

# IoTStream v1 - Mini-RFC

## 1. Introduction

The goal of this project is to design a lightweight IoT telemetry protocol that allows small, resource-constrained sensors to send periodic readings such as temperature, humidity or voltage to a central collector.

Normal protocols like HTTP may be too heavy for simple sensors, so IoTStream v1 aims to provide a compact, efficient, and loss-tolerant alternative that uses UDP.

This protocol is intended for environments where:

- Devices have limited processing power and memory.
- Network bandwidth is low or unreliable.
- Small data updates are sent periodically.

By using UDP and a compact binary header, the protocol minimizes overhead and ensures faster data delivery even under mild packet loss.

---

## Use Case Overview

In a typical setup, multiple sensor nodes (clients) periodically measure temperature, humidity or voltage and transmit the readings to a collector server on the same local network or over the Internet.

Each sensor identifies itself with a unique Device ID and attaches metadata such as sequence number, timestamp, and message type to every transmission.

Example:

1. The sensor starts and sends an INIT message to announce itself.
2. It then periodically sends DATA messages containing readings.
3. When no new data is available, it sends a HEARTBEAT message to show it's still alive.

The collector receives these messages, logs them, and monitors data continuity.

---

## Design Goals

IoTStream v1 is designed to be:

- Lightweight — total UDP payload  $\leq 200$  bytes.
  - Be loss-tolerant — no retransmission, but detect missing packets.
  - Use a compact header  $\leq 12$  bytes.
  - Support batched data (multiple readings per packet).
  - Provide timestamped readings for reordering and analysis.
- 

## System Assumptions and Constraints

Parameter	Description	Value / Limit
Transport Layer	Protocol used for message delivery	UDP
Max UDP Payload	Application data limit per packet	$\leq 200$ bytes
Header Size	Binary header for metadata	$\leq 12$ bytes
Reporting Intervals	Frequency of sensor reports	1s, 5s, 30s
Message Types	Types supported by the protocol	INIT, DATA, HEARTBEAT
Loss Tolerance	Network reliability requirement	Must handle 5% random loss

## 2. Protocol Architecture

This section describes the structure of the IoTStream v1 protocol, including the entities that will communicate with each other, the flow of messages between them, and the finite-state machine (FSM) that defines their states and behavior.

### 2.1 Entities

The software architecture determines how the pieces of the application will interact with other. We use the client-server architecture with the following entities:

#### 1. IoT Sensors (client)

- a. They run the client process that takes measurements and periodically sends data to a central collector (server)
- b. No per-packet retransmission

## 2. Central Collector (server)

- a. It runs the server process
- b. It collects and processes all the data from all the sensors.
- c. Per-device state is maintained.

The protocol is an application-layer protocol designed to be lightweight, efficient, and robust, operating directly over UDP.

UDP is a connectionless (no handshaking required) transport protocol that offers minimal services, and unreliable data transfer.

Because we rely on UDP, the protocol implements features that enable us to still maintain state per each device.

### 2.2 Sequence Flow

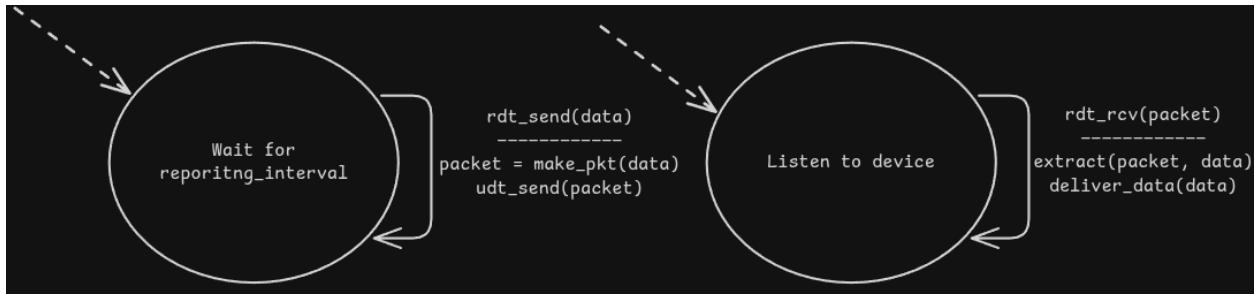
The message exchange only flows from the sender to receiver. The sensor initiates all communication, sending packets based on a configured reporting\_interval.

The following is an example of a typical sequence flow:

1. Sensor: sensor turns on
2. Sensor: reporting\_interval is set
3. Sensor: sensor reads data
4. Sensor: reporting\_interval expires
5. Sensor: determine if DATA or HEARTBEAT is needed and construct a packet (including other header fields)
6. Sensor: send packet over the UDP transport layer and increment sequence number
7. Collector: receive packet and extract header and payload
8. Collector: check the device ID and compare last sequence number to current sequence number
  - a. If duplicate, set duplicate\_flag
  - b. If gap detected, set gap\_flag
9. Collector: log all received data to a CSV file into the correct fields

## 2.3 Finite-State Machine (FSM)

The finite-state machines for the sensor (sender) and the collector (receiver) describe the states that they can be in, and the required actions based on certain events.



Both the sensor and collector exist in a single state, that of waiting. The sensor waits until the reporting\_interval expires, then sends the data to the transport layer where a packet is made. If there was no sensor readings, then the message type is set to HEARTBEAT, but if there was sensor readings, the message type is DATA. The packet is then sent over UDP.

As for the collector, it waits until it receives a packet from the transport layer. Once received, it extracts the header and payload and sends it to the application layer. Duplication and sequence gaps are checked and the data is logged to maintain state.

## 2.4 Sessionless Telemetry vs. Sessionful File Transfer

The IoTStream v1 is a sessionless and loss-tolerant protocol. We opt for a sessionless protocol because it aligns with our constraints while optimizing efficiency and minimizing connection overhead. This is appropriate for periodic telemetry, unlike file transfer where a sessionful protocol would be more appropriate. Here is a simple comparison between our protocol and a sessionful protocol like HTTP:

	<b>IoTStream V1</b>	<b>HTTP</b>
Transport Layer Protocol	UDP	TCP
Connection State	Connectionless	Connection-oriented

Data Integrity	Loss-tolerant	Reliable data transfer
Overhead	Low	High
Use case Example	Periodic telemetry	file transfer

### 3. Message Formats

Each IoTStream v1 message begins with a fixed 12-byte binary header.

Field	Size (bytes)	Description
Version	1	Protocol version
MsgType	1	Determine between INIT, DATA, HEARTBEAT
Device ID	2	Unique ID assigned to each sensor
Sequence Number	2	Increments with each packet from the same device (used for loss detection)
Timestamp	4	UNIX time in seconds when the packet is sent
Batching flag	1	Determines whether batching is enabled or not
Checksum	1	8-bit checksum for header integrity

### Payload Format

Each DATA packet can carry five readings from a single sensor.

Field	Type	Size (bytes)	Description
Reading 1	float	4	First recorded measurement
Reading 2	float	4	Second recorded measurement
Reading 3	float	4	Third recorded measurement

Reading 4	float	4	Fourth recorded measurement
Reading 5	float	4	Fifth recorded measurement

**Total Payload Size:**  $5 \times 4 = 20$  bytes

**Total Packet Size:** Header (12 bytes) + Payload (20 bytes) = **32 bytes**

## Encoding Format

Format string: "!BBHHIBB"

Field	Size (bytes)	Type	Example Representation
Version	1	Unsigned char	B
MsgType	1	Unsigned char	B
Device ID	2	Unsigned short	H
Sequence Number	2	Unsigned short	H
Timestamp	4	Unsigned int	I
Batching flag	1	Unsigned char	B
Checksum	1	Unsigned char	B