Backend — Networking & Protocol

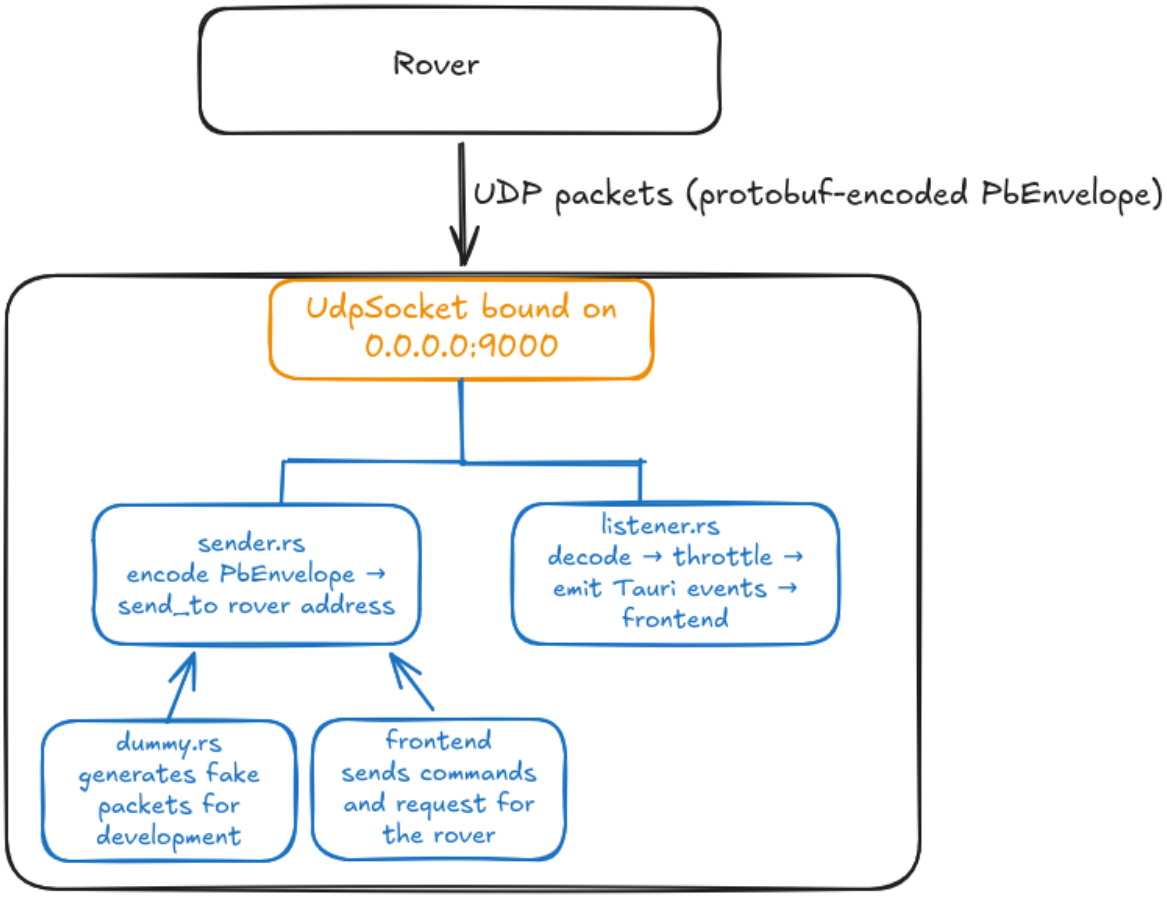
Location: src-tauri/src/network/

This module owns the UDP socket and all communication between the base station and the rover. It is split into four files: service, listener, sender and dummy.

Overview

The socket is created once in `service.rs`, wrapped in an `Arc`, and shared between the listener and the sender/dummy so they all use the same bound port.

> Graph subject to change as communication gets finalized



service.rs — UDP Socket

`UdpService` is a thin wrapper that binds a UDP socket and holds it in an `Arc<UdpSocket>` so it can be shared across async tasks. It is registered as Tauri managed state at startup so any command can access the socket via `State<'_, UdpService>`.

- Binds to address passed from lib.rs — listens on all interfaces on the chosen port
- `socket()` returns a cloned `Arc` to the socket for use in other tasks

> `socket2` allows to customize the socket, otherwise the buffer size will be set by the system and be too small

listener.rs — Incoming Packet Handler

`run_listener()` is the main receive loop. It runs for the lifetime of the app as a spawned async task. On every received UDP datagram it:

1. Decodes the raw bytes as a `PbEnvelope` using prost
2. Extracts the inner payload variant
3. Checks a per-payload **throttle** — events are forwarded to the frontend at most once every **100ms** per payload type, regardless of how fast the rover sends
4. Emits a Tauri event to the frontend with the decoded message as the payload

Throttling

Each payload type has its own independent `Throttle` instance. This prevents high-frequency streams (e.g. IMU at 50Hz) from flooding the frontend with more updates than it can usefully render.

Tauri events emitted

These are the event names the frontend can listen to with `listen()`:

> **Note for future developers:** If you add new protobufers you must add them here or you want be able to listen to them

sender.rs — Outgoing Packet Sender

> This file is work in progress

`send_envelope()` is the single outgoing send function. It takes a `PbEnvelope`, encodes it to bytes using prost, and sends it to the target address over UDP. A `hex_dump()` helper (currently commented out) can be re-enabled to log outgoing packet bytes for debugging.

Usage pattern in any command:

```
sender::send_envelope(&socket, "192.168.1.x:9000", envelope).await?;
```

dummy.rs — Development Simulator

The simulator generates realistic fake rover data so the UI can be developed and tested without physical hardware. It is started via the `start_dummy_streams` or `start_detection_sim` commands from `network.rs` and stopped with `stop_dummy_streams`.

Stream table

Each stream has an independent send interval and a generator function that produces time-varying data:

Stream	Interval	Notes
IMU	20ms (50Hz)	Sinusoidal accelerometer, gyro, magnetometer
GPS	200ms	Slow position drift around a fixed coordinate (52.2297°N, 6.8978°E)
pH	500ms	pH value oscillating around 7.0
Arm control signals	50ms	Simulated joint control inputs
Arm diagnostics	500ms	6 motors with dummy RPM/voltage
Arm feedback	100ms	Occasionally simulates an obstruction error
Arm positions	50ms	All joint angles oscillating
Arm target	200ms	Target XYZ + jaw state
Arm obstructions	300ms	
Drive diagnostics	500ms	6 drive + 4 steering motors
Drive motor	50ms	Distance to go, turning radius
Drive progress	100ms	Countdown from 10m
Sensor board diagnostics	500ms	Composite board health snapshot
Detected objects	50ms	Generates up to 12 bounding boxes for objects

Network simulation

The simulator can optionally apply **jitter** (random delay up to `jitter_ms`) and **packet loss** (random drop with probability `packet_loss`) to simulate real wireless conditions. The full simulator uses 30ms jitter and 2% packet loss; the IMU-only simulator uses no jitter or loss.