LTE MBMS Gateway

Version: 2018-07-10

Table of Contents

1 Introduction

LTEMBMSGW is a LTE MBMS Gateway. It can easily be used with the Amarisoft LTE eNodeB to build an LTE MBMS test system.

2 Features

- User configurable list of service and multicast components.
- M2AP protocol support.
- Generate one stream per service over the M1 interface (GTP + SYNC protocols).
- Built-in test RTP packet generator.
- Remote API using WebSocket.

3 Requirements

3.1 Hardware requirements

- LTEMBMSGW can run on the same PC as the Amarisoft eNodeB if a simple and compact solution is needed. Otherwise, any reasonnably recent PC with at least one Gigabit Ethernet port is acceptable.

3.2 Software requirements

- A 64 bit Linux distribution. Fedora 26 is the officially supported distribution.
  The following distributions are known as compatible:
  - Fedora 17 to 27
  - Cent OS 7
  - Ubuntu 12 to 16

4 Installation

We assume that the Fedora 20 distribution is running and that the network access thru the Gigabit Ethernet port is correctly configured.

LTEMBMSGW can be run directly from the directory when it was unpacked. No need for explicit installation.

4.1 License key installation

LTEMBMSGW needs a license key file to run. It is the same license file as LTEMME. *It is associated to your PC, so if you replace it or change its hardware configuration you must contact Amarisoft to get a new license key.*

The following steps are needed to get this license file:

- Run LTEMBMSGW:

  ```
  ./ltembmsgw config/mbmsgw.cfg
  ```

  It says that the license key is not present and prints a 16 digit hexadecimal code.
- Send by mail this hexadecimal code to your contact at Amarisoft. You will get back the `ltemme.key` license key file.
- Copy the `ltemme.key` file to the `${HOME}/.amarisoft/` directory (`${HOME}` is the home directory of the `root` user). You can use the shell variable `AMARISOFT_PATH` to change this path.

Once the license key is installed, `ltembmsgw` should start normally.

4.2 Initial testing

- Start the eNodeB with the example MBMS configuration:

  ```
  ./lteenb config/enb-mbms.cfg
  ```
- Start the program as root with the default configuration. This configuration contains several MBMS services. For each service, RTP dummy streams are generated:

  ```
  ./ltembmsgw config/mbmsgw.cfg
  ```
- Verify that the MBMS GTP data is correctly received by the eNodeB with the `mbms` command in the eNodeB monitor. You should see a non zero bitrate for each service and zero packet error.
- Verify that you can receive the corresponding services on your LTE device. The exact setup depend on your device.

When this basic test work, you can customize the eNodeB and MBMS Gateway configuration to use your own generated multicast services.

5 Configuration reference

5.1 Configuration file syntax

The main configuration file uses a syntax very similar to the Javascript Object Notation (JSON) with an extension to support complex numbers and a few mathematical operations. The supported types are:

- 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, .... ]`

The basic operations `+`, `-`, `*` and `/` are supported with numbers and complex numbers.

The numbers `0` and `1` are accepted as synonyms for the boolean values `false` and `true`.

5.2 Properties

`log_filename`
: String. Set the log filename. If no leading `/`, it is relative to the configuration file path. See [Log file format], page 14.

`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, RRC and PCDP layers).
  - time=[sec|short|full]. Display the time as seconds, time only or full date and time (default = time only).
  - file=cut. Close current file log and open a new one.
  - file.rotate=now. Rename current log with timestamp and open new one.
  - file.rotate=*size*. Rename current log every time it reaches *size* bytes open new one. Size is an integer and can be followed by K, M or G.
  - file.path=*path*. When log rotation is enabled, 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: `gtpu`, `m2ap`

`com_addr` Optional string. Address of the WebSocket server remote API. See [Remote API], page 10.
If set, the WebSocket server for remote API will be enabled and bound to this address.
Default port is 9000.
Setting IP address to 0.0.0.0 will make remote API reachable through all network interfaces.

`com_name` Optional string. Sets server name. MBMSGW 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.

`license_server`
Configuration of the Amarisoft license server to use.
Object with following properties:

`server_addr`
String. IP address of the license server.

`name` Optional string. Text to be displayed inside server monitor or remote API.

`tag` Optional string. If set, server will only allow license with same tag.

Example:

```
license_server: {
    server_addr: "192.168.0.20"
}
```

`gtp_bind_addr`
String. Set source IP address (and an optional port) of the GTP-U packets. The default value is `"0.0.0.0:2152"`.

Syntax:

- `"1.2.3.4"` (use default port)
- `"1.2.3.4:5678"` (use explicit port)
- `"2001:db8:0:85a3::ac1f:8001"` (IPv6 address and default port)
- `"[2001:db8:0:85a3::ac1f:8001]:5678"` (IPv6 address and explicit port)

`mce_id` Integer. Range: 0 to 65535. Global MCE Identifier used in M2 signaling.

`enb_time_offset`
Optional integer (default = 0). Offset in ms applied to the MBMSGW International Atomic Time (TAI) so as to generate a time that should match the eNB RF time. The current value can be retrieved by typing the `time` monitor command in eNB or MBMSGW prompt. This is used to synchronize the two components so as to have meaningful timestamps in the SYNC packets (indicating the start of the MCH Scheduling Periods).

Note: the MBMSGW derives the TAI from the UTC OS clock and the `right/UTC` OS time zone.

`time_offset`
: Integer. Default time offset in ms added to all the SYNC timestamps. Can be overridden by the `time_offset` property of each service. It is recommended to set it to at least 2 MCH Scheduling Period to avoid having the eNB dropping SYNC packets due to a timestamp equal to the current MCH Scheduling Period.

  Note: the MBMS Gateway uses the system real time clock as clock source. If synchronous transmission is needed, it should be synchronized to the eNodeB RF time.

`services`
: Array of objects. Contain the definition of each service.

  Property of each service:

  `tmgi`
  : Object. Service identifier (only used for error reporting). Contain the following fields:

    `plmn`
    : String (5 or 6 digits). PLMN identity of the service.

    `service_id`
    : Integer. 24 bit service identity.

  `service_area_id`
  : Integer. Range: 0 to 65535. MBMS service area identifier for this service.

  `session_id`
  : Optional integer. Range: 0 to 255. MBMS session identifier for this service.

  `gtp_addr`
  : String. IP address (and optional port) to which the GTP packets are sent. It is normally a multicast address. Several services can share the same IP address if they have a different TEID.

  `gtp_teid`
  : 32 bit integer. GTP TEID on which the GTP packets are sent.

  `autostart`
  : Optional boolean (default = true). Indicates if service is automatically started when the eNB connects to the MBMS Gateway or if it should be manually launched with the service_start command.

  `scheduling_period`
  : Range: from 8 to 1024. Must be a power of two. Duration of the scheduling period in 10 ms units. Must match the corresponding MCH scheduling period configured in the eNodeB.

  `time_offset`
  : Optional integer. Time offset in ms added to the SYNC timestamps. If not provided, the default time offset is used.

  `forward_mode`
  : Optional boolean (default = false). If set, gateway won't add sync headers and only forward packet to the eNB.

  `tos`
  : Optional integer (default = 0). IPv4 header TOS field (6 bits DSCP + 2 bits ECN).

  `traffic_class`
  : Optional integer (default = 0). IPv6 header traffic class field (6 bits DSCP + 2 bits ECN).

`ttl` Optional integer (default = 64). IP header TTL field.

`components`
Array of object. A service contains several components. Each component is the data coming from a given IP address (usually multicast).

Component properties:

`ip_addr` String. IPv4/v6 address and port from which the MBMS Gateway listen to.

`if_addr` Optional string (default = `"0.0.0.0"`). IP address of the network interface for the multicast join.

`sim` Optional boolean (default = false). If true, RTP packets coming from `ip_addr` are generated using a RTP payload of `rtp_payload_len` bytes and a bitrate of `bitrate`.

`rtp_payload_len`
Optional integer. Only meaningful if sim = true. RTP payload length in bytes (default = 1460).

`bitrate` Optional integer. Only meaningful if sim = true. Bitrate in bit/s of the generated RTP stream. The bitrate includes the size of the IP, UDP and RTP headers.

`area_info_list`
Array of object. Each object defines the parameters of one MBSFN area:

`area_id` Range: 0 to 255. Area identifier.

`non_mbsfn_region_length`
Enumeration: 1, 2. Number of CCH symbols. For 1.4 MHz downlink, only `2` is allowed.

`mcch_config`
Object. MCCH configuration:

`mcch_repetition_period`
Range: 32 to 256, power of two. MCCH repetition period (in 10 ms frames).

`mcch_offset`
Range: 0 to 10. MCCH offset.

`mcch_modification_period`
Enumeration: 512, 1024. (in 10 ms frames).

`mcch_sf_alloc`
Bit string. Length = 6 (1 frame). In FDD, the bits correspond to subframes 1, 2, 3, 6, 7, 8. In TDD, the bits correspond to subframes 3, 4, 7, 8, 9.

`signalling_mcs`
Enumeration: 2, 7, 13, 19. MCS for MCCH and MCHSI transmission. MCCH and MCHSI are critical to decode the MBMS data (MTCH), so their MCS should be lower than the one of the data.

`mbsfn_area_configuration`
Object. MBSFN area configuration. Most of the content of this object is transmitted in the MCCH.

`common_sf_alloc`
: Array of object. Defines the subframes dedicated to this MBSFN area. Each object has the following fields:

  `radio_frame_allocation_period`
  : Range: 1 to 32, power of two. Allocation period (in 10 ms frames).

  `radio_frame_allocation_offset`
  : Range: 0 to 7. offset in the allocation period (in 10 ms frames).

  `subframe_allocation`
  : Bit string. Length = 6 (1 frame) or 24 (4 frames). In FDD, the bits correspond to subframes 1, 2, 3, 6, 7, 8. In TDD, the bits correspond to subframes 3, 4, 7, 8, 9.

`common_sf_alloc_period`
: Range: 4 to 256, power of two. Common subframe allocation period (in 10 ms frames). The PMCH are allocated consecutively during this period.

`pmch_info_list`
: Array of objects. List of PMCH. Each PMCH has the following properties:

  `pmch_config`
  : Object. PMCH physical parameters.

    `sf_alloc_count`
    : Integer >= 1. Number of subframes allocated to this PMCH per common period.

    `data_mcs` Range: 0 to 28. MCS used for the MBMS data (MTCH).

    `data_mcs2`
    : Optional integer. Range: 0 to 27. If provided, `data_mcs` is ignored and an alternate MCS table is used to allow 256QAM MBMS. Note: 256QAM MBMS is an optional release 12 feature, so not all UEs can receive a PMCH using `data_mcs2`.

    `mch_scheduling_period`
    : Range: 4 to 1024, power of two. Scheduling period (in 10 ms frames) for the MCH. MCHSI is transmitted with this periodicity. Must be >= common_sf_alloc_period. For the first PMCH, must be <= mcch_repetition_period. Note: only release 12 UEs support the

value 4, so the effective range to support all UEs is 8 to 1024.

`mbms_session_info_list`
: Array of objects. List of sessions in this PMCH. Each session has the following properties:

  `tmgi`
  : Object. Temporary Mobile Group Identity.

    `plmn`
    : String (5 or 6 digits). PLMN identity.

    `service_id`
    : 24 bit integer. Service identity.

  `logical_channel_identity`
  : Range: 0 to 28. MAC logical channel identity. Must be different for each session in the PMCH. 0 is reserved for the MCCH in the first PMCH.

6 Remote API

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

6.1 Messages

Messages exchanged between client and LTEMBMSGW 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.

All messages have at least following definition:

`message` String. Represent type of message. This parameter is mandatory and depending on its value, other parameters will apply.
If message is a response from server, response message will have same message member.

`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 double. Represent the delay before executing the message.
If not set, the message is executed when received.
Note that some command (*log_get*, *log_reset*, *config_get*, *config_set*, *stats*) can't be executed in future.

`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 `config_get` command.

6.2 Errors

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

6.3 Sample nodejs program

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

```
yum install nodejs npm
npm install nodejs-websocket
```

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

```
./ws.js 127.0.0.1:9000 '{"message": "config_get"}'
```

6.4 Common messages

`config_get`
: Retrieve current config.

  Response definition:

  `type` Always "MBMSGW"

  `name` String representing server name.

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

  `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 5,

  `max_size` See [log_options], page 5,

  `count` Number. Number of bufferizer logs.

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

  `logs` Object. Represent logs configuration. Same structure as config_get (See [config_get logs member], page 12).
  All elements are optional.

`log_get` Get logs.
: 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` Optionnal 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 geenrated for this time, response will be sent.

  `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 5.
  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 5.

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. Number of seconds since start of session or start of day.
- `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.

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

Note that only one request can be sent by client.
If a request is sent before previous one has returned, previous one will be sent without matchine min/max/timeout conditions.

`log_reset` Resets logs buffer.

`quit` Terminates ltembmsgw.

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

`stats` Provides statistics.
Every time this message is received by server, statistics are reset.

Response definition:

`time` Time in seconds since LTEMBMSGW starting.

`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.

6.5 LTE messages

`service_start`
: Start a service.
Message definition:

  `service_id`
  : Integer. Identifier of service to start.

`service_stop`
: Stop a service.
Message definition:

  `service_id`
  : Integer. Identifier of service to stop.

7 Log file format

7.1 M2AP and GTP-U layers

When a message is dumped, the format is:

```
time layer - message
```

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

```
time layer dir ip_address short_content
        long_content
```

`time` Time using the selected format.

`layer` Indicate the layer (`[M2AP]` or `[GTPU]` here).

`dir` Direction: `TO` or `FROM`.

`ip_address`
source or destination IP address, depending on the `dir` field.

`short_content`
Single line content.

`long_content`

- M2AP: full ASN.1 content of the M2AP message if `layer.max_size > 0`.
- GTPU: hexadecimal dump of the message if `layer.max_size > 0`.

8 License

Abbreviations

MBMS	Multimedia Broadcast Multicast Service
SYNC	MBMS synchronisation protocol
TMGI	Temporary Mobile Group Identity