MQTT Subscribe Control Feature - Analysis & Proposal

Version: 1.1.0 | **Date:** January 2026 | **Status:** Proposal

---

Executive Summary

Fitur **MQTT Subscribe Control** memungkinkan gateway untuk menerima perintah write register dari cloud/IoT platform melalui protokol MQTT. Ini melengkapi kontrol lokal via BLE yang sudah ada (v1.0.8), memberikan kemampuan kontrol remote secara online.

Use Case:

- Kontrol setpoint temperature dari dashboard cloud
- Toggle relay/coil dari aplikasi mobile via internet
- Automated control dari IoT rules engine
- SCADA integration via MQTT broker

Current State Analysis

Infrastructure yang Sudah Ada

Berdasarkan investigasi codebase, **infrastruktur MQTT Subscribe sudah disiapkan** namun **belum diimplementasikan**:

```
// MqttManager.cpp:866-867 - Config sudah di-load tapi TIDAK DIGUNAKAN
defaultTopicSubscribe = defaultMode["topic_subscribe"] | "device/control";
defaultTopicSubscribe.trim();
```

Config Structure (server_config.json):

```
{
  "mqtt_config": {
    "default_mode": {
      "topic_subscribe": "device/control"  // <-- Sudah ada, belum dipakai
    }
  }
}
```

Write Register via BLE (v1.0.8) - Sudah Implemented

```
{
  "op": "write",
  "type": "register",
```

```
  "device_id": "D7A3F2",
  "register_id": "R3C8D1",
  "value": 25.5
}
```

Fitur yang sudah ada:

- Reverse calibration: `raw = (value - offset) / scale`
- Auto function code selection (FC5, FC6, FC16)
- Min/max validation
- Writable flag per register

Industry Best Practices (Research Results)

1. EMQX Neuron (Professional IIoT Gateway)

Write Request Topic: Configurable, default `/neuron/{node}/write/req` **Write Response Topic:** Configurable, default `/neuron/{node}/write/resp`

Single Tag Write:

```
{
  "uuid": "cd32be1b-c8b1-3257-94af-77f847b1ed3e",
  "node": "modbus",
  "group": "grp",
  "tag": "tag0",
  "value": 1234
}
```

Multiple Tags Write:

```
{
  "uuid": "cd32be1b-c8b1-3257-94af-77f847b1ed3e",
  "node": "modbus",
  "group": "grp",
  "tags": [
    {"tag": "tag0", "value": 1234},
    {"tag": "tag1", "value": 5678}
  ]
}
```

Response:

```
{
  "uuid": "cd32be1b-c8b1-3257-94af-77f847b1ed3e",
```

```
  "error": 0
}
```

2. Kepware IoT Gateway (Industrial Standard)

Write Topic: `iotgateway/write` **Format:** JSON with tag path and value

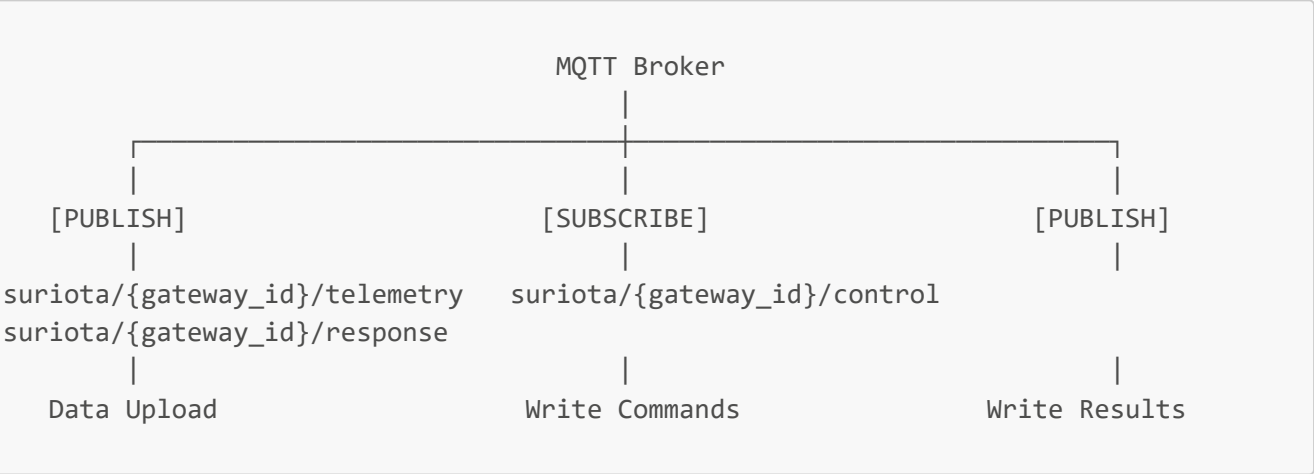
3. Teltonika RUT956 (Router + Modbus Gateway)

Community discussion menunjukkan kebutuhan serupa:

- Subscribe topic untuk trigger Modbus write
- Custom script untuk bridge MQTT → Modbus

Proposed Implementation

Topic Structure



Recommended Topics

Purpose	Topic Pattern	Direction
Data Upload	`suriota/{gateway_id}/telemetry`	Gateway → Broker
Write Command	`suriota/{gateway_id}/control`	Broker → Gateway
Write Response	`suriota/{gateway_id}/response`	Gateway → Broker
Status Report	`suriota/{gateway_id}/status`	Gateway → Broker

Example:

```
suriota/MGate1210_A3B4C5/control    ← Subscribe for commands
suriota/MGate1210_A3B4C5/response   → Publish write results
```

Protocol Specification

Write Single Register

Request (Cloud → Gateway):

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440000",
  "op": "write",
  "device_id": "D7A3F2",
  "register_id": "R3C8D1",
  "value": 25.5
}
```

Response (Gateway → Cloud):

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440000",
  "status": "ok",
  "device_id": "D7A3F2",
  "register_id": "R3C8D1",
  "written_value": 25.5,
  "raw_value": 255,
  "timestamp": 1704067200000
}
```

Write Multiple Registers

Request:

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440001",
  "op": "write_batch",
  "device_id": "D7A3F2",
  "registers": [
    {"register_id": "R3C8D1", "value": 25.5},
    {"register_id": "R4D9E2", "value": 100},
    {"register_id": "R5E0F3", "value": true}
  ]
}
```

Response:

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440001",
  "status": "partial",
```

```
  "device_id": "D7A3F2",
  "results": [
    {"register_id": "R3C8D1", "status": "ok", "written_value": 25.5},
    {"register_id": "R4D9E2", "status": "ok", "written_value": 100},
    {"register_id": "R5E0F3", "status": "error", "error_code": 315, "error":
"Register not writable"}
  ],
  "timestamp": 1704067200000
}
```

Read Register (On-Demand)

Request:

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440002",
  "op": "read",
  "device_id": "D7A3F2",
  "register_id": "R3C8D1"
}
```

Response:

```
{
  "uuid": "550e8400-e29b-41d4-a716-446655440002",
  "status": "ok",
  "device_id": "D7A3F2",
  "register_id": "R3C8D1",
  "value": 25.5,
  "raw_value": 255,
  "timestamp": 1704067200000
}
```

Configuration Schema

Enhanced Server Config

```
{
  "mqtt_config": {
    "enabled": true,
    "broker_address": "broker.hivemq.com",
    "broker_port": 1883,
    "client_id": "MGate1210_A3B4C5",
    "username": "",
    "password": "",
```

```
    "publish_mode": "default",
    "default_mode": {
      "enabled": true,
      "topic_publish": "suriota/MGate1210_A3B4C5/telemetry",
      "interval": 5,
      "interval_unit": "s"
    },

    "subscribe_mode": {
      "enabled": true,
      "topic_control": "suriota/MGate1210_A3B4C5/control",
      "topic_response": "suriota/MGate1210_A3B4C5/response",
      "qos": 1
    }
  }
}
```

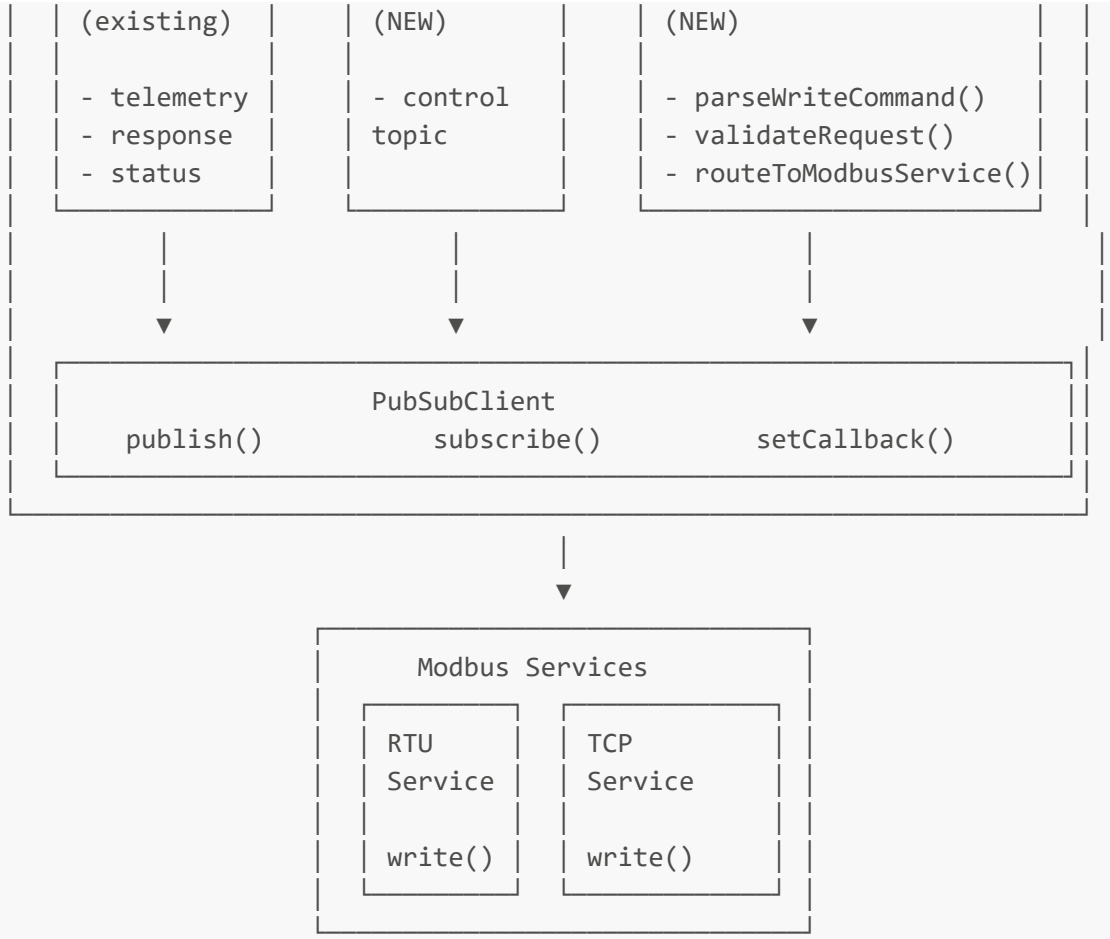
Per-Register Subscribe Config (Customize Mode)

Untuk kontrol granular, user bisa menentukan register mana yang bisa di-write via MQTT:

```
{
  "mqtt_config": {
    "subscribe_mode": {
      "enabled": true,
      "allowed_registers": [
        {
          "device_id": "D7A3F2",
          "registers": ["R3C8D1", "R4D9E2", "R5E0F3"]
        },
        {
          "device_id": "A1B2C3",
          "registers": ["*"]  // Allow all writable registers
        }
      ]
    }
  }
}
```

Implementation Architecture

Component Diagram



Data Flow

```
Cloud Dashboard
      │
      │ Publish to: suriota/MGate1210_A3B4C5/control
      ▼
┌──────────────────┐
│   MQTT Broker    │
└──────────────────┘
      │
      │ Subscribe callback triggered
      ▼
┌──────────────────────────────────────────────────────────────┐
│ MqttManager::onMqttMessage(topic, payload, length)           │
│   │                                                          │
│   ├─► Parse JSON payload                                     │
│   ├─► Validate request (uuid, op, device_id, etc.)           │
│   ├─► Check if register is allowed for MQTT write            │
│   ├─► Route to appropriate handler:                          │
│   │     - "write" → handleWriteCommand()                     │
│   │     - "write_batch" → handleBatchWrite()                 │
│   │     - "read" → handleReadCommand()                       │
│   │                                                          │
│   └─► Publish response to response topic                     │
└──────────────────────────────────────────────────────────────┘
```

```
        │
        │ Call existing write infrastructure
        ▼
┌──────────────────────────────────────────────────────────────┐
│ ModbusRtuService::writeRegisterValue() (existing v1.0.8)     │
│ ModbusTcpService::writeRegisterValue() (existing v1.0.8)     │
└──────────────────────────────────────────────────────────────┘
        │
        │ Modbus FC5/FC6/FC16
        ▼
┌──────────────────┐
│  Modbus Device   │
└──────────────────┘
```

Security Considerations

1. Authentication

Broker-Level:

- Use MQTT username/password authentication
- Enable TLS/SSL for encrypted communication
- Use ACL (Access Control List) on broker

Message-Level:

- Optional: Add `auth_token` field in payload
- Optional: Sign messages with shared secret

2. Authorization

Register-Level Control:

```
{
  "subscribe_mode": {
    "allowed_registers": [...]  // Whitelist approach
  }
}
```

Device-Level Control:

- Only allow writes to explicitly enabled devices
- Respect `writable` flag per register

3. Rate Limiting

```
// Prevent command flooding
struct WriteRateLimiter {
```

```
    uint32_t maxWritesPerMinute = 60;
    uint32_t currentCount = 0;
    unsigned long windowStart = 0;
};
```

4. Validation

- UUID required for request tracking
- Device ID validation
- Register ID validation
- Value range check (min_value, max_value)
- Data type validation

Error Codes

Code	Domain	Description
400	MQTT	Invalid JSON payload
401	MQTT	Missing required field (uuid, op, device_id)
402	MQTT	Unknown operation
403	MQTT	Device not found
404	MQTT	Register not found
405	MQTT	Register not allowed for MQTT write
315	Modbus	Register not writable
316	Modbus	Value out of range
318	Modbus	Write operation failed

Implementation Phases

Phase 1: Core Subscribe (v1.1.0)

Scope:

- ☐ Add MQTT subscribe callback in MqttManager
- ☐ Implement `handleWriteCommand()` - single register
- ☐ Add response publishing
- ☐ Basic config for control/response topics

Files to Modify:

- `MqttManager.h` - Add callback, new methods
- `MqttManager.cpp` - Implement subscribe logic
- `ServerConfig.h/cpp` - New subscribe_mode config

Phase 2: Batch & Read (v1.1.1)

Scope:

- [ ] Implement `handleBatchWrite()` - multiple registers
- [ ] Implement `handleReadCommand()` - on-demand read
- [ ] Add QoS configuration
- [ ] Enhanced error responses

Phase 3: Security & Advanced (v1.1.2)

Scope:

- [ ] Per-register allow list
- [ ] Rate limiting
- [ ] Optional message signing
- [ ] TLS support

Testing Strategy

Unit Tests

- JSON parsing validation
- Request validation
- Error code generation

Integration Tests

```
# test_mqtt_subscribe.py
import paho.mqtt.client as mqtt
import json
import uuid

def test_write_single_register():
    client = mqtt.Client()
    client.connect("broker.hivemq.com", 1883)

    # Subscribe to response topic
    client.subscribe("suriota/MGate1210_A3B4C5/response")

    # Send write command
    cmd = {
        "uuid": str(uuid.uuid4()),
        "op": "write",
        "device_id": "D7A3F2",
        "register_id": "R3C8D1",
        "value": 25.5
    }
    client.publish("suriota/MGate1210_A3B4C5/control", json.dumps(cmd))
```

```
    # Wait for response...
```

Manual Tests

- [ ] Write single register via MQTT
- [ ] Write batch registers via MQTT
- [ ] Write with invalid device ID
- [ ] Write to non-writable register
- [ ] Write value out of range
- [ ] Concurrent BLE + MQTT writes

Comparison: BLE vs MQTT Control

Aspect	BLE Control (v1.0.8)	MQTT Control (v1.1.0)
Range	Local (~10m)	Global (Internet)
Latency	Low (~50ms)	Medium (~200-500ms)
Security	BLE pairing	Broker auth + TLS
Reliability	Direct connection	Broker dependent
Use Case	Mobile app config	Cloud/SCADA control
Concurrent	1 client	Multiple clients

Mobile App Integration

Untuk mobile app yang ingin menggunakan MQTT control (bukan BLE):

```
class MqttControlService {
  final String brokerHost;
  final String gatewayId;

  String get controlTopic => 'suriota/$gatewayId/control';
  String get responseTopic => 'suriota/$gatewayId/response';

  Future<WriteResult> writeRegister({
    required String deviceId,
    required String registerId,
    required dynamic value,
  }) async {
    final uuid = Uuid().v4();

    // Publish command
    await mqttClient.publish(
      controlTopic,
      jsonEncode({
```

```
          'uuid': uuid,
          'op': 'write',
          'device_id': deviceId,
          'register_id': registerId,
          'value': value,
        }),
      );

      // Wait for response with matching UUID
      return await waitForResponse(uuid, timeout: Duration(seconds: 10));
    }
  }
```

Summary

Kesimpulan Analisis:

1. **Infrastructure Ready** - Config `topic_subscribe` sudah ada, tinggal implementasi
2. **Reuse v1.0.8** - Write logic sudah ada, cukup route dari MQTT callback
3. **Industry Standard** - Format JSON seperti EMQX Neuron sudah proven
4. **Security Important** - Perlu whitelist register yang boleh di-write via MQTT

Recommended Next Steps:

1. Approve proposal ini
2. Implementasi Phase 1 (Core Subscribe) di v1.1.0
3. Testing dengan MQTT broker publik
4. Deploy ke production dengan broker private + TLS

Document Version: 1.0 | **Last Updated:** January 2026

SURIOTA R&D Team | support@suriota.com