STRIDEtastic(1): Meshtastic and Security

Francisco Wulf
Pontificia Universidad Católica de Chile
francisco.wulf@uc.cl

***Abstract*—This document presents the first research increment after the initial introduction (*Increment 0*) in the investigation *STRIDEtastic: Threat Modeling and Security Evaluation of Meshtastic Networks through Practical and Theoretical Attacks*. This increment focuses on establishing the theoretical framework for the study, defining Meshtastic, describing it's user and security features, and attack surfaces.**

I. Introduction

Increment 1 establishes the theoretical and technical foundation for the research. It begins by defining and describing the Meshtastic system, including both architectural and operational aspects.

The section covers the LoRa communication protocol, routing mechanisms, supported interfaces, and the overall attack surface. It also reviews the security mechanisms implemented in Meshtastic, highlighting strengths and weaknesses across different communication layers and interfaces.

II. What is Meshtastic?

Meshtastic is an open-source project that enables long-range, off-grid, encrypted communications by leveraging inexpensive LoRa (Long Range) radio modules [1] . It allows users to form decentralized ad-hoc mesh networks without relying on traditional telecommunications infrastructure, making it especially suitable for environments where conventional networks are unavailable or unreliable. LoRa modules used by Meshtastic operate in unlicensed ISM bands, meaning that, in most regions, they can be used without requiring an amateur radio license. Figure 1 shows an overview on how Meshtastic communication happens. Nodes communicate with each other via the LoRa interface. Users can interact with the mesh by using standalone devices or by pairing their devices with an user interface like a mobile app (via BLE) or the Web Client (via TCP/HTTP).

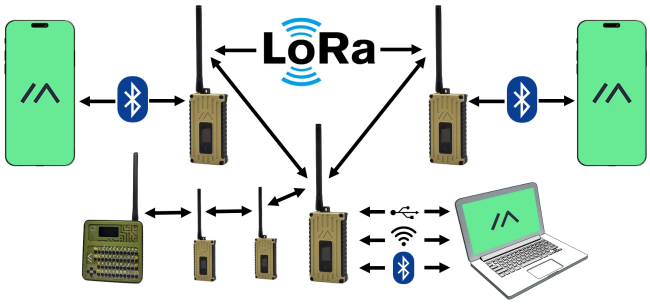

Fig. 1. Meshtsatic network communication overview.

A. Key Features

The main features [2] of Meshtastic include:

- **Long-range communication:** Depending on terrain and antenna setup, LoRa can achieve distances of several kilometers between nodes. Mostly sensitive to Line of Sight (LoS).
- **Decentralized mesh networking:** No central coordinator/authority is required; (almost) all nodes act as relays within the mesh.
- **Encrypted communication:** Payloads are encrypted to protect message confidentiality over the air.
- **Low-power operation:** Designed for battery-powered devices with efficient power management, i.e. embedded systems.
- **Multi-platform support:** Communication with Android, iOS, Python API, and web interfaces.
- **Off-grid operation:** Nodes can communicate without any dependency on cellular networks or the Internet.

B. General Operation

At the radio layer, a Meshtastic mesh is a set of nodes configured with the same spreading factor, center frequency, and bandwidth [3]. A node will only see and process messages from other nodes using the same values for these parameters. This configuration defines a *radio mesh*

Nodes communicate by broadcasting messages over LoRa. When a node receives a message, it rebroadcasts it to extend its reach through the network. This broadcast process is constrained by a *hop limit* to prevent infinite retransmissions; a typical hop limit setting is 3 or 4. The underlying *flooding* approach ensures that messages are propagated across the mesh within the hop limit.

Meshtastic supports up to 8 logical channels per node. All radios within a mesh retransmit messages from **all** channels in the mesh, but **channel configuration** controls which messages a node can encrypt/decrypt and display to the user. Channels can be public (default settings) or private (with encryption keys configured).

C. User Operation

From the user's perspective, Meshtastic provides a straightforward interface for communicating over the mesh through companion applications and client tools. A typical usage flow involves pairing a smartphone (Android or iOS) with a Meshtastic radio via Bluetooth or connecting over TCP/HTTP through a web client, enabling the user to send and receive messages through the mesh.

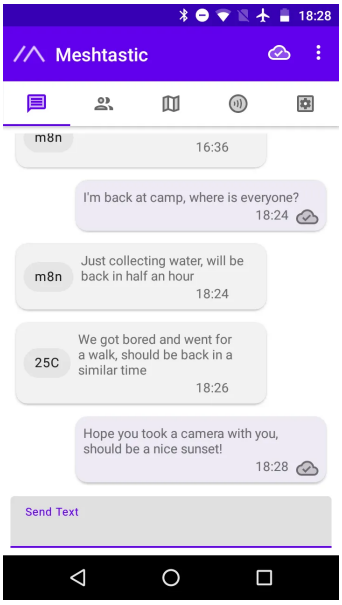

Fig. 2. Meshtastic Android UI channel conversation.

The core user-facing functionalities include:

- **Channel-based messaging:** Users can join one or more channels (up to eight per device) to exchange messages with other participants in the mesh. Channels can be public or secured with encryption keys.
- **Direct messages (DMs):** Node-to-node private communication.
- **User and device information sharing:** Nodes can broadcast metadata such as long name, hardware model and public key.
- **Geolocation sharing:** Devices equipped with GPS can periodically share coordinates with specified precision, allowing for location-aware applications like GPS trackers.
- **Sensor data transmission:** Environmental and other sensor readings connected to the node can be shared over the mesh.
- **Local and Remote Node Configuration:** Configure a node directly via a HTTP/TCP/BLE/Serial interface, or configure remote nodes via the LoRa radio.
- **Other applications:** Traceroute, Range Test and ATAK (Android Team Awerness Kit) plugin.

These interactions are typically performed through the Meshtastic mobile app, which provides a chat-style interface for messaging, configuration menus for channel and device settings, and visualizations for geolocation data. The same functionalities are also accessible through desktop and web clients, as well as via command-line tools using the node's serial or TCP/HTTP interfaces. Figure 2 shows a screenshot of the Meshtastic Android UI for interacting with a node via BLE interface.

D. Interfaces and Attack Surface

Meshtastic nodes can interface with other devices and networks through multiple communication paths:

- **LoRa RF link:** Primary interface for mesh communication between nodes.
- **Bluetooth Low Energy (BLE):** Allows pairing with smartphones and tablets running the Meshtastic app.
- **USB Serial:** Provides a direct wired connection for configuration and message exchange via the Python API and Meshtastic CLI.
- **TCP/HTTP API:** Enables interaction with the node over a network connection, including a local web interface.
- **MQTT modules:** Integrates the mesh with IP-based networks by publishing/receiving messages via an MQTT broker.

Each of these interfaces constitutes part of the system's attack surface. While LoRa communications are encrypted at the payload level, packet headers travel in plaintext. Other interfaces, such as the TCP API, may not enforce authentication by default, making interface exposure a significant security consideration. The BLE and serial interfaces, while requiring proximity or physical access, also provide configuration-level control and must be securely configured.

E. LoRa Communication Stack and LoRa Meshtastic Routing Algorithm

a) Use of Protocol Buffers (Protobufs): Meshtastic encodes its messages using Google Protocol Buffers (protobufs), a compact and efficient serialization format designed for transmitting structured data. Protobufs allow the definition of message schemas that can be versioned and extended while maintaining backwards compatibility. In Meshtastic, all payloads sent over the LoRa link are protobuf-encoded, ensuring minimal overhead and consistent parsing across heterogeneous devices and clients. The protobuf messages encapsulate user data (text, telemetry, control messages) as well as internal routing and administrative commands.

b) Layer 0 – LoRa Radio Link: Layer 0 corresponds to the physical layer over LoRa. Each transmission consists of a 16-byte preamble, a 2-byte Sync Word (0x2B for Meshtastic, acting as a magic number to distinguish networks), approximately 2 bytes of downchirp, and the transmitted frame. This signal depends on the selected physical channel (frequency slot), spreading factor, and coding rate.

c) Layer 1 – Unreliable 0-Hop Messaging: Messages are transmitted using Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA): before sending, the node senses the channel to check if it is free; if busy, it waits for a random backoff before retrying. This reduces packet collisions in dense networks [3]. At this layer, the packet structure [3] includes:

- **Packet Header (unencrypted):**
  - *Destination*: Address/identifier of the target node or broadcast.
  - *Sender*: Address/identifier of the transmitting node.

OSI packet	OSI layer	Meshtastic layer	Protobuf serialization
Payload	Application	Layer4	App-specific protobuf e.g. Telemetry protobuf
Segment	Transport	Layer2+Layer3	Data protobuf
Packet	Network	Layer2+Layer3	MeshPacket protobuf
Frame	Data Link	Layer1	MeshPacket protobuf
Signal	Physical	Layer0	LoRa signal

Fig. 3. Mapping OSI model to Meshtastic layers.

  - *Packet ID*: Unique identifier for deduplication and tracking.
  - *Packet Flags*: Control bits indicating type of message, reliability, etc.
  - *Channel Hash*: Short hash identifying the logical channel.
  - *Next Hop*: ID of the node to forward to in direct routing.
  - *Relay Node*: ID of the node that relayed the packet last.
- **Packet Data (encrypted):** Contains the protobuf-encoded payload (user messages, telemetry, commands). Encryption ensures confidentiality, while the unencrypted header enables intermediate nodes to perform routing without access to payload contents.

d) Layer 2 – Reliable 0-Hop Messaging: For reliable delivery to a directly reachable node, the protocol uses acknowledgment messages. The *WantAck* is set to request an ACK from the receiving node. If no ACK is received within a timeout, the packet may be retransmitted according to retry rules.

e) Layer 3 – Multi-Hop Messaging: This layer handles propagation across multiple nodes in the mesh.

Broadcasting using Managed Flooding: Messages intended for the entire mesh are broadcast with a hop limit. Each node receiving the packet decrements the hop count and rebroadcasts it unless the hop limit reaches zero or the packet has already been seen.

The ”managed” aspect involves delaying rebroadcasts slightly and checking if neighbors have already transmitted the packet to reduce redundant transmissions [3]. It works by giving priority to ”further away” nodes, although distance is estimated via signal strength and signal-to-noise ratio. Figure 4 shows an example setup with 4 nodes forwarding `MeshPackets` under managed flooding.

Every node has a role in the mesh and the role is configurable. These roles determine the behavior of the device in the routing algorithm: CLIENT nodes rebroadcast messages, CLIENT_MUTE do not, HIDDEN does not broadcast device information packets, ROUTER and REPEATER have priority in the managed flooding algorithm, etc.

Direct Messages using Next Hop Routing: For unicast messages, the *Next Hop* field in the header specifies the immediate neighbor on the path toward the destination. Routing tables are maintained locally, storing the latest node that relayed a packet to the destination and succeeded (got an ACK). Figure 5 shows an example on how three nodes exchange information during a Next Hop Routing scheme.

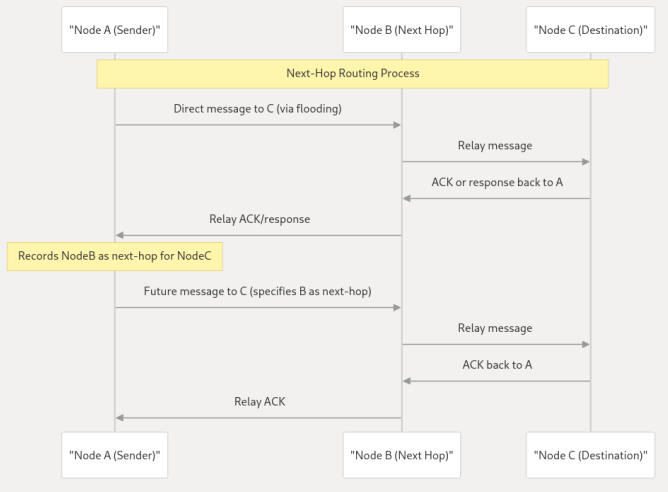

Fig. 5. Three Nodes Communication under Next Hop Routing scheme.

f) Layer 4 - Meshtastic Applications: The Meshtastic firmware includes a variety of applications that allow different behavior, such as sending text messages, sharing device infor-

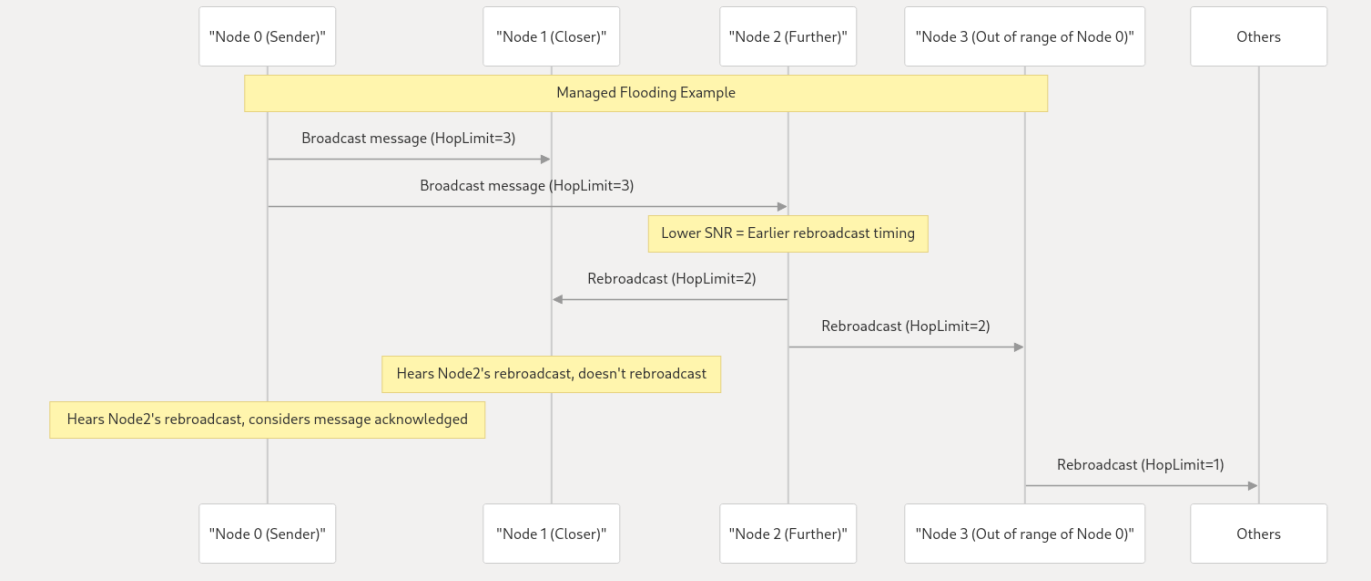

Fig. 4. Example on how Managed Flooding forwarding works.

mation, transmitting telemetry, participating in a traceroute, and more. All of this is done through mesh applications. Each application is associated with a portnum, which ranges from 0 to 511 [4].

g) Mapping to the OSI Model: Although the Meshtastic documentation organizes the protocol in four layers (Layer 0 to Layer 3), its functionality can also be interpreted with a simplified OSI model with five layers: *physical, data link, network, transport*, and *application*. This mapping is not strictly one-to-one, since some Meshtastic layers encapsulate features spanning more than one OSI layer. Nonetheless, it provides a useful framework for analysis. Figure 3 shows a general idea of how the Meshtastic layers can be mapped to the OSI model layers.

Physical layer (L1): Corresponds to Meshtastic Layer 0, which implements the LoRa Chirp Spread Spectrum physical signal. As Figure 6 shows, the signal contains LoRa preamble, sync word, downchirp and a datalink-layer frame.

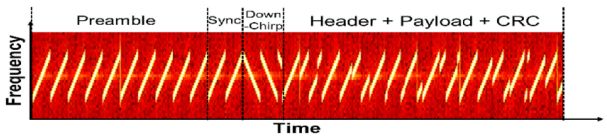

Fig. 6. LoRa Chirp Spread Spectrum signal.

Data link layer (L2): Maps to Meshtastic Layer 1. At this level, a frame encapsulates the *Destination ID* (4B), *Source ID* (4B), and a network-layer packet. Medium access is handled by CSMA/CA, and the node's MAC address corresponds to its Node ID, derived from the lower 4 bytes of its Bluetooth interface MAC address. In Meshtastic, the frame corresponds to a `MeshPacket` protobuf. Figure 7 shows how a datalink-layer frame is structured.

Message serialization in Meshtastic appears to combine the data link and network layers into a single layer, as the 4th column in Figure 3 shows.

Destination ID (4B)	Source ID (4B)	Network-layer Packet

Fig. 7. Meshtastic's datalink-layer frame.

Network layer (L3): Corresponds to features of Meshtastic Layer 2 and Layer 3. As Figure 8 shows, a network-layer packet includes *Packet ID* (4B), *Flags* (1B), *Channel Hash* (1B, identifying the logical channel), *Next Hop Hash* (1B), *Relay Node Hash* (1B), and a transport-layer segment. Flags encode hop control fields such as `hop_start`, `hop_limit`, and `want_ack`, while `next_hop` and `relay` fields support multi-hop forwarding and routing. Importantly, this highlights the distinction between the *physical channel* (frequency slot) and the *logical channel* (similar to a group chat).

In Figure 9 the structure of a MeshPacket protobuf is detailed. This is the actual serialization Meshtastic uses. It's easy to see how this protobuf corresponds to the frame plus the packet in the OSI model.

Packet ID (4B)	Flags (1B)	Channel Hash (1B)	Next Hop Hash (1B)	Relay Node Hash (1B)	Transport Segment (Data protobuf)

Fig. 8. Meshtastic's network-layer packet

Transport layer (L4): The segment transported in the packet corresponds to a `Data` protobuf, functioning similarly to a datagram. Figure 10 shows the structure of the segment/Data protobuf. It carries the *portnum* (1B) identifying the upper-layer application, the *payload*, *want_response* (boolean), *destination* and *source* (4B each), *request_id* and *reply_id* (4B each), and additional control bits such as an *emoji* flag and a *bitfield*. This layer borrows functionality both from Layer 2 (reliable IDs and responses) and Layer 3 (addressing for reliable multi-hop delivery).

Application layer (L5): At the top, the payload corresponds to application-specific protobufs. These represent the different applications embedded in the Meshtastic firmware, analogous to processes in an operating system. Examples of Meshtastic applications include the `TEXT_MESSAGE_APP`, `NODEINFO_APP`, `POSITION_APP`, `TELEMETRY_APP`, and the `ROUTING_APP`. The latter is conceptually similar to `iproute2` in Linux, as it provides routing functionality as an application rather than a network primitive.

III. Security in Meshtastic

A. Meshtastic Encryption

a) Overview: Meshtastic encrypts the *payloads* of packets sent over LoRa using AES256-CTR, with a distinct key for each logical channel. The **packet header is always sent unencrypted** (raw bytes) allow intermediate nodes to retransmit packets even if they do not possess the channel key [5]. By default, there is a primary channel with a known key ('AQ==‘), meaning that the network remains essentially public until the user changes the PSK.

b) PSK, key sizes, and supported modes: Channel *pre-shared keys* (PSKs) can be 0 bytes (no encryption), 16 bytes (AES-128), or 32 bytes (AES-256). While the documentation recommends AES256-CTR for hardware support and performance, the implementation supports either 128- or 256-bit keys depending on channel configuration.

c) Direct Messages (DMs): Since version 2.5.0 (and only when a key exchange has occurred), DMs are protected using public key cryptography (PKC): each node has a Curve25519 public/private key pair, the sender encrypts using the recipient's **public key** and signs the message with their **private key**. This ensures that only the recipient can decrypt the content and that the recipient can verify the authenticity and integrity of the message (signature). Messages between older firmware versions (pre-2.5.0) do not have this protection; they use the AES256-CTR symmetric key encryption algorithm. A known vulnerability [6] affects the Android application, where DMs sent with this legacy encryption are displayed without the stronger PKC protections. This behavior is still present in Meshtastic Web Client as for version 2.6.5. A report has been issued (by me) for this vulnerability. This vulnerability arises because of an insecure way of handling the DMs at a firmware level: if the 'pki_encrypted‘ field of the protobuf packet is not populated, then the device fallbacks to legacy encryption, namely, AES256-CTR symmetric encryption. This allows for a downgrade attack, which if exploited, can be leveraged into DM spoofing. Another security report was created for this. Both security reports are in review stage (triage). If they get released to the public, then they are getting included here.

d) Admin Messages: As of version 2.5.0, *admin messages* includes enhanced protections: stronger encryption with session identifiers (Session IDs). Before 2.5, administrative functions had fewer protections and were more vulnerable to spoofing or remote manipulation. Nodes with admin access are set in configuration via the serial interface.

e) Security properties: confidentiality, integrity, authentication, and PFS [7]:

- **Confidentiality:** Channel payloads encrypted with a PSK (AES-128/256) provide confidentiality against passive observers without the key; DMs use PKC for recipient-specific confidentiality.
- **Integrity:** Meshtastic, in channel mode, **does not by default provide integrity verification (MAC/AEAD) for channel messages**. This means that an adversary with the PSK (or with known plaintext) could manipulate messages. For DMs and Admin (v2.5+), integrity/signature checks are present.
- **Authentication:** Channels do not authenticate the sender's identity: the `Sender` field is derived from parts of the MAC and is trivially spoofable by anyone with the PSK. DMs introduce authentication since messages are signed with the sender's private key.
- **Perfect Forward Secrecy (PFS):** Meshtastic does not offer PFS; if a channel PSK is leaked, all previously captured traffic in that channel can be decrypted ("harvest now, decrypt later"). The PKC used for DMs is also not currently resistant to quantum attacks; Meshtastic acknowledges these limitations as a known risk.

f) Device behavior and APIs (BLE/Serial/Wi-Fi/MQTT): The device decrypts the payload before sending it to a connected client (BLE, serial, Wi-Fi/Ethernet). For MQTT, there is an option to publish the payload encrypted or decrypted depending on the MQTT module's configuration.

B. Security of Non-LoRa Interfaces

While the LoRa RF link is the primary communication channel in Meshtastic, nodes expose several additional interfaces that can significantly increase the attack surface if not properly secured. These interfaces provide configuration, data exchange, and integration capabilities, but vary widely in their default security posture.

a) Serial Interface (USB/UART): The serial port offers the highest level of access to a node, allowing full configuration of almost every parameter and control over device behavior. This level of privilege is considered acceptable under the assumption that physical access to the node is not possible (it is advised not to trust unmonitored nodes). In practice, once a serial session is established, a user can:

- Change channel PSKs and network parameters.
- Enable or disable modules (MQTT, Wi-Fi, Bluetooth, etc.).
- Update firmware or reset the device.

Fig. 9. MeshPacket protobuf structure.

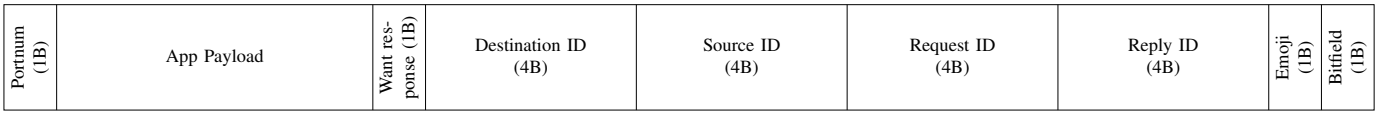

Fig. 10. Data protobuf (transport-layer segment) structure

b) Bluetooth Low Energy (BLE): BLE enables wireless pairing between a node and mobile/desktop clients. Security depends heavily on the pairing configuration:

- **Random PIN** (recommended): A new PIN is generated on connection attempt and displayed on the node's screen (if present), requiring physical proximity and visual access to complete pairing.
- **Fixed PIN** (insecure): A static PIN stored in the configuration; vulnerable to brute-force attacks and device spoofing if BLE is left enabled in untrusted environments.
- **No PIN** (most insecure): This one is not advised.

Potential threats include unauthorized configuration changes, denial-of-service via malformed BLE packets, and brute-force pairing attempts if a fixed PIN is used. Disabling BLE when not in use or enforcing random PIN pairing mitigates these risks.

c) TCP Interface: Meshtastic firmware exposes a TCP server for client connections. This interface has no built-in authentication: **any host with IP connectivity to the node can connect and issue commands**. For this reason, the TCP port should only be bound to private, trusted networks. If an attacker gains access to the same IP network (e.g., by compromising the Wi-Fi AP), the TCP interface becomes a direct remote-control vector.

d) HTTP Interface (Web UI / API): The built-in HTTP server provides a browser-based UI and REST-like API for device interaction. Similar to the TCP server, by default it does not enforce authentication, meaning that anyone with network reachability can issue configuration or messaging commands. While HTTPS can be enabled, in practice many deployments operate over plaintext HTTP, exposing traffic to interception and manipulation.

e) MQTT Module Interface: The MQTT module integrates the mesh with MQTT brokers, allowing nodes to publish and subscribe to topics for message exchange over the Internet. Security considerations include:

- MQTT module can be configured to transmit payloads encrypted or decrypted. Sending decrypted payloads exposes message content to the broker and any subscribers.
- Broker access control (username/password, TLS) should be enabled to prevent unauthorized publishing or subscription.
- While MQTT provides powerful integration capabilities, it extends the attack surface beyond the local network, often into cloud infrastructure.

MQTT broker boilerplate tools exist [8] for building Meshtastic broker moderation rules like rate-limiting, zero-hopping enforcement, fail2ban-like moderation, IP banning, etc. Use of these tools may be leveraged into other attacks (e.g. Denial of Service), as non-malicious nodes can get banned if they gateway malicious packets, sent to the node via its LoRa interface.

IV. Next Steps

In the next increments, we are presenting the STRIDE threat model for Cyber Physical Systems. Then, all other related work is being included (threat modeling, security evaluation, more Meshtastic CVEs and security reports, previous security assessments of Meshtastic, similar studies to similar IoT mesh protocols).

References

[1] Meshtastic, "Introduction," https://meshtastic.org/docs/introduction/, 2025, accessed: 2025-08-12.
[2] ——, "Overview," https://meshtastic.org/docs/overview/, 2025, accessed: 2025-08-12.
[3] ——, "Mesh algorithm overview," https://meshtastic.org/docs/overview/mesh-algo/, 2025, accessed: 2025-08-12.
[4] ——.
[5] ——, "Encryption overview," https://meshtastic.org/docs/overview/encryption/, 2025, accessed: 2025-08-12.
[6] MITRE, "Cve-2025-52883," https://cve.mitre.org/cgi-bin/cvename.cgi?name=2025-52883, 2025, accessed: 2025-08-12.
[7] Meshtastic, "Encryption comments," https://meshtastic.org/docs/about/overview/encryption/comments/, 2025, accessed: 2025-08-12.
[8] ——, "Mqtt integration," https://github.com/meshtastic/mqtt, 2025, accessed: 2025-08-12.