OpenTunnels: A QUIC-Native Protocol for Secure Reverse Tunneling

Channel-bound identity, ephemeral edge routing, and high-fan-in service publication without inbound network exposure

OpenTunnels Project

Technical whitepaper, June 2026

This document describes the OpenTunnels protocol profile implemented by the current Uplink reference stack. It is written as a technical whitepaper rather than an API reference; constants and code paths are included where they define interoperability or security posture.

Abstract

OpenTunnels is a reverse-tunneling protocol for publishing selected services from networks with no inbound reachability. A connector colocated with the service establishes an outbound QUIC session to a public edge; the edge maps each accepted public request to a single bidirectional QUIC stream on that connector-owned session. Identity, route declaration, and byte transport remain separate protocol surfaces: identity is admitted by a short-lived credential or self-hosted edge key, routing exists as live edge state derived from connector registration, and the connector manifest remains final authority for local upstream selection.

The central security property is session-bound authority. A connector signs its registration over the QUIC/TLS exporter channel binding, committing the accepted credential and route declaration to one live transport session. Each forwarded request begins with an OpenSession envelope containing the same transport binding and a replay nonce. Captured registrations and request envelopes fail on any other QUIC session. This paper specifies the architecture, wire objects, security model, data plane, operational profile, and comparative position of OpenTunnels relative to SSH reverse forwarding, VPN overlays, HTTP CONNECT, MASQUE, and managed tunnel services.

1. Introduction

Publishing a service from a private network looks simple until it has to be done securely. The service may sit behind NAT, a home router, a carrier network, a laptop firewall, or a development VM, with no stable public address. Port forwarding exposes the machine directly, requires operator control over the network edge, and makes certificate and firewall management part of the application lifecycle. A reverse tunnel changes the direction of establishment: the private side dials out to a reachable server, and that server relays public traffic back over the established connection.

Reverse tunneling has mature predecessors. SSH remote forwarding [9], HTTP CONNECT [10], WebSocket tunnels, VPN overlays, and managed cloud tunnels all occupy neighboring points in the design space. As public service-publication substrates, these systems frequently reconstruct the security boundary from separate mechanisms: bearer credentials, forwarded HTTP headers, ordered TCP streams carrying layered multiplexers, and control planes whose request-path role varies by deployment. OpenTunnels makes the boundary explicit. The session, credential, route declaration, and connector-local dial authority are bound by the protocol rather than inferred from surrounding automation.

OpenTunnels uses QUIC as the transport substrate. QUIC provides a single encrypted connection with independent bidirectional streams, TLS 1.3 integration, connection identifiers, and loss recovery without TCP head-of-line blocking [1,2,3]. OpenTunnels uses those properties in a reverse orientation: the connector dials the edge, but the edge opens one stream per public request. The result is a tunnel that is outbound-only from the private network while retaining request-level multiplexing, backpressure, and session-bound authentication.

2. Design Goals

The protocol is built around a short list of invariants. They are intended to be testable properties with direct implementation hooks.

- The connector never accepts unsolicited inbound connections from the public network.
- The control plane, when present, acts as issuer and observer outside request byte relay.
- A connector registration is bound to one live QUIC/TLS session and cannot be replayed elsewhere.
- Every public request maps to one bidirectional QUIC stream with an explicit OpenSession envelope.
- The connector manifest is the only dial authority for local upstreams.
- Tenant isolation is expressed as host namespace pinning, owner-scoped access policy, and manifest validation.

- Managed and self-hosted deployments share the same transport; they differ only in how connector credentials are admitted.
- Every parser and admission boundary fails closed under malformed input, saturation, or missing trust material.

These goals produce a narrow protocol surface: selected-service publication through a public edge while the private network remains closed. The profile is consequently smaller than a VPN, service mesh, browser proxy, or universal proxy framework, and its security review can focus on registration, stream admission, manifest dispatch, and bounded byte forwarding.

3. Comparative Design Space

The relevant comparison axis is the threat boundary. For each neighboring technique, the questions are where authority lives, how streams are multiplexed, whether credentials survive capture, how tenancy is represented, and whether publishing a service expands network reachability.

Method	Operational fit	Technical liability	OpenTunnels position
SSH reverse tunnels	Simple remote forwarding through a widely deployed protocol.	Remote forwards inherit SSH channel semantics and leave host policy, tenancy, issuance, and route ownership to surrounding automation.	Preserves outbound establishment while using QUIC streams, signed route declarations, and channel-bound registration.
Traditional TCP/TLS tunnel daemons	Easy public URL for a local service.	Multiplexing, replay protection, and route authorization are usually application-defined layers above one ordered byte stream.	Makes multiplexing, encryption, and per-request admission structural by mapping public requests to independent QUIC streams.
VPNs and WireGuard overlays	Excellent private network access with strong device identity [12].	Expand reachability to network scope; public ingress still needs an edge or gateway.	Publishes manifest-scoped service routes and keeps connector egress loopback-only unless explicitly expanded.
HTTP CONNECT and WebSocket tunnels	Work through existing HTTP infrastructure and proxies [10].	Often inherit TCP stream coupling; security properties depend on a layered tunnel protocol.	Uses HTTP at the public ingress but QUIC for the connector path and binds requests to the transport session.
MASQUE / CONNECT-UDP	Modern QUIC-aware proxying for UDP and IP-like flows [11].	Designed as a general proxy substrate; reverse service publication, tenant host pinning, and connector manifest authority sit outside the RFC profile.	Defines those reverse-publication semantics on the connector-edge path.
Managed tunnel services	Polished global ingress, policy, and operations; Cloudflare Tunnel is a representative example [13].	The managed edge and control product are central to the service model, and protocol semantics are often inseparable from the provider.	Separates the open transport from the product layer and supports self-hosted key-mode edges.
Kubernetes ingress and service mesh	Strong in-cluster routing, policy, and observability.	Require a reachable cluster or overlay substrate before ingress policy can be applied.	Can front a cluster service while also covering single-host and NAT-side services.
P2P NAT traversal	Direct paths can reduce latency and edge bandwidth.	Fallback, public DNS, identity, and enterprise ingress policy are harder to make uniform.	Treats the edge path as the reliable baseline; direct paths can complement it.

Table 1. Comparison by authority, transport, and operational boundary.

OpenTunnels occupies the intersection of reverse reachability, request-stream multiplexing, session-bound connector authority, ephemeral edge routing, and connector-side egress enforcement. Existing systems provide subsets of that space; OpenTunnels makes the combined set a protocol invariant.

4. System Model

The system has three required roles and one optional management layer. The connector runs near the service and initiates the tunnel. The edge owns public ingress and accepts connector sessions. The local service is the private upstream that the connector may dial. The management layer, when used, issues identity tokens, publishes a JWKS, records edge-reported liveness, and stores metering. Request bytes remain on the edge-connector data path.

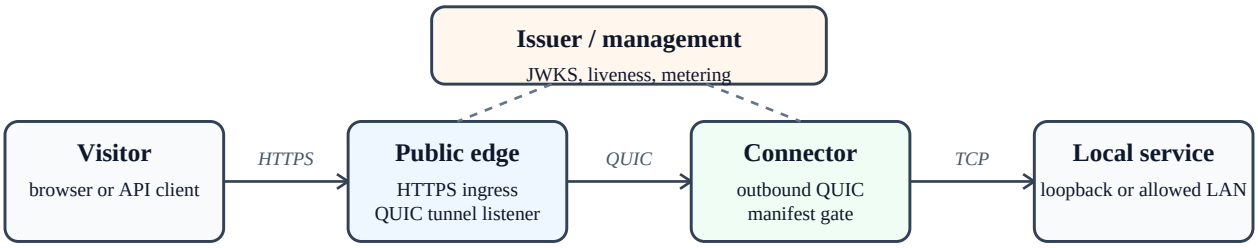


Figure 1. OpenTunnels public ingress model.

The data path is visitor to edge to connector to service. The control path is connector to issuer for a credential, and edge to issuer JWKS for offline verification. A self-hosted deployment can omit the issuer entirely: the edge admits connectors by API key while retaining the same QUIC transport, registration binding, and per-request session validation.

5. Protocol Objects and Framing

OpenTunnels uses two protocol objects on the connector-edge tunnel. TunnelRegistration is the first object on the tunnel and binds connector identity, manifest hash, tunnel epoch, connector signing key, product credential, product declaration, and binding signature. OpenSession is the first object on every request stream and carries the minimal dispatch vector the connector needs: tenant, accepted edge, route, service, transport binding, and replay nonce.

TunnelRegistration field	Purpose
connector_id	Connector identity establishing the tunnel.
manifest_hash	Hash of the connector manifest currently served.
tunnel_epoch	Connector-established tunnel instance identifier.
connector_public_key	Ed25519 public key matched against the registration signature.
credential	Opaque product credential, such as a managed identity token or self-hosted edge key.
declaration	Opaque product declaration interpreted by the edge, including declared apps and access controls.
binding_signature	Signature over a domain-separated message that commits to identity, manifest, epoch, credential, declaration, and channel binding.

Table 2. Connector registration fields.

OpenSession field	Purpose
version	Protocol version, currently 1.
org_id	Tenant or account identifier, matched against connector context and manifest.
edge_id	Public edge identity recorded as the accepted source.
route_id	Route identifier selected by edge host lookup.
service_id	Manifest service identifier to resolve the upstream.
transport_binding	QUIC/TLS channel binding expected by the connector.
nonce	Per-session anti-replay nonce.

Table 3. Per-request session envelope.

The OpenSession stream frame uses a four-byte big-endian length for the canonical OpenSession region, followed by the request payload. Fields are newline-delimited UTF-8 key-value pairs. Duplicate keys and unknown keys are rejected to prevent parser differentials, and newline-bearing values are rejected during encoding. The reference edge caps registration frames at 256 KiB and request and response heads at 64 KiB before allocation.

```
stream := uint32_be(open_session_len) || open_session_kv || payload
open_session_kv := *( key '=' value '\n' )
payload := sanitized public request bytes followed by body bytes
```

6. Registration and Channel Binding

Registration is where a generic credential becomes live-session authority. The connector dials the edge over QUIC, and both peers derive the TLS exporter value for the connection using the channel binding construction standardized for TLS 1.3 [4]. The connector signs a domain-separated message containing its connector identifier, manifest hash, tunnel epoch, product credential, product declaration, and the channel binding. The edge reconstructs that message byte for byte and verifies the signature before admitting the declaration.

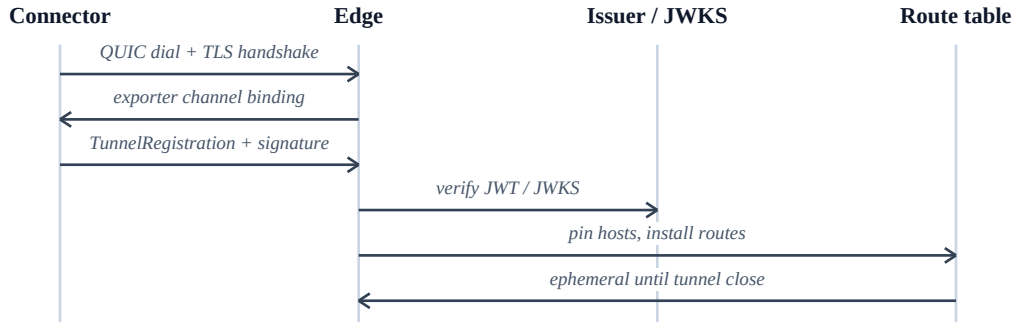


Figure 2. Registration binds the declaration to one QUIC/TLS session.

```
binding_message =
  "opentunnels-edge-registration" || LF ||
  connector_id                    || LF ||
  manifest_hash                   || LF ||
  tunnel_epoch                    || LF ||
  credential                      || LF ||
  declaration                     || LF ||
  tls_exporter_channel_binding
```

This construction enforces two invariants. Because the exporter value changes with the TLS session, a captured registration verifies only on the connection that produced it. Because the signed message covers the credential and declaration, route sets and access policy cannot be rewritten in transit. In managed deployments the edge verifies the connector JWT with EdDSA, JWKS key resolution, issuer checks, audience checks, and expiry checks [5,6,8]. Holder-of-key semantics can bind credentials to a device key with the JWT confirmation claim [7]. In self-hosted mode the credential is an edge API key; registration binding converts the accepted key into connection-specific tunnel authority.

7. Request Stream Validation

Once a tunnel is registered, the edge opens one bidirectional QUIC stream per public request. The first bytes on that stream are an OpenSession header. The connector validates the header before any local upstream connection is made.

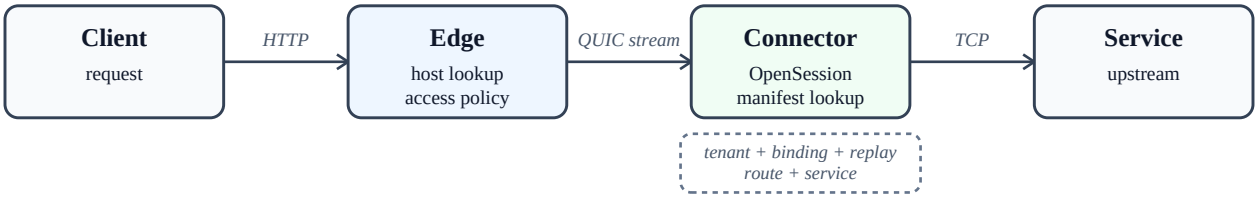


Figure 3. One public request becomes one QUIC stream and one manifest-checked local dial.

```
validate(session, manifest, context, replay_cache):
  require session.org_id == context.org_id == manifest.org_id
  require session.transport_binding == live_quic_channel_binding
  require replay_cache.insert_once(hash(canonical(session)))
  service = manifest.service_for_route(session.service_id, session.route_id)
  require service exists
  return service.upstream
```

These checks define the connector-side reference monitor. The edge supplies the accepted public host decision, and the connector applies its own manifest before any local dial occurs. Route identifiers without a manifest mapping fail before a TCP connection is opened. The replay cache provides single-use deduplication for exact repeats within a live tunnel. Cross-connection replay is stopped earlier by the transport binding: a frame captured from one QUIC connection contains the wrong exporter value for another.

8. Access Control and Tenant Isolation

The multi-tenant edge boundary has three layers. First, the edge pins every declared public host under its own domain and under the namespace derived from the verified connector identity; host claims outside that namespace fail during registration. Second, route policy is enforced before the request is forwarded. A route can be public, deny-all, or gated by owner identity, invited account, team account, source condition, or bearer token. Third, the connector manifest independently rejects off-manifest requests even when an edge is faulty or compromised.

The edge strips untrusted forwarding and internal identity headers from public requests. Only the edge may re-inject values such as X-Forwarded-For, X-Real-IP, OpenTunnels-Request-Id, OpenTunnels-Route-Id, and OpenTunnels-Origin-Identity. In a trusted proxy deployment, source IP resolution depends on an explicit trusted hop configuration; client-supplied forwarding chains are treated as untrusted input.

Invariant	Enforced by
A connector serves only its namespace.	Edge verifies credential and pins hosts at registration.
A private route admits only authorized visitors.	Edge evaluates the route access policy before forwarding.
A rogue edge cannot select arbitrary local targets.	Connector validates route and service against its own manifest.
Captured request frames are connection-specific.	OpenSession transport_binding must equal the live QUIC channel binding.
Forwarded identity is edge-authored.	Edge strips and re-injects internal OpenTunnels and forwarding headers.

Table 4. Tenant and request-boundary invariants.

9. Data Plane

The public ingress side is an HTTP front end. The edge parses one request head, applies host lookup, access policy, rate limits, and header sanitation, then forwards exactly that request over one QUIC stream. The request body and response body are relayed in bounded chunks without whole-message buffering. Keep-alive public connections are safe because each request head is independently parsed and authorized. Upgrade requests such as WebSockets become opaque bidirectional splices after the validated request head.

The connector path is intentionally small. After OpenSession validation, the connector dials the manifest upstream and splices bytes until EOF. HTTP, Server-Sent Events, WebSocket upgrades, large downloads, and other byte streams share the same per-stream path. QUIC's TLS 1.3 record protection provides confidentiality and integrity for the tunnel; the stream payload needs no second application-level sealing layer [2,3].

10. Performance Envelope

OpenTunnels optimizes for high fan-in and bounded active work. The dominant steady-state case is many mostly idle connectors, each holding one persistent QUIC connection. Active traffic consumes streams and flow-control windows; idle tunnels primarily consume connection state.

Property	Reference implementation value
Connector tunnel transport	QUIC over UDP with TLS 1.3.
Maximum accepted QUIC tunnel connections	65,536.
Concurrent bidirectional streams per connection	1,024.
Connection receive window	16 MiB.
Per-stream receive window	2 MiB.
Send window	16 MiB.
Keepalive interval	10 seconds.
Idle timeout	30 seconds.
Registration frame cap	256 KiB.
Request and response head caps	64 KiB each.
Replay cache	1,048,576 entries with 5 minute TTL; fail closed on saturation.

Table 5. Current reference-implementation transport profile.

These numbers characterize the current interoperable profile and expose the intended operating shape: large fan-in, bounded per-request memory, and explicit admission control. Horizontal scale currently uses hostname-affinity

sharding. A public request must arrive at the edge that holds the connector tunnel for that hostname. The design keeps edge state reconstructable and route recovery simple while leaving DNS and placement discipline to operators until peer forwarding or another cross-edge routing design is introduced.

The implementation disables UDP Generic Segmentation Offload by default because several virtualized network stacks have advertised GSO and then dropped segmented packets. That is a conservative throughput tradeoff: on well-characterized Linux hosts, opt-in GSO can improve bulk transfer throughput, but the default favors correctness across containers and desktop virtualization.

11. Operational Model

Edge state is reconstructable from live connectors. If an edge restarts, its route table is empty until connectors reconnect and send fresh registrations. The connector reconnect loop refreshes credentials before re-registration, so short token lifetimes still support long-running formations. During the restart window, unknown or unavailable hosts fail closed with unavailable or unknown-route responses; stale route state is never authoritative.

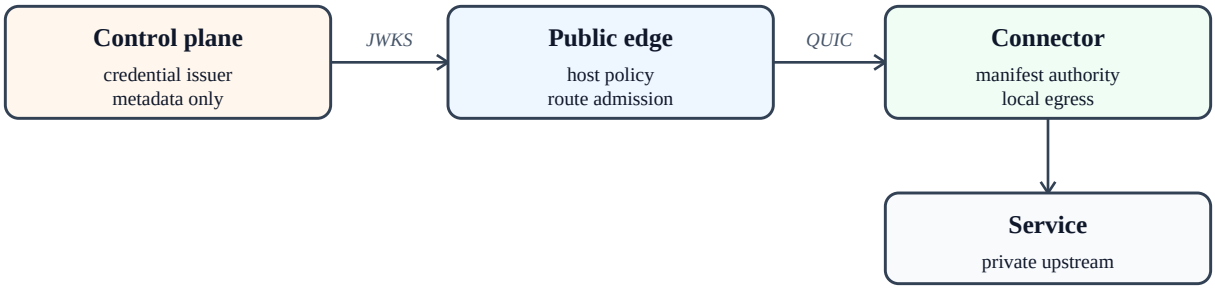
Managed edges use a control-plane issuer and optional telemetry hooks. The edge verifies connector and visitor tokens offline against the JWKS, pushes liveness reports to the management service when configured, and exports usage metrics through an OpenTelemetry path. Self-hosted edges admit connectors with API keys and omit the management stack. In both modes, visitor traffic traverses edge to connector directly; management handles issuance, liveness, and metering metadata.

12. Security Analysis

Adversary	Capability	Primary defenses
Anonymous internet client	Sends arbitrary bytes to public HTTP(S) and tunnel UDP ports.	TLS termination, strict parsing, admission control, timeouts, route policy, and header sanitation.
Malicious tenant	Owns a valid account credential and connector key.	Namespace host pinning, owner-scoped routes, account-scoped tunnel replacement, and connector manifest validation.
Network attacker	Observes or injects packets between connector and edge.	QUIC/TLS 1.3, hostname-verified edge certificate, registration signature over channel binding, and bound OpenSession frames.
Faulty or compromised edge	Speaks the connector protocol after the connector dialed it.	Connector validates live transport binding and refuses routes outside its manifest.
Hostile local config	Controls a checked-in service declaration.	Loopback-only default egress, typed config parsing, no shell execution, and per-dial DNS/IP revalidation.
Resourced DoS actor	Creates high connection or request pressure.	Bounded admission, registration deadlines, head caps, rate limits, and fail-closed replay cache saturation.

Table 6. Threat model summary.

The trust model has explicit roots. A compromised identity issuer can mint credentials. A compromised connector host can serve malicious local services or leak its own secrets. A sufficiently resourced denial-of-service attack can exhaust public edge capacity. OpenTunnels narrows these risks through cryptographic session binding, connector-side manifest enforcement, bounded parsing, fail-closed replay handling, and stateless edge recovery; the underlying trust anchors remain visible.



The management plane issues authority; the data plane enforces and relays without a control-plane proxy hop.

Figure 4. Authority boundaries in the managed deployment.

13. Implementation Profile

A conformant implementation for the current profile should implement the following behaviors.

- Use QUIC with TLS 1.3 and expose the RFC 9266 tls-exporter channel binding to the registration and request paths.
- Encode TunnelRegistration and OpenSession in canonical key-value form with duplicate-field and unknown-field rejection.
- Require the registration signature to commit to credential, declaration, tunnel epoch, manifest hash, and channel binding.
- Open one bidirectional stream per request and place OpenSession before the request payload.
- Reject an OpenSession whose tenant, transport binding, replay id, route id, or service id fails connector-side validation.
- Treat the connector manifest as the only authority for local upstream selection.
- Strip client-supplied forwarding and OpenTunnels internal headers before forwarding public requests.
- Bound all attacker-controlled frame and head reads before allocation.

The reference implementation separates reusable protocol mechanics from the Uplink product model. The OpenTunnels runtime owns connector session validation, tunnel registration, QUIC adapters, and hardening utilities. The Uplink model owns app declarations, host naming, access policies, and edge-side route controls. That split is important for protocol review: channel binding and manifest enforcement are protocol mechanics; billing, dashboards, and account UX are product concerns.

14. Limitations and Open Work

- The current baseline uses hostname-affinity edge shards; cross-edge peer forwarding remains open work.
- High-fan-in and reconnect-churn tests exist as load tooling but should become routine capacity gates.
- Public edge DoS remains a capacity and abuse-management problem beyond per-node admission and rate limits.
- Self-hosted API-key mode is intentionally simple; leaked keys remain bearer secrets until rotation.
- Opt-in GSO should be benchmarked per host class before production enablement.
- A formal third-party cryptographic review of the registration and OpenSession binding would be valuable before treating the profile as frozen.

15. Conclusion

OpenTunnels contributes a compact protocol profile for reverse service publication over QUIC. A connector-established session becomes the unit of authority: registration commits identity, credential, declaration, manifest, epoch, and channel binding; each request stream carries an OpenSession bound to the same transport; the edge holds ephemeral routes derived from live connectors; and the connector manifest remains the final upstream authority.

The resulting system fits between SSH reverse forwarding, managed tunnel products, VPN overlays, and general proxy substrates. It publishes selected services through a public edge while preserving a closed private network boundary, giving operators a protocol surface that is small enough to audit and explicit enough to scale.

References

[1] IETF. RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport. https://www.rfc-editor.org/rfc/rfc9000

[2] IETF. RFC 9001: Using TLS to Secure QUIC. https://www.rfc-editor.org/rfc/rfc9001

[3] IETF. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446

[4] IETF. RFC 9266: Channel Bindings for TLS 1.3. https://www.rfc-editor.org/rfc/rfc9266

[5] IETF. RFC 7519: JSON Web Token (JWT). https://www.rfc-editor.org/rfc/rfc7519

[6] IETF. RFC 7517: JSON Web Key (JWK). https://www.rfc-editor.org/rfc/rfc7517

[7] IETF. RFC 7800: Proof-of-Possession Key Semantics for JSON Web Tokens. https://www.rfc-editor.org/rfc/rfc7800

[8] IETF. RFC 8037: CFRG Elliptic Curve Diffie-Hellman and Signatures in JOSE. https://www.rfc-editor.org/rfc/rfc8037

[9] IETF. RFC 4254: The Secure Shell (SSH) Connection Protocol. https://www.rfc-editor.org/rfc/rfc4254

[10] IETF. RFC 9110: HTTP Semantics. https://www.rfc-editor.org/rfc/rfc9110

[11] IETF. RFC 9298: Proxying UDP in HTTP. https://www.rfc-editor.org/rfc/rfc9298

[12] Jason A. Donenfeld. WireGuard: Next Generation Kernel Network Tunnel. https://www.wireguard.com/papers/wireguard.pdf

[13] Cloudflare. Connect networks with Cloudflare Tunnel. Accessed 2026-06-29. https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/

[14] OpenTunnels reference implementation and Uplink protocol notes, current repository snapshot: stacks/uplink/docs/notes, stacks/uplink/crates/opentunnels, stacks/uplink/crates/opentunnels-runtime, and stacks/uplink/crates/opentunnels-quic.