Computer Networks Major Task

Phase_2(MINI RFC)

Submitted by (Team N0.13):

Abdelrhman Mohammed Mahmoud	23P0370
Abdelrahman Mohamed Bahig	23P0390
Ahmed Mostafa Gomaa Atia	23p0375
Nagy Ahmed Nagy	23p0365
Abdallah Omar Moustafa Taman	22P0129
Kareem Younis Ahmed Foad Yehya	22P0136

Contents

4. Communication Procedures

This section describes the operational flow of a LiteTelemetry session, including session start, regular data exchange, heartbeat handling, error detection, and session shutdown.

4.1 Session Start

A new device session begins when a SensorClient initializes communication with the CollectorServer.
LiteTelemetry does not establish a formal connection (since it is UDP-based); instead, sessions are inferred from the first received packet.

Session-start sequence:

1. **Client boots / application starts**
2. Client assigns:
    - device_id
    - seq = 0 (DATA only)
3. Client spawns a heartbeat timer
4. Client sends first DATA packet:
5. Header:
6. device_id = X
7. seq = 0
8. timestamp = T0
9. msg_type = DATA
10. batch_count = N
11. Payload:
12. N float32 readings
13. Server creates device entry in internal structures:
    - last_seq
    - recent_seq_window
    - last_heartbeat_time

4.2 Normal Data Exchange

During operation, the client alternates between producing sensor readings and transmitting them at the configured interval.

Normal DATA exchange:

1. Client gathers batch_size readings
2. Client sends DATA packet with:
    - incremented seq
3. Server receives packet:
    - Parses header
    - Checks for duplicates
    - Checks for gaps
    - Logs packet to telemetry_log.csv
    - Pushes packet into reorder buffer
4. Server reordering thread periodically sorts buffer by timestamp
5. Reordered data is written to telemetry_reordered.csv

4.3 HEARTBEAT Handling

HEARTBEAT messages are used for liveness tracking only.

- Sent every heartbeat_interval seconds
- **Does NOT increment sequence number**
- Contains no payload
- Server marks device as active upon receipt

Heartbeat handling sequence:

1. Client timer fires (e.g., every 5 seconds)
2. Client sends HEARTBEAT with the current sequence number
3. Server updates:
    - last_heartbeat_time
    - heartbeat_flag in logs

If no heartbeat is received within a timeout window (e.g., 3× heartbeat interval), the server marks the device as **offline**.

4.4 Error Recovery

LiteTelemetry does not use retransmissions (consistent with IoT loss-tolerance and UDP). Instead, the server provides error detection:

- **Duplicate suppression:**
  If seq ∈ recent-seq-window → packet is ignored.
- **Gap detection:**
  If seq > last_seq + 1 → gap detected.
  Logged in CSV as gap_flag = 1.
- **Reordering:**
  Delay and jitter do not affect chronological ordering because server sorts packets by timestamp.

There is **no retransmission, windowing, or ACK-based feedback**.

Gaps are reported but not corrected.

4.5 Session Shutdown

A session ends when the client application stops.

Shutdown sequence:

1. Client sets running = False
2. Client's heartbeat loop terminates
3. Server stops receiving packets and will later mark device offline
4. Reorder buffer flushes final entries

No explicit FIN or teardown packet is used.

5. Reliability & Performance Features

LiteTelemetry prioritizes **efficiency** and **predictable resource usage** over strong reliability guarantees.
This section defines the mechanisms used.

5.1 Duplicate Suppression

Server uses a fixed-size sliding window of recent sequence numbers (deque).
If a packet's seq is already in the deque → it is marked duplicate and ignored.

Deque size defaults to **50** packets.

5.2 Gap Detection

A gap is declared when:

if seq > last_seq + 1:

gap_flag = 1

Gaps occur when:

- Packets are lost
- Packets arrive extremely delayed
- Heartbeat sequence was incorrectly incremented (now fixed)

5.3 Reordering Buffer

Reordering uses timestamps:

- Received packets are inserted into a buffer
- Every 5 seconds, the buffer is sorted by timestamp
- Output goes to telemetry_reordered.csv

This mitigates jitter or delay caused by network impairment.

5.4 Heartbeat Timeout

Server tracks time since last heartbeat:

offline if (now - last_heartbeat_time) > HEARTBEAT_TIMEOUT

Default:

HEARTBEAT_TIMEOUT = 3 × heartbeat_interval

5.5 Timer Rationale

No RTT-based timers or retransmission timeouts are used.
However, timing effects considered:

- Heartbeat interval chosen to ensure minimal false offline detection.
- Reorder flush interval chosen to fit worst-case jitter in experiments (100ms ±10ms).

5.6 Not Implemented (By Design)

To maintain low overhead:

- No retransmission strategies
- No congestion window
- No flow control
- No rate adaptation
- No FEC (Forward Error Correction)

These can be added in future versions.

6. Experimental Evaluation Plan

This section describes how LiteTelemetry is tested under real network impairment.

6.1 Baseline Conditions

Baseline scenario uses:

- No loss
- No delay
- Reporting interval = 1 second
- batch = 1

Expected behavior:

- No gaps
- No duplicates
- Stable timestamps

6.2 Metrics Collected

The server generates metrics.json including:

- packets_received
- reads_processed
- bytes_received
- bytes_per_report
- duplicates

- duplicate_rate
- gaps
- cpu_ms_per_report

6.3 Measurement Method

Metrics come from server-side counters.
Packet traces may be captured with tcpdump for validation.

Example:

sudo tcpdump -i lo -w results/baseline/session.pcap

6.4 Network Impairment Simulation (Linux netem)

Since communication is via 127.0.0.1, impairments are applied to loopback (lo).

Reset

sudo tc qdisc del dev lo root

5% Packet Loss

sudo tc qdisc add dev lo root netem loss 5%

100ms Delay ±10ms Jitter

sudo tc qdisc add dev lo root netem delay 100ms 10ms

Reset after scenario

sudo tc qdisc del dev lo root

6.5 Automated Scenario Script

The provided test_phase2.py script:

- Starts server
- Applies netem
- Runs wrapper client
- Saves results into results/<scenario>/
- Clears netem after each test

Scenarios included:

Scenario	Params
baseline_1s	interval=1, batch=1
loss_5pct	loss=5%
delay_100ms_10ms	delay+jitter
interval_5s	interval=5
interval_30s	interval=30
batch_5	batch=5
batch_10	batch=10

7. Example Use Case Walkthrough

Below is an end-to-end trace illustrating a typical LiteTelemetry session.

7.1 Scenario Setup

- device_id = 1024
- reporting interval = 1s
- batch size = 1
- Server running on localhost (127.0.0.1)

7.2 Sequence of Events

t = 0s — Client Starts

Client generates reading:

25.31°C

Client sends:

DATA:

device_id = 1024

seq = 0

timestamp = 1700000000

batch_count = 1

payload = [25.31]

Server logs:

1024, 0, 1700000000, arrival_time, dup=0, gap=0, hb=0

t = 1s — Second DATA

seq = 1

reading = 25.77

Server:

dup=0, gap=0

t = 5s — HEARTBEAT

msg_type = HEARTBEAT

seq = 5

Server marks device alive.

t = 8s — Artificial delay (netem delay scenario)

Packet arrives late; server reorders by timestamp:

Packet seq=7 inserted earlier in timeline

t = 10s — Session End

Client sets running = False.

7.3 Reordered Output Example

Contents of telemetry_reordered.csv:

device_id, seq, timestamp, readings

1024, 0, 1700000000, 25.31

1024, 1, 1700000001, 25.77

1024, 2, 1700000002, 25.48

7.4 PCAP Excerpt (For Appendix)

Example tcpdump ASCII:

IP 127.0.0.1.56000 > 127.0.0.1.5555: UDP, length 16

device_id=1024 seq=3 timestamp=1700000003