Reading the Unreachable

Lessons from an Outbound-Only Edge Connector for Biometric Attendance Devices on Small-Office Networks

Sundar Shahi Thakuri
Independent researcher

November 2026
DOI: 10.5281/zenodo.24026722

Abstract

Cloud attendance platforms depend on punches recorded by biometric terminals that sit on small-office networks the cloud cannot reach. We describe the design, and the first production lessons, of a *connector*: a single static Go binary that runs on an existing, unmanaged office PC, reads three device families over pull protocols, relays two push protocols, and delivers every punch to the cloud over outbound HTTPS only. We make five contributions. (1) An architecture that decouples device reads from cloud delivery through an `fsync`'d spool with per-punch parking, and authenticates each PC with a locally generated Ed25519 key that signs requests against server-learned time. (2) Reachability techniques for the topologies we met: devices on off-range static addresses, devices one router away, and isolated Wi-Fi. They are built on evidence-gated secondary addresses and bounded discovery. (3) Time semantics for device wall clocks that survive wrong time zones and future-dated clocks. (4) A fail-closed handover protocol that migrates a live fleet from a legacy agent without losing or duplicating punches. (5) A production audit showing that long-lived delivery failures were dominated by onboarding and silent-agent states, not by protocol errors. We close with what remains open: code signing, safe self-update, and reach across sites.

1 Introduction

A biometric attendance terminal stores *punches*: who, when, and sometimes a direction. In a cloud attendance product those punches drive payroll, so a lost punch, a duplicated punch, or a punch on the wrong day is a wrong salary. The terminals sit on office networks behind one or two network address translators (NATs). Some models can be configured to call a cloud endpoint. In our deployment that path proved the least reliable one on small-office networks, and many models cannot use it at all. Something on the office network must therefore fetch the punches. The pragmatic host is a PC that is already there and that nobody on site administers.

That setting inverts the usual assumptions of fleet software. The host is not ours, and neither is the network. Antivirus suites intercept child processes that look like network configuration. Proxies and TLS-inspection appliances sit in the path. Devices are left on factory addresses outside the office's subnet, or are plugged into the ISP's router while the PC sits behind the office's own router. The internet disappears for hours or days, and attendance must not.

This paper reports on the second generation of an edge agent built for this setting. The first generation, a Python service installed by a PowerShell script, ran in production for several months and taught us the field rules we kept (§3). The second is a single statically linked Go binary of about 20,000 lines with 14,000 lines of tests. It runs on Windows (amd64 and 386), macOS (one universal app) and Linux (amd64 and arm64). The contributions are listed in the abstract. Throughout, we describe the system generically and report fleet observations only as proportions.

2 Setting

Devices. Table 1 summarises the five device interfaces the connector speaks. Three are pulled by the PC. Two are *push* interfaces, where the device sends to a configured host. For those the connector acts as a relay on the office network.

Topologies. Four office topologies account for the reachability work (Fig. 1). (a) A flat LAN, where PC and device share a subnet. (b) An *off-range* device left on a factory static address (for example `192.168.1.201`) while the office uses another private range. (c) A device *one router away*: the device is on the ISP router's LAN and the PC is behind the office router. The PC can reach the device by connecting outward through the office NAT, but the device cannot reach the PC, and the device's own cloud push often fails. (d) Wi-Fi with client isolation, where the PC can reach the router and nothing else.

Family	Transport	Dir.	Authentication / identity
ZKTeco	binary/TCP 4370	pull	optional numeric comm key; serial via protocol
ZKTeco	ADMS HTTP push	relay	serial in each request
Hikvision	ISAPI over HTTP	pull	HTTP Digest (MD5) or Basic; serial in device info
Dahua	CGI over HTTP	pull	HTTP Digest; serial via CGI
Sintech	UDP push (5055)	relay	none; matched by MAC

Table 1: Device interfaces. "Relay" means the device sends to the PC, which forwards to the cloud.

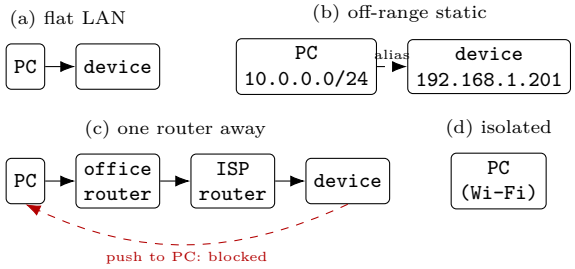


Figure 1: Office topologies that shaped the reachability design.

3 What the first generation taught

The Python agent ran at every connected site for months. Four of its field rules were carried over unchanged:

1. *The first read starts at the device's present.* A terminal can hold a year of punches. Historical import is an explicit, day-by-day operation.
2. *A future-dated device clock cannot stop attendance.* Punches dated more than 24 h ahead are ignored, and the reading position is never allowed past the present. A position already in the future is repaired on the next read.
3. *A backlog older than 20 h is labelled historical.* Otherwise the server's drift filter discards it. We lost a weekend of real punches to that filter before the rule existed.
4. *Failures are reported, not logged.* A silent device must not look like a device where nobody punched.

The first generation's dominant failure class was installation. That included PowerShell 5.1 and 7.4 parsing and exit-status differences, non-ASCII script bytes, services reported as `START_PENDING`, and install-time downloads on filtered networks. That class motivated a single static binary with no runtime.

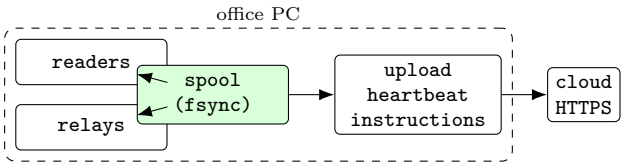


Figure 2: The spool is the only state shared by the device side and the cloud side.

4 Architecture

Figure 2 shows the components. The connector initiates every cloud connection. Inbound sockets exist only for the relays, on the office network, behind a firewall rule scoped to private networks.

Identity. The installer exchanges a six-digit code, issued in the cloud UI, together with the *public* half of an Ed25519 [1] key generated on the PC. The private key never leaves the PC. It is sealed with DPAPI in machine scope on Windows, and kept as a 0600 file in a root-owned directory elsewhere. The service refuses to start if its data directory is writable by anyone other than the system and administrators, because a writable directory could hand it a forged key. A newly paired PC receives no work until a human confirms it in the UI.

Request signing. Each request is signed over a five-line canonical string:

```
METHOD \n path \n timestamp \n nonce \n hex(sha256(body))
```

The server allows a ±300 s window and keeps a nonce cache that fails closed. Office PC clocks are frequently wrong, so the connector signs with the *server's* time. It learns the offset from every response and carries it forward on the monotonic clock. Signature tests run against the server's own golden file and against signatures produced by the server's cryptographic library, not against the connector's own implementation.

Loops. Work is organised as independent loops: read devices (60 s by default), flush the spool (15 s), poll instructions (30 s), heartbeat (60 s), refresh the device list (5 min), and a 30 min backstop read for push devices. When the cloud is unreachable all loops back off together, with jitter, and backlogs drain in bounded batches.

Readers. A reader implements `Info`, `Punches(since, until)`, `Users` and `Close`. Optional interfaces add self-identification and clock control. A device is read only after it proves its identity by its own serial, which is checked on every read rather than cached. Small offices reuse addresses, and a replaced device returns on the same IP with a different serial.

Spool and delivery. Punches are appended to a per-device file and `fsync`'d before any upload. They are

deleted only after the server acknowledges them. Two hazards were found by an adversarial review, and each was fixed with a regression test. First, a manual "read all now" racing the background drain re-sent a batch and marked unsent punches as sent; a reproduction lost 200 of 1,000 punches. Uploads are now serialised. Second, two connector processes could share one spool. The spool now holds an exclusive OS lock, which is released even on a crash. A punch the server refuses *individually* (for example, its device was detached) is *parked* rather than retried with its batch, so that one permanently invalid record cannot block every record behind it.

Diagnosis as a feature. Every failed cloud request is classified as `dns`, `connect_timeout`, `connect_refused`, `connect_reset`, `proxy_auth`, `tls_inspected`, `tls_other`, `http_blocked` or `server_error`. Device read failures are classified as `port_closed`, `timeout` or `no_route`, together with the port. The heartbeat also reports VPN capture of the LAN route, suspected Wi-Fi isolation, the proxy mode and the firewall-rule state. It never carries addresses or secrets. TLS inspection is detected and reported; certificate verification cannot be disabled.

5 Reachability

Evidence-gated secondary addresses. For topology (b), the connector adds a secondary address to the PC's interface inside the device's /24. It does so only when there is evidence that the device is on the same cable, either seen there or found by a search. It refuses ranges that overlap any network the PC is already on and uses only private ranges. It adds at most four aliases, never persistent ones, so a reboot or an uninstall leaves nothing behind. It removes stale aliases at start-up and records each alias in its data directory so that it can restore the ones still needed. On Windows the aliases are managed through the IP Helper API rather than `netsh` (§9). Where IT prefers a fixed block, the connector holds a configured "machine link" block instead. For Hikvision devices it can instead switch the device to DHCP over ISAPI and re-find it, which makes the alias unnecessary.

Bounded discovery. The search combines vendor discovery and port probes:

- Hikvision SADP on UDP 37020. Replies are judged against the PC's own networks, not its aliases. When multicast is filtered, the search falls back to rate-limited unicast inquiries.
- Dahua's discovery protocol.
- A TCP knock on port 4370 for ZKTeco devices, followed by a serial read.
- Further vendor ports, and an OUI fallback for devices that do not identify themselves.

The search covers the PC's own subnets, the /24s of devices the cloud already knows, user-supplied extra networks of any width from /22 to /32, and a bounded sweep one router away for topology (c). A device that stops answering is searched for by serial and MAC after consecutive failed reads, and its new address is reported to the cloud. A new network is noticed only on the adapter that holds the default route, so VPN and virtual adapters do not trigger searches.

Isolation and capture. When a search hears only the gateway, the heartbeat reports suspected isolation, and a definite flag when the PC is on Wi-Fi. When the route to a device leaves through a VPN adapter, the heartbeat reports `vpn_captures_lan`. Each condition maps to one plain-language fix shown in the UI and printed by a `doctor` command.

6 Time

A punch carries the device's wall-clock time. The connector therefore keeps each device's reading position (its *watermark*) in that device's own wall-clock time, not in the PC's time or in UTC. Three hazards shaped the rules:

- *Future clocks.* A device whose clock jumped forward once advanced the watermark into the future, and every later punch fell below it. The fix was to clamp the watermark to the present and to repair a poisoned watermark on the next read.
- *Wrong zones.* A device configured with the wrong time zone reports a UTC offset that does not match the office's local time. Its wall clock may be right (set by hand) or wrong (kept by a time server). The legacy agent converted every punch by the stated offset, which is right only in the second case. We chose wall-clock semantics, which are right in the first: read timestamps ignore the reported offset. Search windows and clock writes use the device's own reported offset, so that a write does not shift its clock by the difference. Because the connector also corrects a display that is more than 30 s off, the two cases converge after the first correction; only a backlog read before it remains exposed.
- *Clock drift.* On device families that expose clock control, the connector compares the device's clock with the server's time on every read. When the device is more than 30 s off, it reports the difference and corrects the clock, at most every 6 h, preserving the device's time mode. Punches already recorded are never rewritten. Our primary deployment zone has a non-whole-hour offset (+05:45), which surfaced every assumption that offsets are whole hours.

A subtle interaction appeared between the watermark and a per-process "record count" cache that skips unchanged ZKTeco devices. Rewinding a watermark did not clear the cache, so the device was never re-read. The cache entry is now dropped on every rewind.

State of an active device	Share
Delivered within the last 24 h	≈53%
Stopped within the last 1–3 days	≈8%
Down for weeks, or never set up	≈39%
configured to push, never contacted the cloud	15%
configured for PC reading, no PC reporting it	10%
legacy agent alive but silently not reading	6%
host PC offline for days	4%
device failing (credentials, unreachable)	4%

Table 2: One snapshot of active devices. Shares are of all active devices.

7 Handover from a live legacy agent

At the time of writing, every connected site still ran the legacy agent. The connector therefore had to take over without an operator. The protocol has three steps.

1. *Import, once.* The connector reads the legacy agent's configuration and imports host, port and credentials. It does so only for active devices owned by the same tenant, only for private addresses, and without overwriting credentials already set in the cloud. Devices wrongly recorded as "pushes by itself" are re-pathed to PC reading. This includes records stored as cloud connections that have been silent for days but have an office address.
2. *Retire verdict.* The server answers "ready" only when the legacy agent has reported recently, has no pending punches, and every device it reads that the connector will not read is demonstrably reaching the cloud on its own. Any ambiguity answers "not ready" and leaves the legacy agent running. The legacy agent is named by its own identifier, never by host name.
3. *Carry positions.* The verdict carries the legacy agent's watermarks. The connector adopts a carried watermark only for devices on its own list, only if it does not move the watermark forward, and never more than 7 days back. It then stops the legacy service through the Service Control Manager.

Legacy settings the new reader does not implement (forcing UDP, or a remapped status field) keep the legacy agent on, and the setting is named in the UI.

8 Field audit

We audited the production fleet, read-only, before enabling the connector anywhere. Table 2 classifies active devices by whether they had delivered punches recently. The fleet comprises several dozen sites.

Two observations stand out. First, the long-lived failures were dominated by *onboarding* states: devices recorded in the cloud that never delivered anything. Next came *silent agents*: processes that heartbeat normally but have not read for weeks and report no error. Protocol and credential errors were the smallest group. A connector, however robust, does not fix the first group. A setup flow that refuses to finish until the first punch arrives would fix it.

Second, comparing the new readers with the legacy agent, behaviour by behaviour, found divergences that no test would have shown. The two read a Hikvision whose zone setting is wrong in opposite ways (§6). The legacy agent fell back to HTTP Basic when a device offered no Digest challenge, and the new reader did not. Transient HTTP 401s caused by digest-nonce reuse had to be retried on a fresh connection before being reported as a wrong password. Some web-device records carried the ZKTeco port 4370, which had to be treated as unset. The audit also found two devices recorded as cloud push but silent for months, while the legacy agent read them every minute. Under the original handover rules the connector would never have been told to read them, so the handover would never have completed. Each divergence became a fix with a regression test.

9 Shipping on unmanaged Windows

Child processes. The first Windows builds invoked `netsh`, `arp`, `route`, `sc.exe` and PowerShell. Endpoint protection products intermittently held such processes, which turned a one-second call into an indefinite hang, and child processes also flashed console windows. The connector now uses the IP Helper API for addresses and routes, the Windows Firewall COM API for its rule, and the Service Control Manager API for services. Two guards keep it that way: a source-level test that fails if the Windows build calls `exec.Command` outside one console-hiding helper, and a test that fails if any test could launch a privileged system tool. The second guard exists because an early test did reach a real elevation prompt.

Distribution without a certificate. The Windows binary is unsigned. A one-line installer fetched by PowerShell (`irm ... | iex`) downloads the binary itself. Files written this way carry no Mark-of-the-Web, so SmartScreen does not intervene. The script verifies a release SHA-256 and deletes any mismatch. Windows 11 Smart App Control blocks unsigned binaries and offers no per-application exception. In that case the script explains the two alternatives: install on another PC in the office, or configure device push.

A hidden window. The installer re-launches itself elevated with `ShellExecute("runas", ..., SW_HIDE)`, so that no console flashes. Windows applies a process's start-up show state to its *first* `ShowWindow` call. As a result the WebView2 setup window stayed invisible while the process ran, and users saw nothing after accepting the elevation prompt. CI did not catch this because its

UI test already runs elevated and skips the relaunch. The fix shows the window a second time and brings it to the foreground. The regression test starts the test binary with a hidden show state and asserts `IsWindowVisible`. We confirmed the test fails without the fix on a hosted Windows runner.

10 Testing without the hardware

The suite needs no device and no network. Readers take injected transports. A reusable contract test checks every reader against the same expectations. A device simulator emulates the device families the team does not own, for end-to-end runs. The suite runs under the race detector, which found a data race between the runner's shutdown and its address-setup goroutine. It also runs on real Windows in CI, where it opens the setup window in WebView2 and keeps a screenshot. The team's rule for every fix is that it leaves exactly one guard behind, and that the guard is verified by reverting the fix and watching the guard fail. A guard that passes with and without its fix is not counted as a guard.

11 Related work

Endpoint fleet agents such as osquery [2] and IoT edge runtimes such as AWS IoT Greengrass and Azure IoT Edge assume a managed host or a dedicated gateway. Our host is neither. Secure software update has a mature design literature. The Update Framework [3] motivates our offline-signed manifests, and our decision to keep self-update disabled until rollback and downgrade protection are complete. Reaching hosts behind NATs is classically addressed by hole punching [4] and ICE [5], and by overlay networks built on WireGuard [6]. Those approaches connect *peers*. Attendance devices cannot run a peer, so the PC must stay the bridge, and our reachability work concerns the last hop on the LAN. Clock discipline follows NTP practice [7]. Here the device clocks are the authoritative timestamp source and can only be corrected coarsely. Desired-state reconciliation, as in cluster managers [8], is the model we intend for self-healing (§12).

12 Limitations and future work

A survey of how comparable systems solve each remaining gap gave the following agenda.

- *Retry policy against device lockouts.* Hikvision terminals lock an address after a handful of failed logins, typically five to seven. The nonce-recovery retry (three attempts on fresh connections) is correct for a transient 401. Repeated every minute against a wrong password, however, it produces about 90 failed logins per 30 minutes and keeps the device locked even after the password is corrected. A persistent 401 must become its own failure kind, back off to hourly, and retry immediately when the credential changes.
- *Human notification as self-healing.* Most long-lived failures in Table 2 need a person. We plan working-hours notices, one fix line per failure kind, with inhibition when the host PC itself is offline (the Alertmanager model), and a per-device expected-punch baseline to catch devices that read successfully but return almost nothing.
- *Liveness, not just restart.* Service managers restart a crashed process but not a hung one. Each loop will record its last completion, and the process will exit non-zero when a core loop stalls beyond ten intervals, so that SCM, systemd or launchd restarts it.
- *Safe self-update.* The update path stays disabled until it installs exactly the verified bytes, refuses a downgrade, and trust-checks its rollback. Health will mean "read a device" rather than "reported in". Rollout will be staged with a server kill switch, and the signed manifest will carry an expiry against freeze attacks [3].
- *Per-device time semantics.* Whether a device's display or its stated instant is right can be decided per device, from how far each sits from the server's time at read, instead of by one global rule.
- *Code signing.* Smart App Control blocks unsigned binaries, and an EV certificate no longer bypasses SmartScreen reputation. An OV certificate on a hardware token, with timestamped releases, is the planned route.
- *Reach without an overlay.* Comparing an outbound-only connector, an opt-in support tunnel, a WireGuard overlay [6], and a per-site appliance against the audit's failure classes, only the appliance addresses a failure we observe (host PC switched off). The overlay adds the largest attack surface and addresses none. We will keep the outbound design, add a time-limited support tunnel over the existing HTTPS channel, and pilot appliances only where the host PC is the measured cause.

13 Conclusion

An outbound-only connector on an unmanaged PC can deliver attendance reliably. That depends on three things. The device side and the cloud side must share nothing but a durable spool. Identity and time must come from the server, not from the PC. Every failure must be reported as a kind that someone on site can act on. Our audit suggests that the next gains lie less in protocols and more in onboarding, in detecting silence, and in shipping signed, self-updating software.

References

[1] S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, 2017.

[2] osquery: SQL powered operating system instrumentation and monitoring. `https://osquery.io`.

[3] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine. Survivable key compromise in software update systems. In *Proc. ACM CCS*, 2010.

[4] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *Proc. USENIX ATC*, 2005.

[5] A. Keranen, C. Holmberg, and J. Rosenberg. Interactive Connectivity Establishment (ICE). RFC 8445, 2018.

[6] J. A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *Proc. NDSS*, 2017.

[7] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4. RFC 5905, 2010.

[8] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, Omega, and Kubernetes. *ACM Queue*, 14(1), 2016.