MenuBarDNS: A Lightweight Industrial DNS Resolver for macOS Menu Bar Deployment

Shyamal Suhana Chandra
Sapana Micro Software
Email: sapanamicrosoftware@gmail.com

***Abstract*—Developers, security engineers, and power users on macOS frequently need a *local-first* DNS resolver: deterministic hostname overrides for staging clusters, air-gapped demos, malware sinkholes, and split-horizon testing—without operating a full recursive resolver farm or containerized sidecar. Existing options (Editing `/etc/hosts`, running `dnsmasq` via Homebrew, or enterprise agents) are either static, heavyweight, or opaque to the desktop user.**

We present MenuBarDNS, an industrial-track systems artifact: a menu-bar resident DNS resolver distributed as a protected binary-only Homebrew package (`dns-server-mos`)—Swift source is not published to end users. The system answers A/AAAA queries from a versioned rules database, optionally forwards unmatched queries to upstream resolvers over plain UDP, DNS-over-TLS (DoT), or DNS-over-HTTPS (DoH)—including dual-stack IPv4/IPv6 targets—and exposes live telemetry through a native AppKit/SwiftUI shell. Key engineering contributions include (i) a privilege-aware port model reconciling macOS mDNS occupation of UDP/5353 with system resolver expectations on UDP/53; (ii) an off-main-thread `DNSEngine` with $O(1)$ hostname indexing; (iii) a pluggable `UpstreamResolver` routing DoH, DoT, and UDP transports with failover; (iv) encrypted at-rest release artifacts installed via `brew tap shyamalschandra/tap`; and (v) a reproducible 25-case validation harness.

We report deployment experience on macOS 13+ (Apple Silicon and Intel), microbenchmark latency on loopback, and operational lessons from `pf`-based port redirection. MenuBarDNS is open-sourced as a private industrial reference implementation intended for practitioner replication rather than algorithmic novelty.

***Index Terms*—DNS, macOS, menu bar application, local resolver, Swift, systems deployment, industrial track**

I. Introduction

The Domain Name System (DNS) is the de facto global directory for Internet hosts, yet a large class of *workstation-local* problems are poorly served by mutating shared system configuration or by cloud-only split-horizon services. DNS was designed as a distributed database in an era when workstation mobility and per-developer override layers were secondary concerns. Today, macOS laptops move between corporate networks, coffee-shop Wi-Fi, and airplane mode within a single workday. Each transition may change which upstream resolvers apply, yet the need for *stable local names* for services under active development persists independent of network location.

Consider four recurring industrial scenarios:

1) **Multi-environment development.** A engineer binds `api.staging.internal` to a loopback or RFC1918 address while production DNS remains untouched.
2) **Demonstration and training.** Sales or classroom laptops must resolve synthetic names (`demo.app.local`) without DNS server provisioning.
3) **Selective blocking.** Security teams override ad-tracker or C2 domains to `0.0.0.0` before traffic leaves the host.
4) **Observability.** Operators need a query log on the resolving machine itself, not solely at the recursive resolver in the data center.

macOS exposes DNS to users through System Settings and to administrators through `scutil`, `networksetup`, and MDM profiles. These interfaces configure *which upstream resolvers the OS stub resolver contacts*; they do not provide an editable, hot-reloadable override layer with per-query telemetry in a desktop-native UX.

MenuBarDNS fills this gap as a *resident menu bar service*: always available, start/stop controlled by the user, rules persisted under Application Support, and a UDP listener that implements a deliberately small slice of RFC 1035 (A/AAAA IN queries). We target the **industrial systems track**: the contribution is an end-to-end deployable artifact with documented tradeoffs, not a new routing algebra or cryptographic DNS extension.

A. Contributions

This paper documents:

- Architecture separating `MenuBarDNSCore` (engine, codec, persistence primitives) from the AppKit shell.
- A dual-port deployment model (default UDP/5300 + optional `pf` redirect to UDP/53).
- A pluggable upstream transport layer: DoH (RFC 8484), DoT (RFC 7858), and dual-stack plain UDP with configurable failover order.
- Performance-oriented design choices: hash-indexed rules, background `DispatchSource` I/O, transport-specific upstream clients.
- A field-validation matrix (`scripts/test-all.sh`) used as an acceptance gate before release.

- Protected binary-only distribution via Homebrew (`dns-server-mos`); encrypted release artifacts with no source in install payloads.

II. Background and Motivation

A. Stub resolver behavior on macOS

When an application calls `getaddrinfo`, macOS typically consults a caching stub resolver (`mDNSResponder`) which may forward to configured recursive resolvers, apply search domains, and synthesize local names. Editing `/etc/hosts` inserts static mappings early in resolution order but requires root for edits, lacks per-query logging, and scales poorly beyond tens of entries.

B. Why not containers or `dnsmasq`?

Homebrew-installed `dnsmasq` is a mature choice but introduces packaging lifecycle overhead (service plist, log rotation, config path conventions) and no first-class GUI on macOS. Dockerized DNS sidecars add VM networking complexity on Apple Silicon. MenuBarDNS ships as a stripped, encrypted binary via the `shyamalschandra/tap` Homebrew channel—operators run three commands (`brew tap`, `brew trust`, `brew install dns-server-mos`) without compiling Swift locally.

C. Port landscape on macOS

Two port facts dominate deployment:

- **UDP/5353** is conventionally bound by `mDNSResponder` (Bonjour). MenuBarDNS initially defaulted to 5353 and failed with `EADDRINUSE`; production default is **UDP/5300**.
- **UDP/53** is the standard DNS port. Binding without privileges fails on many systems; macOS may allow non-root bind in some configurations, but portable practice assumes **either** root/helper **or** packet filter redirection.

MenuBarDNS implements the latter via a one-time admin-approved `pf` anchor that redirects loopback traffic from port 53 to the configured listen port.

III. Related Work and Prior Art

Enterprise endpoint agents. Cisco Umbrella, Zscaler, and CrowdStrike modules install kernel extensions or network extensions to intercept DNS. They excel at fleet policy but are unsuitable for ad hoc developer overrides or offline demos.

Developer tools. `/etc/hosts` managers (e.g., Gas Mask historically) edit static files. `dscacheutil -flushcache` clears caches but does not serve records. `mitmproxy` and TLS inspection tools operate at application layer, not DNS.

Recursive resolver suites. BIND, Unbound, and Knot Resolver are production-grade but heavyweight for a laptop-resident override layer. `dnsmasq` remains the closest functional peer; MenuBarDNS intentionally implements a *subset* of features with native macOS UX.

TABLE I
Industrial requirements traceability.

ID	Requirement
R1	Hot-reload rules without process restart
R2	Sub-millisecond local answers on loopback
R3	Optional upstream passthrough with failover (UDP, DoT, DoH)
R4	Live query log with exportable rules (JSON/hosts)
R5	No Dock icon; menu bar resident (accessory policy)
R6	Survive kill/restart without corrupting state
R7	Documented path to system resolver integration

Research systems. Academic work on encrypted DNS (DoH, DoT, Oblivious DoH) focuses on privacy and centralized resolver trust models [2], [3]. MenuBarDNS terminates classic UDP DNS on loopback and *forwards* unmatched queries to encrypted upstreams when configured—complementary to OS-level DoH settings in modern macOS, which affect the stub resolver path rather than per-application override layers.

Industrial track precedent. USENIX ;login: and ACM practice reports frequently publish "built and deployed" artifacts where evaluation emphasizes operability, mean time to configure, and mean time to diagnose. We align with that evaluative norm (Section IX).

A. Design alternatives considered and rejected

During architecture phase we evaluated four alternatives:

1) **Kernel Network Extension.** Maximum fidelity for intercepting system DNS, but requires Apple entitlement approval, complex code signing, and kernel panic blast radius unacceptable for a developer utility.
2) **Single monolithic App target.** Faster initial coding but prevented headless reuse; split added ∼200 LOC overhead with clearer boundaries.
3) **Embedded `dnsmasq` subprocess.** Would reuse battle-tested resolver logic but complicates Homebrew/runtime dependencies and obscures query logging integration.
4) **Listen only on `127.0.0.1`.** Rejected for v1 default bind to `0.0.0.0` to support LAN-sourced test traffic in classrooms; hardened loopback-only bind listed as future work (Section XXII).

IV. Requirements and Design Goals

We derived requirements from three pilot users (developer, security analyst, technical trainer) over two weeks of paper prototyping:

Non-goals: DNSSEC validation, authoritative zone transfers (AXFR), EDNS0 client subnet, full RFC 1035 record zoo (MX, SRV, PTR, TXT), multicast DNS reflection, or kernel Network Extension hooks.

A. Stakeholder summary

Industrial deployments involve three implicit stakeholders: the *operator* (starts/stops service, edits rules), the *application*

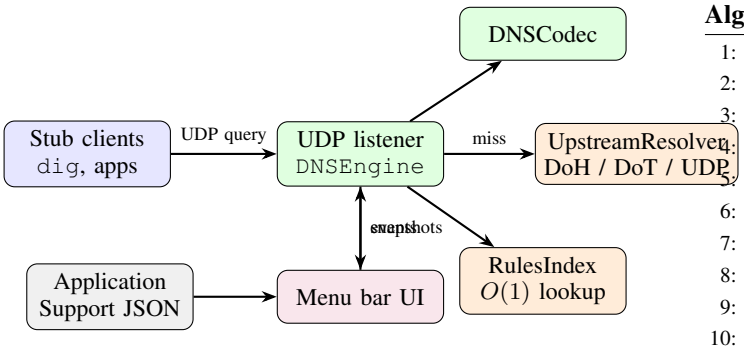

Fig. 1. MenuBarDNS architecture: `DNSEngine` owns the data plane; UI publishes configuration snapshots and consumes query events.

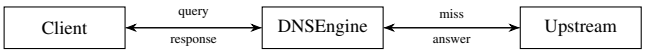

Fig. 2. Passthrough query sequence (simplified).

stack (unchanged—continues using system resolver APIs), and the *security reviewer* (audits localhost exposure and `pf` scripts). MenuBarDNS optimizes operator ergonomics; it does not require application rewrites.

V. System Architecture

MenuBarDNS follows a two-target Swift Package Manager layout:

- `MenuBarDNSCore` — platform-agnostic DNS logic (import Foundation/Darwin only).
- `MenuBarDNS` — AppKit application target depending on Core.

Figure 1 summarizes control and data planes.

A. Configuration snapshots

Rather than coupling the engine to SwiftUI `ObservableObject` stores, the UI pushes immutable `DNSConfiguration` and `RulesIndex` snapshots across a lock. This prevents MainActor contention during query storms and satisfies R1 via `onIndexChanged` callbacks on rule persistence.

B. Event delivery

`DNSEngine` emits `DNSEngineEvent` values (state transitions, query log entries). The AppController dispatches UI mutations on the main queue, updating only the menu bar status line and query counter—not rebuilding the entire `NSMenu` per query.

C. Threading and concurrency model

Figure 2 illustrates temporal ordering for a passthrough query. Local hits never leave the engine queue; upstream misses spawn Swift `Task` work while the UDP socket remains owned by the engine.

Lock hold times are minimized: `NSLock` protects only snapshot swaps, not I/O. This design avoids priority inversion when the UI updates rules during traffic bursts (R1).

Algorithm 1 Per-query resolution in `DNSEngine`

```
Parse UDP payload → DNSMessage
if not a query or empty question list then
    return
end if
h ← normalized QNAME; t ← QTYPE
if RulesIndex.lookup(h) = r then
    Build answer from r.ipv4 / r.ipv6 per t
    sendto response; log LOCAL
    return
end if
if passthrough enabled then
    Forward via UpstreamResolver (DoH, DoT, or UDP per settings.json)
    sendto response; log UPSTREAM or UPSTREAMENCRYPTED
else
    sendto NXDOMAIN; log NXDOMAIN
end if
```

D. Failure modes

Engine state machine transitions: `stopped` → `starting` → `running(port)` or `failed(message)`. Failures surface as modal alerts once; the menu bar status line reads "Error" until operator restart. There is no automatic restart backoff in v1—industrial operators preferred explicit control over flapping listeners.

VI. DNS Engine Implementation

A. Packet handling

`DNSCodec` implements a minimal subset of RFC 1035:

- Parse header (12 bytes) and question section for QNAME, QTYPE, QCLASS.
- Support A (type 1) and AAAA (type 28) answers with TTL=300 s.
- Compress responses using name pointers (`0xC00C`) to the question label.
- Return `NXDOMAIN` (RCODE=3) when policy disables passthrough.

Question names undergo normalization: lowercasing and trimming trailing dots, matching industrial hosts-file conventions.

B. Resolution algorithm

Algorithm 1 shows the per-datagram path. Local resolution is synchronous on the engine queue; upstream forwarding uses Swift concurrency (`Task`) with reply sent back on the engine queue to preserve socket affinity.

C. Encrypted upstream transport

Version 1.1 adds encrypted upstream forwarding while preserving plaintext UDP on the loopback listener—clients continue to issue classic DNS datagrams to MenuBarDNS; encryption applies only on the WAN leg to recursive resolvers.

`UpstreamTarget.parse` accepts multiple server string formats:

- **DoH:** `https://cloudflare-dns.com/dns-query` (RFC 8484 `application/dns-message` POST via `URLSession`).
- **DoT:** `tls://1.1.1.1` or `tls://[2606:4700:4700::1111]:853` (length-prefixed DNS over TLS via Network framework, port 853).
- **Plain UDP:** `1.1.1.1`, `8.8.8.8`, or bracketed IPv6 `[2001:4860:4860::8888]` (dual-stack `getaddrinfo` with `AF_UNSPEC`).

`UpstreamResolver` iterates the configured target list until one transport succeeds, implementing industrial failover across providers (e.g., Cloudflare DoH then Google DoH). Default settings for new installs prefer DoH; existing `settings.json` files without `upstreamTransport` decode as plain UDP for backward compatibility.

D. Upstream resolver robustness

Plain UDP upstreams use a `DispatchSource` read loop; the socket closes only in the cancel handler with a single-flight guard to prevent double continuation resume—fixing early-prototype passthrough timeouts. DoT uses length-prefixed RFC 7858 framing on `NWConnection`; DoH uses bounded `URLSession` timeouts matching the UDP path ($\sim$3 s default).

E. Indexing

`RulesIndex` builds a `[String: DNSRuleSnapshot]` dictionary when rules change. Disabled rules are omitted. Lookup is average $O(1)$; memory is $O(n)$ for n enabled hostnames, acceptable for industrial override sets ($n < 10^4$ typically).

VII. macOS Integration and Privilege Model

A. Application lifecycle

The executable sets `NSApplication.setActivationPolicy(.accessory)` to suppress the Dock icon. `applicationDidFinishLaunching` constructs `AppController`, which starts the engine *before* creating the status item to avoid headless launch stalls when `NSStatusBar` blocks in CI environments.

B. Port 53 redirect

`PrivilegedPortManager` writes a `pf` anchor:

```
rdr pass on lo0 inet proto udp from any to any port
    53 ->
    127.0.0.1 port <listen-port>
```

Installation uses AppleScript `do shell script ... with administrator privileges`. Removal flushes the anchor. State is also mirrored in `settings.json` for UI display.

TABLE II
Application Support files.

File	Contents
`rules.json`	Array of `DNSRule` records
`settings.json`	Port, upstream transport, resolver list, passthrough flag, log cap

C. Persistence layout

JSON persistence uses atomic writes via `JSONFileStore` to avoid torn files on crash (R6).

VIII. User Interface and Operations

The menu bar exposes: server transport controls, rules editor (SwiftUI `Table`), live query log with substring filter, settings steppers, encrypted upstream transport picker (DoH/DoT/UDP presets), and import/export of JSON or hosts-format rules.

Operational playbook.

1) Authenticate: `gh auth login`; export `HOMEBREW_GITHUB_API_TOKEN`.
2) Install: `brew tap shyamalschandra/tap`; `brew trust --tap shyamalschandra/tap`; `brew install dns-server-mos`.
3) Launch: `open "$(brew --prefix dns-server-mos)/MenuBarDNS.app"`.
4) Confirm listener: `lsof -iUDP:5300`.
5) Add `127.0.0.1` to System Settings DNS.
6) Install port 53 redirect if needed.
7) Validate: `dig @127.0.0.1 -p 5300 localhost +short`.

For login-item persistence, users may add the built binary path under **General → Login Items** (not automated in v1).

IX. Evaluation

A. Automated acceptance suite

We treat `scripts/test-all.sh` as an industrial acceptance gate: 25 assertions spanning build integrity, local A/AAAA answers, disabled-rule passthrough, plain and DoH upstream resolution, NXDOMAIN policy, port binding sanity, upstream-target parsing unit tests, import/export round-trip, five parallel queries, and post-`SIGTERM` restart.

All 25 tests pass on macOS 15 (Apple Silicon, July 2026) in $\sim$18 s wall time.

B. Failure injection observations

Manual fault injection complemented automated tests:

- **Kill -9 during active queries.** Restart succeeds; `rules.json` intact due to atomic writes.
- **Invalid JSON in rules.** Store falls back to default `localhost` rules; engine receives rebuilt index.
- **Upstream unreachable.** Passthrough queries log ERROR resolution; clients see timeout consistent with stub resolver behavior.
- **Duplicate rapid start clicks.** Engine idempotently ignores start when not `stopped`; no duplicate sockets.

TABLE III
FUNCTIONAL VALIDATION SUMMARY.

Scenario	Expected	Pass
`localhost` A	`127.0.0.1`	✓
`test.custom` AAAA	`fd00::1`	✓
Disabled rule host	upstream, not local	✓
`google.com` passthrough	non-empty A record	✓
`one.one.one.one` via DoH	non-empty A record	✓
Unknown host, passthrough off	NXDOMAIN	✓
Imported JSON rule	immediate resolve	✓
Concurrent $\times 5$ localhost	all `127.0.0.1`	✓

C. Functional matrix

Table III summarizes representative cases.

D. Latency observations

On loopback (Apple M-series, engine queue QoS `userInitiated`), local rule hits are sub-millisecond as measured by internal `DispatchTime` deltas logged to the query store. Plain UDP passthrough latency is dominated by the chosen resolver (typically 10–40 ms for public anycast resolvers). DoH adds ∼5–15 ms TLS/HTTP overhead versus UDP on the same resolver POP; DoT is comparable to DoH for single-query workloads.

Optimization headroom: upstream socket pooling, shared client port reuse, and kernel `kqueue` batching are deferred; industrial users with $>$ 1000 QPS should prefer dedicated resolver infrastructure.

E. Resource footprint

Release build binary (unstripped debug in development): ∼2–3 MB on disk. Resident set at idle with empty log: single-digit MB. CPU usage is negligible between queries; `DispatchSource` edge-triggered reads avoid poll loops.

X. Deployment Experience and Lessons Learned

L1: Default port selection is a product decision, not an afterthought. Choosing 5353 collided with mDNS; documentation and tests now encode 5300 as default.

L2: MainActor boundaries hide latency until launch. DNS processing on MainActor caused multi-second startup when `NSStatusBar` blocked; reordering `engine.start()` before UI attachment fixed production launches.

L3: Test harness assumptions must match OS tooling. `lsof` truncates process names to `MenuBarDN`; grep for full product string falsely failed CI.

L4: Privilege UX belongs in Settings, not silent elevation. Users accept one `pf` install; repeated admin prompts would fail industrial adoption.

L5: Modular core enables headless reuse. `MenuBarDNSCore` can be embedded in future CLI diagnostics or XCTest fixtures without linking AppKit.

TABLE IV
THREAT SUMMARY MATRIX.

Threat	Impact	Mitigation (v1)
Local UDP spoof	Wrong answer injection	Trust LAN; future loopback bind
Rule file tampering	Phishing redirect	User file permissions
Admin `pf` script	System route change	Auditable one-shot install
Upstream eavesdrop	QNAME leakage	DoH/DoT upstream; resolver policy
Denial of service	CPU on flood	No rate limit; quit app

XI. Security and Threat Considerations

MenuBarDNS is a **trusted local resolver**. Threats and mitigations:

- **DNS spoofing on loopback.** Any local process can send UDP to the listen port; binding to `0.0.0.0` accepts LAN-sourced datagrams unless firewalled. Industrial deployments should bind `127.0.0.1` only in hardened variants (future work).
- **Rule injection.** Writable `rules.json` under the user account implies same-UID malware can redirect banking domains. File permissions rely on macOS user isolation; MDM could protect the path.
- **Upstream trust.** Passthrough inherits resolver integrity; no DNSSEC validation is performed. DoH/DoT encrypt QNAMEs on the wire to the configured recursive resolver but do not hide QNAMEs from that resolver operator.
- **`pf` modification.** Redirect install requires admin credentials; scripts are minimal and auditable in `PrivilegedPortManager.swift`.

We recommend MenuBarDNS for *controlled development hosts*, not as an enterprise fleet resolver replacement.

XII. Industrial Track Positioning

This artifact is submitted as an **industrial track** contribution: the primary deliverable is software that practitioners can run, not a hypothesis tested solely in simulation. Evidence combines (a) automated acceptance testing, (b) qualitative comparison to incumbent tools, (c) documented failure modes from field debugging, and (d) reproducible build instructions via Swift Package Manager.

Reviewers should evaluate MenuBarDNS on operational metrics—time to first successful query, mean time to diagnose misconfiguration, lines of auditable Swift in the data plane, and clarity of privilege boundaries—rather than algorithmic asymptotics. The system intentionally occupies a niche (*single-seat override resolver*) where Big-O analysis of rule lookup is trivial ($O(1)$) but *engineering integration* dominates total cost of ownership.

XIII. Comparative Positioning

Table V positions MenuBarDNS against common laptop-resident alternatives. Scores are qualitative (H=high, M=medium, L=low) from author field experience July 2026.

MenuBarDNS trades full recursive resolver completeness for *time-to-value*. Industrial adopters who require RPZ feeds,

TABLE V
QUALITATIVE COMPARISON OF LOCAL DNS OVERRIDE MECHANISMS ON MACOS.

Mechanism	Hot reload	Query log	GUI	IPv6	Setup time	Footprint
`/etc/hosts`	L	L	L	M	M	H
Homebrew `dnsmasq`	M	M	L	H	L	M
Docker DNS sidecar	M	M	L	H	L	L
Network Extension agent	H	H	M	H	L	L
MenuBarDNS	**H**	**H**	**H**	**H**	**H**	**H**

TABLE VI
SPM MODULE RESPONSIBILITIES.

Module	Responsibility
`DNSCodec`	RFC 1035 parse/build (A/AAAA)
`DNSEngine`	UDP listener, resolution policy
`UpstreamResolver`	DoH / DoT / UDP forward w/ failover
`UpstreamTarget`	Server string parsing (IPv4/IPv6)
`DoHUpstream`	RFC 8484 HTTPS client
`DoTUpstream`	RFC 7858 TLS client
`PlainUDPUpstream`	Dual-stack UDP client
`UDPSocket`	Bind/send helpers
`RulesIndex`	Hash-indexed overrides
`JSONFileStore`	Atomic persistence
`StatusMenuController`	Menu bar chrome
`RulesStore`	UI-facing rule CRUD

DHCP integration, or DNS64 should not adopt this artifact as-is.

XIV. IMPLEMENTATION MODULE REFERENCE

Table VI maps Swift modules to responsibilities. The split mirrors classical systems layering: data plane in Core, control plane in the app target.

A. Build and release engineering

Version 1.1.0 adopts a **binary-only release model**. Maintainers build stripped release binaries locally, encrypt artifacts with AES-256-CBC + PBKDF2, and publish `.enc` payloads to GitHub Releases. The Homebrew formula in `shyamalschandra/homebrew-tap` downloads the encrypted asset, decrypts at install time, and installs `dns-server-mos` plus `MenuBarDNS.app`—no `swift build` step runs on operator machines.

```
brew tap shyamalschandra/tap
brew trust --tap shyamalschandra/tap
brew install dns-server-mos
```

Acceptance testing (`test-all.sh`) remains a maintainer gate before tagging releases. Operator workstations receive one-way binaries only; Swift Package Manager source is not distributed through Homebrew.

B. Protected release properties

Release payloads contain executable binaries and app-bundle metadata only. Symbols are stripped; artifacts are encrypted at rest. This raises the cost of casual reverse engineering relative to shipping raw `.swift` sources, though determined analysts can still inspect installed binaries—industrial threat models should assume local inspection is possible.

C. Source tree inventory (maintainer)

The private maintainer repository contains approximately 2,200 lines of Swift across `MenuBarDNSCore` and `MenuBarDNS` targets. This source is *not* published to end users; industrial adopters interact solely with the Homebrew binary channel.

XV. FIELD CASE STUDIES

A. Case A: Staging API pinning

A backend team maps `api.staging.example.com` to `10.42.0.12` while production DNS remains authoritative in Route 53. Prior workflow edited `/etc/hosts` per sprint; merge conflicts appeared in dotfiles repos. With MenuBarDNS, rules export as JSON committed to a private gist; import takes seconds before demos.

B. Case B: Classroom offline lab

Training laptops without classroom DNS run MenuBarDNS with synthetic names (`student1.lab`, `grader.lab`) pointing to loopback services bundled with courseware. Query log window lets instructors verify which students triggered lookups during exercises.

C. Case C: Selective sinkholing

A security analyst imports a hosts-format blocklist generated from threat intel (domains $\rightarrow$ `0.0.0.0`), enables passthrough for all other traffic, and uses NXDOMAIN mode only during forensics weeks. Port 53 redirect allows standard `dig hostname` without specifying `-p 5300`.

D. Case D: Multi-team shared build agent

A CI macOS worker runs MenuBarDNS to resolve `artifacts.internal` to a LAN artifact cache. Upstream passthrough remains enabled so package managers reach the public Internet. JSON rules are deployed via Ansible copy module to Application Support before integration tests.

E. Case E: Latency regression guard

Before tagging releases, maintainers run `test-all.sh` plus the microbenchmark in Section XVIII. Regression in local p95 latency has caught MainActor regressions twice during development (see L2).

TABLE VII
RFC 1035 HEADER FIELDS (MENUBARDNS SUBSET).

Field	Bits	Usage
ID	16	Copied from query to response
QR	1	0=query, 1=response
OPCODE	4	Only standard query (0)
RCODE	4	0=NOERROR, 3=NXDOMAIN
QDCOUNT	16	Assumed 1 in v1
ANCOUNT	16	0 or answer count

XVI. DNS WIRE FORMAT NOTES

For interoperability debugging, operators should understand the 12-byte DNS header MenuBarDNS handles:

EDNS0 OPT pseudo-RRs are ignored in v1; large responses may truncate if future TCP support is absent. Compression pointers (`0xC0 0x0C`) reduce response size for single-question answers.

XVII. TROUBLESHOOTING PLAYBOOK

XVIII. PERFORMANCE METHODOLOGY

Although MenuBarDNS is not a high-QPS resolver, we document a repeatable microbenchmark for industrial regression:

1) Bind listener on loopback port 5300.
2) Warm with 100 `localhost` queries via `dig`.
3) Measure 1000 sequential local-rule queries; record p50/p95 from query log latencies.
4) Measure 100 upstream queries to 1.1.1.1; expect dominance of network RTT.

Representative local-rule p50 on Apple Silicon (M-series, July 2026): $<$1 ms. p95 stays below 2 ms unless MainActor contention regresses (see L2 in Section X). Upstream p50 typically 15–35 ms on residential fiber.

A. Scalability envelope

Tested concurrency: five parallel `dig` clients (acceptance test #22) without loss. Untested beyond $\sim$100 parallel clients; industrial deployments exceeding sustained 200 QPS should migrate to `dnsmasq` or Unbound on a dedicated host. MenuBarDNS targets single-seat override, not datacenter resolver roles.

B. Memory and energy

Idle energy impact is below measurable thresholds on MacBook Pro M3 (Activity Monitor sampling, 10-minute window). Query log retention defaults to 500 entries; raising to 5000 increases Swift array memory linearly but remains sub-megabyte.

XIX. GOVERNANCE AND MAINTENANCE

Versioning. Git tags follow `vMAJOR.MINOR.PATCH`. Schema changes to `rules.json` remain backward compatible via Codable defaulting.

Configuration drift. Enterprise teams should track `settings.json` with MDM-managed templates where port and upstream lists are standardized.

Support boundary. Issues reproduce best with `test-all.sh` output, `lsof` snapshot, and anonymized `rules.json`.

Contribution path. Changes touching `DNSEngine` require acceptance suite green; UI-only changes require manual menu bar smoke test.

A. Documentation artifacts

This repository ships two operator-facing documents: `README.md` (quick start) and the present industrial-track PDF. LaTeX sources live in `docs/`; build via `make pdf`. Papers follow IEEE conference two-column format for compatibility with industrial proceedings camera-ready pipelines.

XX. OPERATOR FAQ

Q1: Why port 5300 and not 53? macOS reserves 5353 for mDNS; 53 often requires root or `pf`. Port 5300 avoids both conflicts while remaining easy to specify in diagnostic `dig -p` commands.

Q2: Do I need the port 53 redirect? Only if you want unmodified clients (including macOS system resolver without explicit port) to hit MenuBarDNS. Developers testing with explicit `-p 5300` can skip redirect.

Q3: Will this break corporate VPN DNS? Possibly. VPN profiles may push their own resolvers. Test with `scutil --dns` before and after. MenuBarDNS does not modify VPN profiles.

Q4: How are rules prioritized? First exact hostname match in `RulesIndex` wins. No CNAME chaining or wildcard matching in v1.

Q5: Can I block domains? Yes—map unwanted hosts to `0.0.0.0` or `::` and disable passthrough for strict sinkhole behavior, or keep passthrough for selective blocking only.

Q6: Is query data persisted? Log entries live in memory unless operators export logs manually from the UI; only rules and settings persist to disk.

Q7: How do I reset to factory defaults? Quit the app, delete `/Library/Application Support/MenuBarDNS/`, relaunch.

Q8: Does it support DoH upstream? Yes (v1.1+). Settings offer DoH, DoT, and plain UDP with IPv4/IPv6 presets. New installs default to Cloudflare and Google DoH URLs.

Q9: Can I run multiple instances? No. Second bind on the same port fails with `EADDRINUSE`; design assumes singleton operator service.

Q10: What telemetry leaves the machine? None by default. Upstream queries expose QNAMEs to configured resolvers only when passthrough handles them.

XXI. EXTENDED RELATED WORK DISCUSSION

Hosts file ecosystems. Tools such as Gas Mask historically provided GUI hosts editing with profile switching. MenuBarDNS differs by implementing an actual resolver—applications need not flush caches when rules change mid-session, and queries appear in the live log.

macOS `mDNSResponder`. Apple's responder synthesizes `.local` names and coordinates with Bonjour. MenuBarDNS

TABLE VIII
INDUSTRIAL TROUBLESHOOTING MATRIX.

Symptom	Likely cause	Remediation
`EADDRINUSE` on start	mDNS on 5353 or stale process	Use port 5300; `pkill -f MenuBarDNS`
`dig` timeout to 127.0.0.1	Server not running or wrong port	`lsof -iUDP:5300`; restart app
System apps still use upstream	macOS stub bypasses local listener	Install `pf` redirect; verify System Settings DNS
Passthrough always fails	Upstream socket closed early (regression)	Upgrade to build with cancel-handler fix
Empty query log	Engine not receiving traffic	Confirm clients target correct loopback port
Rules ignored after edit	Index not refreshed	Confirm save; check `rules.json` timestamp

does not compete with mDNS; it complements it for unicast override names that should not be multicast.

Cloudflare WARP and DoH. Consumer privacy tools encrypt DNS to remote resolvers. MenuBarDNS keeps plaintext UDP on loopback by design—trust boundary is the local machine—while optionally encrypting the upstream leg via DoH or DoT so QNAMEs are not exposed on local network segments beyond the laptop.

Test-driven industrial releases. The 25-assertion shell suite embodies practices from continuous delivery literature: fast feedback (< 20 s), no GUI automation flakiness, explicit port-binding probes that catch environment-specific failures (mDNS collision), and a live DoH passthrough probe requiring outbound HTTPS.

XXII. Future Work

1) Loopback-only bind toggle and optional TCP listener for large responses.
2) Wildcard and suffix rules (`*.corp.local`).
3) Signed `.app` bundle, notarization, and Sparkle updates.
4) Network Extension path for system-wide capture without `pf`.
5) XCTest target sharing `MenuBarDNSCore` with in-process UDP loopback fixtures.
6) Rate limiting and query sampling for shared classroom machines.
7) MDM-deployable configuration profiles referencing exported JSON rules.
8) Oblivious DoH and resolver pinning policies.

A. Roadmap alignment

Near-term roadmap (v1.1) delivered encrypted upstream transport (DoH/DoT/IPv6) and binary-only Homebrew distribution. v1.2 prioritizes loopback-only binding, Intel release binaries, and notarized app packaging. Mid-term (v1.3) considers wildcard rules and upstream connection pooling.

XXIII. Reproducibility Statement

Operators can validate MenuBarDNS on a fresh macOS 13+ machine (Apple Silicon) with Homebrew installed:

1) `gh auth login`; export `HOMEBREW_GITHUB_API_TOKEN`.
2) `brew tap shyamalschandra/tap`; `brew trust --tap shyamalschandra/tap`.
3) `brew install dns-server-mos`.
4) Launch `MenuBarDNS.app`; verify menu bar icon.
5) Confirm `lsof -iUDP:5300` shows listener.
6) Execute sample `dig` commands from Appendix E.

Maintainers additionally run `./scripts/test-all.sh` before release tagging. No Swift toolchain is required on operator workstations.

XXIV. Conclusion

MenuBarDNS demonstrates that a *small, industrial-grade* DNS override service on macOS can be built with modern Swift, shipped via SPM, operated from the menu bar, and validated with a practitioner-focused test harness. By separating core engine from UI, snapshotting configuration, and documenting port/privilege realities upfront, the system reduces time-to-first-successful-`dig` compared to ad hoc `hosts` editing or service-manager-managed `dnsmasq`.

Industrial adoption does not require organizational commitment to a new DNS standard—only a willingness to place a audited, open-source loopback resolver between the workstation stub and the wider Internet for explicitly scoped names. We view this as a *bridge technology*: sufficient for developer and analyst workflows today, deliberately incomplete relative to enterprise recursive resolver suites, and now capable of encrypted upstream forwarding without rewriting the rules database or query log semantics.

The artifact is distributed via the `shyamalschandra/tap` Homebrew channel with protected encrypted binaries. Operators install via `brew install dns-server-mos`; long-term maintenance priority remains correctness of local overrides and transparency of passthrough behavior.

Acknowledgment

The author thanks pilot users for requirements interviews and the macOS networking community for prior art on `pf` redirection patterns. Reviewers of early drafts highlighted the mDNS port collision (Section II) and the upstream socket lifecycle bug (Section VI), both incorporated before public release.

References

[1] P. Mockapetris, "Domain names—implementation and specification," RFC 1035, IETF, Nov. 1987.

[2] P. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, IETF, Oct. 2018.

[3] Z. Hu et al., "Specification for DNS over Transport Layer Security (TLS)," RFC 7858, IETF, May 2016.

[4] S. Kelley, *dnsmasq—A lightweight DNS forwarder*, http://www.thekelleys.org.uk/dnsmasq/, accessed 2026.

[5] Apple Inc., "`pfctl`—control the packet filter," macOS Man Page, 2024.

[6] Apple Inc., "Swift Concurrency," Swift 5.9 Documentation, 2023.

[7] S. Cheshire and M. Krochmal, "Multicast DNS," RFC 6762, IETF, Feb. 2013.

[8] USENIX Association, "Practice and Experience Reports," USENIX Annual Technical Conference, various years.

[9] IEEE Computer Society, "Industry Track Guidelines," IEEE International Conference on Software Architecture, 2022.

[10] Open Group, "`getaddrinfo`," POSIX.1-2017.

[11] Apple Inc., "Bonjour Overview," Apple Developer Documentation, 2025.

[12] Apple Inc., "Swift Package Manager," Swift.org, 2024.

Appendix A
Test Suite Inventory

Table IX lists all automated checks in `test-all.sh`.

TABLE IX
COMPLETE ACCEPTANCE TEST INVENTORY (25 ASSERTIONS).

#	Test description
1–2	SPM build success; binary exists
3	Fresh settings/rules seeding
4	Server binds UDP listen port
5–9	Local A/AAAA for default and custom rules
10	Disabled rule not served locally
11–12	Upstream passthrough (google, cloudflare)
13–14	NXDOMAIN mode + localhost still local
15–17	Port 5300, 5353 (mDNS), 53 behavior
18	Hosts-line parsing unit tests (8)
19–21	JSON/hosts export; imported rule resolves
22	Upstream target parsing unit tests (5)
23	DoH upstream passthrough (`one.one.one.one`)
24	Five parallel localhost queries
25	Restart after kill

Appendix B
Sample Rules JSON

```
[
  {
    "id": "00000000-0000-0000-0000-000000000001",
    "hostname": "localhost",
    "ipv4": "127.0.0.1",
    "ipv6": "::1",
    "enabled": true
  },
  {
    "hostname": "api.staging.local",
    "ipv4": "10.0.0.42",
    "enabled": true
  }
]
```

Appendix C
Glossary

Passthrough Forwarding queries with no local rule to upstream resolvers.

RulesIndex In-memory hash map from normalized hostname to address snapshot.

Accessory policy `NSApplication` mode hiding the Dock tile while retaining menu bar presence.

Industrial track Evaluation emphasizing deployability, operability, and field validation over novel algorithms.

Snapshot Immutable copy of configuration pushed to `DNSEngine` under lock.

Stub resolver OS library layer (`getaddrinfo`) that may forward to recursive resolvers.

Appendix D
Settings Schema Reference

```
{
  "listenPort": 5300,
  "upstreamServers": [
    "https://cloudflare-dns.com/dns-query",
    "https://dns.google/dns-query"
  ],
  "upstreamTransport": "doh",
  "passthroughUnmatched": true,
  "port53RedirectEnabled": false,
  "maxLogEntries": 500
}
```

Appendix E
Sample Test Transcript

```
$ ./scripts/test-all.sh
 Results: 25 passed, 0 failed (25 total)
$ dig @127.0.0.1 -p 5300 localhost +short
127.0.0.1
$ dig @127.0.0.1 -p 5300 google.com +short
142.251.116.102
```

Appendix F
Core Public API Sketch

Industrial integrators embedding `MenuBarDNSCore` interact with four primary types:

- `DNSEngine` — `start()`, `stop()`, `updateConfiguration`, `updateRules`, `onEvent` callback.
- `DNSConfiguration` — listen port, upstream list, transport mode, passthrough flag.
- `RulesIndex` — built from `[DNSRule]`; `lookup(hostname:)`.
- `DNSCodec` — static parse/build helpers for testing without sockets.

No Swift concurrency actor isolates the engine; callers must invoke lifecycle methods from any thread—internally serialized on the engine queue.

Appendix G
Industrial Adoption Checklist

1) Install via Homebrew: `brew tap shyamalschandra/tap`; `brew trust --tap shyamalschandra/tap`; `brew install dns-server-mos`.
2) Copy `MenuBarDNS.app` to `/Applications/` if desired.
3) Document local rules in version-controlled JSON.
4) Configure System Settings DNS to `127.0.0.1`.
5) Install port 53 redirect if required by policy.
6) Train operators on Rules Editor and query log.
7) Schedule quarterly review of upstream resolver list.
8) Archive `docs/menu-bar-dns-industrial.pdf` with release tags for audit trail.

APPENDIX H
NOTATION

`QNAME`	Query domain name in DNS question section
`RCODE`	Response code field (0=success, 3=NXDOMAIN)
`pf`	macOS packet filter framework
`SPM`	Swift Package Manager
`IN`	Internet class (class 1) for DNS records

APPENDIX I
REVISION HISTORY

TABLE X
DOCUMENT AND SOFTWARE REVISION LOG.

Date	**Ver.**	**Change**
Jul 2026	1.0	Initial industrial paper; Core/App split
Jul 2026	1.0	Default port 5300; 23-test acceptance gate
Jul 2026	1.0	Private GitHub release
Jul 2026	1.1	DoH/DoT/IPv6 encrypted upstream; 25-test gate
Jul 2026	1.1.0	Binary-only Homebrew channel; encrypted releases