GhostWire Transports & Protocols

Contents

Transports & Protocols

What is a Transport?

- **Plain:** A transport is the "road" your messages travel on—Bluetooth, WiFi, LoRa, WebRTC, TCP/IP, and more.
- **Technical:** In GhostWire, each transport is a pluggable module implementing the `Transport` trait, allowing for runtime or compile-time enable/disable.

Supported & Planned Transports

Transport	Status	Use Case / Notes
Bluetooth	Planned	Short-range, mobile-to-mobile, disaster recovery
WiFi	Planned	Local mesh, high bandwidth, urban/rural
LoRa	Planned	Long-range, low-power, off-grid, rural/disaster
WebRTC	Planned	Browser-to-browser, NAT traversal, P2P
TCP/IP	Supported	Standard internet, fallback, bridges to servers
Stealth TCP	Supported	Obfuscated, censorship-resistant, stealth comms

---

How It Works

- The backend manages a registry of active transports.
- Messages are routed over the best available transport.
- Transports can be prioritized, failover is automatic.
- Security and privacy features are enforced across all transports.

Real-World Use Cases

- **Bluetooth:** Protesters form an ad-hoc mesh in a city square, no infrastructure needed.
- **LoRa:** Rural communities relay weather alerts over miles, off-grid.
- **WebRTC:** Journalists chat browser-to-browser, bypassing firewalls.
- **Stealth TCP:** Users in censored regions communicate without detection.

Code/Config Example: Enabling a Transport

```
// Enable LoRa transport in Rust backend
let lora = LoraTransport::new(config);
backend.add_transport(Box::new(lora));

// Enable Bluetooth transport
let bt = BluetoothTransport::new(config);
backend.add_transport(Box::new(bt));
```

Adding New Transports

- Implement the `Transport` trait.
- Register with the backend.
- Configure priorities and failover.

GhostWire: Any network, any time.