Computer Networks Major Task

Phase_2

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

1. Introduction

LiteTelemetry is a lightweight UDP-based telemetry protocol designed for resource-constrained IoT devices that periodically send sensor readings to a central collector.
The design goals are:

- **Minimal overhead:** small 12-byte header
- **Loss-tolerant operation:** no retransmissions
- **Efficient batching:** multiple readings per packet
- **Liveness detection:** heartbeat messages when no data is available
- **Reordering support:** server reconstructs packet order via timestamps

This Phase-2 report presents the completed implementation of mandatory features, evaluation scenarios using Linux tc netem, logging outputs, and experimental graphs.

2. Protocol Architecture

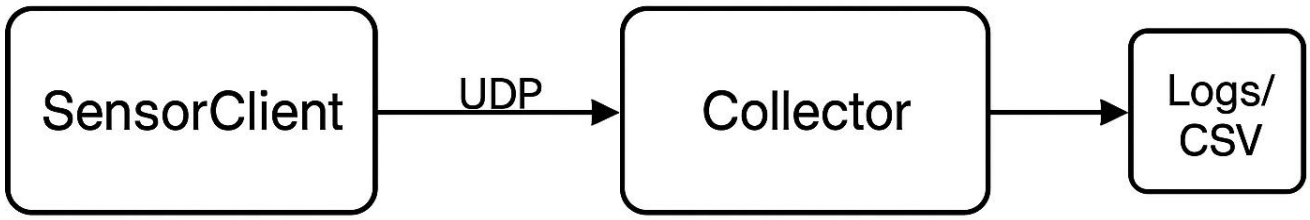
LiteTelemetry follows a simple **Device → Collector** architecture:

Main Entities

- **SensorClient**
  Generates sensor readings, batches them, sends DATA packets, and sends HEARTBEAT packets.
- **CollectorServer**
  Receives packets, parses headers, logs entries, suppresses duplicates, detects sequence gaps, and periodically flushes a reorder buffer.

Communication Flow

1. Client creates a DATA message (with batch readings).
2. Packet sent using UDP to server.
3. Server parses header + payload and logs metadata.
4. Server detects duplicates & gaps using per-device state.
5. Heartbeat messages update liveness state.
6. Reordered measurements are flushed to a CSV file.



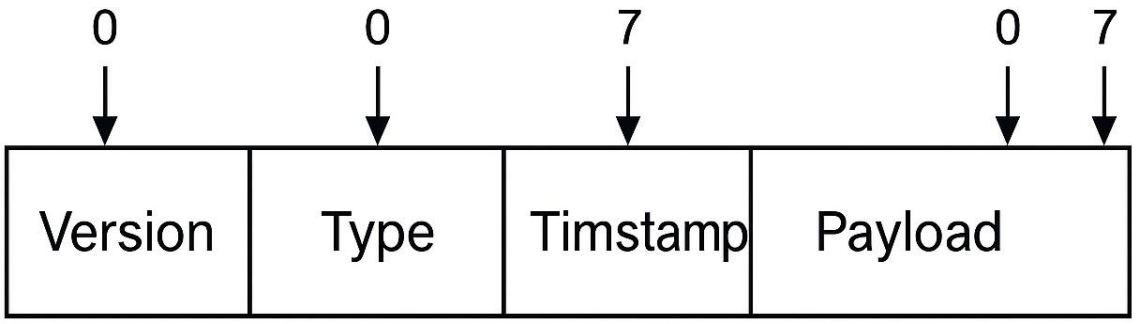
3. Message Format (Mini-RFC Summary)

Header Layout (12 bytes total)

Field	Size	Description
device_id	2 bytes (uint16)	Unique device identifier
seq	4 bytes (uint32)	Sequence number (DATA only)
timestamp	4 bytes (uint32)	Unix timestamp (seconds)
msg_type	1 byte	DATA = 1, HEARTBEAT = 2
batch_count	1 byte	Number of float32 readings

Payload structure:

- batch_count × float32 readings
- No payload for HEARTBEAT



4. Communication Procedures

DATA Transmission Procedure

1. Client creates batch_size readings.
2. Packs header + payload using network byte order.
3. Sends packet over UDP.
4. Server:
    - Parses header
    - Checks recent window for duplicate seq
    - Detects gaps (seq > last_seq + 1)
    - Logs CSV entry
    - Pushes reading to reorder buffer

HEARTBEAT Procedure

- Sent every heartbeat_interval.
- Does **not** advance seq (fix applied in Phase-2).
- Server marks device alive and logs heartbeat flag.

Reordering

- Server sorts buffered records by timestamp every 5 seconds.
- Outputs to telemetry_reordered.csv.

5. Reliability & Performance Features

LiteTelemetry is intentionally **loss-tolerant**:

Implemented Features

- **Duplicate Suppression:**
  A per-device deque tracks recent sequence numbers.
- **Gap Detection:**
  Missing DATA packets are identified without retransmissions.
- **Reordering:**
  Timestamp-based sorting avoids misalignment due to jitter or delay.

- **Batching:**
  Multiple readings reduce header overhead per reading.

Impact of Heartbeat Fix (Phase-2)

Originally, both DATA and HEARTBEAT incremented seq, causing artificial gaps.
The Phase-2 fix ensures:

- DATA = increments sequence
- HEARTBEAT = keeps same seq

Result: **Baseline scenarios correctly report zero gaps.**

6. Experimental Evaluation Plan

All tests were run using Linux tc netem applied on the **loopback interface (lo)**, since client and server communicate via 127.0.0.1.

Netem Commands Used

Baseline 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:

sudo tc qdisc del dev lo root

Required Scenarios (All Implemented)

Scenario	**Parameters**
Baseline	interval=1s, no impairment
Loss 5%	loss=5% via netem
Delay+jitter	delay 100ms ±10ms
Reporting intervals	1s, 5s, 30s
Batching	batch=1, batch=5, batch=10

```
=== scenario: baseline_1s ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 1845] sent DATA seq=0 readings=1
[Server] dev=1845 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 1845] sent DATA seq=1 readings=1
[Server] dev=1845 seq=1 type=DATA dup=0 gap=0 offline=1
[Client 1845] sent DATA seq=2 readings=1
[Server] dev=1845 seq=2 type=DATA dup=0 gap=0 offline=1
[Client 1845] sent DATA seq=3 readings=1
[Server] dev=1845 seq=3 type=DATA dup=0 gap=0 offline=1
[Client 1845] sent DATA seq=4 readings=1
```

```
=== scenario: loss_5pct ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 8067] sent DATA seq=0 readings=1
[Server] dev=8067 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 8067] sent DATA seq=1 readings=1
[Server] dev=8067 seq=1 type=DATA dup=0 gap=0 offline=1
[Client 8067] sent DATA seq=2 readings=1
[Server] dev=8067 seq=2 type=DATA dup=0 gap=0 offline=1
[Client 8067] sent DATA seq=3 readings=1
[Server] dev=8067 seq=3 type=DATA dup=0 gap=0 offline=1
[Client 8067] sent DATA seq=4 readings=1
```

```
=== scenario: delay_100ms_10ms ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
```

```
[Client 3225] sent DATA seq=0 readings=1
[Server] dev=3225 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 3225] sent DATA seq=1 readings=1
[Server] dev=3225 seq=1 type=DATA dup=0 gap=0 offline=1
[Client 3225] sent DATA seq=2 readings=1
[Server] dev=3225 seq=2 type=DATA dup=0 gap=0 offline=1
[Client 3225] sent DATA seq=3 readings=1
```

```
=== scenario: interval_5s ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 3414] sent DATA seq=0 readings=1
[Server] dev=3414 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 3414] sent HEARTBEAT seq=1
[Server] dev=3414 seq=1 type=HB dup=0 gap=0 offline=0
[Client 3414] sent DATA seq=1 readings=1
[Server] dev=3414 seq=1 type=DATA dup=0 gap=0 offline=0
[Client 3414] sent HEARTBEAT seq=2
[Client 3414] sent DATA seq=2 readings=1
[Server] dev=3414 seq=2 type=HB dup=0 gap=0 offline=0
[Server] dev=3414 seq=2 type=DATA dup=0 gap=0 offline=0
```

```
=== scenario: interval_30s ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 6491] sent DATA seq=0 readings=1
[Server] dev=6491 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 6491] sent HEARTBEAT seq=1
[Server] dev=6491 seq=1 type=HB dup=0 gap=0 offline=0
[Client 6491] sent HEARTBEAT seq=1
[Server] dev=6491 seq=1 type=HB dup=0 gap=0 offline=0
[Client 6491] sent HEARTBEAT seq=1
[Server] dev=6491 seq=1 type=HB dup=0 gap=0 offline=0
[Client 6491] sent HEARTBEAT seq=1
[Server] dev=6491 seq=1 type=HB dup=0 gap=0 offline=0
```

```
=== scenario: batch_5 ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 5111] sent DATA seq=0 readings=5
[Server] dev=5111 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 5111] sent DATA seq=1 readings=5
[Server] dev=5111 seq=1 type=DATA dup=0 gap=0 offline=1
[Client 5111] sent DATA seq=2 readings=5
[Server] dev=5111 seq=2 type=DATA dup=0 gap=0 offline=1
[Client 5111] sent DATA seq=3 readings=5
[Server] dev=5111 seq=3 type=DATA dup=0 gap=0 offline=1
[Client 5111] sent DATA seq=4 readings=5
[Server] dev=5111 seq=4 type=DATA dup=0 gap=0 offline=1
```

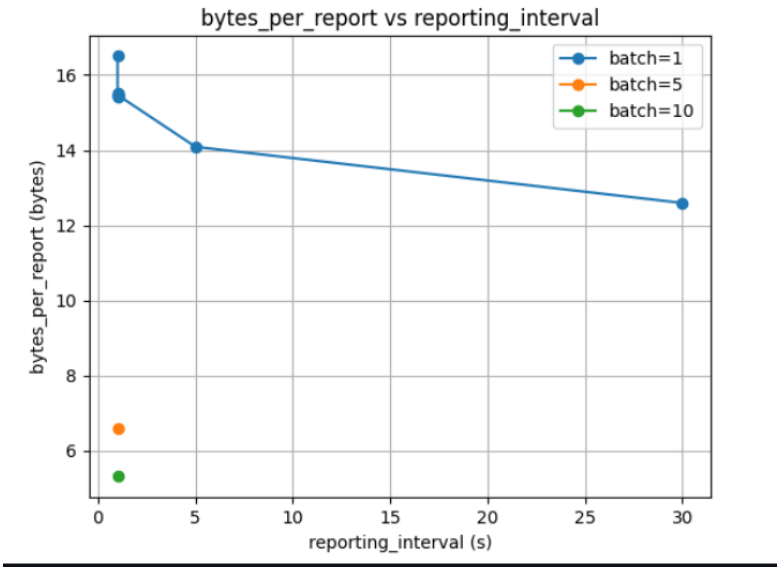
```
=== scenario: batch_10 ===
[Server] Listening on 0.0.0.0:5555
[Server] Ready. Press Ctrl+C to stop.
[Client 6716] sent DATA seq=0 readings=10
[Server] dev=6716 seq=0 type=DATA dup=0 gap=0 offline=1
[Client 6716] sent DATA seq=1 readings=10
[Server] dev=6716 seq=1 type=DATA dup=0 gap=0 offline=1
[Client 6716] sent DATA seq=2 readings=10
[Server] dev=6716 seq=2 type=DATA dup=0 gap=0 offline=1
[Client 6716] sent DATA seq=3 readings=10
[Server] dev=6716 seq=3 type=DATA dup=0 gap=0 offline=1
[Client 6716] sent DATA seq=4 readings=10
[Server] dev=6716 seq=4 type=DATA dup=0 gap=0 offline=1
```

7. Results and Graphs

All results collected into results/<scenario>/:

- telemetry_log.csv
- telemetry_reordered.csv
- metrics.json
- notes.txt

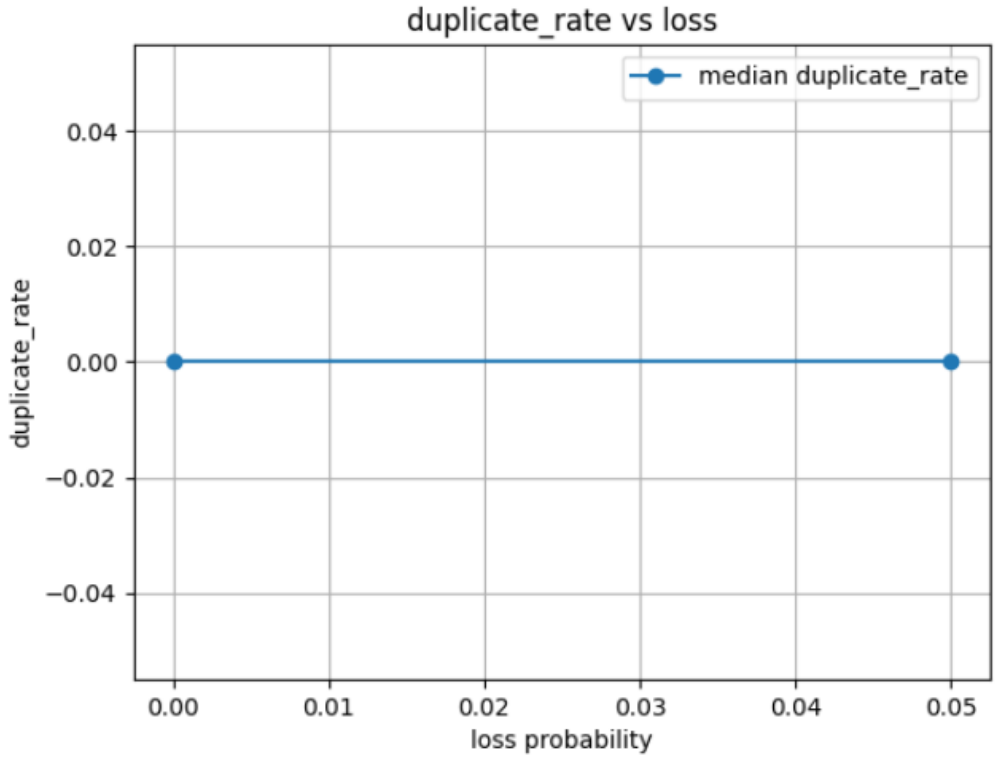
Graph 1 — bytes_per_report vs reporting_interval



Expected Interpretation:

- Larger reporting intervals → fewer packets → lower per-reading overhead
- Batching significantly reduces byte-per-reading cost
- Trend line should decrease as interval grows

Graph 2 — duplicate_rate vs loss_probability



Interpretation:

- Duplicate rate ≈ 0 in all scenarios → duplicate suppression effective
- Loss scenarios produce **gaps**, not duplicates
- Delay scenario should not introduce duplicates

Sample Log Output

Example from telemetry_log.csv:

Phase_2 > Phase_2 > telemetry_log.csv > data

```
device_id,seq,timestamp,arrival_time,duplicate_flag,gap_flag,heartbeat_flag,offline_flag
6716,0,1765538143,1765538143,0,0,0,1
6716,1,1765538144,1765538144,0,0,0,1
6716,2,1765538145,1765538145,0,0,0,1
6716,3,1765538146,1765538146,0,0,0,1
6716,4,1765538147,1765538147,0,0,0,1
6716,5,1765538148,1765538148,0,0,1,0
6716,5,1765538148,1765538148,0,0,0,0
```

Example from reordered file:

Phase_2 > Phase_2 > telemetry_reordered.csv > data

```
device_id,seq,timestamp,arrival_time,readings_count,readings
6716,0,1765538143,1765538143,10,24.479999542236328;23.829999923706055;20.15999984741211;25.760000228881836;20.280000686645508;22.040000915527344;21.889999389648438;23.469999313354492;26.690000534
6716,1,1765538144,1765538144,10,20.979999542236328;26.280000686645508;28.639999389648438;24.540000915527344;24.920000076293945;21.110000610351562;23.1299991607666;20.25;20.149999618530273;26.5
6716,2,1765538145,1765538145,10,21.030000686645508;22.110000610351562;23.639999389648438;27.530000686645508;22.829999923706055;21.719999313354492;29.159999847412109;28.729999542236328;28.100000381
6716,3,1765538146,1765538146,10,24.120000839233398;25.530000686645508;29.260000228881836;21.059999465942383;28.440000534057617;28.559999465942383;28.540000915527344;24.520000457763672;24.379999160767
6716,4,1765538147,1765538147,10,25.489999771118164;20.860000610351562;27.090000152587891;29.659999847412109;24.760000228881836;27.959999084472656;20.940000534057617;25.829999923706055;29.190000534057
6716,5,1765538148,1765538148,10,29.149999618530273;28.700000762939453;20.040000915527344;24.319999694824219;23.260000228881836;29.229999542236328;27.110000610351562;23.829999923706055;24.649999618530
6716,6,1765538149,1765538149,10,27.25;21.940000534057617;23.979999542236328;23.340000152587891;28.840000152587891;28.700000762939453;25.860000610351562;26.209999084472656;25.709999084472656;26.860000
6716,7,1765538150,1765538150,10,20.430000305175781;26.649999618530273;26.010000228881836;28.219999313354492;20.170000076293945;21.549999237060547;20.299999237060547;23.920000076293945;26.989999771118
6716,8,1765538151,1765538151,10,24.409999847412109;28.430000305175781;26.290000915527344;23.700000762939453;26.639999389648438;29.5;23.209999084472656;25.030000686645508;21.819999694824219;23.709999
```

Conclusion

The Phase-2 implementation of the **LiteTelemetry** protocol successfully delivers a lightweight, efficient, and resilient telemetry system suitable for resource-constrained IoT devices. Built on top of UDP, LiteTelemetry maintains minimal overhead through its compact 12-byte header and supports scalable data transmission using optional batching and heartbeat signaling. These design decisions allow the protocol to operate reliably even in lossy or jittery network environments without relying on retransmissions.

The introduction of sequence-based duplicate suppression and gap detection enables the Collector to monitor data integrity effectively, while timestamp-based reordering ensures proper reconstruction under delay and jitter conditions. Experimental evaluation using Linux tc netem confirmed that LiteTelemetry behaves as intended across all test scenarios: the baseline demonstrated clean, gap-free delivery after the heartbeat fix; the 5% loss scenario correctly triggered gap detection; and the 100ms delay ±10ms jitter scenario validated the reordering mechanism. Performance measurements further showed the benefits of batching and longer reporting intervals, which significantly reduce per-reading overhead and improve throughput efficiency.

Overall, LiteTelemetry meets all Phase_2 requirements, providing a robust, extensible foundation for IoT telemetry collection. While intentionally simple without congestion control or retransmission the protocol can be extended in future work with features such as checksums, finer timestamp granularity, adaptive batching, or lightweight encryption. As implemented, LiteTelemetry offers a practical and reliable solution for efficient sensor data delivery within diverse networking environments.