CSE351
COMPUTER NETWORKS
FALL 2025

# Project Proposal


**Prepared By**

Omar Mohamed Mostafa 22P0197
Ezzeldin Ismail Kaoud 22P0141
Ahmed Wael Raafat 22P0221
Mark Edward Aziz 22P0181
Youssef Amr Farouk 2200844
Ahmed Abbady Mohamed 22P0308


**Submitted To**

Prof. Ayman Bahaa-Eldin
Dr. Karim Emara

# Introduction and Motivation

The goal is to design and implement a **custom, lightweight, application-layer protocol** that allows small IoT devices to periodically send readings such as temperature, humidity, or voltage to a central collector.

The communication must use **UDP** as the transport layer, not TCP, and the protocol should be efficient, compact, and robust against moderate packet loss.
In addition to designing the protocol, we'll build a working client and server implementation and evaluate its performance under different network conditions in future phases.

Most IoT sensors operate under strict constraints — limited CPU, memory, and battery power — and often send very small messages (just a few bytes).
However, many existing IoT communication methods like **HTTP**, **MQTT**, or **CoAP** rely on **TCP**, which introduces unnecessary complexity for these simple use cases.

TCP guarantees delivery and ordering, but it does so at a cost:

- It requires setting up and maintaining connections.
- It retransmits lost packets, increasing latency and power usage.
- It adds larger headers and more state at both ends.

For low-power IoT sensors that only need to send periodic updates, these features are excessive. In most cases, the application can tolerate small packet losses — it just needs to *know* if something went missing.

Our goal is to build a **minimal, UDP-based telemetry protocol** that is:

- **Lightweight:** uses compact binary headers to minimize bandwidth.
- **Loss-tolerant:** doesn't retransmit, but detects gaps using sequence numbers.
- **Simple:** small enough to run on basic microcontrollers.
- **Extensible:** allows batching multiple readings in a single packet when needed.

# Proposed Protocol Approach

Our protocol is a simple and efficient UDP-based telemetry system that allows small IoT sensors to send data readings to a central collector. The main design goal is to make communication as lightweight and fast as possible, while still keeping track of message order, duplicates, and missing packets.

Even though UDP is unreliable by nature, our protocol adds a small structure on top of it — a custom header, simple message types, and sequence tracking — to make it reliable enough for real-time sensor reporting without adding unnecessary complexity.

Components:

The protocol works between two components:

- **The Sensor (Client):** acts as the data source. It simulates a small device that periodically sends temperature readings or heartbeat messages.
- **The Collector (Server):** acts as the receiver. It listens for incoming messages, checks their validity, and stores all the information in a CSV log for later analysis.

Each sensor can be configured to send data at different intervals and in different batch sizes. The collector can handle multiple devices and maintain a separate internal state for each one, allowing it to detect lost or duplicated messages per device.

Message Types:

1. **INIT:** sent by the sensor when it first connects, to announce itself.
2. **INIT_ACK:** sent by the collector to confirm it recognized and registered the sensor.
3. **DATA:** carries one or more actual readings from the sensor.
4. **HEARTBEAT:** sent when no new data is available, just to indicate that the device is still alive.

This minimal message set allows initialization, data exchange, and connection maintenance, while keeping the protocol compact and easy to parse.

Message Structure:

Each message starts with a **12-byte header**, which contains the key information needed to identify and process the packet.
The header includes:

- A **magic number**, which acts as a signature to filter out irrelevant or random UDP packets.
- A combined **version and message type** field, which specifies which kind of message it is.
- A **device ID**, to identify the sending sensor.
- A **sequence number**, to track the order of messages and detect gaps or duplicates.
- A **timestamp**, which records when the packet was created (relative to the device's start time).

Following the header, some message types include a **payload**. For example, a data message can contain multiple readings packed together, while a heartbeat message includes no payload.

Communication Flow:

1. **Initialization:**
   When the sensor starts, it sends an INIT message to the collector, introducing itself and declaring its capabilities (for example, the data format it supports). The collector replies with an INIT_ACK to confirm that it's ready to receive data.
2. **Data Reporting:**
   The sensor then begins sending DATA messages at regular intervals (for example, every second).
   Each message contains one or more readings, depending on the configuration.
3. **Heartbeat Messages:**
   If there are no new readings to send, the sensor sends a HEARTBEAT message instead. This ensures the collector knows the device is still active and not disconnected.
4. **Session End:**
   After the configured duration ends, the sensor sends one last HEARTBEAT and stops transmission.