Advanced Room Server Topics

Running Multiple Rooms

A single room server can host multiple rooms (channels), each with independent access control and settings. This is common for community networks that run both a public room and a private emergency operations room.

Room architecture

In MeshCore, a "room" is a message space with:

- A name (visible to clients connecting to the server)
- An optional PSK (empty = public room; set = encrypted private room)
- Its own message history stored separately
- Its own participant list

Clients choose which rooms to join when they connect to a room server. A client can be in multiple rooms simultaneously.

Example multi-room configuration

```
rooms:
 # Public room - anyone can join, no encryption - name: "RegionMesh-Public"
 psk: ""
 description: "Public community channel for the region"
 max_history: 1000 # Messages to keep in memory

 # Emergency net - shared PSK with authorized operators - name: "EmergencyNet"
 psk: "secretkeyhere"
 description: "For emergency coordinators only"
 max_history: 500

 # Local club - private to members - name: "ClubNet"
 psk: "clubsecretkey"
 description: "For club members"
 max_history: 200
```

Distributing room PSKs

PSK distribution should happen through a separate secure channel (encrypted email, in-person, Signal, etc.), not over the mesh network itself. Anyone with the PSK can read all messages in that room - treat it like a shared password and change it if an unauthorized person may have obtained it.

For public emergency preparedness networks, the EmergencyNet PSK is typically shared with vetted local emergency management personnel, ARES/RACES members, and CERT team leads - not published publicly.

Monitoring room activity

The room server exposes a simple status endpoint that can be queried for operational monitoring:

```
# Number of connected clients per room
curl http://localhost:7070/status

# Or via the room server CLI (if available in your version)
python3 manage.py status
```

Internet Bridging and MQTT

Room servers with internet connectivity can bridge LoRa mesh traffic to internet-connected clients, enabling phone users without LoRa hardware to participate in the mesh network. MQTT integration allows mesh traffic to be monitored and analyzed with standard tools.

Internet bridge architecture

When internet bridging is enabled, the room server acts as a relay between:

- Local LoRa radio nodes (connected via the room server's radio port or via radio gateways)
- Internet-connected MeshCore clients (phones, computers using TCP)

This allows a person in another city to send and receive messages with local mesh participants, as long as both have a path to the room server - one via radio, one via internet.

Security considerations for internet bridging

An internet-exposed room server requires proper security:

- **TLS/SSL:** Enable HTTPS/TLS for all internet connections. Without it, messages transit in plaintext to the server (even if end-to-end encrypted between clients).
- **Authentication:** Configure the server to require client authentication. Open rooms accessible from the internet can attract abuse.
- **Firewall rules:** Restrict access to the room server port. If only local clients need to connect directly, block external access to the TCP port and use a reverse proxy (nginx/caddy) for TLS termination.
- **Rate limiting:** Apply per-client message rate limits to prevent a single client from flooding the network.

Exposing a room server to the internet

Using a reverse proxy with TLS (recommended):

```
# Example nginx configuration
server {
```

```
 listen 443 ssl;
 server_name mesh.yournetwork.com;

 ssl_certificate /etc/letsencrypt/live/mesh.yournetwork.com/fullchain.pem;
 ssl_certificate_key /etc/letsencrypt/live/mesh.yournetwork.com/privkey.pem;

 location / {
 proxy_pass http://127.0.0.1:7070;
 proxy_http_version 1.1;
 proxy_set_header Upgrade $http_upgrade;
 proxy_set_header Connection "upgrade";
 }
}
```

Get free TLS certificates from Let's Encrypt using certbot.

MQTT integration

MQTT bridging forwards all room messages to an MQTT broker for monitoring, logging, and integration with other systems:

```
mqtt:
 enabled: true
 broker: "mqtt://localhost:1883" # Or your MQTT broker address
 topic_prefix: "meshcore" # Base topic for all messages

 # Topics published:
 # meshcore/messages/{room} - all messages in a room
 # meshcore/nodes/{node_id} - node status and position updates
 # meshcore/status - server health metrics
```

Setting up an MQTT broker

```
# Install Mosquitto MQTT broker
sudo apt install -y mosquitto mosquitto-clients

# Start and enable
sudo systemctl enable mosquitto
```

```
sudo systemctl start mosquitto
```

Visualizing with Grafana + InfluxDB

A common monitoring stack for mesh networks:

1. Install InfluxDB (time-series database)
2. Install Telegraf with MQTT input plugin to consume mesh MQTT topics and write to InfluxDB
3. Install Grafana and create dashboards showing: message rate per room, active nodes over time, node battery levels, coverage heatmaps from GPS data

This stack can run on the same MeshCore Room Server (running on dedicated nRF52840 or ESP32 hardware)

Alerting on node failure

Use MQTT + Node-RED or a simple Python script to alert when a node stops checking in:

```
import paho.mqtt.client as mqtt
import time

nodes = {}
ALERT_TIMEOUT_SECONDS = 3600 # Alert if not heard in 1 hour

def on_message(client, userdata, msg):
 node_id = msg.topic.split('/')[-1]
 nodes[node_id] = time.time()

def check_timeouts():
 now = time.time()
 for node_id, last_seen in nodes.items():
 if now - last_seen > ALERT_TIMEOUT_SECONDS:
 print(f"ALERT: {node_id} has not been heard in over 1 hour!")

# Subscribe to node status topics
client = mqtt.Client()
client.on_message = on_message
client.connect("localhost", 1883)
client.subscribe("meshcore/nodes/+")
client.loop_start()
```

```
while True:
 check_timeouts()
 time.sleep(300)
```