Mini-RFC: Compact Telemetry Protocol (CTP)

1. Introduction

This document specifies the Compact Telemetry Protocol (CTP), a lightweight, application-layer protocol designed for transporting simple sensor telemetry over UDP. CTP is intended for use in environments with constrained IoT devices where low power consumption, minimal bandwidth usage, and tolerance to moderate packet loss are primary design goals.

The protocol facilitates a "fire-and-forget" communication model where sensors periodically transmit small data readings to a central collector. It is intentionally designed without support for acknowledgements or retransmissions. The underlying assumption is that for many telemetry applications, such as environmental monitoring, the timely delivery of the *most recent* data is more valuable than the guaranteed delivery of every historical message. However, CTP provides the collector with sufficient metadata to detect packet loss, identify duplicate transmissions, and reorder packets that arrive out of sequence, thereby ensuring data integrity at the application level.

Assumptions and Constraints:

- **Transport:** The protocol will operate exclusively over UDP.
- **Packet Size:** CTP messages are designed to be small and fit within a single, non-fragmented UDP datagram. Application payloads are expected to be under 200 bytes.
- **Reliability:** The network is assumed to be unreliable. CTP provides mechanisms to detect losses but does not implement protocol-level recovery via retransmission.
- **Security:** This specification does not cover security aspects such as encryption or authentication, which would be handled by a higher-level framework or a secure transport layer if required.

2. Protocol Architecture

CTP is based on a simple and efficient client-server architectural model. The design intentionally places the majority of the complexity and state management on the server-side, allowing the client-side implementation to remain as simple and lightweight as possible.

Entities:

- **Sensor (Client):** This entity represents the data producer. A Sensor's role is to collect readings, construct a CTP packet, and transmit it to a Collector. It will

maintain minimal state—primarily a sequence counter that is incremented with each packet transmission. This simplicity is critical for implementation on resource-constrained microcontrollers.

- **Collector (Server):** This entity represents the data consumer. The Collector is a more powerful, stateful application that listens for incoming CTP packets on a well-known UDP port. It is responsible for parsing packets, maintaining a context for each unique communicating Sensor, and performing data integrity checks.

Sequence Flow Description:
Communication is unidirectional from the Sensor to the Collector. The Collector does not send any responses or acknowledgements back to the Sensor.

A typical operational flow for a DATA message is as follows:

1. A Sensor's reporting timer expires.
2. The Sensor gathers one or more readings from its hardware (e.g., temperature, humidity).
3. It constructs a CTP DATA packet, populating the header with its unique ID, the next sequence number, and the current timestamp. The readings are placed in the payload.
4. The Sensor transmits the UDP datagram to the Collector's pre-configured IP address and port.
5. The Collector receives the datagram, validates its structure, parses the header and payload, updates its internal state for that Sensor, and logs the telemetry data.

An alternative flow for a HEARTBEAT message occurs when a Sensor has no new data to report but needs to signal its continued operation:

1. A Sensor's liveness (heartbeat) timer expires.
2. The Sensor constructs a CTP HEARTBEAT packet, which contains only the header with the next sequence number.
3. The Sensor transmits the UDP datagram to the Collector.
4. The Collector receives the packet, notes the timestamp and sequence number to confirm the Sensor is still online, and updates its state accordingly.

Finite-State Machine (FSM) Description:
The Sensor's operational logic can be described as a single-state machine (ACTIVE) driven by two independent timers.

- **State:** ACTIVE
    - This is the only operational state for the Sensor.
- **Events and Actions:**
    - **Event:** Reporting Timer expires.
        - **Condition:** New sensor data is available.
        - **Action:**
            1. Increment the internal sequence number.
            2. Construct a DATA packet using the new sequence number and data.
            3. Transmit the packet to the Collector.
            4. Reset the Heartbeat Timer.
    - **Event:** Heartbeat Timer expires.
        - **Condition:** The Reporting Timer has not triggered data transmission recently.
        - **Action:**
            1. Increment the internal sequence number.
            2. Construct a HEARTBEAT packet.
            3. Transmit the packet to the Collector.
            4. Reset the Heartbeat Timer.

3. Message Formats

CTP messages consist of a compact, fixed-size binary header followed by an optional payload. To ensure cross-platform compatibility, all multi-byte integer fields within the header are to be encoded in network byte order (big-endian).

Header Field Table:

The CTP header is 9 bytes in total and is structured as follows:

Field	Size (bytes)	Size (bits)	Data Type	Description
DeviceID	2	16	Unsigned Integer	A unique identifier for sending sensor device. This allows the Collector to demultiplex streams.
SeqNum	2	16	Unsigned Integer	A monotonically increasing sequence number, incremented for each packet sent (both DATA and HEARTBEAT).
Timestamp	4	32	Unsigned Integer	Unix epoch seconds, indicating when the reading was taken. Used for ordering and analysis.
MsgType	1	8	Unsigned Integer	The message type. Defines how the Collector should interpret the packet.

- **Implementation Note:** The header can be serialized in Python using struct.pack('!HHIB', DeviceID, SeqNum, Timestamp, MsgType). The ! prefix ensures network byte order.

Message Type Definitions:

1. **DATA Message (MsgType 0x01)**
    - **Purpose:** To transmit one or more sensor readings.
    - **Structure:** Consists of the 9-byte CTP header followed by a variable-length payload. The format of the payload is application-specific but must be known to both the Sensor and Collector.
    - **Proposed Payload Format:** For this project's prototype, the payload will be a fixed 12 bytes, containing three 4-byte single-precision floating-point numbers representing temperature, humidity, and voltage, respectively. They will also be packed in network byte order.
        - Implementation Note: struct.pack('!fff', temp, humidity, voltage)
2. **HEARTBEAT Message (MsgType 0x02)**
    - **Purpose:** To signal to the Collector that the Sensor is still online, even when no new data is being transmitted.

    - **Structure:** Consists only of the 9-byte CTP header. This message type does **not** have a payload, minimizing its size on the wire.

Example DATA Message Trace:
Below is a byte-level example of a DATA packet.

- **Scenario:** Device 1001 sends its 50th packet at Unix time 1666885568 with readings of Temp=25.5°C, Humidity=45.2%, Voltage=4.8V.
- **Header Fields:**
    - DeviceID: 1001 (Hex: 0x03E9)
    - SeqNum: 50 (Hex: 0x0032)
    - Timestamp: 1666885568 (Hex: 0x635AE1C0)
    - MsgType: 1 (Hex: 0x01)
- **Payload Fields (as single-precision floats):**
    - Temperature: 25.5 (Hex: 0x41CCCCCD)
    - Humidity: 45.2 (Hex: 0x4234CCCC)
    - Voltage: 4.8 (Hex: 0x4099999A)
- **Final Packet (21 bytes):**
    - **Header (9 bytes):** 03 E9 00 32 63 5A E1 C0 01
    - **Payload (12 bytes):** 41 CC CC CD 42 34 CC CC 40 99 99 9A