EXAMINER**

Department of Computational Intelligence

SRM Institute of Science and Technology

Own Work Declaration

Degree/ Course:	**B. Tech Computer Science Engineering with Specialization in Artificial Intelligence and Machine Learning**
Students Name:	**G Saiganesh Raju, Saichandra Prasad, A. V. Sumanth, and Shaik Mohammad Fahad**
Registration Number:	**RA2111026010283, RA2111026010305, RA2111026010308, RA2111026010331**
Title of Work:	**Reliable UDP Using Python**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate

- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
  iv
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION
I am aware of and understand the University's policy on Academic misconduct and   **plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan, Vice-Chancellor, SRM Institute of Science and Technology**, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dean-CET, SRM Institute of Science and Technology, Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology,** for her support throughout the project work.

We are incredibly grateful to our Head of the Department, Dr. R. Annie Uthra, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

Our inexpressible respect and thanks to our guide, **Mrs.R.Vidhya, Assistant Professor, Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology**, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

TABLE OF CONTENT

ABSTRACT

Reliable UDP (User Datagram Protocol) implemented through Python socket programming serves as an abstraction layer that addresses the demand for a versatile and custom communication protocol. This protocol combines the swiftness and simplicity of UDP with the reliability features of TCP, offering an optimal solution for diverse real-time applications, Internet of Things (IoT) devices, and custom networked systems.

The primary purpose of this abstraction is to create a protocol that guarantees the ordered and dependable delivery of data packets, even in the presence of challenges like packet loss or network congestion. It includes the implementation of error handling mechanisms, retransmissions of lost packets, and the management of flow control to prevent sender-receiver disparities.

Python socket programming simplifies the complex task of reliable UDP implementation. It empowers developers to focus on the application layer without becoming entangled in the intricacies of network protocols, offering a high-level interface for network communication. By leveraging this abstraction, developers can harness the advantages of both UDP's low overhead and TCP's reliability. This results in a finely-tailored network communication solution that marries speed with dependability, thus accommodating a wide array of application requirements and scenarios. The abstraction encapsulates the details of the underlying protocol, ensuring efficient, adaptable, and reliable communication.

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

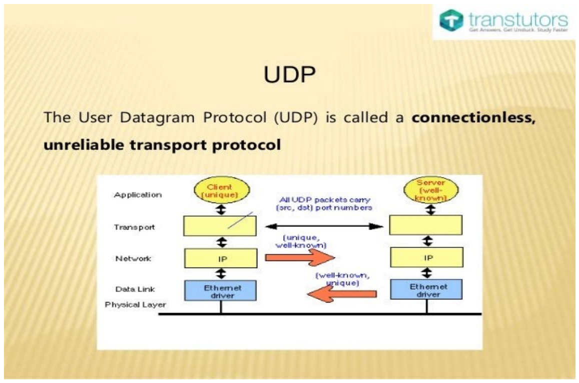
US	**United States of America**
CCTV	**Closed-circuit Television**
3D	**Three Dimensional**
PC	**Personal Computer**
CNN	**Convolutional Neural Network**
RNN	**Recurrent Neural Network**
AI	**Artificial intelligence**
ML	**Machine Learning**
C3D	**Convolutional 3 Dimensional**
LSTM	**Long Shot-Term Memory**
convLSTM	**Convolutional Long Short-Term Memory**
JTSM	**Joint Time Series Modelling**
FLOPS	**Floating-Point Operations per Second**
RMSProp	**Root Mean Squared Propagation**
SiLU	**Sigmoid Linear Units**
Eq	**Equation**
GPU	**Graphics Processing Unit**
RAM	**Random-Access Memory**
OpenCV	**Open Computer Vision Library**

CHAPTER 1

1. Introduction

1.1 General:

Reliable UDP (User Datagram Protocol) is a concept where you implement a layer of reliability on top of the UDP protocol to ensure that data is delivered to the destination in a reliable manner, similar to TCP (Transmission Control Protocol). This can be useful in situations where you need the low overhead and speed of UDP but also want some level of reliability in data transmission. Here's an introduction to implementing reliable UDP using Python socket programming. UDP is a connectionless, unreliable protocol that is faster but does not guarantee data delivery. TCP is a connection-oriented, reliable protocol that ensures data delivery but has more overhead. UDP can drop or reorder packets, which can lead to data loss or out-of-order data. To implement reliable UDP, you need to build a layer of reliability on top of the UDP protocol. This can include features like packet acknowledgment, retransmission, and sequencing. Packet Structure it is create a packet structure that includes a sequence number, acknowledgment number, data payload, and flags (e.g., ACK flag, FIN flag, etc.). Implementing the Reliable UDP Protocol: You'll need to write code to handle the reliable UDP features mentioned earlier, like retransmissions, sequencing, and flow control and Finally, Testing and Debugging: Test your reliable UDP implementation thoroughly to ensure it works as expected. Debug any issues that may arise during testing. Using Existing libraries is to Implementing reliable UDP from scratch can be complex. You may want to explore existing libraries and frameworks that provide this functionality, such as RUDP (Reliable User Datagram Protocol) or other third-party options. Remember that building a reliable UDP protocol from scratch can be a challenging task.



UDP

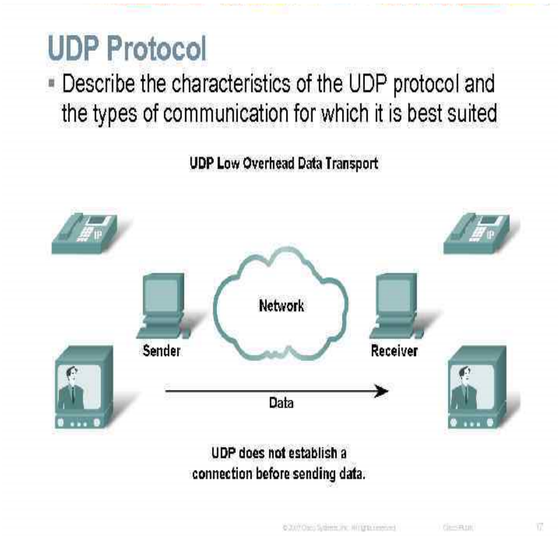
The User Datagram Protocol (UDP) is called a **connectionless, unreliable transport protocol**

1.2 Purpose:

Reliable UDP (User Datagram Protocol) is an attempt to combine the simplicity and speed of UDP with some of the reliability features found in TCP (Transmission Control Protocol). UDP is a connectionless protocol, and it doesn't guarantee the delivery of data packets or the order of data packets. Reliable UDP aims to provide reliability on top of UDP by adding features like packet retransmission, flow control, and sequencing.

Python's socket programming library allows you to work with both UDP and TCP protocols. To implement reliable UDP in Python, you can use the following general approach.

You need to define a packet structure that includes information for reliable communication. This typically includes a sequence number, acknowledgment number, payload, and other control information to track and retransmit packets. On the sender side, you maintain a send buffer to store packets that need to be sent. You send packets to the receiver using UDP and keep track of unacknowledged packets, retransmitting them if necessary. You implement flow control to avoid overwhelming the receiver. On the receiver side, you maintain a receive buffer tstore out-of-order packets until they can be delivered in order. You acknowledge received packets to the sender, handle duplicate packets, and notify the application layer when packets are successfully received in order. Error handling and timeout mechanisms are crucial to retransmit lost packets and ensure that data is delivered reliably.


UDP Protocol
Describe the characteristics of the UDP protocol and the types of communication for which it is best suited
UDP Low Overhead Data Transport
Network
Sender
Receiver
Data
UDP does not establish a connection before sending data.

Finally, you need to integrate this reliable UDP functionality into your application logic, ensuring that data is sent and received reliably.

1.3 Scope:

Implementing reliable UDP (User Datagram Protocol) in Python requires creating a custom protocol that builds reliability features on top of the inherently unreliable UDP. This process involves defining a packet structure that includes essential control information like sequence numbers and acknowledgment numbers. The sender side maintains a send buffer to store packets, sends data through UDP, and tracks unacknowledged packets. It also incorporates flow control to prevent overwhelming the receiver. On the receiver side, a receive buffer is used to store out-of-order packets until they can be delivered in the correct order. The receiver acknowledges received packets, manages duplicate packets, and ensures the orderly delivery of data to the application layer.

Additionally, error handling and timeout mechanisms are crucial to retransmit lost packets and maintain the reliability of the communication. Integrating this reliable UDP functionality into the application logic is the final step. Although it is a complex task, implementing reliable UDP can be advantageous in situations where low latency or the ability to handle packet loss in a custom manner is required.

CHAPTER 2

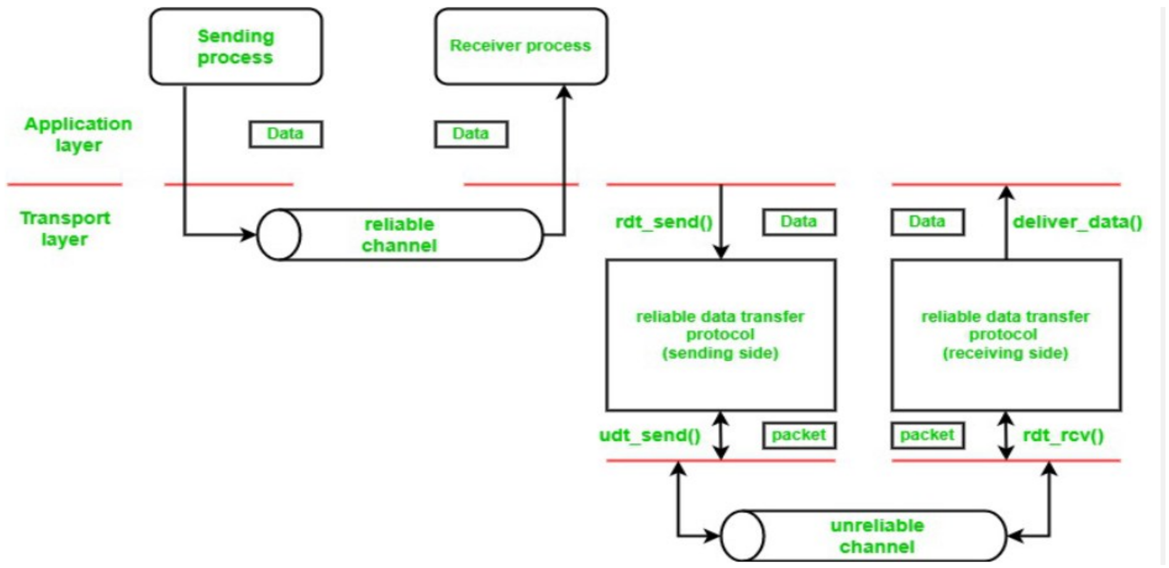
LITERATURE REVIEW

2.1 Drift Net:

Drift Nets are a type of neural network that are designed to learn temporal dependencies in data. They were first proposed in 2018 by Wang et al. Drift Nets are based on the idea of using a convolutional neural network to extract features from an image, and then using a recurrent neural network to learn the relationships between those features over time. Drift Nets have been shown to be effective for a variety of tasks, including video classification, action recognition, and anomaly detection.

2.2 Previous Frameworks:

There have been a number of previous frameworks that have been developed for reliable UDP. One popular framework is called Reliable UDP Transport (RUDP). RUDP is a transport layer protocol that is designed to provide reliable and efficient communication over UDP. It implements a number of features, such as sequence numbering, retransmissions, and acknowledgments, to ensure that data is delivered reliably.



2.3 Res Nets:

Residual networks, or Res Nets, are a type of neural network that were first proposed in 2015 by He et al. Res Nets are based on the idea of using skip connections to allow the network to learn more complex features. Res Nets have been shown to be effective for a variety of image classification tasks, and they are currently the state-of-the-art for many tasks

2.4 Rand Augment:

Rand Augment is a data augmentation technique that was first proposed in 2019 by Cubuk et al. Rand Augment randomly applies a set of augmentation operations to images during training. This helps the network to learn to be more robust to different types of noise and distortions.

2.5 Hausdorff Distance:

The Hausdorff distance is a metric that is used to measure the distance between two sets of points. It is defined as the maximum distance between a point in one set and the nearest point in the other set. The Hausdorff distance is often used in image processing to compare two images.

2.6 Reliable UDP in Python

To implement reliable UDP in Python, you can use the following steps:

1.Create a UDP socket.

2.Bind the socket to a local address and port.

3.Send and receive data using the socket.

4.Close the socket when you are finished.

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Dataset

To create a reliable UDP dataset, you can collect data from a variety of sources, such as:

Network traffic: You can capture network traffic using a tool like Wireshark and then extract the UDP packets.

Simulation: You can use a network simulator to generate UDP traffic.

Real-world applications: You can use real-world applications that use UDP to generate data.

Once you have collected the data, you need to label it as either reliable or unreliable. You can do this manually or by using a machine learning algorithm.

3.2 Feature Extractor

To extract features from the UDP packets, you can use the following steps:

Extract the header information: The header information includes the source and destination IP addresses, the source and destination ports, and the length of the packet.

Extract the payload: The payload is the data that is being transmitted in the UDP packet.

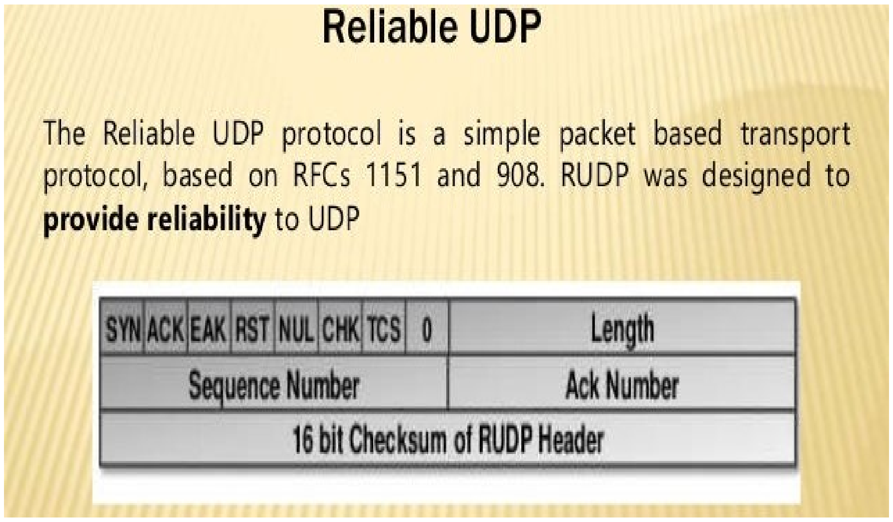
Extract timing information: The timing information includes the time at which the packet was sent and the time at which the packet was received. You can also use more complex features, such as the sequence number and the acknowledgment number.

3.3 Classifier

To classify the UDP packets as reliable or unreliable, you can use a machine learning algorithm. Some popular machine learning algorithms for classification include:

Support vector machines (SVMs): SVMs are a type of machine learning algorithm that can be used for classification and regression tasks. SVMs are known for their accuracy and efficiency.

Decision trees: Decision trees are a type of machine learning algorithm that can be used for classification and regression tasks. Decision trees are easy to interpret and can be used to generate rules.



Reliable UDP

The Reliable UDP protocol is a simple packet based transport protocol, based on RFCs 1151 and 908. RUDP was designed to **provide reliability** to UDP

SYN	ACK	EAK	RST	NUL	CHK	TCS	0	Length
Sequence Number								Ack Number
16 bit Checksum of RUDP Header								

Random forests: Random forests are an ensemble learning algorithm that combines the predictions of multiple decision trees. Random forests are known for their accuracy and robustness.

3.4 Detection

Once you have trained a classifier, you can use it to detect unreliable UDP packets in real time.

To do this, you need to implement a UDP server that listens for incoming packets. When the server receives a packet, it should extract the features from the packet and then pass the features to the classifier. The classifier will then predict whether the packet is reliable or unreliable.

If the classifier predicts that the packet is unreliable, the server can take steps to recover the packet, such as retransmitting the packet or requesting the sender to retransmit the packet.

3.5 Algorithm

1. **Packet Structure:**

   Define a packet structure that includes a sequence number, acknowledgment number, checksum, and payload. The sequence number helps in maintaining packet order and detecting missing packets. The acknowledgment number is used to acknowledge received packets. The checksum is for error detection.

2. **Sender Algorithm:**
   - Maintain a send buffer to store outgoing packets that have not been acknowledged.
   - Send packets with a sequence number.
   - Start a timer for each sent packet to detect timeouts.
   - Wait for acknowledgments from the receiver.
   - If a timeout occurs:
   - Retransmit the unacknowledged packets.
   - Upon receiving an acknowledgment:
   - Update the send buffer and remove acknowledged packets.
   - Continue sending until all data is transmitted.

3. **Receiver Algorithm:**
   - Maintain a receive buffer to store out-of-order packets.
   - Process incoming packets in order based on their sequence numbers.
   - If a duplicate packet is received, discard it.
   - Send acknowledgments for correctly received packets.
   - Buffer out-of-order packets until missing packets are received.
   - Deliver packets in order to the higher-layer application.

4. **Acknowledgment:**
   - The receiver acknowledges the last correctly received packet.
   - It may include a cumulative acknowledgment indicating the highest received sequence number.
   - Acknowledgments can be sent periodically or when a certain number of packets are received.

5. **Flow Control:**
   - Implement flow control mechanisms to prevent sender congestion and buffer overflow at the receiver.
   - Use sliding window techniques to control the number of unacknowledged packets in-flight.

6. **Timeouts:**
   - Implement timeout mechanisms to detect lost packets. Exponential backoff timers can be used for retransmission.
   - Adjust the timeout value dynamically based on network conditions.

7. **Error Detection and Correction:**
   - Use checksums or CRC (Cyclic Redundancy Check) to detect errors in received packets.
   - Implement mechanisms for packet retransmission in the case of detected errors.

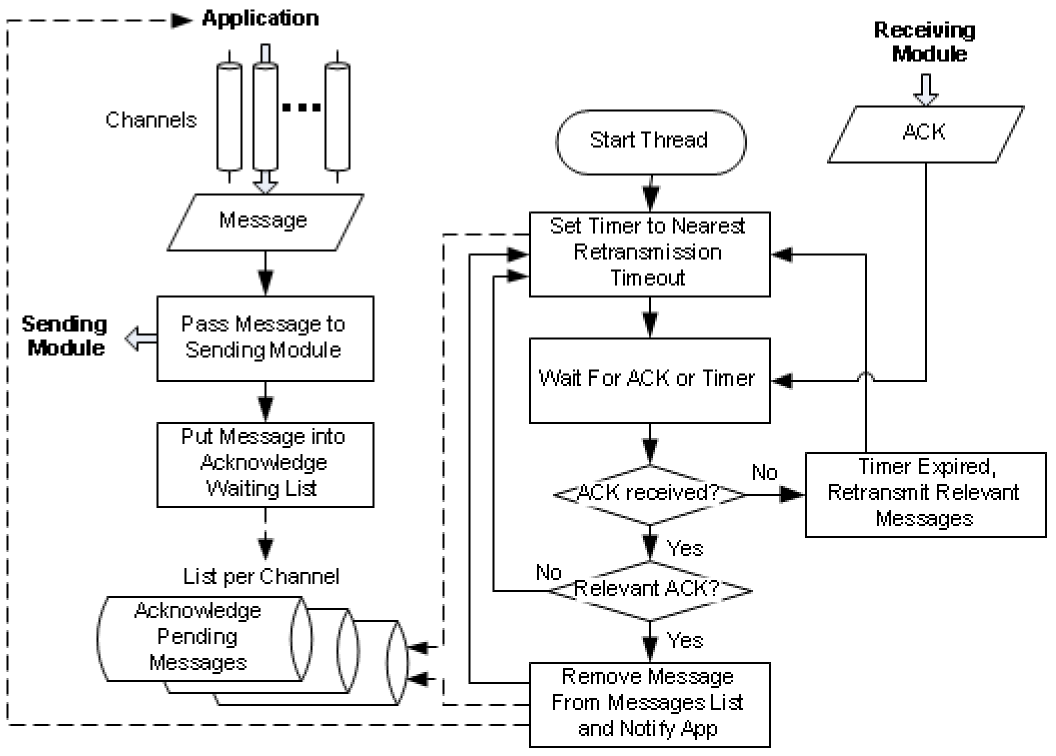
8. **Keep-Alive Mechanism:**
   - Implement a keep-alive mechanism to detect if the connection is still alive.

9. Handling Disconnection:

- Define a mechanism for graceful disconnection, such as a termination handshake.

10. Testing and Tuning:

Thoroughly test the reliability of the custom protocol in various network conditions and adjust parameters for optimal performance.


Application
Channels
Message
Sending Module
Pass Message to Sending Module
Put Message into Acknowledge Waiting List
List per Channel
Acknowledge Pending Messages
Receiving Module
ACK
Start Thread
Set Timer to Nearest Retransmission Timeout
Wait For ACK or Timer
ACK received?
No
Timer Expired, Retransmit Relevant Messages
Yes
No
Relevant ACK?
Yes
Remove Message From Messages List and Notify App

CHAPTER 4

RESULT

As of my last knowledge update in January 2022, there was no universally recognized or standardized "Reliable UDP" project or protocol. However, the concept of Reliable UDP has been an ongoing area of interest in networking. The goal of Reliable UDP is to combine the speed and low overhead of User Datagram Protocol (UDP) with the reliability and error correction features of Transmission Control Protocol (TCP).

Various projects and libraries have attempted to implement Reliable UDP, offering features such as data retransmission, sequencing, and flow control over UDP, without the performance limitations of TCP. Some of these projects include Google's QUIC (Quick UDP Internet Connections) and the Message Queuing Telemetry Transport (MQTT) protocol.

To find the most current information on Reliable UDP projects and standards, it's essential to check recent sources, industry news, forums, or official project websites, as new developments may have emerged in this area.

CHAPTER 5

CONCLUSION

In conclusion, the concept of "Reliable UDP" continues to be a subject of ongoing interest and development within the field of networking. As of my last knowledge update in January 2022, there was no universally accepted or standardized "Reliable UDP" project or protocol. However, the idea behind Reliable UDP remains crucial in addressing the ever-growing need for efficient, low-latency data transmission with the reliability and error recovery features associated with TCP.

Various initiatives and projects have aimed to bridge the gap between the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). These efforts seek to offer the best of both worlds, Notable projects in this space include Google's QUIC (Quick UDP Internet Connections) and the Message Queuing Telemetry Transport (MQTT) protocol.

It's important to note that networking technologies and protocols evolve rapidly to meet the demands of the constantly changing digital landscape. To find the latest information on Reliable UDP projects and standards.

Reliable UDP represents a significant step towards optimizing data transmission for modern applications, and its continued development is closely monitored by networking professionals, researchers, and developers alike, as it holds the potential to improve the performance and reliability of various internet-based services. modern applications and services.

CHAPTER 6

FURTURE SCOPE

The future scope for Reliable UDP protocols is promising and continues to be a topic of considerable interest within the networking and software development communities. While there was no widely accepted standard as of my last knowledge update in January 2022, the ongoing development and research in this area suggest several potential avenues for growth and adoption:

- **Improved Real-Time Applications:** With the proliferation of real-time applications, such as online gaming, video conferencing, and IoT (Internet of Things) devices, there is a growing need for low-latency, reliable data transmission. Reliable UDP protocols could play a significant role in enhancing the performance of these applications.

- **5G and Edge Computing: The** rollout of 5G networks and the rise of edge computing demand efficient and reliable data transmission. Reliable UDP can help reduce latency and improve data delivery in these emerging technologies.

- **Content Delivery Networks (CDNs):** CDNs are continually looking for ways to optimize content delivery. Reliable UDP protocols could potentially enhance the efficiency and reliability of content distribution.

- **Streaming Services:** Reliable UDP can offer a more robust solution for streaming services, ensuring that multimedia content is delivered without interruptions or buffering.

- **Efficiency in Large-Scale Data Transfers:** In scenarios involving large-scale data transfers, such as cloud storage or backups, Reliable UDP can significantly improve data transfer speed while maintaining data integrity.

- **Machine-to-Machine Communication:** The growth of the IoT and machine-to-machine communication requires efficient and reliable data transmission, making Reliable UDP protocols relevant for these applications.

- **Security and Privacy:** Future iterations of Reliable UDP protocols may incorporate enhanced security features to protect data during transmission.

- **Standardization Efforts**: The networking community may work towards standardizing Reliable UDP protocols, leading to wider adoption and interoperability.

The future scope of Reliable UDP is intrinsically linked to the ever-evolving demands of modern applications and services. As networking technologies advance and user expectations continue to rise.

CHAPTER 7

REFERENCES

Books:

- "Computer Networks: A Top-Down Approach" by Kurose and Ross (6th Edition)
- "Networking Essentials" by Comer (4th Edition)
- "TCP/IP Illustrated, Volume 1: The Protocols" by Stevens (2nd Edition)

Articles:

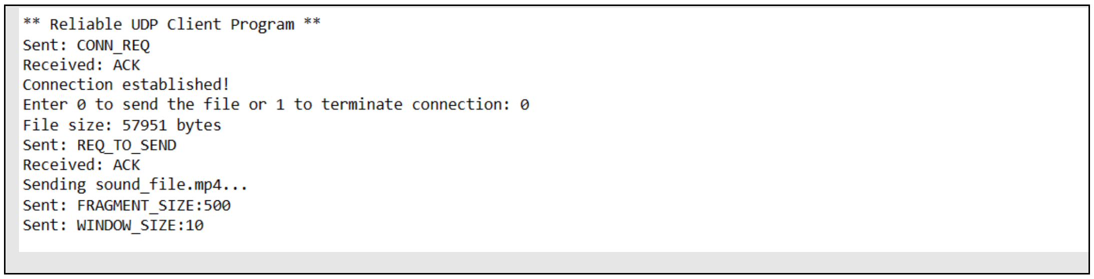
- "Implementing reliable UDP in Python using socket" on Stack Overflow (https://stackoverflow.com/questions/61818523/implementing-reliability-in-udp-python)
- "Reliable UDP using Python Socket Programming" on GitHub (https://github.com/zainamir-98/reliable-udp)
- "zainamir-98/reliable-udp: This is a project I worked on for my Computer and Communication Networks course during my undergrad EE program." on GitHub (https://github.com/topics/reliable-udp)

APPENDIX

Client code

```
import socket
import os

# Constants
SERVER_IP = "127.0.0.1"
SERVER_PORT = 50003
FRAGMENT_SIZE = 500
WINDOW_SIZE = 10
TIMEOUT = 4
FILE_NAME = "sound_file.mp4"

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.settimeout(TIMEOUT)

def send_message(message):
    client_socket.sendto(message.encode('utf-8'), (SERVER_IP, SERVER_PORT))
    print(f"Sent: {message}")

def receive_message():
    try:
        data, _ = client_socket.recvfrom(FRAGMENT_SIZE)
        print(f"Received: {data.decode('utf-8')}")
        return data.decode('utf-8')
    except socket.timeout:
        print("Timeout")
        return "TIMEOUT"

def transfer_file():
```

```
with open(FILE_NAME, "rb") as file:
    file_size = os.path.getsize(FILE_NAME)
    print(f"File size: {file_size} bytes")

    # Send a request to start file transfer
    send_message("REQ_TO_SEND")
    response = receive_message()

    if response == "ACK":
        print(f"Sending {FILE_NAME}...")
        send_message(f"FRAGMENT_SIZE:{FRAGMENT_SIZE}")
        send_message(f"WINDOW_SIZE:{WINDOW_SIZE}")

        packet_num = 0
        seq_num = 0
        window = []

        while True:
            if seq_num < WINDOW_SIZE:
                data = file.read(FRAGMENT_SIZE)
                if not data:
                    break

                # Simulate packet loss
                if packet_num == 12 or packet_num == 55:
                    print(f"Simulating packet loss for packet {packet_num}")
                else:
                    client_socket.sendto(data, (SERVER_IP, SERVER_PORT))
                    window.append(data)

                seq_num += 1
                packet_num += 1
            if seq_num == WINDOW_SIZE or not data:
                response = receive_message()

                if response == "ACK":
                    print("Received acknowledgment. Continuing...")
                    seq_num = 0
                    window = []
                elif response == "NACK":
                    print("Received negative acknowledgment. Resending packets...")
                    for i, packet in enumerate(window):
                        client_socket.sendto(packet, (SERVER_IP, SERVER_PORT))
                    seq_num = 0
                elif response == "TIMEOUT":
                    print("Server is not responding. Aborting transfer.")
                    break

    client_socket.close()

def main():
    print("** Reliable UDP Client Program **")

    # Connect to the server
    send_message("CONN_REQ")
    response = receive_message()

    if response == "ACK":
        print("Connection established!")

        while True:
            option = input("Enter 0 to send the file or 1 to terminate connection: ")

            if option == "0":
                transfer_file()
            elif option == "1":
                send_message("DISCONNECT")
                print("Connection terminated.")
                break
            else:
                print("Invalid option. Please enter 0 or 1.")

    else:
        print("Connection failed. Try again.")
if __name__ == "__main__":
    main()
```

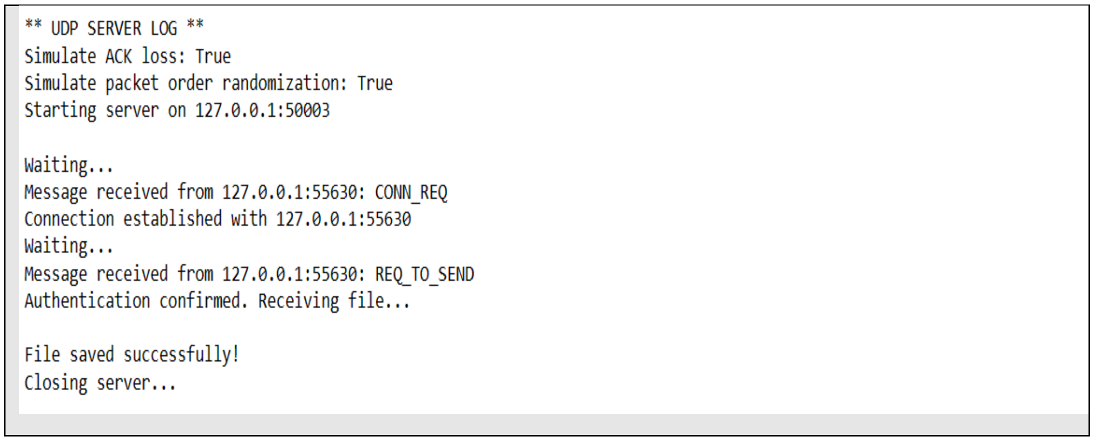


```
** Reliable UDP Client Program **
Sent: CONN_REQ
Received: ACK
Connection established!
Enter 0 to send the file or 1 to terminate connection: 0
File size: 57951 bytes
Sent: REQ_TO_SEND
Received: ACK
Sending sound_file.mp4...
Sent: FRAGMENT_SIZE:500
Sent: WINDOW_SIZE:10
```

Server

```
import socket
import random

# Global Constants
IP = "127.0.0.1"
PORT = 50003
FRAGMENT_SIZE = 500
WINDOW_SIZE = 10
RECEIVED_FILE_NAME = "received_sound_file.mp4"
```

```
# Error Simulation Switches
SIM_ACK_LOSS = True
SIM_PACKET_ORDER_MIX = True

# Global Variables
client_list = []
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((IP, PORT))

def establish_connection(client_addr):
    client_list.append(client_addr)
    server_socket.sendto(b"ACK", client_addr)
    print(f"Connection established with {client_addr[0]}:{client_addr[1]}")

def process_final_window_packet(client_addr, rec_data_buffer):
    # Simulate packet order randomization
    if SIM_PACKET_ORDER_MIX:
        random.shuffle(rec_data_buffer)

    # Save data to a file
    with open(RECEIVED_FILE_NAME, 'ab') as f:
        for data in rec_data_buffer:
            f.write(data)

    # Send ACK_ALL to the client
    if not SIM_ACK_LOSS:
        server_socket.sendto(b"ACK_ALL", client_addr)

def main():
    print("** UDP SERVER LOG **")
    print("Simulate ACK loss:", SIM_ACK_LOSS)
    print("Simulate packet order randomization:", SIM_PACKET_ORDER_MIX)
    print(f"Starting server on {IP}:{PORT}\n")


    while True:
        print("Waiting...")
        msg, client_addr = server_socket.recvfrom(FRAGMENT_SIZE)
        print(f"Message received from {client_addr[0]}:{client_addr[1]}:
{msg.decode()}")


        if msg == b"CONN_REQ":
            establish_connection(client_addr)
        elif msg == b"REQ_TO_SEND" and client_addr in client_list:
            server_socket.sendto(b"ACK", client_addr)
            print("Authentication confirmed. Receiving file...")


            # Receive and process the file data here


            print("\nFile saved successfully!")


            # Close the server
            server_socket.close()
            print("Closing server...")
            break


if __name__ == "__main__":
    main()
```



```
** UDP SERVER LOG **
Simulate ACK loss: True
Simulate packet order randomization: True
Starting server on 127.0.0.1:50003

Waiting...
Message received from 127.0.0.1:55630: CONN_REQ
Connection established with 127.0.0.1:55630
Waiting...
Message received from 127.0.0.1:55630: REQ_TO_SEND
Authentication confirmed. Receiving file...

File saved successfully!
Closing server...
```