# CSCD58 Project Report

## 1. Introduction and Project Description

**Project Title:** LAN Speed Testing Tool

**Github Link:** https://github.com/Minjun1Kim/LANSpeedTestTool

**Group Members:**


- Minjun Kim (1007705146)
- Ahmad Hakim (1007904207)


**Project Overview:** This project aims to create a LAN speed testing tool similar to iperf with unique twists, supporting various tests (upload, download, ping, jitter) over different protocols (TCP, UDP, and ICMP). The tool allows users to measure throughput, round-trip time (RTT), jitter, and analyze network performance both in real and simulated environments, such as Mininet. Additionally, a simple web component is integrated for jitter visualization.

**Goals and Objectives:**

- Implement server-client architecture to measure network performance metrics.
- Support multiple test types:
  - Upload: Client sends data to server (TCP/UDP).
  - Download: Server sends data to client (TCP/UDP).
  - Ping: Measure RTT and packet loss (UDP or ICMP).
  - Jitter: Evaluate variations in RTT using TCP.
- Allow selection of protocol for ping tests (UDP or ICMP).
- Test the tool concurrently in a Mininet environment to ensure it handles multiple clients.
- Integrate a web front-end to visualize jitter results using a simple Flask-based application.

## 2. Relation to CSCD58 Course Content

This project aligns with the core themes and learning outcomes of the course CSCD58:

- **Network Layers and Protocols**:
  The tool uses both TCP and UDP sockets, reinforcing understanding of transport-layer concepts. The ICMP-based ping test interacts with the network layer directly using raw sockets.
- **Sockets Programming**:
  Implementing server and client modes with both TCP and UDP sockets directly builds upon the socket programming concepts taught in the course.
- **Mininet Simulation**:
  Using Mininet to simulate a network topology and test the tool's performance under

controlled conditions relates to course modules about network emulation and experimentations.

- **Performance Metrics (RTT, Throughput, Jitter)**:
  The project measures performance metrics such as RTT, jitter, and throughput, connecting to topics on network performance, latency, and bandwidth covered in the textbook "Computer Networks: A Systems Approach."
- **Concurrent Testing**:
  Handling multiple concurrent clients in a controlled environment (Mininet) reflects the course's focus on real-world issues like congestion, concurrency, and scalability.
- **Data Visualization**:
  The web component for jitter visualization relates to understanding, interpreting, and presenting network data effectively, a skill emphasized in practical network analysis.

## 3. Team Contributions

Both Minjun and Ahmad collaborated closely, engaging in peer programming sessions, code reviews, and testing. While responsibilities often overlapped, the primary areas of focus for each member were:

- **Minjun Kim**:
  Primarily focused on the jitter test implementation, the upload test logic, and the ping test (ICMP). Worked on refining throughput calculations, MB/s conversions, and ensuring tests integrate smoothly with the server's concurrency. Also, implemented the Web feature.
- **Ahmad Hakim**:
  Mainly handled the ping test (UDP) and download test functionality, as well as the upload test logic. Focused on ensuring correct RTT calculations, refining packet size and interval logic. Resolved a lot of issues arising from the main features.

**Collaboration Approach:**

- Both members tested each other's code, communicated frequently, and contributed to the repository equally. They participated in design discussions, bug fixes, and enhancements.
- Peer code reviews and pair programming sessions ensured shared knowledge and consistent coding standards.

## 4. Instructions for Running and Testing

This is mentioned in the **README.md** file in detail.

# CSCD58 Project Report

## 5. Implementation Details and Documentation

**Architecture:**

- **Server:**
  Spawns threads for TCP/UDP clients and a raw socket thread for ICMP. Measures performance (bytes, time) and returns results.
- **Client:**
  Connects to the server, sends test type and parameters. For upload, continuously sends data until the duration expires. For download, receives data until duration lapses. For ping, sends packets and measures RTT. For jitter, sends multiple packets to compute RTT variation.

**Data Structures and Key Functions:**

- **create_socket()**: Creates and binds server sockets (TCP/UDP).
- **handle_icmp_ping()**: Handles ICMP Echo Requests and replies with Echo Replies.
- **run_icmp_ping_test(), run_ping_test()**: Client-side logic for ICMP or UDP-based ping.
- **run_udp_upload_test(), run_udp_download_test()**: Client logic for UDP tests.
- **run_tcp_upload_test(), run_tcp_download_test()**: Client logic for TCP tests.
- **calculate_checksum()**: Calculates ICMP checksums.

**Unit Conversions and Output:**

- Throughput and results now displayed in MB and seconds.
- RTT, jitter, and packet loss metrics displayed for ping and jitter tests.

## 6. Analysis and Discussion

**Results:**

- ICMP ping results closely match standard **ping**.
- Concurrent tests show the server can handle multiple clients simultaneously, with minimal packet loss and stable RTTs.
- TCP tests show expected throughput and line rate performance in a simple Mininet setup.
- Jitter tests produce expected RTT variation graphs, easily visualized via the web component.

**Comparison to Standard Tools:**

- RTT and packet loss align with **ping**.
- Throughput values are similar to **iperf** for simple topologies.

- The simple Flask-based UI adds value by visualizing jitter trends, something **ping** or **iperf** do not directly provide.

## 7. Concluding Remarks and Lessons Learned

**Conclusions:**

- Implementing both TCP and UDP tests, along with ICMP ping and jitter calculations, thoroughly exercises network programming concepts.
- Running in Mininet environment confirms that the tool can handle realistic network scenarios, concurrency, and partial packet losses.
- The web component offers an enhanced user experience for analyzing jitter.

**Lessons Learned:**

- Managing threads for TCP, UDP, and ICMP tests was a key challenge. We learned to carefully synchronize shared resources and use proper thread detachment techniques to avoid memory leaks or dangling threads. Handling multiple clients concurrently required a robust design to ensure that each test session operated independently without interference.
- Working with raw sockets required root permissions and careful handling of ICMP packet construction and parsing. Debugging ICMP checksum mismatches highlighted the importance of precise implementation at the packet level.
- Mininet proved to be a valuable tool for simulating network topologies and stress-testing our application. We encountered and resolved issues related to duplicate packets and congestion, gaining insights into how real networks behave under load.
- We learned the value of presenting results in user-friendly units like MB/s and seconds. This change made the output easier to interpret, especially during demonstrations and testing.
- Pair programming, frequent testing, and detailed discussions on Git helped ensure all team members contributed equally and understood every part of the project. Debugging multi-threaded server issues was a collective effort that improved our understanding of concurrent programming.

## 8. Future Work

- Future iterations could allow **all test types (upload, download, ping, jitter)** to be conducted using either **UDP, TCP, or ICMP**, providing users with greater flexibility to match real-world testing scenarios.
- Add more test parameters (e.g., number of packets instead of duration).
- Extend the web UI for more sophisticated real-time visualizations.
- Integrate advanced scheduling or QoS tests.