CSE361: Computer Networking

Project Proposal

Submitted to:

Dr. Karim Emara

Eng. Noha Wahdan

Submitted by:

Belal Anas Mohamed (23P0193)

Toka Elsayed Abdelrahman (23P0044)

Maryam Mohamed Abdelmoniem (23P0056)

Mariam Tarek (23P0214)

Carol Eid Aageeb (23P0333)

Jana Mohammed Ahmed (23P0050)

Date of Submission:

7th of November 2025

Contents

1.0 INTRODUCTION

The rapid growth of the Internet of Things (IoT) has increased the demand for simple, reliable, and low-overhead communication protocols suitable for small, resource-limited devices. Protocols like MQTT and CoAP are widely used but can be excessive for devices that transmit only small sensor readings.

To address this, our project introduces Tiny-Telemetry Protocol (TTP v1), a lightweight UDP-based protocol for low-power IoT sensors. The goal is to ensure fast, efficient, and loss-tolerant data transmission without complex retransmission mechanisms.

TTP enables sensors to send readings such as temperature or humidity to a central collector while tolerating network loss and delay. This project will design and implement the protocol, analyze its performance under different network conditions, and measure its efficiency and reliability.

2.0 PROJECT OBJECTIVES

The main objectives of this project are as follows:

The main objectives are to:

- Design a lightweight UDP-based telemetry protocol for IoT sensors.
- Implement a Python client–server system to demonstrate real-time sensor data transmission.
- Analyze protocol performance under delay, loss, and jitter using Linux netem.
- Evaluate key metrics such as packet loss detection, duplicate suppression, and average bytes per reading.
- Prepare a Mini-RFC and final report documenting the protocol's design, implementation, and performance.

3.0 PROPOSED SYSTEM DESIGN

The proposed system consists of two main components:

3.1 Sensor Client (Device):

The client simulates an IoT sensor that periodically sends telemetry data packets to the server using UDP sockets. Each message includes essential fields such as **Device ID, Sequence Number, Timestamp, Message Type**, and optional flags. The client supports two message types:

- **DATA:** Contains one or more sensor readings.
- **INIT:** Empty packet that starts connection
- **HEARTBEAT:** Sent when no new data is available to indicate device liveness.

The client will support a **configurable reporting interval** (1s, 5s, or 30s) and an optional **batching mode** to send multiple readings in one packet for improved efficiency.

3.2 Collector Server:

The server receives packets from multiple clients, maintains per-device state, and logs incoming data to a CSV file. It is responsible for:

- Detecting and reporting sequence gaps (indicating lost packets).
- Performing duplicate suppression and packet reordering based on timestamps.
- Recording metrics such as received packet count, duplicates, and missing sequence numbers.

Overall communication is **stateless and connectionless**, allowing scalability and low resource consumption.

4.0 IMPLEMENTATION APPROACH

The implementation will be developed in **Python** using the built-in socket and struct libraries to handle binary message encoding and UDP communication. Logging will be done through Python's csv and datetime modules, and controlled experiments will use **Linux netem** to emulate packet loss and delay.

Testing scripts will automate the following scenarios:

- **Baseline (no impairment):** Validate that ≥99% of packets arrive correctly.
- **Loss (5%):** Verify sequence gap detection and duplicate suppression.
- **Delay + Jitter (100ms ±10ms):** Confirm proper packet ordering and stable logging.

Each test will be repeated multiple times, and performance metrics such as **bytes per report**, **packet loss rate**, and **CPU usage per report** will be collected and plotted.

5.0 Expected Outcomes and Future Work

The expected outcome of this phase is a **functional prototype** demonstrating successful sensor-to-server data transmission using Tiny-Telemetry over UDP. The protocol will be validated in a baseline environment and later tested under varying loss and delay conditions.

In future phases, we plan to:

- Implement **adaptive batching** to optimize payload size based on network conditions.
- Improve the collector's data analysis and visualization capabilities using Python tools such as Matplotlib.

Ultimately, Tiny Telemetry Protocol aims to serve as an **efficient and educational prototype** for understanding how real-world telemetry systems function over unreliable transport layers, emphasizing simplicity, performance measurement, and robustness under constrained network conditions.