REFERENCE GUIDE

THE XLX REFLECTOR REFERENCE

A written, sourced guide to the XLX (xlxd) multi-protocol digital-voice 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 Complete Guide to URF Reflectors*

Preface

Anyone who’s spent time with XLX 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is scattered — a piece in the source code, a piece on a forum, a piece on someone’s dashboard, and some of it out of date.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 vendor and 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 talkgroup 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 XLX (`xlxd`) multi-protocol digital-voice 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 `xlxd` and `ambed` source code and documentation read directly — and corroborated by independent operators running live XLX reflectors and by the relevant standards and reference material. It is a companion to *The Complete Guide to URF Reflectors*, and because URF is a direct descendant of XLX the two overlap by design; where XLX and URF genuinely differ, this guide says so plainly rather than assuming the reader knows the newer software. 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, patent-expiry estimates, and default ports — are marked *"as of"* where they appear. Two active codebases share the name: the classic `LX3JL/xlxd` that most reflectors run and the modernized `n7tae/new-xlxd`; this guide treats the classic build as canonical and flags the fork's differences where they matter. 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 XLX in a nutshell. An **XLX reflector** is a server running the open-source `xlxd` software that joins amateur digital-voice radios into shared rooms and, across a family of protocols, bridges between them so that users of different modes can talk to one another. It speaks the **D-STAR family** (DPlus, DExtra, DCS, and Icom's G3 terminal), **DMR**, and **Yaesu System Fusion** — three worlds, two voice codecs (AMBE for D-STAR, AMBE+2 for DMR and Fusion). It organizes traffic into up to twenty-six independent **modules** lettered A–Z, some of which can be **transcoded** so that different modes share one room, with the help of a separate transcoder program, `ambed`. Because `ambed` does all its vocoding on DVSI hardware, any module that bridges D-STAR to DMR or Fusion needs a real AMBE dongle; DMR and Fusion, which share the AMBE+2 codec, can bridge with no vocoder hardware at all.

To **stand one up**: on a Debian or Ubuntu server with a public address, install the dependencies, clone `xlxd` (and `ambed` beside it if you are transcoding D-STAR), configure it — on the classic build by editing the compiled headers, on `new-xlxd` through the interactive `rconfig` script — build and install, and copy the dashboard into your web server. To **use one without a radio**, connect with a software client like DroidStar or BlueDV. The reflector runs as an ordinary system service, and if audio ever goes silent the usual cause is a user on the wrong module or a stopped transcoder.

XLX is mature, widely deployed, and the reflector the whole D-STAR/DMR/Fusion ecosystem was built around. It is also the software that **URF** later extended: if you need M17, P25, NXDN, integrated AllStar, or software-only transcoding, URF is the newer superset and this guide will point you there where it is the better fit. Everything after this — the mechanics, the internals down to the packet and the FEC math, capacity math, operations and troubleshooting, the law, the trade-offs, the costs, and the wider ecosystem of bridges and codecs XLX lives in — is that same story told in full.

Chapter 1 — What an XLX Reflector Actually Is

To understand an XLX 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. This is how a handheld radio in California can be heard, in real time, by an operator in Germany.

An **XLX reflector** is a particular, influential kind of reflector: a *multi-protocol* one. The software that runs it, `xlxd`, calls itself “The XLX Multiprotocol Gateway Reflector Server,” and it is released under the GNU General Public License. The word that matters most is *multi-protocol*. Amateur digital voice is not one technology but several mutually incompatible ones, each with its own way of framing audio and its own voice codec. XLX’s insight, when it arrived in 2015–2016, was to speak the whole D-STAR family at once — the three older linking protocols DPlus, DExtra, and DCS — and then to reach beyond D-STAR into **DMR** and **Yaesu System Fusion**, transcoding audio between them. That made it the first widely-successful multi-protocol reflector with transcoding, and it became the backbone of a huge amount of amateur digital-voice infrastructure.

It helps to place XLX against its own successor, because a reader in 2026 will inevitably ask how the two relate. In 2022 the XLX codebase was extended into `urfd` — “A Universal Reflector with M17, a superset of XLX” — which added M17, P25, NXDN, and integrated USRP/AllStar audio and replaced XLX’s transcoder with a hybrid one that can run some codecs in software. URF is, in its own words, a *superset* of XLX: everything XLX does, URF also does, plus more. The honest framing, then, is that XLX is the mature, proven, D-STAR/DMR/Fusion reflector at the heart of the existing ecosystem, and URF is the newer, broader option built on the same ideas. This guide is candid about that throughout: where XLX is the right tool — an established D-STAR-centric or D-STAR-plus-DMR-plus-Fusion network — it says so, and where URF’s additions genuinely matter, it says that too. One boundary is worth stating at the outset because it shapes several later chapters: the two are cousins, not the same

program, and a URF reflector's own documentation puts it flatly — "you can't interlink urfd with xlxd." XLX links to other XLX reflectors; URF links to other URF reflectors.

Chapter 2 — Where XLX Came From

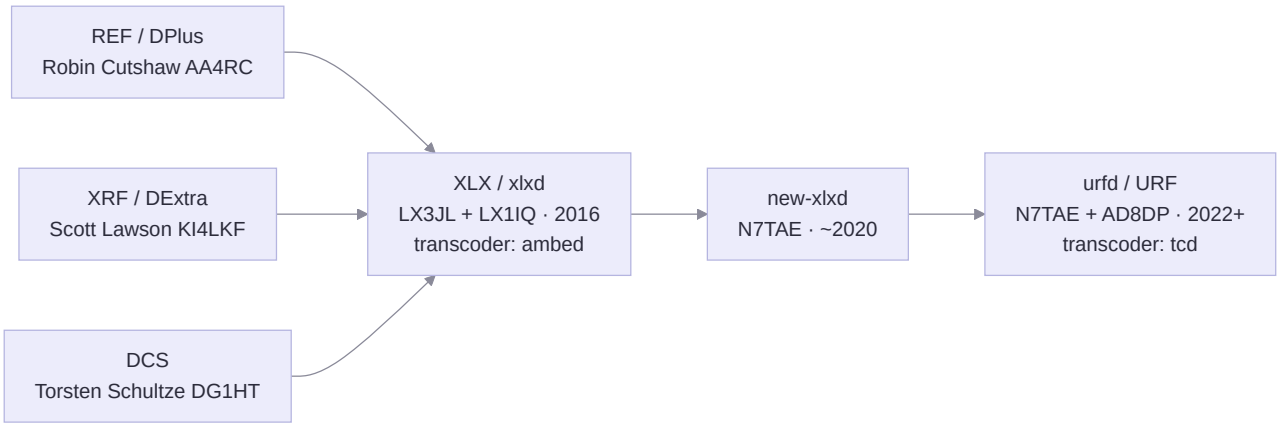
XLX did not appear out of nowhere. It is one link in a chain of reflector software that stretches back roughly twenty years, and knowing that chain explains a great deal about why XLX looks and behaves the way it does — and where it sits relative to what came after.

The story begins with D-STAR, the first mainstream amateur digital-voice mode. In D-STAR's early days three different reflector systems grew up, each speaking its own linking protocol. The first was **REF**, built on a protocol called DPlus and written by Robin Cutshaw, AA4RC; it was tied to a central registration system known as US-Trust and offered a handful of rooms lettered A through E. Then came **XRF**, using an open protocol called DExtra written by Scott Lawson, KI4LKF — notable as the first *open-source* reflector protocol, a deliberate open replacement for the proprietary DPlus. Lawson stopped active work around 2013. Alongside these grew **DCS**, a centrally-run system that originated in Germany, created by Torsten Schultze, DG1HT.

The breakthrough came in 2015 and 2016 with **XLX**, written by Jean-Luc Deltombe (LX3JL) and Luc Engelmann (LX1IQ) as part of a Radio Amateurs du Luxembourg working group. XLX's insight was to speak all three of the older D-STAR protocols — DPlus, DExtra, and DCS — at once, plus its own protocol for linking XLX servers to each other, and then to reach *beyond* D-STAR entirely into DMR and Yaesu Fusion, transcoding audio between them with a separate component called `ambed`. A reproduction of the XLX team's own timeline dates the project's conception to October 2015, its first beta to 7 November 2015, and its adoption of the GPL to January 2016, which is why the source carries a 2016 copyright.

Around 2020, Thomas A. Early (N7TAE) produced `new-xlxd`, described as "an improved XLX reflector." This was a modernization rather than an expansion: the code was rewritten to use modern C++ containers, smart pointers, and `std::future` for thread management, and to allow a clean systemd shutdown that the original design made awkward. It added no new protocols. That groundwork set the stage for the next leap. In 2022, Early — joined by Doug McLain (AD8DP) — released `urfd`, the Universal Reflector, which added M17, P25, NXDN, and USRP and replaced `ambed` with a new hybrid transcoder, `tcd`. So the lineage runs in a straight line: the three D-STAR systems fed into XLX, XLX was modernized into `new-xlxd`, and `new-xlxd` was extended into `urfd`. XLX sits squarely in the middle of that chain — the point at which reflectors stopped being D-STAR-only and became multi-protocol.

The chain, in one picture:



Chapter 3 — The Module Model

The single most important structural idea in an XLX reflector is the **module**. An XLX reflector is not one room but as many as twenty-six of them, lettered A through Z, and each module is a completely independent conference. People linked to module A hear each other; people on module B hear each other; the two do not mix unless the operator has deliberately linked them. This lets one reflector host, say, a busy daily net on one module, a quiet technical rag-chew on another, and a mode-bridging experiment on a third, all at once and without interference.

The design is flexible in ways worth knowing. A reflector can run as few as one module or as many as twenty-six. The reason an operator would deliberately not transcode every module comes down to the vocoder hardware, which is limited: a transcoded module is one where the reflector converts between codecs so that different modes can talk, while an untranscoded module simply passes audio straight through — which means clients "will only hear other clients using the same codec." Because of that, operators commonly dedicate specific modules to specific modes and tell their users exactly which module to select, and, by convention, name a module after the mode it is meant to carry. This module model is the one thing that carried unchanged all the way up the lineage: `new-xlxd` and `urfd` use the very same twenty-six-channel scheme, which is why module-thinking is common vocabulary across the whole family of software.

Chapter 4 — The Modes It Carries

A single running `xlxd` presents itself to the world across a focused set of protocols. Reading from its documentation, since version 2.5.x it supports the **D-STAR family** (the DPlus, DCS, and DExtra linking protocols, plus Icom's G3 terminal and access-point mode), the **DMR** world (the DMR+ dongle protocol and MMDVM), and **Yaesu System Fusion** (served as a YSF Master, with Wires-X and, in the classic build, IMRS inter-repeater linking). In effect, if you run D-STAR, DMR, or Fusion, an XLX reflector can talk to your radio or hotspot.

It is just as important to be clear about what XLX does **not** carry, because this is the sharpest line between XLX and its successor. XLX has no native **M17**, no **P25**, no **NXDN**, and no integrated **USRP/AllStar** audio — those are all URF additions. So where the URF guide talks about six voice modes and four codecs, XLX's world is smaller and, in one way, simpler: three modes and just two voice codecs, **AMBE** for D-STAR and **AMBE+2** for DMR and Fusion. That smaller codec set has a pleasant consequence covered in the next two chapters — DMR and Fusion, sharing AMBE+2, can be bridged with no vocoder hardware at all.

One point about the Fusion side is easy to get wrong and worth stating clearly: XLX serves Fusion as its own **YSF Master**, providing twenty-six Wires-X rooms that are self-contained. They are the reflector's own modules, not entries in the separate worldwide YSFReflector directory, so you do not register an XLX's YSF side at ysfreflector.de. (There is a subtlety here — the dashboard configuration does contain YSF/Wires-X registration fields for the hot-spot frequency database — which is taken up in Chapter 32, on disagreements.)

Before going deeper, here is the operator's-eye summary of what the reflector carries — the codec each mode uses, its default listening port, whether it can take part in a transcoded (mode-bridging) module, and where to look for the connect procedure. The codec and modulation detail is expanded in Chapter 19, the ports in Appendix A, and the connect steps in Appendix B.

Mode	Voice codec	Default UDP port	Transcode-capable?	How users connect
D-STAR	AMBE	DPlus 20001, DExtra 30001, DCS 30051, G3 40000	Yes — needs DVSI hardware to bridge to DMR/Fusion	URCALL linking, G3 terminal, or DTMF (Ch 8)
DMR	AMBE+2	DMR+ DMO 8880, MMDVM 62030	Yes — no hardware needed to bridge to Fusion	Private-call sequence via DMRGateway (Ch 8)
Fusion (YSF)	AMBE+2	42000	Yes — no hardware needed to bridge to DMR	YSF Master → Wires-X room or DG-ID (Ch 8)
Not carried by XLX (URF adds these): M17 · P25 · NXDN · USRP/AllStar				

Chapter 5 — Transcoding: The Heart of the Whole Thing

Everything interesting about a multi-mode reflector comes down to one hard problem, and it is worth slowing down to understand it, because it explains the hardware, the software architecture, and most of the operational headaches that follow.

Each digital-voice mode encodes speech with a **vocoder** — the algorithm that turns a human voice into a low-bitrate stream of data and back again. D-STAR uses a proprietary codec called AMBE. DMR and Fusion use a related proprietary codec, AMBE+2. The consequence is simple and, in one direction, brutal: a D-STAR radio and a DMR radio produce completely different, mutually unintelligible data for the same spoken words. Put both users on the same reflector module and, unless something actively converts between their codecs in real time,

they will sit in silence. The documentation states the general rule plainly: without transcoding, clients “will only hear other clients using the same codec.”

XLX has a smaller codec world than its successor, and that produces a distinction worth internalizing before anything else. There are only two codecs in play, and one of them — AMBE+2 — is shared by *both* DMR and Fusion. So bridging DMR and Fusion is not really transcoding at all in the expensive sense: the audio is the same codec on both sides and only needs reframing, which XLX can do with **no vocoder hardware**. As one transcoding how-to puts it, “C4FM and DMR do not require any transcoding hardware (AMBE) to work together.” The moment D-STAR enters the picture, though, you are converting between AMBE and AMBE+2 — two genuinely different codecs — and that requires real DVSI hardware.

The thing that does the hardware conversion is a separate program from the reflector itself, called `ambed`, the transcoder. Unlike URF’s hybrid `tcd` — which can run several codecs in software — `ambed` is an **all-hardware** transcoder: every AMBE and AMBE+2 vocoding operation runs on a DVSI chip. That single fact shapes XLX operations more than any other. It means a transcoding XLX reflector that carries D-STAR *always* needs a DVSI dongle; there is no software-only path of the kind URF later introduced. The reflector and `ambed` talk to each other over **UDP** — a controller channel on port 10100 and per-stream channels on 10101–10199 — and, because that link is UDP, the transcoder can sit on the same machine as the reflector or on a completely different one on the network. The internals of how a frame makes that round trip are described in Chapter 9.

Chapter 6 — The Vocoder Hardware

Because AMBE and AMBE+2 cannot legally or practically be done in open software on the XLX/ambed stack, anyone running a transcoding XLX reflector that carries D-STAR needs real DVSI hardware, and it helps to understand what that hardware is. DVSI’s AMBE chips come in fixed channel counts, and those counts set hard limits on how many streams you can transcode at once. The single-channel chip is the AMBE-3000, which you will find inside devices sold as the DVMEGA DVstick-30 and the NW Digital Radio ThumbDV (around one hundred US dollars, *as of mid-2026*). The three-channel chip is the AMBE-3003, sold as the DVstick-33 among others. There are larger devices too — a six-channel USB-3006 and a twelve-channel USB-3012 that packs three AMBE-3003 chips onto one interface.

The counting is where XLX differs subtly from URF, and it follows directly from `ambed` being all-hardware. Bridging D-STAR to DMR or Fusion means running *two* vocoders at once — one to decode AMBE and one to encode AMBE+2 (and vice versa) — so, as one operator’s build notes put it, you need “a minimum of 2” AMBE channels for D-Star transcoding with the other modes. That is why the classic all-hardware code yields fewer simultaneous transcoded modules from a given device than URF’s hybrid transcoder does: URF can push AMBE+2 into software and reserve the chip for D-STAR’s AMBE, but `ambed` must spend hardware channels on both sides of every D-STAR bridge. The practical upshot is that a single three-channel AMBE-3003 is the usual choice for a transcoding XLX reflector, and DMR ↔ Fusion modules cost no channels at all.

There is one hardware quirk that trips up nearly every first-time builder, and it is worth flagging loudly. The DVSI USB devices use FTDI chips, and they need FTDI’s own D2XX driver — but that driver will only work if the normal Linux kernel `ftdi_sio` and `usbserial` drivers are removed first. If those standard kernel modules

grab the device, the transcoder can’t. On a properly installed system the service startup handles this; the trap is only sprung if you try to run the transcoder by hand, in which case you have to unbind the kernel drivers yourself.

Chapter 7 — Inside the Reflector

It is worth opening the reflector up and seeing how it actually moves a voice stream, because the design is elegant and it makes the later operational advice make sense. What follows was read from the `xlxd` source and its modernized `new-xlxd` fork.

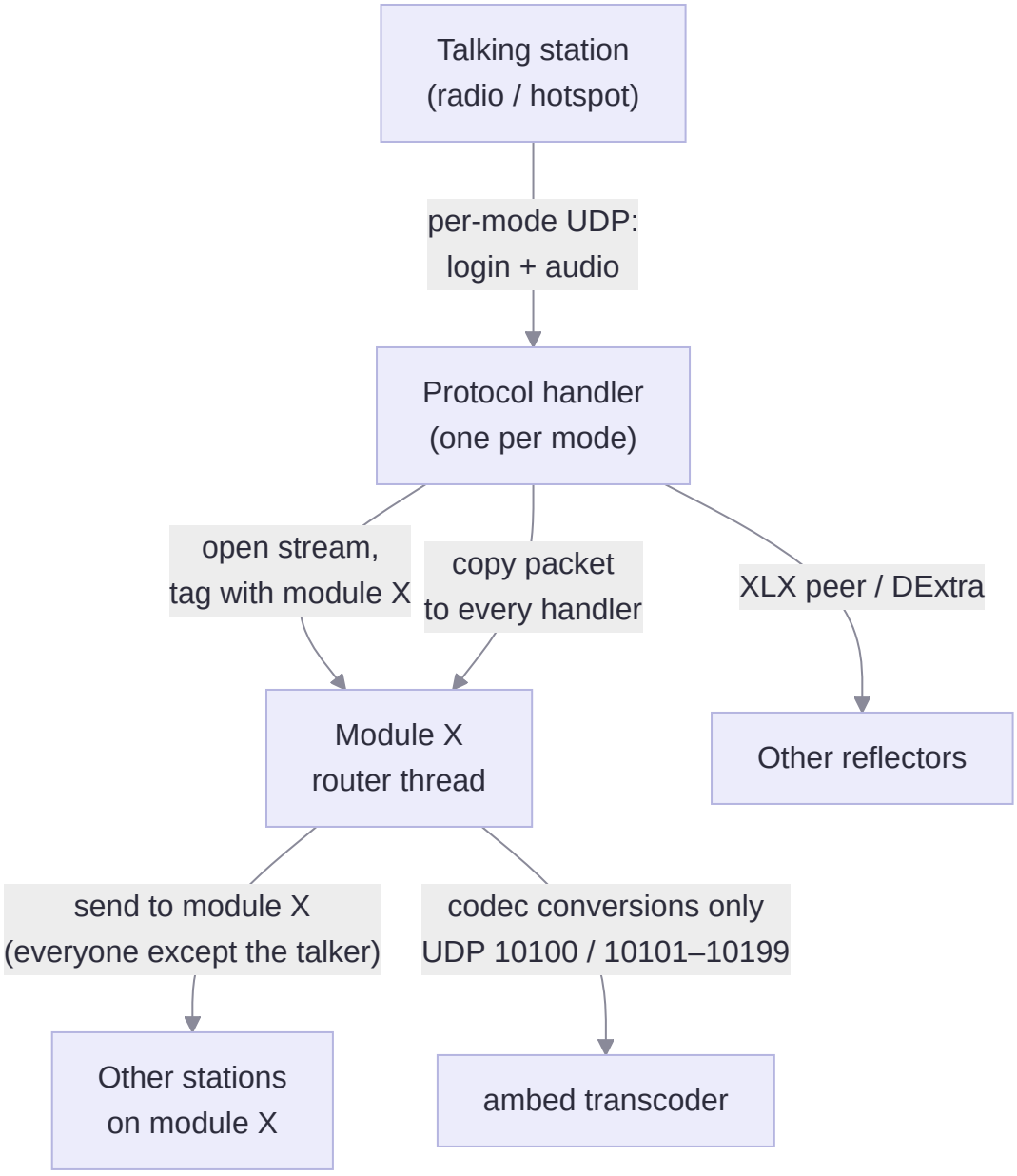
At the center sits a single reflector object that owns everything: a list of recently-heard users, a list of connected clients, a list of linked peer reflectors, and a set of protocol handlers — one for each mode it speaks. It also owns, for each module, a stream object and a dedicated **router thread**. Each protocol handler runs its own loop, listening on its own UDP sockets for its own mode’s traffic.

When someone keys up, the sequence is precise. The relevant protocol handler receives a voice header, validates that it came from a known client who isn’t already transmitting, and opens a stream. Loop prevention rejects any stream that duplicates one already open, which is how the reflector avoids echoing traffic in circles. It tags the packet with the module the client is linked to and marks that client as the current “master” of the module — the one whose audio everyone else will hear — matching subsequent frames by stream and sender so stray packets are discarded, and closing the stream on a short timeout when the audio stops.

The actual fan-out — getting the audio to everyone else — is the job of the per-module router thread. It copies each packet into every protocol handler’s outbound queue, adjusting the destination callsign for each protocol, and here is the universal rule that makes a reflector a reflector: it sends to everyone on the module *except* the station currently transmitting. Transcoding is woven in with care: only frames that need a codec conversion are diverted to `ambed`, and a frame that arrived already in the right codec is left alone.

One behavior from the internals matters because it echoes across the whole ecosystem: the reflector writes an XML **status file** every few seconds describing its peers, linked nodes, and recently-heard users. This is the file the dashboard reads, and it is the *original* of the format that later software imitates — when a URF reflector “reports itself as an XLX reflector,” the shape it is copying is precisely this XLX status file. In XLX it is simply the reflector describing itself, in its own name.

In block-diagram form, a single voice stream moves through the reflector like this:



Chapter 8 — How Each Mode Connects, and the Security Behind It

Every mode has its own way of logging in and its own way of choosing a module, and while the full protocol-by-protocol detail is more than most readers need, the shape of it is genuinely useful — especially the parts an operator has to explain to their users. XLX is where most of these conventions were established, so this is the canonical description; the URF versions are inherited from here.

D-STAR selects a module the old-fashioned way, by putting a linking command in the radio's URCALL field: the reflector's callsign, then the module letter, then an "L" to link or "U" to unlink, all in the eighth character position

— so `XLX307CL` links to module C on reflector 307. Many radios and hotspots can send the same command as **DTMF** tones instead, which is easier on a handheld; the exact DTMF digits are set by the gateway or hotspot software rather than by the reflector, so you follow your hotspot's table. For Icom's G3 terminal and access-point mode you register on a G3 gateway, assign a module letter, point Icom's RS-MS3A/W software at the reflector, and open UDP port 40000 if it isn't forwarded.

DMR is the most involved, because DMR has no natural concept of "pick a room." Connecting through the standard DMRGateway software (which is what Pi-Star and WPSD use), a user selects the reflector and module with private calls and then talks on a talkgroup. The convention is captured in the table below; under the hood the reflector maps DMR talkgroups 4001–4026 to modules A–Z, with 4000 meaning "unlink." There is a notorious trap here that catches everyone at least once: the `XLXHosts.txt` file that ships with DMRGateway defaults to Module D, which is often not the module the transcoder is active on, so a user who doesn't override it lands on the wrong, silent module and assumes the reflector is broken.

Action	What you send	Example (module E, reflector 307)
Connect to a reflector	Private call to `68` + reflector number	`68307`
Select a module	Private call to `64000` + module number (A=1 … Z=26)	`64005`
Talk	Transmit on talkgroup `6`	TG `6`
Disconnect	Private call to `64000`	`64000`
Query status	Private call to `65000`	`65000`
Module via talkgroup (DMR+)	TG `4001` – `4026` = module A–Z; TG `4000` = unlink	TG `4005` = module E

Fusion connects your hotspot to the XLX YSF Master, after which you choose a module either with the radio's Wires-X button (the twenty-six rooms) or by DG-ID, where the reflector maps DG-IDs to modules — though the exact mapping is set per reflector, so you read each reflector's own instructions. DMR IDs are resolved for display by downloading `dmrid.dat` from the XLXAPI server at `xlxapi.rlx.lu`.

Behind all of this sits a security model that is simpler than people assume. The reflector authorizes callsigns against a **blacklist** and a **whitelist**, read from files that support a limited wildcard. An empty whitelist lets everyone through; a non-empty one requires a match; and in either case a callsign that matches the blacklist is refused. The default files ship configured for a "totally open network." Peer reflectors are authorized more strictly through the interlink map (Chapter 10). A whitelist, then, is the tool that turns a public reflector into a private, invitation-only one.

Chapter 9 — Inside the Transcoder

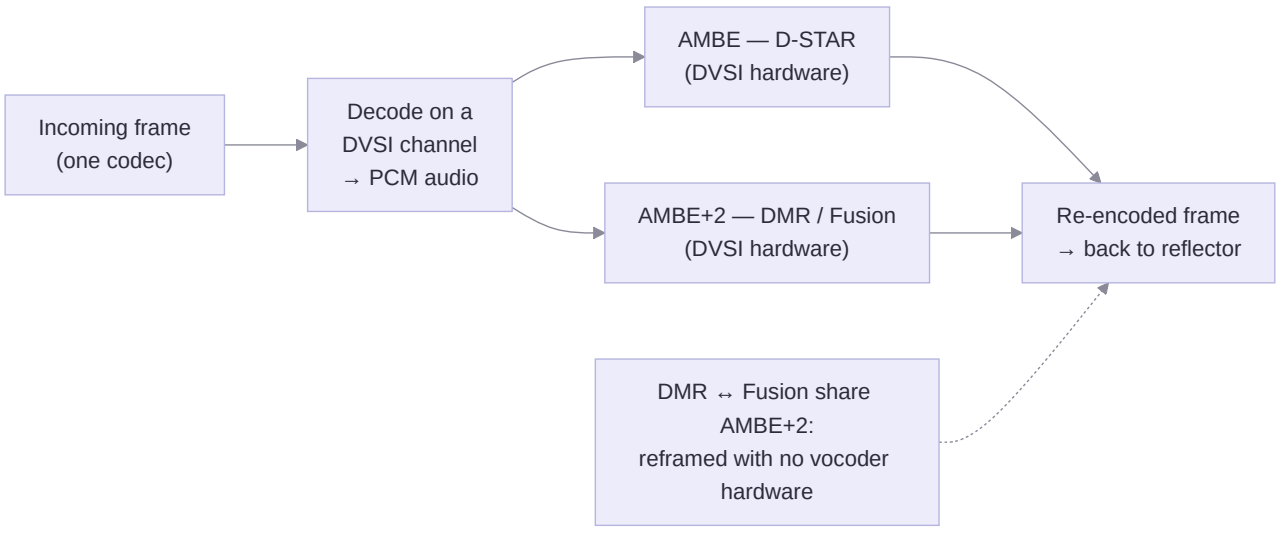
The transcoder deserves its own look, because it is where the real signal-processing happens and where the hardware quirks live.

`ambed` is a separate daemon that the reflector talks to over UDP: a controller channel on port 10100 and a pool of per-stream channels on 10101–10199. When a module needs a codec conversion, the reflector hands the frame to `ambed`, which decodes it on a DVSI channel to plain PCM audio and re-encodes it on another DVSI channel into the target codec, then returns it. Because every vocoding step is done on hardware, `ambed` must discover and open the DVSI devices at startup — through the FTDI driver, by serial number, at the correct baud rate — and configure each vocoder channel with the DVSI parameter words for the codec it will run. If it cannot find the hardware it expects, it will not transcode, and the symptom on the air is the classic one: callsigns appear but no audio is heard.

Two design choices are worth contrasting with URF, because they explain XLX's operational character. First, the link is **UDP** rather than TCP. That keeps the transcoder loosely coupled — it can run remotely, and either side can restart without a formal connection teardown — but it also means the reflector and transcoder rely on their own keepalives and retries rather than a guaranteed ordered channel. Second, and most important, `ambed` is **all-hardware**: there is no build option to run AMBE+2 in software, as URF's `nostar/tcd` offers. The practical rule that falls out of both chapters is therefore blunt — if your XLX reflector bridges D-STAR to anything, you own a DVSI dongle; if it only bridges DMR and Fusion, which share AMBE+2, you may not need one at all.

There is a small but important configuration coupling to remember: the set of modules the reflector marks as transcoded and the channels `ambed` has available must line up. If you ask the reflector to transcode more modules than the hardware can serve, some modules simply won't bridge.

The pipeline inside the transcoder, where a D-STAR frame becomes an AMBE+2 one and back:



Chapter 10 — Interlinking and the Wider Network

Reflectors gain their reach by linking to each other, and XLX does this through a file called `xlxd.interlink`. The native mechanism is the **XLX peer protocol** (UDP port 10002 in the modernized build), which links two XLX reflectors and shares a whole list of modules between them: each line of the interlink file names another reflector, its address, and the modules to share. Interlinks are mutual — both reflectors must list each other, or the link never forms — and linking is module-to-module of the same letter, so module A links to module A, never A to B.

There is a second, narrower mechanism worth knowing: a single-module **DExtra peer link**, which links one local module to one remote module over the older DExtra protocol. It is more limited — the documentation notes that "only one DExtra peer link per node is supported" — but it is how an XLX reflector reaches certain D-STAR endpoints that don't speak the XLX peer protocol. Between them, these two forms are the whole of XLX's reflector-to-reflector story, and they are exactly what the later software inherited.

The hard boundary from Chapter 1 holds firm here and is worth repeating from the XLX side: an XLX reflector interlinks with other **XLX** reflectors (and, through the same file, with DMR networks such as Brandmeister), but it cannot interlink with a `urfd` reflector — the two speak related but not identical peer protocols. If you need to join an XLX network to a URF one, you bridge them externally with the toolkit in Chapter 20 rather than interlinking them directly.

Chapter 11 — XLX vs XRF: The Two Build Variants

One feature of the XLX source surprises newcomers: the same code builds two different kinds of reflector, and choosing between them is one of the first decisions an operator makes. The build is either a full **XLX** reflector or a lighter **XRF** reflector, and they are meaningfully different animals.

A full **XLX** reflector is the multi-protocol server this guide is mostly about: it speaks the D-STAR family, DMR, and Fusion; it can carry a transcoder; and it is eligible to "call home" to the central registry so it appears in the host files hotspots download. This is what you build if you want mode-bridging or a DMR/Fusion presence.

An **XRF** reflector is D-STAR only. It accepts inbound DExtra, DPlus, and DCS connections and can make an outbound DExtra peer link, but it carries no DMR or Fusion, no transcoder, and — this is stated emphatically in the documentation — you must **not** enable the calling-home feature on an XRF build, because that feature is for XLX systems only. An XRF reflector is the right choice for a purely D-STAR community that wants a clean, lightweight reflector with no vocoder hardware and no multi-mode complexity. It is, in spirit, a modern continuation of the original XRF reflectors from the DExtra era.

The decision, then, is straightforward. If your network is D-STAR only and you have no intention of bridging modes, an XRF build is simpler, needs no dongle, and stays out of the multi-mode registry entirely. If you want DMR or Fusion, or any mode-bridging, you build XLX. And if you later find you also want M17, P25, NXDN, or AllStar, that is the point at which the newer URF software — not either XLX variant — becomes the answer (Chapter 18).

Chapter 12 — How a Reflector Is Found and Registered

For radios and hotspots to connect to a reflector, its existence and address have to be published somewhere. XLX predates the distributed-directory approach that later software adopted, so its discovery model is the older, central-registry one — and since XLX is where that model was built, this is its home ground.

The core mechanism is **"calling home."** The dashboard can be configured to periodically report the reflector to a central XLX API server at `xlxapi.rlx.lu`, which keeps a live list of who is up and generates the host files that hotspots download. The same API server is where the reflector fetches `dmrid.dat` to turn DMR IDs into callsigns for display. Layered on top is **DVRef**, a web registry with user accounts where an operator registers a reflector and is assigned a five-digit number, and the **host files** that ultimately populate the reflector dropdown menus in Pi-Star and WPSD, which are mirrored and refreshed regularly.

The warning that comes with calling home is important enough to repeat, because getting it wrong steps on someone else's reflector. XLX, XRF, and URF callsign suffixes all share a single numbering space — **XLX307, XRF307, and URF307 are the same "307."** So you must not enable calling home unless you are sure you will not be infringing on an existing reflector with the same suffix. Pick an unused number and, ideally, register it before you turn calling home on. And, as Chapter 11 noted, an XRF build must never call home at all.

One honest contrast with the newer software: XLX has **no Ham-DHT** presence. The decentralized, serverless directory that URF reflectors publish into is a later addition; XLX discovery depends on the central XLXAPI, DVRef, and the host-file mirrors. In practice this works well and has for years, but it is a genuine architectural difference — XLX leans on central services where URF can be found peer-to-peer.

Chapter 13 — Connecting Without a Radio

You do not actually need a radio or a hotspot to use an XLX reflector; several software clients let a computer or phone connect directly, which is invaluable for testing your own reflector (a per-mode test procedure is in Chapter 14). The most versatile is **DroidStar**, which connects on the D-STAR, DMR, and Fusion sides — exactly the modes XLX carries — and runs on Linux, Windows, macOS, Android, and iOS; you pick the mode, choose the `XLXnnn` host, and select the module. PA7LIM's long-standing **BlueDV** also connects on D-STAR, DMR, and Fusion, though for the vocoding it needs a DVSI dongle on the client machine. For over-the-air testing, any standard MMDVM hotspot running Pi-Star or WPSD will reach an XLX reflector in all three modes.

One difference from the URF toolkit is simply the absence of the M17-only clients: there is no mvoice or Module17 in an XLX workflow, because XLX carries no M17. The client list is the D-STAR/DMR/Fusion set, which is well-established and mature.

Chapter 14 — Building and Operating One

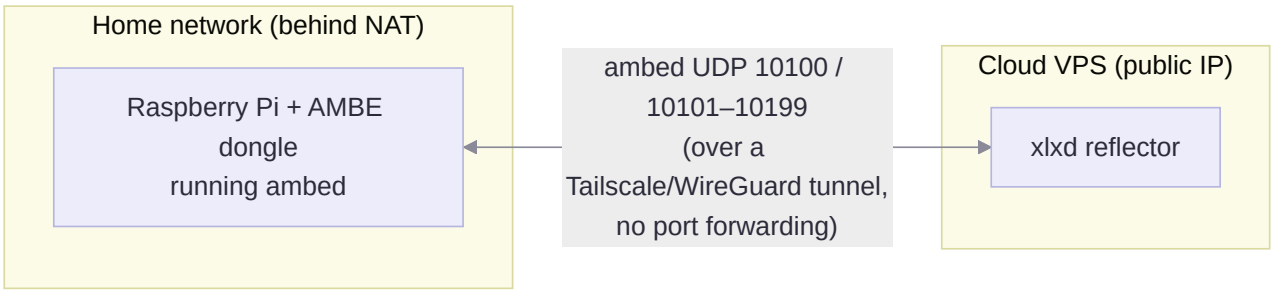
Bringing an XLX reflector up follows a well-worn path. The software runs on systemd-based Linux, with Debian or Ubuntu recommended, and needs a machine with a permanent internet connection and a public address. You install the dependencies (git, Apache, PHP, the build tools), and clone the `xlxd` repository — and, if you intend to transcode D-STAR, the `ambed` transcoder alongside it.

Configuration is where the two codebases differ most, and it is the single biggest day-to-day contrast with URF. XLX is configured at **build time**, not through a runtime `.ini` file. On the classic `LX3JL/xlxd` build you edit a header — historically `main.h` — to set things like the YSF hot-spot frequency and other compile-time options, then build; the reflector's identity and much of its behavior are baked into the compiled binary, and changing them means editing and recompiling. On the modernized `n7tae/new-xlxd` build this is smoothed over by an interactive script, `rconfig`, which asks you the questions — callsign, IPv4/IPv6 binding, transcoder port, G3 support — and *generates* the C++ headers and makefile fragments for you, so you never hand-edit the source. Either way, the separate dashboard configuration file, `config.inc.php`, holds the operator-facing settings: email, country, comment, the module names and ports the dashboard displays, and the calling-home toggle.

Installation and day-to-day management on `new-xlxd` go through the `radmin` menu — clean, compile, install/uninstall, and follow the reflector or transcoder logs behind single keystrokes — with the services running under systemd. The classic build uses an `/etc/init.d/xlxd` startup script instead. Updating is a `git pull` followed by a rebuild; because your configuration lives in generated headers and the dashboard file rather than scattered edits, a rebuild does not clobber your settings. Because XLX has no tagged releases, record the **git commit** you built from — it is the only reliable way to answer "which version am I running."

The remote transcoder

An XLX deployment is often not a single machine. The reflector commonly lives on a public cloud server, while the DVSI dongle and `ambed` live at the operator's home. Because the reflector-to-transcoder link is UDP on well-known ports, you can run `ambed` on a small machine at home with the dongle plugged in and point it at the cloud reflector — keeping the expensive, physically-present hardware where you can reach it. A private mesh VPN such as Tailscale (WireGuard-based, no port-forwarding) is the tidy way to carry that UDP link across a home NAT without exposing anything to the public internet; the reflector knows nothing about the VPN, but its clean split between a public server and a remote transcoder makes one a natural fit.



Verifying it works: a per-mode test procedure

A reflector that compiles and starts is not yet a reflector that works — the failure that matters most, callsigns-appear-but-no-audio, only shows up when real traffic flows. Before you announce a new reflector, walk this checklist using the software clients from Chapter 13; the only step that needs a dongle is the D-STAR transcode.

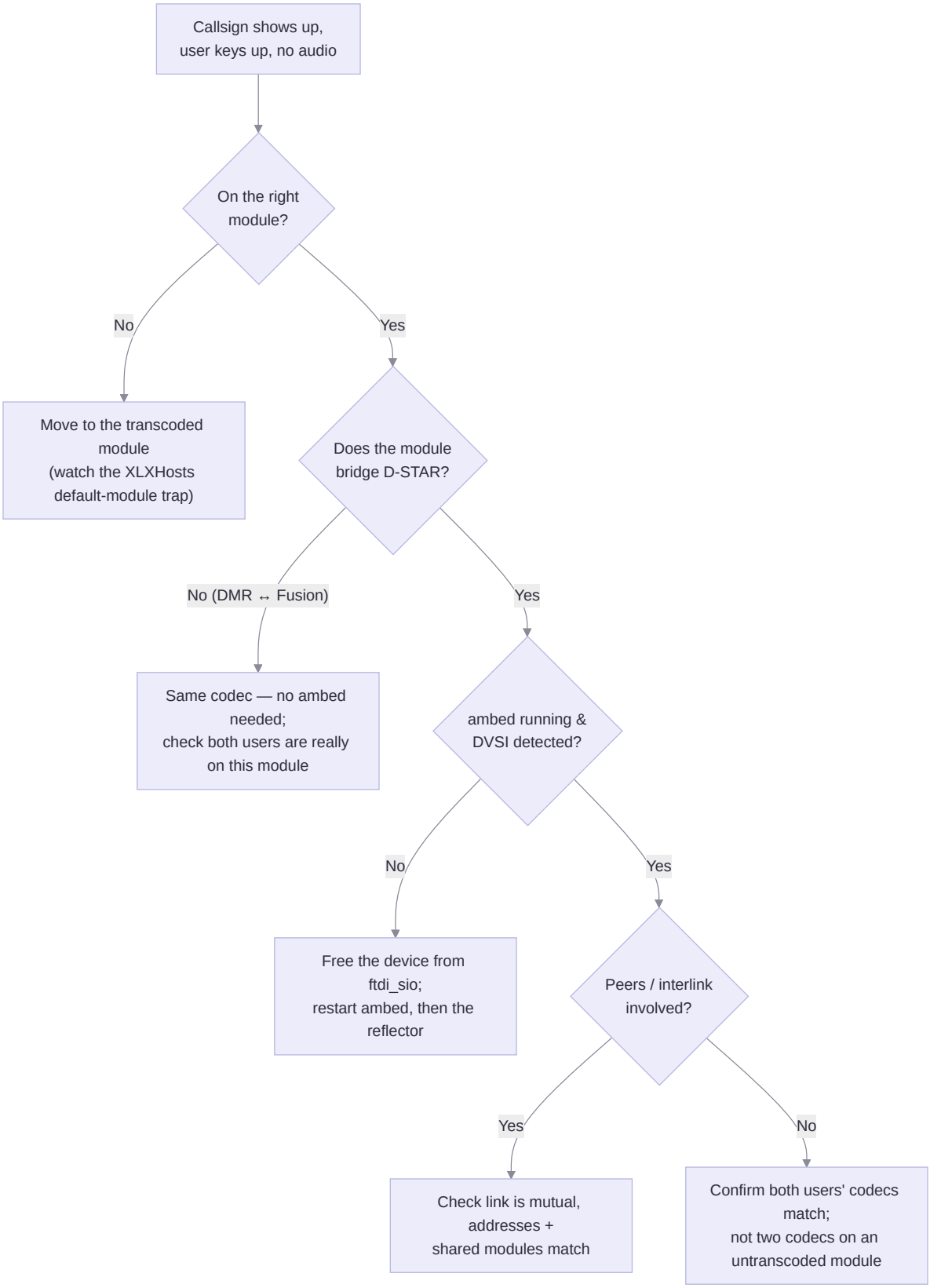
#	Check	How	Pass criteria
1	Services up	Confirm `xlxd` (and `ambed`, if transcoding) are running; watch the logs via `radmin`	Both active; `ambed` reports its DVSI channels
2	Single-mode audio	Two DroidStar instances on the same untranscoded module, same mode	Each hears the other; dashboard shows both
3	DMR ↔ Fusion bridge	DroidStar in DMR + a client in Fusion on a shared module (no dongle needed)	Each hears the other — proves AMBE+2 reframing
4	D-STAR transcode	D-STAR client + a DMR/Fusion client on the transcoded module; key up each side	Both directions heard — proves `ambed` + DVSI
5	Dashboard truth	Compare dashboard “Last Heard” to who actually keyed up	Callsigns, module, and timestamps correct
6	Discovery	Confirm DNS resolves and the reflector appears in the host files	A hotspot elsewhere can find and link
7	Failure drill	Stop `ambed`, key up across the transcoded module, then restart it	You reproduce the silent-audio symptom and recover — so you recognize it live

Chapter 15 — When Things Go Wrong

The failure modes of an XLX reflector are, thankfully, a short and knowable list, and almost all of them present as the same symptom: someone connects, their callsign shows up, they key up — and no one hears them. The trick is knowing the handful of causes.

The most serious is **transcoder failure**. If `ambed` has died, or cannot find its DVSI hardware, a module that is supposed to bridge D-STAR to DMR or Fusion will forward the signalling but not the audio — the connection works, the callsign appears, but there is silence. The fix is to confirm the DVSI device is present (the usual culprit is the `ftdi_sio` kernel driver holding the device) and restart the transcoder. Closely related is the simpler case of a **user on the wrong module**: two users on an untranscoded module using different codecs, or a user who landed on a module that isn’t the transcoded one, will hear nothing. This is most often the “XLXHosts default-module trap” from Chapter 8 — the host file quietly points at Module D — and the fix is to select the correct module explicitly. Peers that refuse to link are almost always a case of the link not being mutual, or a module or address mismatch. And a reflector no one can find is a discovery problem — check that its DNS name is published and that it appears in the host files.

The same logic as a decision tree, for the silent-audio symptom:



Chapter 16 — Performance, Sizing, and Hardening

There are no published, quantified performance figures for `xlxd`; what the documentation says is that a reflector needs "a 24/7 internet connection which can support 20 voice streams or more." That phrasing is a useful anchor, and the rest can be estimated from first principles.

Estimating bandwidth and CPU from first principles

A digital-voice stream on the wire is small. What actually leaves the server is the mode's network frame plus IP/UDP overhead — budget on the order of **~25 kbit/s per active stream per direction**. The key multiplier is that a reflector *fans out*: one talker keying a module with *N* listeners sends the audio out *N* times, and only one person talks at a time per module. So a busy module with 30 listeners costs roughly 30 × 25 kbit/s ≈ **0.75 Mbit/s** while someone is talking; even a reflector with several active modules and a few hundred clients lands in the low single-digit megabits per second, which is why a modest VPS is never the bottleneck. The documentation's "20 voice streams" guidance sits comfortably inside that. CPU on the reflector side is light integer work; the real cost is transcoding, and on XLX that cost is *offloaded to the DVSI chip* rather than the CPU — a genuine advantage of the all-hardware design, since the host barely notices even a busy transcoded module. The bounded resource is not CPU or bandwidth but DVSI channels (Chapter 6).

Security, hardening, and hygiene

XLX's built-in security is the callsign blacklist and whitelist described in Chapter 8, and nothing more — everything else is ordinary Linux server hygiene and falls to the operator. Run a firewall that denies by default and opens only the specific protocol ports actually in use (Appendix A), use key-only SSH, and add something like fail2ban for SSH and the dashboard. One caveat deserves emphasis: fail2ban does nothing for the open UDP reflector ports, which — because UDP has no handshake — are inherently exposed to spoofed-source floods; the URF-style mitigations apply equally here (rate-limit per-source at the firewall with `nftables`, don't open ports for modes you don't run, and lean on upstream protection for a real volumetric flood). Keep the clock disciplined with NTP so the dashboard timestamps are right, cap the logs so they don't fill a small VPS, and mirror your IPv4 firewall rules on IPv6 if you run dual-stack. For monitoring, the cleanest health signal is the status file: a small script that alerts if it goes stale or if `ambed` has dropped catches the silent-transcoder failure before your users do (Chapter 21 shows how to parse it). Back up the small set of files you edited — the generated config or `main.h`, the interlink, blacklist and whitelist, and `config.inc.php` — and record the git commit you built from.

Chapter 17 — The Law (United States)

Running a reflector puts you squarely inside the amateur regulations, and in the United States those are FCC Part 97. This is a factual summary, not legal advice, and operators elsewhere are under their own countries' rules. Two provisions matter most. The first is station identification, under §97.119, which requires each station to transmit its assigned callsign "at the end of each communication, and at least every 10 minutes during a communication." In the reflector context, this duty falls on each transmitting radio station — the hotspot, repeater, or user's radio

— rather than on the reflector as an internet relay. The second is the prohibition on encryption under §97.113(a)(4), which bars “messages encoded for the purpose of obscuring their meaning.” The FCC has affirmed this prohibition, dismissing a petition that sought to relax it.

For an XLX operator the encryption rule is refreshingly simple to comply with, because none of the modes XLX carries offers user encryption in normal amateur use: D-STAR, DMR, and Fusion are all transmitted in the clear on the air. The lively debate over amateur encryption — M17’s AES option versus its ECDSA signing, and where each falls under the rule — belongs to M17, which XLX does not carry; a reader interested in that distinction will find it in the URF guide. On XLX, the practical takeaway is just the ordinary one: identify properly, and don’t attempt to obscure the meaning of a transmission.

A closing note on jurisdiction: everything above is United States law. The prohibition on obscuring-encryption is in fact a treaty-level principle — ITU Radio Regulation 25.2A provides that transmissions between amateur stations of different countries “shall not be encoded for the purpose of obscuring their meaning” — and national licences transpose it, so the identification duty and the encryption ban hold in spirit almost everywhere, but the exact wording and intervals belong to your own national regulator.

Chapter 18 — Choosing What to Run

A few decisions recur for anyone standing up a reflector, and the research points to reasonably clear answers.

Classic `xlxd` or `new-xlxd`? The classic `LX3JL/xlxd` is the original and still the most widely deployed; it is battle-tested and the thing most existing documentation describes. The `n7tae/new-xlxd` fork is a modernization — cleaner C++, smart pointers, a graceful systemd shutdown, dual-stack IPv4/IPv6, and the friendlier `rconfig` / `radmin` workflow instead of hand-edited headers. For a brand-new build, `new-xlxd` is the more comfortable starting point; for an established reflector that already works, there is little reason to switch. Either way the on-air behavior is the same.

XLX or XRF? Covered in Chapter 11: XRF for a pure-D-STAR community that wants a light reflector and no dongle, XLX for anything involving DMR, Fusion, or mode-bridging.

XLX or URF? This is the decision most worth being candid about. Choose **XLX** when your world is D-STAR, DMR, and Fusion — an established D-STAR-centric network, a club that lives in those three modes, or an operator who values a mature, widely-understood, heavily-documented platform. Choose **URF** when you want any of what XLX cannot do: native **M17**; integrated **P25** or **NXDN**; integrated **USRP/AllStar** without an external bridge; **software-only transcoding** so you can bridge DMR/Fusion/M17/P25 without buying a DVSI dongle (D-STAR still needs the chip even on URF); or decentralized **Ham-DHT** discovery. Since URF is a genuine superset of XLX, a URF reflector can also serve every XLX mode — so for a from-scratch multi-mode build in 2026, URF is often the more future-proof choice, and this guide says so plainly. XLX’s enduring strengths are its maturity, its huge installed base, and its fit for networks that are D-STAR-centric and have no M17/P25/NXDN ambitions.

Hardware? A single three-channel AMBE-3003 is the usual choice for a transcoding XLX reflector; a single-channel 3000 suffices only for the lightest D-STAR bridging. And remember the XLX-specific saving:

DMR ↔ Fusion needs no dongle at all.

Chapter 19 — The Modes and Their Codecs, Compared

The guide has explained each mode where it came up, but a reference ought to lay them side by side, because the differences between them are exactly what makes transcoding necessary. XLX carries three modes across two codecs — a tighter picture than URF's six-and-four.

Mode	Modulation / bandwidth	Voice codec (bitrate)	Access code	Station ID	Standard
D-STAR	GMSK, ~6.25 kHz (4800 bit/s on air)	AMBE, 3600 bit/s (2400 voice + 1200 FEC)	none (callsign routing)	callsign + US-Trust registration	JARL — open protocol, closed codec
DMR	4FSK, 12.5 kHz two-slot TDMA (9600 bit/s)	AMBE+2, ~2450 bit/s voice	Color Code (0–15)	DMR ID via RadioID	ETSI — open standard, closed codec
Fusion (C4FM)	C4FM/4FSK, 12.5 kHz (9600 bit/s)	AMBE+2	DG-ID (00–99)	callsign (no registry)	Yaesu — open spec, closed codec

The striking thing is how much the modes share at the radio layer and how they split at the codec layer — the layer that actually matters for interoperability. D-STAR stands alone on AMBE; DMR and Fusion both use AMBE+2. That single fact is the whole shape of transcoding on XLX: bridging DMR and Fusion is cheap (same codec, reframed in software), while bridging either of them to D-STAR is the expensive case that needs the DVSI chip.

Codec	Owner	Used by	Bitrate	Patent status
AMBE	DVSI	D-STAR	2–9.6 kbit/s (3600 for D-STAR)	1997 patent **expired** (~2017 for the D-STAR variant)
AMBE+2	DVSI	DMR, Fusion	~3600 bit/s (2450 voice)	still under DVSI license (a remaining patent is estimated to run to ~2028 — *as of mid-2026*)

The FEC math, briefly

D-STAR's AMBE bitrate in the first table is split into "voice + FEC," and it is worth a paragraph on what that means. **Forward error correction** adds redundant bits so a receiver can *correct* errors without asking for a retransmission — essential on a radio link where there is no time to ask again. A **Hamming code** adds parity bits so a single-bit error can be located and flipped back; a **Golay code** is stronger — the binary Golay(23,12), often extended to (24,12), encodes 12 data bits into 23 or 24 and can correct up to three bit-errors in a block, which is

why it guards the most important bits of a voice frame. The arithmetic falls out of the table: D-STAR's AMBE frame is 3,600 bit/s total — 2,400 bit/s of codec payload wrapped in 1,200 bit/s of FEC and framing, a 2:1 protection ratio — so roughly a third of what a D-STAR radio transmits is not voice but the redundancy that keeps it intelligible right up to the "digital cliff." FEC protects a codec's bits in transit; it does nothing to bridge two *different* codecs, which is why a transcoder still has to decode all the way to PCM audio and re-encode — the quality cost of which is Chapter 28.

Chapter 20 — Bridging Beyond XLX: DVSwitch and the Wider Network

An XLX reflector's transcoder solves one problem beautifully — letting users of D-STAR, DMR, and Fusion talk to each other within one reflector — but it does not connect that reflector to the rest of the digital-voice universe. It cannot follow an arbitrary Brandmeister talkgroup beyond what the interlink file allows, and it has **no integrated USRP/AllStar** at all — that integration is a URF feature, not an XLX one. For everything outside XLX's own world, operators reach for a separate toolkit called **DVSwitch**, and it matters more here than in the URF context precisely because XLX lacks the built-in USRP bridge.

DVSwitch is a suite of small programs that plumb one network to another. The two central pieces are **MMDVM_Bridge** and **Analog_Bridge**. MMDVM_Bridge logs into a digital network — a Brandmeister or TGIF talkgroup, an XLX reflector, a YSF room — exactly as a hotspot would, and decomposes the stream into a mode-neutral form called TLV. Analog_Bridge is the codec engine and the analog leg: it transcodes that stream to and from plain PCM and moves the PCM over the simple UDP protocol called **USRP** — the same USRP that AllStar speaks. Analog_Bridge does its vocoding with an AMBE dongle, a networked dongle via PA7LIM's AMBEServer, or a software vocoder.

The practical division of labour is clean. Use XLX's own transcoder to let a D-STAR user and a DMR user *on the same reflector* hear each other. Use DVSwitch to connect that reflector to *something outside it*: to inject a specific Brandmeister talkgroup, to reach EchoLink or analog through AllStar over USRP, or to bridge an XLX network to a URF one (since the two cannot interlink directly). A large multi-network system is typically an XLX reflector at the center with a small fleet of MMDVM_Bridge-and-Analog_Bridge pairs hanging off its modules, each pair reaching out to one external network.

Chapter 21 — The Reflector's Status File, and Building Your Own Tools

Anyone who wants to monitor a reflector, build a custom dashboard, or feed its activity into some other system needs to know what data the reflector exposes — and XLX exposes it as an XML status file that has become a de-facto standard across the whole ecosystem.

The reflector regenerates an **XML status file** every few seconds. After a version element it contains three sections: a **linked-peers** list, a **linked-nodes** list (the repeaters and hotspots connected in), and a **heard-users** list. Each connected peer appears with its callsign, address, shared modules, protocol, and connect and last-heard timestamps; each node with its callsign, module, and protocol; and each recently-heard station with callsign, module, and a last-heard time. This is the file the built-in PHP dashboard parses, and it is the natural thing for any external monitoring to read. It is worth appreciating that this format is the *original* — the "XLX-shaped status file" that URF reflectors deliberately imitate so they slot into XLX dashboards and registries; on an XLX reflector it is simply the reflector describing itself.

Because the guide promises tools and not just formats, here is a complete, dependency-free reader. It parses the status file, prints a one-line "who's connected and who was last heard" summary, and exits non-zero if the file is stale — so it doubles as the health probe from Chapter 16, catching a silent transcoder before your users do.

```
#!/usr/bin/env python3
# xlxstat.py — summarise an xlxd XML status file; exit 2 if stale.
# Usage: xlxstat.py /var/www/db/xlx.status.xml
import sys, os, time
import xml.etree.ElementTree as ET

path = sys.argv[1]
age  = time.time() - os.path.getmtime(path)          # file is rewritten every few seco
root = ET.parse(path).getroot()

peers = len(root.findall(".//PEER"))
nodes = len(root.findall(".//NODE"))
last  = []
for s in root.findall(".//STATION")[:5]:
    cs  = s.findtext("Callsign", "?")
    mod = s.findtext("Module", "?")
    last.append(f"{cs}@{mod}")

summary = f"peers={peers} nodes={nodes} last_heard={', '.join(last) or '(none)'}"
print(f"{summary} age={age:.0f}s")
if age > 60:
    print("STALE: status not updated in 60s — check xlxd/ambed", file=sys.stderr)
    sys.exit(2)
```

The exact filename and element names vary a little between the classic and modernized builds, so point the script at your reflector's actual status file and adjust the tags if needed — but the three-section shape is stable, and anything the dashboard shows you can compute yourself: a Grafana feed, a Discord notifier, or a network-wide "who's talking" board.

Chapter 22 — Deployment: DNS, HTTPS, and Watching the

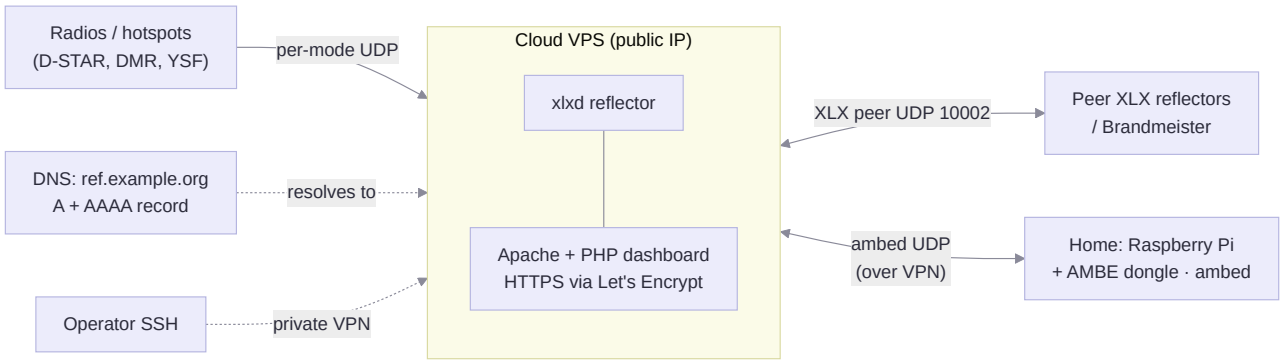
Traffic

Turning a working reflector into a properly deployed public service takes a few steps the software itself does not perform, and a complete reference should cover them even though some are ordinary web-hosting practice.

It starts with **DNS**. The reflector should have a stable hostname — an A record, and an AAAA record if the host has IPv6 — that maps something like `ref.example.org` to the server's public address. That hostname, not a bare IP, is what belongs in the host files and the dashboard URL, so the address can change without breaking every user's configuration.

Next is **HTTPS for the dashboard**. The dashboard is a PHP site served by Apache; serving it securely is standard practice. The usual tool is Let's Encrypt via Certbot — `certbot --apache` gets a certificate and edits the Apache configuration to serve it in one step, and renews automatically. Because the dashboard only reads the reflector's generated files, it can even run on a different machine behind a reverse proxy that terminates TLS. Finally, many operators like to see their dashboard traffic without a heavy commercial tracker; a lightweight, privacy-respecting, self-hostable analytics package such as GoatCounter fits a hobby community page well and is entirely optional.

A complete deployment, with the pieces of the last several chapters in one place:



Chapter 23 — 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 module carries which mode. For most users it is the only part of a reflector they ever touch directly, and it is the page they leave open and glance at all day, so it deserves its own chapter separate from the plumbing that feeds it (Chapter 21) and the deployment steps that serve it (Chapter 22). It is also worth noting that this dashboard is the *original* from which the whole family's dashboards descend.

What it is, and how it is set up

The dashboard ships with `xlxd` as a PHP web application, deployed into the web root (classically `/var/www/db`). It is deliberately simple: it does not talk to the reflector directly or hold state of its own — it

only *reads* the XML status file the reflector regenerates every few seconds and renders it as HTML. That means it can run on the same server as the reflector or on a separate web host with access to the file, and restarting the reflector never disturbs it. Setup is the last step of a build (Chapter 14): copy the dashboard into your web server and edit its single configuration file, `config.inc.php`, which holds the operator-facing settings — your sysop **email**, **country**, a free-text **comment**/description, the **module names and ports** the dashboard displays, and the **calling-home** toggle (leave it off until you have confirmed a collision-free callsign suffix, per Chapter 12). Then serve it over HTTPS. An XLX-specific optional extra is the Wires-X hot-spot frequency registration interface, which needs the YSF frequency database and an SQL schema loaded; most operators leave it off.

What it shows, and how operators and users read it

A dashboard presents the reflector's live state as a few stacked panels, all built from the status file's three sections: an identity banner (callsign and description); a **module / room list** with each active module A–Z and the operator's description of what it carries, so a newcomer can tell which module to select; a **linked-nodes** list and a **peer-reflectors** list with callsign, module, protocol, a masked address, and connect/last-heard times; and the panel everyone actually watches, the **"Last Heard" list** — the most recent stations to key up, with callsign, module, mode, and timestamp. The Last Heard list is the reason hams keep the page open: after you transmit, your callsign appearing there, on the module and mode you expect, is the quickest confirmation the reflector heard you — and its first diagnostic value when something is wrong (if your callsign shows but no one hears you, you have the transcoder-or-wrong-module symptom from Chapter 15).

How hams access it

Access is deliberately frictionless: the dashboard is an ordinary web page at the reflector's hostname — `https://ref.example.org` — opened in any browser on a computer or phone, with **no login required** to view. That URL is what an operator publishes, what appears next to the reflector in the DVRef registry and host-file listings, and what club members bookmark. Viewing is read-only for the public; there is no web-based administration, so an operator changes configuration on the server (Chapter 14), not through the dashboard. The dashboard is a *view*, not the reflector: if it is down but the reflector is up, traffic still flows; you have simply lost the window into it.

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

XLX307 — D-STAR / DMR / Fusion ● online · updated 5s ago

Modules: A — Multi-mode | B — DMR | D — Fusion

LAST HEARD

Callsign	Module	Mode	Last heard (UTC)
W1ABC	A	D-STAR	2026-08-08 17:42:10 ◀ transmitting
K2AS	A	DMR	2026-08-08 17:41:58
DL1XYZ	D	Fusion	2026-08-08 17:39:12
G4XYZ	B	DMR	2026-08-08 17:35:47

CONNECTED NODES

Callsign	Module	Protocol	Address	Since (UTC)
W1ABC B	A	DExtra	73.x.x.x	16:02
DL1XYZ	D	YSF	84.x.x.x	15:47

LINKED PEERS

Callsign	Shared modules	Protocol	Since (UTC)
XLX589	A	XLX peer	Aug 07 09:14

Chapter 24 — The AMBE Patents and the Open-Codec Movement

To really understand why XLX is built the way it is — why it needs DVSI hardware to bridge D-STAR, why the transcoder is all-hardware — you have to understand the patent history of the codecs, because that history is the hidden force shaping the whole field.

For most of amateur digital voice's life, the codecs were a closed door. The Multi-Band Excitation family — IMBE, then AMBE, then AMBE+2 — was developed and patented by a single company, Digital Voice Systems, Inc., and using them meant buying DVSI's hardware or licensing their software. This is why D-STAR, DMR, and Fusion all depend on hardware vocoder chips, and it is the reason a transcoding XLX reflector needs real DVSI dongles for any D-STAR bridging. It also sat awkwardly with amateur radio's culture of building your own equipment: you could not legally write your own software to encode or decode these modes.

That door has been slowly opening as the patents age out. The original 1997 Multi-Band Excitation patent has expired, and the specific patents on the AMBE variant used by D-STAR expired around 2017 — a fact widely attributed to Bruce Perens's analysis. That expiry is what made open software decoders legally defensible for the older modes, and it is why the *later* URF transcoder can do some AMBE work in software at all. The important caveat is that AMBE+2 — the codec used by DMR and Fusion — is *not* yet free; it remains under DVSI license,

with a remaining patent estimated to run until roughly 2028 (*as of mid-2026*). This is precisely why `ambed`, written in XLX's era, is all-hardware: when it was built, doing this in software was neither legal nor practical.

The open answer to all of this — M17, built on the patent-free Codec2 — arrived after XLX and is carried by URF, not by XLX. That is not a criticism of XLX so much as a matter of timing: XLX is the mature reflector of the proprietary-codec era, and the open-codec future landed in its successor. Understanding that history explains both why XLX depends on DVSI and why, if the open modes matter to you, the newer software is where to look.

Chapter 25 — A Worked Example, and Migrating from Older Reflectors

To make all of this concrete, consider a realistic build: a three-module XLX reflector called XLX307 on a small cloud server, with module A carrying a transcoded D-STAR/DMR/Fusion bridge, module B a DMR room, and module D a Fusion room — and the AMBE dongle living on a Raspberry Pi at home, reaching the reflector over a private VPN.

On `new-xlxd` you would run `./rconfig` and answer its prompts — callsign `XLX307`, the IPv4/IPv6 bindings, the transcoder port, whether to enable G3 — and it generates the compiled configuration for you. You then declare the interlink and edit the dashboard config. A minimal `xlxd.interlink` peers module A with a neighbouring reflector:

```
# xlxd.interlink — one peer per line: callsign, address, shared modules
XLX589 xlx589.example.org A
```

and the dashboard's `config.inc.php` carries the operator-facing settings:

```
$CallingHome   = false;             // leave off until your suffix is confirmed free
$SysopEmail    = "you@example.org";
$Country       = "US";
$Comment       = "XLX307 — D-STAR / DMR / Fusion";
$ModuleNames['A'] = "Multi-mode (transcoded)";
$ModuleNames['B'] = "DMR only";
$ModuleNames['D'] = "Fusion only";
```

On the Raspberry Pi at home, `ambed` runs with the DVSI dongle plugged in and its controller/stream ports (10100 and 10101–10199) reachable from the reflector across the VPN. With that, module A transcodes D-STAR to DMR and Fusion; modules B and D carry single-mode traffic (and, note, B and D could even be bridged to each other with no dongle, since DMR and Fusion share AMBE+2). Point a DNS record and an HTTPS certificate at the dashboard, walk the Chapter 14 test procedure, and it is a complete, working, properly-deployed reflector.

Migrating from older reflectors. For a D-STAR community coming from a REF, XRF, or DCS reflector, XLX is a natural home: it speaks all three of those legacy D-STAR protocols inbound, so existing users can keep connecting the way they always have while you add DMR and Fusion alongside. The realistic path is to stand XLX up beside the old reflector, move users and any interlinks over, and retire the old one once traffic has shifted. And for the operator who later wants what XLX cannot do — M17, P25, NXDN, AllStar, or software-only transcoding — the onward path is URF, which is a superset of XLX and slots into the same dashboards and host files; the concepts (modules, interlink, transcoding) carry across directly, though the two cannot interlink as peers during the transition.

Chapter 26 — The Case For XLX: Strengths and Benefits

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

Its central strength is **maturity and reach**. XLX is the reflector the modern D-STAR/DMR/Fusion ecosystem was built around: an enormous installed base, years of operational hardening, and more community documentation, how-tos, and operator experience than any other multi-protocol reflector. When something goes wrong at 2 a.m., the odds that someone has already written up your exact problem are higher with XLX than with anything newer. For a D-STAR-centric club or a network living in those three modes, that maturity is worth a great deal.

Its second strength is that it is **focused and well-understood**. Three modes and two codecs is a smaller surface than URF's six-and-four, and that simplicity shows: fewer ports to open, fewer moving parts, and a transcoding story you can hold in your head — D-STAR needs the chip, DMR and Fusion don't need one to talk to each other. The all-hardware transcoder, for all that it requires a dongle, has a real virtue: transcoding load lands on the DVSI chip, not the host CPU, so a modest server stays light even when a transcoded module is busy.

Finally, there are the virtues of how it is built and licensed. XLX is open source under the GPL, so an operator can read exactly what it does. It runs happily on a small VPS or even a Raspberry Pi, its costs are modest, and — through the classic `xlxd` and the modernized `new-xlxd` — it remains available in both a battle-tested form and a cleaner, dual-stack, actively-modernized one. It is, in short, the dependable workhorse of amateur multi-protocol reflectors.

Chapter 27 — The Case Against: Limitations and Trade-offs

An honest reference has to be as clear about the weaknesses as the strengths, and XLX has real ones — some inherent to when it was built, some to how it works.

The most consequential limitation is **scope**: XLX carries D-STAR, DMR, and Fusion, and nothing else. No M17, no P25, no NXDN, and no integrated USRP/AllStar. If any of those matter to you, XLX cannot do them and its successor URF is simply the better tool — this guide does not pretend otherwise. Bound up with this is the **all-hardware transcoder**: any module that bridges D-STAR to another mode requires a DVSI dongle, with no

software fallback of the kind URF introduced, so there is an unavoidable hardware cost to D-STAR mode-bridging. (The consolation, unique to XLX's small codec set, is that DMR ↔ Fusion needs no dongle at all.)

There are architectural limits, too. Interlinking is XLX-to-XLX (plus a single DExtra peer link and DMR-network links), and an XLX reflector **cannot interlink with a URF reflector**, which splits it from the growing URF side of the ecosystem unless you bridge externally. Discovery leans on **central services** — the XLXAPI, DVRef, host-file mirrors — rather than the decentralized Ham-DHT the newer software can use. Configuration is a **build-time** affair (edit-and-recompile on the classic build; the `rconfig` generator softens this on `new-xlxd`) rather than a live config file. Its built-in security is minimal — a callsign blacklist and whitelist and nothing more — so firewalling and DDoS resilience are entirely the operator's problem, and its open UDP ports are inherently exposed. And it has no tagged releases and no published performance figures, so capacity planning is estimation rather than measurement.

None of this makes XLX a poor choice; it makes it a *tool with a shape and an era*. For a mature D-STAR/DMR/Fusion network it is close to ideal. For a from-scratch build that wants the open modes or software transcoding, URF is the more future-proof answer.

Chapter 28 — Voice Quality and Latency

The quality cost of transcoding is real, measurable, and worth understanding honestly, because it is the one downside a bridging reflector cannot engineer away.

Every cross-codec conversation — that is, any D-STAR-to-DMR-or-Fusion contact on XLX — is an instance of what the literature calls *tandem vocoding*: the audio is decoded from the sender's codec into plain PCM and re-encoded into the receiver's codec, and each step is lossy. DVSI's own transcoding white paper, corroborated by an independent NTIA study, states the mechanism directly: "during the process of transcoding from one vocoder to another, additional degradation occurs," and puts a number on it from a Modified Rhyme Test — "an average loss of more than 12% in intelligibility." Read that as indicative rather than gospel, but the direction is not in doubt: a transcoded QSO sounds worse than a same-mode one. XLX has one quiet advantage here, though: because it always vocodes on the DVSI *hardware* rather than a software vocoder, its AMBE handling is the quality benchmark, not the compromise — and its most common bridge, DMR ↔ Fusion, is not a true transcode at all (same codec, reframed), so it suffers no vocoding loss.

Latency is the other cost. No one has published an end-to-end figure for an XLX transcoded round trip, but first principles bound it: each codec works on 20 ms frames, a transcode adds a decode-to-PCM and a re-encode, and there is network round-trip time between the talker, the reflector, the transcoder (which may be a separate machine), and the listener. A same-mode contact across a reflector is often in the ballpark of 100–200 ms mouth-to-ear; a transcoded D-STAR ↔ DMR contact, with the extra decode/re-encode and frequently an extra hop to a remote `ambed`, realistically lands a few hundred milliseconds higher. Treat that as an order-of-magnitude estimate. The practical mitigations follow from the mechanism: keep heavily-used single-mode traffic on untranscoded modules, reserve the transcoded module for genuine cross-codec bridging, and keep `ambed` near the reflector on the network.

Chapter 29 — Who Should Run One, and What It Costs

The clearest way to decide whether to run an XLX reflector is to look at who runs them and why. The recurring rationale is a club or regional group that spans D-STAR, DMR, and Fusion and wants those communities in one place instead of on separate infrastructure; a D-STAR-centric network that values a mature, well-documented platform; or an operator continuing an established XLX or XRF presence. XLX is a poorer fit for someone who needs M17, P25, NXDN, or AllStar, or who wants software-only transcoding — those point to URF — and for someone who wants a purely single-mode reflector with the least possible fuss, where a simpler purpose-built server may serve better.

The costs are modest and worth stating concretely, with the caveat that prices change (*figures below are as of mid-2026*). On the hosting side, an XLX reflector is light enough for a small virtual server: entry plans with one virtual CPU and a gigabyte or two of memory run about five to six US dollars a month from providers like Linode, Vultr, or DigitalOcean, and cheaper still — a few euros — from Hetzner. On the hardware side, the XLX-specific rule is the one to remember: if you will bridge D-STAR to DMR or Fusion you need a DVSI dongle — a single-channel DVstick-30/ThumbDV at around one hundred US dollars, or a three-channel DVstick-33 for a busy reflector at more — but if your reflector is D-STAR-only (or bridges only DMR and Fusion, which share a codec) you need no vocoder hardware at all. So the realistic entry cost is a few dollars a month, plus a one-time dongle purchase only if D-STAR mode-bridging is on the menu.

Chapter 30 — Frequently Asked Questions

Do I need a radio to use an XLX reflector? No. Software clients — DroidStar for all three modes, BlueDV (with a DVSI dongle) for the vocoding — let a computer or phone connect directly, which is also the easiest way to test your own reflector.

Do I need a DVSI dongle to run one? Only if you will bridge D-STAR to DMR or Fusion. DMR and Fusion share the AMBE+2 codec, so bridging just those two needs no hardware; and a D-STAR-only or single-mode reflector needs none either. The moment a module mixes D-STAR with another mode, you need a dongle — there is no software transcoding path on XLX.

How many modules can be transcoded? It depends on your DVSI channels: bridging D-STAR to another mode spends two channels, so a three-channel AMBE-3003 handles roughly one or two D-STAR bridges at once, while DMR ↔ Fusion modules cost no channels.

What's the difference between XLX and XRF? Same source, two builds: XLX is the full multi-mode reflector (D-STAR + DMR + Fusion, transcoding, calling-home eligible); XRF is D-STAR only, no transcoder, and must not call home. See Chapter 11.

Should I run XLX or URF? Run XLX for a mature D-STAR/DMR/Fusion network. Run URF if you want M17, P25, NXDN, integrated AllStar, software-only transcoding, or DHT discovery — URF is a superset of XLX and can serve every XLX mode too. See Chapter 18.

Should I build classic `xlxd` *or* `new-xlxd`*?* New builds are more comfortable on `new-xlxd` (rconfig/radmin, systemd, dual-stack); an established classic reflector has little reason to switch. On-air behavior is the same.

Can an XLX reflector link to a URF reflector? Not directly — they cannot interlink. Bridge them externally with DVSwitch (Chapter 20).

I connected but no one can hear me — why? Almost always the wrong or an untranscoded module (the shipped host file's default module is a classic trap), or the transcoder is down on a D-STAR bridge (callsigns-but-no-audio). Select the correct module; if that isn't it, restart `ambed` and confirm the DVSI device is present. Chapter 15 has a decision tree.

How do people find my reflector? Through the host files (enable calling home to `xlxapi.rlx.lu` — but only with a collision-free callsign suffix) and by listing on DVRef. XLX has no Ham-DHT.

Chapter 31 — Getting Help

There is no single official support forum for XLX, so it helps to know where to look. The primary channels are the GitHub repositories and their issue trackers: `LX3JL/xlxd` for the classic build and `n7tae/new-xlxd` for the modernized fork, along with the `ambed` transcoder source. Beyond GitHub, the community has produced an unusual amount of practical documentation over the years — the New England Digital Radio, N5AMD, KB9MWR, and Wayne Technical Fanatics write-ups are all well-known starting points for setup and transcoding, and hotspot-side configuration for connecting to an XLX reflector is Pi-Star and WPSD forum territory. Broader digital-voice questions are answered on the RadioReference forums, and for bridging to other networks the DVSwitch groups.io lists are the gathering place. Because XLX and URF share a lineage and an author for the modern forks, much URF-side advice transfers directly — the companion URF guide is a useful second reference.

Chapter 32 — Where the Sources Disagree

In keeping with the principle of not smoothing over genuine conflicts, a handful deserve to be named. The sharpest is the **AMBE-3003 "two versus three streams" question**: the chip has three channels, but how many *usable* transcoded modules that yields depends on whether both sides of a bridge must spend a hardware channel — which, on the all-hardware `ambed`, they do for D-STAR bridging — so real-world capacity is often quoted lower than the raw channel count. There is an apparent contradiction in the **Fusion registration** guidance — you needn't register your XLX's YSF side at ysfreflector.de, yet the dashboard has Wires-X/frequency registration fields — which resolves once you see that the YSF Master functions without registration and those fields merely feed the optional hot-spot frequency database. The **interlink port** and exact peer-protocol details are quoted slightly differently across sources and between the classic and modernized builds, so verify against your own build. And there are **classic-vs-`new-xlxd` differences** in config workflow (hand-edited headers vs `rconfig`), status-file naming, and startup (init.d vs systemd) that mean a how-to written for one build may not match the other line-for-line. Where this guide gives a specific value, it reflects the primary sources read during research; where a live build differs, trust the build.

Chapter 33 — 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. Several registry and dashboard behaviors — the exact fields the live XLXAPI expects, the precise status-file element names on a given build, the current DVRef workflow — are described from documentation and operator accounts rather than confirmed against a running reflector during research. No quantified performance benchmark for `xlxd` exists; the bandwidth, CPU, and latency figures in Chapters 16 and 28 are first-principles estimates, explicitly labelled as such, not measurements. The exact behavior of `ambed` when its hardware is missing, and the precise channel accounting for a lone AMBE-3003, should be re-confirmed on real hardware. And because `xlxd` has loosely-tagged releases, the honest way to record "which version am I running" is to note your built commit hash. None of these gaps undermines the picture; they are simply the edges where documentation runs out and a practitioner's fifteen minutes would be worth more than any amount of further reading.

Glossary

AllStar (AllStarLink) — a VoIP network of amateur repeaters and nodes built on the Asterisk PBX; XLX reaches it only via an external DVSwitch/USRP bridge.
AMBE — the proprietary DVSI voice codec used by D-STAR.
AMBE+2 — the proprietary DVSI codec used by DMR and Fusion; shared by both, so bridging them needs no vocoder hardware.
ambed — XLX's all-hardware transcoder daemon; converts between AMBE and AMBE+2 on DVSI chips over a UDP link to the reflector.
Analog_Bridge / MMDVM_Bridge — DVSwitch components that bridge digital modes to each other, to analog, and to AllStar over USRP.
BrandMeister — one of the two large worldwide DMR networks; XLX can link to it through the interlink file.
Calling home — the optional feature that reports an XLX reflector to the central registry at `xlxapi.rlx.lu`; must not be enabled on an XRF build.
C4FM — Continuous 4-level FM, the 4FSK modulation used by Yaesu System Fusion.
Codec — hardware or software that encodes and decodes a data stream; here, the voice coder.
config.inc.php — the dashboard configuration file (email, country, comment, module names/ports, calling-home toggle).
DCS — one of the three legacy D-STAR reflector protocols; originated in Germany.
DExtra — the open D-STAR reflector protocol behind XRF reflectors; XLX also uses it for a single-module peer link.
DG-ID (Digital Group ID) — a Fusion group-access number; on XLX it can select a room/module.
DMR — Digital Mobile Radio, the ETSI two-slot TDMA standard.
DMRGateway — the software that lets one hotspot connect to several DMR networks at once.
DMR+ (IPSC2) — a DMR network; XLX accepts its dongle (DMO) protocol.
DPlus — the original D-STAR reflector protocol, behind REF reflectors.

DroidStar — a multi-mode software client (D-STAR/DMR/Fusion) that connects from a computer or phone.
D-STAR — the JARL digital-voice protocol; the oldest mainstream amateur digital mode.
DVRef — the web registry/directory where reflectors are listed.
DVSI (Digital Voice Systems, Inc.) — the company that owns the AMBE and AMBE+2 codecs.
DVstick-30 / DV3000 / ThumbDV — single-channel AMBE dongles (AMBE-3000 chip).
DVstick-33 / DV3003 — three-channel AMBE transcoding dongles (AMBE-3003 chip).
DVSwitch — a suite of tools for bridging digital-voice modes and networks to each other and to analog.
FEC (Forward Error Correction) — redundant data that lets a receiver correct errors without retransmission.
FTDI D2XX — the USB driver the DVSI dongles require; it conflicts with the kernel `ftdi_sio` driver.
G3 — Icom's D-STAR terminal and access-point gateway mode.
Golay / Hamming codes — block FEC codes that let a receiver correct bit-errors; guard a voice frame's most important bits.
GPL — the open-source license under which `xlxd` is released.
Host file — the address list hotspots download to populate their reflector menus.
Hotspot — a small personal device that links a radio to internet digital-voice networks.
IMRS — an internet-linking protocol built into newer Yaesu repeaters; supported by the classic XLX build.
Interlink — a mutual, same-letter link between reflectors, via the XLX peer protocol or a single DExtra link.
main.h — the classic-build compiled header where certain options (e.g. YSF frequency) are set before building.
Module — one of a reflector's up-to-26 independent rooms, lettered A–Z.
new-xlxd — N7TAE's modernized C++ fork of XLX (rconfig/radmin, systemd, dual-stack).
Reflector — a server that relays a mode's traffic among all connected stations; a conference bridge.
rconfig / radmin — the `new-xlxd` interactive configuration generator and administration menu.
Tandem vocoding — decoding one codec to audio and re-encoding it into another; the source of transcoding's quality loss.
URF / urfd — the universal reflector that extends XLX with M17, P25, NXDN, USRP, and software transcoding; a superset of XLX that cannot interlink with it.
URCALL — the D-STAR field where a linking command (reflector + module + L/U) is entered.
USRP — a simple UDP protocol carrying PCM audio between DVSwitch/AllStar components; not integrated in XLX.
US-Trust — the original central registration/trust-server system for D-STAR gateways.
Vocoder — the algorithm that turns speech into a low-bitrate data stream and back; the reason transcoding is needed.
VPS (virtual private server) — a rented virtual machine, the usual host for a reflector.
Wires-X — Yaesu's proprietary internet repeater-linking system; XLX presents 26 Wires-X rooms.
XLX (xlxd) — the multi-protocol reflector this guide is about.
XRF — the D-STAR-only build variant of the XLX source (DExtra/DPlus/DCS in, DExtra out; no transcoder).
YSF (Yaesu System Fusion) — Yaesu's C4FM digital-voice system; on XLX it is served as a self-contained YSF Master.
YSFReflector — the open, decentralized Fusion reflector network, separate from XLX's YSF Master.

Appendix A — Network Ports

All ports are UDP except HTTP/HTTPS, which are TCP. You do not need to open ports for modes you have disabled. *Values are typical defaults; a given build (classic vs new-xlxd) or reflector may override them, so verify against your own configuration.*

Service	Port	Transport	Notes
Dashboard	80 / 443	TCP	web
DMR+ DMO	8880	UDP	
XLX interlink	10002	UDP	reflector-to-reflector (XLX peer protocol)
ambed controller	10100	UDP	reflector ↔ transcoder
ambed streams	10101–10199	UDP	per-stream transcoding channels
G3 presence / request	12345–12346	UDP	Icom terminal
DPlus	20001	UDP	D-STAR (REF-style)
DExtra	30001	UDP	D-STAR (XRF-style); also single-module peer link
DCS	30051	UDP	D-STAR
G3 terminal	40000	UDP	Icom terminal/access-point
YSF	42000	UDP	Fusion (YSF Master)
MMDVM	62030	UDP	DMR
Not used by XLX (URF adds these): M17 17000 · P25 41000 · NXDN 41400 · USRP 32000 · Ham-DHT 17171			

Appendix B — How to Connect, by Mode (quick reference)

Mode	How the user selects a module
D-STAR	URCALL = reflector callsign + module letter + `L`/`U` (e.g. `XLX307C L`); or DTMF (digits set by your hotspot); or the dashboard
D-STAR G3 (terminal)	Register on a G3 gateway, assign a module letter, RS-MS3A/W, reflector IP, forward UDP 40000
DMR	Private call `68` + reflector number, then `64000` + module number (A=1…Z=26); talk on **TG 6**; `64000` to disconnect; `65000` for status. On DMR+, TG `4001`–`4026` selects module A–Z directly and TG `4000` unlinks.
YSF (Fusion)	Connect to the YSF Master, then Wires-X room, or DG-ID → module (mapping is per-reflector)

Appendix C — Configuration at a Glance

Classic `LX3JL/xlxd`: configuration is compiled in — edit the build header (historically `main.h`, e.g. the YSF hot-spot frequency and compile-time options), build, and start via `/etc/init.d/xlxd`. The dashboard is served from `/var/www/db`.

Modernized `n7tae/new-xlxd`: run `./rconfig` to answer prompts (callsign, IPv4/IPv6 binding, transcoder port, G3 toggle, YSF database) — it *generates* the C++ headers and makefile fragments — then use `./radmin` to clean, compile, install/uninstall, and follow the reflector or transcoder logs; services run under systemd.

Shared files: `xlxd.interlink` (peers: callsign, address, shared modules) and, for XRF, `xrfd.interlink`; `xlxd.blacklist` / `xlxd.whitelist` (callsign deny/allow, wildcard `*`; empty = open network); optional `xlxd.terminal` (G3 linking); and the dashboard's `config.inc.php` (email, country, comment, module names/ports, calling-home toggle). Record the git commit you built from, since releases are loosely tagged.

Appendix D — A Timeline of the Reflector Lineage

mid-2000s — REF reflectors (DPlus protocol), Robin Cutshaw AA4RC; tied to the US-Trust system.
late 2000s–2013 — XRF reflectors (DExtra), Scott Lawson KI4LKF; the first open reflector protocol; work ceased in 2013.
~2012 — DCS reflectors, originating in Germany with Torsten Schultze DG1HT.
October 2015 — XLX conceived by the Radio Amateurs du Luxembourg working group (LX3JL, LX1IQ).
7 November 2015 — first XLX beta released.
January 2016 — XLX adopts the GPL; the "© 2016" copyright dates from here.
2016 onward — XLX adds DMR and Fusion with AMBE (`ambed`) transcoding; grows into the backbone of

multi-protocol amateur digital voice.
~2017 — the patents on the D-STAR AMBE codec variant expire, making software vocoders legally defensible (later exploited by URF, not XLX).
~2020 — N7TAE releases `new-xlxd`, a modernization of the XLX codebase.
2022 — N7TAE and Doug McLain AD8DP release `urfd` (URF), extending XLX with M17, P25, NXDN, and USRP and replacing `ambed` with the hybrid `tcd`.

Index

Because XLX 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
AMBE / AMBE+2 codecs	Ch 5, **6**, 19, **24**; Glossary
ambed transcoder	Ch 5, 6, **9**, 14; Glossary
Bandwidth / capacity math	**Ch 16**, 29
Blacklist / whitelist	**Ch 8**, 16, 27
Building / installing	**Ch 14**, 18, 25
Calling home	**Ch 12**, 11, 23
config.inc.php	**Ch 23**, 14, 25; App C
Costs	**Ch 29**
Dashboard	**Ch 23**, 14, 21, 22
Decision tree (no audio)	**Ch 15**
DMR connection / talkgroups	**Ch 8**; App B
DMR ↔ Fusion (no dongle)	**Ch 5**, 6, 9, 19
DNS / HTTPS deployment	**Ch 22**
DVSI hardware / dongles	**Ch 6**, 9, 18, 29
DVSwitch bridging	**Ch 20**
FEC (Golay, Hamming)	**Ch 19**; Glossary
Finding / registering	**Ch 12**; Glossary
Interlinking (XLX peer, DExtra)	**Ch 10**, 25, 27
Law / FCC Part 97	**Ch 17**
Latency	**Ch 28**
Migrating (REF/XRF/DCS → XLX)	**Ch 25**, 2
Modules	**Ch 3**, 4, 8, 10
Monitoring / status file	Ch 7, **21**, 16
new-xlxd vs classic xlxd	Ch 2, 14, **18**, 32
Ports	Ch 7, 16; **App A**

Topic	Where
rconfig / radmin	**Ch 14**; App C; Glossary
Security / hardening	Ch 8, **16**, 27
Testing / verification	**Ch 14**, 13
Transcoding (concept)	**Ch 5**, 9, 28
Troubleshooting	**Ch 15**, 14, 30
Voice quality	**Ch 28**, 27
XLX vs URF	Ch 1, **18**, 27, 29, 30
XLX vs XRF	**Ch 11**, 18
YSF / Fusion as YSF Master	**Ch 4**, 8, 32

Sources

The backbone of this guide is the primary source code and documentation of the software itself: the classic `xlxd` reflector and its dashboard and ambed transcoder (`github.com/LX3JL/xlxd`); the modernized fork (`github.com/n7tae/new-xlxd`); and, for the lineage forward, `new-xlxd` 's successor `urfd` and its transcoder `tcd` (`github.com/n7tae/urfd` , `github.com/nostar/urfd`).

Independent operators and how-to authors running live XLX reflectors provided real-world corroboration and practical detail: New England Digital Radio (`newenglanddigitalradio.com`) on XLXD setup and ambed transcoding; N5AMD's digital-voice resource (`n5amd.com`) on building XLX and transcoding servers and the multi-reflector installer; KB9MWR's XLX transcoding-system procedure (`qsl.net/kb9mwr`); and Wayne Technical Fanatics (`ww8tf.club`) on building and updating a transcoding XLX Pi.

Standards, specifications, and reference material grounded the technical and historical chapters: the FEC and codec references (Wikipedia on the Binary Golay code, Hamming code, Multi-Band Excitation, Project 25, NXDN, and System Fusion; DVSI's AMBE-3000/3003 product pages; the FTDI D2XX driver documentation); F4FXL and amateurradionotes on reflector history; and the XLX team's own project timeline. The legal chapter draws on 47 CFR §97.119 and §97.113 (Cornell Legal Information Institute) and the ITU Radio Regulation 25.2A text. The voice-quality material came from DVSI's transcoding white paper and the corroborating NTIA tandem-vocoder study. Costs came from GigaParts, NW Digital Radio, and dxcanada for the DVstick/ThumbDV, and a 2026 VPS-price comparison. Client software is documented from the DroidStar and BlueDV project pages, and the wider-network chapter from the DVSwitch organization's Analog_Bridge and MMDVM_Bridge repositories.

The sizing, testing, and operations material — the bandwidth and CPU estimates (Chapter 16), the latency estimate (Chapter 28), the FEC arithmetic (Chapter 19), the per-mode test procedure (Chapter 14), and the status-

file reader (Chapter 21) — is derived from the codec bitrates and protocol behavior cited above combined with standard Linux server and networking practice; where a figure is an estimate rather than a measurement, the text says so, and Chapter 33 records what a live operator could confirm. This guide is a companion to *The Complete Guide to URF Reflectors*, which covers the modes and software beyond XLX's scope in the same style.

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.