

Network Disclosure — LocalTranscript

LocalTranscript · Documentation for IT and data protection

Status: version 0.6.65, September 2026

This document is aimed at IT security officers and data protection officers, and lists every network connection that LocalTranscript can technically trigger.

1. Core promise and architecture

Meeting audio, transcripts, summaries and voiceprints do not leave the device.

LocalTranscript records meetings, transcribes the audio into text, determines who spoke when, and on request runs AI actions (summaries, task lists, free-form prompts) against the transcript. In the default configuration, every one of these steps runs entirely locally: transcription (whisper.cpp), speaker recognition (pyannote, natively via ONNX Runtime), voice matching, and local AI summarisation (Gemma via llama.cpp) are processes on the workstation itself; the application works offline once the models have been downloaded. The **processing location** is therefore the respective workstation, and in the default configuration no **third-country transfer** takes place, because no meeting data leaves the device.

Meeting content has no upload path, unless a user explicitly configures a cloud provider (connection #5 below) with their own API key. This, too, can be centrally prohibited via **administrative policy** (`allowCloud: false`, see section 3 and our document "Administrative policy").

The following list shows in full which network traffic the application can trigger, what is transmitted in each case, and how it can be turned off.

2. How to verify this yourself (auditability)

- **Network audit log in the application:** Settings → System & Support → Diagnostics → Network Log shows every outgoing connection attempt (timestamp, host, purpose, allowed/blocked) and can be exported as a JSON-Lines file. The file is called `network-audit.jsonl` and is located in the app's local data folder (on Windows: `%LOCALAPPDATA%\<App-ID>\network-audit.jsonl`).
- **Hard Offline Mode** (Settings → Privacy) blocks every connection in the table below before any DNS resolution, except to loopback/LAN targets. Since version 0.6.65, every HTTP client is built on the same central factory function (`net_guard`), which re-checks the allowlist on every redirect, never routes loopback/LAN traffic through a `*_PROXY` environment variable, and disables proxies in offline mode. Before 0.6.65, redirects were followed without being re-checked, and a proxy variable could theoretically redirect a loopback request; meeting content was never sent as a result of this. The automatic update check is refused in the application core and logs every attempt; before 0.6.65 it was only locked at the surface level. Since 0.6.65, the same checkpoint also covers handing a link to the system browser, the reachability check in the first-run wizard, and the mobile companion's realtime channel (see the footnote below the table).

- **Administrative policy:** An administrator-owned, machine-wide `policy.json` enforces and locks Hard Offline Mode, prohibits cloud providers independent of offline mode, disables the automatic update check, turns off the mobile companion, MCP, feedback, voiceprints, Markdown export and dictation, allows additional internal hosts, and skips the first-run wizard. It applies when settings are loaded and saved and on every command, not only in the interface. Details in our document "Administrative policy".
- **Crash and performance diagnostics have no automatic destination.** The application contains no telemetry SDK. Crash and performance data stays in the local log file and leaves the device only if someone explicitly attaches the log excerpt via the feedback window (connection #7 below).
- **Or capture it yourself:** run Wireshark or a comparable tool against the app process. With offline mode enabled, no traffic is visible; without offline mode, only the optional update check and the downloads that you trigger yourself.

3. Connections

The numbering follows the internal reference list and is not reassigned, for traceability reasons. **Rows 3 and 4 are missing:** they concerned a licence API that was removed without replacement when the product switched to a permanently free local product, and no longer generates any network traffic.

#	Purpose	Target	When	What is sent	Turn off	Via administrative policy
1	Automatic update check + download	GitHub Releases (public release repository)	At app startup (after settings are loaded) or manually via "Check for updates"	Just an HTTP GET, no user data; usual HTTP metadata (IP, User-Agent) reaches GitHub; updates are signature-verified (Ed25519) before installation. Beta channel users additionally query the stable manifest once, to detect whichever is newer — same kind of request, no user data	Hard Offline Mode pauses it; brand-specific update flag	<code>autoUpdate: false</code> forces the stable channel and refuses checking and downloading entirely
2	AI model download	Hugging Face (primary) → GitHub release repositories (fallback mirror and runtime library, each hash-verified)	First-run wizard, model switch, or repair — always triggered by the user	HTTP GET of model/program files; no user data	Do not run downloads; offline mode blocks it	Subject to Hard Offline Mode (<code>offlineMode</code>); additional internal mirror hosts can be allowed via <code>allowedHosts</code>

5	Cloud transcription / cloud AI (optional providers)	OpenAI, Anthropic, or Infomaniak (Switzerland) — freely chosen	Only if a cloud provider is explicitly configured with its own API key; local is the default	Meeting audio (cloud transcription) or transcript text (cloud AI actions) to the chosen provider, under the user's own account	Do not configure a provider; "Local" is the default; Hard Offline Mode blocks it	<code>allowCloud: false</code> prevents configuration and use entirely — both online and offline
6	Calendar retrieval (iCal)	The calendar URL entered by the user themselves	Only if a calendar URL is configured	HTTP GET of the iCal feed	Do not configure it; offline mode blocks it (LAN addresses remain allowed)	No dedicated flag; generally subject to Hard Offline Mode as enforced via <code>offlineMode</code>
7	In-app feedback ("Send feedback")	Brand's feedback endpoint	Only if the window is opened and "Send" is clicked	The text entered, an optional reply email address, a random identifier generated per installation (since 0.6.65; no hardware characteristics), app version, any attached screenshots, and — only if the checkbox is enabled — the last 200 KB of the diagnostic log. Since 0.6.65 the log content has been cleaned up: no more transcript fragments, contact names, calendar data, or licence keys in it. Never audio or voiceprints	Do not send; brand-specific flag; Hard Offline Mode blocks it	<code>feedback: false</code> refuses submission at the command level
8	Mobile companion — LAN delivery (inbound)	A paired phone on the same local network; no server involved	Only if the companion is explicitly enabled in settings (default: OFF), after a consent screen that describes exactly this table	The desktop opens a port for inbound recordings and announces itself via local network discovery (mDNS). Everything is end-to-end encrypted (X25519 + XChaCha20-Poly1305) and every request is additionally authenticated (HMAC, key exchange via the pairing QR code). Nothing goes to the internet	Companion switch (default off); brand-specific flag; the Windows firewall can additionally block the inbound bind	<code>companion: false</code> prevents activation
9	Mobile	Brand's relay	Only if the	Exclusively end-to-end	Companion	<code>companion:</code>

	companion — encrypted relay (when no LAN path exists)	endpoint	companion is active, a brand relay address is configured, and no direct LAN path to the paired device exists	encrypted data blocks (recording up, transcript/summary down) plus small encrypted status messages, addressed via a meaningless routing identifier. The relay operator sees identifier, block size, timestamp, and IP address — never the content. Sole documented exception: if a paired device demonstrably cannot read the encrypted envelope (outdated software version), certain metadata (device label, timestamps, duration, size) is transmitted unencrypted to the relay operator — never the audio or transcript content itself	switch (default off); an empty brand relay address restricts new pairings to the LAN	false prevents activation of the desktop receiver (already stored pairing data on the phone is not revoked by this)
10	Mobile companion — push notification	Brand's relay endpoint → Apple Push Notification Service	Only if companion and relay are active and the desktop has just uploaded a finished transcript to the phone's relay mailbox	A meaningless routing identifier; the push message itself contains no meeting content and no meeting identifier — either a silent wake-up call or, with explicit consent on the phone, a generic text notification "Transcript ready"	Same switches as #9	companion: false (as for #8/#9)
11	Local MCP endpoint — Streamable HTTP (loopback, inbound)	Nothing leaves the machine: the service binds exclusively to 127.0.0.1, no access from the LAN is possible	Only while the MCP integration is enabled in settings (default: OFF) and the app is running	Meeting data via seven narrowly scoped tools (list/search/read meetings; change title, notes, tags) — exclusively via loopback to a process on the same machine	Switch disabled by default; close the app; brand-specific flag	mcp: false prevents activation — also for the separate helper process of the stdio variant
12	Mobile companion app — dictation	On iOS 16–25: Hugging Face. On iOS 26+: Apple's	Not reachable in the current	HTTP GET of model files; no user data, no audio, no text	Do not use dictation; the feature flag	Concerns the separate phone app, not the desktop

	speech model download (phone side)	own system services (no app-initiated connection)	App Store version — the feature is fully hidden. If unlocked in future: only on first use of the dictation feature, if the model is still missing		removes the entry point entirely	administrative policy
13	Mobile companion app — voice ID model download (phone side)	GitHub, a single approx. 26.5 MB model, one-time	Only after voice ID is enabled in the phone app and consent is confirmed	HTTP GET of a model file; nothing is uploaded — no audio, no embeddings, no contact data	Do not enable voice ID; the feature flag removes the feature entirely	Concerns the separate phone app, not the desktop administrative policy

4. What never has a network path

- For recording, transcription, speaker recognition, speaker/contact matching, voiceprint creation, semantic search, and local AI summarisation, there is no code path that uploads meeting content, unless someone has configured connection #5 themselves.
- The meeting database, audio recordings, and contact voiceprints reside in a local data folder of the app (Settings → Privacy → "Your Data" shows and opens this folder; "Delete All Data" removes it completely and verifiably).
- **Local MCP integration via stdio** (not to be confused with the HTTP variant in row 11): this transport path opens no TCP/UDP port, makes no HTTP requests, and has no LAN or internet mode. The switch, disabled by default, is re-checked on every tool call. This is a local process trust decision, not a network connection.
- A phone recording delivered via the mobile companion is never uploaded automatically. The optional automatic further processing sits behind an allowlist that refuses every cloud provider, unless the user has explicitly chosen it.

5. Local interfaces (inbound, LAN/loopback only)

These three services open a port on the device itself to accept **inbound** connections. None of them establishes an outbound connection to the internet; all are already recorded in section 3 (rows 8 and 11).

Service	Binding	Access protection	Note
Local MCP endpoint	127.0.0.1 — loopback	Since 0.6.65: a bearer token regenerated on every start, which every	Before 0.6.65 there was deliberately no authentication, meaning that on a

(HTTP)	only, no LAN	request must carry; the token is stored in a file with restricted access permissions, reissued on every start, and deleted on exit. A wrong or missing request authorisation is rejected	shared machine (e.g. a terminal server) every other logged-in user account could reach the port. Since then, the boundary is the operating-system user account
Local llama-server (Gemma inference)	127.0.0.1 — loopback only	Since 0.6.65: an access key regenerated on every start, which every app request must carry	Prevents another user account on the same machine from making use of the local AI inference
Mobile companion LAN listener	A port assigned by the operating system, reachable on the local network (not loopback only)	End-to-end encryption (X25519 + XChaCha20-Poly1305) and per-request HMAC authentication; key exchange exclusively via the pairing QR code	Disabled by default; activation requires an explicit consent screen

6. How these connections are routed (proxies)

Most connections from section 3 are established by the application's Rust core, not by the embedded webview. For these, the following applies: **the application does not use the Windows system proxy**. Only the environment variables `HTTPS_PROXY`, `HTTP_PROXY`, and `ALL_PROXY` are honoured; a proxy configured in Windows Internet Settings or via a PAC script is not consulted.

This has two consequences:

- **On a machine with proxy-mandatory internet access, these connections fail.** The browser reaches a destination, the app does not — this looks like "the app is broken" or "our firewall is blocking the provider," although the cause is that the app does not use the proxy.
- **A failed connection transmits no data.** Nothing is routed to a location that was not configured.

This is deliberate, not an oversight: if the app used the system proxy, the same mechanism would technically also route **loopback traffic** through the proxy, and thereby potentially forward the meeting transcript that is sent locally to `http://127.0.0.1:<port>` to a corporate proxy. Local-first must remain local even when the IT department mandates a proxy. The loopback clients are therefore explicitly exempted from every proxy.

Hard Offline Mode works independently of this: it refuses non-LAN targets before any DNS resolution takes place at all — so for a blocked request, no proxy is contacted at all.

7. Enforcement in the code (for technical reviewers)

This section is aimed at anyone who wants to inspect the source code themselves.

- **One central checkpoint.** Every outgoing HTTP connection in the Rust core runs through it, writes a log line, and, in Hard Offline Mode, refuses every non-LAN target before any DNS

resolution takes place.

- **One shared client factory.** Every HTTP client in the application is created through the same construction function, which re-checks the allowlist on every redirect hop and never lets a client marked "local only" leave the machine or the LAN, even during online operation.
- **A lint rule against new bypasses.** The build configuration prohibits a raw HTTP client anywhere else in the code, so that a new network call cannot accidentally bypass the checkpoint without breaking the build.
- **The administrative policy as an additional layer.** The administrative policy file is read when settings are loaded and saved, and is consulted by the checkpoint as well as by every protected command, so that a manually edited settings file cannot re-enable a feature prohibited by policy.

8. Change history (version 0.6.65)

- Hard Offline Mode enforced structurally rather than merely by convention: central checkpoint including re-checking on every redirect hop; no proxy use for local or offline connections.
- Automatic update check and download are now refused in Rust (previously only blocked at the surface level) and logged in every case.
- Administrative policy (`policy.json`) introduced: Hard Offline Mode can be enforced, cloud providers can be locked organisation-wide, further features (update, companion, MCP, feedback, voiceprints, Markdown export, dictation) can be centrally disabled.
- Authentication introduced for the local MCP HTTP endpoint (access token regenerated on every start); previously reachable unprotected on a shared machine.
- The local `llama-server` process now receives an access key regenerated on every start, so that other local user accounts cannot make use of the AI inference.
- Diagnostic log content cleaned up: no longer contains transcript fragments, contact names, calendar data, or licence keys.
- Model and runtime downloads are now cryptographically verified throughout (hash or signature check); a previously used, less robust version resolution was removed.
- Mobile companion behaviour in Hard Offline Mode tightened: no network beacon, no relay polling service; the realtime channel is now also checked and logged like every other connection.