

Building an M17 ↔ AllStarLink Bridge with a Live Web Dashboard

A complete, from-scratch guide to linking an AllStarLink node to the M17 digital voice network using **USRP2M17**, with a private AllStar node, a DTMF-controlled reflector switcher, and a live-updating web dashboard.

This guide uses placeholders throughout -- replace them with your own values:

Placeholder	Meaning	Example
<YOUR_CALLSIGN>	Your amateur radio callsign	NØCALL
<RADIO_NODE>	Your existing AllStar node connected to your radio/hotspot	500123
<PRIVATE_NODE>	A new private node number you'll create for this bridge (private range 1000-1999)	1100
<PI_USER>	The Linux user account USRP2M17 will run as	pi (or your own username)

1. Architecture

```
Radio/Hotspot (<RADIO_NODE>) <-> private node <PRIVATE_NODE> (USRP 34008:32008) <-> USRP2M17 <-> M17 reflector
```

The M17 bridge runs on its own private AllStar node, completely independent of any DMR/D-STAR/YSF bridge (DVSwitch, etc.) you may already have running on the same system, with one caveat: when you dial a reflector, the switcher script (section 6b) first drops any OTHER links on your radio node, then links it to the private M17 node. So switching to M17 will unlink a DVSwitch node or any other node that was linked to your radio node at the time.

Prerequisites: a working AllStarLink v3 install with at least one existing radio/hotspot node, sudo access, and PHP + Apache (or another web server) for the web dashboard.

2. Create the Private AllStar Node

This is the internal "node" that the M17 bridge software talks to. It never keys a physical radio directly -- it's a software-only link between your existing radio node and USRP2M17. Pick a number in the AllStarLink private node range (1000-1999) that isn't already used on your system. For example, if a DVSwitch bridge already uses 1000, use something like 1100 for M17. Private nodes don't need to be registered with AllStarLink, but they can only link to nodes on the same server.

`/etc/asterisk/custom/rpt/m17.conf` (create this file):

```
[<PRIVATE_NODE>](node-main)
duplex = 0 ; CRITICAL: must be 0, or audio passes C0S/PTT but all samples get dropped
rxchannel = USRP/127.0.0.1:34008:32008
```

Make sure this file gets `#include`d from your main `rpt.conf` (AllStarLink's default `rpt.conf` typically already includes everything under `/etc/asterisk/custom/rpt/`; check the top of your `rpt.conf` for an `#include` directive covering that directory).

`/etc/asterisk/rpt.conf`, in the `[nodes]` section, add:

```
<PRIVATE_NODE> = radio@127.0.0.1/<PRIVATE_NODE>,NONE
```

Restart Asterisk after these changes:

```
sudo systemctl restart asterisk
```

3. Build USRP2M17

USRP2M17 is part of the `MMDVM_CM` project and bridges AllStar's USRP protocol to the M17 network protocol.

```
cd ~
git clone https://github.com/nostar/MMDVM_CM
cd MMDVM_CM/USRP2M17
```

On some newer compilers you'll hit missing-type build errors. If so, add this line near the top of each of these four files (after the existing `#include` lines):

```
#include <cstdint>
```

Files that may need it: `Conf.h`, `Conf.cpp`, `ModeConv.h`, `ModeConv.cpp`.

Then build:

```
make
```

This produces the `USRP2M17` binary in the same directory.

4. Configure USRP2M17

`/etc/USRP2M17.ini` (this path is hardcoded by the binary):

```
[M17 Network]
; Address= and Name= can be left blank -- the reflector switcher fills them in
Callsign=<YOUR_CALLSIGN>
Address=
Name=
LocalPort=32011
DstPort=17000
GainAdjustdB=3
Daemon=0
Debug=1

[USRP Network]
; Address must stay 127.0.0.1 -- this talks to the private AllStar node
Address=127.0.0.1
DstPort=32008
LocalPort=34008
GainAdjustdB=-6
Debug=1

[Log]
; NOTE: use a semicolon for comments, not #. PHP's parse_ini_file()
; (used by the dashboard) throws a syntax error on lines starting with #.
; Keep comments on their own lines -- never after a value on the same line,
; or USRP2M17 reads the comment as part of the value.
DisplayLevel=1
FileLevel=1
FilePath=.
FileRoot=USRP2M17
```

A quick pick for a first reflector to put in `Address=` / `Name=` while testing: `M17-M17` at `107.191.121.105`, module `C`. Once the DTMF switcher (below) is set up, it manages these two fields for you and you can leave them blank or point anywhere to start.

Note on your `Callsign=` value: the dashboard reads this value back out of this same file to decide which decoded transmissions are "yours" versus everyone else's (see section 9). Keep it accurate and don't leave it blank once you're running the dashboard.

5. systemd Service

`/etc/systemd/system/usrp2m17.service` :

```
[Unit]
```

```
Description=USRP2M17 Bridge for AllStar node <PRIVATE_NODE>
After=network.target asterisk.service

[Service]
Type=simple
User=<PI_USER>
WorkingDirectory=/home/<PI_USER>/MMDVM_CM/USRP2M17
ExecStart=/home/<PI_USER>/MMDVM_CM/USRP2M17/USRP2M17
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl daemon-reload
```

Important: leave this service **disabled** at boot (don't run `systemctl enable`). Only start it manually or via the DTMF switcher below. If it's enabled, a fresh reboot will silently reconnect to whatever reflector was last saved in the `.ini` file -- which is surprising behavior for anyone expecting a clean boot with nothing connected.

To start it manually when you want to test:

```
sudo systemctl start usrp2m17
sudo systemctl status usrp2m17
```

6. DTMF Reflector Switcher

This lets you change reflectors from the radio itself, by dialing DTMF codes.

6a. Reflector list

`/etc/asterisk/m17_reflectors.conf` -- four well-known public M17 reflectors are listed below as a starting point:

```
[M17_M17]
modemaster=1
name=M17-M17
module=C
address=107.191.121.105

[M17_KCW]
modemaster=2
name=M17-KCW
module=B
address=172.234.217.28

[M17_DVS]
modemaster=3
name=M17-DVS
module=A
address=150.136.69.97

[M17_117]
modemaster=4
name=M17-117
module=B
address=96.224.250.24
```

To add a fifth (or more) reflector: append another section following the exact same pattern, incrementing `modemaster` by one each time:

```
[M17_MYNEW]
modemaster=5
name=M17-MYNEW
```

```
module=A
address=203.0.113.42
```

`modemaster=` is the digit you dial after `*BB`; `name=` is the reflector's name as shown on the dashboard; `module=` is the module letter on that reflector you want to join; `address=` is the reflector server's IP or hostname. You can look up other reflectors' addresses on the public M17 reflector list (search "M17 reflector list" -- several community-maintained lists exist) or the reflector's own web dashboard if it has one.

Note: some reflectors (like M17-DVS in the example above) are actually multi-mode gateways (e.g. DVSwitch-style bridges to DMR/YSF/etc.) rather than plain M17-to-M17 reflectors. The dashboard's activity classification (section 9) is specifically designed to handle these correctly.

6b. The switcher script

`/usr/local/bin/m17_link.sh :`

```
#!/bin/bash
set +H

NODE_ID=<RADIO_NODE>
LINKED_NODE_ID=<PRIVATE_NODE>
CONFIG_FILE=/etc/asterisk/m17_reflectors.conf
INI_FILE=/etc/USRP2M17.ini
LOG_FILE=/var/log/asterisk/m17_link.log

echo "$(date): Executing with $1" >> "$LOG_FILE"

# Dialing *BB1 on the radio: the first B is the rpt.conf function code,
# and "B1" becomes the extension passed here. Strip the leading "B" so
# only the digits remain.
DTMF=$1
DTMF=${DTMF:1}

if [[ -z "$DTMF" ]]; then
    echo "$(date): Invalid/empty DTMF command" >> "$LOG_FILE"
    exit 1
fi

disconnect_reflector() {
    echo "$(date): Stopping usrp2m17 (leaving M17 network)" >> "$LOG_FILE"
    systemctl stop usrp2m17
}

disconnect_node() {
    echo "$(date): Unlinking $NODE_ID from $LINKED_NODE_ID" >> "$LOG_FILE"
    asterisk -rx "rpt cmd $NODE_ID ilink 1 $LINKED_NODE_ID"
}

if [[ "$DTMF" == "00" ]]; then
    disconnect_reflector
    disconnect_node
    echo "$(date): Full disconnect complete" >> "$LOG_FILE"
    exit 0
fi

if [[ "$DTMF" == "0" ]]; then
    disconnect_reflector
    echo "$(date): Reflector disconnected, link to $LINKED_NODE_ID left intact" >> "$LOG_FILE"
    exit 0
fi

MODEMASTER="$DTMF"

SECTION=$(awk -v mm="$MODEMASTER" '
/^\[.*\]/ { section=$0; in_section=0 }
$0 ~ "modemaster="mm"$" { in_section=1 }
in_section { print section; exit }
' "$CONFIG_FILE")
```

```

if [[ -z "$SECTION" ]]; then
    echo "$(date): No reflector found for modemaster $MODEMASTER" >> "$LOG_FILE"
    exit 1
fi

NAME=$(awk -v section="$SECTION" '
$0 == section { in_section=1; next }
in_section && /^\[.*\]/ { exit }
in_section && $1 ~ /^name=/ { split($0, a, "="); print a[2]; exit }
' "$CONFIG_FILE")

MODULE=$(awk -v section="$SECTION" '
$0 == section { in_section=1; next }
in_section && /^\[.*\]/ { exit }
in_section && $1 ~ /^module=/ { split($0, a, "="); print a[2]; exit }
' "$CONFIG_FILE")

ADDRESS=$(awk -v section="$SECTION" '
$0 == section { in_section=1; next }
in_section && /^\[.*\]/ { exit }
in_section && $1 ~ /^address=/ { split($0, a, "="); print a[2]; exit }
' "$CONFIG_FILE")

# Make sure the radio node is linked to the private M17 node. If it is
# linked to something else, drop those links first (ilink 6 = disconnect all).
CURRENT_CONNECTION=$(asterisk -rx "rpt lstats $NODE_ID" | grep "ESTABLISHED")
if [[ -n "$CURRENT_CONNECTION" && "$CURRENT_CONNECTION" != "$LINKED_NODE_ID" ]]; then
    echo "$(date): Disconnecting from other linked nodes first" >> "$LOG_FILE"
    asterisk -rx "rpt cmd $NODE_ID ilink 6 1"
fi
if [[ "$CURRENT_CONNECTION" != "$LINKED_NODE_ID" ]]; then
    echo "$(date): Linking $NODE_ID to $LINKED_NODE_ID" >> "$LOG_FILE"
    asterisk -rx "rpt fun $NODE_ID *$LINKED_NODE_ID"
    sleep 2
fi

# Scoped edit: only touch Address=Name= within the [M17 Network] section.
# An earlier, unscoped `sed` version accidentally corrupted the SEPARATE
# [USRP Network] Address= line too, since both sections have a line
# starting with "Address=".
awk -v addr="$ADDRESS" -v name="$NAME" -v mod="$MODULE" '
BEGIN { in_m17 = 0 }
/^\[M17 Network\]/ { in_m17 = 1; print; next }
/^\[.*\]/ { in_m17 = 0; print; next }
in_m17 && /^Address=/ { print "Address=" addr; next }
in_m17 && /^Name=/ { print "Name=" name " " mod; next }
{ print }
' "$INI_FILE" > /etc/USRP2M17.ini.tmp && mv /etc/USRP2M17.ini.tmp "$INI_FILE"

systemctl restart usrp2m17

echo "$(date): Tuned to $NAME $MODULE ($ADDRESS)" >> "$LOG_FILE"
exit 0

```

```

sudo chmod +x /usr/local/bin/m17_link.sh
sudo touch /var/log/asterisk/m17_link.log
sudo chown asterisk:asterisk /var/log/asterisk/m17_link.log

```

Every DTMF command is logged to `/var/log/asterisk/m17_link.log`, which is the first place to look if a code seems to do nothing.

6c. Wire it up to DTMF

`/etc/sudoers.d/m17_link` (grant the whole script, not individual commands piecemeal -- piecemeal sudoers rules break every time the script gains a new command):

```
asterisk ALL=(root) NOPASSWD: /usr/local/bin/m17_link.sh
```

`/etc/asterisk/rpt.conf`, in the `[functions]` section -- pick a free function code. This example uses `B`. If `B` is already taken on your system (for example by a DVSwitch mode-select), pick another letter and use that same letter in the `extensions.conf` pattern below.

```
B = autopatchup,context=m17_link,noct=1,farenddisconnect=1,dialtime=60000,quiet=1
```

`/etc/asterisk/extensions.conf`:

```
[m17_link]
exten => _B.,1,System(sudo /usr/local/bin/m17_link.sh ${EXTEN})
exten => _B.,n,Hangup
```

(How the codes work: dialing `*BB1` triggers function `B` from `rpt.conf`, and the digits dialed after it, `B1`, become the extension. That extension has to start with `B` to match the `_B.` pattern, which is why every code starts with `*BB`. `${EXTEN}` passes `B1` to the script, and the script strips the leading `B`.)

```
sudo systemctl restart asterisk
```

Resulting DTMF codes on the radio, using the example reflector list above and function code `B` (dialed as `*BB` + digits):

- `*BB1` -> M17-M17 C, `*BB2` -> M17-KCW B, `*BB3` -> M17-DVS A, `*BB4` -> M17-117 B
- `*BB0` -> disconnect reflector, keep node linked
- `*BB00` -> full disconnect (also unlinks the radio node from the private node)

7. The Web Dashboard

A four-file PHP app (`lib.php`, `activity_fragment.php`, `index.php`, `disconnect.php`), styled after the classic Pi-Star/DVSwitch dashboard look, that live-decodes M17 traffic from the USRP2M17 debug log and shows current status, a live TX/RX indicator, and recent activity correctly split between your own transmissions and everyone else's.

7a. Prerequisites

```
sudo apt install php php-mbstring apache2 # or your preferred web server + PHP
sudo usermod -aG systemd-journal www-data # lets the dashboard read the journal without needing sudo journalctl
sudo mkdir -p /var/lib/m17dashboard
sudo touch /var/lib/m17dashboard/debug.log
sudo touch /var/lib/m17dashboard/pipeline.lock
sudo chown -R www-data:www-data /var/lib/m17dashboard
sudo mkdir -p /var/www/html/m17
sudo systemctl restart apache2 # so www-data picks up the new systemd-journal group
```

`/etc/sudoers.d/m17_dashboard`:

```
www-data ALL=(root) NOPASSWD: /bin/systemctl is-active usrp2m17
www-data ALL=(root) NOPASSWD: /bin/systemctl stop usrp2m17
```

(The second line lets the dashboard's "Disconnect Reflector" button in step 7e actually stop the bridge, not just report its status -- this is a standard part of the dashboard, not an optional extra, since it's your only way to kill a stuck-PTT or runaway transmission without physical access to the radio.)

7b. `lib.php` -- shared decode/parsing logic

Save as `/var/www/html/m17/lib.php`:

```
<?php
/* Shared M17 dashboard logic -- included by both index.php and
   activity_fragment.php so the two never drift out of sync. */

$STATE_DIR = '/var/lib/m17dashboard';
$GW_LOG     = "$STATE_DIR/gateway_activity.log";
```

```

$LOCAL_LOG = "$STATE_DIR/local_activity.log";
$STATE_FILE= "$STATE_DIR/last_check.txt";
$DEBUG_LOG = "$STATE_DIR/debug.log";
$LIVE_STATE_FILE = "$STATE_DIR/live_state.txt";
$MAX_HISTORY = 200;
$DISPLAY_ROWS = 25;
const LIVE_STATE_ACTIVE_SECONDS = 3;

function read_live_state($file) {
    $raw = trim(@file_get_contents($file) ?: '');
    if ($raw === '') return 'idle';
    [$dir, $epoch] = array_pad(explode('|', $raw), 2, '');
    $epoch = (int) $epoch;
    if (!$epoch || (time() - $epoch) > LIVE_STATE_ACTIVE_SECONDS) return 'idle';
    return $dir === 'sent' ? 'tx' : ($dir === 'recv' ? 'rx' : 'idle');
}

$M17_ALPHABET = " ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-./.";

function m17_decode_callsign($bytes) {
    global $M17_ALPHABET;
    if (count($bytes) != 6) return null;
    $n = 0;
    foreach ($bytes as $b) { $n = $n * 256 + $b; }
    if ($n == 0) return null;
    if ($n == 0xFFFFFFFF) return "@ALL";
    if ($n > 262143999999999) return null;
    $out = "";
    while ($n > 0) { $out .= $M17_ALPHABET[$n % 40]; $n = intdiv($n, 40); }
    return trim($out);
}

function extract_base_callsign($callsign) {
    if ($callsign === '@ALL') return null;
    $tokens = preg_split('/[\\s\\-\\/]+/', trim($callsign));
    foreach ($tokens as $t) {
        if (strlen($t) > 1) return strtoupper($t);
    }
    return null;
}

$PREFIX_TABLE = [
    'KL'=>'US', 'KH'=>'US', 'KP'=>'US', 'AL'=>'US', 'NL'=>'US', 'WL'=>'US',
    'K'=>'US', 'W'=>'US', 'N'=>'US',
    'AA'=>'US', 'AB'=>'US', 'AC'=>'US', 'AD'=>'US', 'AE'=>'US', 'AF'=>'US', 'AG'=>'US', 'AH'=>'US', 'AI'=>'US', 'AJ'=>'US', 'AK'=>'US',
    'A2'=>'BW', 'A3'=>'TO', 'A4'=>'OM', 'A5'=>'BT', 'A6'=>'AE', 'A7'=>'QA', 'A8'=>'LR', 'A9'=>'BH',
    'AM'=>'ES', 'AN'=>'ES', 'AO'=>'ES',
    'AP'=>'PK', 'AQ'=>'PK', 'AR'=>'PK', 'AS'=>'PK',

    'VE'=>'CA', 'VA'=>'CA', 'VO'=>'CA', 'VY'=>'CA', 'CF'=>'CA', 'CG'=>'CA', 'CY'=>'CA', 'CZ'=>'CA', 'XJ'=>'CA', 'XK'=>'CA', 'XL'=>'CA', 'XM'=>'CA', 'XN'=>'CA', 'XO'
=>'CA',
    'XE'=>'MX', 'XF'=>'MX', 'XG'=>'MX', 'XH'=>'MX', 'XI'=>'MX',
    'G'=>'GB', 'M'=>'GB', 'ZE'=>'GB',
    'EI'=>'IE', 'EJ'=>'IE',
    'F'=>'FR', 'TM'=>'FR', 'TH'=>'FR',

    'DL'=>'DE', 'DA'=>'DE', 'DB'=>'DE', 'DC'=>'DE', 'DD'=>'DE', 'DF'=>'DE', 'DG'=>'DE', 'DH'=>'DE', 'DJ'=>'DE', 'DK'=>'DE', 'DM'=>'DE', 'DO'=>'DE', 'DP'=>'DE', 'DQ'
=>'DE', 'DR'=>'DE',
    'I'=>'IT', 'IZ'=>'IT', 'IK'=>'IT', 'IW'=>'IT',
    'EA'=>'ES', 'EB'=>'ES', 'EC'=>'ES', 'ED'=>'ES', 'EE'=>'ES', 'EF'=>'ES', 'EG'=>'ES', 'EH'=>'ES',
    'CT'=>'PT', 'CR'=>'PT',
    'PA'=>'NL', 'PB'=>'NL', 'PC'=>'NL', 'PD'=>'NL', 'PE'=>'NL', 'PF'=>'NL', 'PG'=>'NL', 'PH'=>'NL', 'PI'=>'NL',
    'ON'=>'BE', 'OO'=>'BE', 'OP'=>'BE', 'OQ'=>'BE', 'OR'=>'BE', 'OS'=>'BE', 'OT'=>'BE',
    'SM'=>'SE', 'SA'=>'SE', 'SB'=>'SE', 'SC'=>'SE', 'SD'=>'SE', 'SE'=>'SE', 'SF'=>'SE', 'SG'=>'SE', 'SH'=>'SE', 'SI'=>'SE', 'SJ'=>'SE', 'SK'=>'SE', 'SL'=>'SE',
    'LA'=>'NO', 'LB'=>'NO', 'LC'=>'NO', 'LD'=>'NO', 'LE'=>'NO', 'LF'=>'NO', 'LG'=>'NO', 'LH'=>'NO', 'LI'=>'NO', 'LJ'=>'NO', 'LN'=>'NO',
    'OH'=>'FI', 'OF'=>'FI', 'OG'=>'FI', 'OI'=>'FI', 'OJ'=>'FI',
    'OZ'=>'DK', 'OU'=>'DK', 'OV'=>'DK', 'OW'=>'DK', 'OX'=>'DK', 'OY'=>'DK',
    'HB'=>'CH', 'HE'=>'CH',

```

```

'OE'=>'AT',
'OK'=>'CZ','OL'=>'CZ',
'OM'=>'SK',
'HA'=>'HU','HG'=>'HU',
'YO'=>'R0','YP'=>'R0','YQ'=>'R0','YR'=>'R0',
'LZ'=>'BG',
'SV'=>'GR','SX'=>'GR','SY'=>'GR','SZ'=>'GR','J4'=>'GR',
'SP'=>'PL','SN'=>'PL','SO'=>'PL','SQ'=>'PL','SR'=>'PL','3Z'=>'PL',
'UR'=>'UA','US'=>'UA','UT'=>'UA','UU'=>'UA','UW'=>'UA','UX'=>'UA','UY'=>'UA','UZ'=>'UA','EM'=>'UA','EN'=>'UA','E0'=>'UA','EP'=>'UA',

'RA'=>'RU','RN'=>'RU','R0'=>'RU','RQ'=>'RU','RU'=>'RU','RV'=>'RU','RW'=>'RU','RX'=>'RU','RZ'=>'RU','R2'=>'RU','R3'=>'RU','R4'=>'RU','R5'=>'RU','R6'
=>'RU','R7'=>'RU','R8'=>'RU','R9'=>'RU','UA'=>'RU','UB'=>'RU','UC'=>'RU',

'JA'=>'JP','JE'=>'JP','JF'=>'JP','JG'=>'JP','JH'=>'JP','JI'=>'JP','JJ'=>'JP','JK'=>'JP','JL'=>'JP','JM'=>'JP','JN'=>'JP','JO'=>'JP','JP'=>'JP','JQ'
=>'JP','JR'=>'JP','JS'=>'JP',
'BV'=>'TW','BX'=>'TW','BM'=>'TW','B0'=>'TW','BP'=>'TW','BQ'=>'TW',
'BA'=>'CN','BD'=>'CN','BG'=>'CN','BH'=>'CN','BI'=>'CN','BJ'=>'CN','BL'=>'CN','BN'=>'CN','BT'=>'CN','BY'=>'CN','BZ'=>'CN',
'HL'=>'KR','DS'=>'KR','D7'=>'KR','D8'=>'KR','D9'=>'KR','6K'=>'KR','6L'=>'KR','6M'=>'KR','6N'=>'KR','7K'=>'KR','7L'=>'KR','7M'=>'KR',
'VU'=>'IN','VT'=>'IN','8T'=>'IN','AT'=>'IN','AU'=>'IN','AV'=>'IN','AW'=>'IN',
'VK'=>'AU','AX'=>'AU',
'ZL'=>'NZ','ZM'=>'NZ','ZK'=>'NZ',
'ZS'=>'ZA','ZR'=>'ZA','ZT'=>'ZA','ZU'=>'ZA',

'PY'=>'BR','PP'=>'BR','PQ'=>'BR','PR'=>'BR','PS'=>'BR','PT'=>'BR','PU'=>'BR','PV'=>'BR','PW'=>'BR','ZV'=>'BR','ZW'=>'BR','ZX'=>'BR','ZY'=>'BR','ZZ'
=>'BR',
'LU'=>'AR','L0'=>'AR','LP'=>'AR','LQ'=>'AR','LR'=>'AR','LS'=>'AR','LT'=>'AR','LV'=>'AR','LW'=>'AR','AY'=>'AR','AZ'=>'AR',
'CE'=>'CL','CA'=>'CL','CB'=>'CL','CC'=>'CL','CD'=>'CL','XQ'=>'CL','XR'=>'CL','3G'=>'CL',
'TA'=>'TR','TB'=>'TR','TC'=>'TR',
'4X'=>'IL','4Z'=>'IL',
'HS'=>'TH','E2'=>'TH',
'DU'=>'PH','DV'=>'PH','DW'=>'PH','DX'=>'PH','DY'=>'PH','DZ'=>'PH','4D'=>'PH','4E'=>'PH','4F'=>'PH','4G'=>'PH','4H'=>'PH','4I'=>'PH',
'YB'=>'ID','YC'=>'ID','YD'=>'ID','YE'=>'ID','YF'=>'ID','YG'=>'ID','YH'=>'ID',
];
$ISO_TO_NAME = [
'US'=>'USA','CA'=>'Canada','MX'=>'Mexico','GB'=>'UK','IE'=>'Ireland',
'FR'=>'France','DE'=>'Germany','IT'=>'Italy','ES'=>'Spain','PT'=>'Portugal',
'NL'=>'Netherlands','BE'=>'Belgium','SE'=>'Sweden','NO'=>'Norway','FI'=>'Finland',
'DK'=>'Denmark','CH'=>'Switzerland','AT'=>'Austria','CZ'=>'Czechia','SK'=>'Slovakia',
'HU'=>'Hungary','RO'=>'Romania','BG'=>'Bulgaria','GR'=>'Greece','PL'=>'Poland',
'UA'=>'Ukraine','RU'=>'Russia','JP'=>'Japan','TW'=>'Taiwan','CN'=>'China',
'KR'=>'South Korea','IN'=>'India','AU'=>'Australia','NZ'=>'New Zealand','ZA'=>'South Africa',
'BR'=>'Brazil','AR'=>'Argentina','CL'=>'Chile','TR'=>'Turkey','IL'=>'Israel',
'TH'=>'Thailand','PH'=>'Philippines','ID'=>'Indonesia',
'BW'=>'Botswana','TO'=>'Tonga','OM'=>'Oman','BT'=>'Bhutan','AE'=>'UAE',
'QA'=>'Qatar','LR'=>'Liberia','BH'=>'Bahrain','PK'=>'Pakistan',
];
function callsign_flag_img($callsign) {
global $PREFIX_TABLE, $ISO_TO_NAME;
if ($callsign === null) return '';
for ($len = 3; $len >= 1; $len--) {
    $p = substr($callsign, 0, $len);
    if (isset($PREFIX_TABLE[$p])) {
        $iso = strtolower($PREFIX_TABLE[$p]);
        $name = $ISO_TO_NAME[$PREFIX_TABLE[$p]] ?? $PREFIX_TABLE[$p];
        // Requesting an 80x60 source image (while still DISPLAYING it at
        // 20x15 via the width/height attributes below) keeps the flag
        // sharp even when the page is pinch-zoomed on a phone -- the
        // original 20x15 source image looked visibly blurry once
        // enlarged, since there was no extra pixel data to zoom into.
        return '';
    }
}
return '';
}

function qrz_link($callsign) {
    $base = extract_base_callsign($callsign);

```

```

    if ($base === null) return htmlspecialchars($callsign);
    return '<a href="https://www.qrz.com/db/' . urlencode($base) . '" target="_blank" style="color:#0645AD;">'
        . htmlspecialchars($callsign) . '</a>';
}

function callsign_core($s) {
    if ($s === null) return null;
    if ($s === '@ALL') return '@ALL';
    $tokens = preg_split('/\s+/', trim($s));
    $score = '';
    foreach ($tokens as $t) { if (strlen($t) > strlen($score)) $score = $t; }
    return $score;
}

function looks_like_callsign($s) {
    if ($s === null) return false;
    $score = callsign_core($s);
    if ($score === '@ALL') return true;
    if (preg_match('/^M17-[A-Z0-9]{1,10}$/i', $score)) return true;
    if (preg_match('/^[A-Z]{1,2}[0-9][A-Z0-9]{1,4}$/i', $score)) return true;
    return false;
}

function flush_frame($hexbuf, $cur_time, $direction, &$new_events) {
    if (count($hexbuf) < 18) return;
    if ($hexbuf[0] != 0x4D || $hexbuf[1] != 0x31 || $hexbuf[2] != 0x37 || $hexbuf[3] != 0x20) return;
    $src_cs = m17_decode_callsign(array_slice($hexbuf, 12, 6));
    $dst_cs = m17_decode_callsign(array_slice($hexbuf, 6, 6));
    if ($src_cs === null) return;
    if (!looks_like_callsign($src_cs)) return;
    if ($dst_cs !== null && !looks_like_callsign($dst_cs)) return;
    $new_events[] = ['time' => $cur_time, 'callsign' => $src_cs, 'target' => $dst_cs ?? '?', 'direction' => $direction];
}

const SESSION_MERGE_GAP_SECONDS = 5;

function collapse_batch($events) {
    $sessions = []; $prev = null;
    foreach ($events as $e) {
        $gap = 999;
        if ($prev) {
            $prev_epoch = strtotime($prev['last_time']);
            $cur_epoch = strtotime($e['time']);
            $gap = ($prev_epoch !== false && $cur_epoch !== false) ? abs($cur_epoch - $prev_epoch) : 999;
        }
        if ($prev && $prev['callsign'] === $e['callsign'] && $prev['target'] === $e['target'] && $gap <= SESSION_MERGE_GAP_SECONDS) {
            $prev['count']++; $prev['last_time'] = $e['time'];
        } else { if ($prev) $sessions[] = $prev; $prev = $e + ['count' => 1, 'last_time' => $e['time']]; }
    }
    if ($prev) $sessions[] = $prev;
    return $sessions;
}

function append_persistent($logfile, $sessions, $max_history) {
    if (empty($sessions)) return;
    $existing = file_exists($logfile) ? file($logfile, FILE_IGNORE_NEW_LINES) : [];
    foreach ($sessions as $s) {
        $line = $s['time'] . '|' . $s['callsign'] . '|' . $s['target'] . '|' . $s['count'];
        if (!empty($existing)) {
            $lastParts = explode('|', end($existing));
            $last_epoch = strtotime($lastParts[0]);
            $new_epoch = strtotime($s['time']);
            $gap = ($last_epoch !== false && $new_epoch !== false) ? abs($new_epoch - $last_epoch) : 999;
            if ($lastParts[1] === $s['callsign'] && $lastParts[2] === $s['target'] && $gap <= SESSION_MERGE_GAP_SECONDS) {
                $mergedCount = (int)$lastParts[3] + $s['count'];
                $existing[count($existing) - 1] = $s['time'] . '|' . $s['callsign'] . '|' . $s['target'] . '|' . $mergedCount;
                continue;
            }
        }
    }
}

```

```

    }
    $existing[] = $line;
}
if (count($existing) > $max_history) $existing = array_slice($existing, -$max_history);
file_put_contents($logfile, implode("\n", $existing) . "\n");
}

// Local RF Activity keeps only the most recent session PER CALLSIGN (one
// line each) instead of accumulating full history. A brand-new session for
// a callsign replaces its old line outright; a continuation of the same
// PTT (same target, within SESSION_MERGE_GAP_SECONDS) still merges into it.
function append_persistent_last_only($logfile, $sessions) {
    if (empty($sessions)) return;
    $existing = file_exists($logfile) ? file($logfile, FILE_IGNORE_NEW_LINES) : [];

    $by_callsign = [];
    foreach ($existing as $i => $l) {
        $p = explode('|', $l);
        if (count($p) < 4) continue;
        $by_callsign[$p[1]] = $i;
    }

    foreach ($sessions as $s) {
        $line = $s['time'] . '|' . $s['callsign'] . '|' . $s['target'] . '|' . $s['count'];
        if (isset($by_callsign[$s['callsign']])) {
            $idx = $by_callsign[$s['callsign']];
            $p = explode('|', $existing[$idx]);
            $last_epoch = strtotime($p[0]);
            $new_epoch = strtotime($s['time']);
            $gap = ($last_epoch !== false && $new_epoch !== false) ? abs($new_epoch - $last_epoch) : 999;
            if ($p[2] === $s['target'] && $gap <= SESSION_MERGE_GAP_SECONDS) {
                $mergedCount = (int)$p[3] + $s['count'];
                $existing[$idx] = $s['time'] . '|' . $s['callsign'] . '|' . $s['target'] . '|' . $mergedCount;
            } else {
                $existing[$idx] = $line;
            }
        } else {
            $existing[] = $line;
            $by_callsign[$s['callsign']] = count($existing) - 1;
        }
    }

    file_put_contents($logfile, implode("\n", $existing) . "\n");
}

// Filters out sub-threshold "transmissions" at DISPLAY time (not storage
// time -- nothing is permanently discarded). Tuned down from an initial
// 1.5s to 0.3s after finding that some genuinely-short but real keyups
// (0.4-0.7s test taps) were being hidden along with actual decode noise.
// Sub-0.3s frames are almost always a marginal/weak-signal decode glitch --
// e.g. a single frame whose destination field flips between two different
// values within the same second, which is a telltale sign of a corrupted
// receive rather than an intentional PTT -- not a perfect filter, but a
// reasonable line given the data available.
const MIN_DISPLAY_DURATION_SECONDS = 0.3;

function read_display_rows($logfile, $n) {
    if (!file_exists($logfile)) return [];
    $lines = file($logfile, FILE_IGNORE_NEW_LINES);
    $rows = [];
    foreach (array_reverse($lines) as $l) {
        $p = explode('|', $l);
        if (count($p) < 4) continue;
        $count = (int)$p[3];
        if ($count * 0.04 < MIN_DISPLAY_DURATION_SECONDS) continue;
        $rows[] = ['time' => $p[0], 'callsign' => $p[1], 'target' => $p[2], 'count' => $count];
        if (count($rows) >= $n) break;
    }
}

```

```

return $rows;
}

function run_pipeline() {
    global $STATE_FILE, $DEBUG_LOG, $GW_LOG, $LOCAL_LOG, $MAX_HISTORY;

    $lock_fp = fopen('/var/lib/m17dashboard/pipeline.lock', 'c');
    if ($lock_fp === false || !flock($lock_fp, LOCK_EX | LOCK_NB)) {
        if ($lock_fp !== false) fclose($lock_fp);
        return;
    }

    $cursor = trim(@file_get_contents($STATE_FILE));
    if ($cursor !== '' && strpos($cursor, 's=') === 0) {
        $cmd = "journalctl -u usrp2m17 --after-cursor=" . escapeshellarg($cursor) . " --show-cursor --no-pager 2>&1";
    } else {
        // No valid cursor yet (first run, or state file predates this change) --
        // fall back to a short relative window just this once to bootstrap one.
        $cmd = "journalctl -u usrp2m17 --since " . escapeshellarg('-30 seconds') . " --show-cursor --no-pager 2>&1";
    }

    $raw = shell_exec($cmd);
    $lines = $raw ? explode("\n", $raw) : [];

    // The last line is "-- cursor: s=..." (no trailing "--") -- pull it out and persist it,
    // and don't feed it to the frame parser below.
    $new_cursor = null;
    while (!empty($lines) && trim(end($lines)) === '') { array_pop($lines); }
    if (!empty($lines) && preg_match('/^-- cursor: (.+)/', trim(end($lines)), $cm)) {
        $new_cursor = $cm[1];
        array_pop($lines);
    }

    // Debug: record what we actually got, so a "nothing decoded" state is diagnosable
    file_put_contents($DEBUG_LOG, gmdate('Y-m-d H:i:s') . " lines=" . count($lines) . " cursor=" . ($new_cursor ?? 'none') . "\n", FILE_APPEND);

    $new_events = [];
    $cur_direction = null; $cur_time = null; $hexbuf = [];

    foreach ($lines as $line) {
        // Capture as (Month)(Day)(Time) then redisplay as Time Month Day
        if (preg_match('/^(w+)(\d+)(\d\d:\d\d:\d\d).*M17 Network Data (Received|Sent)/', $line, $m)) {
            if ($cur_direction) flush_frame($hexbuf, $cur_time, $cur_direction, $new_events);
            $cur_time = "{$m[3]} {$m[1]} {$m[2]}";
            $cur_direction = ($m[4] === 'Received') ? 'recv' : 'sent';
            $hexbuf = [];
            continue;
        }
        if ($cur_direction && preg_match('/:s*[DM]:s*[d\-\+][d:.\+][0-9A-Fa-f]{4}:s+([0-9A-Fa-f]{2,60})\s+\/', $line, $m2)) {
            foreach (preg_split('/\s+/', trim($m2[1])) as $bh) {
                if (preg_match('/^[0-9A-Fa-f]{2}$/', $bh)) $hexbuf[] = hexdec($bh);
            }
        }
    }
    if ($cur_direction) flush_frame($hexbuf, $cur_time, $cur_direction, $new_events);

    file_put_contents($DEBUG_LOG, gmdate('Y-m-d H:i:s') . " decoded_events=" . count($new_events) . "\n", FILE_APPEND);

    // Classify by comparing the DECODED SRC callsign against our own
    // configured callsign -- NOT by trusting the raw "Sent"/"Received"
    // text in the USRP2M17 log line (see notes on bugs below).
    $ini_for_classify = @parse_ini_file('/etc/USRP2M17.ini', true);
    $own_callsign = extract_base_callsign($ini_for_classify['M17 Network']['Callsign'] ?? '');

    // Gateway Activity is the full chronological log of EVERYTHING heard --
    // remote stations AND your own transmissions -- matching how WPSD's own
    // "last heard" table works (it does not filter out the local radio).
    // Local RF Activity is separately filtered to only your own callsign,

```

```

// and is deduplicated down to one row per callsign by
// append_persistent_last_only() below.
$gw_sessions_new = collapse_batch($new_events);
$local_sessions_new = collapse_batch(array_values(array_filter($new_events, fn($e) => extract_base_callsign($e['callsign']) ===
$own_callsign)));

// Record the most recent frame's direction for the live TX/RX indicator.
if (!empty($new_events)) {
    $last_dir = end($new_events)['direction'];
    global $LIVE_STATE_FILE;
    file_put_contents($LIVE_STATE_FILE, $last_dir . '|' . time());
}

// Filter out sub-threshold noise BEFORE it reaches the per-callsign
// Local RF Activity store. Gateway Activity can safely let display-time
// filtering handle this (a noise row just sits harmlessly among real
// history), but Local RF Activity keeps only ONE row per callsign --
// so a stray noise blip (e.g. triggered by the audio path reconnecting
// during a reflector switch) would silently REPLACE a real, meaningful
// session instead of just being hidden. Filtering here means noise
// never gets the chance to overwrite something real in the first place.
$local_sessions_new = array_values(array_filter(
    $local_sessions_new,
    fn($s) => $s['count'] * 0.04 >= MIN_DISPLAY_DURATION_SECONDS
));

append_persistent($GW_LOG, $gw_sessions_new, $MAX_HISTORY);
append_persistent_last_only($LOCAL_LOG, $local_sessions_new);
if ($new_cursor !== null) {
    file_put_contents($STATE_FILE, $new_cursor);
}

flock($lock_fp, LOCK_UN);
fclose($lock_fp);
}

// Converts a "H:i:s M j" (no year) timestamp string into a short relative
// "X mins ago" style string, the same idea as WPSD's own timeago() helper --
// but only used for Local RF Activity, since that table shows only the
// single latest session per callsign and a plain count of time since it
// happened is genuinely useful there. Not applied to Gateway Activity,
// which is a scrolling multi-station log where every row already has its
// own absolute timestamp visible.
function time_ago_string($time_str) {
    // Our stored times are naive local-clock strings (as journalctl printed
    // them, in the system's own local timezone) with no explicit offset. If
    // PHP's default timezone setting doesn't match the system's actual local
    // timezone, strtotime() silently misinterprets the string and every
    // "ago" value comes out wrong by a fixed multi-hour offset -- even a
    // transmission from a few seconds ago reading as "4 hours ago". Pinning
    // the same explicit system timezone here removes that ambiguity.
    static $tz_set = false;
    if (!$tz_set) {
        $sys_tz = trim(@shell_exec('cat /etc/timezone 2>/dev/null'));
        date_default_timezone_set($sys_tz !== '' ? $sys_tz : 'America/Toronto');
        $tz_set = true;
    }
    $epoch = strtotime($time_str);
    if ($epoch === false) return '';
    $diff = time() - $epoch;
    if ($diff < 0) $diff = 0;
    if ($diff < 60) return $diff . ' sec' . ($diff == 1 ? '' : 's') . ' ago';
    $mins = intval($diff, 60);
    if ($mins < 60) return $mins . ' min' . ($mins == 1 ? '' : 's') . ' ago';
    $hours = intval($mins, 60);
    if ($hours < 24) return $hours . ' hour' . ($hours == 1 ? '' : 's') . ' ago';
    $days = intval($hours, 24);
    return $days . ' day' . ($days == 1 ? '' : 's') . ' ago';
}

```

```

}

function rows_to_html($rows, $show_ago = false) {
    if (empty($rows)) {
        return '<tr><td colspan="5" style="text-align:center;">No recent activity</td></tr>';
    }
    $html = '';
    foreach ($rows as $s) {
        $base = extract_base_callsign($s['callsign']);
        $dur = round($s['count'] * 0.04, 1);
        if ($show_ago) {
            $ago = time_ago_string($s['time']);
            $dur_cell = $dur . 's' . ($ago !== '' ? ' <span style="color:#888;font-size:10px;">(' . $ago . ')</span>' : '');
        } else {
            $dur_cell = $dur;
        }
        $html .= '<tr>'
            . '<td>' . htmlspecialchars($s['time']) . '</td>'
            . '<td>' . qrz_link($s['callsign']) . '</td>'
            . '<td class="text-center">' . callsign_flag_img($base) . '</td>'
            . '<td>' . htmlspecialchars($s['target']) . '</td>'
            . '<td>' . $dur_cell . '</td>'
            . '</tr>';
    }
    return $html;
}

```

7c. activity_fragment.php -- JSON endpoint polled every 2 seconds

Save as `/var/www/html/m17/activity_fragment.php` :

```

<?php
require_once __DIR__ . '/lib.php';
run_pipeline();

$svc_active = trim(shell_exec('sudo systemctl is-active usrp2m17 2>/dev/null'));
$svc_ok = ($svc_active === 'active');

$gateway_rows = read_display_rows($GW_LOG, $DISPLAY_ROWS);
$local_rows = read_display_rows($LOCAL_LOG, $DISPLAY_ROWS);

// Reflector/Module/Address must be re-sent on every poll too -- these were
// previously only rendered once at initial page load, so they'd go stale
// if the reflector changed while the page stayed open (e.g. iOS "Add to
// Home Screen" apps resume from a suspended state rather than reloading).
$ini = @parse_ini_file('/etc/USR2M17.ini', true);
$m17_name = $ini['M17 Network']['Name'] ?? '';
$m17_address = $ini['M17 Network']['Address'] ?? '';
list($room_name, $room_module) = array_pad(explode(' ', $m17_name), 2, 2, '');

$REFLECTOR_DASHBOARDS = [
    'M17-M17' => 'http://m17.openquad.net',
    'M17-117' => 'http://96.224.250.24:8087/#/', // confirmed working -- needs this specific port + route, plain IP alone does not work
    'M17-KCW' => 'http://m17.kansascitywide.com', // confirmed working
    'M17-DVS' => 'https://150.136.69.97/?conf=M17Reflector', // confirmed working (DVSwitch multi-mode reflector's own dashboard)
];
// null (not a guessed fallback URL) when we don't have a confirmed dashboard
// link for this reflector -- see index.php for why guessing was unsafe.
$reflector_url = $REFLECTOR_DASHBOARDS[$room_name] ?? null;

global $LIVE_STATE_FILE;
$live_state = read_live_state($LIVE_STATE_FILE);

header('Content-Type: application/json');
echo json_encode([
    'gw_html' => rows_to_html($gateway_rows),
    'local_html' => rows_to_html($local_rows, true),

```

```

'svc_ok'      => $svc_ok,
'room_name'   => $room_name,
'room_module' => $room_module,
'm17_address' => $m17_address,
'reflector_url' => $reflector_url,
'live_state'  => $live_state,
});

```

7d. index.php -- the main page

Save as `/var/www/html/m17/index.php`:

```

<?php
require_once __DIR__ . '/lib.php';
run_pipeline();

$ini = @parse_ini_file('/etc/USRPM17.ini', true);
$m17_callsign = $ini['M17 Network']['Callsign'] ?? '';
$m17_name     = $ini['M17 Network']['Name'] ?? '';
$m17_address  = $ini['M17 Network']['Address'] ?? '';
list($room_name, $room_module) = array_pad(explode(' ', $m17_name, 2), 2, '');

// NOTE: only hyperlink a reflector name when a URL for it has been
// independently confirmed by actually visiting it -- see notes on bugs
// for why guessing (or falling back to the bare IP) isn't safe.
$REFLECTOR_DASHBOARDS = [
    'M17-M17' => 'http://m17.openquad.net',
    'M17-117' => 'http://96.224.250.24:8087/#/', // confirmed working -- needs this specific port + route, plain IP alone does not work
    'M17-KCW' => 'http://m17.kansascitywide.com', // confirmed working
    'M17-DVS' => 'https://150.136.69.97/?conf=M17Reflector', // confirmed working (DVSwitch multi-mode reflector's own dashboard)
];
$reflector_url = $REFLECTOR_DASHBOARDS[$room_name] ?? null;

$svc_active = trim(shell_exec('sudo systemctl is-active usrp2m17 2>/dev/null'));
$svc_ok = ($svc_active === 'active');

$gateway_rows = read_display_rows($GW_LOG, $DISPLAY_ROWS);
$local_rows   = read_display_rows($LOCAL_LOG, $DISPLAY_ROWS);

$rawuptime = shell_exec('cat /proc/uptime');
$uptime_secs = (int) substr($rawuptime, 0, strpos($rawuptime, " "));
$up_h = intval($uptime_secs, 3600); $up_m = intval($uptime_secs % 3600, 60);
$uptime = "{$up_h}h {$up_m}m";
$free_mem = trim(shell_exec("free -m | awk 'NR==2{printf \"%.0f%%\", \\$3*100/\\$2 }'"));
$disk_used = trim(shell_exec("df -h | awk '\\$NF==\\\"/\\\"{printf \"\\$s\\\",\\$5}'"));
$cpuload = sys_getloadavg();
$cpuTempHTML = "<td style=\\\"background:#fff\\\">---</td>";
if (file_exists('/sys/class/thermal/thermal_zone0/temp')) {
    $raw_t = (int) trim(shell_exec('cat /sys/class/thermal/thermal_zone0/temp'));
    $cpuTempC = $raw_t > 1000 ? round($raw_t / 1000) : round($raw_t);
    $cpuTempF = round($cpuTempC * 9 / 5 + 32);
    $bg = $cpuTempC < 55 ? '#1d1' : ($cpuTempC < 70 ? '#fa0' : '#f00');
    $cpuTempHTML = "<td style=\\\"background:$bg\\\">{$cpuTempC}&deg;C / {$cpuTempF}&deg;F</td>";
}
$platform = trim(shell_exec("cat /proc/device-tree/model 2>/dev/null")) ?: 'Generic aarch64 class computer';

$FIRST_YEAR = 2026;
$cur_year = (int) date('Y');
$year_str = ($cur_year <= $FIRST_YEAR) ? "$FIRST_YEAR" : "$FIRST_YEAR - $cur_year";
$footer_callsign_base = extract_base_callsign($m17_callsign);
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>M17 Bridge Dashboard</title>

```

```

<style>
* { box-sizing: border-box; }
body { background-color:#f8f8f8f8; font:11pt arial, sans-serif; margin:0; padding:10px; }
.wrap { max-width:920px; margin:0 auto; }
table { border-collapse:collapse; width:100%; }
th { background:#2f6690; color:#fff; padding:3px 8px; font-size:11px; border:1px solid #1c4a68; text-align:left; }
td { padding:3px 8px; font-size:11px; border:1px solid #ddd; background:#ffffff; word-break:break-word; }
tr:nth-child(even) td { background:#fdf6e3; }
.card { box-shadow:0 0 8px #999; border-radius:8px; overflow:hidden; margin-top:10px; background:#fff; }
.card-title { background:#2f6690; color:#fff; font-weight:bold; padding:5px 10px; font-size:12px; }
.card-body { padding:6px; background:#e8e8e8; }
.grid { display:grid; grid-template-columns:220px 1fr; gap:12px; align-items:start; }
@media (max-width:680px) { .grid { grid-template-columns:1fr; } }
.center-table th, .center-table td { text-align:center; }
/* Column widths tuned for a phone screen: Time and Dur(s) need the most
room (especially Dur(s) once it carries the "(X ago)" text), while
Callsign and Target rarely exceed something like "NOCALL M" or
"M17-DVS A", so they can stay narrow with nowrap. Country is sized to
"1%" + nowrap, which tells the browser to shrink it to just fit its
header text/flag rather than claim a fixed percentage. */
.activity-table th:nth-child(1), .activity-table td:nth-child(1) { width:26%; white-space:nowrap; }
.activity-table th:nth-child(2), .activity-table td:nth-child(2) { width:15%; white-space:nowrap; }
.activity-table th:nth-child(3), .activity-table td:nth-child(3) { width:1%; white-space:nowrap; }
.activity-table th:nth-child(4), .activity-table td:nth-child(4) { width:22%; white-space:nowrap; }
.activity-table th:nth-child(5), .activity-table td:nth-child(5) { width:24%; }
.activity-table .text-center { text-align:center; }
@media (max-width:600px) { table { font-size:11px; } th, td { padding:3px 5px; } }
</style>
</head>
<body>
<div class="wrap">

<div class="card">
  <div class="card-title" style="text-align:center; font-size:16px; padding:10px;">M17 Bridge Dashboard</div>
</div>

<div class="grid">
<div>

  <div class="card">
    <div class="card-title">Status</div>
    <div class="card-body">
      <table style="margin-bottom:8px;">
        <tr><th colspan="2">Bridge Info</th></tr>
        <tr><th width="50%">Callsign</th><td style="color:#b44010; font-weight:bold;"><?= htmlspecialchars($m17_callsign) ?></td></tr>
        <?php if ($svc_ok): ?>
        <tr><th width="50%">Reflector</th><td id="reflectorCell" style="font-weight:bold;"><?php if ($reflector_url): ?><a href="<?=
htmlspecialchars($reflector_url) ?>" target="_blank" style="color:#ef7215;"><?= htmlspecialchars($room_name) ?></a><?php else: ?><span
style="color:#ef7215;"><?= htmlspecialchars($room_name) ?></span><?php endif; ?></td></tr>
        <tr><th width="50%">Module</th><td id="moduleCell"><?= htmlspecialchars($room_module) ?></td></tr>
        <tr><th width="50%">Address</th><td id="addressCell"><?= htmlspecialchars($m17_address) ?></td></tr>
        <?php else: ?>
        <tr><th width="50%">Reflector</th><td id="reflectorCell" style="color:#999;">&mdash;</td></tr>
        <tr><th width="50%">Module</th><td id="moduleCell" style="color:#999;">&mdash;</td></tr>
        <tr><th width="50%">Address</th><td id="addressCell" style="color:#999;">&mdash;</td></tr>
        <?php endif; ?>
      </table>

      <table style="margin-bottom:8px;">
        <tr><th colspan="2">TRX Info</th></tr>
        <tr><td id="trxBox" colspan="2" style="text-align:center; <?= $svc_ok ? 'background:#e8a800;color:#000;' :
'background:#ffffff;color:#b0b0b0;font-weight:bold;' ?>>
          <?= $svc_ok ? 'Listening' : 'OFFLINE' ?>
        </td></tr>
      </table>

      <table>
        <tr><th colspan="2">M17 Net</th></tr>

```

```

        <tr><td id="netBox" colspan="2" style="text-align:center;">
            <?= $svc_ok ? '<span style="color:#b5651d;font-weight:bold;">Linked</span>' : '<span style="color:#b0b0b0;font-weight:bold;">Not
Linked</span>' ?>
        </td></tr>
    </table>
    <button id="disconnectBtn" onclick="m17Disconnect()" style="width:100%; padding:8px; background:<?= $svc_ok ? '#c0392b' : '#999' ?>;
color:#fff; border:none; border-radius:4px; font-weight:bold; font-size:12px; cursor:<?= $svc_ok ? 'pointer' : 'not-allowed' ?>; margin-top:2px;"
<?= $svc_ok ? '' : 'disabled' ?>>
        Disconnect Reflector
    </button>
</div>
</div>

</div>
<div>

    <div class="card">
        <div class="card-title">Gateway Activity</div>
        <div class="card-body">
            <table class="activity-table">
                <tr><th>Time</th><th>Callsign</th><th>Country</th><th>Target</th><th>Dur(s)</th></tr>
                <tbody id="gwBody"><?= rows_to_html($gateway_rows) ?></tbody>
            </table>
        </div>
    </div>

    <div class="card">
        <div class="card-title">Local RF Activity</div>
        <div class="card-body">
            <table class="activity-table">
                <tr><th>Time</th><th>Callsign</th><th>Country</th><th>Target</th><th>Dur(s)</th></tr>
                <tbody id="localBody"><?= rows_to_html($local_rows, true) ?></tbody>
            </table>
        </div>
    </div>

</div>
</div>

    <div class="card">
        <div class="card-title">Hardware Info</div>
        <div class="card-body" style="overflow-x:auto;">
            <table class="center-table" style="min-width:640px;">
                <tr>
                    <th>Hostname<br><span style="color:#effd5f; font-size:10px;">IP: <?= str_replace(' ', '<br>', trim(shell_exec('hostname -I'))) ?></span>
                </th>
                    <th>Kernel<br>release</th>
                    <th colspan="2">Platform<br><span style="color:#effd5f; font-size:12px;">Uptime: <?= $uptime ?></span></th>
                    <th>Disk<br>used</th>
                    <th>Memory<br>used</th>
                    <th>CPU Load</th>
                    <th>CPU Temp</th>
                </tr>
                <tr>
                    <td><?= php_uname('n') ?></td>
                    <td><?= php_uname('r') ?></td>
                    <td colspan="2"><?= htmlspecialchars($platform) ?></td>
                    <td><?= $disk_used ?></td>
                    <td><?= $free_mem ?></td>
                    <td><?= round($cpuLoad[0],1) ?> / <?= round($cpuLoad[1],1) ?> / <?= round($cpuLoad[2],1) ?></td>
                    <td><?= $cpuTempHTML ?>
                </tr>
            </table>
        </div>
    </div>

<div style="text-align:center; font-size:8pt; margin-top:10px; color:#555;">
    <?= $year_str ?><br>

```

Based on Pi-Star Dashboard, © Andy Taylor (MW0MMWZ)


```
<?php if ($footer_callsign_base): ?>
```

```
Adapted for USRP2M17 bridge by <a href="https://www.qrz.com/db/<? urlencode($footer_callsign_base) ?>" target="_blank" style="color:#0645AD;"><? htmlspecialchars($footer_callsign_base) ?></a>
```

```
<?php else: ?>
```

```
Adapted for USRP2M17 bridge by <? htmlspecialchars($m17_callsign) ?>
```

```
<?php endif; ?>
```

```
</div>
```

```
</div>
```

```
<script>
```

```
function m17Poll(){
```

```
    fetch('activity_fragment.php', {cache:'no-store'}).then(function(r){ return r.json(); }).then(function(data){
```

```
        document.getElementById('gwBody').innerHTML = data.gw_html;
```

```
        document.getElementById('localBody').innerHTML = data.local_html;
```

```
        var trx = document.getElementById('trxBox');
```

```
        var net = document.getElementById('netBox');
```

```
        var reflectorCell = document.getElementById('reflectorCell');
```

```
        var moduleCell = document.getElementById('moduleCell');
```

```
        var addressCell = document.getElementById('addressCell');
```

```
        var disconnectBtn = document.getElementById('disconnectBtn');
```

```
        if (data.svc_ok) {
```

```
            if (data.live_state === 'tx') {
```

```
                trx.style.background = '#d00'; trx.style.color = '#fff'; trx.style.fontWeight = 'bold';
```

```
                trx.textContent = 'TX';
```

```
            } else if (data.live_state === 'rx') {
```

```
                trx.style.background = '#0b0'; trx.style.color = '#030'; trx.style.fontWeight = 'bold';
```

```
                trx.textContent = 'RX';
```

```
            } else {
```

```
                trx.style.background = '#e8a800'; trx.style.color = '#000'; trx.style.fontWeight = 'normal';
```

```
                trx.textContent = 'Listening';
```

```
            }
```

```
            net.innerHTML = '<span style="color:#b5651d;font-weight:bold;">Linked</span>';
```

```
            reflectorCell.style.color = '';
```

```
            reflectorCell.innerHTML = data.reflector_url
```

```
                ? ('<a href="' + data.reflector_url + '" target="_blank" style="color:#ef7215;">' + data.room_name + '</a>')
```

```
                : ('<span style="color:#ef7215;">' + data.room_name + '</span>');
```

```
            moduleCell.style.color = ''; moduleCell.textContent = data.room_module;
```

```
            addressCell.style.color = ''; addressCell.textContent = data.m17_address;
```

```
            disconnectBtn.disabled = false;
```

```
            disconnectBtn.style.background = '#c0392b';
```

```
            disconnectBtn.style.cursor = 'pointer';
```

```
        } else {
```

```
            disconnectBtn.disabled = true;
```

```
            disconnectBtn.style.background = '#999';
```

```
            disconnectBtn.style.cursor = 'not-allowed';
```

```
            trx.style.background = '#fffed'; trx.style.color = '#b0b0b0'; trx.style.fontWeight = 'bold';
```

```
            trx.textContent = 'OFFLINE';
```

```
            net.innerHTML = '<span style="color:#b0b0b0;font-weight:bold;">Not Linked</span>';
```

```
            reflectorCell.style.color = '#999'; reflectorCell.textContent = '\u2014';
```

```
            moduleCell.style.color = '#999'; moduleCell.textContent = '\u2014';
```

```
            addressCell.style.color = '#999'; addressCell.textContent = '\u2014';
```

```
        }
```

```
    }).catch(function({});
```

```
}
```

```
setInterval(m17Poll, 2000);
```

```
function m17Disconnect(){
```

```
    var currentName = document.getElementById('reflectorCell').textContent.trim();
```

```
    if (!confirm('Disconnect from ' + currentName + '?')) return;
```

```
    var btn = document.getElementById('disconnectBtn');
```

```
    btn.disabled = true;
```

```
    btn.textContent = 'Disconnecting...';
```

```
    fetch('disconnect.php', {method:'POST', cache:'no-store'}).then(function(r){ return r.json(); }).then(function(data){
```

```
        btn.textContent = 'Disconnect Reflector';
```

```
        m17Poll();
```

```
    }).catch(function({}
```

```

    btn.textContent = 'Disconnect Reflector';
    btn.disabled = false;
  });
}
</script>

</body>
</html>

```

7e. `disconnect.php` -- web "Disconnect Reflector" button target

This is a core part of the dashboard, not an optional add-on: if your radio ever gets stuck transmitting (stuck PTT) or a simplex station holds the frequency without stopping, you have no way to dial a DTMF disconnect code from the radio itself. This endpoint is what makes the "Disconnect Reflector" button on the dashboard (already wired up in `index.php` above) actually work -- clicking it stops the bridge immediately from a phone or browser, same effect as the `*BB0` DTMF code (stops the M17 link only; the private node stays linked to your radio). Without this file in place, the button is visible but does nothing.

Save as `/var/www/html/m17/disconnect.php`:

```

<?php
/* Web "Disconnect" button target -- stops the USRP2M17 bridge only
   (equivalent to the *BB0 DTMF code: reflector disconnect, private
   node stays linked to the radio). This exists specifically for the
   case where the radio itself is stuck transmitting (stuck PTT, or a
   simplex station holding the frequency without stopping) and dialing
   a DTMF code from the radio isn't possible -- stopping the bridge
   service immediately halts anything reaching the M17 network,
   regardless of the local radio's PTT state. */
header('Content-Type: application/json');

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    http_response_code(405);
    echo json_encode(['ok' => false, 'error' => 'POST required']);
    exit;
}

shell_exec('sudo systemctl stop usrp2m17 2>&1');
$active = trim(shell_exec('sudo systemctl is-active usrp2m17 2>/dev/null'));

echo json_encode(['ok' => ($active !== 'active'), 'status' => $active]);

```

7f. Permissions

```

sudo chown www-data:www-data /var/www/html/m17/*.php
sudo chmod 644 /var/www/html/m17/*.php

```

Visit `http://<your-pi-ip>/m17/` to see it.

8. Testing

1. `sudo systemctl start usrp2m17` and confirm `sudo systemctl status usrp2m17` shows active.
2. Dial `*BB1` (or your chosen function code + digit) on the radio and confirm the dashboard's Reflector field updates within a couple of seconds.
3. Key up on the radio, talk for a few seconds, and confirm the TRX Info box turns red and shows "TX" while you're transmitting, then a row appears in **Local RF Activity** with your correct callsign, target, and a duration matching your actual key-up time. Key up again a few minutes later with the same callsign and confirm the OLD row is replaced, not duplicated -- Local RF Activity should only ever show one row per callsign.
4. Have someone else transmit on the reflector and confirm the TRX Info box turns green with "RX", and their transmission shows up in **Gateway Activity** alongside your own transmissions, in chronological order, with no per-callsign deduplication (Gateway Activity keeps the last 200 sessions and shows the newest 25; Local RF Activity keeps one row per callsign).
5. Dial `*BB00` and confirm the radio node unlinks from the private node, TRX Info goes to OFFLINE, and Reflector/Module/Address go blank (dashes).

6. Test the "Disconnect Reflector" button: while linked to a reflector, click it, confirm the prompt, and verify the bridge stops (TRX Info goes to OFFLINE, Reflector/Module/Address go blank) within a couple of seconds -- then reconnect via `*BB<code>` on the radio to confirm it comes back up normally afterward.
7. Check `sudo tail -20 /var/lib/m17dashboard/debug.log` -- each line should show a real `cursor=s=...` value (never `cursor=none` after the very first line), and `decoded_events` should be non-zero whenever real traffic occurred in that window.
8. Check that the "(X ago)" text next to your Local RF Activity duration reads sensibly (seconds for the first minute, then minutes) -- if it's off by several hours, your Pi's PHP default timezone doesn't match its actual system timezone; see the notes below.

9. Notes on Bugs We Hit Building This (and Why the Code Looks the Way It Does)

These are already baked into the code above as comments, but worth calling out explicitly since they're easy to reintroduce if you modify the scripts:

- **Never classify "did I transmit this" by trusting the raw "Sent"/"Received" text in USRP2M17's log line.** On a plain M17-to-M17 reflector those labels happen to line up with "did I originate this," but on a multi-mode/cross-bridge gateway (e.g. a DVSwitch-style reflector bridging DMR/YSF/M17), a frame carrying a REMOTE station's own callsign can still get logged as "Sent" by USRP2M17's internal socket-direction bookkeeping -- verified against a real capture where a frame literally logged as "M17 Network Data Sent" decoded to a real remote station's callsign, not the local one. The dashboard instead compares the **decoded SRC callsign** against the callsign configured in `/etc/USRP2M17.ini`, which is what the M17 protocol itself actually guarantees means "who originated this."
- **Never track journal position with a time-based `--since` window; use `journalctl`'s own cursor (`--after-cursor` / `--show-cursor`) instead.** An earlier version of this dashboard tried to track "where we last left off" using timestamps, with a small deliberate overlap "to be safe" against missing a frame at a poll boundary -- but that overlap caused the same real transmission to get logged multiple times whenever another station's traffic was interleaved in between, since the merge-dedup logic only compares against the single immediately-preceding line. A cursor identifies an exact journal entry, so resuming from one is byte-precise with no overlap and no double-counting. **Watch the cursor line's exact format** -- it's `-- cursor: s=...` with no trailing `--`; a regex that expects a trailing `--` will never match and will silently fall back to reprocessing a rolling window forever, which looks exactly like duplicate/inflated durations.
- **`journalctl --since` with an absolute timestamp is parsed in the system's LOCAL timezone even when `--utc` is also passed** -- `--utc` only changes how timestamps are *displayed*, not how an absolute `--since` argument is *parsed*. Feeding it a UTC-generated timestamp silently breaks incremental polling. If you ever go back to a time-based approach for any reason, use a relative offset like `-N seconds` instead, which has no timezone to get wrong.
- **The M17 base40 callsign decoder builds the string by appending, not prepending.** Getting this backwards produces a perfectly plausible-looking but completely reversed callsign (e.g. real `N0CALL` decodes as `LLAC0N`), which is easy to mistake for random data corruption rather than a logic bug.
- **Room names like `M17-KCW` contain a dash as part of the actual name**, not as a mode-tag separator -- don't run destination/target strings through a "strip trailing tag" helper designed for personal callsigns, or valid room names get mangled.
- **Cap how far back merged/duplicate log rows can span.** Without a time-gap check, any two transmissions sharing the same callsign and target -- even minutes apart -- silently merge into one ever-growing row instead of logging as separate events.
- **Filter out very short "transmissions" at display time, not storage time.** A fraction-of-a-second keyup is far more likely to be a noise/COS-tail trigger (or a garbled frame that happened to pass the plausibility filter) than an intentional PTT. Hiding anything under the `MIN_DISPLAY_DURATION_SECONDS` threshold (0.3s, see the tuning note further down) in the display layer keeps the Gateway log clean without permanently discarding data. Local RF Activity is the one exception, see the next note.
- **Local RF Activity must filter noise BEFORE storing, not just at display time.** Because Local RF Activity keeps only one row per callsign, a stray sub-threshold blip from your own callsign (for example, the audio path reconnecting during a reflector switch) would replace your last real session in the log, and then get hidden at display time, so your real row would just vanish. `run_pipeline()` now drops sub-threshold local sessions before `append_persistent_last_only()` ever sees them. Gateway Activity doesn't need this, since a noise row there only sits among real history without replacing anything.
- **The live TX/RX indicator needs its own short-lived state file, separate from the persisted activity logs**, tracking only the most recent frame's direction and timestamp; it should read as active only within a few seconds of that timestamp, falling back to idle/Listening automatically once transmissions stop.
- **The dashboard's TRX Info colors are: green for RX (someone else transmitting), amber/yellow for idle Listening, and red for TX (you transmitting)** -- deliberately distinct from a common alternate scheme (green idle / amber RX) so a quick glance tells you which state you're in without reading the text.
- **Guard against two overlapping pipeline runs processing the same journal window twice.** If one poll's `journalctl` + parsing work takes longer than the 2-second polling interval (e.g. catching up on a large backlog after a gap in traffic), the next poll can start before the first has saved its cursor -- both read the same starting point, both process the same batch of frames, and both append the same sessions to the log. The symptom looks exactly like a decode bug (rows with identical timestamps and durations appearing more than once) but is really a concurrency bug. Fixed with a non-blocking file lock (`flock(..., LOCK_EX | LOCK_NB)`) around the whole pipeline -- if a run is already in progress, the next call just skips processing that round entirely rather than racing it.

- **Merge continuous transmissions by actual TIME GAP between frames, not by strict array adjacency.** On a busy multi-mode gateway, a single frame from another station's near-simultaneous chatter can land in the middle of someone's continuous transmission -- requiring frames to be immediately consecutive to merge would then split one real, uninterrupted PTT into two shorter logged rows. This was confirmed against a reflector's own "Last Heard" page: two transmissions that were genuinely 32 seconds and 59 seconds long were each showing up split into two shorter pieces here. The fix merges any two frames from the same callsign+target whenever the actual time gap between them is small (a few seconds), regardless of what else got logged in between.
- **Don't guess a reflector's own dashboard URL, and don't assume the bare IP is a safe fallback either.** A URL guessed for one reflector (based on a plausible-looking hostname pattern) turned out to point at a completely different reflector's dashboard instead. Falling back to the bare IP address wasn't safe either -- that same address, visited directly, turned out to host an unrelated web hosting company's site with no connection to M17 at all. The dashboard only hyperlinks a reflector name when a URL for it has been independently confirmed by actually visiting it; otherwise it shows plain, non-linked text rather than sending someone to the wrong place.
- **strtotime() on a naive (no-timezone) time string uses PHP's default timezone setting, which can silently differ from the Pi's actual system timezone.** The "X ago" formatting for Local RF Activity initially came out wrong by a fixed multi-hour offset (a transmission from seconds ago read as "4 hours ago") because PHP's default timezone wasn't pinned to match the system clock. Fixed by explicitly reading `/etc/timezone` and calling `date_default_timezone_set()` with it before any `strtotime() / time()` comparison happens.
- **Gateway Activity should include your OWN transmissions too, not just remote stations.** The very first version filtered Gateway Activity down to remote-only traffic, on the assumption that "your own stuff belongs in Local Activity instead." But that's not how a real last-heard log works (confirmed by checking the actual M17-DVS reflector's own last-heard page, and by checking WPSD's own Gateway Activity table) -- a gateway/last-heard table is a full chronological record of everything heard, local and remote mixed together. Local RF Activity is a *separate, additional* view scoped to just your own callsign, not a replacement destination that Gateway Activity should exclude.
- **Local RF Activity needs its own dedicated append function that keeps only the latest session PER CALLSIGN, not the shared append function (which keeps a rolling history of up to 200 rows) used for Gateway Activity.** Reusing `append_persistent()` (which just keeps appending) for the local log meant "show only my last session" never actually took effect until a second function, `append_persistent_last_only()`, was written specifically to replace-in-place rather than append.
- **The merge-gap logic (merging frames from the same callsign+target within a few seconds into one session) can occasionally merge two things a human would consider separate, and occasionally NOT merge things a human would consider one continuous transmission -- and there is no way to tell the difference from timing data alone.** Confirmed both directions during testing: (1) one continuous real key-up got logged by the M17-DVS reflector's own software as two separate frames a few seconds apart (likely due to a periodic LSF/header re-send mid-transmission), which our merge window correctly glued back into one accurate row -- arguably *more* correct than the reflector's own display; but (2) two genuinely separate quick taps a few seconds apart would merge the same way and look like one continuous session. This is a fundamental ambiguity in the data, not a bug with a clean fix -- `SESSION_MERGE_GAP_SECONDS` is a tuning knob, not a guarantee.
- **Weak/marginal-signal decodes can produce a burst of near-zero-duration "transmissions" from the same callsign within the same second, sometimes with the destination field flipping between two different values frame-to-frame.** This looks exactly like garbage/decode noise (and reflectors' own last-heard pages typically don't show it, since they have their own internal filtering), not like a real deliberate key-up. `MIN_DISPLAY_DURATION_SECONDS` filters this at display time -- initially set to 1.5s (too aggressive, hid real short taps), reduced to 0 (showed the decode-noise bursts), and settled at 0.3s as a working middle ground. There's no perfect threshold here since duration alone can't distinguish "genuinely brief real keyup" from "corrupted frame burst" -- 0.3s is a judgment call based on what was actually observed, not a formula.
- **A CSS class referenced in markup does nothing unless it's actually defined in the stylesheet.** The flag `<td>` was given `class="text-center"` (copied from the WPSD reference dashboard's own markup pattern) without realizing this dashboard's own `<style>` block never defined that class -- so the flags silently stayed left-aligned until a `.text-center { text-align:center; }` rule was added.
- **A low-resolution source image looks blurry once a page is pinch-zoomed, even if its on-page display size never changes.** The country flags were requested from `flagcdn.com` at their exact 20x15 display size, which looked fine at normal zoom but visibly pixelated once the page was zoomed in on a phone. Requesting a larger source image (80x60) while keeping the same `width="20" height="15"` HTML display attributes fixed this -- more source pixel data to zoom into, same on-page footprint.
- **A field that can only ever hold one value doesn't need its own row.** The Bridge Info table originally had a "Mode" row hardcoded to always display "M17" -- since this dashboard only ever bridges M17, that row could never show anything else, so it was pure redundancy rather than information. Removed entirely rather than left in for consistency with a generic template.
- **Setting disabled on a button doesn't change how it looks, only how it behaves.** The Disconnect Reflector button correctly stopped accepting clicks while offline (the `disabled` HTML attribute was already set), but it still rendered in the same solid red as when active, since `disabled` alone doesn't touch any CSS. It needed an explicit style change (gray background, `cursor: not-allowed`) applied in two places: once in the initial PHP render (`index.php`), and again in the JS poll loop (`m17Poll()`), so the color stays correct both on page load and as the bridge's live state changes without a reload.
- **AllStarLink's link-unkey courtesy tone gets forwarded to M17 and shows up as a short transmission from your own callsign.** The radio node's `(node-main)` template sets `linkunkeyct = ct8`, a tone whose whole documented purpose is to be sent to linked nodes when a transmission received over a link unkeys. Since the private M17 node is one of those linked nodes, the tone goes out through USRP2M17 to the reflector as real audio, indistinguishable from a short PTT. There is no `rpt.conf` setting that plays it locally only: hearing it and forwarding it are the same mechanism. The options are to blank `linkunkeyct` on the radio node (you lose the local cue too) or accept the occasional short extra row in the dashboard. This build keeps the tone and treats the extra

rows as cosmetic.

- **Diagnose the real cause before adding filtering code.** Those courtesy-tone rows were first blamed on reflector switching, and suppression logic was added that checked a `last_switch.txt` timestamp file. `m17_link.sh` never creates that file, so the code could never do anything. It was removed once the real cause was found. If you add logic that depends on a file or signal, confirm that something actually produces it.
- **Wi-Fi packet loss can split one transmission into two rows (and drop audio).** `ping -D` showed 1.4-2.4 second spikes on the Pi's 5 GHz Wi-Fi. A gap longer than `SESSION_MERGE_GAP_SECONDS` (5s) splits one key-up into two rows, and shorter gaps can still cause audible dropouts. DVSwitch has a bridge-level hang time setting (`minTxTimeMS`) that smooths over this, but USRP2M17 has no equivalent in `/etc/USRP2M17.ini`. The merge window is left at 5s on purpose. A wired Ethernet connection is the real fix if dropouts are frequent. To test the path to a specific reflector, ping its own address (e.g. `ping -D 96.224.250.24`), not a general host like 8.8.8.8, since routing quality varies by destination.
- **Don't map a whole letter to one country when the ITU splits it.** The prefix table originally had `'A'=>'US'`, so every callsign starting with A got a US flag. But only AA-AL are American: A2-A9 are other countries (e.g. A6 is the United Arab Emirates, A7 Qatar), AM-AO are Spain, AP-AS Pakistan, and so on. A UAE station showed a US flag until the single-letter `A` entry was replaced with the real two-character prefixes. A callsign whose prefix isn't in the table now shows no flag rather than a wrong one.