Request for Comments: DCT-001

Data Collection Telemetry (DCT) Protocol

Specification Version 1.0

DCT Protocol Working Group
dct-protocol@example.com

December 2025

Status of This Memo

This document specifies a protocol for the transmission of telemetry data from IoT devices to collection servers. Distribution of this memo is unlimited.

Abstract

The Data Collection Telemetry (DCT) Protocol is a lightweight, binary protocol designed for efficient transmission of sensor and telemetry data from resource-constrained IoT devices to centralized collection servers. This specification defines the message formats, transmission procedures, and operational semantics of the protocol. DCT operates over UDP and employs delta compression, batching, and efficient binary encoding to minimize bandwidth consumption while providing mechanisms for loss detection and device management.

Contents

1 Introduction

1.1 Purpose

The Data Collection Telemetry (DCT) Protocol addresses the need for an efficient, lightweight protocol suitable for transmitting periodic sensor readings from IoT devices. Unlike text-based protocols such as HTTP/JSON or verbose binary protocols, DCT is optimized for:

- Minimal header overhead (8 bytes fixed header)
- Delta compression for slowly-changing values
- Optional batching to amortize header costs
- Simple implementation suitable for microcontrollers

1.2 Scope

This specification defines:

- Message format and encoding
- Connection establishment and teardown
- Data transmission modes
- Sequence number handling
- Time synchronization
- Error detection mechanisms

1.3 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Device
: An IoT sensor or data source that transmits telemetry data.

Server
: The collection endpoint that receives and processes telemetry.

Session
: A logical connection between a device and server.

Keyframe
: An absolute value transmission.

Delta
: A relative value transmission (difference from previous).

Batch
: Multiple data points aggregated in a single packet.

2 Protocol Overview

2.1 Transport

DCT operates over the User Datagram Protocol (UDP) as defined in RFC 768. The default port number is **5000**, though implementations **MAY** use alternative ports.

2.2 Byte Order

All multi-byte numeric fields in DCT messages **MUST** be transmitted in network byte order (big-endian).

2.3 Session Model

DCT uses a soft-state session model:

1. Device initiates session with STARTUP message
2. Server responds with STARTUP_ACK containing Device ID
3. Device transmits data using assigned Device ID
4. Either party may terminate with SHUTDOWN
5. Sessions timeout after prolonged inactivity

3 Message Format

3.1 Common Header

All DCT messages share a common 8-byte header:

Table 1: DCT Message Header Format

Byte Offset							
0	1	2	3	4	5	6	7
Ver/Type	Device ID		Sequence		Time Offset		Length

3.1.1 Version/Type Field (1 byte)

- **Bits 7-4**: Protocol version (current: 0x01)
- **Bits 3-0**: Message type identifier

Combined as: `(version « 4) | type`

3.1.2 Device ID (2 bytes)

A 16-bit unsigned integer assigned by the server during registration. Devices **MUST** use 0x0000 in STARTUP messages before receiving their assigned ID.

3.1.3 Sequence Number (2 bytes)

A 16-bit unsigned integer that **MUST** be incremented for each message sent by a device. The sequence number wraps from 65535 to 0.

3.1.4 Time Offset (2 bytes)

A 16-bit unsigned integer representing seconds elapsed since the base timestamp established during TIME_SYNC. Maximum representable offset is 65535 seconds (18.2 hours).

3.1.5 Payload Length (1 byte)

An 8-bit unsigned integer specifying the number of payload bytes following the header. Maximum payload size is 255 bytes.

3.2 Struct Format

In Python struct notation, the header format is:

```
!BHHHB
```

Where:

- `!` = Network byte order (big-endian)
- `B` = Unsigned char (Version/Type)
- `H` = Unsigned short (Device ID)
- `H` = Unsigned short (Sequence)
- `H` = Unsigned short (Time Offset)
- `B` = Unsigned char (Length)

4 Message Types

Table 2: DCT Message Types

Value	Name	Description
0x01	MSG_STARTUP	Device registration request
0x02	MSG_STARTUP_ACK	Server registration acknowledgment
0x03	MSG_TIME_SYNC	Time base synchronization
0x04	MSG_KEYFRAME	Absolute value transmission
0x05	MSG_DATA_DELTA	Delta value transmission
0x06	MSG_HEARTBEAT	Keep-alive signal
0x07	MSG_BATCHED_DATA	Aggregated data transmission
0x0B	MSG_SHUTDOWN	Session termination
0x0C	MSG_BATCH_INCOMPLETE	Incomplete batch marker

4.1 MSG_STARTUP (0x01)

Sent by a device to initiate a session with the server.

```
Header:
  Version/Type: 0x11 (version 1, type 1)
  Device ID:    0x0000 (not yet assigned)
  Sequence:     0x0000 (initial)
  Time Offset:  0x0000
  Length:       6 or 7
```

```
Payload:
  Bytes 0-5:  MAC Address (6 bytes)
  Byte 6:     Batch Size (optional, 1 byte)
```

If Batch Size is provided and greater than 1, the device requests batched transmission mode.

4.2 MSG_STARTUP_ACK (0x02)

Sent by the server to acknowledge device registration.

```
Header:
  Version/Type: 0x12 (version 1, type 2)
  Device ID:    Assigned device ID
  Sequence:     0x0000
  Time Offset:  0x0000
  Length:       2 or 4

Payload (New Device):
  Bytes 0-1:  Assigned Device ID (2 bytes)

Payload (Reconnecting Device):
  Bytes 0-1:  Device ID (2 bytes)
  Bytes 2-3:  Last Known Sequence (2 bytes)
```

4.3 MSG_TIME_SYNC (0x03)

Establishes the base timestamp for relative time calculations.

```
Header:
  Version/Type: 0x13 (version 1, type 3)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  0x0000
  Length:       4

Payload:
  Bytes 0-3:  Unix Timestamp (4 bytes, unsigned)
```

Devices **SHOULD** send TIME_SYNC:

- Immediately after receiving STARTUP_ACK
- Periodically (recommended: every hour)
- When time offset approaches 65535

4.4 MSG_KEYFRAME (0x04)

Transmits an absolute sensor value.

```
Header:
  Version/Type: 0x14 (version 1, type 4)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  Seconds since base time
  Length:       2

Payload:
  Bytes 0-1:  Value (2 bytes, signed, big-endian)
```

Value range: -32768 to +32767
Keyframes **SHOULD** be sent:

- As the first data message after registration
- Periodically (recommended: every 10th message)
- When delta would exceed 8-bit range
- After extended silence

4.5 MSG_DATA_DELTA (0x05)

Transmits the difference from the previous value.

```
Header:
  Version/Type: 0x15 (version 1, type 5)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  Seconds since base time
  Length:       1

Payload:
  Byte 0:  Delta (1 byte, signed)
```

Delta range: -128 to +127
The receiver reconstructs the absolute value:

$$\text{current\_value} = \text{previous\_value} + \text{delta} \tag{1}$$

4.6 MSG_HEARTBEAT (0x06)

Indicates device liveness without transmitting data.

```
Header:
  Version/Type: 0x16 (version 1, type 6)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  Seconds since base time
  Length:       0

Payload: None
```

Heartbeats **SHOULD** be sent when:

- Sensor value unchanged beyond threshold
- Maintaining session during idle periods

4.7 MSG_BATCHED_DATA (0x07)

Aggregates multiple data points in a single packet.

```
Header:
  Version/Type: 0x17 (version 1, type 7)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  0x0000 (not used)
  Length:       Variable
```

```
Payload:
  Repeated entries, each containing:
    - Time Offset: 2 bytes
    - Type:        1 byte (0x04 or 0x05)
    - Data:        1-2 bytes (depending on type)
```

Entry size: 4 bytes (delta) or 5 bytes (keyframe)

4.8 MSG_SHUTDOWN (0x0B)

Gracefully terminates a session.

```
Header:
  Version/Type: 0x1B (version 1, type 11)
  Device ID:    Assigned ID
  Sequence:     Current sequence
  Time Offset:  Seconds since base time
  Length:       0

Payload: None
```

5 Connection Lifecycle

5.1 Registration

1. Device sends MSG_STARTUP with MAC address
2. Server checks if MAC is known:
   - New device: Assigns new Device ID
   - Known device: Returns existing ID and last sequence
3. Server sends MSG_STARTUP_ACK
4. Device stores assigned ID for session

5.2 Data Transmission

After registration, devices transmit data following this pattern:

1. Send MSG_TIME_SYNC to establish base time
2. Send MSG_KEYFRAME with initial value
3. For subsequent readings:
   - If $|\text{delta}| \leq 127$: Send MSG_DATA_DELTA
   - If $|\text{delta}| > 127$: Send MSG_KEYFRAME
   - If unchanged: Send MSG_HEARTBEAT (optional)
4. Periodically send MSG_KEYFRAME for recovery

5.3 Session Termination

Normal termination:

1. Device sends MSG_SHUTDOWN
2. Server marks device as inactive
3. Server **MAY** retain device state for reconnection

Timeout termination:

1. Server monitors last-seen timestamps
2. If no message received for timeout period, session is marked inactive
3. Default timeout: 10x average message interval

6 Sequence Number Handling

6.1 Wraparound

Sequence numbers are 16-bit unsigned integers that wrap from 65535 to 0. Implementations **MUST** handle wraparound correctly using modular arithmetic:

$$\text{forward\_step} = (\text{received\_seq} - \text{expected\_seq}) \mod 65536 \tag{2}$$

6.2 Gap Detection

When a gap is detected:

1. Mark missing sequence numbers
2. Accept the new packet
3. If missing packet arrives later, clear from missing set

6.3 Duplicate Detection

A packet is considered duplicate if:

- Sequence number is behind current and not in missing set
- Exact sequence already received (within window)

7 Time Synchronization

7.1 Base Time

Each device maintains a base timestamp established via MSG_TIME_SYNC. All subsequent Time Offset values are relative to this base:

$$\text{actual\_time} = \text{base\_time} + \text{time\_offset} \tag{3}$$

7.2 Re-synchronization

Devices **SHOULD** re-synchronize when:

- Time offset exceeds 60000 (approaching limit)
- Clock drift detected
- After extended disconnection

8 Error Detection

8.1 Packet Classification

Servers classify each received packet as:

- **Normal**: Expected sequence, no issues
- **Duplicate**: Already received
- **Gap**: Sequence jump indicates loss
- **Delayed**: Previously missing, now arrived

8.2 Recovery Strategies

DCT does not mandate retransmission. Recommended strategies:

- Periodic keyframes for delta recovery
- Interpolation for missing values
- Logging gaps for analysis

9 Security Considerations

9.1 Current Limitations

This version of DCT does not include:

- Authentication
- Encryption
- Integrity protection

9.2 Deployment Recommendations

For production use:

- Deploy on trusted networks only
- Use VPN or IPsec for transport security
- Consider DTLS wrapper for untrusted networks
- Implement rate limiting on servers

10 IANA Considerations

This document requests no IANA actions. The default port 5000 is used by convention and is not registered.

A Message Type Quick Reference

Table 3: Message Type Summary

Type	Code	Payload	Direction
STARTUP	0x01	6-7 B	Device → Server
STARTUP_ACK	0x02	2-4 B	Server → Device
TIME_SYNC	0x03	4 B	Device → Server
KEYFRAME	0x04	2 B	Device → Server
DATA_DELTA	0x05	1 B	Device → Server
HEARTBEAT	0x06	0 B	Device → Server
BATCHED_DATA	0x07	Variable	Device → Server
SHUTDOWN	0x0B	0 B	Bidirectional

B Example Message Captures

B.1 STARTUP Message

```
Hex: 11 00 00 00 00 00 00 06 AA BB CC DD EE FF
     |  |     |     |     |  |_________________|
     |  |     |     |     |   MAC Address
     |  |     |     |     Length (6)
     |  |     |     Time Offset (0)
     |  |     Sequence (0)
     |  Device ID (0)
     Version 1, Type 1
```

B.2 KEYFRAME Message

```
Hex: 14 00 05 00 0A 00 3C 02 00 64
     |  |     |     |     |  |___|
     |  |     |     |     |  Value (100)
     |  |     |     |     Length (2)
     |  |     |     Time Offset (60)
     |  |     Sequence (10)
     |  Device ID (5)
     Version 1, Type 4
```

B.3 DATA_DELTA Message

```
Hex: 15 00 05 00 0B 00 3D 01 05
     |  |     |     |     |  |
     |  |     |     |     |  Delta (+5)
     |  |     |     |     Length (1)
     |  |     |     Time Offset (61)
     |  |     Sequence (11)
```

```
    |  Device ID (5)
    Version 1, Type 5
```

References

[1] J. Postel, "User Datagram Protocol," RFC 768, August 1980.

[2] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997.