LTE Probe

Version: 2025-05-21

Table of Contents

1 Introduction

lteprobe is a real time LTE downlink and uplink passive protocol logger. It can also be used to get the temporary identifiers (C-RNTI, TMSI) of UEs. It takes I/Q samples as input from SDR devices. All the signal processing is done on the PC CPU.

2 Features

- Supports FDD bandwidths from 1.4 to 20 MHz for downlink and uplink.
- Automatically acquires the cell parameters from the System Information Blocks.
- Decodes both downlink and uplink signals.
- Supports MIMO 2x2 and tranmission modes 1 to 3.
- Tracks several UEs and bearer contexts at the same time.
- Supported PHY messages: PDCCH, PCFICH, PHICH, PDSCH, PRACH, PUSCH.
- Log PHY, MAC, RLC, PDCP, RRC, NAS and IP messages in textual logs compatible with the other Amarisoft tools.
- Plots for QAM constellations and channel response.
- Log received IP packets to Wireshark compatible PCAP files.
- Keep a journal of connected UEs with eNodeB and local timing advances (for approximate UE location), PRACH SNR, RNTI and TMSI.
- Log all paging messages.

3 Requirements

3.1 Hardware requirements

- A PC:
  - Quad core Intel CPU (Core i5 or Core i7) supporting AVX2 (i.e. a CPU using at least the Haswell microarchitecture).
  - At least 2 GB of RAM.
  - At least 1 GB of hard disk space.
  - The video adapter does not matter.
- Radio front end:
  - 2 PCIe SDR boards (FDD) or 1 PCIe SDR board (TDD)
- Appropriate antennas for the intended LTE frequencies or cables and attenuators to connect to UE(s) and eNodeB(s).

3.2 Software requirements

- A 64 bit Linux distribution. Fedora 39 is the officially supported distribution.
  The following distributions are known as compatible:
  - Fedora 22 to 39
  - Cent OS 7
  - Ubuntu 14 to 22

  Your system requires at least GLIBC 2.17.

  Other distributions can be used provided the radio frontend drivers are available for them.

4 Installation

4.1 Radio front end setup

Two PCIe SDR boards must be installed in the PC for FDD operation. They must be time and frequency synchronized (connect them with the clock cable). GPS synchronization is not needed.

Read the PCIe SDR documentation (`trx_sdr.pdf`) to have more information about the hardware and software setup.

4.2 LTEPROBE installation

Decompress the LTEPROBE archive to a convenient place. The executable `lteprobe` can be launched from this directory.

4.3 Initial testing

Update the configuration file `config/probe.cfg` to select the EARFCN of a known LTE carrier (modify the line with `dl_earfcn`). Then start lteprobe:

```
./lteprobe config/probe.cfg
```

If an LTE carrier is found, the following messages are displayed:

```
got SIB1
got SIB2
```

If you don't see them, check the EARFCN and cabling.

Then for each received PRACH the following message is displayed:

```
PRACH: sequence_index=x ta=y snr=z dB
```

All the receiving uplink and downlink messages are written to the text log `/tmp/probe.log`. The log of detected UEs is in `/tmp/profue_ue.log`. The decoded IP packets can be analyzed with Wireshark using the PCAP file `/tmp/probe.pcap`.

4.4 Useful tips

- If you want to log all the RRC and IP messages, be sure that neither RRC nor DRB data are encrypted. If the RRC is encrypted, `auto_rb_add` can be set to log some RLC/PDCP information, but it cannot work reliably because the RRC reconfiguration messages are lost.
- Use the `epre` information in the PUSCH/PDSCH logs (PHY layer) to modify the receive gain (`rx_gain` parameter) in order not to saturate the receiver. The maximum value should be smaller than -15 dBFS.
- You must use at least 2 downlink antennas to receive the MIMO 2x2 transmitted data.
- Assuming the signal is strong enough, `ul_timing_offset` can be modified to select the distance of the received UEs (the current PUSCH receiver only tolerates an uplink timing offset between -4 and 4). The timing advance measured for each PUSCH (`ta=` in the PUSCH PHY logs) can be used to adjust `ul_timing_offset`.

5 Configuration reference

5.1 Configuration file syntax

The main configuration file uses a syntax very similar to the Javascript Object Notation (JSON) with few extensions.

1. Supported types:
   - Numbers (64 bit floating point). Notation: `13.4`
   - Complex numbers. Notation: `1.2+3*I`
   - Strings. Notation: `"string"`
   - Booleans. Notation: `true` or `false`.
   - Objects. Notation: `{ field1: `*`value1`*`, field2: `*`value2`*`, .... }`
   - Arrays. Notation: `[ `*`value1`*`, `*`value2`*`, .... ]`
2. The basic operations `+`, `-`, `*` and `/` are supported with numbers and complex numbers. `+` also concatenates strings. The operators `!`, `||`, `&&`, `==`, `!=`, `<`, `<=`, `>=`, `>` are supported too.
3. The numbers `0` and `1` are accepted as synonyms for the boolean values `false` and `true`.
4. {} at top level are optional.
5. `"` for property names are optional, unless the name starts with a number.
6. Properties can be duplicated.
   If properties are duplicated, they will be merged following [JSON merge rules], page 6, with overriding occuring in reading direction (last overrides previous).
   Ex:
   ```
   {
       value: "foo",
       value: "bar",
       sub: {
           value: "foo"
       },
       sub: {
           value: "bar"
       }
   }
   ```
   Will be equivalent to:
   ```
   {
       value: "bar",
       sub: {
           value: "bar"
       }
   }
   ```
7. Files can be included using *include* keyword (must not be quoted) followed by a string (without :) representing the file to include (path is relative to current file) and terminating by a comma.
   Arrays can't be included.
   Merge will be done as for duplicate properties.
   If *file1.cfg* is:
   ```
       value: "foo",
       include "file2.cfg",
       foo: "foo"
   ```

And *file2.cfg* is:

```
    value: "bar",
    foo: "bar"
```

Final config will be:

```
{
    value: "bar",
    foo: "foo"
}
```

8. A C like preprocessor is supported. The following preprocessor commands are available:

   `#define` *var expr*
   : Define a new variable with value *expr*. *expr* must be a valid JSON expression. Note that unlike the standard C preprocessor, *expr* is evaluated by the preprocessor.

   `#undef` *var*
   : Undefine the variable *var*.

   `#include` *expr*
   : Include the file whose filename is the evaluation of the string expression *expr*.

   `#if` *expr*
   : Consider the following text if *expr* is true.

   `#else`
   : Alternative of `#if` block.

   `#elif`
   : Composition of `#else` and `#if`.

   `#endif`
   : End of `#if` block.

   `#ifdef` *var*
   : Shortcut for `#if defined(var)`

   `#ifndef` *var*
   : Shortcut for `#if !defined(var)`

   In the JSON source, every occurrence of a defined preprocessor variable is replaced by its value.

9. Backquote strings: JSON expression can be inserted in backquote delimited strings with the `${expr}` syntax. Example: `‘abc${1+2}d‘` is evaluated as the string `"abc3d"`. Preprocessor variables can be used inside the expression. Backquote strings may span several lines.

5.1.1 JSON merge rules

Merge overriding direction depends on context, i.e source may override destination or the opposite.
JSON merge is recursive for Objects and Arrays.

Example, merging

```
{
    foo: { value: "bar" },
    same: "one",
    one: 1
}
```

with

```
{
    foo: { value: "none", second: true },
    same: "two",
    two: 1
}
```

Will become:

```
{
    foo: { value: "bar", second: true },
    same: "one",
    one: 1
    two: 1
}
```

assuming first object overrides second one.

In case of Array merging, the final array length will be the maximum length of all merged arrays.
For each element of the final array, merge will be done considering defined elements only.
Ex:

```
{
    array: [0, 1, 2, { foo: "bar" } ],
    array: [3, 4],
    array: [5, 6, 7, { bar: "foo" }, 8 ]
}
```

Will be merged to:

```
{
    array: [5, 6, 7, { foo: "bar", bar: "foo" }, 8 ],
}
```

5.2 Global properties

`log_filename`
: String. Set the log filename. If no leading `/`, it is relative to the configuration file path. See [Log file format], page 19.

`log_options`
: String. Set the logging options as a comma separated list of assignments.

- *layer*.level=*verbosity*. For each layer, the log verbosity can be set to `none`, `error`, `info` or `debug`. In debug level, the content of the transmitted data is logged.
- *layer*.max_size=*n*. When dumping data content, at most `n` bytes are shown in hexa. For ASN.1, NAS or Diameter content, show the full content of the message if `n > 0`.
- *layer*.payload=[0|1]. Dump ASN.1, NAS, SGsAP or Diameter payload in hexadecimal.
- *layer*.key=[0|1]. Dump security keys (NAS and RRC layers).
- *layer*.crypto=[0|1]. Dump plain and ciphered data (NAS and PCDP layers).
- phy.signal=[0|1]. Dump binary received signal data of the physical layer to another file (*log_filename*.bin). The currently available data are QAM constellations and channel estimation for PDSCH, PUSCH and SRS. The GUI can be used to display them. Note: the size of the binary signal data is larger than the textual logs, so they should be enabled only when needed.

- bcch=[0|1]. Enable or disable BCCH log. The BCCH is always transmitted, so it gives large logs when enabled.
- time=[sec|short|full]. Display the time as seconds, time only or full date and time (default = time only).
- time.us=[0|1]. Dump time with microseconds precision.
- file=cut. Close current file log and open a new one.
- file.rotate=now. Move and rename to the same directory or to the directory pointed by `file.path` and open a new log file (Headers are kept).
- file.rotate=*size*. Every time log file size reaches *size* bytes, move and rename to the same directory or to the directory pointed by `file.path`, and open a new log file (Headers are kept).
  Size is an integer and can be followed by K, M or G.
- file.rotate=#*count*. Everytime number of logs in log file reaches *count*, move and rename to the same directory or to the directory pointed by `file.path`, and open a new log file (Headers are kept).
  Size is an integer and can be followed by K, M or G.
- file.path=*path*. When log rotation is enabled (`file.rotate` set), rename and move current log to this path instead of initial log path.
- append=[0|1]. (default=0). If 0, truncate the log file when opening it. Otherwise, append to it.

Available layers are: `phy`, `mac`, `rlc`, `pdcp`, `rrc`, `nas`, `ip`

`log_sync` Optional boolean (default = false). If true, logs will be synchronously dumped to file.
Warning, this may lead to performances decrease.

`pcap_filename`
String. Set the filename of the Wireshark compatible IP logs. For each IP packet, specific MAC addresses are used to identify the eNodeB, UE and DRB identity. The MAC address `02:00:00:00:00:00` corresponds to the eNodeB. The MAC address `06:00:00:UEID1:UEID0:DRBID` corresponds to the UE whose 16 bit ID is `UEID1:UEID0` and DRB identity `DRBID`.

`ue_filename`
String. Set the filename of the log of detected UEs. In the log, `#` indicates a comment (used in the first lines to give the information about the cell). The following columns are defined:

`TIME`	Local time (ms resolution) of the connection.
`RAR RNTI`	Temporary RNTI in the Random Access Response message.
`RAR TA`	Timing alignment information in the Random Access Response message. Can be used to estimate the distance to the eNodeB with a 150 meter resolution provided a calibration is done to know the origin.
`PRACH TA`	Timing alignment information computed by lteprobe. Can be used to estimate the distance to the UE with a 150 meter resolution provided a calibration is done to know the origin. Note: if the UE is too far the PRACH may not be received.
`PRACH SNR`	PRACH SNR measurement in dB.

`MMEC/M-TMSI`
: MMEC/M-TMSI information of the connected UE. It is available only if the PRACH and first PUSCH message (msg3) were successfully received.

`rf_driver`
: Object. Parameters of the radio driver. See `trx_sdr.pdf` to have more information.

`rx_gain`
: Float or array of floats. Receive gain in dB. The range is device dependent. With an array of floats a different gain is specified for each channel.

`rx_epre_in_dbfs`
: Optional boolean (default = false). In the logs, the EPRE (Energy Per Resource Element) is displayed in dBm if the RF interface provides its reference receive power and if rx_epre_in_dbfs = false. Otherwise it is displayed in dBFS (Decibels relative to Full Scale).

`rx_epre_offset`
: Optional float (default = 0). Offset in dB applied to all the receive EPRE measurements.

`bandwidth`
: Optional float. Defines the maximum received bandwidth. It can be 20, 15, 10, 5, 3 or 1.4. The RF sample rate is set accordingly
If omitted, `sample_rate` has to be set.

`sample_rate`
: Optional float. Sample rate in MHz. It is normally automatically set depending on the radio head capabilities and selected bandwidth.
It is mandatory if `bandwidth` is not set.

`sample_rate_num`
: Optional integer. Main sample rate used for the LTE signal processing in 1.92 MHz units (hence 3 means 5.76 MHz). It is normally automatically set depending on the radio head capabilities and selected bandwidth. If the resulting rate is different from `sample_rate`, a fractional sample rate interpolator is used to convert the sample rate.

`auto_rb_add`
: Boolean. If true, the MAC Logical Channel Identity (LCID) is used to automatically determine the DRB/SRB type. It cannot be 100% reliable, so it is useful only when the RRC reconfiguration messages configuring the SRB/DRB cannot be received (e.g. if the RRC is encrypted).

`dump_pcch`
: Boolean. If true, the paging data is logged.

`pdsch_max_turbo_its`
: Optional integer (default = 6). Maximum number of iterations for the downlink (PDSCH) turbo decoder.

`pusch_max_turbo_its`
: Optional integer (default = 6). Maximum number of iterations for the uplink (PUSCH) turbo decoder.

`inactivity_timer`
: Optional integer (default = 10000). If no data is received for a given UE during `inactivity_timer` ms, then the corresponding UE context is automatically destroyed. It is necessary in order to reduce the number of concurrent UE contexts.

`cells` Array of objects. Define the cell parameters. Currently only a single cell is supported. The following properties are available in each cell:

`dl_earfcn`
Integer. Downlink EARFCN of the cell.

`ul_earfcn`
Optional integer. Uplink EARFCN. If omitted it is automatically determined from the downlink EARFCN assuming a standard TX-RX gap.

`n_antenna_dl`
Optional. 1, 2, 4 or 8 (default = 1). Number of downlink receive antennas.

`n_antenna_ul`
Optional 1, 2, 4 or 8 (default = 1). Number of uplink receive antennas.

`n_id_cell`
Optional. Range -1 to 503 (default = -1). Select the Physical cell ID. `-1` indicates that the best received cell should be used. The exact Physical cell ID is displayed in the logs.

`ul_timing_offset`
Optional integer. Adjust the time offset between the uplink and the downlink, in 1/1.92 microsecond units. It is useful to modify it in case the received UEs are very far (the current PUSCH receiver only tolerates an uplink timing offset between -4 and 4).

`custom_freq_band`
Optional object or array of objects. Define a non standard LTE or NR frequency band. Standard bands can also be overriden by this option. If the uplink information is not provided, it is assumed to be the same as the downlink (TDD band). Use an array of objects if you want to define more than one custom band.

For LTE bands, the following parameters are available:

`band` Range: 1 to 256.

`dl_earfcn_min`
Range: 0 to 262143.

`dl_earfcn_max`
Range: 0 to 262143.

`dl_freq_min`
Float. Low DL frequency in MHz.

`ul_earfcn_min`
Optional integer. Range: 0 to 262143.

`ul_earfcn_max`
Optional integer. Range: 0 to 262143.

`ul_freq_min`
Optional Float. Low UL frequency in MHz.

`ntn` Optional boolean. True if this is a NTN band.

For NR bands, the following parameters are available:

`band_nr` Range: 1 to 1024. NR band number.

- `dl_freq_min`
  Float. Range: 0 to 65535. Minimum DL frequency in MHz. Use 0 if no DL.
- `dl_freq_max`
  Float. Range: 0 to 65535. Maximum DL frequency in MHz. Use 0 if no DL.
- `ul_freq_min`
  Float. Range: 0 to 65535. Minimum UL frequency in MHz. Use 0 if no UL. If not provided, use the same value as DL (TDD).
- `ul_freq_max`
  Float. Range: 0 to 65535. Maximum UL frequency in MHz. Use 0 if no UL.
- `ssb_scs` Array of integers. List of allowed SSB subcarrier spacing for this band. Allowed values: 15, 30, 120 or 240.
- `f_raster` Enumeration: 100, 15, 15_30, 15_30_100, 60_120, 100_enhanced. Frequency raster in kHz.
- `ssb_case_c`
  Boolean. True if SSB case C is enabled on this band.
- `min_40mhz_bw`
  Boolean. True if the minimum allowed bandwidth on this band is at least 40 MHz. This information is used to select the CoReSet #0 table in standalone mode.
- `delta_gscn`
  Optional enumeration: 1, 3, 7, 16 (default = 1). GSCN step size.
- `ntn` Optional boolean. True if this is a NTN band.

`com_addr` Optional string. Address of the WebSocket server remote API. See [Remote API], page 12.
If set, the WebSocket server for remote API will be enabled and bound to this address.
Default port is 9005.
Setting IP address to [::] will make remote API reachable through all network interfaces.

`com_name` Optional string. Sets server name. UE by default

`com_ssl_certificate`
Optional string. If set, forces SSL for WebSockets. Defines CA certificate filename.

`com_ssl_key`
Optional string. Mandatory if *com_ssl_certificate* is set. Defines CA private key filename.

`com_ssl_peer_verify`
Optional boolean (default is false). If *true*, server will check client certificate.

`com_ssl_ca`
Optional string. Set CA certificate. In case of peer verification with self signed certificate, you should use the client certificate.

`com_log_lock`
Optional boolean (default is false). If *true*, logs configuration can't be changed via `config_set` remote API.

`com_log_us`
: Optional boolean (default is false). If *true*, logs sent by `log_get` remote API response will have a `timestamp_us` parameters instead of `timestamp`

`com_auth`
: Optional object. If set, remote API access will require authentication.
Authentication mechanism is describe in [Remote API Startup], page 14, section.

  `passfile`
  : Optional string. Defines filename where password is stored (plaintext).
  If not set, `password` must be set

  `password`
  : Optional string. Defines password.
  If not set, `passfile` must be set.

  `unsecure`
  : Optional boolean (default false). If set, allow password to be sent plaintext.
  NB: you should set it to true if you access it from a Web Browser (Ex: Amarisoft GUI) without SSL (https) as your Web Browser may prevent secure access to work.

`com_log_count`
: Optional number (Default = 8192). Defines number of logs to keep in memory before dropping them.
Must be between 4096 and 2097152).

6 Remote API

You can access LTEPROBE via a remote API.
Protocol used is WebSocket as defined in RFC 6455 (`https://tools.ietf.org/html/rfc6455`).

Note that Origin header is mandatory for the server to accept connections.
This behavior is determined by the use of `nopoll` library.
Any value will be accepted.

To learn how to use it, you can refer to our the following tutorial (`https://tech-academy.amarisoft.com/RemoteAPI.html`).

6.1 Messages

Messages exchanged between client and LTEPROBE server are in strict JSON format.

Each message is represented by an object. Multiple message can be sent to server using an array of message objects.

Time and delay values are floating number in seconds.

There are 3 types of messages:

- Request

  Message sent by client.
  Common definition:

  `message` String. Represent type of message. This parameter is mandatory and depending on its value, other parameters will apply.

  `message_id`
  Optional any type. If set, response sent by the server to this message will have same message_id. This is used to identify response as WebSocket does not provide such a concept.

  `start_time`
  Optional float. Represent the delay before executing the message.
  If not set, the message is executed when received.

  `absolute_time`
  Optional boolean (default = false). If set, `start_time` is interpreted as absolute.
  You can get current clock of system using `time` member of any response.

  `standalone`
  Optional boolean (default = false). If set, message will survive WebSocket disconnection, else, if socket is disconnected before end of processing, the message will be cancelled.

  `loop_count`
  Optional integer (default = 0, max = 1000000). If set, message will be repeated `loop_count` time(s) after `loop_delay` (From message beginning of event).
  Response will have a `loop_index` to indicate iteration number.

`loop_delay`
: Optional number (min = 0.1, max = 86400). Delay in seconds to repeat message from its `start_time`. Mandatory when `loop_count` is set > 0.

- Response

  Message sent by server after any request message as been processed.
  Common definition:

  `message` String. Same as request.

  `message_id`
  : Optional any type. Same as in request.

  `time` Number representing time in seconds since start of the process.
  Usefull to send command with absolute time.

  `utc` Number representing UTC seconds.

- Events

  Message sent by server on its own initiative.
  Common definition:

  `message` String. Event name.

  `time` Number representing time in seconds.
  Usefull to send command with absolute time.

6.2 Startup

When WebSocket connections is setup, LTEPROBE will send a first message with name set to `com_name` and type set to UE.

If authentication is not set, message will be `ready`:

```
{
    "message": "ready",
    "type": "UE",
    "name": <com_name>,
    "version": <software version>,
    "product": <Amarisoft product name (optional)>
}
```

If authentication is set, message will be `authenticate` :

```
{
    "message": "authenticate",
    "type": "UE",
    "name": <com_name>,
    "challenge": <random challenge>
}
```

To authenticate, the client must answer with a `authenticate` message and a `res` parameter where:

```
res = HMAC-SHA256( "<type>:<password>:<name>", "<challenge>" )
```

`res` is a string and HMAC-SHA256 refers to the standard algorithm (`https://en.wikipedia.org/wiki/HMAC`)

If the authentication succeeds, the response will have a `ready` field set to `true`.

```
{
    "message": "authenticate",
    "message_id": <message id>,
    "ready": true
}
```

If authentication fails, the response will have an `error` field and will provide a new challenge.

```
    {
        "message": "authenticate",
        "message_id": <message id>,
        "error": <error message>,
        "type": "UE",
        "name: <name>,
        "challenge": <new random challenge>
    }
```

If any other message is sent before authentication succeeds, the error `"Authentication not done"` will be sent as a response.

6.3 Errors

If a message produces an error, response will have an error string field representing the error.

6.4 Sample nodejs program

You will find in this documentation a sample program: `ws.js`.
It is located in `doc` subdirectory.
This is a nodejs program that allow to send message to LTEPROBE.
It requires nodejs to be installed:

```
    dnf install nodejs npm
    npm install nodejs-websocket
```

Use relevant package manager instead of NPM depending on your Linux distribution.

Then simply start it with server name and message you want to send:

```
    ./ws.js 127.0.0.1:9005 '{"message": "config_get"}'
```

6.5 Common messages

`config_get`
: Retrieve current config.

  Response definition:

  `type` Always "UE"

  `name` String representing server name.

  `logs` Object representing log configuration.
  With following elements:

  - `layers` Object. Each member of the object represent a log layer configuration:

    - `layer name` Object. The member name represent log layer name and parameters are:

      - `level` See [log_options], page 7,

`max_size` See [log_options], page 7,

`key` See [log_options], page 7,

`crypto` See [log_options], page 7,

`payload` See [log_options], page 7,

`signal` Optional boolean. See [log_options], page 7,

`count` Number. Number of bufferizer logs.

`rotate` Optional number. Max log file size before rotation.

`rotate_count`
Optional number. Max log count before rotation.

`path` Optional string. Log rotation path.

`bcch` Boolean. True if BCCH dump is enabled (eNB only).

`mib` Boolean. True if MIB dump is enabled (eNB only).

`locked` Optional boolean. If `true`, logs configuration can't be changed with `config_set` API.

`config_set`
Change current config.
Each member is optional.
Message definition:

`logs` Optional object. Represent logs configuration. Same structure as config_get (See [config_get logs member], page 15).
All elements are optional.
Layer name can be set to `all` to set same configuration for all layers.
If set and logs are locked, response will have `logs` property set to `locked`.

`log_get` Get logs.
This API has a per connection behavior. This means that the response will depend on previous calls to this API within the same WebSocket connection.
In practice, logs that have been provided in a response won't be part of subsequent request unless connection is reestablished. To keep on receiving logs, client should send a new `log_get` request as soon as the previous response has been received.
If a request is sent before previous request has been replied, previous request will be replied right now without considering specific min/max/timeout conditions.

Message definition:

`min` Optional number (default = 1). Minimum amount of logs to retrieve.
Response won't be sent until this limit is reached (Unless timeout occurs).

`max` Optional number (default = 4096). Maximum logs sent in a response.

`timeout` Optional number (default = 1). If at least 1 log is available and no more logs have been generated for this time, response will be sent.

`allow_empty`
Optional boolean (default = false). If set, response will be sent after timeout, event if no logs are available.

`rnti`	Optional number. If set, send only logs matching rnti.
`ue_id`	Optional number. If set, send only logs with matching ue_id.
`layers`	Optional Object. Each member name represents a log layer and values must be string representing maximum level. See [log_options], page 7. If *layers* is not set, all layers level will be set to *debug*, else it will be set to *none*. Note also the logs is also limited by general log level. See [log_options], page 7.
`short`	Optional boolean (default = false). If set, only first line of logs will be dumped.
`headers`	Optional boolean. If set, send log file headers.
`start_timestamp`	Optional number. Is set, filter logs older than this value in milliseconds.
`end_timestamp`	Optional number. Is set, filter logs more recent than this value in milliseconds.
`max_size`	Optional number (default = 1048576, i.e. 1MB). Maximum size in bytes of the generated JSON message. If the response exceeds this size, the sending of logs will be forced independently from other parameters.

Response definition:

`logs`	Array. List of logs. Each item is a an object with following members:

`data`	Array. Each item is a string representing a line of log.
`timestamp`	Number. Milliseconds since January 1st 1970. Not present if `com_log_us` is set in configuration.
`timestamp_us`	Number. Microseconds since January 1st 1970. Only present if `com_log_us` is set in configuration.
`layer`	String. Log layer.
`level`	String. Log level: *error*, *warn*, *info* or *debug*.
`dir`	Optional string. Log direction: *UL*, *DL*, *FROM* or *TO*.
`ue_id`	Optional number. UE_ID.
`cell`	Optional number (only for PHY layer logs). Cell ID.
`rnti`	Optional number (only for PHY layer logs). RNTI.
`frame`	Optional number (only for PHY layer logs). Frame number (Subframe is decimal part).
`channel`	Optional string (only for PHY layer logs). Channel name.
`src`	String. Server name.
`idx`	Integer. Log index.

`headers`	Optional array. Array of strings.

`discontinuity`
: Optional number. If set, this means some logs have been discarded due to log buffer overflow.

`microseconds`
: Optional boolean. Present and set to true if `com_log_us` is set in configuration file.

`log_set` Add log.
Message definition:

`log`	Optional string. Log message to add. If set, *layer* and *level* are mandatory.
`layer`	String. Layer name. Only mandatory if *log* is set.
`level`	String. Log level: *error*, *warn*, *info* or *debug*. Only mandatory if *log* is set.
`dir`	Optional string. Log direction: *UL*, *DL*, *FROM* or *TO*.
`ue_id`	Optional number. UE_ID.
`flush`	Optional boolean (default = false). If set, flushes fog file.
`rotate`	Optional boolean (default = false). If set, forces log file rotation.
`cut`	Optional boolean (default = false). If set, forces log file reset.

`log_reset`
: Resets logs buffer.

`license` Retrieves license file information.
Response definition:

`products`	String. List of products, separated by commas.
`user`	String. License username.
`validity`	String. License end of validity date.
`id`	Optional string. License ID.
`id_type`	Optional string. License ID type. Can be `host_id` or `dongle_id`
`uid`	Optional string. License unique ID.
`filename`	Optional string. License filename.
`server`	Optional string. License server URL.
`server_id`	Optional string. License server ID.

`quit` Terminates lteprobe.

`help` Provides list of available messages in *messages* array of strings and events to register in *events* array of strings.

`stats` Report statistics for LTEPROBE.
Every time this message is received by server, statistics are reset.
Warning, calling this message from multiple connections simultaneously will modify the statistics sampling time.

Response definition:

`cpu`	Object. Each member name defines a type and its value cpu load in % of one core.
`instance_id`	Number. Constant over process lifetime. Changes on process restart.

7 Log file format

7.1 PHY layer

When a PHY message is dumped (debug level), the format is:

```
     time layer dir ue_id cell rnti frame.subframe channel:short_content
             long_content
```

`time`	Time using the selected format.
`layer`	Layer (`[PHY]` here).
`dir`	UL (uplink) or DL (downlink).
`ue_id`	eNodeB UE identifier (hexadecimal, unique among all cells).
`cell`	Cell index (hexadecimal).
`rnti`	Associated RNTI (hexadecimal) or `-` if none.
`frame.subframe`	Frame number (0-1023) and either subframe number (0-9) for LTE and NB-IoT cells or slot number for NR cells.
`channel`	PHY channel name (e.g. PUSCH, PUCCH, PRACH, SRS, PSS, PBCH, PCFICH, PDSCH, PHICH, PDCCH, EPDCCH, ...).
`short_content`	Single line content.
`long_content`	Hexadecimal dump of the message if `phy.max_size > 0`.

In the uplink messages, `epre` is the relative Energy per Resource Element in dB. The origin 0 dB corresponds to `tx_gain_offset` dBFS.

If UE power control is enabled, `p` is the absolute transmit power in dBm.

If the UE channel simulator is enabled, `p` is the absolute power before the channel simulation is applied. Moreover, if the UE channel simulator is enabled, `epre` is clamped to 0 dB to avoid a potential saturation in the RF interface.

7.2 MAC and RRC layers

When a message is dumped, the format is:

```
     time layer - ue_id message
```

When a PDU is dumped (debug level), the format is:

```
     time layer dir ue_id short_content
             long_content
```

`time`	Time using the selected format
`layer`	Layer (`[MAC]` or `[RRC]` here).
`dir`	UL (uplink) or DL (downlink).
`ue_id`	eNodeB UE identifier (hexadecimal, unique among all cells).
`cell_id`	Primary cell index.
`short_content`	Single line content.

`long_content`
- MAC: hexadecimal dump of the message if `layer.max_size > 0`.
- RRC: full ASN.1 content of the RRC message if `layer.max_size > 0`.

`long_content`
- MAC, RLC, PDCP: hexadecimal dump of the message if `layer.max_size > 0`.
- RRC: full ASN.1 content of the RRC message if `layer.max_size > 0`.

7.3 RLC, PDCP and NAS layers

When a message is dumped, the format is:

```
time layer - ue_id message
```

When a PDU is dumped (debug level), the format is:

```
time layer dir ue_id short_content
        long_content
```

`time`	Time using the selected format
`layer`	Layer (`[RLC]`, `[PDCP]`, or `[NAS]` here).
`dir`	UL (uplink) or DL (downlink).
`ue_id`	eNodeB UE identifier (hexadecimal, unique among all cells).
`short_content`	Single line content. • RLC, PDCP: preceded by the SRB or DRB identifier.
`long_content`	• NAS: full content of the NAS message if `layer.max_size > 0`.

7.4 IP layer

When a IP data PDU is dumped (debug level), the format is:

```
time layer dir short_content
        long_content
```

`time`	Time using the selected format
`layer`	Indicate the layer (`[IP]` here).
`dir`	UL (uplink) or DL (downlink).
`short_content`	Single line content (at least the IP protocol and the source and destination address).
`long_content`	Optional hexadecimal dump of the PDU if `ip.max_size > 0`.

8 Limitations

Currently unsupported features:

– TDD
– PUCCH decoding
– CSI (CQI, RI) decoding
– SRS decoding
– More precise UE location with multi-antenna system
– PUSCH decoding with large UL/DL timing offset
– Carrier aggregation

Known limitations:

– the maximum received bitrate is currently limited.

9 Change history

9.1 Version 2025-03-14

- the crc=KO log is renamed to crc=FAIL

9.2 Version 2024-09-13

- `license` remote API is added
- `com_logs_lock` parameter is renamed to `com_log_lock`. `com_logs_lock` is still supported for backward compatibility
- `com_log_us` parameter is added

9.3 Version 2024-06-14

- OpenSSL library is upgraded to 1.1.1w

9.4 Version 2023-06-10

- `com_logs_lock` parameter added to disable logs configuration change via remote API

9.5 Version 2023-03-17

- `com_addr` parameter now uses [::] address instead of 0.0.0.0 in the delivered configuration file to allow IPv6 connection

9.6 Version 2022-12-16

- `utc` parameter is added to remote API response messages

9.7 Version 2022-06-17

- OpenSSL library is upgraded to 1.1.1n
- `start_timestamp` and `end_timestamp` are added to `log_get` API

10 License