

# **Project Proposal - IoT Telemetry Protocol**

**Course:** [Computer Networking/ CSE361]

**Project:** IoT Telemetry Protocol

**Phase:** 1 – Core Protocol Design and Prototype

---

## **1. Assigned Scenario**

This proposal describes the implementation of Project 1: IoT Telemetry Protocol (Sensor Reporting) using a custom UDP-based protocol named IoT Telemetry Protocol.

IoT Telemetry Protocol enables small, resource-constrained sensors to periodically send telemetry data (temperature, humidity, voltage ...) to a central collector server in an efficient, loss-tolerant way.

---

## **2. Motivation**

Traditional application protocols such as HTTP or MQTT are too heavy for small IoT devices that have:

- limited memory and CPU resources,
- low or unreliable network bandwidth, and
- strict power constraints.

IoT was designed to:

- Operate over UDP to remove handshakes and retransmission overhead,
- Use a compact 12-byte binary header,
- Tolerate up to ~5 % random packet loss,
- Support configurable reporting intervals (1 s, 5 s, 30 s), and
- Remain simple enough for constrained devices.

The result is a lightweight telemetry channel that supports data continuity through timestamps and sequence numbers rather than TCP-style reliability.

## **Proposed Protocol Approach**

### **Transport Layer**

<b>Property</b>	<b>Value</b>
<b>Protocol</b>	<b>UDP</b>
<b>Port</b>	<b>5005</b>

Property	Value
Direction	<b>Sensor (Client) → Collector (Server)</b>
Connection	<b>Connectionless (no session setup)</b>
Retransmission	<b>None – loss tolerant</b>

---

## Entities

- **IoT Sensor (Client)** – builds and sends telemetry packets.
  - **Collector (Server)** – listens on UDP port 5005, decodes headers, and logs data.
- 

## Message Types

Type	Code	Description
INIT	0	<b>Sent once on startup to find the device.</b>
DATA	1	<b>Sent periodically (1 Hz) carrying five float readings.</b>
HEARTBEAT	2	<b>(Reserved for future phase) used when no new data available.</b>

## Binary Header Format (12 bytes)

Field	Size (bytes)	Description
Version	1	<b>Protocol version</b>
MsgType	1	<b>0 = INIT, 1 = DATA, 2 = HEARTBEAT</b>
Device ID	2	<b>Unique sensor identifier</b>
Sequence Number	2	<b>Increment per packet</b>
Timestamp	4	<b>UNIX time (seconds)</b>
Batching Flag	1	<b>0 = single reading, 1 = batched</b>
Checksum	1	<b>8-bit header checksum placeholder</b>

Total: 12 bytes      Python format: '! BBHHIBB'

---

## Payload Format (DATA only)

Each DATA packet holds five readings (floats):

Field	Type	Size
-------	------	------

**Reading 1–5      float × 5      20 bytes total**

**DATA packet size: 12 (header) + 20 (payload) = 32 bytes.**

---

### **Finite State Flow (Simplified)**

**Sensor: START**

↓

**Send INIT → Collector receives & logs.**

↓

**Periodic 1 s timer**

↓

**Send DATA[n] → Collector parses, check sequence, logs.**

↳ repeat

### **Prototype Implementation**

<b>Part</b>	<b>Description</b>
<b>client.py</b>	<b>Sends one INIT + 60 DATA packets (1 Hz).</b>
<b>server.py</b>	<b>Receives UDP packets, unpacks header &amp; payload, prints decoded fields.</b>
<b>script.py</b>	<b>Automates baseline run (60 s) and captures baseline_test.pcap.</b>
<ul style="list-style-type: none"><li>• <b>Language:</b> Python 3</li><li>• <b>Libraries:</b> socket, struct, subprocess, tshark (optional)</li></ul>	

---

## **4. Expected Outcomes**

- Successful UDP communication between client and server.
  - Correct decoding of header (! BBHIBB) and float payload.
  - Baseline (no-loss) test achieves  $\geq 99\%$  packet delivery.
  - Logs and pcap trace generated as proof of functionality.
- 

## **5. References**

1. RFC 768 – User Datagram Protocol (UDP)
2. IoT Telemetry Protocol Mini-RFC (Team Design Document)
3. Python Standard Library Docs – socket, struct, subprocess.
4. Course Specification – IoT Telemetry Protocol (Phase 1)