Telive - Tetra live receiver v1.9 (beta version, subject to change)
(c) 2014-2016 Jacek Lipkowski <sq5bpf@lipkowski.org>

TL;DR: Anyone saying “too long; didn't read” will be considered a lazy idiot. I've spent a lot of time writing this documentation, and telive users are expected to read all of it (multiple times if needed). Questions resulting from failure to do so will be ignored at best. I'm not a big fan of the “TL;DR” abbreviation.

Description

Telive is a program which can be used to display information like signaling, calls, short SDS, location information etc from a TMO Tetra network. It is also possible to log the signalling information, listen to the audio in realtime and to record the audio. Playing the audio and re-compressing it into ogg is done via external scripts. The software is based upon modified osmocom-tetra software. Location information can be written to a KML file, to be opened in Google Earth and similar software. The software can optionally control the receiver to automatically tune based on signalling information, or to provide scanning functionality.

Prerequisites

Before installing please read all licenses, disclaimers, documentation for all installed packages, and about the Tetra protocol (i will not explain term like MNC, Colour Code, Usage identifier, SSI etc).

This software is tested on Debian GNU/Linux 8 using the install_telive.sh script and gnuradio from debian packages, this is the preffered platform. Other linux distributions should work, but i have not tested them well. It should probably be possible to run this software on other systems, but i haven't tried, and don't intend to (however if it works for you, please send me a detailed description so that i can add it to the docs).

Receiving is done using a RTL2832/R820T DVB-T USB dongle. Probably other receivers supported by gnuradio will work with minor modifications.

For a quick start use the easy install script, and work through the configuration examples.

Easy install script

The following script can be used to install everything automatically under Debian 8, Ubuntu 14 and 15, and Linux Mint 17.2 and 17.3. With minor modifications it should also work under Debian 6/7 and other debian-like distributions, if a supported gnuradio version is installed via some other way – either using pybombs or the SBRAC build-gnuradio script. To install verify that you have sudo configured for your user, and that you have internet access, and do this:

```
wget https://raw.githubusercontent.com/sq5bpf/telive/master/scripts/install_telive.sh
# now here you should read the script, don't just blindly run scripts off the internet
chmod 755 install_telive.sh
./install_telive.sh
```

Manual install

This has been moved to the FAQ section “How can I manually compile/install this software?”. Please use the Easy install script whenever possible.

Optional step

Get the latest wireshark from the repository. Wireshark can parse GSMTAP messages which are sent via tetra-rx and display the decoded Tetra frames. Also install some music player, which can play ogg files (most of them can), such as audacious.

Theory of operation

The software is based upon a modular design, please refer to Fig. 1 for a simple example. The RF signal is received using a receiver program, providing one or more channels. The receiver program uses a RLT2832 ased DVB-T dongle as a sdr, and outputs gnuradio c-file data. This channel data is sent using some transport mechanism (named pipes or UDP datagrams) to a script (receiver1 or receiver1udp) containing a CQPSK demodulator program (simdemod2.py) and a program that decodes the tetra frames (a fork of tetra-rx from the osmo-tetra software). The decoded data is sent via UDP datagrams to a program (rxx script, which runs the telive program) that will try to make sense of this data: extract signalling information, call data etc. The telive program can play the decoded audio data, record the data (to be later recompressed into ogg files by the tplay script). The telive program can optionally control the receiver via a XMLRPC interface, this is useful for scanning, automatically tuning multiple channels etc. A single telive instance should service only data from channels from one tetra LA.

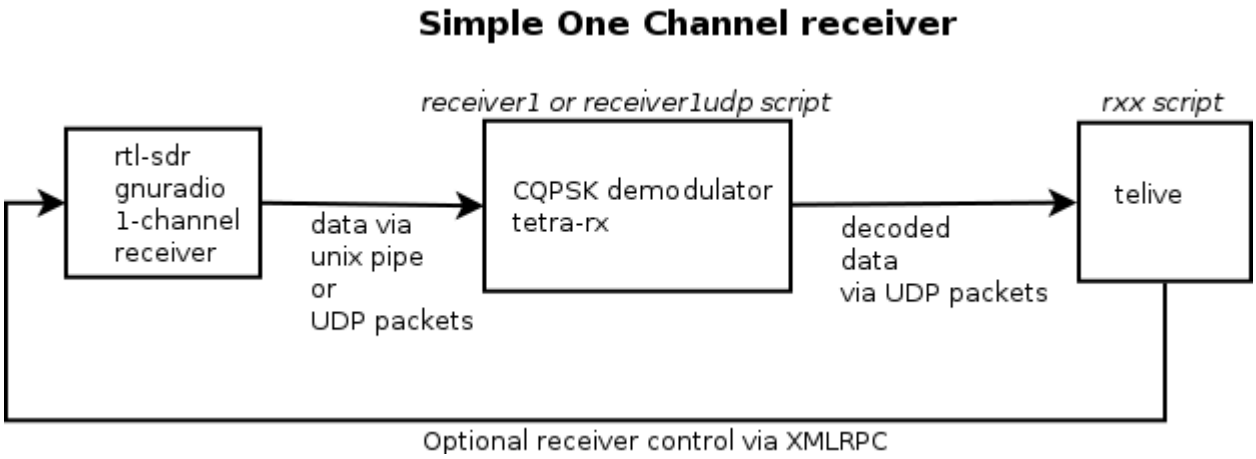

Fig 1. Simplified receiver flow

More advanced setups will have multi-channel receivers, feeding each channel to a separate demodulator/decoder, and feeding the results from these to one or more telive instances.

The receiver program

The standard receiver is based upon a RTL2832 DVB-T dongle and gnuradio. It's conveniently provided as an gnuradio-companion flow graph (without any additional python code), so that it can be trivially modified. Each channel is 36ks/s gnuradio cfile output, this data is output either via a file sink to a named pipe (the pipes are called /tmp/fifoN, where N=1,2,3,4,...), or via UDP packets. Transport via UDP packets is preferred, because this allows for the receiver to be restarted without restarting the demodulator script. Optionally some flowgraphs have an XMLRPC control interface that can be used by telive to discover the receiver type and control it (the default is to have it on port 42000/tcp).

The receiver programs are provided in telive/gnuradio-companion, with descriptive README.txt files in each directory. The telive/gnuradio-companion/receiver_xmlrpc directory will contains also two headless (no GUI) receivers compiled to python code (1 channel receiver, and 6 channel receiver so far). These should be launched via the kill_wrapper script in the same directory like so:

./kill_wrapper ./telive_6ch_gr37_udp_xmlrpc_headless.py

Any software with the same data format can be substituted for the receiver program.

The demodulator/decoder script

Data from the receiver program is passed via a named pipe (this software uses /tmp/fifoN m where N=1,2,3,4,...) or via UDP packets to a demodulator/decoder script (osmo-tetra-sq5bpf/src/receiver1 for named pipes, osmo-tetra-sq5bpf/src/receiver1udp for UDP). This script will pipe the data into simdemod2.py (CQPSK demodulator), and pipe the resulting data to tetra-rx (tetra decoder). The decoded signalling information and voice data is passed via UDP packets to the telive program.

The original osmo-tetra software from Osmocom used another program float_to_bits, which would take the output from the CQPSK demodulator and convert it into discrete values that were later fed into tetra-rx (this is a bit similar to a “bit slicer”). This functionality is integrated into the tetra-rx program from osmo-tetra-sq5bpf.

The tetra-rx program can calculate a running average from the data from the CQPSK modulator, which can show how much the signal is mistuned. This value can be used to automatically remove the bias from the input data (this helps receive mistuned signals), and can be later passed to the telive program, which can use this to automatically adjust the frequency correction PPM value. This feature is called AFC for Automatic Frequency Correction (not a good use of the term, but close enough).

Command line parameters of all programs are described further in this document.

The telive program

The telive program (executed by the rxx script) is the heart of the telive suite. The rxx script sets various environment variables, which are used for configuration data, and runs the telive program. Telive depends on 203x60 terminal size. The software will still function, but smaller terminals will result in a garbled screen.

The telive program receives this information:

- signalling/network information (which calls are made/broken, network parameters, SDS etc)
- voice traffic
- location information (this is sent via SDS) - some location protocols are decoded
- AFC information, which shows how much the signal is mistuned
- a receiver identifier (identifies each decoder sending data to telive, must be unique)

The telive program accepts data from one or more tetra-rx instances, and tries to match the signalling data (which SSIs made the call etc) to the voice data. Based on this it will record the voice calls, and allow live playing. It will also log signalling information (including SDS etc).

Location data is written to a KML file (filename is configurable), which can be read with Google Earth, showing the locations on a map.

Telive can also control the receiver via an XMLRPC interface. This can be used to:
- adjust the receiver frequency correction PPM value from the incoming AFC data
- tune unused receivers to other frequencies within the same LA. In a multi-channel receiver setup only one frequency must be tuned correctly, the rest od the channels will be tuned automatically.
- implement scanning
- manually control the receiver baseband frequency, offsets, PPM and gain. This is especially useful for receivers without a GUI.

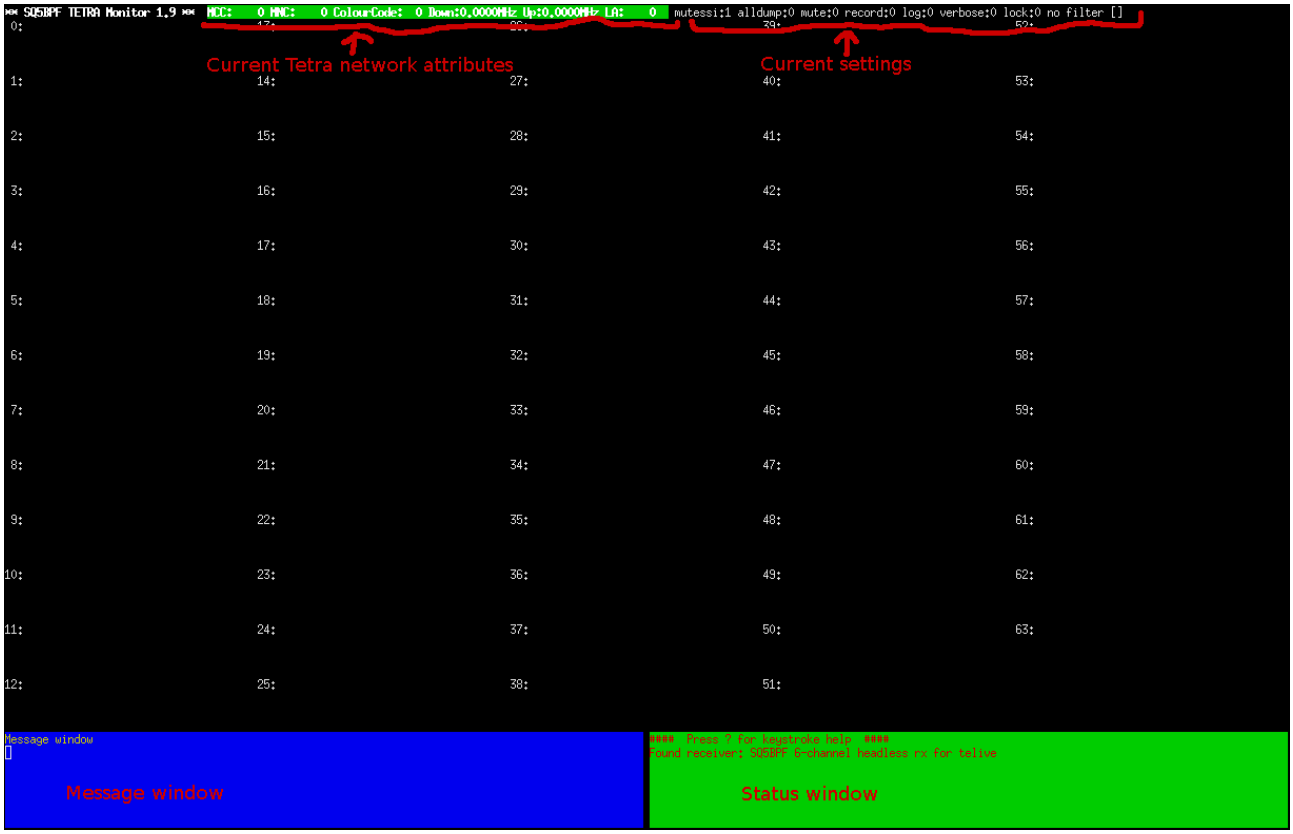

Fig 2. Telive with main window active

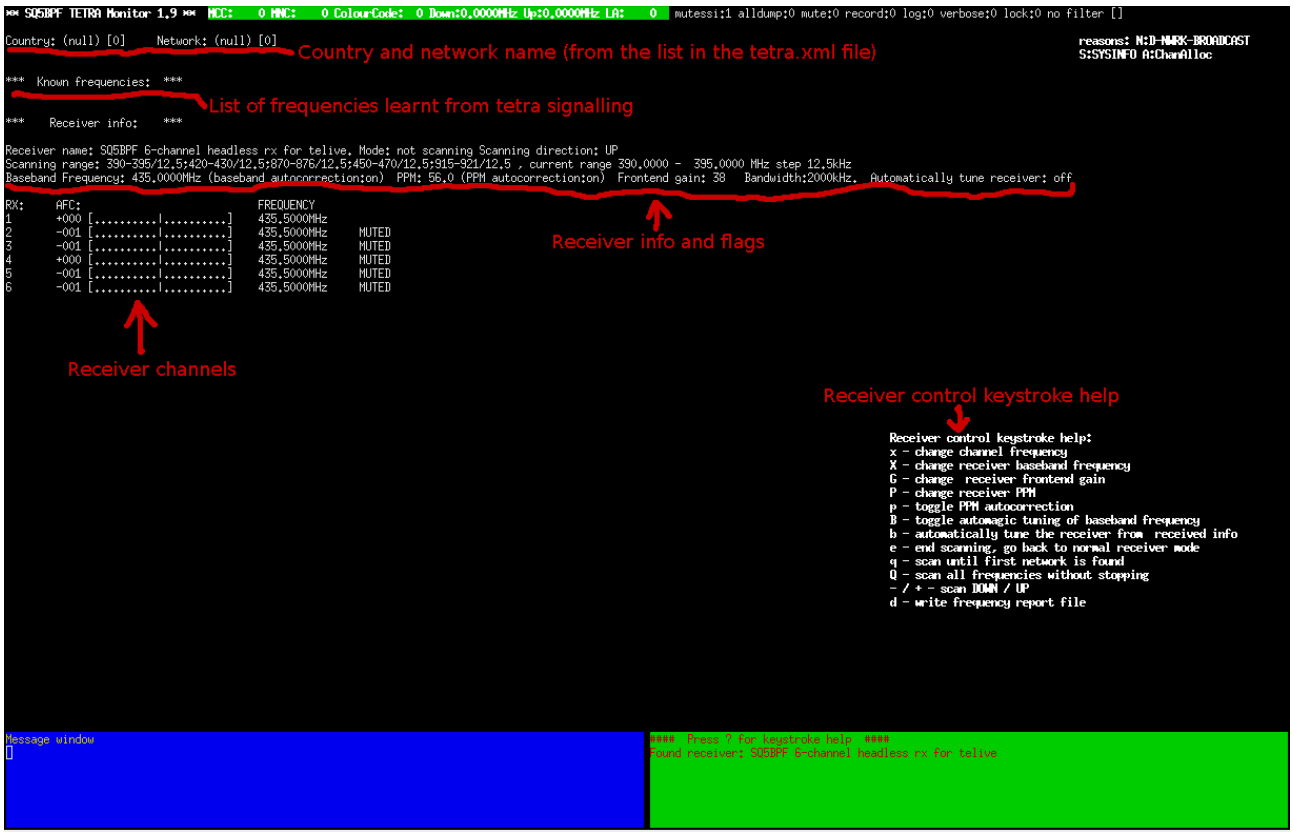

Fig 3. Telive with the frequency window active

The telive program interface has various windows, please see Fig 2 (program with the main window active) and Fig 3 (program with the frequency window active). There windows are:
- a status bar on the first line of the screen
- the main window (the black area with numbers 0-63, seen on Fig 2)
- the frequency window (the black area with the receiver description seen on Fig 3)
- the message window (blue window on the left-bottom)
- the status window (green window on the right-bottom)

The status bar displays the main program parameters: the program version, tetra network parameters and flags. The tetra network parameters are:
- MCC - country code
- MNC - network code
- CC - colour code
- LA - location area
- Down/Up - downlink and uplink frequency.

The flags are:
- *mutessi* - allow recording and playback of data for a usage identifier with no SSI data (useful, because it supressed the playback of encrypted data)
- *alldump* - don't filter signaling information, show all in the log and in the message window
- *mute* - mute playback audio (but not the recording)
- *record* - record audio
- *log* - log signalling information to file (default telive.log)
- *verbose* – verbosity level (more shows more debug info)
- *filter* – a filter that can allow or prevent certain SSIs from playing (but not recording)
- *lock* – shows locking in a setup with multiple telive instances. If 1 is displayed, then another receiver is playing like audio, and this receiver is blocked from playing

The main window (Fig 2) has a list of possible usage identifiers (0-63, where 0-2 are reserved), and aggregates signaling information (SSI addresses etc) for each usage identifier. If there are any voice frames, then this information can be recorded or played back immediately. Later the recording is renamed to a filename containing the date, time, the last 3 SSI numbers seen for this usage identifier (if known) and the call identifier (if known). If you see OK near the number then there is voice traffic present on it. If you see PLAY then this is the currently playing channel. Beside the number the detected SSIs are displayed and the call identifier in [square bractets]. SSIs can be resolved to textual descriptions using the *ssi_descriptions* file.

The frequency window (Fig 3) contains:
- the country and network name (if known), resolved from the list in the tetra.xml file
- a list of frequencies learned from tetra signaling, with information how it was discovered: N – adjacent cells from D-NWRK-BROADCAST messages, S – this network from D-SYSINFO messages, A – other channels from this LA learnt via channel allocation messages.
- receiver status. If a receiver is found via the XMLRPC interface it will display the receiver name name, gain, baseband frequency, frequency correction in PPM. Please refer to "XMLRPC receiver interface" for a full explanation
- receiver channels: a list containing the receiver id, AFC value (if known), frequency (if known) and flags (currently shows if the receiver has been muted).

The message window contains a list of tetra signalling messages received from the tetra decoder (D-SETUP, D-RELEASE etc). Please refer to the tetra documentation for a full explanation of these messages, and to the osmo-tetra-sq5bpf program source.

The status window contains a various status messages concerning the telive program state (when a new network is discovered etc). Increasing the verbosity level increses the number of messages shown. It also shows the keystroke help when ? is pressed.

Keystroke commands

Keystroke commands for the telive program:

? - show help in the status window
m - toggle *mutessi*
M - toggle *mute*
r - refresh screen
R - toggle *record*
a - toggle *alldump*
l - toggle logging
s - stop play (if there are multiple channels active, this will end the active playback and search for another channel to play).
V/v – increase/decrease verbosity
f – toggle SSI filter disabled/enabled/inverted (inverted passes traffic not matching the filter)
F – enter SSI filter expression
t – toggle between the usage identifier window, and the frequency window
z – forget all information learned by telive, read the receiver parameters again

These options are present only if a compatible receiver with the XMLRPC interface is present (keystroke help is present on the frequency window if a compatible receiver is present):
x - change channel frequency

X - change receiver baseband frequency
G - change receiver frontend gain
P - change receiver PPM
p - toggle PPM autocorrection (corrects the PPM value based on incoming AFC data)
B - toggle automagic tuning of baseband frequency (changes the baseband to fit all requested frequencies)
b - automatically tune the receiver from received info
e - end scanning, go back to normal receiver mode
q - scan until first network is found
Q - scan all frequencies without stopping
- / + - scan DOWN / UP
d - write frequency report file

Environment variables for controlling telive

Telive configuration is done via environment variables. These are usually set via the rxx script. Please read the comments in this script, and modify it to suit your needs (make a copy first). This list doesn't contain some internal tuning parameters (the rxx script also contains descriptions of some of them).

TETRA_OUTDIR – the directory where telive records voice calls. If unset /tetra/in is used (files from /tetra/in will be later recompressed into ogg files into the /tetra/out directory by the tetrad script)
TETRA_LOGFILE – the file that signalling information is logged to. If unset telive.log is used
TETRA_PORT – the udp port which is used for communication with tetra-rx. If unset 7379 is used
TETRA_SSI_FILTER – the SSI filtering expression (see the section about filtering expressions)
TETRA_SSI_DESCRIPTIONS – name of the file containing textual descriptions of SSIs. If unset ssi_descriptions is used
TETRA_XMLFILE - an xml file containing MCC and MNC codes for known networks, if unset defaults to tetra.xml
TETRA_LOCK_FILE - lock file to use between multiple instances of telive, so that they don't all play at the same time
TETRA_KEYS – contains a list of characters that will be parsed by telive, just as keystrokes would (so for example if you set it to Rff it will enable record and inverted SSI filter), don't use this for inputting the filter expression (use TETRA_SSI_FILTER for this)
TETRA_KML_FILE - if set, any received location information will be written periodically to this file in KML format
TETRA_KML_INTERVAL - this will set the maximum KML file refresh rate. If unset, this will default to 30 seconds

Environment variables for receiver control:
TETRA_GR_XMLRPC_URL - url for the XMLRPC interface of the gnuradio receiver, defaults to none (no receiver control)
TETRA_SCAN_LIST - a list of frequency ranges to scan in the form: low frequency in MHz - high frequency in MHz / step in kHz, semicolon delimited, no spaces. example: "390.2-390.3/12.5;420-420.8/12.5;425-425.8/12.5"
TETRA_FREQLOGFILE - a file which will list every newly found frequency, with timestamp (useful when scanning), if unset defaults to telive_frequency.log
TETRA_FREQUENCY_REPORT_FILE - a file where frequency info will be dumped when the d key is pressed, is unset defaults to telive_frequency_report.txt
TETRA_RX_GAIN - the receiver gain to set at startup if unset, then the receiver gain is left as-is
TETRA_RX_PPM - the frequency correction coefficient in ppm if unset, then the receiver ppm is

left as-is
TETRA_RX_PPM_AUTOCORRECT - autocorrect the PPM value from received AFC data. 1 enables, 0 disables. is unset then 1 is assumed
TETRA_RX_BASEBAND_AUTOCORRECT - autocorrect the baseand frequency to try to cover all requested frequencies. 1 enables, 0 disables. is unset then 1 is assumed
TETRA_RX_BASEBAND - the receiver baseand frequency in MHz
TETRA_RX_TUNE - semicolon delimited list of frequencies to tune the receivers to, terminated by ; (also at the end). Example: "390.100;390,200;390,300;"

Telive output files

Telive records the calls in in ACELP codec format into the /tetra/in directory (this can be changed by setting the TETRA_OUTDIR environment variable). You can use the script tplay to play them again, or the tetrad script to automatically encode them into OGG format into the /tetra/out directory (change the script to service another directory).

Telive will write several log files:
– a log with signalling information, SDS etc. This file is named telive.log (this can be changed by the TETRA_LOGFILE environment variable)
- a log with location information in KML form. In the rxx script the filename /tetra/log/tetra1.kml is used (this can be changed by the TETRA_KML_FILE environment variable)
- a log with every newly found frequency during scanning. This file is named telive_frequency.log (this can be changed by the TETRA_FREQLOGFILE environment variable)
- a report of all found networks, sorted by the MNC value. This file is named telive_frequency_report.txt (this can be changed by the TETRA_FREQUENCY_REPORT_FILE environment variable). This report is generated when the d key is pressed.

Filtering support

Telive has the ability to filter the usage identifiers which are played live based upon SSIs. There are 3 modes of operation: filtering can either be disabled, enabled (only conversations with SSIs matching the filter are played live), or inverted (only conversations with SSIs NOT matching the filter are played live). The filtering mode is toggled with f (small f).

To use a filter you have to enter a filter expression, either by pressing F (upper case f) and entering a filter expression, or by passing the expression in the TETRA_SSI_FILTER environment variable. The filter expressions are ksh extended filename matching expressions (for a better explanation please search for shell wildcard expresions, or for fnmatch and FNM_EXTMATCH). For systems which lack support for extended expressions (such as OSX) only standard expressions will work, so +(1000|1001) won't work, but 100? will.

Quick expression tutorial:

? wildcard that matches one character
* wildcard that matches any amount of characters (don't use this for SSIs, because 10* will match 100 and 10001, and this is usually undesirable)

[1-5] matches 1,2,3,4,5
[2389] matches 2,3,8,9
[this is from https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_chapter/libc_10.html]
The patterns are written in the form explained in the following table where *pattern-list* is a |

separated list of patterns.

`?(`*pattern-list*`)`
: The pattern matches if zero or one occurences of any of the patterns in the *pattern-list* allow matching the input string.

`*(`*pattern-list*`)`
: The pattern matches if zero or more occurences of any of the patterns in the *pattern-list* allow matching the input string.

`+(`*pattern-list*`)`
: The pattern matches if one or more occurences of any of the patterns in the *pattern-list* allow matching the input string.

`@(`*pattern-list*`)`
: The pattern matches if exactly one occurence of any of the patterns in the *pattern-list* allows matching the input string.

`!(`*pattern-list*`)`
: The pattern matches if the input string cannot be matched with any of the patterns in the *pattern-list*.

Examples:

1000 – match SSI 1000
10?? - match SSI 1000-1099
+(1000|[234]0??|?????) - extended pattern, matches 1000, 2000-2099, 3000-3099, 4000-4099, and any 5 digit SSIs

To have the filtering expression work upon telve startup, set the TETRA_SSI_FILTER variable to the expression string, and add f to the TETRA_KEYS variable (for example mf enables mutessi and filtering, mff wull enable mutessi and interted filtering).

Location information

The tetra-rx from the osmo-tetra-sq5pf fork has been modified to include SDS parsing, and has support for location protocols sent via SDS. Currently LIP Short Location Report, and a proprietary Simple Location System User Application 0x80 used in Spain, Germany and France (not sure what company is behind this).

If a filename is provided in the TETRA_KML_FILE environment variable, then telive will write location information to this file periodically (uncomment the line in the rxx script that sets this variable for a demo). This file can be opened via Google Earth, or any other software supporting KML.

For near realtime tracking the KML file can be referenced as a Network Link (refer to the Google Earth documentation for further information). A demo is provided in the example_google_earth.kml file – it will reload /tetra/log/tetra1.kml every 5 seconds. This will allow the station locations to be observed in near-realtime on the Google Earth map. It is also possible to serve the KML file via HTTP to multiple remote machines running Google Earth, and modify example_google_earth.kml to reference the file via a http url (a local http server like apache or tinyhttpd will have to be installed).

The maximum refresh frequency (in seconds) is set via the environment variable TETRA_KML_INTERVAL. Writing the KML file is done in the same process as the rest of telive, and thus can block the rest of telive for a moment. If you have a lot of location information, and hear a lot of stuttering when KML writing is enabled, then increase TETRA_KML_INTERVAL.

Not much location protocols are implemented at this moment. Please provide samples of telive.log with unimplemented protocol dumps, and with your approximate longitude / latitude. This will help implement them.

Text descriptions of SSIs and networks

The file ssi_descriptions contains descriptions of SSIs. These will be shown right of the SSI numbers in the telive main window. The format is:

SSI_number<exactly one space>SSI_description

Please strictly adhere to this format, no empty lines etc. The supplied ssi_descriptions file has a few examples. The filename can be overridden with the TETRA_SSI_DESCRIPTIONS environment variable.

The tetra.xml file contains a XML file with a list of MCCs (country names) and MNCs (network names). This name will be show for the country and network name on the telive screen, and in the files produced during scanning. The filename can be overridden with the TETRA_XMLFILE environment variable. If you know of a new network, then please send me the MCC/MNC, name (preferably with a description, or an url of the company website). If your country's radio authority publishes a list of TMNCs, please send me a link to it.

Receiver control via XML RPC

The gnuradio receivers can be optionally controlled by telive via a XMLRPC interface. Gain, PPM, baseband frequency and receiver offsets can be all controlled. This enables features like scanning, automatic tuning of channels and automatic correction of PPM.

The URL for the XMLRPC interface can be set via the TETRA_GR_XMLRPC_URL environment variable. Upon startup telive will query the receiver to get the name of the receiver, number of channels and their offsets, baseband frequency, gain, and frequency correction in ppm.

If telive receives AFC data from channels receiving tetra traffic, then these values can be used to autocorrect the PPM value in the receiver. This is controlled by the TETRA_RX_PPM_AUTOCORRECT environment variable. The enables compensation of the frequency correction coefficient due to temperature changes etc.

During tuning telive can try to shift the baseband frequency so that all requested channels are in the received bandwidth. This is controlled by the TETRA_RX_BASEBAND_AUTOCORRECT environment variable.

Telive can also automatically tune unused receiver channels from received data. This is controlled by the TETRA_AUTO_TUNE environment variable. If some channel is received, then the control channel frequency will be automatically tuned, and channel allocation messages from the control channel will be used to tune additional channels. In practice all that is needed is to know one random channel to tune the receiver to all channels within one LA. This feature works also with scanning for the first avaliable network: if the scanner finds a network it will stop, and if there is unencrypted traffic all channels will be tuned automatically.

It is best to use only the headless (without GUI) receivers. The WX GUI leaks memory when gui text elements change, and if the displayed frequency is changed often enough it will take

up alll avaliable memory. There are GUI receivers with the XMLRPC interface avaliable, but they aren't recommended for prolonged use.

Scanning

Receiver control via XMLRPC enables scanning. Telive has 3 modes of operation:

- normal receiver mode (no scanning). This is the default mode telive starts in. The e key switches to this mode.
- scanning until a network is found. When it is found go back to normal mode. The q key switches to this mode.
- scanning all frequencies continuously. Already known frequencies will be skipped during scanning. The Q key switches to this mode.

When a new network is found, the time and network parameters (MCC, MNC, LA, CC, frequencies) will be logged into the telive_frequency.log file. This can be changed by setting the TETRA_FREQLOGFILE environment variable. During scanning telive will remember learned information. The list of known frequencies can be dumped to a report file by pressing the d key. This is logged to the telive_frequency_report.txt file (this can be changed by setting the TETRA_FREQUENCY_REPORT_FILE environment variable).

The intended mode of operation is having an unattended receiver continuously scan all frequencies, logging every found channel to the telive_frequency.log file. If some new network appears, this will be visible in the log file, along with the time that it was found.

The default list of scanned frequencies is controlled by the TETRA_SCAN_LIST environment variable. This is a list of ranges in the format: low frequency in MHz - high frequency in MHz / step in kHz, semicolon delimited, no spaces. For example “433.1125-435/12.5;437-439/25”, this will scan starting with 433.1125MHz upto 435MHz with 12.5kHz step, and 437MHz to 439MHz with 25kHz step.

The scanning algorithm is currently very suboptimal. It uses only the first receiver channel. The algorithm will wait upto 350ms for burst messages, and if they are found it will wait upto 2s for SYSINFO messages. If the previous channel contained tetra traffic, then it will wait an extra second after tuning to the next channel. If there aren't any tetra channels found, then the scan rate for 12.5kHz step is about 30s/1MHz.

For scanning please use only the headless receivers. If the receiver will be used for scanning only, then use the 1-channel receivers: telive_1ch_gr37_udp_xmlrpc_headless.py or telive_1ch_gr37_udp_xmlrpc_headless_slow.py (256ks/s sample rate, which results in even less CPU load).

Receiver examples

The following are some receiver examples (ranging from the simplest to the more advanced). These are to illustrate various features of the gnuradio receiver/osmo-tetra-sq5bpf/telive combo.

Simple 1 channel setup example with UDP transport and no receiver control:

This is a simple receiver for one frequency only (it is best to use the control channel frequency, as it contains the most signaling information). On networks which use more channels per LA it will not record all calls, because they may be on other frequencies.

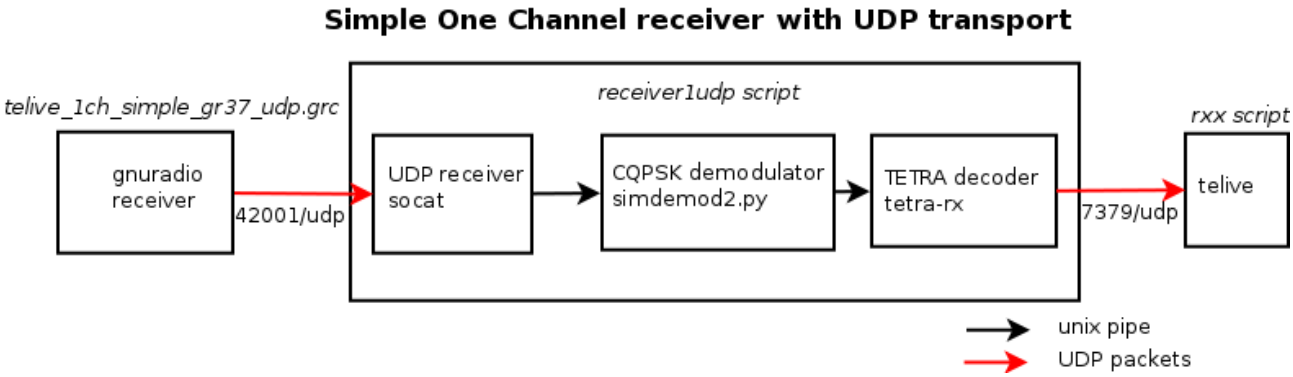

Fig 4. Simple one channel receiver

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute
./receiver1udp 1
There should be no errors. This listens on UDP port 42001 , and listens for incoming data, piping it into simdemod2.py, and tetra-rx. The resulting data is sent via UDP packets to port 7379 to the telive process.

Open a 203x60 characters xterm with:
/usr/bin/xterm -font fixed -bg black -fg white -geometry 203x60
Change to the telive directory and execute:
./rxx

This launches the telive console. In the telive console pressing shift-R enables recording of voice calls, and pressing L enables logging.

In another xterm launch /tetra/bin/tetrad - this will recode voice in the ACELP format into OGG format, and put it into /tetra/out

Connect a rtl-sdr dongle and antenna. Open telive_1ch_simple_gr37_udp.grc (from the telive/gnuradio-companion/receiver_udp directory) in gnuradio-companion, and run it. Currently telive_1ch_simple_gr37_udp.grc defaults to 391MHz and frequency correction to 56ppm. Please adjust the frequency, and ppm to receive a known strong tetra signal (preferably unencrypted). Frequency values in gnuradio should be entered in Hz (without the unit), k M prefixes are allowed. So to enter 435.225 MHz please enter 435.225M (and press the Enter key), to enter -112.5kHz use -112.5k (and press Enter).

If necessary, correct the ppm value so that the spectrum looks "symmetrical". It is also possible to click on the spectrum display to tune the receiver. This can also be adjusted by observing the AFC value in the frequency window (set PPM so that it is close to 0).

If all is set up correctly, the xterm where receiver1udp is run should scroll a lot of text, while the telive console should display the MCC, MNC, Colour Code and uplink/downlink frequencies for the control channel. If there is any traffic you should see some SSI numbers on the console, and maybe hear some voice (voice is muted with the key command shift-M).

Six channel setup example with UDP transport

This is a receiver for six frequencies, one for the signaling channel, and the rest for other channels from the same network The setup is similar to Fig. 2, but there are six receiver1udp processes, and gnuradio receives six channels simultaneously. This enables traffic analysis on all channels from one LA. Analysing more channels requires much more CPU resources, if there is not enough CPU avaliable, then the examples can be scaled to a smaller number of channels. Please refer to Fig 5

Six Channel receiver with UDP transport, all channels from the same LA

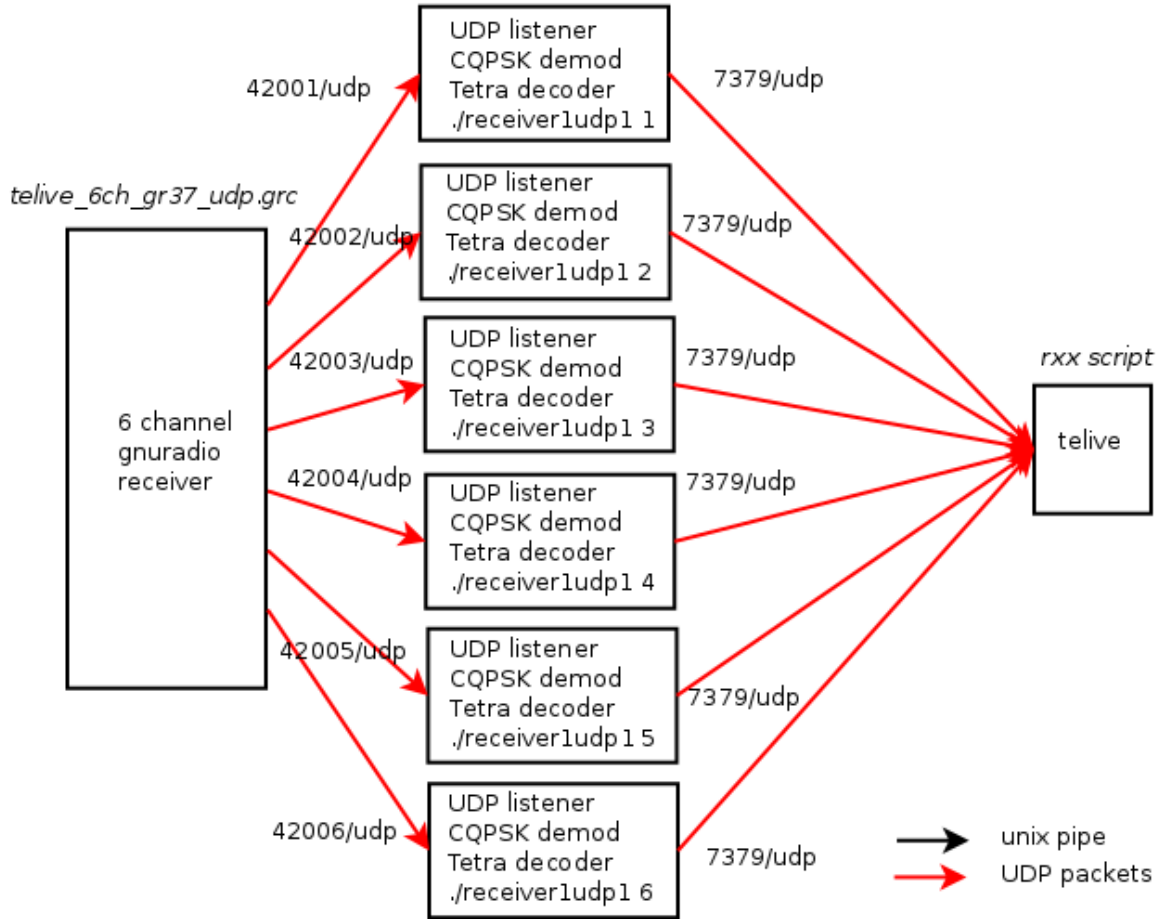

Fig 5. Six channel receiver setup

Open an 6 xterm terminals, in each of them change directory to the osmo-tetra-sq5bpf/src directory and execute:

`./receiver1udp 1` (in the 1st terminal)
`./receiver1udp 2` (in the 2nd terminal)
`./receiver1udp 3` (in the 3rd terminal)
`./receiver1udp 4` (in the 4th terminal)
`./receiver1udp 5` (in the 5th terminal)
`./receiver1udp 6` (in the 6th terminal)

Alternatively this can be done automatically, change to the osmo-tetra-sq5bpf/src directory and execute:

```
for i in  `seq 1 6`; do xterm -T RX$i  -e ./receiver1udp $i & done
```

There should be no errors. This creates 6 receiver1udp instances, listening to UDP ports 42001,

42002, 42003, 42004, 42005, 42006, and sending data to the telive process with receiver IDs 1, 2, 3, 4, 5, 6. The resulting data from these processes is sent via UDP packets to port 7379 to the telive process.

In another xterm launch /tetra/bin/tetrad - this will recode voice in the ACELP format into OGG format, and put it into /tetra/out

Open another xterm with:
```
/usr/bin/xterm -font  fixed -bg black -fg white -geometry 203x60
```
Change to the telive directory and execute:
```
./rxx
```

Connect a rtl-sdr dongle and antenna. Open telive_6ch_gr37_udp.grc (from the telive/gnuradio-companion/receiver_udp directory) in gnuradio-companion, and run it. Please enter the frequency correction in ppm. Please adjust the baseband frequency to somwhere in the middle between all frequencies that are to be received (the received channels can't be more than 1MHz from the baseband frequency if the dongle is operating on 2MHz sampling rate). Please input the difference from the basband frequency to the channel frequencies. For example to receive channels 435.400 MHz, 435.500 MHz, 435.600 MHz, 436.200 MHz, 436.400 MHz , 436.500 MHz, one could set the baseband to 436M, and the offsets to -600k (because 436MHz-0.6MHz=435.400MHz), -500k, -400k, 200k, 400k, 500k.

If necessary, correct the ppm value so that the received channel spectrum looks "symmetrical". This can also be adjusted by observing the AFC values in the frequency window (set PPM so that they all are close to 0).

Please note that you will get more data then with the 1 channel receiver setup. If the tetra network attributes change in telive, then the two channels are not from the same network, and this won't work correctly. In this case the “too many changes, are you sure you're listening to the same network” message appears in the statis window.

Four channel, two networks setup with named pipe transport

This is a receiver for four frequencies, to monitor two separate Tetra networks, each with two channels. Named pipes are used instead of UDP transport from the receiver to the demodulator/decoder processes, because this is more reliable. Using UDP for transport can result in lost frames, especially during high CPU load. Also using UDP packets results users more resources than named pipes. The downside is that the process consuming the output from the pipe (the receiver1 process) has to be started first, and the process feeding the pipe (the gnuradio receiver) has to be started after that. Each time this sequence has to be repeated. This has been a problem for new users, and therefore most examples are based on the UDP transport.

Four Channel receiver with unix pipe transport,
2 channels from one LA,
2 channels from a different LA (or different network)

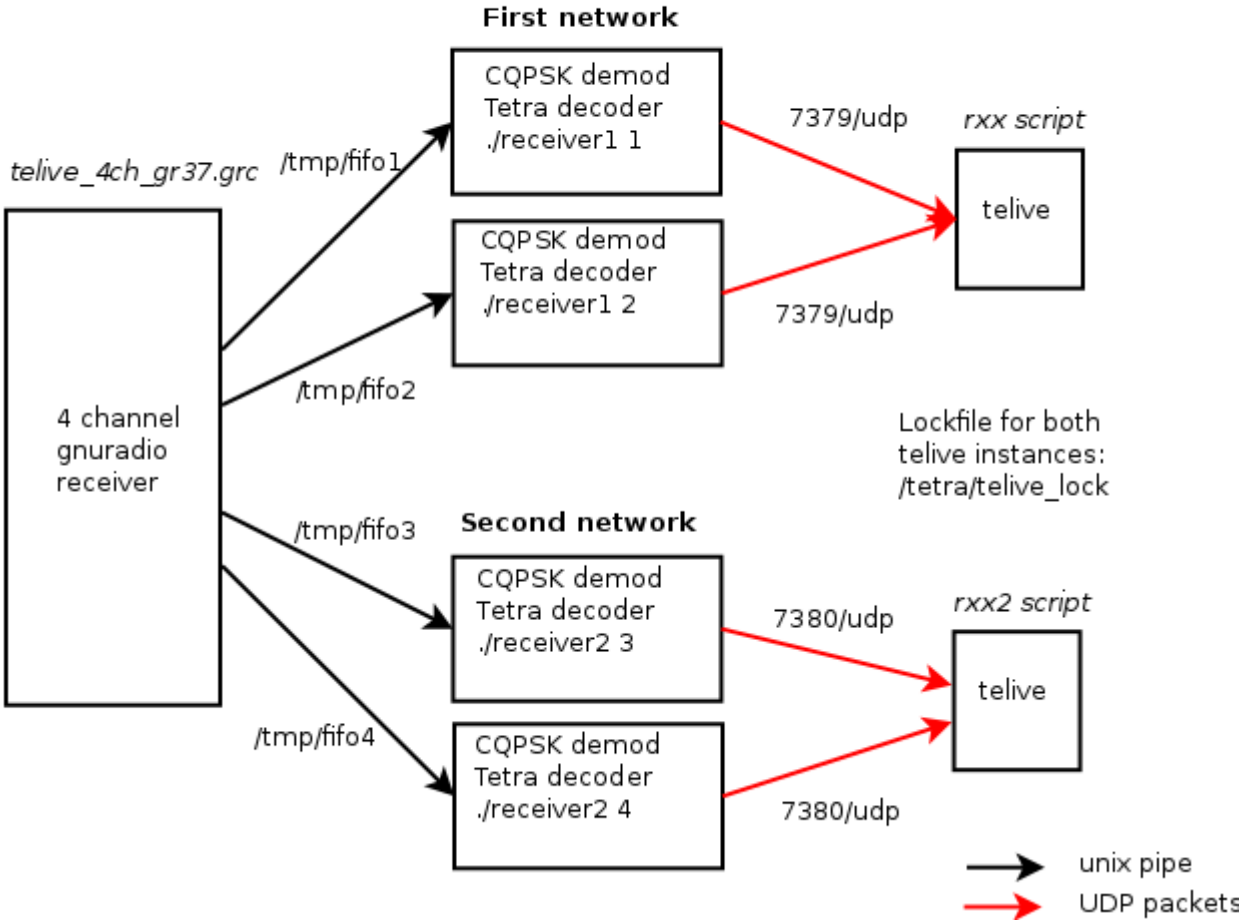

Fig 6. Four channel, two networks setup

Software servicing Network 1:

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute
./receiver1 1
There should be no errors. This creates /tmp/fifo1 , and listens for incoming data, piping it into simdemod2.py, and tetra-rx . This process will send data via UDP to 7379/udp.

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute
./receiver1 2
There should be no errors. This creates /tmp/fifo2 , and listens for incoming data, piping it into simdemod2.py, and tetra-rx . This process will send data via UDP to 7379/udp.

Open another xterm with:
/usr/bin/xterm -font fixed -bg black -fg white -geometry 203x60
In the new xterm window that opened:
Check with stty -a that there are indeed 60 rows and 203 columns.
Change to the telive directory and execute:
./rxx
This telive process will listen to 7379/udp

Network 2:

Create a directory /tetra2 which is a copy of /tetra. Setting up the various paths is left as an exercise to the reader.

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute
./receiver2 3
There should be no errors. This creates /tmp/fifo3 , and listens for incoming data, piping it into simdemod2.py, float_to_bits and tetra-rx . This process will send data via UDP to 7380/udp (this is a different port from network 1, which uses 7379/udp. This value is set via an environment variable in the receiver2 script).

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute
./receiver2 4
There should be no errors. This creates /tmp/fifo4 , and listens for incoming data, piping it into simdemod2.py, float_to_bits and tetra-rx . . This process will send data via UDP to 7380/udp.

Open another xterm with:
/usr/bin/xterm -font fixed -bg black -fg white -geometry 203x60
Change to the telive directory and execute:
./rxx2
This telive process will listen to 7380/udp (This value is set via an environment variable in the rxx2 script).

Connect a rtl-sdr dongle and antenna. Open telive_4ch_gr37.grc (from the telive/gnuradio-companion/receiver_pipe) in gnuradio-companion, and run it. Currently telive_4ch_gr37.grc defaults to 435.500MHz, 434.750MHz, 435.750MHz, 435.800MHz and 56ppm. Please adjust the tuner baseband frequency, offsets and ppm to receive a known strong tetra signals (preferably unencrypted) on all channels. Correct the ppm value so that the spectrum looks "symmetrical" (or use the AFC values).

Having two telive instances play voice at the same time can be confusing, so a common lockfile can be used to stop this. While one telive instance is playing, the other one will be muted. Set TETRA_LOCK_FILE=/tetra/telive_lock to do this.

Six channel setup example with UDP transport and receiver with no GUI controlled via XMLRPC

This is very similar to the “Six channel setup example with UDP transport” example, however the receiver has no GUI, and is controlled via the XMLRPC interface from within telive. The GUI uses CPU resources, and often is not needed.

Having telive control the receiver enables:
- automatic tuning of channels from the received signalling information. It is enough to tune in one channel, and the rest (the control channel and other channels) will be automatically tuned.
- scanning until a channel is found. This can be combined with automatic tuning, the first channel is scanned, and the rest of the channels are automatically tuned from the first network found.
- continuously scanning for all networks. The telive program keeps a list of all discovered frequencies, and excludes them from further scanning. It also produces a log with newly found frequencies and the time they were found. If the telive program is left to scan the spectrum for a long time, this can be used to show when new channels are activated (for short-lived networks etc). A report of all frequencies found, sorted by MNC can be produced by pressing the d key.

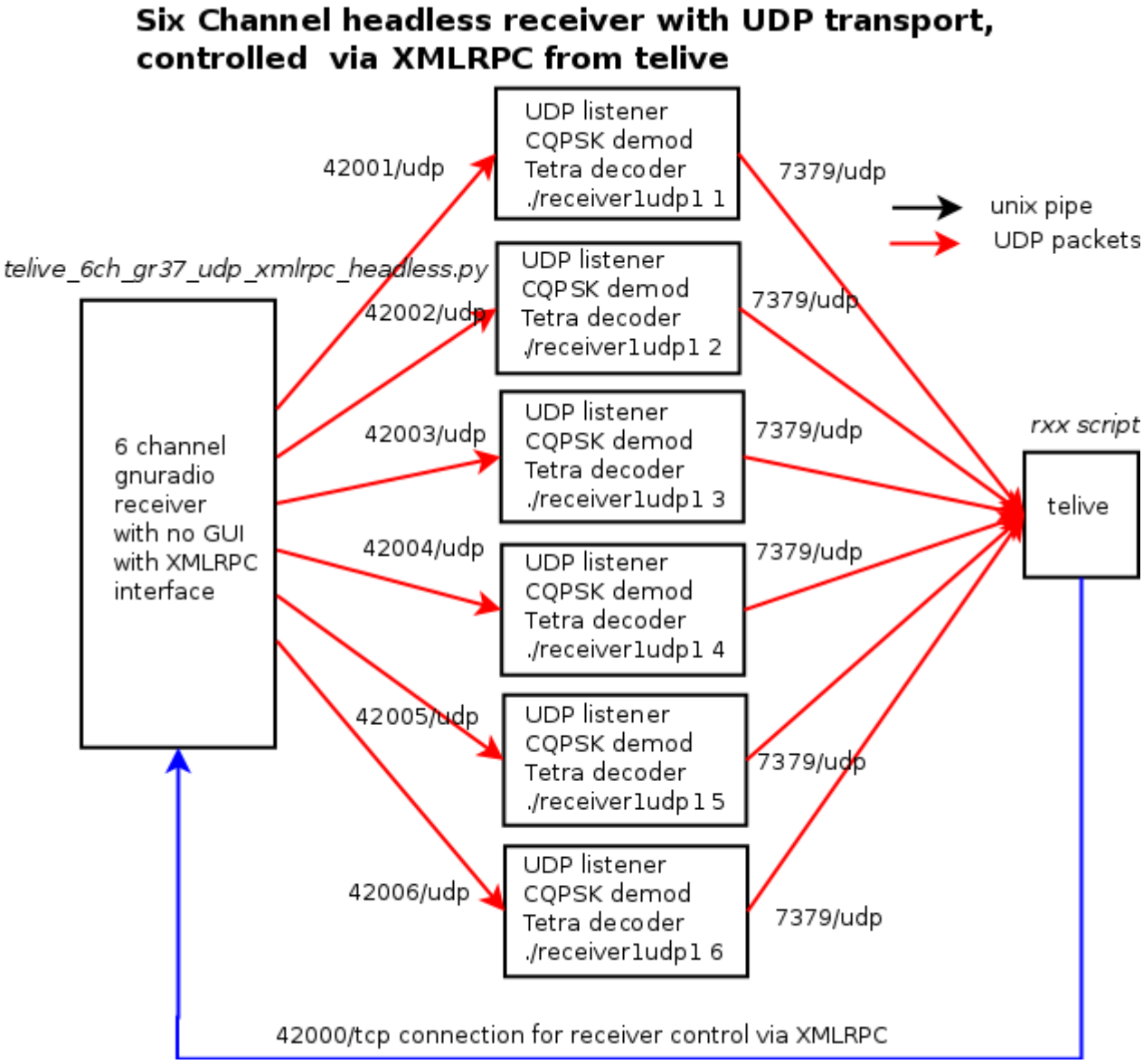

Fig 6. Six channel receiver setup, without receiver GUI and with control via XMLRPC

Open an xterm, and change the directory to telive/gnuradio-companion/receiver_xmlrpc, and execute:

```
./kill_wrapper ./telive_6ch_gr37_udp_xmlrpc_headless.py
```

Automatically open an 6 xterm terminals, in each of them change directory to the osmo-tetra-sq5bpf/src directory and execute:

```
for i in  `seq 1 6`; do xterm -T RX$i  -e ./receiver1udp $i & done
```

There should be no errors. This creates 6 receiver1udp instances, listening to UDP ports 42001, 42002, 42003, 42004, 42005, 42006, and sending data to the telive process with receiver IDs 1, 2, 3, 4, 5, 6. The resulting data from these processes is sent via UDP packets to port 7379 to the telive process.

Open an xterm, change directory to the osmo-tetra-sq5bpf/src directory and execute

Open another xterm with:
/usr/bin/xterm -font fixed -bg black -fg white -geometry 203x60
Change to the telive directory and execute:
./rxx_xmlrpc_example

The rxx_xmlrpc_example script should be edited first to set desired receiver parameters, at least the PPM value should be set.

In the telive screen press t – this changes to the frequency window. The frequency window should show the receiver name and parameters.

Press x – this tunes a receiver. Enter the receiver number and frequency in MHz separated by one space (for example: 1 435.1125) and press Enter. Refer to the keystroke help to set other receiver parameters on the fly: gain, PPM etc.

Press q – this scans until the first avaliable network is found. Since automatic tuning of unused channels is enabled, this will tune other receiver channels too when traffic is present and the network is unencrypted.

Press Q – this scans all networks, and repeats the process. Scanned frequencies and the time they were found are logged in the telive_frequency.log file. Pressing d will dump a report of all found frequencies to telive_frequency_report.txt . For continuous scanning it is better to use a 1 channel receiver: telive_1ch_gr37_udp_xmlrpc_headless.py or a 1 channel receiver with 256ks/s sample rate (takes even less CPU, but can have aliasing problems):
telive_1ch_gr37_udp_xmlrpc_headless_slow.py.

Using GUI programs for longer scanning is discouraged, because of problems with memory leaks in the gnuradio-companion WX GUI.

Other setups

This software is a combination of loosely coupled modules, so that the user can easily exchange them, or connect them in a different way. This architecture makes it much more flexible than a if this was one big monolithic program.

Please look at the scripts, the gnuradio-companion flowgraphs and the program sources. You can use multiple gnuradio receivers each with its own rtl-sdr dongle, and feed the results via named pipes or UDP to multiple receiver processes. The results can be displayed via multiple telive processes, each for one Tetra network.

The grc flowgraphs don't have any additional python code in them, all functionality has been achieved with what gnuradio-companion allows. This makes it very easy to modify them in gnuradio-companion without any programming knowledge.

Please see the telive/gnuradio-companion directory and it's subdirectories, and read the README.txt files. There are 3 directories:

receiver_pipe - grc flowgraphs with named pipe tansport
receiver_udp - grc flowgraphs with udp transport
receiver_xmlrpc - grc flowgraphs with udp transport and xmlrpc interface for receiver control

End notes

This is code that i've written for my own pleasure. It is very ugly, but i've could either polish it so it looks nice, or "release early, release often" (per the open source methodology). Polishing it would take forever, and i believe that if one has a nice toy, he should share it with other children, rather then keep it to himself.

This code runs under linux, and was tested under Debian 8 64-bit version. Other debian-like distributions can also work if they have a recent enough gnuradio (for example Linux Mint 17.3 is known to work). I will not answer questions how to make it work under operating systems different than linux, however if you make it work, send me a description, so that i can put it into the documentation. Also i haven't really tried to make this user-friendly. The documentation is crap, despite my best efforts to write it. Basically read all of the documentation, the ETSI Tetra docs, and if you still don't understand something, then RTFS.
The architecture is modular, and you could probably use another receiver instead of gnuradio (like rtl_fm). This hasn't been tried, but should work, and should result in less CPU utilization.
The telive program uses the usage identifier as the key. This concept works, but is not correct. A more correct approach would be to use the notification id and receiver id. This will probably be implemented in some later version. The SDS decoding might not work in many cases. So far i've seen only type 0x82 (Text messaging), 8 bit encoding type SDS. 7-bit decoding might not work etc.

If you can provide samples of interesting traffic legally, then please do. It would be best if you have your own TMO Tetra network, that you could monitor, while changing various settings etc.

The original osmo-tetra software, and osmo-tetra-sq5bpf (which is a forked version) don't understand fragmentation, so for example long SDS messages (which are sent as fragments) don't work.

The core of this software: osmocom-tetra, libosmocore was written by Osmocom (i've only filled in some blanks, that others probably didn't want to fill in, such as the SDS decoder etc). As stated on their web page http://tetra.osmocom.org/trac/wiki/FAQ "Can I use OsmocomTETRA to listen to police radio?
[...] Also, if this is your interest in this project: Please simply go away, we don't want to talk to you. " - please respect this.

Disclaimer

I disclaim any liability because of the use of this software. If someone breaks something it's their fault. Also please observe the licenses. The telive software is licensed GPLv3. The osmosom-tetra software is licensed GNU Affero v3, and libosmocore is licensed GPLv2. The codecs are licensed by ETSI, and if i understand correctly, you can build binaries for yourself from the publicly available source code, but might not be able to provide the binaries to others. IANAL, so don't take this as legal advice, if someone knows better, then please correct me on this.

TODO

Implement a better protocol to communicate with telive
Implement Location Information Protocol and Data Services

Don't use popen() for playback because of buffering.
Actually try to read the ETSI docs and not fall asleep after 20 pages.
Clean up the code, rewrite all comments in english (if there are any polish comments or variable names left).
Write proper documentation

FAQ

What does this software do?

It enables one to listen and record unencrypted TETRA calls, decode some SDS messages and log signalling information.

Where can I learn more about TETRA?

The TETRA standard is avaliable from ETSI, go to http://etsi.org and search for TETRA. You will get more documents than you will ever want to read. Please note that TETRAPOL is something completely different (the only similarity being the first 5 letters :).

What operating systems does it run on?

It runs on linux. Currently debian 8 is the preferred platform. It was developed under Debian 6.0 64-bit and Debian 7.0 64-bit, later Debian 8 with gnuradio 3.7. Others have reported that it works under Kali linux, Mint and Ubuntu. Any Unix/Linux operating system should work with some tweaking, if you can get gnuradio working on it.

Does it work on Microsoft windows?

No. And i'm not interested in it. However you can run it in a virtual machine. Beware that a virtual machine doesn't run as fast as a normal installation, and might not be suitable for more advanced setups (monitoring multiple channels etc). A better approach is to temporarily boot linux from a USB pendrive, ready images with the telive software are published from time to time on the radioreference forums.

I don't understand all this, could you provide a nice description how to get it to work for people who don't know much about computers or radio?

Not really. Go away. And please don't send emails asking for this. Thanks!

If Linux is a problem, then go find some friendly linux user with at least basic skills (ie. one that can do something more then point the cursor at icons and press a button), and he will be able to help you.

What is a usage identifier (also called usage marker)?

From the tetra docs: “A traffic usage marker is a 6-bit MAC label used during circuit mode calls for transmitter preemption, for prevention of crossed calls and for channel maintenance purposes. The BS shall assign a traffic usage marker before any traffic transmission takes place on an assigned channel.”

This identifier is present during various call setup procedures, and also accompanies voice frames. This identifier is used by telive to bind voice traffic with signaling information. The identifiers 0-3

are reserved, and you should not see them in use in telive.

How can I record calls?

Press shift-R (the top line should read record:1). The calls are recorded in ACELP format in the directory /tetra/in. The script tetrad will recompress them into OGG files and put them in:
/tetra/out/YYYYMMDD/traffic_YYYYMMDD_HHMMSS_idxUU_callidZZZ_SSI1_SSI2_SSI3.ogg
YYYYMMDD is year month date (like 20141127)
HHMMSS is hour minute second (like 230145)
UU is the usage identifier
ZZZ is the call identifier
SSI1, SSI2, SSI3 are the last 3 SSI numbers associated with this usage identifier

How can I log signalling?

Press L (the top line should read log:1). The log is in telive.log (this can be changed by setting the environment variable TETRA_LOGFILE). This log contains the signalling information in a readable form (not as long as the tetra-rx output), and SDS messages (the text messages should be decoded).
The most information is present in the tetra-rx output. To log it do:
./receiver1 1 2>&1 |tee -a /tmp/logfile
Beware that this will eat away many megabytes of disk space per minute.

I don't hear anything or something stutters, but the recordings are fine if I play them on another system

This has nothing to do with telive. Find some wave file and play it using aplay. Fiddle with the system until it's fixed (maybe change the mixer settings? See support forums for your distribution for possible fixes).

The telive program exits when it sees voice traffic. Why?

There is a problem running /tetra/bin/tplay , or a problem running sdecoder/cdecoder/aplay. Note: as of version 0.9 telive should no longer exit, but print an error message.

How can I check if the codecs are compiled ok, and if audio works?

testfile.acelp contains a compressed voice sample. Please try:
/tetra/bin/tplay < testfile.acelp
You should hear “Hello Tetra”, if not debug the tplay script, sdecoder/cdecoder/aplay. Also look at the mixer settings, maybe the audio has been muted?

Something segfaults, how can I help debug it?

Please email me the result of these commands:
gdb program_that_segfaulted core
bt

The usage identifiers at the bottom are all on the same line, like 9:10:11:12: etc

This means that the xterm screen size is not 203x60 (maybe your window manager resized it). The

software should still work, but the screen will be a bit unreadable.

I get fseek failed from receiver1

This means that something has closed /tmp/fifo... Most probably the gnuradio-companion receiver has died. Check for errors in the gnuradio-companion console. This happens only when data is sent via named pipes.

I hear mostly unreadable audio

This is probably audio from encrypted calls, or some non-voice data. Enable mutessi – this will ignore usage identifiers which don't have at least one SSI number assigned. BTW for this reason mutessi is enabled by default on recent telive versions.

Can I have the software start with some defined state, so that I don't have to change any settings when it starts?

For telive look at the environment variables that it uses. For example in rxx you can put:
export TETRA_KEYS=RMl
This will work exactly the same as if the keys R (record), M (mute) and l (logging) were pressed.

For osmo-tetra-sq5bpf look at the receiver1 script, the UDP port number and receiver number can be set there.

For the gnuradio-companion flowgraphs it is best to make a copy of the flowgraph, load it into gnuradio-companion, edit the variables (like ppm, frequency, offset etc), and save it again. The flowgraph can also be compiled into a python script, that can be launched directly without gnuradio-companion (click Build->Generate, a file with a .py extension will appear in the same directory as the grc flowgraph).

Can I use a different receiver?

You can use any receiver that will be able to write baseband data to a pipe in the same format as gnuradio does.

How can I manually compile/install this software?

This section is intended only for people who would like to install on distributions which are not supported by the easy install procedure using install_telive.sh. Please use the easy install procedure whenever you can. Please study the install_telive.sh script first anyway.

Packages from your distribution

Install oggenc (package vorbis-tools under debian), sox (package sox), aplay (package alsa-utils), ncurses development libraries (package libncurses-dev). You will need all of the development packages, but that will be installed by build-gnuradio. If you have installed any gnuradio packages, then uninstall/purge them.

Gnuradio 3.6 or 3.7.x (3.7.5 or higher) with osmosdr support.
Please see this page:

https://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGRFromSource

Probably the easiest way is to use the build-gnuradio script provided by SBRAC:

http://www.sbrac.org/files/build-gnuradio

Please run ./build-gnuradio (installs gnuradio 3.7)
or ./build-gnuradio -o (the -o parameter installs version 3.6, use this only if you need version 3.6 for some other software).

After building, test if the software works. There are many gnuradio-companion scripts on the internet, see if they work, and also use them to find the right ppm value for your rtl-sdr dongle.

libosmocore-sq5bpf

This is Osmocom libosmocore, the original software is here:
http://bb.osmocom.org/trac/wiki/libosmocore
Please read all documentation for this project.
The version in my repository is not patched in any way, but may be patched in the future.
To compile and install:
git clone https://github.com/sq5bpf/libosmocore-sq5bpf
cd libosmocore-sq5bpf
autoreconf -i
./configure
make
sudo make install

osmo-tetra-sq5bpf

This is a patched version of Osmocom osmo-tetra, the original software is here:
http://tetra.osmocom.org/trac/wiki/osmo-tetra

To compile:
git clone https://github.com/sq5bpf/osmo-tetra-sq5bpf
cd osmo-tetra-sq5bpf
cd src
make

The scripts receiver1 and receiver2 assume that you are in this directory (osmo-tetra-sq5bpf/src).

Tetra codecs

Please read the instructions in osmo-tetra-sq5bpf/etsi_codec-patches , including README_sq5bpf. This shows how to obtain and compile the codecs. Put the tetra codecs in /tetra/bin (created in the next step).

To compile/install:
 git clone https://github.com/sq5bpf/install-tetra-codec
 cd install-tetra-codec
 chmod 755 install.sh
./install.sh

telive

To compile:
git clone https://github.com/sq5bpf/telive
cd telive
make
sudo mkdir /tetra
sudo chown YOURUSER.YOURGROUP /tetra
chmod 755 install.sh
./install.sh

The scripts rxx and rxx2 assume that you are in this directory.

tetra-rx reports Air Encryption:1, does this mean that all is encrypted?

No, this means that someone has paid money for the encryption license for their TETRA infrastructure. To use encryption each radio needs to have encryption enabled too, which also costs. So probably there will still be some radios (which are not used for secret communications), without encryption.

How do I find the ppm value quickly?

Although not recommended, this can be done also with this software. Set up everything up as in the "Simple 1 channel setup example", and set the frequency to a known local strong transmitter. This doesn't have to be tetra, I usually use the VOLMET from the local airport for this. Now set the ppm slider so that the channel signal spectrum is symmetrical to 0kHz (in case of AM the carrier should be at 0kHz on the spectrum graph). The ppm value drifts with temperature, so determining the ppm should be done after the receiver has warmed up after at least 10 minutes.

The receiver doesn't work, no matter what frequency I tune it to. Gnuradio complains about PLL unock

You are probably not using the right format. Gnuradio expects to have frequency input in Hz. You can also use prefixes like k for kilo, M for mega etc. Quick example in the 1-channel demo: to listen to 435.2125MHz, you could use 435MHz baseband frequency and 212.5kHz offset (or for example 436MHz baseband and -787.5kHz offset). To tune enter 435M (not 435MHz!) for baseband frequency and press Enter, enter 212.5k for offset.

I get a warning about terminal size in the status window

The terminal size is different than 203x60.The program will still work, but the display may be mangled.

I get "PLAYBACK PROBLEM!! (fix tplay)" in the status window

There is a problem running tplay for live listening. Most probably codecs are not avaliable, there is some permissions problem etc. Fix tplay, and verify that it works by playing the acelp test file, and restart telive.

I get "Too much changes. Are you monitoring only one cell?" in the status window

The network parameters MCC/MNC/LA/Colour Code/frequencies are changing too fast. This is common in a multi-channel setup, where the channels are from different cells. Please use one telive instance to monitor only channels from one cell, otherwise it will get confused. For monitoring multiple cells multiple telive instances can be used (this is described in the “Four channel, two networks setup” exmaple).

I get errors about the device being claimed by another driver

Probably the dvb_usb_rtl28xxu driver is loaded, this is the driver that can be used to watch TV using these dongles. To disable the module loading create a file /etc/modprobe.d/rtlsdr.conf containing the text:
blacklist dvb-usb-rtl28xxu
After this reboot

I get strange errors, I'm using Ubuntu

Some people report that a reboot solves these problems.
Ubuntu is a strange case generally, because it is preferred by non-technical users, and it is hard to say if these are real problems caused by the distribution itself, or if they stem from the users' lack of ability to use the system and to RTFM.

I installed another gnuradio version and everything broke. How do I fix it?

The easiest way would be to reinstall the system. This can also be caused by installing a package that has gnuradio as a dependency (gqrx etc), when another version (not from the package manager) was already installed.

I have trouble when downloading software using the build-gnuradio script

Either the repository is not available at this time, or you have some Internet connectivity problems. Wait a few hours and try again.

On which frequencies can I find tetra signals?

According to wikipedia, in Europe the downlink signals are on:
emergency services:
390-395MHz
commercial services:
395-400MHz
420-430MHz
460-470MHz
915-933MHz

Countries outside of Europe will probably have different bandplans.

OMG! Someone can listen to my secret tetra transmissions! The sky is falling!

Well, actually not. All these digital systems (Tetra, DMR, NXDN etc) have an option to either encrypt traffic or not. This is a major advantage over analog systems, where it was hard to encrypt. Now if a system doesn't have encryption enabled, then one can assume that it is a conscious choice

of the system designer, and that the system is intended to be open to monitoring.

Of course the encryption algorithms are not public, so it is hard to tell if there are any flaws in them. Probably sooner or later someone will reverse engineer or leak them (as was with GSM A5/1 and other algorithms). And since there are not public, they were not subjected to public scrutiny, and may contain weaknesses (as was with GSM A5/1 and other algorithms). Please consider that all your encrypted traffic might be recorded today, and broken a few years from now. This is true of any technology, not just tetra.

But publishing this software makes it possible to monitor something that was not "monitorable" before. This changes everything

Well, actually not. The protocol specifications were publicly available for a long time and osmocom-tetra code has been available since 2011, patches to record audio have been floating around, and people had developed their own private versions. Also there is commercial software that enables TETRA monitoring like w-code from Wavecom. Publishing this code just makes the process less magical, and security hates magic.

Changelog:

20160620: added info about v1.9, beta version, if you find errors/omissions in this document then please let me know --sq5bpf
20151030: added info about version 1.5 and about install_telive.sh --sq5bpf
20150228: added info what TETRA_KML_INTERVAL is for --sq5bpf
20150227: added info about version 1.0 and using location information --sq5bpf
20150130: added info about version 0.9 --sq5bpf
20141208: added info about testfile.acelp --sq5bpf
20141207: added info about telive_1ch_simple.grc --sq5bpf
20141207: added a bit to the FAQ --sq5bpf
20141206: Changed telive version to 0.8, added a bit to the FAQ --sq5bpf
20141127: Added the FAQ. Clarified a few things –sq5bpf