MULTI-SOURCE REFERENCE

THE mrefd (M17) REFLECTOR REFERENCE

A written, sourced guide to the mrefd M17 reflector — what it is, where it came from, how it works from the packet upward, how you connect to it, and how you build, configure, operate, secure, and update one.

Compiled 8 August 2026 · sourced from primary documentation and live-operator practice

Edition 1.0 — companion to *The URF Reflector Reference* and *The XLX Reflector Reference*

Preface

Anyone who’s spent time with mrefd (M17) reflectors knows the information is scattered — a piece in the source code, a piece on a forum, a piece on someone’s dashboard, and some of it out of date. This document pulls it together.

It’s compiled from numerous searches using many different sources — the reflector’s source code, the project documentation, and the pages of operators running live systems — and assembled into one reference, so the answer’s in a single place the first time you look. It’s built to save you the hunt: get oriented, find the exact command, port, or setting you need, and see where sources agree and where they conflict.

It’s a starting point, not the last word — a springboard for your own research; since reflectors and software change, confirm anything critical against your own running system.

Free to use, copy, and share, and searchable — jump straight to what you need.

Contents

A written, sourced guide to the mrefd M17 reflector — what it is, where it came from, how it works from the packet upward, how you connect to it, and how you build, configure, operate, secure, and update one.

Compiled 8 August 2026. This guide is built from primary sources — chiefly the `mrefd` source code and documentation read directly, and the M17 Project's protocol material — and corroborated by independent operators running live M17 reflectors. It is the third in a set, a companion to *The URF Reflector Reference* and *The XLX Reflector Reference*. Because mrefd carries only M17, it is far simpler than those two — there is no transcoder and no vocoder hardware anywhere in this guide — so the pages that its siblings spend on transcoding are spent here on the parts of M17 that are genuinely distinctive. Where the sources disagree, the disagreement is described rather than smoothed over; where something could not be established from a reliable source, that is said plainly rather than guessed. Figures that drift with time — prices and default ports — are marked *"as of"* where they appear. mrefd is a young, fast-moving project, so config-file names and features have shifted between versions; where this guide gives a specific value it reflects the current source, and Chapter 24 records the known naming differences. The legal chapter concerns United States law and is a factual summary, not legal advice.

Quickstart — the whole guide in brief

If you read nothing else, this is mrefd in a nutshell. An **mrefd reflector** is a server running the open-source `mrefd` software that joins **M17** radios and hotspots into shared rooms. It is single-mode by design: it speaks only M17, which uses the open, patent-free **Codec2** voice codec — so every connected client speaks the same codec and there is *nothing to transcode*. That one fact removes the entire apparatus its siblings need: no transcoder daemon, no DVSI dongle, no vocoder hardware, and no per-stream conversion cost. It organizes traffic into up to twenty-six independent **channels** lettered A–Z, handles both M17 **Stream mode** (voice) and **Packet mode** (short data), relays **encrypted** streams on channels configured to permit them, and is discovered peer-to-peer over **Ham-DHT**.

To **stand one up**: on a Debian or Ubuntu server with a public address, install the dependencies, clone `mrefd`, set the compile-time options in `mrefd.mk` and the runtime settings in `mrefd.cfg`, build and install with `make && sudo make install`, and copy a dashboard into your web server. Because there is no hardware, a micro-VPS or a Raspberry Pi is ample. To **use one without a radio**, connect with N7TAE's **mvoice** — an M17-only desktop client that runs Codec2 in software and needs no dongle at all — or with DroidStar; and you can echo-test your own audio at any time by keying the built-in **#PARROT**.

mrefd is the lean, purpose-built M17 reflector, written by the same author as URF and sharing its code ancestry; it gets new M17 features first and delivers M17 at its best. Its sibling **URF** also carries M17, but among many modes and with a transcoder — so this guide is candid throughout about the one decision that matters most: run mrefd for a dedicated M17 room, and reach for URF when M17 needs to share a room with D-STAR, DMR,

Fusion, or P25. Everything after this — the mode itself down to the packet, encryption and the law, interlinking, discovery, operations, the trade-offs, and the costs — is that same story told in full.

Chapter 1 — What an mrefd Reflector Actually Is

To understand an mrefd reflector you first have to understand a reflector. In amateur digital voice, a reflector is a server on the internet whose job is to take whatever one connected station transmits and repeat it out to every other station connected to the same place. It is the digital-voice equivalent of a conference bridge on a telephone system: a dozen operators around the world can all link their radios or hotspots to one reflector, and when any of them keys up, the rest hear it. The reflector holds no transmitter of its own; it moves audio between people over IP.

An **mrefd reflector** is a particular, focused kind of reflector: a *single-mode* one. The software that runs it, `mrefd` — the M17 Reflector Daemon — speaks only **M17**, the open, community-developed digital-voice protocol built on the patent-free Codec2 codec. It is written by Thomas A. Early, N7TAE (copyright 2020–2025), and, in the project's own words, "most of the code is originally based on groundbreaking development of XLXD." That heritage matters: mrefd is a sibling of `urfd`, sharing the same code ancestry and the same twenty-six-channel way of thinking, but where URF is a universal, many-mode reflector with a transcoder, mrefd is deliberately just the M17 part, done well.

That single-mode focus is the whole character of the software, and it is worth stating plainly up front because it removes most of what makes the other reflectors complicated. Every client on an mrefd reflector speaks M17, and M17 speaks Codec2, so there is never any codec conversion to do. There is no transcoder program, no DVSI hardware, no AMBE or AMBE+2, and none of the operational failure modes that come with them. What remains is a lean, fast reflector whose interesting parts are all M17-specific: its two sub-modes, its encryption model, its version-aware interlinking, and its peer-to-peer discovery. This guide is candid about mrefd's place in the ecosystem throughout: it is the best tool for a dedicated M17 room, and where M17 needs to reach other modes, its sibling URF — which can transcode — is the better choice (Chapters 16 and 17).

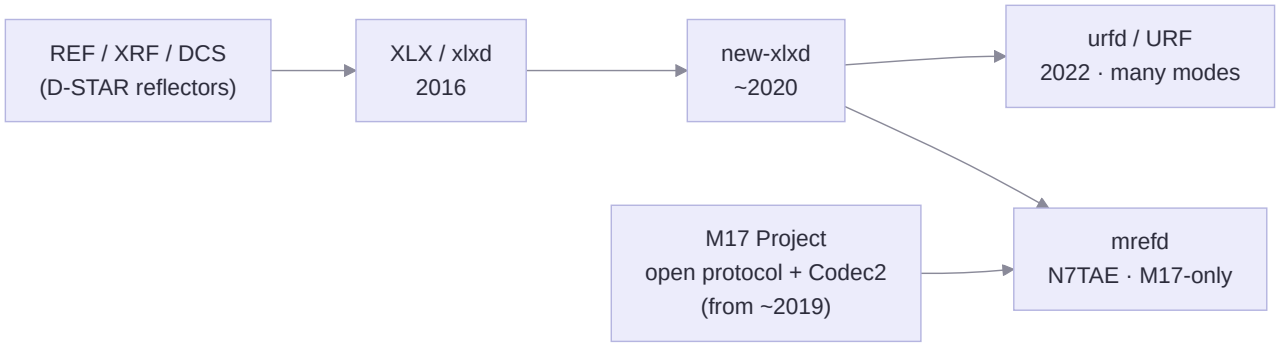
Chapter 2 — Where mrefd Came From

mrefd sits at the meeting point of two histories: the lineage of the reflector software it is built on, and the story of the M17 protocol it exists to serve.

The **software** lineage is the same chain the other guides trace. Three early D-STAR reflector systems — REF (DPlus), XRF (DExtra), and DCS — fed into **XLX** (LX3JL and LX1IQ, 2015–2016), the first widely-successful multi-protocol reflector with transcoding. Thomas A. Early (N7TAE) modernized that code into `new-xlxd` around 2020, and in 2022 extended it — with Doug McLain (AD8DP) — into `urfd`, the universal reflector that added M17, P25, NXDN, and USRP. **mrefd** is N7TAE's M17-only sibling of that work: the same modern C++ codebase and the same channel model, stripped down to serve one mode. Its copyright headers reflect the XLXD ancestry, and its development has been active and fast-moving (well over four hundred commits).

The **protocol** history is what gives mrefd its reason to exist. M17 is the product of the M17 Project, a community effort begun around 2019 to build a fully open digital-voice mode — “no patents, no royalties, and no licensing or legal barriers” — in deliberate contrast to the proprietary AMBE family that D-STAR, DMR, and Fusion depend on. Its voice mode uses David Rowe’s open **Codec2**; its protocol, framing, and even reference hardware (the Module17 board) are open. Because M17 is patent-free and software-friendly, an M17 reflector needs no vocoder hardware and no licensed codec — which is precisely why mrefd can be as simple as it is. In lineage terms, then, mrefd is where the open-codec future the other guides describe actually lands in a dedicated reflector.

The two threads that meet in mrefd:



Chapter 3 — The Channel Model

The central structural idea in mrefd is the **channel** — the same concept its siblings call a module. An mrefd reflector is not one room but as many as twenty-six of them, lettered A through Z, and each channel is a completely independent conference. Each channel accepts connections from M17 repeaters, hotspots, and other reflectors, and traffic arriving on a channel is sent to all the other clients on that same channel. People on channel A hear each other; people on channel B hear each other; the two do not mix unless the operator has deliberately interlinked them (Chapter 8).

Because mrefd is single-mode, the channel model is simpler here than in the transcoding reflectors. There is no notion of a “transcoded” versus an “untranscoded” channel — every channel carries M17, every client speaks Codec2, and everyone on a channel can always hear everyone else on it. An operator therefore uses channels purely for organization: a general calling channel, a net channel, a technical channel, a channel that permits encryption, a channel interlinked to a wider M17 network. The convention, as with the other reflectors, is to describe each channel’s purpose in the dashboard so users know which one to select. This same twenty-six-channel scheme is shared across the whole family — mrefd, urfd, and xlxd all think in A–Z — which is why an operator familiar with one feels at home on the others.

Chapter 4 — M17 in Depth: Stream Mode, Packet Mode, and Codec2

Where the other guides need a chapter to survey the several modes their reflectors carry, mrefd carries exactly one — so this chapter goes the other way and looks at M17 itself in a little depth, because understanding the mode explains most of what the reflector does.

On the air, M17 is a **4FSK** mode occupying roughly 9 kHz and running at 9600 bits per second, comparable to DMR or Fusion at the radio layer. What sets it apart is everything above that. Its voice mode uses **Codec2** at 3200 bit/s — an open, patent-free codec that, as its documentation notes, "uses half the bandwidth of Advanced Multi-Band Excitation to encode speech with similar quality." Every M17 transmission begins with a **Link Setup Frame (LSF)** that carries the source and destination addresses and metadata; the callsign is encoded directly into the signal rather than looked up in an external registry, so M17 stations identify themselves natively. A four-bit **Channel Access Number (CAN, 0–15)** gates channel access, analogous to a DMR color code.

M17 has two sub-modes, and mrefd handles both simultaneously. **Stream mode** is continuous voice (or voice plus data) — the mode a normal QSO uses, a steady flow of frames from key-up to key-down. **Packet mode** carries short, self-contained data bursts — text messages and similar — outside a continuous stream. An operator rarely has to think about the distinction, but it is why mrefd is described as carrying "voice-only and voice+data": both ride the same reflector.

One small but real piece of housekeeping the reflector enforces at this level is **callsign validity**. A connecting client's callsign must look like a real one — in the source's own description, it "begins with an optional digit, followed by one or two letters followed by one or two digit, followed by one to four letters." A station whose callsign doesn't fit that shape is refused, which quietly keeps malformed or spoofed identifiers off the reflector.

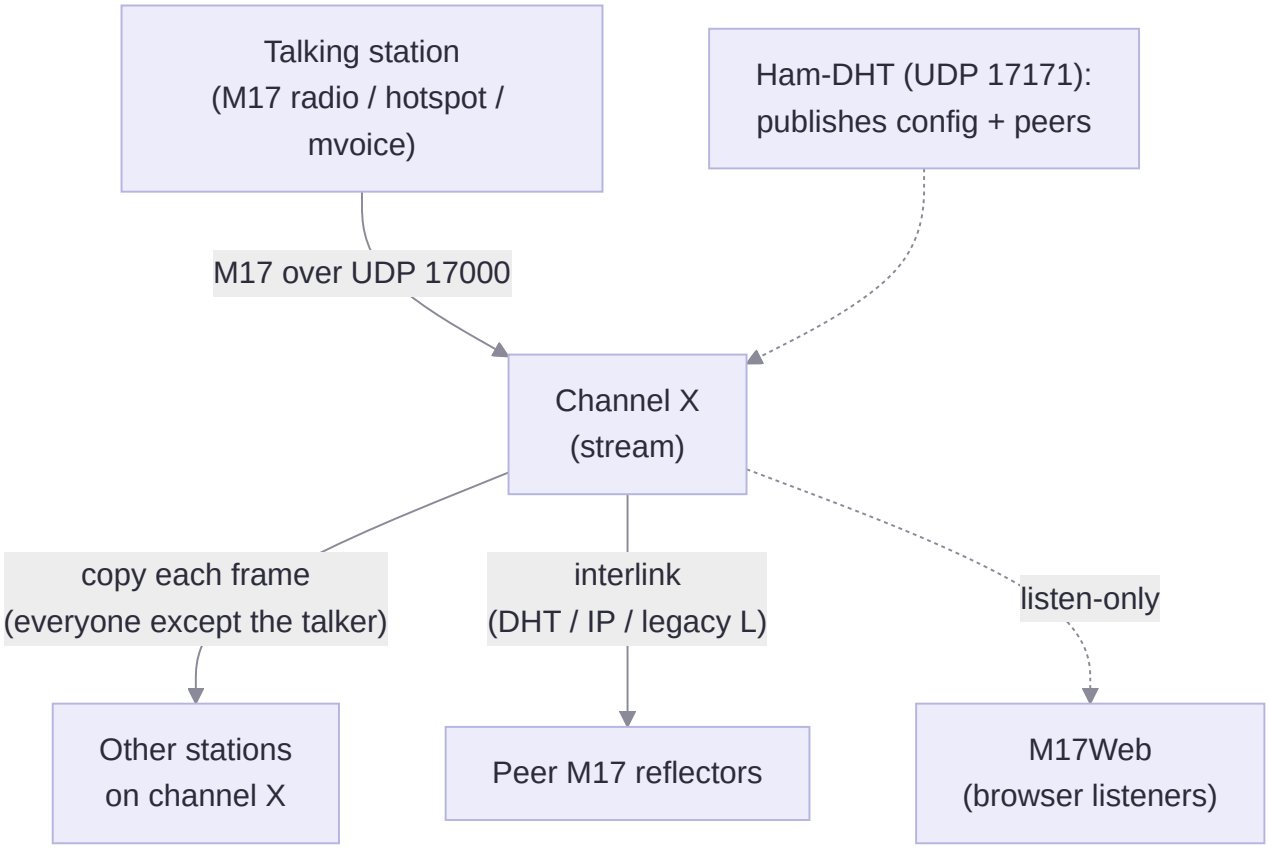
Chapter 5 — Inside the Reflector

It is worth opening the reflector up and seeing how it moves a voice stream, because with no transcoder in the path the design is about as clean as a reflector gets. What follows was read from the `mrefd` source, which carries its XLXD heritage in its structure.

At the center sits a reflector object that owns the connected clients, the linked peer reflectors, and a per-channel stream. Each M17 client — a hotspot, a repeater, a software client, or a peer reflector — connects over UDP and is associated with a channel. When someone keys up, the reflector opens a stream on that client's channel, marks the sender as the current talker, and then does the one thing that makes a reflector a reflector: it copies each frame out to *every other* client on the channel, and to any interlinked peers, *except* the station currently transmitting. Because every client speaks M17, that fan-out is a straight copy — no codec conversion, no diversion to a transcoder, no round-trip to a dongle. This is why an mrefd reflector is so light and so low-latency compared with a transcoding one: the frame that arrives is, byte for byte, the frame that leaves.

Two details are worth knowing because they surface later. First, **encryption is transparent to the reflector**: an encrypted M17 stream is relayed unmodified, and the reflector never decrypts it — it only decides, per channel, whether to permit encrypted traffic at all (Chapter 7). Second, the reflector publishes its state to the **Ham-DHT** network as a two-part document — its configuration for incoming clients, and its peer relationships — which is how other reflectors and dashboards discover it without a central server (Chapter 9). There is no separate transcoder process to monitor, no hardware to fail, and correspondingly fewer ways for audio to go missing.

A single voice stream through the reflector — note the absence of any transcoder:



Chapter 6 — How Clients Connect

Connecting to an mrefd reflector is simpler than to its siblings, because there is only one mode to connect in. What varies is the client, and M17 has a healthy and growing set — including one that needs no radio and no hardware at all, which makes testing your own reflector trivial.

The natural software client is N7TAE's **mvoice**, an M17-only desktop application that runs Codec2 in software and needs no AMBE dongle. You type the target reflector's callsign, pick a channel (module) letter, and talk — it is the quickest way to get on M17 from a computer and the easiest way to exercise a reflector you are building. **DroidStar** also speaks M17 (alongside the other modes it supports) and runs on desktop and mobile. Over the air, any standard **MMDVM hotspot** running M17 will reach an mrefd reflector, and the M17 Project's dedicated

Module17 board is a purpose-built open-hardware option. Because M17 is Codec2, none of these needs a vocoder chip — the codec runs in software everywhere.

Addressing follows M17's convention. Reflectors are named with an `M17-` or `URF-` style callsign; you select the reflector and then a single channel letter A–Z, and your client links to that channel. (As a practical detail carried over from the wider ecosystem, an M17-named reflector callsign is seven characters where a URF-named one is six.) The reflector will accept your connection only if your own callsign is well-formed and not blacklisted — the callsign-validity rule from Chapter 4 is applied at link time — so a malformed or disallowed callsign simply won't link.

Chapter 7 — Encryption, Signing, and the Law

M17 is unusual among amateur digital-voice modes in offering real cryptography, and mrefd is built to relay it. Because encryption is both a genuine feature and a legal minefield, it deserves a single chapter that treats the technology and the law together — on a single-mode M17 reflector they are the same conversation.

The M17 specification defines three options: **no encryption**, a simple **scrambler**, and true **AES** (a block cipher, used with a nonce). Encryption in M17 is *end-to-end*: it is applied in the transmitting radio and carried in the payload and the link-setup frame, and the reflector itself never holds a key and never decrypts anything. What mrefd controls is narrower and per-channel — whether a given channel will **permit** encrypted streams to pass at all. An encrypted stream on a permitting channel is relayed byte-for-byte to the other clients; on a channel that does not permit encryption it is refused. This per-channel permit is also enforced across links: as the documentation notes, the DHT "will prevent you from interlinking a channel where the peer module has encryption enabled and you do not, or vis versa" — so two reflectors cannot accidentally join a channel with mismatched encryption policy (Chapter 8).

The legal picture, for a United States operator, turns on FCC Part 97 §97.113(a)(4), which bars "messages encoded for the purpose of obscuring their meaning." M17's **AES** option is genuine encryption — it obscures meaning — and so, on a US amateur station, it runs afoul of that prohibition; the scrambler is best treated the same way, since its purpose is likewise to obscure. The practical takeaway is that a US mrefd operator should not permit encrypted streams on channels used for normal amateur traffic. There is, however, an important distinction that M17 makes possible: **ECDSA signing** is not encryption. As Bruce Perens has argued, a cryptographic signature "leaves the message text in the clear, and adds authenticating data" — it proves *who* sent a transmission without hiding *what* was said, and so it is defensible under a rule that forbids only the obscuring of meaning. Signing is the compliant way to get authentication on M17.

A note on jurisdiction: the encryption prohibition is a treaty-level principle — ITU Radio Regulation 25.2A bars transmissions between amateur stations of different countries that are "encoded for the purpose of obscuring their meaning" — and national licences transpose it, so the AES-versus-signing distinction holds in spirit almost everywhere, but the exact wording belongs to your own regulator. Some jurisdictions and closed groups do use M17 encryption where their rules allow; that is why mrefd offers the per-channel permit at all. This chapter is a factual summary, not legal advice.

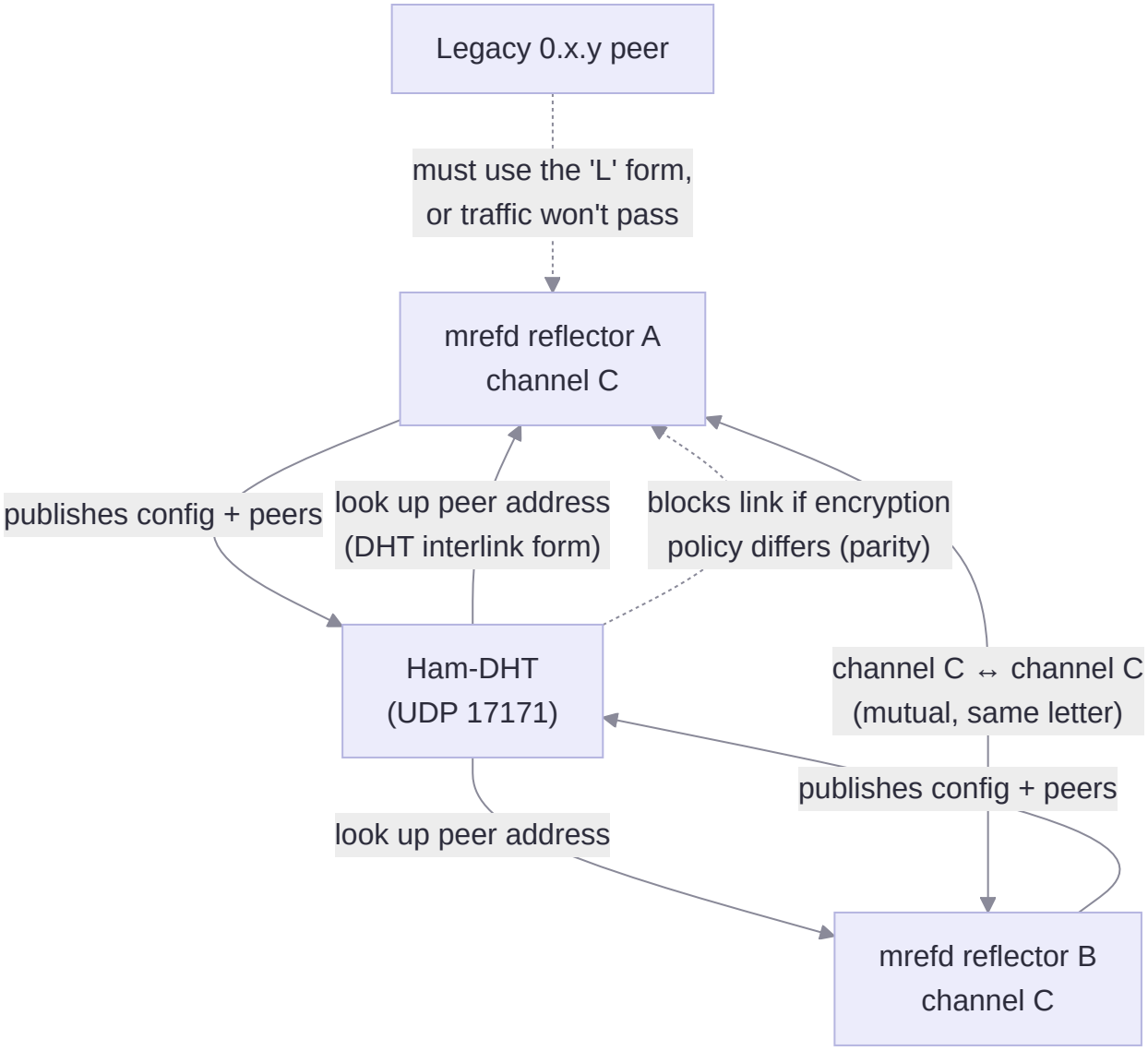
Chapter 8 — Interlinking and the Wider Network

Reflectors gain their reach by linking to each other, and mrefd does this through a file called `mrefd.interlink`. Interlinking joins a channel on your reflector to the same-letter channel on a peer, so that the two rooms become one; as with the other reflectors, links are mutual and module-to-module of the same letter.

What is distinctive on mrefd is that the interlink file supports **three different entry forms**, and choosing the right one matters. The first and recommended form is **DHT-based**: you name the peer reflector and the channel, and mrefd looks up its address over Ham-DHT automatically — no IP or port to maintain. The second is a **full IP-and-port** specification, for peers that are not on the DHT. The third is a **legacy "L" form**, kept for backward compatibility with older 0.x.y reflectors.

That legacy form is also the single biggest interlinking trap on M17, and it is worth stating in bold: if you link to a legacy reflector *without* using the "L" form, the link may appear to succeed while **traffic will not pass in either direction**. A link that forms but carries no audio is exactly the kind of failure that sends an operator hunting in the wrong place, so when a peer is an older reflector, use the legacy form. The DHT also adds a safety check the manual forms cannot: it enforces **encryption parity**, refusing to interlink a channel whose encryption policy differs from the peer's (Chapter 7), which prevents a whole class of confusing half-working links.

Interlinking and discovery, with the encryption-parity check the DHT enforces:



Chapter 9 — Finding and Registering: Ham-DHT and the M17 Registries

For radios and hotspots to connect to a reflector, its existence and address have to be published somewhere. mrefd is **DHT-native**: it was built in the era of Ham-DHT and prefers it, which makes discovery cleaner than the central-registry model the older XLX software depends on.

Ham-DHT is a decentralized directory built on the OpenDHT library (a Kademlia distributed hash table). Each mrefd reflector publishes a **two-part document** to the DHT — its configuration (the connection parameters an incoming client needs) and its peers (its interlink relationships) — keyed so that other reflectors, clients, and dashboards can look it up directly, with no central server in the loop. A reflector joins the network by bootstrapping through a known node, and the `ham-dht-tools` utilities can query and map the whole thing from

anywhere. This is the same DHT that URF uses; on mrefd it runs on UDP port 17171 and is the primary way both interlinking (Chapter 8) and client discovery happen.

Alongside the DHT sit the human-facing **M17 registries and host lists** — the directories and dashboards where operators list their reflectors and where a reflector acquires its recognizable `M17-xxx` name, and the host files that populate client and hotspot menus. Between the DHT for machines and the registries for people, an mrefd reflector is straightforward to find. Worth noting by contrast with the XLX guide: mrefd has no "calling home" to a central XLX API — discovery here is DHT-first, which is one of the ways the M17 side of the ecosystem is architecturally more modern.

Chapter 10 — Special Features: Parrot and M17Web

Two mrefd features don't fit the other chapters but are among the first things a new operator or user meets, so they get their own short chapter.

The first is the built-in **#PARROT** echo test. Key up with your destination set to `#PARROT` and the reflector records your transmission and plays it straight back to you, so you can hear exactly how you sound over M17 — invaluable for checking a radio, a hotspot, or your own audio levels without needing another operator on frequency. The reflector accepts up to about a ten-second transmission (a cap of 500 Stream-mode packets), and the audio returns roughly forty milliseconds after you unkey, addressed back only to you. One limitation follows directly from Chapter 7: parroting does *not* work with encrypted streams, since the reflector would have to handle content it deliberately never decrypts.

The second is **M17Web**, a listen-only way to follow a reflector from a web browser. Launched by the M17 Project in 2025, it lets anyone open a page and hear a reflector's live traffic without a radio, a hotspot, or even a client install — a low-friction way to monitor activity, share a net with newcomers, or simply listen in. It is a listen-only path: M17Web receives, it does not transmit. Together, #PARROT (test your own audio) and M17Web (hear everyone else's) make an mrefd reflector unusually easy to get started with and to demonstrate.

Chapter 11 — Building and Operating One

Bringing an mrefd reflector up is the least involved build in this set of guides, precisely because there is no transcoder or hardware to integrate. The software runs on systemd-based Linux, with Debian or Ubuntu recommended, on a machine with a permanent internet connection and a public address. You install the dependencies, clone the `mrefd` repository, and configure it.

Configuration lives in a small set of files. **`mrefd.mk`** holds compile-time settings, and **`mrefd.cfg`** holds the runtime operational parameters — the reflector's identity, its channels, encryption permits, the DHT bootstrap, and so on. Alongside these are **`mrefd.blacklist`** and **`mrefd.whitelist`** for callsign access control (an empty whitelist means an open reflector; a non-empty one restricts access to listed callsigns) and **`mrefd.interlink`** for peer links (Chapter 8). A genuinely convenient touch: the reflector *watches these files*

and updates itself dynamically whenever anything changes, so adding an interlink or blacklisting a callsign takes effect without a restart.

You build and install with `make && sudo make install`, which compiles the binary and installs the systemd service. Day to day it is an ordinary service — `sudo systemctl start`, `stop`, `restart mrefd`, and `sudo journalctl -u mrefd -f` to watch it live. Because a single host can carry more than one reflector, running **multiple instances** is supported: give each a different listening port, its own service file, and its own configuration paths. Updating is a `git pull` and rebuild; since your settings live in the config files rather than in edited source, a rebuild does not clobber them — and because releases move quickly, record the git commit you built from so you can answer "which version am I running."

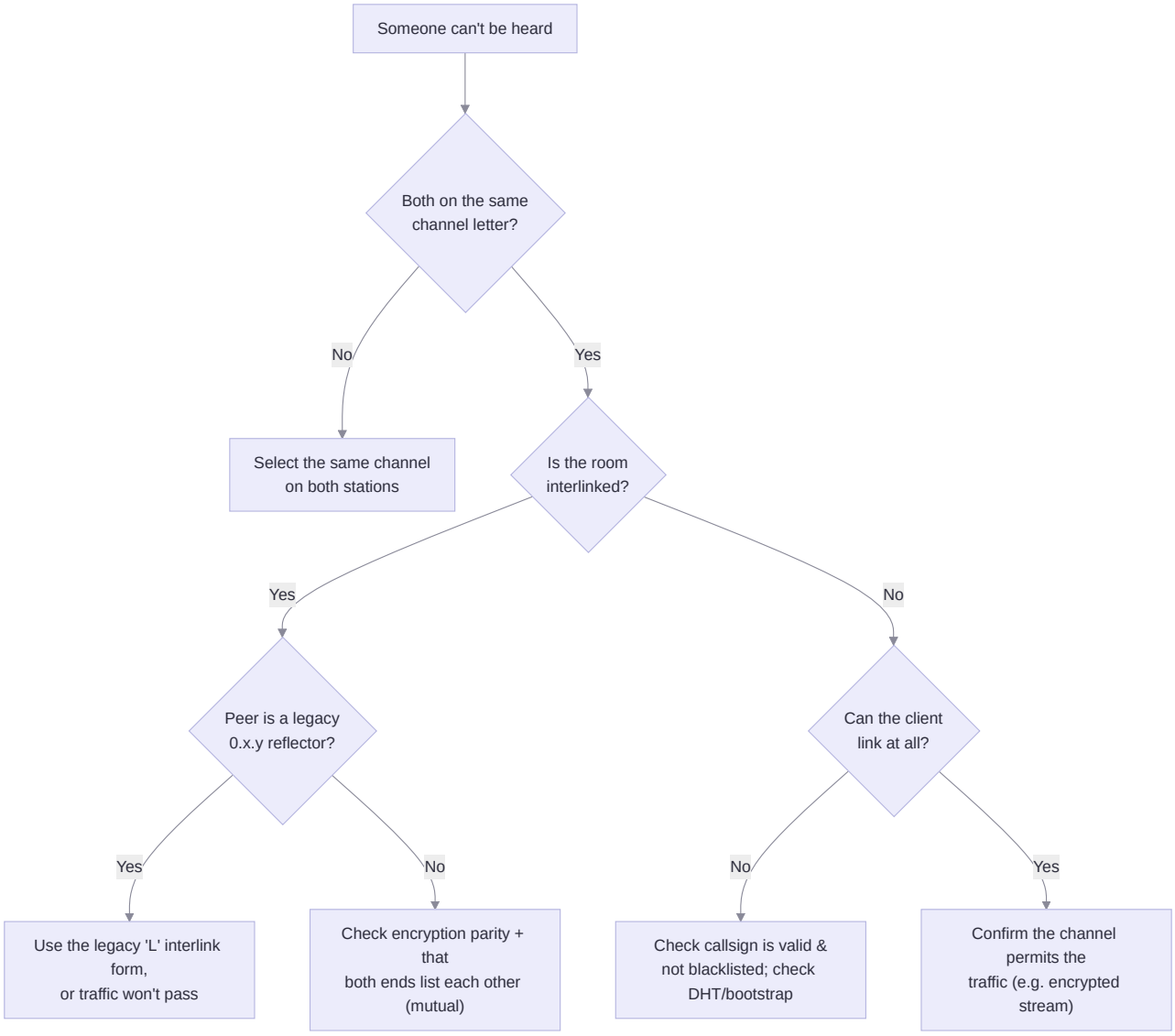
A quick way to verify a fresh build works: connect with **mvoice** (no dongle needed), link to a channel, and key **#PARROT** — if you hear yourself echoed back, the reflector is accepting connections, opening streams, and fanning audio correctly. Then confirm the reflector appears on the DHT (with `ham-dht-tools`) and that your dashboard shows your test transmission in its Last Heard list.

Chapter 12 — When Things Go Wrong

The failure modes of an mrefd reflector are a short list — shorter than its siblings', because the whole transcoder-and-hardware category of problems simply doesn't exist. There is no dongle to go missing and no transcoder to die, so "callsigns appear but there's no audio" is far less common here. The handful of real causes are worth knowing.

The most common is a **user on the wrong channel** — two people expecting to talk but linked to different channel letters. Because every channel carries the same codec, being on the same channel is all that's required, so this is usually just a matter of confirming both stations selected the same module. Next is the **legacy-"L" interlink trap** from Chapter 8: a link to an older reflector made without the legacy form can appear to succeed while no audio passes in either direction — if an interlinked room is silent, check the interlink form first. Closely related is an **encryption-parity mismatch**: a channel that permits encryption cannot cleanly interlink with a peer channel that doesn't, and the DHT will refuse the link (which is the system working as intended, even if it looks like a failure). Then there are the ordinary ones: a **blacklisted or malformed callsign** that won't link (remember mrefd enforces callsign validity), and a **discovery problem** — a reflector that can't be found — which on a DHT-native reflector means checking the bootstrap node and that the reflector is publishing to the DHT.

The short decision tree — note there is no transcoder branch:



Chapter 13 — Performance, Sizing, and Hardening

mrefd is the lightest reflector in this set to host, and the sizing story is correspondingly short. There is no published benchmark, but first principles bound it easily.

Sizing from first principles

An M17 stream on the wire is small — budget on the order of **~25 kbit/s per active stream per direction** once IP/UDP overhead is counted. As always, a reflector *fans out*: one talker on a channel with *N* listeners sends the audio out *N* times, and only one person talks at a time per channel. So a busy channel with 30 listeners costs roughly 0.75 Mbit/s while someone is talking, and a whole reflector with several active channels lands in the low single-digit megabits — nothing for any VPS. The crucial difference from the transcoding reflectors is on the **CPU** side: because mrefd never transcodes, it does no signal processing at all — a keyed-up stream is a straight

memory copy out to the other clients. There is no per-channel vocoder instance, no DVSI round-trip, nothing heavy. This is why mrefd runs comfortably on a **micro-VPS or a Raspberry Pi** and why a single modest host can carry several instances (Chapter 11). The practical limit is bandwidth and file descriptors at very large client counts, not CPU.

Security and hygiene

mrefd's built-in security is the callsign blacklist and whitelist, plus the callsign-validity check and per-channel encryption permits — everything else is ordinary Linux server hygiene and falls to the operator. Run a firewall that denies by default and opens only the ports you actually use — for mrefd that is a short list: the M17 port (UDP 17000 by default), the DHT port (UDP 17171), and the dashboard's 80/443 (Appendix A). Use key-only SSH and something like fail2ban for SSH and the dashboard, remembering that fail2ban does nothing for the open UDP ports, which — because UDP has no handshake — are exposed to spoofed-source floods; rate-limit per source at the firewall and lean on upstream protection for a real flood. Keep the clock disciplined with NTP so dashboard timestamps are right, cap the logs, and mirror your IPv4 firewall rules on IPv6. Back up the small set of config files (they hot-reload, so they are the whole of your state) and record the git commit you built from. Monitoring is easy: watch that the service stays active and that the reflector is still publishing to the DHT.

Chapter 14 — The Dashboard: Setup, Use, and Access

The dashboard is the reflector's public face — the web page hams open to see who is connected, who is talking, and which channel carries what. For most users it is the only part of a reflector they ever touch directly, so it deserves its own chapter.

What it is, and how it is set up

There is a small choice of M17 dashboards, all of which read the reflector's published state and render it as a web page. mrefd ships a simple PHP dashboard in its `php-dash` folder; the M17 Project maintains a more polished one, `ref-dash`; and DU8BL maintains a Go-based dashboard kept current with recent mrefd versions. All are deliberately simple readers — they hold no state of their own and do not talk to the reflector directly, so a dashboard can run on the same host or a separate web server, and restarting the reflector never disturbs it. Setup is the last step of a build: deploy the dashboard into your web server, point it at the reflector's data, set the operator-facing details (callsign, description, channel names, contact), and serve it over HTTPS (Chapter 15).

What it shows, and how hams use it

An M17 dashboard presents the reflector's live state as a few panels: an identity banner (the reflector's `M17-xxx` callsign and description); a **channel list** with each channel's purpose, so a newcomer knows which to select; a **linked-peers / connected-nodes** list showing callsign, channel, and connect time; and the panel everyone watches, the **"Last Heard" list** — the most recent stations to key up, with callsign, channel, and timestamp. After you transmit (or key #PARROT), seeing your callsign appear in Last Heard on the expected channel is the quickest confirmation the reflector heard you.

How hams access it

Access is frictionless: the dashboard is an ordinary web page at the reflector's hostname, opened in any browser with no login to view. That URL is what an operator publishes and what appears in the M17 registries and host lists. Viewing is read-only; there is no web-based administration, so an operator changes configuration on the server (Chapter 11), not through the dashboard. And uniquely on M17, a reflector can pair its dashboard with **M17Web** (Chapter 10) so visitors can not only see who is active but actually *listen* in the browser.

A representative M17 reflector dashboard layout (illustrative — a live dashboard's styling varies):

M17-EXA — Open M17 Reflector ● online · updated 3s ago

Channels: A — Calling | B — Net | C — Encrypted (permit)

LAST HEARD

Callsign	Channel	Mode	Last heard (UTC)
W1ABC	A	M17 Stream	2026-08-08 17:42:10 ◀ transmitting
K2AS	A	M17 Stream	2026-08-08 17:41:58
VK3XYZ	B	M17 Stream	2026-08-08 17:39:12
N0CALL	A	#PARROT	2026-08-08 17:37:40

CONNECTED CLIENTS

Callsign	Channel	Type	Since (UTC)
W1ABC	A	hotspot (M17)	16:02
VK3XYZ	B	mvoice	15:47

INTERLINKED PEERS

Callsign	Channel	Link form	Since (UTC)
M17-USA	A	DHT	Aug 07 09:14

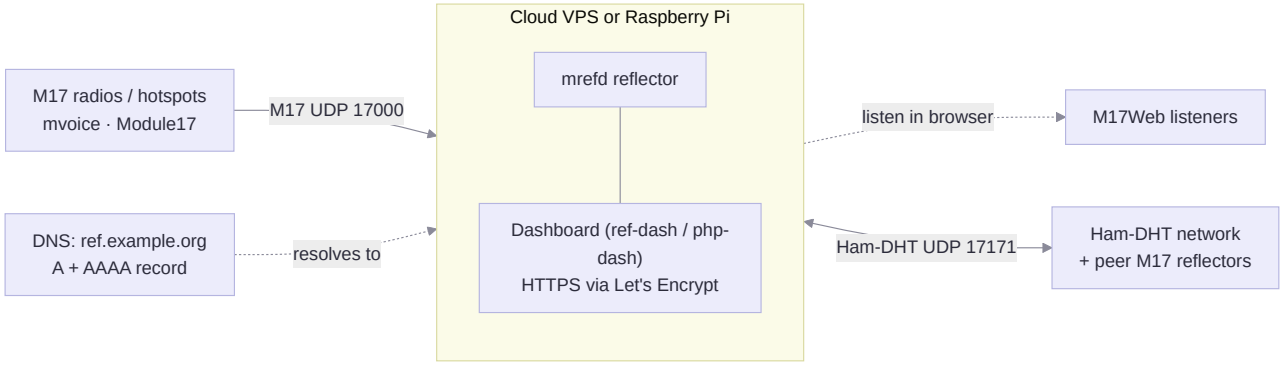
Chapter 15 — Deployment: DNS, HTTPS, and Watching the Traffic

Turning a working reflector into a properly deployed public service takes a few steps the software itself does not perform, though on mrefd there are fewer of them than on a transcoding reflector.

It starts with **DNS**: give the reflector a stable hostname — an A record, and an AAAA record if the host has IPv6 — so that `ref.example.org` maps to the server's public address and can change without breaking every user's configuration. Next is **HTTPS for the dashboard**: it is a web page and should be served securely, which on an Apache host is a one-command job with Let's Encrypt via `certbot --apache`, renewing automatically;

because the dashboard only reads the reflector's data, it can even live on a separate machine behind a reverse proxy. Finally, if you want to **see your dashboard traffic** without a heavy commercial tracker, a lightweight, privacy-respecting, self-hostable analytics package such as GoatCounter suits a hobby community page and is entirely optional.

A complete mrefd deployment in one picture:



Chapter 16 — Bridging Beyond mrefd: Reaching Other Modes and Networks

Because mrefd is M17-only, its clean simplicity comes with a natural boundary: an mrefd reflector connects M17 stations to each other, and nothing more. The moment you want an M17 user to talk to a D-STAR, DMR, or Fusion user, you have left what mrefd does — and that is by design. Reaching other modes is a job for other software, and there are two clean ways to do it.

The first, and usually the best, is to **link out to a URF reflector**. URF is a superset of the whole family: it carries native M17 *and* the other modes, with a transcoder. So an M17 room can reach D-STAR, DMR, Fusion, and P25 by having M17 traffic land on a URF reflector's M17 module, which URF then transcodes onto its other modules. In practice, an operator who wants both a pure M17 experience and cross-mode reach often runs (or joins) a URF reflector for the bridging and uses mrefd where dedicated, unbridged M17 is wanted. The second route is the general-purpose one from the other guides: **DVSwitch** and its USRP/analog bridges, which can plumb an M17 feed to AllStar, EchoLink, or other networks where no direct link exists.

The division of labour is clean once you see it, and it is the same principle in reverse from the other guides. Use **mrefd** when you want M17 at its best — lowest latency, no transcoding loss, newest M17 features, cheapest to run. Use **URF** (or a bridge) when M17 must share a room with other modes and you accept the transcoding cost that comes with crossing codecs. Neither is "better"; they are tools for different jobs, and they interoperate happily because they share an author and a lineage.

The bridging fabric, and where M17 meets it

The analog and VoIP world speaks **USRP** — "8 kHz signed 16-bit PCM samples" over UDP — and DVSwitch's **Analog_Bridge** is the universal adapter between that plain-PCM world and coded digital modes. There is one M17-specific wrinkle worth stating plainly: mrefd carries only **Codec2**, whereas the DVSwitch codec chain grew up around the AMBE family, so the cleanest way to give M17 users analog and VoIP reach is usually to **link the mrefd channel to a URF reflector's M17 module** (Chapter 17) and let URF place the audio onto USRP. Whatever anchors the USRP end — a URF reflector, or a Codec2-capable bridge where one is available — the destinations below are then reached in exactly the same way, and the mechanics are identical to those in the URF and XLX guides.

Reaching AllStar

AllStarLink is a VoIP network of amateur nodes built on the Asterisk PBX and its `app_rpt` application; every node has a number, and users link nodes with DTMF — `*3<node>` to connect two-way, `*2<node>` to connect monitor-only, `*1<node>` to drop one node, `*76` to drop all, and `*70` / `*75` for status. The link is a **private AllStar node** (often numbered 1999) whose channel is the USRP driver — in `rpt.conf`, `rxchannel = USRP/127.0.0.1:34001:32001` — with a DVSwitch Analog_Bridge mirroring those ports (`[USRP] rxPort = 34001`, `txPort = 32001`) on the other end. With the M17 side anchored through URF, AllStar users and M17 users then hear each other across that bridge.

Reaching EchoLink

EchoLink (Jonathan Taylor, K1RFD) links licensed amateurs' computers, phones, and RF nodes over the internet. It uses UDP `5198` (RTP audio) and `5199` (RTCP control), with a central directory (commonly cited on TCP `5200`) that **validates each user's callsign** before first use; nodes come as plain users, `-L` simplex links, `-R` repeaters, and conference servers (a leading `*`). Nothing in this family speaks EchoLink directly — the route is *through AllStar*, whose EchoLink channel driver (`chan_echolink`, configured in `echolink.conf`) presents EchoLink nodes as AllStar nodes using a node-number offset: the EchoLink number is padded to six digits and prefixed with a `3`, so node 1234 becomes `3001234` and you connect with `*3 3001234`. The full chain is therefore *mrefd ↔ URF (M17 module) ↔ USRP ↔ Analog_Bridge ↔ AllStar (chan_echolink) ↔ EchoLink*.

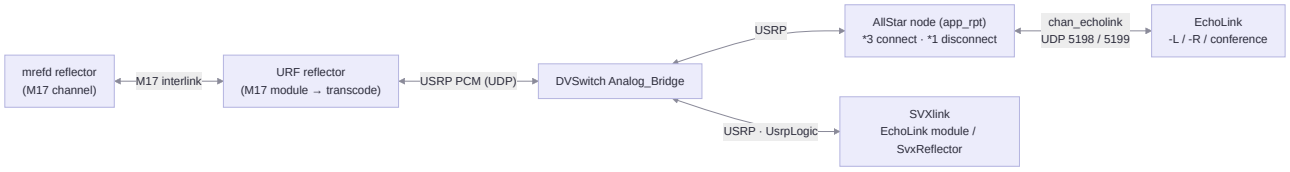
Reaching SVXlink and SvxReflector

SVXlink (Tobias Blomberg, SM0SVX) is an open-source Linux repeater controller with an **EchoLink module** (so an SVXlink node can itself be an EchoLink node) and **SvxReflector**, its own reflector network for linking SVXlink systems. It is bridged to digital voice over the same USRP fabric through SVXlink's **UsrpLogic**, which connects it to a DVSwitch Analog_Bridge; a common community build (F5VMR's SVXlink-with-UsrpLogic) packages this, and because SVXlink has no built-in way to pick digital destinations, operators drive the connection from the DVSwitch Mobile app or a fixed configuration.

The general pattern

The through-line is that anything which can source or sink USRP PCM can be joined to the wider network, and for M17 the anchor point is almost always a URF reflector doing the codec crossing. The cost is the one this

guide is otherwise happy to avoid: crossing from Codec2 to any other codec is a transcode, with the intelligibility and latency penalty from Chapter 20 — which is exactly why a *pure* mrefd room, unbridged, sounds best. Bridge when you need reach; stay on mrefd when you want M17 at its cleanest.



Chapter 17 — Choosing What to Run: mrefd vs URF

For anyone standing up an M17 presence, the one decision that matters most is mrefd versus URF, because both carry M17 and they are built by the same author on shared code. The choice is genuinely about what you want, and it is worth being candid.

Run **mrefd** when M17 is the point. It is the lean, dedicated M17 reflector: no transcoder, no hardware, the cheapest to host, the lowest latency (a keyed stream is a straight copy), and no transcoding loss because there is only one codec end to end. It also tends to get **new M17 features first** — M17Web, interlink refinements, encryption handling — because it is the M17-focused project. For an M17 club, an M17 experimenter, or anyone who wants a clean high-quality M17 room, mrefd is the right tool.

Run **URF** when M17 must live alongside other modes. URF carries M17 *and* D-STAR, DMR, Fusion, P25, and NXDN, and its transcoder lets users of those modes actually talk to your M17 users. If your goal is a multi-mode community where an M17 operator and a DMR operator share a conversation, that is exactly what URF is for, and mrefd cannot do it alone (Chapter 16). The cost is complexity and, for D-STAR bridging, DVSI hardware — the whole apparatus this guide is happy to be free of.

Because the two share a lineage, an operator loses little by choosing wrong and changing later: the channel/module model, the interlink concepts, and the DHT discovery all carry across. A common and sensible pattern is to run **both** — an mrefd for dedicated M17 and a URF for the cross-mode bridge — and interlink or point users between them. The honest one-line summary: **mrefd for M17 done purely; URF for M17 connected to everything else.**

Chapter 18 — The Case For mrefd: Strengths and Benefits

Having taken the machine apart, it is worth stepping back and asking plainly what mrefd is *good* at.

Its central strength is **simplicity, and everything that flows from it**. With one mode and one codec there is no transcoder, no DVSI hardware, and no codec conversion — which means no dongle to buy, nothing to fail in the audio path, and a host so light that a micro-VPS or a Raspberry Pi runs a busy reflector without noticing. It is the cheapest reflector in this set to stand up and the easiest to keep running.

Its second strength is **M17 quality and currency**. Because every client speaks Codec2 end to end, there is no tandem vocoding and therefore no transcoding loss — audio is as good as M17 gets — and latency is low because a frame is copied, not processed. As the M17-focused reflector, it also tends to receive new M17 capabilities first: per-channel encryption handling, interlink improvements, and the M17Web listen-in-browser experience all live naturally here.

Finally, there are the virtues of how it is built. mrefd is **open all the way down** — an open reflector, for an open protocol, on an open codec, under the GPL — which is philosophically the whole point of M17. It is **DHT-native**, so discovery and interlinking need no central server; its config files **hot-reload**, so routine changes need no restart; it supports **multiple instances** on one host; and it comes with genuinely useful touches like the **#PARROT** self-test. For its intended job — a dedicated M17 room — it is hard to beat.

Chapter 19 — The Case Against: Limitations and Trade-offs

An honest reference has to be as clear about the weaknesses as the strengths, and mrefd's limitations follow directly from its focus.

The defining limitation is that it is **M17-only**. It cannot, by itself, connect an M17 user to a D-STAR, DMR, or Fusion user; cross-mode contact requires linking to a URF reflector or an external bridge (Chapter 16). For a multi-mode community, mrefd alone is not the answer. Bound up with this is the reality that M17, while growing quickly, still has a **smaller user base** than the long-established DMR and D-STAR ecosystems, so an mrefd reflector may see less traffic than a DMR talkgroup — a matter of the mode's adoption rather than the software.

There are sharper edges, too. The **legacy-"L" interlink trap** (Chapter 8) is a real gotcha that can produce a silent, seemingly-successful link. Built-in security is **minimal** — a callsign blacklist, whitelist, and validity check — so firewalling and DDoS resilience are the operator's job, and the open UDP ports are inherently exposed. M17's **AES encryption is not usable on US amateur stations** (Chapter 7), so a headline M17 feature is off-limits for many operators. And mrefd is a **young, fast-moving project**: configuration file names and features have changed between versions, documentation is still catching up in places, and a how-to written a year ago may not match today's source (Chapter 24). None of this makes mrefd a poor choice; it makes it a focused one — superb for M17, and reliant on its siblings for everything beyond it.

Chapter 20 — Voice Quality and Latency

Here mrefd has a genuine, structural advantage over every transcoding reflector, and it is worth stating clearly because it is the payoff of the single-mode design.

On a transcoding reflector, any cross-mode contact is *tandem vocoding* — decode one codec to audio, re-encode into another — and each stage measurably degrades intelligibility (the literature puts the loss at more than ten percent in some configurations). On mrefd there is **no tandem vocoding at all**, because there is only one codec: the Codec2 frame that arrives is the Codec2 frame that leaves, byte for byte. The audio quality a listener hears is

simply the quality of the sender's M17 encoding, undiminished by the reflector. Codec2 at 3200 bit/s is not quite AMBE+2 on a like-for-like clip — its own author characterizes AMBE as "a little better, Codec 2 is a bit clicky or impulsive" — but that is the codec's own character, and on M17 you hear it once, not compounded by a transcode.

Latency is the other payoff. With no transcode stage and no DVSI round-trip, an mrefd hop adds only network round-trip time and a small jitter buffer — there is no decode-to-PCM-and-re-encode in the path, and no separate transcoder machine to reach. A same-network M17 contact through an mrefd reflector is therefore typically toward the low end of the few-hundred-millisecond range that all internet-linked digital voice occupies, and noticeably tighter than a transcoded cross-mode contact on a busier reflector. Quality and latency both degrade only when you deliberately bridge M17 out to another mode (Chapter 16), at which point the transcoding cost is incurred by whatever does the bridging, not by mrefd itself.

Chapter 21 — Who Should Run One, and What It Costs

The audience for an mrefd reflector sorts itself out from everything above. It is an excellent fit for an **M17 club or regional group** that wants a dedicated home for the mode; for an **experimenter or enthusiast** drawn to M17's open, hackable nature; for anyone who wants a **cheap, low-maintenance** reflector with no hardware to buy or maintain; and as the **M17 half of a larger setup**, run alongside a URF reflector that handles cross-mode bridging. It is a poor fit for someone who needs multi-mode interoperability from a single server (that is URF), or who wants to reach the large existing DMR or D-STAR populations directly without bridging.

The costs are the lowest in this set of guides, and worth stating concretely (*figures as of mid-2026*). There is **no hardware cost at all** — no DVSI dongle, because M17 needs no vocoder chip. On the hosting side, mrefd is light enough for the smallest servers: a micro-VPS with one virtual CPU and a gigabyte of memory, at roughly a few US dollars a month (or a few euros from the cheaper providers), is more than enough, and a spare Raspberry Pi will do it for the cost of electricity. So the realistic entry cost is essentially just a few dollars a month of hosting — and nothing more. For an operator who has been eyeing the DVSI dongle prices in the other two guides, that is the headline: mrefd is the reflector you can run for the price of a coffee a month.

Chapter 22 — Frequently Asked Questions

Do I need a radio to use an mrefd reflector? No. N7TAE's mvoice is an M17-only desktop client that runs Codec2 in software and needs no dongle, and DroidStar works too — either lets a computer connect directly, which is also the easiest way to test your own reflector.

Do I need a DVSI dongle or any hardware? No, never. M17 uses the open Codec2 codec, which runs in software everywhere, and mrefd does no transcoding — so there is no vocoder hardware anywhere in an M17 reflector. This is the biggest single difference from XLX and URF.

Should I run mrefd or URF? Run mrefd for a dedicated, low-cost, high-quality M17 room. Run URF if M17 needs to share a room with D-STAR, DMR, Fusion, or P25 via transcoding. They share an author and lineage, so it is easy to run both. See Chapter 17.

Is M17 encryption legal? In the United States, AES (and the scrambler) obscure meaning, which Part 97 forbids, so they should not be used on normal amateur channels; M17's ECDSA *signing*, which authenticates without hiding the message, is the compliant way to get authentication. Elsewhere, check your own regulations. See Chapter 7.

What is #PARROT? A built-in echo test: key up with your destination set to `#PARROT` and the reflector plays your transmission back to you (up to ~10 seconds), so you can hear how you sound on M17. It does not work with encrypted streams.

Can people listen without a radio or client? Yes — M17Web streams a reflector's traffic to a web browser, listen-only, with nothing to install (Chapter 10).

My interlink formed but no audio passes — why? Almost certainly the legacy-"L" trap: linking to an older (0.x.y) reflector without the legacy interlink form lets the link form but carries no traffic. Use the legacy form for older peers. See Chapter 8.

How do people find my reflector? Over Ham-DHT (mrefd publishes its config and peers to the DHT — set your bootstrap node) and through the M17 registries and host lists, which is also where it gets its `M17-xxx` name. See Chapter 9.

Can one server run several reflectors? Yes. Run multiple instances with distinct ports, service files, and config paths (Chapter 11).

Chapter 23 — Getting Help

There is no single official support desk for mrefd, so it helps to know where to look. The primary channel is GitHub: the `n7tae/mrefd` repository and its issue tracker carry the source and reach the author directly, and the dashboards live in their own repositories (the M17 Project's `ref-dash`, the bundled `php-dash`, and DU8BL's Go dashboard). For M17 more broadly — the protocol, clients, hardware, and M17Web — the **M17 Project** runs an active community on Matrix and Discord along with an M17-Users mailing list on groups.io, and its website carries the protocol specification and project news. Hotspot-side configuration for connecting to an M17 reflector is Pi-Star and WPSD territory, and broader digital-voice questions are answered on the RadioReference forums. Because mrefd shares an author and a code lineage with URF, much of the URF-side advice transfers directly — the companion URF and XLX guides are useful second references, especially on Ham-DHT and interlinking concepts.

Chapter 24 — Where the Sources Disagree

In keeping with the principle of not smoothing over genuine conflicts, a few deserve to be named — and on a young, fast-moving project there are more moving targets than in the mature XLX world. The sharpest is **configuration-file naming**: current mrefd uses `mrefd.mk` for compile-time settings and `mrefd.cfg` for runtime parameters, but the wider family (and some older mrefd documentation) uses `.ini` and `.h` conventions, and how-tos written for different versions name these files differently. Where this guide gives a name it reflects the current source; if your build differs, trust the build. The **interlink forms** — and especially the exact behavior of the legacy-"L" form — are described consistently in outline but vary in detail across versions, so verify against your own reflector. **Dashboard choice** is genuinely plural (ref-dash, php-dash, DU8BL's Go dashboard), and which is "standard" depends on who you ask and when. And because M17 features have been landing quickly — M17Web is a 2025 addition — any capability described here may have grown by the time you read it. None of this undermines the picture; it is simply the nature of documenting active software.

Chapter 25 — What Only a Live Operator Can Confirm

Finally, in the same spirit of honesty, a few things in this guide rest on lighter evidence than the rest, and a working operator could firm them up. The exact current **config keys** in `mrefd.cfg` and `mrefd.mk`, and the precise section names, should be confirmed against the version you build, since they have moved between releases. The precise **fields of the two-part DHT document** mrefd publishes, and the exact **M17Web** setup steps, are described from documentation and project announcements rather than confirmed on a running reflector during research. The specific behavior of the **legacy-"L"** interlink against a real 0.x.y peer, and the current default **ports** if changed from those given, are worth checking live. And because mrefd releases move quickly and are loosely tagged, the honest way to record "which version am I running" is to note your built commit hash. None of these gaps undermines the guide; they are the edges where a practitioner's fifteen minutes on a live reflector would be worth more than any amount of further reading.

Glossary

4FSK — the four-level frequency-shift-keying modulation M17 uses on the air.
AES — the strong block cipher M17 can use for end-to-end encryption; obscures meaning, so not usable on US amateur channels.
AllStar (AllStarLink) — a VoIP network of amateur nodes built on the Asterisk PBX (app_rpt); reached from M17 by bridging through URF/DVSwitch over USRP.
CAN (Channel Access Number) — M17's four-bit channel-access code (0–15), analogous to a DMR color code.
Channel — one of an mrefd reflector's up-to-26 independent rooms, lettered A–Z (the same idea the other reflectors call a module).
Codec2 — David Rowe's open, patent-free voice codec used by M17 (3200 bit/s in M17's voice mode).

DHT (distributed hash table) — a decentralized, serverless key/value directory; the basis of Ham-DHT.

DroidStar — a multi-mode software client (including M17) that connects from a computer or phone.

DVSwitch — a suite of tools (Analog_Bridge, MMDVM_Bridge) for bridging digital-voice modes and networks to each other and to analog over USRP.

ECDSA signing — a cryptographic signature that authenticates a transmission without hiding its content; the encryption-legal way to prove identity on M17.

EchoLink — a long-established VoIP system linking licensed amateurs' computers, phones, and RF nodes; reached from M17 through AllStar's EchoLink channel driver.

Ham-DHT — the amateur distributed-hash-table directory (built on OpenDHT) that mrefd uses to publish and discover reflectors and peers.

Hotspot — a small personal device that links a radio to internet digital-voice networks; MMDVM hotspots can run M17.

Interlink — a mutual, same-letter link between two reflectors; on mrefd it has DHT, IP/port, and legacy "L" forms.

LSF (Link Setup Frame) — the first frame of an M17 transmission, carrying source/destination addresses and metadata.

M17 — the open, patent-free amateur digital-voice protocol built on Codec2, developed by the M17 Project.

M17Web — a listen-only way to hear a reflector's traffic in a web browser (M17 Project, 2025).

Module17 — the M17 Project's open-hardware M17 modem/board.

mrefd — the M17 Reflector Daemon, N7TAE's M17-only reflector; this guide's subject.

mrefd.cfg / mrefd.mk — the runtime and compile-time configuration files (naming varies by version; see Chapter 24).

mvoice — N7TAE's M17-only desktop client; runs Codec2 in software, needs no dongle.

#PARROT — mrefd's built-in echo test; plays your transmission back so you can hear your own audio.

Packet mode — M17's short-data-burst sub-mode, carried alongside voice.

Reflector — a server that relays a mode's traffic among all connected stations; a conference bridge.

Scrambler — M17's simpler encryption option; like AES, it obscures meaning, so treat it as off-limits on US amateur channels.

Stream mode — M17's continuous-voice sub-mode, used for a normal QSO.

SVXlink — an open-source Linux repeater controller (SM0SVX) with an EchoLink module and its own SvxReflector network; bridged to digital voice over USRP via UsrpLogic.

SvxReflector — SVXlink's reflector network for linking SVXlink systems together.

URF / urfd — the universal, many-mode reflector (a superset of XLX) that also carries M17 and can transcode; mrefd's sibling.

USRP — a simple UDP protocol carrying PCM audio ("8 kHz signed 16-bit"); the lingua franca of analog/VoIP bridging.

UsrpLogic — the SVXlink component that connects it to a DVSwitch Analog_Bridge over USRP.

XLX / xlxd — the multi-protocol reflector whose codebase mrefd is derived from.

Appendix A — Network Ports

mrefd's port list is short — one of the perks of a single-mode reflector with no transcoder. All ports are UDP except the dashboard's web ports. *Values are typical defaults; the M17 port in particular is configurable, and a build may differ, so verify against your own configuration.*

Service	Port	Transport	Notes
M17	17000	UDP	all M17 client and interlink traffic (configurable)
Ham-DHT	17171	UDP	discovery: publishes config + peers; required for DHT interlinking
Dashboard	80 / 443	TCP	web (and M17Web listen-in-browser, if enabled)
Not present on mrefd: any transcoder port, any DVSI/AMBE device, and the per-mode ports (D-STAR, DMR, YSF, P25, NXDN, USRP) that the multi-mode reflectors open.			

Appendix B — How to Connect and Select a Channel

Client	How to connect and pick a channel
mvoice (desktop, no dongle)	Enter the reflector's `M17-xxx` / `URF-xxx` callsign, choose a channel letter A–Z, and transmit. Codec2 runs in software.
DroidStar (desktop / mobile)	Select M17, choose the reflector host, pick the channel letter, connect.
MMDVM hotspot (Pi-Star / WPSD)	Configure the M17 mode with the reflector and channel; connect over the air from your M17 radio.
Module17 (open hardware)	The M17 Project's dedicated board; connect per its documentation, then select the channel.
Echo test	On any client, set the destination to `#PARROT` and key up to hear your own audio played back (≤ ~10 s; not for encrypted streams).
Listen only	If the reflector offers M17Web, open its page in a browser to hear live traffic with nothing installed.

Appendix C — Configuration at a Glance

Configuration files (naming reflects current source; see Chapter 24 for version differences): `mrefd.mk` holds compile-time settings; `mrefd.cfg` holds runtime parameters (identity, channels, encryption permits, DHT bootstrap, ports); `mrefd.blacklist` and `mrefd.whitelist` control callsign access (empty whitelist = open reflector; non-empty = restricted to listed callsigns); and `mrefd.interlink` defines peer links. The reflector **watches these files and hot-reloads** changes without a restart.

The three interlink forms in `mrefd.interlink`: (1) *DHT* — name the peer and channel; the address is looked up over Ham-DHT (recommended, and it enforces encryption parity); (2) *full IP and port* — for peers not on the DHT; (3) *legacy "L"* — required for older 0.x.y reflectors, without which a link may form but pass no traffic.

Build and run: `make && sudo make install`; manage with `systemctl start/stop/restart mrefd` and watch with `journalctl -u mrefd -f`. For multiple reflectors on one host, give each instance a distinct port, service file, and config path. Record the git commit you built from, since releases are loosely tagged.

Appendix D — A Timeline of M17 and the Reflector Lineage

2015–2016 — XLX (`xlxd`) created by LX3JL and LX1IQ; the multi-protocol reflector codebase mrefd descends from.
~2019 onward — the M17 Project develops a fully open digital-voice protocol built on the patent-free Codec2.
~2020 — N7TAE releases `new-xlxd`, a modernization of the XLX codebase; mrefd's M17-only development begins on this lineage (copyright 2020–2025).
2022 — N7TAE and Doug McLain (AD8DP) release `urfd` (URF), extending the codebase with M17, P25, NXDN, and USRP — mrefd's many-mode sibling.
~2017 — the D-STAR AMBE patents expire (context for the open-codec movement M17 answers).
March 2025 — the M17 Project launches M17Web, browser-based listen-only access to reflector traffic.
2020–2025 — mrefd under active development (well over four hundred commits), adding DHT-native discovery, per-channel encryption handling, and interlink refinements.

Index

Because mrefd has no fixed pagination and this guide is read as much on screen as on paper, this index points to *chapters and appendices* rather than page numbers. Bold entries are where a topic is defined or treated in depth.

Topic	Where
Bandwidth / capacity math	**Ch 13**
Blacklist / whitelist	Ch 6, **11**, 13; App C
Building / installing	**Ch 11**; App C
Callsign validation	**Ch 4**, 6, 12
Channel model	**Ch 3**, 5, 8
Codec2	Ch 2, **4**, 20; Glossary
Costs	**Ch 21**
Dashboard	**Ch 14**, 15
Decision tree (no audio)	**Ch 12**
Deployment (DNS / HTTPS)	**Ch 15**
Encryption / scrambler / AES	**Ch 7**, 8, 10, 19; Glossary
ECDSA signing	**Ch 7**; Glossary
Ham-DHT / discovery	Ch 5, 8, **9**; Glossary
Interlinking (3 forms)	**Ch 8**, 12; App C
Law / FCC Part 97	**Ch 7**
Latency	**Ch 20**, 5
Legacy “L” interlink trap	**Ch 8**, 12, 19, 22
M17 (protocol)	Ch 1, 2, **4**; Glossary
M17Web (listen-only)	**Ch 10**, 14; Glossary
mrefd vs URF	Ch 1, 16, **17**, 22
Multiple instances	**Ch 11**; App C
mvoice	**Ch 6**, 11; App B
#PARROT echo test	**Ch 10**, 11; App B
Packet vs Stream mode	**Ch 4**; Glossary
Ports	Ch 13; **App A**

Topic	Where
Reflector internals	**Ch 5**
Security / hardening	**Ch 13**, 7
No transcoding (single codec)	**Ch 1**, 5, 20
Bridging to other modes	**Ch 16**, 17
Who should run one	**Ch 21**, 17

Sources

The backbone of this guide is the primary source code and documentation of the software itself: the `mrefd` reflector, its configuration files, and its bundled dashboard (`github.com/n7tae/mrefd`); and, for the shared lineage and concepts, its siblings `urfd` (`github.com/n7tae/urfd`, `github.com/nostar/urfd`) and the ancestral `xlxd` (`github.com/LX3JL/xlxd`), along with the Ham-DHT tooling (`github.com/n7tae/ham-dht-tools`).

The M17 protocol, codec, hardware, and browser-listening material came from the M17 Project: the project site and protocol specification (`m17project.org`, `spec.m17project.org`), the M17Web announcement ("M17Web: Listen to M17 reflector traffic in your browser," March 2025), the Module17 hardware documentation, and David Rowe's Codec2 project (`rowetel.com`, `github.com/drowe67/codec2`). The Wikipedia article on M17 provided corroborating background on modulation, framing, and history.

Dashboards and live-operator corroboration came from the M17 Project's `ref-dash` (`github.com/M17-Project/ref-dash`), DU8BL's Go-based dashboard, and a range of live M17 reflector dashboards (for example those in the `openquad.net`, `xreflector.net`, and various club domains) that illustrate the dashboard layout and naming conventions described here. Client software is documented from the mvoice and DroidStar project pages.

The legal chapter draws on 47 CFR §97.113 (Cornell Legal Information Institute), the ITU Radio Regulation 25.2A text, and Bruce Perens's analysis of cryptographic signing versus encryption. The analog and VoIP bridging material (Chapter 16) drew on the DVSwitch project and wiki, the AllStarLink manual and DTMF references, the AllStarLink EchoLink channel-driver documentation, and the SVXlink project (SM0SVX) with F5VMR's and G4NAB's SVXlink-over-USRP guides. The sizing, latency, and hardening material (Chapters 13 and 20) is derived from the M17/Codec2 bitrates and mrefd's no-transcoding architecture combined with standard Linux server and networking practice; where a figure is an estimate rather than a measurement, the text says so, and Chapter 25 records what a live operator could confirm. This guide is a companion to *The URF Reflector Reference* and *The XLX Reflector Reference*, which cover the transcoding, multi-mode, and hardware topics that lie outside mrefd's single-mode scope.

This is the written edition — researched and sourced, set out as prose to be read rather than scanned. Where a claim rests on a single source or could not be independently confirmed, the text says so — and on a project moving as quickly as mrefd, verifying against your own build is always worth the few minutes.