PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERÍA
Dirección de Investigación e Innovación
Programa IPre de Investigación en Pregrado

STRIDEtastic: Framework de Observabilidad para Redes Meshtastic

Francisco Wulf Tondreau[a], Prof. Miguel Gutiérrez Gaitán[b]

[a] *Departamento de Ciencias de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile. 6to año, francisco.wulf@uc.cl*

[b] *Departamento de Ingeniería Eléctrica, Escuela de Ingeniería, Pontificia Universidad Católica de Chile. Profesor, miguel.gutierrez@uc.cl*

Resumen

Las redes Meshtastic, basadas en microcontroladores con radios LoRa y topología en malla, han ganado popularidad por su bajo costo, facilidad de despliegue y aplicaciones que van desde montañismo hasta entornos industriales y militares. Sin embargo, estas redes carecen de mecanismos centralizados de monitoreo y observabilidad, lo que dificulta la evaluación de su desempeño y la detección de amenazas de seguridad.

Este trabajo propone STRIDEtastic, un framework de observabilidad extensible que centraliza la captura, análisis y visualización de redes Meshtastic. La investigación se basó en cuatro etapas: revisión técnica del stack de Meshtastic, modelado de amenazas mediante STRIDE, desarrollo de la plataforma y validación mediante medición de latencia en la red comunitaria chilena (mqtt.meshchile.cl). STRIDEtastic integra captura multi-interfaz, decodificación y descifrado de paquetes, publicación controlada de mensajes, y visualización avanzada de topología, telemetría y métricas históricas mediante dashboards interactivos.

Se evaluó la latencia de dos nodos bajo condiciones controladas, observándose que, a diferencia del nodo conectado vía MQTT, el nodo con interfaz LoRa presentó picos de retraso de hasta diez veces durante ráfagas de mensajes, evidenciando congestión en los enlaces LoRa durante este tipo de eventos. STRIDEtastic permite detectar, disectar y analizar estos eventos en tiempo real, centralizando información dispersa y facilitando el monitoreo activo de la red.

El framework aporta una solución integral para la observabilidad de redes Meshtastic, habilitando tanto a usuarios como a administradores a supervisar el estado de la red, identificar anomalías y extender las herramientas de visualización según sus necesidades. STRIDEtastic representa un avance hacia redes más seguras, auditables y confiables en contextos diversos.

Palabras clave: *Redes, Meshtastic, Observabilidad, Ciberseguridad.*

---

1. Introducción

Meshtastic es un **FIRMWARE** especializado para correr en microcontroladores con interfaces *LoRa,* radios de baja frecuencia y largo alcance. La implementación contempla una pila (*stack*) de comunicación propia, que habilita formar redes distribuidas, *ad-hoc* y con topología en malla (*mesh*) (Meshtastic, 2025). Permite a los nodos enviar telemetría, información de usuario, posición, etc. todo de manera segura si es que así se configura los nodos. Estas funcionalidades han capturado la atención de aficionados y profesionales en Chile y el Mundo, que ven casos de uso en aplicaciones desde montañismo hasta uso industrial. Fernández (2025) y Cass (2024) argumentan que estas redes son populares debido a la accesibilidad de los componentes y a su facilidad de uso. También se han desarrollado distintas aplicaciones en torno a Meshtastic, como sistemas de Bulletin Board (BBS), una primitiva de red social (Cass, 2025); y como bots de comandos y de alertas automatizadas (Nanjarí, 2025).

Esta nueva ola en el uso de Meshtastic ha levantado la necesidad de aumentar el nivel de control sobre este tipo de redes, especialmente en términos de capacidad de monitoreo y observabilidad. Ejemplos más concretos van desde casos de uso industriales como la sensometría agrícola o marítima hasta incluso *awerness* militar (Aneiros, 2024). Parámetros como cantidad de nodos conectados, latencia, posición, canales de comunicación y **TOPOLOGÍA DE RED** son claves para entender cosas como son la configuración de red en un momento dado o la detección de patrones anómalos de comportamiento.

1.1. Tecnologías existentes para la obtención de inteligencia de redes Meshtastic

Actualmente existen varias aplicaciones que permiten al usuario entender cómo se conforma una red de este tipo. A continuación, se detallan cuatro aplicaciones relevantes que ayudan a esto:

1.1.1. Clientes oficiales Meshtastic

Esta es la solución por defecto para que los usuarios comunes adquieran información sobre sus redes. El usuario se conecta a la red a través de un nodo y luego conecta el nodo a través de *Bluetooth Low Energy (BLE)* o *WiFi* a una de las siguientes interfaces gráficas: (1) *Meshtastic Android* (Meshtastic, 2025), (2) *Meshtastic iOS* (Meshtastic, 2025) o (3) *Meshtastic Client* (Meshtastic, 2025). Estas interfaces gráficas (*GUIs*) permiten revisar los mensajes de los nodos en cada canal, enviar mensajes directos a otros nodos, visualizar el mapa de nodos, y configurar los mismos.

1.1.2. *Meshtastic Map*

Meshtastic Map (Cottle, 2025) es una aplicación web que se basa en una vista de mapa la cual permite al usuario observar la telemetría de cada nodo, incluyendo

información del usuario, modelo, sensores ambientales, posición, etc. Esta información es relevante, pero carece de información novedosa como topología de red y latencia. Lo mismo se puede lograr con las aplicaciones móviles de Meshtastic y un nodo. Finalmente, no cuenta con obtención activa de información, lo que significa que no publica mensajes a la red, limitando la información que es capaz de capturar.

1.1.3. *Meshtastic Network Visualization*

Meshtastic Network Visualization (Stefaniak, 2025) es una página web estática que permite observar el grafo de topología de red en diferentes *layouts*. También permite visualizar ciertas métricas de estadísticas de mensajes por tipo. Carece de muchas funcionalidades para visualizar entender la red, sin embargo, el grafo de topología de red es una característica clave para la observabilidad de red.

1.1.4. *Mesh Monitor*

Finalmente, Mesh Monitor (Hand, 2025) es la solución más completa actualmente. Esta plataforma web permite ver los nodos como un mapa, seguir la actividad de la red a nivel de estadísticas e incluso ver la actividad por canales y ver los mensajes que pudieron ser decodificados. Sin embargo, carece de una forma de visualizar la topología de red ni visualizar los paquetes uno por uno. De todos modos, cuenta con un monitor de llaves públicas duplicadas o de baja entropía, funcionalidad novedosa.

Todas estas implementaciones (1) no permiten visualizar la red de manera intuitiva, (2) no integran toda la información en una sola plataforma, (3) no permiten obtener información de la red de manera activa y (4) no permiten a los usuarios extender fácilmente las formas de visualizar la información. En definitiva, ninguna de estas centraliza monitoreo y observabilidad de redes Meshtastic.

En base a esto, el presente trabajo propone un ***Framework* de Observabilidad Extensible** que centraliza la obtención de información y análisis de redes *Meshtastic*, combinando captación, publicación automatizada, cifrado y codificación de paquetes; con procesamiento y visualización de información. Este framework habilita a usuarios y administradores, aumentando de manera centralizada la capacidad de monitoreo sobre este tipo de redes.

2. Metodología

Esta investigación tuvo como inicio un problema abierto sobre la evaluación de seguridad en redes Meshtastic. En base a esto, la metodología consistió en cuatro etapas principales: primero (1) se realizó una revisión técnica que permitió entender el funcionamiento de Meshtastic en detalle. Luego, (2) se realizó un **MODELO DE AMENAZAS** (*threat model*) y un análisis **STRIDE** sobre este tipo de redes, análisis el cual permitió levantar la necesidad de un framework de observabilidad. En tercer lugar, (3) se desarrolló dicho framework como una aplicación web, para, finalmente, (4) probar tal implementación en un caso de uso real.

2.1. Revisión técnica de la implementación de Meshtastic

La revisión técnica se enfocó en el stack de comunicación de Meshtastic para fundamentar los requisitos del framework. En la capa física (*Layer* 0) se describió el uso de **LORA CHIRP SPREAD SPECTRUM** con preámbulo de 16 *bytes*, *sync word* igual a 0x2B y **CONTROL DE ACCESO CSMA/CA** (Meshtastic, 2025).

Todos los nodos de un mesh deben compartir frecuencia, ancho de banda, factor de dispersión y tasa de codificación (*coding rate*) para poder detectar las tramas (*frames*). Sobre esta capa se construye una capa de enlace (*Layer 1*) donde cada frame

contiene un paquete de red y NodeIDs (direcciones) de origen/destino de 4 bytes, como muestra la Figura 1. Esto permite hacer direccionamiento entre nodos y hacer enrutamiento (*routing*) en capas superiores. La Figura 2 muestra cómo se relacionan la implementación de capas del *stack* de Meshtastic con el modelo OSI.

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

Figura 1: Estructura de la trama de Meshtastic en su capa de enlace.

OSI packet	OSI layer	Meshtastic layer	Protobuf serialization
Payload	Application	L4	App-specific protobuf e.g. Telemetry protobuf
Segment	Transport	L2+L3	Data protobuf
Packet	Network	L2+L3	MeshPacket protobuf
Frame	Data Link	L1	MeshPacket protobuf
Signal	Physical	L0	LoRa signal

Figura 2: Mapeo del Stack de Meshtastic con el modelo OSI.

En la capa de red (*Layers 2–3*), los *MeshPacket protobuf*, que codifican la información del paquete a transmitir, también encapsulan metadata de *packet_id*, *flags* (*hop_start*, *hop_limit*, *want_ack*, etc.), *channel_hash*, *next_hop* y *relay_node*, como se muestra en la Figura 3. Meshtastic opera con un algoritmo híbrido: "*managed flooding*" para *broadcasting* (se da prioridad a nodos con mejor métrica RSSI/SNR) y "*next-hop routing*" para mensajería directa (*Direct Messages*, DMs), manteniendo tablas locales basadas en observaciones recientes de retransmisión exitosa (Meshtastic, 2025).

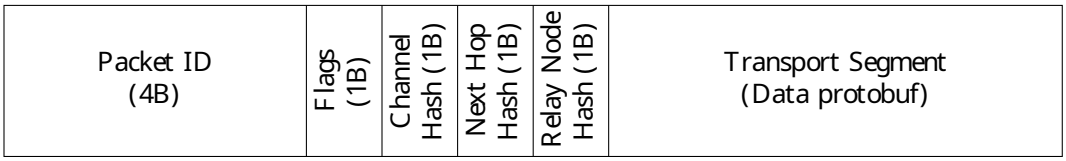


Figura 3: Estructura del paquete de red de Meshtastic.

En la capa de transporte, el *Data protobuf* porta el número de puerto (*portnum*) (0–511) que identifica a la aplicación (TEXT_MESSAGE_APP, POSITION_APP, TELEMETRY_APP, ROUTING_APP, ADMIN_APP, etc.), la carga útil (*payload*), flags como *want_response* y campos *request_id/reply_id* como se muestra en la Figura 4, lo que combina transporte confiable y direccionamiento lógico. Finalmente, cada cliente Meshtastic presenta al usuario: canales lógicos (hasta ocho por nodo) con llaves compartidas (PSK) propias, DMs sobre claves *Curve25519*, broadcasting de *NodeInfo/Position/Telemetry*, *traceroutes*, etc.

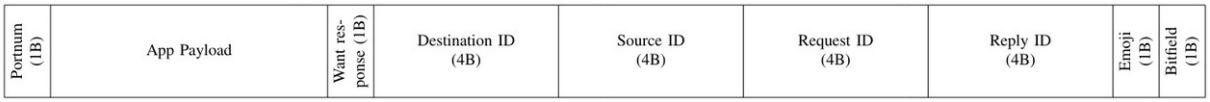


Figura 4: Estructura del segmento de transporte de Meshtastic.

Respecto a seguridad, *Meshtastic* cifra únicamente los payloads de canal mediante *AES-CTR* con claves compartidas (0, 16, 128 o 256 *bytes*). El canal primario mantiene una *PSK* por defecto (*AQ*==) y desde la versión 2.5 los mensajes directos y administrativos adoptan criptografía de curva elíptica (Meshtastic, 2025).

2.2. Análisis STRIDE

Se realizó un threat model basado en *STRIDE-for-CPS* (Khan, 2017) que ofrece una guía ligera orientada a sistemas ciberfísicos, lo que cuadra con el caso de Meshtastic: radios LoRa, interfaces TCP/BLE/serial, aplicaciones móviles y **BROKERS MQTT**. Dentro

del alcance del modelo, se cubrió nodos con firmware *Meshtastic* (*hardware*, almacenamiento local y aplicaciones), sus interfaces de gestión (BLE, serial, TCP/HTTP), los puentes hacia internet (brokers MQTT y clientes *Python/web*), los usuarios finales y las fuentes externas *GPS* (*GNSS*). Se definieron cuatro fronteras de confianza: (1) nodo local, (2) *mesh LoRa*, (3) infraestructura/broker y (4) dispositivos cliente. Los diagramas de flujo de datos (DFDs) son ideales para threat modeling (Saßnick *et al.*, 2024). Así, con este dominio en mente, se elaboró el DFD de la Figura 5, que describe siete flujos principales (LoRa, BLE, serial, TCP, MQTT, GNSS y vínculos broker-cliente) y sirvieron para ejecutar STRIDE por elemento e interacción.

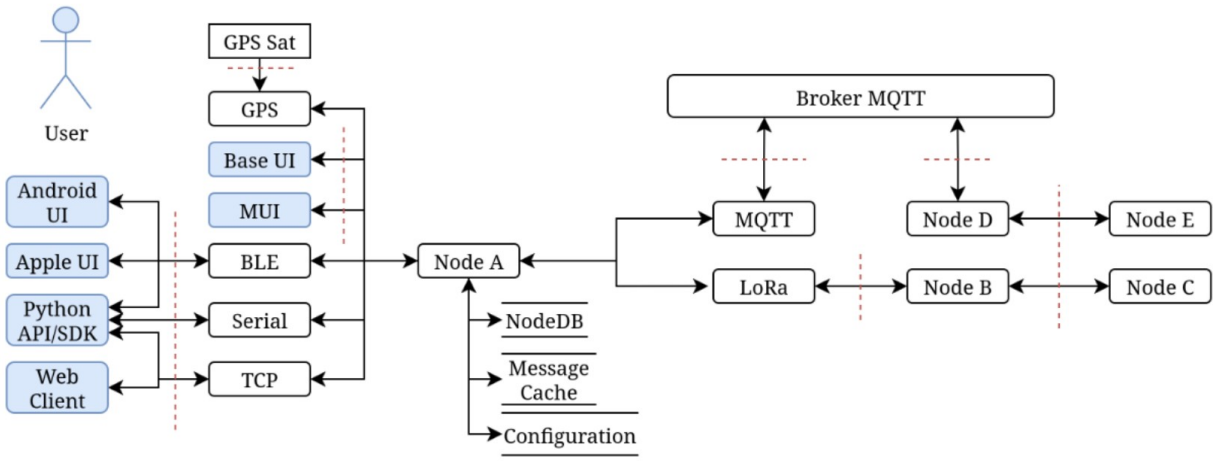


Figura 5: DFD de una red Meshtastic enfocado en el usuario de un nodo.

En base al análisis, se pudo identificar amenazas relevantes por categoría *STRIDE*. En *Spoofing*, la presencia de *PSK* por defecto y cabeceras *(headers)* en claro habilita la suplantación de identidad, degradación criptográfica (*downgrade*) de DMs a AES-CTR y el aprovechamiento de MQTT/TCP/LoRa sin autenticación para leer y publicar tráfico. En manipulación de la información *(tampering)*, un atacante con la PSK de un canal puede modificar payloads (nombre, ubicación, mensajes de texto) y metadatos de los headers (hop limit, channel hash) en tránsito. La Repudiación es factible por la falta

de *logs* firmados y de indicadores criptográficos en las GUIs. La divulgación de información *(information disclosure)* ocurre por escucha pasiva (*eavesdropping*) de LoRa, uso de llaves por defecto y brokers en donde se publican paquetes de la red. En Denegación de Servicio (DoS), destacan inundación de la red *(flooding)*, interferencia (*jamming*) y agujero de gusano (*worm hole*), que transporta mensajes de un lado de la red al otro amplificando la cantidad de saltos que puede dar los paquetes. Finalmente, la Elevación de privilegios es posible a través de puertos TCP/HTTP y acceso serial, ambos sin autenticación.

De todo este análisis se concluyó que existen vectores de ataque capaces de comprometer confidencialidad, integridad y disponibilidad de la red, pero hoy casi no hay mecanismos que permitan monitorear y detectar estos eventos en tiempo real ni correlacionarlos con métricas operacionales. Las soluciones revisadas sólo muestran información útil para el usuario común (mapas básicos, mensajes ya descifrados, telemetría) y, en el mejor de los casos, estadísticas agregadas por tipo de mensaje; sin embargo, no exponen qué canales están activos, qué nodos están inyectando tráfico, cómo se configura la topología, ni permiten disectar el tráfico paquete a paquete permitiendo reconstruir incidentes o vincular una anomalía de una métrica con algún evento específico.

2.3. Desarrollo del Framework

El framework se diseñó como una plataforma basada en contenedores capaz de capturar, inyectar y visualizar redes Meshtastic de manera centralizada. La solución integra una interfaz de aplicación (API) basada en *Django-Ninja/Celery* para orquestar servicios, un *frontend Next.js* para interacción avanzada y un stack de observabilidad basado en *TimescaleDB* y *Grafana*. El desarrollo se abordó iterativa e incrementalmente: primero se priorizó la ingestión pasiva (*sniffer* LoRa y conectores MQTT), luego los

servicios activos (inyección e interfaz de control) y finalmente la capa de visualización y métricas.

2.3.1. Requisitos

Se identificó al menos los siguientes requisitos funcionales: (1) Capturar tráfico desde múltiples interfaces y almacenarlo en una base de datos consultable y en **PCAP** disectable en Wireshark. (2) Decodificar MeshPacket y descifrar Data protobuf. (3) Publicar mensajes legítimos y controlados. (4) Exponer API para nodos, capturas, métricas y publicación de mensajes, y (5) Visualizar mapa, grafo, métricas, canales, etc.

Respecto a los requisitos no funcionales, se establecieron los siguientes como base general: (1) Procesamiento *near-real-time* (<5 s) para eventos de captura y publicación. (2) Escalabilidad horizontal de tareas. (3) Observabilidad integrada (dashboards preconfigurados, logs centralizados) y (4) Extensibilidad (permitir al usuario extender el framework con sus propios dashboards).

2.3.2 Diseño y Arquitectura

La arquitectura del framework (Figura 6) se organiza en cuatro capas: captura/inyección, backend, persistencia y observabilidad. Los conectores Serial (a un sniffer LoRa) y MQTT entregan tráfico al backend Django-Ninja, donde servicios especializados (Capture/Sniffer/Puiblisher/PKI) y *workers* Celery procesan, decodifican y orquestan trabajos de publicación. La información estructurada se almacena en TimescaleDB y los PCAP completos quedan en disco para análisis posterior, mientras que la capa de presentación combina el dashboard Next.js para exponer grafo de topología (Figura 7) y acciones, el panel administrador de Django para administrar la base de datos y los dashboards de Grafana para exponer métricas históricas y permitir extensibilidad.

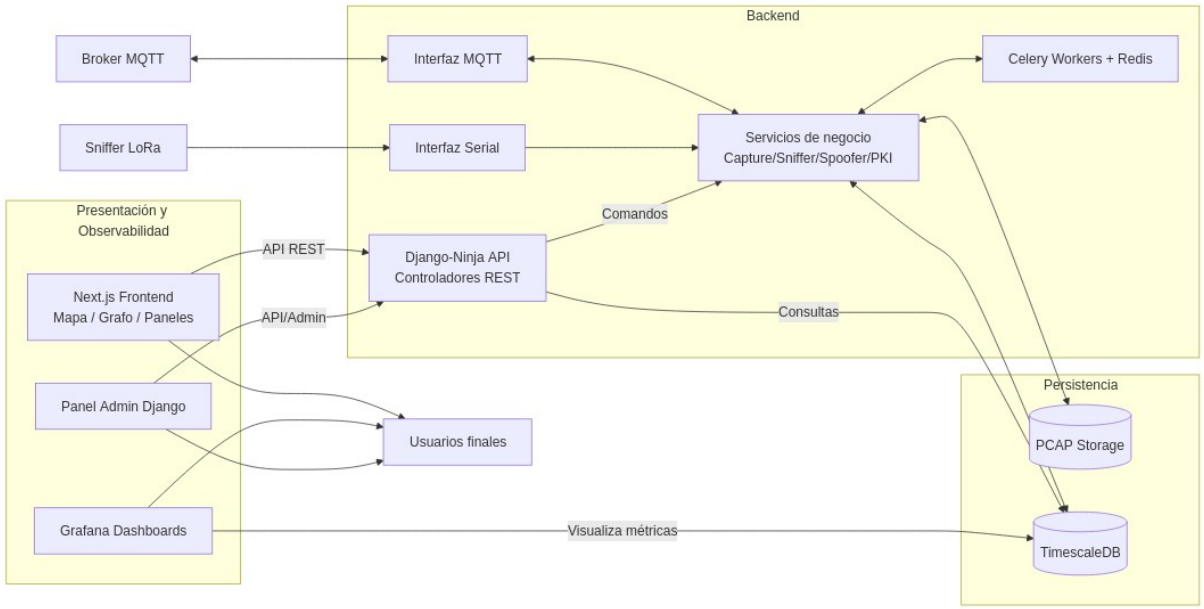


Figura 6: Arquitectura general del Framework STRIDEtastic.

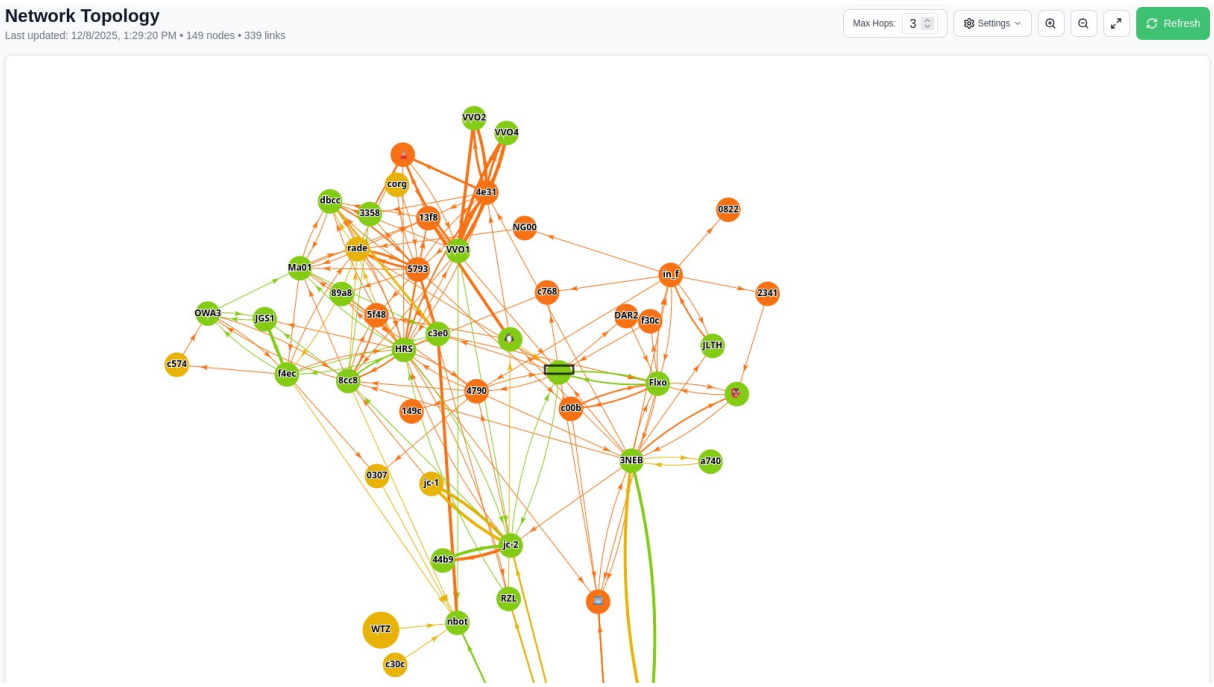


Figura 7: Grafo de topología de red mostrando parte de la mesh chilena.

2.3.3. Características implementadas

Finalmente, se implementaron las siguientes funcionalidades:

- Captura multi-interfaz capaz de conectarse a múltiples brokers MQTT y a la interfaz serial de un nodo físico.
- Capturas PCAP-NG desde UI: paneles para iniciar/detener sesiones, ver progreso y descargar PCAPs sin salir del dashboard.
- Cifrado y descifrado AES-CTR por PSK de canal.
- Cifrado y descifrado PKI Curve25519 para DMs.
- Detección de llaves débiles y duplicadas.
- Nodos virtuales e identidades criptográficas con llaves Curve25519 para publicar mensajes legítimos.
- Publicación recurrente, reactiva y configurable de paquetes.
- Dashboard Next.js con grafo interactivo de topología de red y análisis de rutas, mapa Leaflet, telemetría, controles de publicación y desglose de paquetes por direcciones.
- Dashboard Grafana extensible y configurable.

2.4. Caso de uso: monitoreo de latencia en la mesh chilena (mqtt.meshchile.cl)

Para validar el framework en un entorno real se conectó directamente al *broker* público mqtt.meshchile.cl, puerta de enlace de la comunidad Meshtastic en Chile que agrupa nodos *gateway* (conectados al broker MQTT) y nodos remotos que acceden por radio LoRa a un nodo gateway. Sobre este escenario se plantearon dos experimentos de latencia entre el framework y dos nodos: un nodo gateway y un nodo remoto.

Los experimentos corresponden a (1) Horario normal, ventanas cortas (10 minutos) y larga (1 hora). Se envían solicitudes de traceroute lo que permite comparar ambos perfiles de conectividad. (2) Evento de alto tráfico: durante la ejecución del

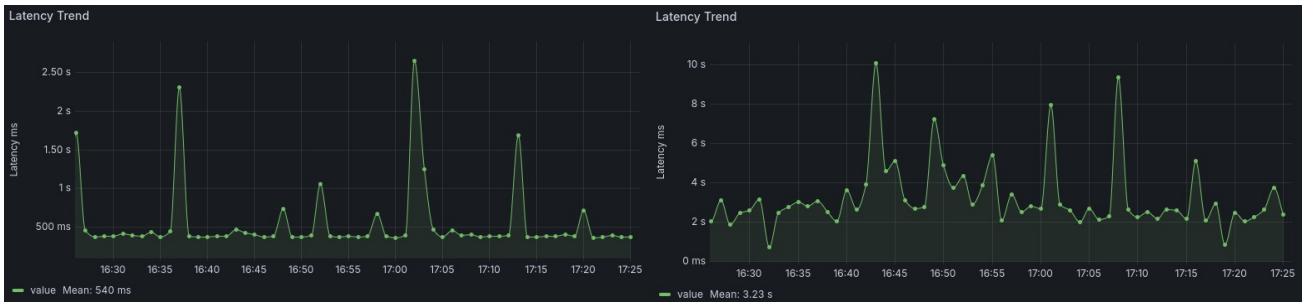
comando “#nodos” del bot comunitario “nbot”. Este comando genera una lista de ~40 páginas y existe la hipótesis de que congestiona los canales LoRa. Se mide la latencia hacia el nodo gateway y el nodo remoto antes/durante/después del comando, con el objetivo de observar el impacto sobre cada ruta.

3. Resultados y discusión

Luego de haber desarrollado el framework, se realizó los experimentos descritos en la metodología. La Figura 8 muestra una comparativa de las latencias medidas para un nodo gateway y un nodo remoto, entre las 16:40 y 16:50. De esta podemos apreciar que la latencia media es de ~500 ms para el nodo gateway y de ~3970 ms para el nodo remoto. Además, la Figura 9 muestra las latencias de los mismos nodos entre las 16:25 y 17:25. Se observan intervalos de latencia de ~400ms a ~2600ms para el nodo gateway y de ~1s a ~10s para el nodo remoto. En ambos casos, podemos ver picos en la latencia, sin embargo, se evidencia mayor variabilidad para el nodo remoto. Esto es de esperarse debido al menor ancho de banda y **COLLISION AVOIDANCE** del canal LoRa.

(a) Latencia nodo gateway (b) Latencia nodo remoto

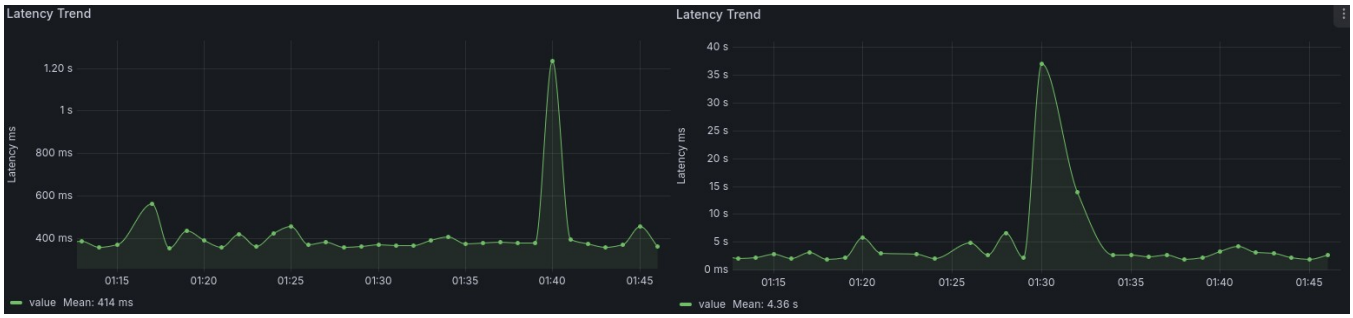
Figura 8: Resultados medición de latencia (10 minutos) para nodos gateway y remoto.



(a) Latencia nodo gateway (b) Latencia nodo remoto

Figura 9: Resultados medición de latencia (1 hora) para nodos gateway y remoto.

Por otro lado, la Figura 10 compara las latencias medidas durante el evento de congestión. El evento ocurrió a las 01:30 am y a tal hora se puede notar que hay variabilidad en la latencia con el gateway, no así para el nodo remoto que tiene un pico de latencia de ~37 segundos, lo que representa casi 10 veces sobre el promedio de 4.36 segundos.



(a) Latencia nodo gateway (b) Latencia nodo remoto

Figura 10: Resultados medición de latencia durante evento de congestión para nodos gateway y remoto.

Luego de realizado los experimentos y analizado sus resultados, se concluyó que un evento de ráfaga de mensajes es capaz de degradar la red para nodos remotos conectados por enlaces LoRa. Esto lo más probable es que se deba a que durante la

ocurrencia del evento, los enlaces LoRa se hayan congestionado debido a la alta cantidad de paquetes, el bajo ancho de banda y la política de collision avoidance.

4. Conclusiones

Las mediciones de latencia controlada mostraron que el nodo gateway mantiene latencias promedio del orden de 0.5 s (rango 0.4–2.6 s), mientras que el nodo remoto escala hasta 3.9 s promedio (1–10 s) y alcanza picos de 37 s durante ráfagas de envío de paquetes. Estas diferencias confirman que eventos de congestión en el canal LoRa degradan severamente los enlaces remotos y el framework permite detectar la ocurrencia de estos eventos, disectarlos y analizarlos.

En términos prácticos, el framework entrega un aporte concreto al problema planteado: centraliza la captura multicanal y la disección de paquetes, automatiza la medición de latencia y expone métricas históricas listas para análisis. Esto habilita a usuarios comunes y administradores a monitorear el estado y la calidad de servicio y extender los dashboards según sus necesidades. Así, STRIDEtastic representa un avance hacia redes Meshtastic más seguras, auditables y confiables en contextos diversos.

Agradecimientos

A todas las personas que marcaron mi paso por la universidad, especialmente a Maximiliano Militzer por haberme abierto las puertas al mundo Meshtastic y al Profesor Miguel Gutiérrez por darme la oportunidad de investigar, apoyarme y confiar en mi visión.

Glosario

- **BROKER MQTT**: MQTT es un protocolo de mensajería ligero basado en el modelo publicador-suscriptor, en donde el Broker MQTT corresponde a un componente central de MQTT que actúa como intermediario entre dispositivos (publicadores) y clientes (suscriptores), gestionando la entrega de mensajes, autenticación y calidad de servicio (QoS).

- **CONTROL DE ACCESO CSMA/CA, COLLISION AVOIDANCE**: Método utilizado en redes de comunicación para gestionar el acceso de múltiples nodos a un canal físico compartido, evitando colisiones de transmisión. CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) consiste en que un nodo primero “escucha” el canal antes de transmitir; si el canal está libre, envía su mensaje, y si está ocupado, espera un intervalo aleatorio antes de reintentar.

- **DATA FLOW DIAGRAM (DFD)**: Representación gráfica que muestra cómo los datos se mueven dentro de un sistema, incluyendo su origen, procesamiento, almacenamiento y destino.

- **FIRMWARE**: Software que proporciona control de bajo nivel en un dispositivo digital. Constituye la primera capa de abstracción entre el

hardware y el software, definiendo cómo el dispositivo opera y se comunica. Se almacena normalmente en almacenamiento persistente (e.g. memoria flash).

- **FRAMEWORK DE OBSERVABILIDAD DE RED**: Conjunto estructurado de herramientas, métodos y componentes que permiten obtener, procesar y analizar información sobre el funcionamiento interno de una red. Un framework de observabilidad de red integra mecanismos de monitoreo, registro de eventos (logging), medición de métricas, trazas del tráfico y visualización, con el fin de evaluar el rendimiento, detectar anomalías, diagnosticar fallas y comprender el comportamiento dinámico de los nodos y enlaces.

- **LORA CHIRP SPREAD SPECTRUM**: Técnica de modulación utilizada en comunicaciones LoRa que transmite información mediante señales de frecuencia variable llamadas chirps. Esta técnica aumenta la robustez frente a interferencias, ruido y pérdidas de señal.

- **MODELO DE AMENAZAS**: Marco estructurado que permite identificar, analizar y priorizar posibles riesgos de seguridad dentro de un sistema o red. El objetivo de un modelo de amenazas es comprender cómo un atacante podría comprometer la confidencialidad, integridad o disponibilidad de los activos, considerando vectores de ataque, capacidades del atacante y vulnerabilidades existentes.

- **PCAP (Packet Capture)**: Formato de archivo y técnica utilizada para registrar y almacenar el tráfico de red en forma de paquetes individuales.

Los archivos PCAP permiten analizar, inspeccionar y disectar comunicaciones entre nodos, ya que contienen información detallada de cada paquete, incluyendo cabeceras, payload y metadatos temporales. Se utilizan comúnmente en herramientas de análisis de redes como Wireshark para diagnosticar problemas, estudiar protocolos o reconstruir eventos de comunicación.

- **STRIDE**: Metodología de modelado de amenazas utilizada para identificar riesgos de seguridad en sistemas y redes. STRIDE es un acrónimo que representa seis categorías de amenazas: Spoofing (suplantación de identidad), Tampering (manipulación de información), Repudiation (repudiación), Information Disclosure (divulgación de información), Denial of Service (denegación de servicio) y Elevation of Privilege (elevación de privilegios).

- **TOPOLOGÍA DE RED**: Estructura o distribución en que se configuran, organizan y conectan los nodos dentro de una red de comunicaciones. Describe la disposición física o lógica de los dispositivos, así como el flujo de datos entre ellos.

Referencias

Aneiros Suárez, R., & Nocelo López, R. (2024). *Diseño e implementación de una red de mando y control para la Armada basada en Meshtastic*. https://calderon.cud.uvigo.es/items/7498bf23-98c8-4412-b35c-cc50f5d8ac70/full

Cass, S. (2024). I*t's Meshtastic!> LoRa-based Tech Brings Mesh Radio to Makers.* IEEE Spectrum, 61(6), 16-18.

Cass, S. (2025). *Putting a BBS on the Air: Microsocial Media Comes to Meshtastic*. IEEE Spectrum, 62(5), 16-18.

Cottle, L. (2025). *GitHub - liamcottle/meshtastic-map: A map of all Meshtastic nodes heard via MQTT.* GitHub. https://github.com/liamcottle/meshtastic-map

Fernández de Vega, J. J. (2025). *Estudio y diseño de una red Meshtastic® de nodos de sensores LoRa®*.

Khan, R., McLaughlin, K., Laverty, D., & Sezer, S. (2017, September). *STRIDE-based threat modeling for cyber-physical systems*. In 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe) (pp. 1-6). IEEE.

Hand, R. (2024). *Meshmonitor.org*. https://meshmonitor.org/

Meshtastic. (n.d.). *Introduction | Meshtastic.org*. https://meshtastic.org/docs/introduction/

Meshtastic. (n.d.). *Overview | Meshtastic.org*. https://meshtastic.org/docs/overview/

Meshtastic. (2024). *Mesh Broadcast Algorithm | Meshtastic.org*. https://meshtastic.org/docs/overview/mesh-algo/

Meshtastic. (2025). *Comments on Meshtastic's Encryption | Meshtastic.org*. https://meshtastic.org/docs/about/overview/encryption/comments/

Meshtastic. (2025). *Meshtastic Encryption | Meshtastic.org*. https://meshtastic.org/docs/overview/encryption/

Meshtastic. (2025, Noviembre 30). *GitHub - meshtastic/Meshtastic-Android: Android application for Meshtastic*. GitHub. https://github.com/meshtastic/Meshtastic-Android

Meshtastic. (2025). *GitHub - meshtastic/Meshtastic-Apple: Apple iOS, iPadOS & macOS Clients For Meshtastic*. GitHub. https://github.com/meshtastic/Meshtastic-Apple

Meshtastic. (2025, Octubre 16). *GitHub - meshtastic/web: Meshtastic Web Client/JS Monorepo*. GitHub. https://github.com/meshtastic/web

Nanjarí, S. (2025, Julio 29). *MeshBot - Bot Avanzado para Meshtastic Chile*. Mesh Chile. https://foro.meshchile.cl/t/meshbot-bot-avanzado-para-meshtastic-chile/30

Saßnick, O., Rosenstatter, T., Schäfer, C., & Huber, S. (2024, Mayo). *STRIDE-based Methodologies for Threat Modeling of Industrial Control Systems: A Review*. In 2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS) (pp. 1-8). IEEE.

Stefaniak, F. (2025). *Meshtastic Network Data Visualization and Processing Tools*. https://filipspl.github.io/meshtastic-network-visualization/