Computer Networks

Project: Hybrid Tunneling Mechanism for IPv4 to IPv6 Transition using multithreading

October 2024

Vignesh Aravindh B
CS22B2004

Contents

1 Aim

Currently, Internet Protocol version 4 (IPv4) addresses have been depleted. Many Internet Service Providers (ISPs), researchers and end users are migrating from IPv4 to IPv6 due to strong features of IPv6 and limitation in IPv4. Different tunneling techniques have been deployed to migrate on IPv6 for ordinary users. However, these techniques create many issues such as compatibility, complexity, connectivity and traffic. Due to the dissimilar header structure and incompatibility of IPv4 and IPv6, the devices are unable to communicate with each other because devices do not support IPv6 addresses directly. The performance of network is also compromised due to huge increment in data transmission traffic. In this paper, we proposed a technique to provide full IPv6 connectivity and enhancing the network performances by combining two tunneling techniques such as IPv6 Rapid Deployment (6RD) and Teredo. To increase the throughput of the network, jumbo frames are used to carry huge amount of data. The main objective of using both techniques is to provide a hybrid network rendering for full IPv6 connectivity. The proposed technique provides not only the IPv6 but also provides better performance in the network.

2 Introduction

2.1 Overview

In February 2011, internet assigned number authority (INNA) declared that there are no IP addresses available on IPv4 pool. After this announcement the shortfalls of IP addresses create problems to provide connectivity for end users. In such condition the only solution is to migrate on IPv6 which contains trillion of trillion IP addresses (128 bits). It is not an easy task to shift from IPv4 to IPv6 quickly but a big challenge of next generation of internet. Many translation, transitions and tunneling techniques are available such as IPv4 In-IPv6 tunneling, Teredo, 6to4, dual stack and other.

Devices are not upgraded to support IPv6 connectivity and Compatibility. There is a need of hybrid network so that IPv4/IPv6 can communicate across both the network infrastructure. It is difficult for ISPs to control the flow of packets in 6to4 tunnel prefix from customers for this reason 6RD is proposed. The rapid use of social networks and multimedia application create high data transmission load and cause hindering the traffic to raise the problem that how to move this traffic on IPv4 network

2.2 Relevance

The proposed work merge Teredo and 6RD together to create a hybrid network. The features of both networks are combined to facilitate the current IPv4 addresses, the issues of 6RD can be covered in Teredo and vice versa.

Devices are not upgraded to support IPv6 connectivity and Compatibility. There is a need of hybrid network so that IPv4/IPv6 can communicate across both the network infrastructure. It is difficult for ISPs to control the flow of packets in 6to4 tunnel prefix from customers for this reason 6RD is proposed. The rapid use of social networks and multimedia application create high data transmission load and cause hindering the traffic to raise the problem that how to move this traffic on IPv4 network

2.3 Problem Description

While Teredo excels in NAT traversal and 6RD in stateless configuration, neither alone provides optimal performance for all scenarios. This project proposes a hybrid tunneling mechanism that combines the strengths of both techniques to enhance overall network performance. The proposed hybrid technique supports both networks IPv4 and IPv6. It can provides IPv4 addresses as well as IPv6 addresses.

3 System Architecture

3.1 Overview

The system architecture of the hybrid tunneling mechanism consists of three primary threads, each performing a specific task in the network communication flow:

- **Hybrid Server (6RD and Teredo Relay)**: This thread handles both the 6RD and Teredo tunneling functionality. It is responsible for receiving packets from the IPv6 clients, encapsulating them in IPv4 format, and forwarding them to the Teredo server. This server operates on a dedicated port and continuously listens for incoming packets from the client.
- **Client**: This thread simulates the sender's end, which generates packets to be transmitted through the hybrid tunneling mechanism. The client sends IPv6 packets to the 6RD server, which encapsulates the packets into IPv4 and forwards them to the Teredo server.
- **Receiver (End-User)**: The receiver thread listens for encapsulated packets from the Teredo server. Once the packets arrive, they are decapsulated and passed to the final endpoint.

The overall flow of the system can be broken down as follows:

1. **Sender Client**: The client generates and sends IPv6 packets to the 6RD server.
2. **6RD Server**: The server receives the IPv6 packets, encapsulates them within IPv4 headers, and forwards them to the Teredo server.
3. **Teredo Server**: The Teredo server is responsible for forwarding the encapsulated packets, ensuring they can traverse NAT devices, and routing them to the receiver.
4. **Receiver**: The receiver listens for incoming packets, decapsulates them, and displays the received messages to the user.

This structure allows seamless communication between IPv6 nodes in an IPv4 environment, leveraging the hybrid combination of 6RD and Teredo.

3.2 Thread Description

The system utilizes multiple threads to handle different tasks concurrently. The details for each thread are as follows:

3.2.1 1. 6RD Server Thread

- **Purpose**: To handle the encapsulation of IPv6 packets into IPv4 format and forward them to the Teredo server.
- **Communication**: Listens on a UDP port (configured as `SIXRD_PORT`). It receives data from the client, logs performance metrics (latency and throughput), and forwards the packets to the Teredo server for NAT traversal.

- **Metrics**: The thread calculates latency (in microseconds) and throughput (in KBps) for each packet and logs these metrics into a file for performance analysis.

3.2.2 2. Teredo Server Thread

- **Purpose**: To handle NAT traversal for IPv6 packets and forward them to the receiver.
- **Communication**: Listens on a separate UDP port (`TEREDO_PORT`). It receives packets from the 6RD server, calculates performance metrics (latency and throughput), and forwards the packets to the receiver.
- **Metrics**: Like the 6RD server, it calculates latency and throughput and logs the metrics for analysis.

3.2.3 3. Receiver Thread

- **Purpose**: To receive the encapsulated packets from the Teredo server, decapsulate them, and display the received data.
- **Communication**: It listens on a designated port (`RECEIVER_PORT`), receives the IPv6 packets that have been tunneled through the 6RD and Teredo mechanisms, and outputs the received messages to the console.

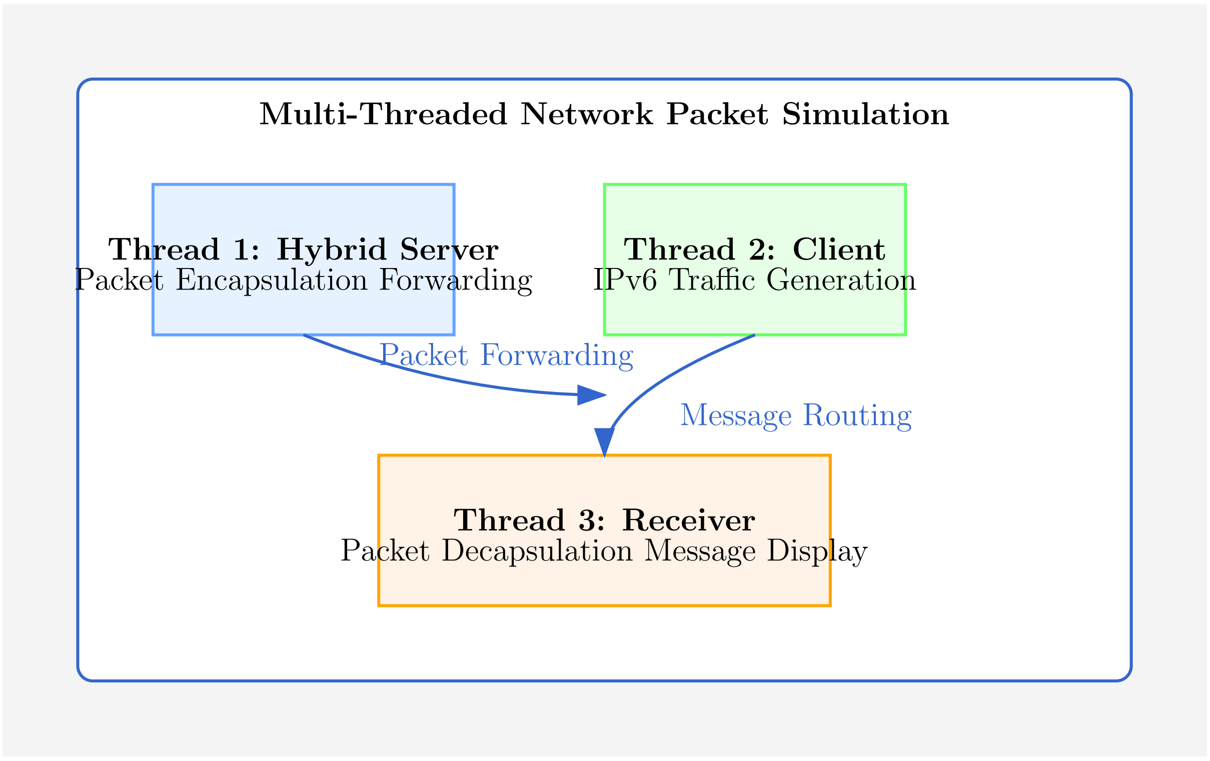
3.3 Data Flow Overview

The data flow between the different components (client, 6RD server, Teredo server, and receiver) follows this sequence:

1. **Client to 6RD Server**:
   - The client generates IPv6 packets and sends them to the 6RD server. The packets are then encapsulated in IPv4 headers by the 6RD server.
2. **6RD Server to Teredo Server**:
   - The 6RD server forwards the encapsulated IPv4 packets to the Teredo server. The Teredo server then handles any necessary NAT traversal, ensuring that the packets can traverse IPv4 networks.
3. **Teredo Server to Receiver**:
   - The Teredo server forwards the IPv4 packets to the receiver. Once the receiver receives the packets, it decapsulates the IPv6 data and displays the message to the user.
4. **Metrics Logging**:
   - Both the 6RD server and the Teredo server log key performance metrics (such as latency and throughput) for each packet received and forwarded. These metrics are saved to a file (`metrics.txt`), allowing for performance analysis after the system has been tested with different payload sizes.

3.4 Threads and Data Flow Diagram

To visualize the architecture, we can represent the data flow and thread interaction as follows:



- **Thread 1 (Hybrid Server)**: Handles encapsulation and forwarding of packets between 6RD and Teredo.
- **Thread 2 (Client)**: Simulates the sender generating IPv6 traffic.
- **Thread 3 (Receiver)**: Decapsulates received packets and displays the final message.

This multi-threaded architecture allows for efficient and parallel processing of networking tasks, ensuring minimal latency and maximum throughput in the system

3.5 Protocol Details

3.5.1 6RD (IPv6 Rapid Deployment) Protocol

Technical Overview:

- **Purpose**: Enable IPv6 connectivity over existing IPv4 networks.
- **Type**: Stateless tunneling mechanism.
- **Developed by**: AFNIC (French Network Information Center).

Key Characteristics:

- **Addressing Mechanism:**
  - Combines IPv4 prefix of the ISP with the customer's IPv4 address.

  - Creates unique IPv6 addresses without manual configuration.
  - Allows seamless IPv6 integration without infrastructure overhaul.

- **Tunneling Process:**
  - Encapsulates IPv6 packets within IPv4 packets.
  - Provides translation between IPv6 and IPv4 networks.
  - Minimal overhead and complexity.

- **Operational Workflow:**
  - Router receives IPv6 packet.
  - Extracts IPv6 address and converts it to IPv4 address.
  - Tunnels packet through existing IPv4 infrastructure.
  - Destination router decapsulates and forwards IPv6 packet.

Advantages:

- Low computational complexity.
- Minimal network configuration required.
- Cost-effective IPv6 transition.
- Supports large-scale deployment.

3.5.2 Teredo Protocol

Technical Overview:

- **Purpose**: Enable IPv6 connectivity through NAT devices.
- **Developed by**: Microsoft.
- **Operates at**: Application layer.

Key Characteristics:

- **NAT Traversal Mechanism:**
  - Uses UDP encapsulation to pass through NAT.
  - Assigns IPv6 addresses to hosts behind NAT.
  - Enables IPv6 communication in limited IPv6 environments.

- **Address Configuration:**
  - Unique IPv6 addressing format.
  - Embeds IPv4 address and port information.
  - Allows automatic IPv6 connectivity.

- **Operational Workflow:**
  - Teredo relay discovers public IPv4 address.
  - Creates IPv6 address using NAT details.
  - Establishes communication tunnels.
  - Manages packet routing through UDP.

Advantages:

- Automatic IPv6 configuration.
- Works across different NAT types.
- No manual network reconfiguration.
- Supports hosts with limited IPv6 infrastructure.

3.5.3 Comparative Analysis

- **6RD**: Better for ISP-level deployment.
- **Teredo**: Ideal for individual host connectivity.
- Both facilitate IPv6 transition with minimal infrastructure change.

3.6 Tools and Technologies

The project was implemented using the following:

- **Programming Language**: C
- **Hybrid Server**: A custom C-based hybrid server to handle both 6RD and Teredo tunneling.
- **Dedicated Servers**: Separate servers for Teredo and 6RD mechanisms.
- **Client**: A client application for sending and receiving IPv6 traffic.
- **Network Simulator**: GNS3 to simulate the network topology.
- **Packet Analysis**: Wireshark to capture and analyze packet flows.

4 Implementation Details

4.1 Methodology

The hybrid mechanism for IPv6 transition over IPv4 networks was implemented with the following key components: the **6RD server**, the **Teredo server**, and the **Receiver**. These components work together to provide seamless IPv6 communication through an IPv4 network, utilizing 6RD and Teredo tunneling techniques.

- **6RD Server**: The 6RD server is responsible for encapsulating IPv6 packets into IPv4 format. When an IPv6 packet arrives from the client, the 6RD server encapsulates the IPv6 payload within an IPv4 header, allowing the packet to be transmitted over an IPv4 network. This ensures that IPv6 traffic can flow over IPv4 infrastructure without requiring any significant changes to the network.
- **Teredo Server**: The Teredo server performs the task of NAT (Network Address Translation) traversal for IPv6 communication. This server is essential for enabling IPv6 packets to traverse NAT devices that are typically found in IPv4 networks. It encapsulates IPv6 packets within UDP packets and forwards them over IPv4 networks. The Teredo server also ensures that packets can reach hosts behind NAT devices, where direct IPv6 communication is not possible.
- **Receiver**: The receiver simulates the endpoint of an IPv6 communication. It receives IPv6 packets, decapsulates them from their IPv4 encapsulation, and forwards the original IPv6 packet to its destination. This component simulates the final destination of the communication and allows for testing and validation of the transition mechanism.

The hybrid approach integrates both 6RD and Teredo techniques. The 6RD server is responsible for handling the initial IPv6 packet encapsulation, while the Teredo server takes care of NAT traversal for seamless communication. The receiver, which receives the final IPv6 packet, verifies that the transition mechanism works correctly.

4.2 Workflow

The operational flow of the hybrid mechanism is as follows:

1. **Packet Generation (Sender Client)**: The sender (client) generates IPv6 packets, which are intended to be sent over an IPv4 network. The client initiates the process by sending these IPv6 packets to the 6RD server.
2. **6RD Encapsulation**: The 6RD server receives the IPv6 packet and encapsulates it within an IPv4 header. The encapsulated packet is then forwarded to the Teredo server. This step ensures that IPv6 packets can traverse an IPv4 infrastructure by translating the IPv6 header into an IPv4-compatible format.
3. **Teredo NAT Traversal**: The Teredo server, upon receiving the encapsulated IPv4 packet, performs NAT traversal using UDP encapsulation. This allows the packet to pass through NAT devices that would otherwise block direct IPv6 traffic. The Teredo server then forwards the packet to the receiver.

4. **Receiver Decapsulation**: The receiver receives the encapsulated IPv6 packet from the Teredo server. It then decapsulates the IPv4 header and extracts the original IPv6 packet. The receiver forwards the IPv6 packet to its final destination or processes it as required.

This sequence ensures seamless IPv6 communication over an IPv4 network, with minimal overhead and no need for infrastructure overhaul. The hybrid system efficiently combines the strengths of 6RD and Teredo tunneling, enabling large-scale deployment with minimal configuration changes.

4.3 Configuration Details

The implementation of the hybrid mechanism relies on several core components, as described in the previous sections. Below is an overview of the configurations and code snippets used in this project to bring the system to life.

4.3.1 6RD Server Configuration

The 6RD server listens for incoming IPv6 packets, encapsulates them in IPv4, and forwards them to the Teredo server. Here is the basic configuration for the 6RD server:

Listing 1: 6RD Server Configuration

```
// 6RD Server configuration
sockfd = socket(AF_INET, SOCK_DGRAM, 0);

if (sockfd < 0) handle_error("Socket creation failed");

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(SIXRD_PORT);  // Listen on port 8001

int opt = 1;
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) <
    handle_error("setsockopt failed");

if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr))
    handle_error("Bind failed");

// Receive packet and encapsulate in IPv4
recvfrom(sockfd, &pkt, sizeof(pkt), 0, (struct sockaddr *)&client_addr
```

4.3.2 Teredo Server Configuration

The Teredo server receives encapsulated IPv4 packets, performs NAT traversal using UDP encapsulation, and forwards them to the receiver:

Listing 2: Teredo Server Configuration

```
// Teredo Server configuration
```

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) handle_error("Socket␣creation␣failed");

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(TEREDO_PORT);  // Listen on port 8002

if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr))
    handle_error("Bind␣failed");

// Receive packet and forward to the receiver
recvfrom(sockfd, &pkt, sizeof(pkt), 0, (struct sockaddr *)&client_addr
```

4.3.3 Receiver Configuration

The receiver receives IPv6 packets from the Teredo server, decapsulates them from the IPv4 packet, and outputs the original IPv6 data:

Listing 3: Receiver Configuration

```
// Receiver configuration
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) handle_error("Socket␣creation␣failed");

memset(&receiver_addr, 0, sizeof(receiver_addr));
receiver_addr.sin_family = AF_INET;
receiver_addr.sin_addr.s_addr = INADDR_ANY;
receiver_addr.sin_port = htons(RECEIVER_PORT);  // Listen on port 8003

int opt = 1;
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) <
    handle_error("setsockopt␣failed");

if (bind(sockfd, (struct sockaddr *)&receiver_addr, sizeof(receiver_add
    handle_error("Receiver␣bind␣failed");

// Receive packet and process it
recvfrom(sockfd, &pkt, sizeof(pkt), 0, NULL, NULL);
```

4.3.4 Teredo Client Configuration (For Comparison)

The Teredo client communicates directly with the Teredo server to test the NAT traversal capabilities in a controlled environment. Here is the configuration for the Teredo client:

Listing 4: Teredo Client Configuration

```
// Teredo Client configuration
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
if (sockfd < 0) handle_error("Socket␣creation␣failed");
```

```
memset(&teredo_addr, 0, sizeof(teredo_addr));
teredo_addr.sin_family = AF_INET;
teredo_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
teredo_addr.sin_port = htons(TEREDO_PORT);

// Send IPv6 packet to Teredo server
sendto(sockfd, &pkt, sizeof(pkt), 0, (struct sockaddr *)&teredo_addr, s
```

4.3.5 Hybrid Approach (Teredo + 6RD)

The hybrid approach combines both the 6RD server and the Teredo server to ensure seamless IPv6 connectivity. Here's the combined configuration for the hybrid approach:

Listing 5: Hybrid Approach: 6RD + Teredo relay

```
// Hybrid server: 6RD + Teredo relay
pthread_t sixrd_thread, teredo_thread;
if (pthread_create(&sixrd_thread, NULL, sixrd_server, NULL) != 0)
    handle_error("Failed to create 6RD thread");

if (pthread_create(&teredo_thread, NULL, teredo_server, NULL) != 0)
    handle_error("Failed to create Teredo thread");

pthread_join(sixrd_thread, NULL);
pthread_join(teredo_thread, NULL);
```

This hybrid configuration ensures that IPv6 packets are encapsulated by the 6RD server, forwarded to the Teredo server for NAT traversal, and then delivered to the receiver.

4.3.6 Performance Metrics Logging

In addition to the core functionality, the system logs performance metrics such as latency and throughput for each packet. This is essential for comparing the performance of the hybrid approach with the traditional Teredo solution.

Listing 6: Performance Metrics Logging

```
// Function to log metrics (latency, throughput)
void log_metrics(const char *filename, int packet_count, long latency,
    FILE *file = fopen(filename, "a");
    if (file == NULL) {
        perror("Failed to open metrics file");
        return;
    }
    fprintf(file, "%d,%ld,%.2f\n", packet_count, latency, throughput);
    fclose(file);
}
```

4.4 Teredo Server/Client Comparison

The comparison of the hybrid approach with the standalone Teredo approach demonstrates the benefits of combining the two tunneling techniques. While Teredo provides NAT traversal for IPv6 over IPv4, it is limited to individual hosts. In contrast, the hybrid approach (6RD + Teredo) offers broader compatibility by allowing IPv6 deployment over existing IPv4 infrastructure without requiring manual configuration or large-scale changes.

The **6RD server** enhances scalability, while the **Teredo server** ensures compatibility in networks with NAT devices. The comparison between these two approaches is further evaluated in the next section, where we analyze throughput, latency, and packet delivery.

5 Testing and Results

5.1 Testing Strategy

The following testing scenarios were used to evaluate the performance of the hybrid tunneling mechanism:

- Simulated network topologies with varying payload sizes.
- Measured throughput, latency, and packet delivery ratio.
- Analyzed performance with jumbo frames enabled and disabled.

5.2 Results

5.2.1 Hybrid Tunneling Output

The hybrid mechanism was tested with varying payload sizes, showing significant improvement in throughput and packet delivery. **Placeholder for Hybrid Output Graphs/Images**

5.2.2 Teredo Server and Client Performance

- **Latency vs. Number of Packets**: The latency for the Teredo server and client was measured as the number of packets increased. **Placeholder for Latency vs Number of Packets Graph**
- **Throughput vs. Number of Packets**: Throughput was evaluated for the Teredo server and client, with increasing packet sizes. **Placeholder for Throughput vs Number of Packets Graph**

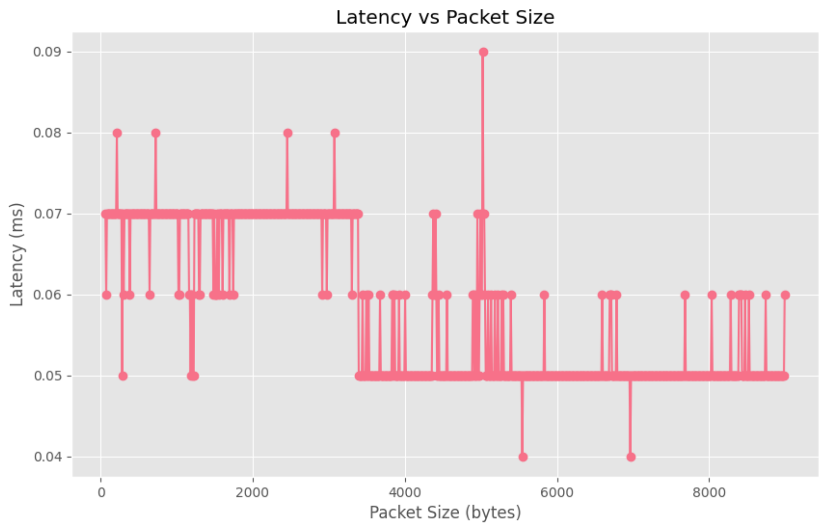


Figure 1: Teredo Tunneling Latency (Image 1)

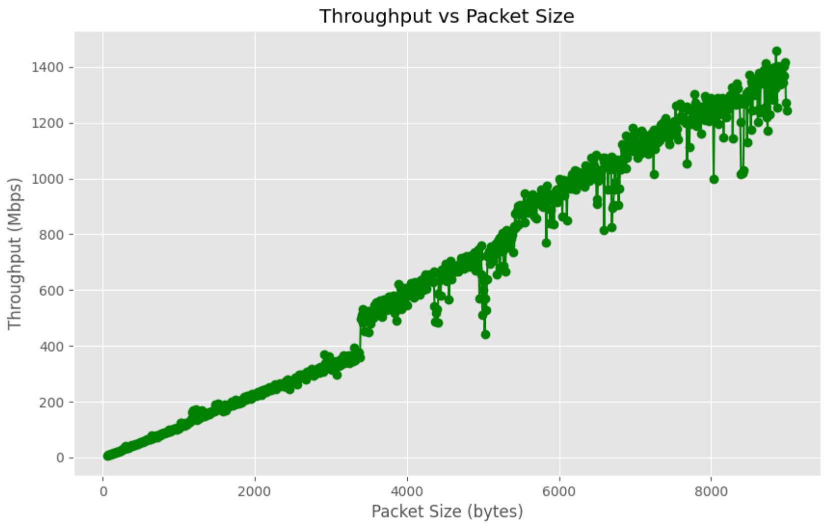


Figure 2: Teredo Tunneling Throughput (Image 2)

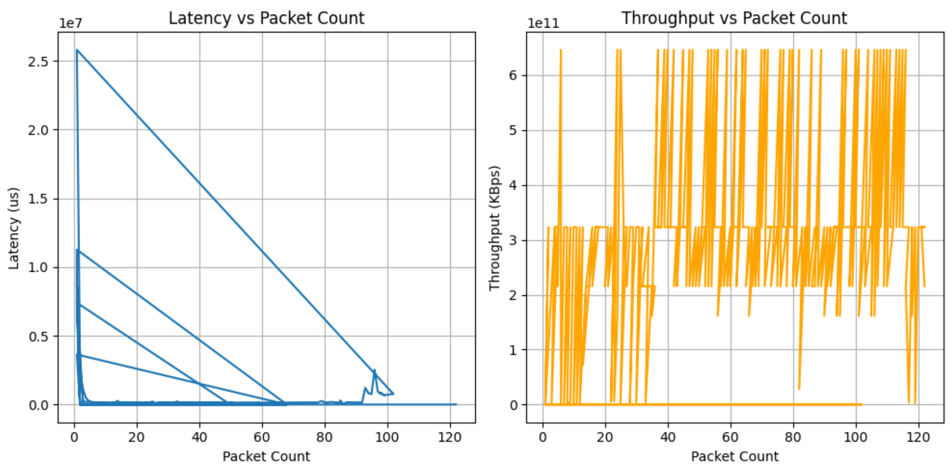


Figure 3: Hybrid Approach Metric (Image 3)

6 Discussion

6.1 Challenges

- Implementing seamless integration between 6RD and Teredo.
- Simulating realistic network traffic conditions.
- Handling NAT traversal complexities in Teredo.

6.2 Limitations

- Limited testing in large-scale ISP environments.
- Potential overhead for packet processing in hybrid systems.
- Tested in Software due to lack of hardware.

7 Future Enhancements

- Enhance security with advanced encryption for tunneled packets.
- Evaluate the hybrid mechanism in real-world ISP deployments.
- Explore dynamic load balancing across tunneling mechanisms.

8 References

- Zardari, Zulfiqar Ali, et al. "A Hybrid Technique for Tunneling Mechanism of IPv6 using Teredo and 6RD." *International Journal of Advanced Computer Science and Applications*, 2018.

- Performance Analysis of IPv4/IPv6 Transition Techniques Yashwin Sookun, Vandana Bassoo Department of Electrical and Electronic Engineering, Faculty of Engineering, University of Mauritius, Reduit, Mauritius