SlipNet — Complete User Guide

> Official channel: @SlipNet_app Source code: https://github.com/anonvector/SlipNet SlipNet is **NOT** on Google Play, the App Store, or any other marketplace. Only download it from the official channel or GitHub.

Table of Contents

---

1. What SlipNet is — and why it exists

SlipNet is a free, **source-available** anti-censorship tool. The main client is an Android app, with a matching command-line client for macOS, Linux, and Windows. It tunnels your internet traffic through different protocols — DNS, QUIC, SSH, HTTPS, VLESS, or Tor — so you can reach the open internet even on networks that block, throttle, or deep-packet-inspect normal traffic.

Unlike a typical VPN that ships with a single protocol, SlipNet offers many **transports** so you can pick the one that survives on whatever network you're stuck on:

- If your ISP only lets DNS traffic out, use a DNS tunnel — DNSTT, NoizDNS, or VayDNS.
- If only HTTPS gets through, use NaiveProxy or VLESS fronted through a CDN like Cloudflare.
- If your network does deep traffic inspection and recognises tunnel-shaped flows, layer SSH on top of any of the above for zero DNS leaks and an extra layer of encryption.
- In the worst case, fall back to Tor with Snowflake, obfs4, or Meek bridges (Full edition only).

SlipNet sits in the same family as projects like Tor, Psiphon, and Outline VPN, but with one critical difference: SlipNet is **pure client software**. There is no central network or company-run infrastructure — you either run your own server (a few minutes with SlipGate) or get a config from someone you trust. That model has three real advantages:

- **Privacy.** Your traffic exits through a server you or someone you know controls — not an unknown company's exit node.
- **Resilience.** With no central infrastructure to filter, blocking the project is impractical. If a single server gets blocked, swap the IP and update one DNS record.
- **Flexibility.** You choose the protocol, the CDN, the domain, even the country your traffic exits from — control that no commercial VPN gives you.

The full source code lives on GitHub under a **source-available** license — readable, studyable, and open to contributions, but not licensed for redistribution or republishing on app stores. SlipNet is **not** on Google Play, the App Store, or any other marketplace; the only official sources are the @SlipNet_app Telegram channel and the project's GitHub repository.

---

2. Supported tunnel types

Tunnel	Protocol	What it does	Best for
DNSTT	DNS (KCP + Noise)	Tunnels traffic inside DNS queries	Networks that only allow DNS
NoizDNS	DNS (KCP + Noise)	DNSTT with DPI evasion (base36/hex, CDN prefix stripping)	Heavily inspected networks
VayDNS	DNS (KCP + Curve25519)	Optimized DNS tunnel; tunable record types, QNAME length, rate limit	Advanced users on flaky DNS paths
Slipstream	QUIC	High-performance QUIC tunnel	Fast, clean networks
SSH	SSH	Standalone SSH tunnel	Simple, encrypted, leak-proof
NaiveProxy	HTTPS (Caddy + Chromium TLS)	HTTPS with authentic Chrome TLS fingerprint	DPI bypass over HTTPS
VLESS	WebSocket (over TLS or plain)	VLESS-over-WebSocket via CDN (e.g. Cloudflare)	CDN fronting, very common in Iran
DOH	DNS over HTTPS	Encrypts DNS only — no traffic tunnel	Bypassing DNS-only blocks
Tor	Tor + Snowflake/obfs4/Meek	Anonymity network	Strong anonymity
+ SSH variants	(DNSTT/NoizDNS/VayDNS/Slipstream/NaiveProxy/VLESS) + SSH	Adds SSH on top — zero DNS leaks, extra encryption	Maximum security

How to choose:

- New user, no idea? → **DNSTT** (default).
- DPI is aggressive? → **NoizDNS** or **NoizDNS + SSH**.
- Need to fly under Cloudflare? → **VLESS** or **NaiveProxy**.
- Want raw speed on a clean link? → **Slipstream**.
- Need anonymity? → **Tor** (Full edition only).

---

3. Editions: Full vs. Lite

SlipNet ships in two editions — the difference is the size of the APK and which protocols are bundled.

Feature	Full	Lite
DNSTT, NoizDNS, VayDNS	✓	✓
Slipstream (QUIC)	✓	✓
SSH	✓	✓
DoH	✓	✓
VLESS	✓	✓
NaiveProxy	✓	—
Tor (Snowflake / obfs4 / Meek)	✓	—
Approx. APK size	~50 MB	~20 MB

Full is the default recommendation. **Lite** is for slow connections, low-storage phones, or when Tor and NaiveProxy aren't needed.

---

4. Installing the APK

1. Open the official Telegram channel @SlipNet_app or the GitHub Releases page.
2. Download the APK that matches your phone's CPU. Most modern phones are `arm64-v8a`. If you don't know, download the **universal** APK.
3. On your phone: **Settings → Security → Install unknown apps** → allow your browser/Telegram to install APKs.
4. Tap the downloaded APK and install.
5. Open SlipNet. On first launch, Android will ask for **VPN permission** — tap **OK**.
6. Recommended one-time setup: **Settings → Battery optimization → Don't optimize SlipNet**, so Android doesn't kill it in the background.

> ⚠️ If you find SlipNet on Google Play, App Store, or any other store, it is **not** ours. Don't install it.

---

5. You need a server — use SlipGate

SlipNet is a *client*. To use it you must connect to a **server** running a compatible tunnel.

The official, supported way to run a server is **SlipGate** — a one-command Linux installer that sets up every protocol SlipNet supports.

> https://github.com/anonvector/slipgate

A separate operator guide for SlipGate is available alongside this document. In short: get a $5/month VPS, point a domain at it, run a single install command, create a user, and SlipGate prints a `slipnet://` link you paste straight into the app.

If you don't run your own server, you can:

- Get a `slipnet://` link from a friend who runs SlipGate, or
- Use a public test link sometimes shared in @SlipNet_app.

---

6. Adding your first profile

There are three ways to add a server profile.

A) Paste a `slipnet://` link (easiest)

1. Open SlipNet → tap the **+** button at the bottom right.
2. Choose **Import from URI**.
3. Paste the `slipnet://...` link.
4. Done — the profile appears in your list.

B) Import a JSON file

If a friend exported their profiles (encrypted or plain JSON):

1. Tap the menu (⋮) on the home screen → **Import Profiles**.
2. Select the `.json` file.
3. If it's encrypted, enter the password they shared.

C) Build a profile manually

Useful when your provider gives you raw values (domain + key) instead of a URI.

1. Tap **+** → **Add Profile**.
2. Give it a **Name**.
3. Pick a **Tunnel Type** (the form changes per type — see §11 for what each one needs).
4. Fill in the required fields.
5. Tap **Save**.

---

7. Connecting and disconnecting

1. Tap a profile in the list to select it (a checkmark appears).
2. Tap the big **Connect** button at the bottom.
3. The first time, Android shows a VPN-permission dialog → **OK**.
4. When the icon turns green and you see live upload/download numbers, you're connected. All your traffic now flows through the tunnel.

To disconnect, tap **Connect** again. To switch profiles, disconnect first, pick a different profile, then reconnect.

Quick toggle without opening the app:

- Pull down the notification shade → tap the **SlipNet Quick Settings tile** (you may need to add it from the tile picker the first time).
- Or use the **home-screen widget** for one-tap connect/disconnect.

Live ping test: while a profile is selected, tap **Real Ping** or **Simple Ping** to measure latency. **Sort by Ping** orders the profile list fastest-first.

---

8. The DNS Resolver Scanner

DNS-tunnel profiles (DNSTT, NoizDNS, VayDNS) only work through resolvers that obey a few RFC behaviours and don't tamper with replies. Many ISP resolvers fail one or more of those checks — they hijack NXDOMAIN, strip EDNS, refuse long names, or block tunnel-shaped traffic at DPI. The **Scanner** finds resolvers that pass.

Open: **menu → DNS Resolver Scanner**.

The scanner has four big things you control:

1. **Scan Mode** — what kind of test to run
2. **Configuration** — test domain, ports, timeouts, parallelism
3. **IP source** — where the candidate IPs come from
4. **Results & Apply** — review and push the survivors back into your profile

---

8.1. Scan modes

A toggle at the top picks one of four modes. Each is a different trade-off between speed, depth, and certainty.

Simple

"Scans DNS resolvers and automatically tests each one through the tunnel. Only resolvers that pass the tunnel test are shown."

The one-tap mode for everyone. The scanner runs the DNS-compatibility probe **and** an end-to-end tunnel test in one pipeline, and only displays resolvers that genuinely deliver traffic. Use this if you don't want to think.

Requires a profile with a valid public key (so the tunnel test has a real server to talk to).

Advanced

"Scan DNS resolvers first, then optionally run tunnel test separately."

The classic two-stage flow. Stage 1 is the DNS probe and gives every resolver a 0–6 score (see §8.5). You can review results, sort, filter, and then optionally trigger an E2E tunnel test on resolvers that passed the score threshold. Use this when you want to inspect the probe details, or when you don't have a profile yet.

E2E

"Tests each resolver directly through the tunnel, skipping DNS compatibility checks. Slower but tests real connectivity."

Skips the DNS probe entirely. For each candidate IP it just opens a real tunnel and tries an HTTP request. Use this when you already have a list of "known DNS-tunnel-capable" IPs and you only want to know which ones currently reach your specific server fast.

Prism

"Server-verified scan: only resolvers that cryptographically prove they reach your specific server are shown. Requires SlipGate installed on your server."

The strongest mode. Sends HMAC-authenticated probes that only a real SlipGate server can sign correctly, so a resolver that "works" but actually delivers traffic to an attacker's middlebox will fail the check. Each resolver is probed N times; resolvers that get ≥ *threshold* signed responses pass. Use this when you suspect transparent-proxy interception or want defense against tunnel hijacking.

Only available when your selected profile has a valid public key and your server runs SlipGate.

8.2. Configuration panel

These fields appear above the scan button. Most have sane defaults — only touch them if a scan is failing.

Common to Simple / Advanced

- **Test Domain** — the name the probe will query. Use *your tunnel's domain* (e.g. `t.example.com`); a resolver that can route long random subdomains under it is the one that can carry a real tunnel. Locked profiles fall back to the profile's own domain.
- **Port** — usually `53`. Override if your server runs on a non-standard DNS port.
- **Timeout (ms)** — per-resolver timeout for the DNS probe. Default 3000. Drop to 1500 if your candidate list is huge and full of dead IPs.
- **Workers** — parallel probes. Default 50. Lower it on flaky links to avoid drops.

E2E-only

- **Resolver Port** — port the resolver listens on (53 unless overridden).
- **E2E Timeout** — per-resolver tunnel-test timeout. Default 15000 ms.
- **E2E Concurrency** — parallel tunnel tests. 1–10. **Slipstream max 1** because QUIC tunnels can't share a port.
- **Test URL** — what the tunnel will fetch to prove connectivity. Default `http://www.gstatic.com/generate_204` (returns 204 No Content — small, fast, doesn't depend on DNS resolution inside the tunnel).
- **Full Verification** — when on, fetches the test URL through the live tunnel; when off, only confirms the tunnel handshake completes.

Prism-only

- **Probes** — number of HMAC-signed requests sent to each resolver. More probes = stronger statistical confidence, slower scan. Default 5.
- **Pass threshold** — minimum number of probes that must come back correctly signed. Default 2.

- **Timeout (per resolver)** — overall budget split across probes. The app rejects settings where per-probe time would drop below 200 ms.
- **Response size** — bytes the server should pad responses to (0 = server default). Useful if your server is configured for a specific reply size.
- **Pre-filter dead resolvers** — runs a quick DNS check first to drop unresponsive IPs before spending probe budget on them. Recommended for large lists.

Scan Transport (Advanced and Simple only) — `UDP`, `TCP`, or `Both`. **Both** runs UDP first, then retries only the resolvers that didn't pass over TCP. Useful in networks where UDP DNS is rate-limited or filtered, since some resolvers respond fine over TCP. Each result shows badges (`UDP`, `TCP`) for which transport it survived.

8.3. IP source panel

Tabs across the bottom decide *where the candidate IPs come from*. Pick exactly one source per scan.

- **Default** — the app's built-in curated list of resolvers known to work historically. Smallest, fastest scan; good first attempt.
- **Import** — load IPs from a file (`.txt`, one per line or comma-separated). Use this to scan a list someone shared, or your own previous results.
- **Country** — generates random IPs from a chosen country's address allocations. Pick a country and a sample count (default 2000, max 100 000). For Iran users this is often the goldmine — domestic resolvers usually beat foreign ones for tunnel purposes because local DPI doesn't bother inspecting them as aggressively.
- **Custom** — paste a CIDR (`5.144.0.0/14`), an IP range (`5.144.0.1–5.147.255.254`), comma-separated IPs, or a single IP. The preview counter shows how many IPs you'd be scanning before you commit.
- **IR DNS Ranges** — Iranian ISP DNS allocations grouped by `/8` octet. Pick which `/8` groups to include (the count badge shows total IPs per group). Heavier than IR DNS Lite but more thorough.
- **IR DNS Lite** — a pre-curated subset of Iranian DNS ranges, smaller and faster than the full IR DNS list. Good default for Iranian users.
- **Recent DNS** *(button at the bottom)* — replays the IPs from the last scan, useful for iterating on settings without regenerating the candidate set.
- **Load Last Scan IPs** — pulls back resolvers that already passed in a previous run (separates "Working IPs" from "E2E Passed IPs" so you can re-run only the proven set).

Toggles that apply to most sources:

- **Shuffle** — randomise scan order. Default on. Helps avoid hot-spotting one ISP.
- **Expand neighbours** — when scanning a custom range, also probe a few IPs on either side of any hit, since DNS resolvers often live in clusters.

8.4. Results & Apply

After the scan finishes, tap **View Results**. Each resolver row shows:

- The resolver IP and the score (0–6) from the DNS probe.
- A breakdown badge: `NS✓ TXT✓ RND✓ DPI✓ EDNS✓(1232) NXD✓` — see §8.5.

- For Simple/E2E modes: tunnel setup time, HTTP latency, total round-trip.
- For Prism: `Probes 4/5` (passed/total) and a `Verified` badge.
- Transport badges (`UDP`, `TCP`) showing which the resolver survived.
- `WORKING` / `CENSORED` / `TIMEOUT` / `ERROR` status.

You can:

- **Sort** by score, by latency, or by E2E success.
- **Filter** by passed-probe count (Prism), by score range (Advanced), or by "all working".
- **Search** by IP substring.
- **Copy** or **Export** visible IPs (E2E-passed only / Stage-1 working only / your selection).
- **Re-test Tunnel** on a single result — useful if the network just changed.
- **Apply Selected** — pushes the selected resolvers into the active profile (max 8). The next time you connect, those resolvers are used.

You can resume a scan that was interrupted (the app prompts on next entry). E2E tests can be paused/continued mid-run.

8.5. The 0–6 score, decoded

The DNS-compatibility score is the count of these six probes that passed. Each is one binary point.

Probe	What it tests	Why it matters
NS	Resolver follows `NS` records and returns A records for the parent zone	Tunnel servers are typically delegated subdomains; a resolver that ignores NS delegation can't even find the tunnel server
TXT	Resolver returns TXT records	DNSTT/NoizDNS/VayDNS encode return data in TXT (or other types — see VayDNS settings); broken TXT = no downstream traffic
RND	Resolver actually queries upstream for random subdomains it has never seen	Some "smart" resolvers cache aggressively or short-circuit unknown names — a DNS tunnel makes up a fresh subdomain per query, so this must work
DPI	A `dnstt`-style long base32-encoded TXT query gets through	Detects DPI boxes that fingerprint tunnel-shaped DNS by entropy/length/record-type; if this fails, your queries will be dropped silently
EDNS	Resolver supports EDNS0 large responses (>512 bytes); badge shows max payload (512 / 900 / 1232)	Tunnel throughput depends on response size — 1232 is the dnsflagday.net target; resolvers stuck at 512 will be brutally slow
NXD	Resolver returns proper `NXDOMAIN` for non-existent names instead of hijacking to a parking IP	Hijacking resolvers spoof answers, which corrupts the tunnel framing — a hijacker scores `NXD✗` and is unusable

A score of **6** is fully tunnel-compatible. **5** is usually fine — the missing point is often EDNS payload size, which only hurts speed. **3 or below** is rarely worth using. The Pass threshold field lets you set the minimum.

8.6. Practical recipes

- **Iranian user, default profile, just want to connect**: Simple mode → IR DNS Lite source → Start.
- **Iranian user, exhaustive search**: Simple mode → IR DNS Ranges source → pick all groups → bump sample count to 5000.
- **Foreign tunnel-tester**: Advanced mode → Country source → set country → score threshold 5.
- **You suspect your ISP is intercepting**: Prism mode → IR DNS Lite → 8 probes / threshold 5.
- **You already have a working list, just verify speed**: E2E mode → Import that list → high concurrency.
- **UDP DNS is being rate-limited**: any mode → Scan Transport = `Both`.

---

9. Profile Chains — combining tunnels

The **Chains** tab lets you stack multiple profiles end-to-end. Example chains:

- `NoizDNS → SSH → destination` — DPI-resistant transport with SSH on top, zero DNS leaks.
- `VLESS (Cloudflare) → SSH` — CDN fronting plus encrypted final hop.
- `Tor → NaiveProxy` — anonymity plus HTTPS exit.

Build a chain:

1. Open the **Chains** tab.
2. Tap **+ New Chain**.
3. Add profiles in order — first hop at the top, last hop at the bottom.
4. Save and select the chain like any normal profile.

When you connect, the app validates that the chain is consistent (compatible transports, no loops) and streams the chain as one tunnel.

---

10. Settings reference

Open **Settings** to tune the app.

Connection

- **Auto-connect on boot** — reconnect when the phone restarts.
- **Auto-reconnect** — reconnect if the VPN drops unexpectedly.
- **Auto-disconnect after** — set an idle timeout.
- **Block all if VPN drops** — kill switch (no leaks).
- **VPN MTU** — lower it (e.g., 1280) if some sites won't load.
- **DNS Workers** — fewer = more stable on restricted networks; "per-query" creates a fresh connection per lookup.
- **Disable QUIC** — forces apps to use TCP; often noticeably faster over a tunneled link.

Routing

- **Split Tunneling** — choose which apps use the VPN (allowlist or bypass).
- **Domain Routing** — only specific domains go through the tunnel.

- **Geo-bypass** — route traffic to IPs/sites in your selected country *outside* the VPN, so local websites stay fast and unaffected by the tunnel.
- **Bypass VPN** — let certain apps skip the VPN entirely.
- **Append HTTP Proxy to VPN** — also expose a local HTTP proxy.

DNS

- **Global DNS Resolvers** — override resolvers for all profiles.
- **Remote DNS Server** — what runs on the far side of the tunnel.
- **DNS Resolver Scanner** — see §8.

Security

- **SSH Cipher** — prefer AES-128-GCM, ChaCha20, or AES-128-CTR (legacy).
- **Bandwidth Limit** — cap upload/download.
- **Hotspot mode** — share the tunnel with other devices over Wi-Fi.

Appearance

- **Dark mode** — Dark / AMOLED Dark / Auto.

Diagnostics

- **Debug logging** — verbose logs for support.
- **Device ID / IP** — copy for support requests.
- **Check for updates** — pulls the latest release.

---

11. Per-tunnel quick reference

What each tunnel type asks for when you build a profile manually.

DNSTT / NoizDNS

- **Domain** — your tunnel subdomain, e.g., `t.example.com`
- **Public Key** — server's Curve25519/Noise key (hex)
- **DNS Transport** — UDP / TCP / DoT / DoH
- **DNS Resolvers** — IPs of resolvers to send queries through
- *NoizDNS:* **Stealth Mode** — slower but harder to detect

VayDNS

- Domain + Public Key (as above)
- **Record Type** — TXT / CNAME / A / AAAA / MX / NS / SRV
- **Max QNAME Length** — wire size of each query
- **Rate Limit (RPS)** — outgoing queries per second
- **Idle Timeout / Keep-Alive / UDP Timeout**
- **ClientID Size** — must match server (default 2; 8 in DNSTT-compat mode)

Slipstream

- Domain + Public Key

- **Congestion Control** — BBR / DCUBIC
- **Keep-Alive Interval**
- **Authoritative Mode**, **GSO**

SSH (standalone or as the last hop)

- **SSH Host / Port** (default 22)
- **Username + Password** *or* **Private Key** (with passphrase)
- **Cipher** — AES-128-GCM / ChaCha20 / AES-128-CTR

SSH transport options (work with any SSH-based tunnel):

- **SSH over TLS** — wrap SSH in TLS with custom SNI (domain fronting).
- **HTTP CONNECT proxy** — route through an HTTP CONNECT proxy with custom Host header.
- **SSH over WebSocket** — `ws://` or `wss://` with custom path & Host (Cloudflare-friendly).
- **SSH Payload** — send raw bytes before the SSH handshake to disguise it. Supports `[host]`, `[port]`, `[crlf]`, `[cr]`, `[lf]`.

NaiveProxy

- **Server hostname** + **port** (usually 443)
- **Proxy username / password**

VLESS

- **UUID** — your VLESS user ID (`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`)
- **Domain** — the hostname behind the CDN; used as TLS SNI and WS Host
- **Security** — `tls` (recommended) or `none`
- **Transport** — WebSocket only (raw TCP isn't supported in the app — VLESS in SlipNet is positioned as a CDN-fronted tunnel)
- **WS Path** — e.g., `/`, `/vless`
- **CDN IP / Port** — a Cloudflare clean IP if you're fronting through CF (otherwise direct server IP, port 443)
- **TLS SNI** — what gets put in the ClientHello (must match the CDN cert when using CDN routing)
- **SNI Fragmentation** — TLS DPI-evasion: split / pad the ClientHello (try this if your ISP detects VLESS)
- **Header obfuscation** *(WS only)* — randomized browser-like headers

SOCKS5 (chain to a remote SOCKS5 proxy)

- **Remote SOCKS5 host / port**
- **Username / password** (optional, RFC 1929)
- **DNS server** for in-tunnel DNS-over-TCP (default `8.8.8.8`)

⚠️ Why SOCKS5 can show "Connected" but still not work

SlipNet's SOCKS5 mode is a **passthrough** — it just opens a local listener on your phone and forwards every downstream request to the remote SOCKS5 proxy. There is **no upfront end-to-end probe** of the upstream. The "Connected" status only confirms two things:

1. The local SOCKS5 listener bound on your phone, and

2. Some downstream connection completed the SOCKS5 handshake (CONNECT + auth) with the upstream.

It does **not** confirm that the upstream proxy can actually reach the open internet. The proxy may be silently dropping outbound traffic, returning forged CONNECT replies, sitting behind a cloud firewall that only allows internal CIDRs, or itself behind a censored uplink — and SlipNet has no way to tell from the SOCKS5 protocol alone, because SOCKS5 carries **no server identity, no end-to-end key exchange, and no liveness signal**.

SSH vs. SOCKS5

Bringing up an SSH tunnel requires every one of these steps to succeed before the instance reports itself connected:

- TCP connect to the SSH endpoint
- SSH version banner exchange
- Key exchange (ECDH/DH) — produces a shared session key
- User authentication (password or pubkey) — verified by the server
- A real channel opened over the encrypted session

If any step fails, SSH never goes green. So when SSH says "connected", an encrypted, authenticated channel really exists and is actually carrying bytes. That's why **SSH-based tunnels fail loudly** when something is wrong, while plain SOCKS5 can show a falsely green status.

	SOCKS5	SSH
Server identity check	None	Implicit via auth (and host-key, if pinned)
End-to-end key exchange	None	Yes (ECDH/DH)
"Connected" implies traffic flows	No	Yes
Encryption to the server	No (plaintext to the proxy)	Yes
Detects MITM / silent drops	No	Yes

> **Same symptom on DNS tunnels.** DNSTT, NoizDNS, and VayDNS in their **standalone form** (without the `+ SSH` wrapper) expose a local SOCKS5 listener as their downstream interface. The DNS tunnel itself *is* authenticated end-to-end (Noise / Curve25519 handshake, so the tunnel link is real), but what happens *after* the server-side terminator — the actual outbound to the open internet — is plain SOCKS5/forwarding with no end-to-end check. So a DNSTT/NoizDNS/VayDNS profile can also show **"Connected" while real traffic silently fails** if the server's outbound forwarder is broken or geo-restricted. The **`+ SSH` variants** (DNSTT+SSH, NoizDNS+SSH, VayDNS+SSH) fix this by running a real SSH session *through* the DNS tunnel — the SSH handshake only completes when the server can genuinely carry traffic to its destination, restoring the liveness guarantee.

Practical takeaway:

- Treat SOCKS5 mode as a **transport adapter** for a proxy you already trust — not as a tunnel test. If the proxy is broken or geo-restricted, you'll see green with zero traffic flowing.
- For a real anti-censorship tunnel, prefer **SSH** or one of the SSH-wrapped variants (NoizDNS+SSH, Slipstream+SSH, VLESS+SSH, NaiveProxy+SSH, …) — they fail loudly when something is wrong.

- If you must run plain SOCKS5, layer **SSH on top** when the upstream supports it — the `+ SSH` family treats the upstream as a transport and runs a real SSH session through it, restoring liveness and authentication guarantees.

DOH

- **DoH Server URL** — e.g., `https://cloudflare-dns.com/dns-query`
- *(DOH only encrypts DNS — your traffic still goes out normally.)*

Tor (Full edition)

- **Bridge type** — Snowflake / obfs4 / Meek / Direct / Custom bridge lines
- **Auto-detect Best Bridge** — lets the app pick

---

12. Troubleshooting

Symptom	Try this
Can't connect at all	Run the **Scanner**, replace resolvers; switch DNS Transport (UDP → DoT → DoH); try **NoizDNS** instead of DNSTT
Connects, then drops in seconds	Lower **VPN MTU** (1280), set **DNS Workers = 1** or *per-query*
Connected but no internet	Disable **QUIC** in settings; toggle **Auto-reconnect**
Phone kills SlipNet in background	Settings → **Battery optimization** → don't optimize SlipNet
YouTube / streaming buffers	Enable **Disable QUIC**; lower **Max Query Size** on DNS tunnels; try **Slipstream** or **VLESS**
Heavy DPI environment	Use **NoizDNS + SSH**, **VLESS + SNI Fragmentation**, or **Tor + Snowflake** (Full)
VLESS times out via Cloudflare	Set **CDN IP** to a known clean Cloudflare IP; verify **TLS SNI** matches your CF cert
SOCKS5 says "Connected" but no internet	Upstream proxy isn't actually reaching the open internet — SOCKS5 has no end-to-end check. Test the proxy independently, or wrap it with SSH (use a `+ SSH` variant) so failure becomes visible.
Need to share logs with support	Settings → **Debug logging** → reproduce the issue → **Export logs**

---

13. Sharing & backups

- **Share a single profile**: open it → menu → **Export** → choose `slipnet://` URI or JSON.
- **Export everything**: home menu → **Export All Profiles** (plain) or **Export All (Encrypted)** with a password.

- **Backup your settings**: Settings → menu → **Export Settings**.
- **Share the APK** with friends via Bluetooth from the home menu (useful during internet shutdowns).

Config URI types

SlipNet exports use three URI schemes. They are **not** equivalent — the security model is very different.

Scheme	What it is	Encryption key
`slipnet://`	Plain base64-encoded profile	None — anyone can decode it
`slipnet-enc://`	Single **locked** profile (single-recipient share)	App-baked key from SlipNet's native library
`slipnet-bundle-enc://`	Multi-profile **password-encrypted** backup bundle	Derived from your password

`slipnet://` — readable by anyone who can run base64. Use only for trusted recipients.

`slipnet-enc://` — produced when you export a single profile with a lock (password, optional expiration date, optional device-binding, hidden resolvers, no-resharing). The payload is AES-256-GCM encrypted, **but the AES key is baked into the SlipNet app itself**, not derived from your password. The password you set is checked by the recipient's app — it controls **how** the recipient can use the profile (use only, no editing, expires on date X, bound to one device, resolvers hidden). It does **not** keep your config secret from anyone with the SlipNet APK. Treat it as a **sharing-control** mechanism, not real confidentiality.

`slipnet-bundle-enc://` — produced by **Export All (Encrypted)**. This one *is* real password-derived encryption (your password is the key material). Suitable for off-device backups.

⚠️ Even encrypted configs are unsafe to post publicly

Encrypting the config bytes does **not** protect the **identity** of your server, domain, or CDN host. The moment a `slipnet-enc://` link reaches a public channel:

- Anyone with the SlipNet APK can decrypt it and read the domain, IP, SNI, CDN host, and credentials.
- Censors monitor public channels for new configs and add the domains/IPs to their blocklists within hours.
- The domain itself becomes a permanent blocklist target — encryption cannot undo that.

This is doubly serious if you bought your **VPS or domain inside the country whose censorship you're trying to bypass**:

- Local registrars and hosting providers can be **compelled to reveal ownership, suspend the service, or hand over logs**.
- Domains under a national TLD (e.g. `.ir`, `.cn`, `.ru`, or any TLD operated under the same jurisdiction) can be seized administratively, regardless of how strong your tunnel encryption is.
- A VPS or domain traceable to a person inside that country exposes the **operator**, not just the config.

Rules of thumb:

- For wider distribution, host on a **VPS bought outside the country** with a domain registered through an **off-shore registrar**. On that same VPS, **enable Warp from inside SlipGate** so the server's egress goes through Cloudflare Warp and the real VPS IP is never exposed downstream.

- Share configs **privately** (1-to-1, end-to-end-encrypted chat) — not in groups, not on websites, not pinned in channels.
- Use `slipnet-enc://` lock features (expiration, device-bind, hide-resolvers, no-resharing) to **slow down** abuse, not to make a config publicly shareable.

> ⚠️ A `slipnet://` link contains your credentials. Never post yours in a public group. ⚠️ A `slipnet-enc://` link is **not** safe to post publicly either — the SlipNet app can decrypt it. The lock controls usage, not secrecy. The encryption protects the *credentials* on the wire; it does **not** protect the *domain identity* once the link is in the open.

---

14. Donations

Development is unpaid. If SlipNet helps you, donation addresses (BEP-20 / ERC-20 / Arbitrum, Monero, etc.) are kept up to date in the project README on GitHub: https://github.com/anonvector/SlipNet#donations

---

15. Stay safe

- Only download SlipNet from the **official Telegram channel** or **GitHub**.
- Verify the APK signature if you're paranoid (signing fingerprint is on the GitHub release page).
- Never share your `slipnet://` link with strangers.
- Use **encrypted export** for backups.
- Verify your server is genuine with **Prism** (server-verified scan) — works with SlipGate.
- If you run your own server: never hand out `root` or any shell-login account as a VPN credential — always create dedicated SlipGate users.

Channel: @SlipNet_app Source: https://github.com/anonvector/SlipNet