Mini-RFC: MiniTelemetry Protocol v1.0

1. Introduction

Protocol Name: MiniTelemetry v1.0

Purpose:
MiniTelemetry is a lightweight protocol for transmitting telemetry data from IoT devices to a centralized server using UDP. It supports batching of readings, heartbeat messages for liveness detection, and handles packet loss, duplication, and reordering.

Use Case:

- Collecting temperature and humidity readings from distributed IoT devices.
- Efficient periodic reporting with minimal network overhead.

Motivation:
Existing protocols like MQTT or CoAP often rely on TCP or introduce significant overhead. MiniTelemetry provides a simple, UDP-based solution with reliability features including ACKs, retransmissions, and a reordering buffer.

Assumptions & Constraints:

- Maximum packet size ~2 KB (UDP).
- Default reporting interval is 2 seconds.
- Packet loss is tolerated; retransmission ensures eventual delivery.
- Each device has a unique `DeviceID` and sequential `SeqNum`.

2. Protocol Architecture

Entities:

1. **Client/Device:** Sends telemetry data (DATA) and heartbeat messages.

2. **Server:** Receives packets, validates sequence numbers, detects duplicates or gaps, logs readings to CSV, and sends ACKs.

Sequence Flow (ASCII Diagram):

```
Client                      Server
   |                           |
   |--- INIT ------------->|   Register device
   |                           |
   |--- DATA Batch ------>|   Check seq, detect duplicates/gaps
   |<--- ACK -------------|   Confirm reception
   |--- HEARTBEAT ------->|   Update last_seen timestamp
   |                           |
   |--- Retransmit ------>|   Resend unacknowledged packets
```

Finite-State Machine (simplified):

Client FSM:

```
[INIT] --> [DATA] --> [HEARTBEAT] --+--> [DATA]
                                    |
                                    +--> [RETRANSMIT_UNACKED]
```

Server FSM:

```
[WAIT_FOR_PACKET] --> [PARSE_HEADER] --> [CHECK_DUPLICATE/GAPS]
                    --> [ACK_PACKET] --> [STORE_IN_BUFFER]
                    --> [FLUSH_BUFFER]
```

3. Message Formats

Header (10 bytes total):

Field	Size (bits)	Description	Byte Offset
Version	4	Protocol version (v1 = 1)	0
MsgType	4	0=INIT,1=DATA,2=HEARTBEAT,3=ACK	0

DeviceID	16	Unique device identifier	1–2
SeqNum	16	Sequence number	3–4
Timestamp	32	Unix epoch seconds	5–8
BatchCnt	8	Number of readings in payload	9
Padding	16	Reserved for alignment	10–11

Payload (12 bytes per reading):

Field	Size (bits)	Description
Temp	32	Celsius
Humidity	32	% RH
TS	32	Reading timestamp

Struct packing examples:

```
# Header
      header = struct.pack("!B H H I B 2x",
                           msg_type,
                           device_id,
                           seq,
                           timestamp,
                           batch_count)
```

```
# Payload
      for (t, h, ts) in batch:
          payload += struct.pack("!f f I", t, h, ts)
```

Notes:

- `batch_count` allows multiple readings per UDP packet.
- Padding ensures proper 4-byte alignment for payload parsing.

4. Communication Procedures

Session Start (INIT):

- Device sends INIT message → Server registers device with `DeviceID` and initial `SeqNum`.

Normal Data Exchange (DATA):

1. Device collects telemetry readings and groups them into batches.
2. Device sends DATA packet with batch_count > 0.
3. Server checks for duplicates (`SeqNum <= last_seq`) and gaps (`SeqNum - last_seq > 1`).
4. Server stores readings in a reordering buffer (250 ms window) and sends ACK.

Heartbeat (HEARTBEAT):

- Sent periodically every 5 seconds.
- Server updates `last_heartbeat` timestamp to detect offline devices.

Error Recovery / Retransmission:

- Device keeps unacknowledged packets in `unacked_packets`.
- Timeout = 3 s → retransmit if ACK not received.
- Duplicate and gap flags recorded for metrics.

Session Shutdown:

- Optional; client can stop sending data. Server continues logging until process termination.

5. Reliability & Performance Features

- **Retransmission:** Timeout-based (3 s), unacked packets are resent.
- **Duplicate Detection:** `SeqNum <= last_seq` → marked as duplicate.
- **Gap Detection:** `SeqNum - last_seq > 1` → sequence gap counter incremented.
- **Reordering Buffer:** 250 ms window to sort out-of-order readings before CSV storage.
- **Metrics Collected:**
    - Bytes per report
    - Packets received
    - Duplicate rate

- Sequence gaps
- CPU time per report

6. Experimental Evaluation Plan

Metrics:

- Duplicate rate, sequence gap count, bytes per report, CPU per report.

Baselines:

- Compare single-reading packets vs. batch packets (`--batch` parameter).

Network Simulation (Linux netem):

 # baseline

./test_netem.sh baseline

Results :

=== TEST COMPLETED ===

=== Metrics Summary ===

bytes_per_report: 20.00

packets_received: 14

duplicate_rate: 0.000

sequence_gap_count: 0

cpu_ms_per_report: 0.616

5% packet loss :

```
./test_netem.sh loss
sudo tc qdisc add dev $IFACE root netem loss 5%
```

Results :

```
=== TEST COMPLETED ===

=== Metrics Summary ===
bytes_per_report: 19.07
packets_received: 33
duplicate_rate: 0.364
sequence_gap_count: 2
cpu_ms_per_report: 0.126
```

100 ms delay with 10 ms jitter

```
sudo tc qdisc add dev $IFACE root netem delay 100ms 10ms
```

Results :

```
=== TEST COMPLETED ===

=== Metrics Summary ===
bytes_per_report: 19.90
packets_received: 27
duplicate_rate: 0.000
sequence_gap_count: 0
cpu_ms_per_report: 0.427
```

Cpu usage is higher because of delay

7. Example Use Case Walkthrough

- **Session Start:**
    - Device 103 sends INIT (Seq 1) → Server registers.
- **Data Batch:**
    - Device sends batch of 10 readings:

```
100,11,1766427006,1766427024.942385,0,0,26.92,69.19
100,11,1766427008,1766427024.942385,0,0,24.99,66.09
100,11,1766427010,1766427024.942385,0,0,28.93,53.59
100,11,1766427012,1766427024.942385,0,0,25.38,56.46
100,11,1766427014,1766427024.942385,0,0,21.93,67.4
100,11,1766427016,1766427024.942385,0,0,31.48,49.79
100,11,1766427018,1766427024.942385,0,0,34.04,48.27
100,11,1766427020,1766427024.942385,0,0,32.15,43.21
100,11,1766427022,1766427024.942385,0,0,32.83,52.28
100,11,1766427024,1766427024.942385,0,0,30.98,59.49
```

```
 4 heartbeats then a 10 readings batch
```

    - Server buffers readings, sorts by payload timestamp, writes to CSV.
- **Heartbeat:**
    - Device sends heartbeat every 5 s → Server updates last_seen.

8. Limitations & Future Work

- Only supports one-way telemetry; no full TCP-like reliability.
- No encryption or authentication.
- Heartbeat only indicates liveness, not full connection health.
- Future improvements:
    - Adaptive batching
    - Forward Error Correction (FEC)
    - Secure transport (TLS/DTLS)
    - Configurable reorder window

9. References

1. RFC 768 – User Datagram Protocol (UDP)
2. MQTT Protocol Specification
3. Python `struct` documentation: https://docs.python.org/3/library/struct.html
4. Linux `tc`/netem: https://wiki.linuxfoundation.org/networking/netem