剖析 LoRaWAN Gateway 核心代码

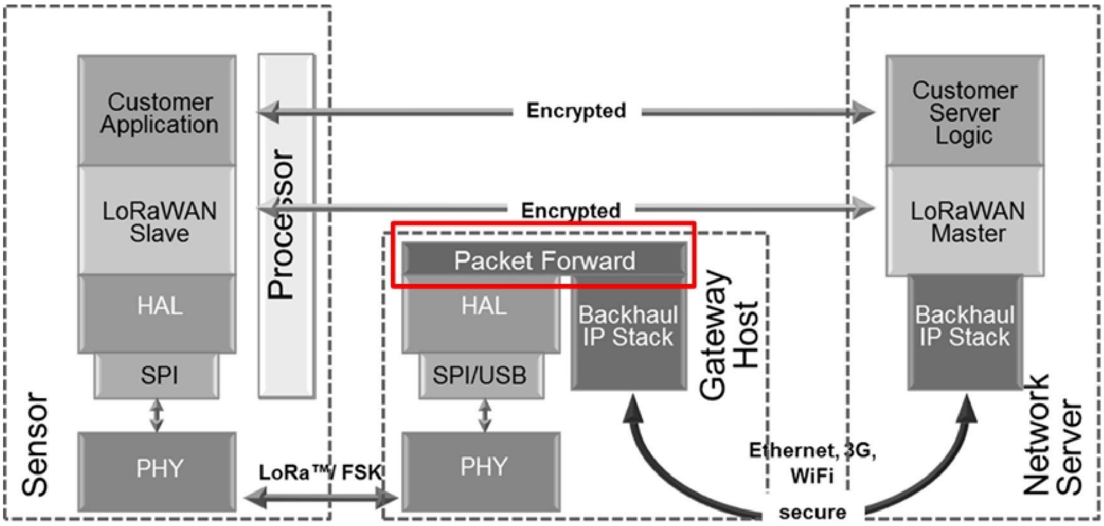
引言

Packet forwarder 是 LoRaWAN Gateway 的核心代码，超过 6000 行。除去 JSON parse 库，代码约 4000 行。

1、 体系结构和功能

如下图所示，packet forwarder 运行在 gateway 的 host 上面，主要实现的功能是转发集中器和服务器之间的报文以及发送 beacon 以同步节点。

摘自 project readme：forwards RF packets receive by the concentrator to a server through a IP/UDP link, and emits RF packets that are sent by the server. It can also emit a network-wide GPS-synchronous beacon signal used for coordinating all nodes of the network。



1.1 功能

1、 为 gateway 的核心控制单元，启停 concentrator 和 GPS
2、 转发（非透明）concentrator 和 server 之间的上下行应用报文
3、 管理 GPS，接收 GPS 时间和位置信息，作为同步/协调的基础
4、 组织 beacon 报文，驱动 concentrator 下发 beacon 以同步各节点

5、 收集上下行报文发送数量、成功率等网络状态信息，并周期性汇报给 server

2、如何使用

2.1 目录结构

├──cfg

| ├ ─ ─ global_conf.json.PCB_E286.EU868[PCB_E286.EU868/ US902].basic[beacon/ gps]

├──global_conf.json

├──local_conf.json

├──Makefile

├──readme.md

├──inc

| ├── base64.h

| ├──jitqueue.h

| ├──parson.h

| ├──time_h

| ├──timersync.h

| └──trace.h

├──src

| ├── base64.c ----------------------------Base64 编解码库

| ├──jitqueue.c ---------------------------Just-in-Time 下行发送队列管理

| ├──lora_pkt_fwd.c

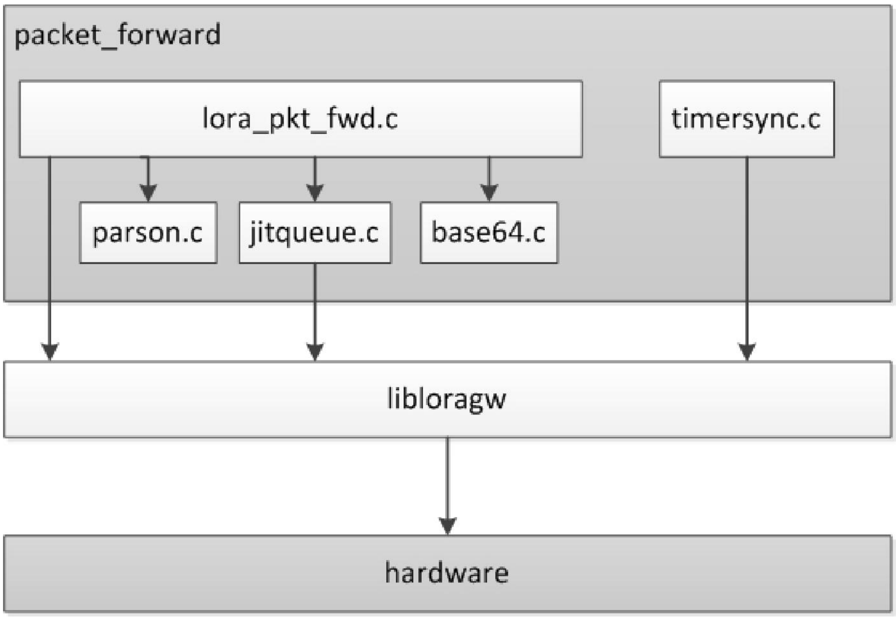
| ├──parson.c -----------------------------JSON 解析

| └──timersync.c -------------------------系统时间、集中器时间、GPS 时间同步

└── update_gwid.sh

官方示例代码中，由工程 lora_gateway 提供库 libloragw，工程 packet_forwarder 中的 lora_pkt_fwd.c 提供 main()。packet_forwarder 查询/控制集中器、GPS 等硬件，是通过 libloragw。

另外，packet_forwarder 使用 Parson library（工程内部文件 parse.c）解析配置文件。



3、 调度

3.1 线程和功能简述

文件 lora_pkt_fwd.c 里 main()创建如下线程：

```
/* threads */
void thread_up(void);
void thread_down(void);
void thread_gps(void);
void thread_valid(void);
void thread_jit(void);
void thread_timersync(void);
```

包括主线程，一共 7 个线程。一句话说明各个线程的功能

Thread	Description	LoRaWAN protocol related
main	周期统计运行状态信息	上下行 cnt
thread_up	周期性从concentrator取mote上行应用报文并组帧成PUSH_DATA消息发送给 server	上行
thread_down	维持 GW 和 serv 之间链路，接收 serv 下行应用报文、组织 beacon 并加入 JiT 队列	下行，beacon
thread_jit	周期性从 JiT 队列取报文并发送到 concentrator（通过 concentrator 发送到 mote）	时序，CLASS A/B/C
thread_gps	持续从 GPS 同步时间和位置信息	
thread_valid	每秒检查一次 GPS 是否已经停止更新时间，并据此决定是否使用 GPS 时间作为基准时间	
thread_timersync	周期性检查 GW host 和 concentrator 时间差	

3.2 线程功能和调度具体描述

1、 主线程，每隔 stat_interval（default 30s）运行一次统计信息打印和上报
2、 thread_up：
   a) 每 FETCH_SLEEP_MS（default 10ms）从集中器取报文（包括主线程的 status report 消息）。
   b) 取到后，选择第一个，组织成 PUSH_DATA JSON 消息通过 sock_up 发送；或者发送 status_report（若有）
   c) 在 sock_up 上以超时方式接收 ACK（无等待重试一次）
3、 thread_down：先发一个 PULL_DATA（12 字节），在接下来的 keepalive_time 内：
   a) 尝试从 server 那里接收消息
   b) 在 JiT 队列中加入 beacon，使得队列 beacon 总数达到 JIT_NUM_BEACON_IN_QUEUE
   c) 在 a 步未收到消息，重复 a,b 步骤，如果收到消息，进行消息处理：

i. 消息格式不正确，回到 a
ii. 为 PULL_ACK 消息，如果不是重复的 ACK，meas_dw_ack_rcv++，否则忽略
iii. PULL_RESP 消息，解析该消息并加入 JiT 队列，发送 ACK 到 server。

4、 thread_gps：循环从 GPS 串口读取数据，并进行匹配：
a) 发现 LGW_GPS_UBX_SYNC_CHAR，调用 lgw_parse_ubx 进行解析，正确解析出时间（解析结果为 UBX_NAV_TIMEGPS），读取 GPS 时间进行系统基准时间同步。读取到的相关信息是：
i. 年月日分秒，考虑时区之后为：UTC（公元纪年秒数）
ii. gps_week （GPS epoch 周数）和 gps_iTOW. gps_iFOW，转换为：gps_time（GPS 纪年秒数）
iii. 根据 UTC、gps_time 和集中器的 cnt 最终更新为 *time_reference_gps*
b) 发现 LGW_GPS_NMEA_SYNC_CHAR，调用 lgw_parse_nmea 进行解析，正确解析出位置信息（解析结果为 NMEA_RMC），调用 gps_process_coords

5、 thread_valid 每秒一次检查 GPS 是否已经停止更新时间，并据此决定是否使用 GPS 时间作为基准时间

6、 thread_jit 每 10ms 从 JiT 队列检查当前是否有报文需要发送到集中器,若有，检查集中器当前是否可发送，如是则取出来并发送，否则等待集中器就绪。

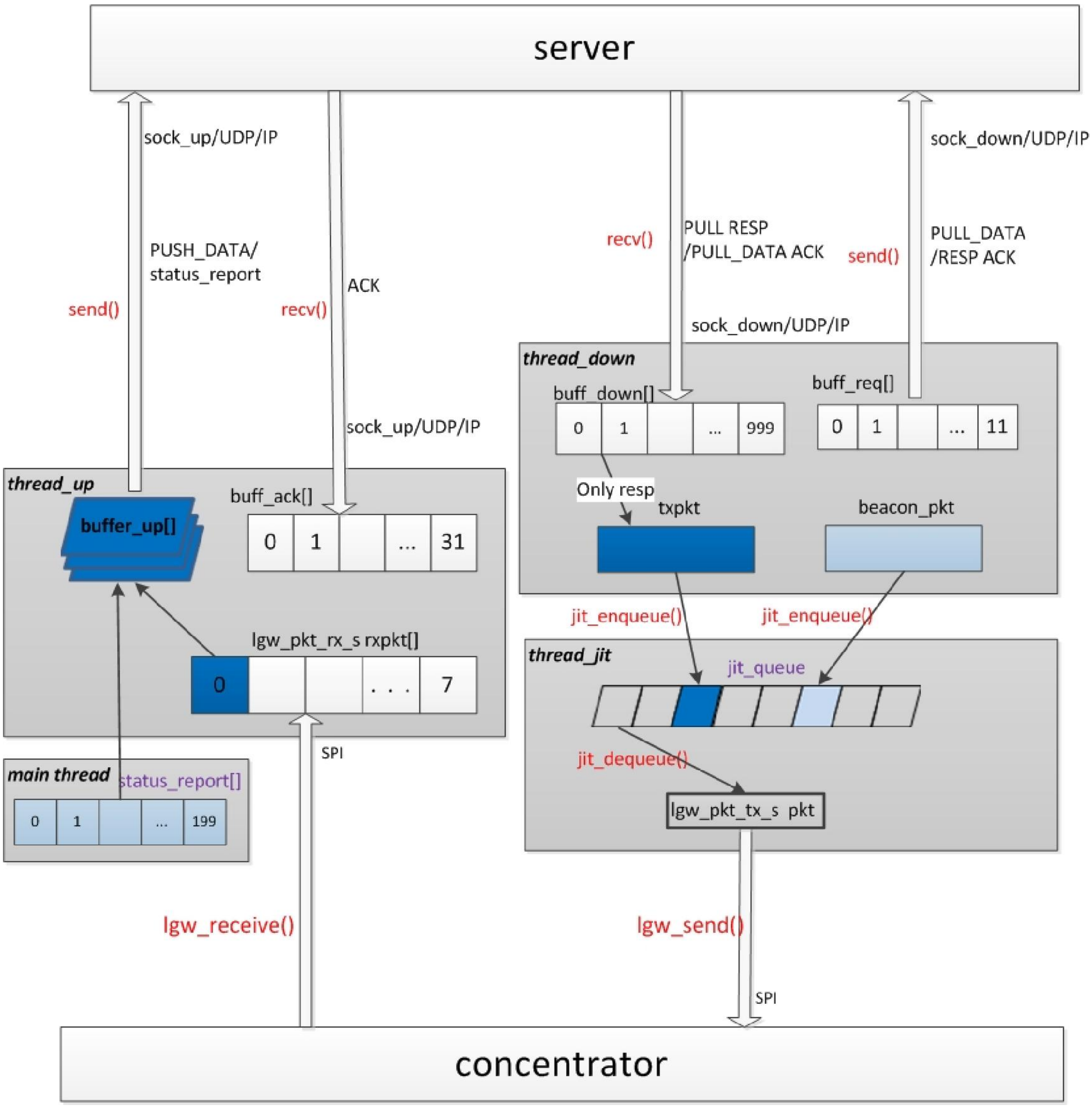
7、 thread_timersync 每 60s（时间精度为 1ms，设晶振最大误差为 20ppm，频率为 1MHz，这意味着大约每 50s 产生 1ms 漂移，这里选择让线程间隔 60s）检查一次系统时钟和 GPS 时钟差。

注意：集中器不具有时钟芯片，其计时是使用计数器的方式，按照 1MHz 的频率进行计数累加，那么计数值=N 也就意味着开始计数以来经历了 N/10^6 秒（取整+小数）。

注意：代码里没有考虑集中器计数溢出：uint32_t sx1301_timecount 计数即 2^32 秒约 71.58h。

a) 读取系统 UNIX 时间
b) 读取集中器 cnt，并转化为集中器时间
c) 计算两者差值并保存到全局变量 *offset_unix_concent*

4、 数据流



说明：

lgw_receive() 代表操作接口

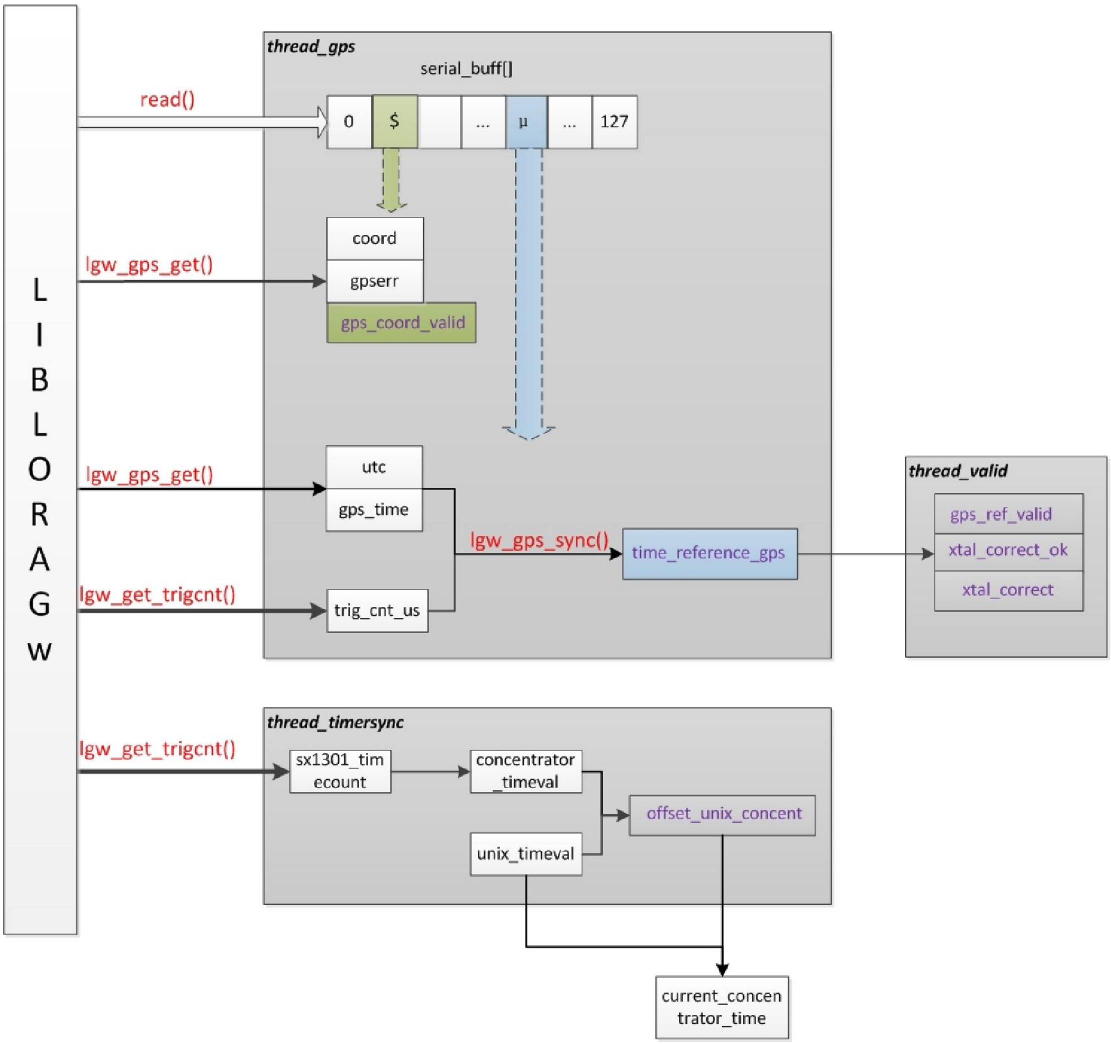
箭头附近 SPI 代表通道类型

status_report[] 代表全局数组

buffer_up[] 代表局部数组

thread_up 代表线程

PUSH_DATA 代表数据类型



说明：

1、serial_buff[]不同的首字符'$', 'µ'，代表不同格式 GPS 消息，导入不同的处理

5、接口

5.1 libloragw 库调用

调用者	调用函数	描述
main()	lgw_version_info()	Lora concentrator HAL library version
	lgw_gps_enable()	Configure a GPS module
	lgw_board_setconf()	Configure the gateway board
	lgw_lbt_setconf()	Configure the gateway lbt function
	lgw_rxrf_setconf()	Configure an RF chain (must configure before start)
	lgw_rxif_setconf()	Configure an IF chain + modem (must configure before start)
	lgw_txgain_setconf()	Configure the Tx gain LUT
	lgw_start()	Connect to the LoRa concentrator, reset it and configure it according to previously set parameters
	lgw_gps_disable()	Restore GPS serial configuration and close serial device
	lgw_stop()	Stop the LoRa concentrator and disconnect it
main thread thread_gps thread_timersync	lgw_get_trigcnt()	Get concentrator count
thread_up	lgw_receive()	Receive pkts from concentrator
	lgw_cnt2utc()	convert packet timestamp to UTC

		absolute time
	lgw_cnt2gps()	convert packet timestamp to GPS absolute time
thread_down	lgw_gps2cnt	convert GPS time to concentrator time
thread_jit	lgw_time_on_air()	Calculate transmission time(in ms)
	lgw_status()	Give the the status of different part of the LoRa concentrator
	lgw_send()	Send a pkt to concentrator
thread_gps	lgw_gps_get()	get GPS time/loc info
	lgw_parse_ubx	Parse Ublox proprietary message
	lgw_gps_sync	update time reference with the new GPS time & timestamp
	lgw_parse_nmea	Parse NMEA message
thread_timersync	lgw_reg_w	Write to a register addressed by name

5.2 PKT_FWD 内部模块之间的接口

调用者	函数	描述
parse_SX1301_configuration() parse_gateway_configuration()	json_parse_file_with_comments() json_object_get_object() json_object_get_value() json_value_get_type() json_object_dotget_value()	JSON parse library
thread_jit	jit_peek()	JiT
	jit_dequeue()	JiT
thread_down	jit_enqueue()	JiT

5.3 网络接口

Sock name	Protocol	Description
sock_up	UDP/IP	Upstream: PUSH_DATA/status_report; Downstream: ACK
sock_down	UDP/IP	Upstream: PULL_DATA/PULL_DESP ACK; Downstream: PULL_DATA ACK/PULL_DESP

6、 数据结构

LoRaWAN 网关与服务器接口协议（JSON 消息）

<table>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>N</th><th>M</th><th>K</th></tr>
<tr><td rowspan="8">0x02</td><td rowspan="8" colspan="2">token</td><td>ID</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>0 - PUSH_DATA</td><td colspan="8">GW EUI=(net_mac_h,net_mac_l)</td><td>JSON</td><td></td><td></td></tr>
<tr><td>1 - PUSH_ACK</td><td colspan="11"></td></tr>
<tr><td>2 - PULL_DATA</td><td colspan="8">GW EUI</td><td colspan="3"></td></tr>
<tr><td>3 - PULL_RESP</td><td colspan="10">JSON</td><td></td></tr>
<tr><td>4 - PULL_ACK</td><td colspan="8">GW EUI</td><td colspan="3"></td></tr>
<tr><td rowspan="2">5 - TX_ACK</td><td>\0</td><td colspan="10"></td></tr>
<tr><td colspan="11">JSON error</td></tr>
</table>

status_report

消息名称	status_report
临时保存者	static char status_report[STATUS_SIZE]，各线程可见
发送者	main thread
发送方向	↑
通道	sock_up
间隔（秒）	stat_interval
内容	Gateway 运行时产生的统计信息。格式为 JSON 字符串 "“stat”:{ “time”:”%s”, // stat_timestamp ”lati”:%.5f, // cp_gps_coord.lat ”long”:%.5f,// cp_gps_coord.lon ”alti”:%i, // cp_gps_coord.alt ”rxnb”:%u, // cp_nb_rx_rcv ”rxok”:%u, // cp_nb_rx_ok ”rxfw”:%u, // cp_up_pkt_fwd ”ackr”:%.1f,//100.0 * up_ack_ratio ”dwnb”:%u,// cp_dw_dgram_rcv ”txnb”:%u // cp_nb_tx_ok }"
补充说明	其中 lati，long，alti 域只有在 GPS 有效的时候提供，否则不提供

PUSH_DATA

消息名称	PUSH _DATA
临时保存者	uint8_t buff_up[TX_BUFF_SIZE]，仅 thread_up 可见
发送者	thread_up
发送方向	↑
通道	sock_up
间隔（秒）	不具备周期性，取集中器间隔时间为 FETCH_SLEEP_MS（ms）

内容	来自 concentrator 的上行报文（lgw_pkt_rx_s 结构体，含 payload 和元数据），转换格式后变成 PUSH_DATA/UDP/IP 报文：
	buff[] = {
	[0] = PROTOCOL_VERSION
	[1] = token_h // 随机数，每次变化
	[2] = token_l // 随机数
	[3] = PKT_PUSH_DATA
	[4..7] = net_mac_h
	[8..11] = net_mac_l
	[12…] = JSON
	"“rxpk”:{
	“tmst”:”%u, // timestamp
	"time":%04i-%02i-%02iT%02i:%02i:%02i.%06liZ,
	"tmms":%llu,
	"chan":%1u,
	"rfch":%1u,
	"freq":%.6lf
	"stat":_
	……
	"data": payload

	}"
补充说明	

PUSH_DATA ACK

消息名称	PUSH _DATA ACK
临 时 保 存 者	uint8_t buff_ack[32]，仅 thread_up 可见
发送者	Server
发送方向	↓
通道	sock_up
间隔（秒）	不具备周期性，取集中器间隔时间为 FETCH_SLEEP_MS（ms）

内容	buff[] = {
	[0] = PROTOCOL_VERSION
	[1] = token_h // 随机数，每次变化
	[2] = token_l // 随机数
	[3] = PKT_PUSH_ACK
	[4..]

补充说明	网关忽略该消息

PULL_DATA

消息名称	PULL_DATA
发送者	thread_down
发送方向	↑
通道	sock_down
间隔（秒）	keepalive_time

内容	Gateway 和 server 之间的维持下行报文通道的“心跳包”
	buff[12] = {
	[0] = PROTOCOL_VERSION
	[1] = token_h // 随机数，每次变化
	[2] = token_l // 随机数
	[3] = PKT_PULL_DATA
	[4..7] = net_mac_h
	[8..11] = net_mac_l

补充说明	对同一个 gateway 而言，PULL_DATA 消息每次不一样的是 token，其余域一样

PULL_DATA ACK

消息名称	PULL_DATA ACK
发送者	Server
发送方向	↓
通道	sock_down
间隔（秒）	取决于 sever

内容	Gateway 和 server 之间的维持下行报文通道的“心跳确认包”
	buff[12] = {
	[0] = PROTOCOL_VERSION
	[1] = token_h // 随机数，每次变化
	[2] = token_l // 随机数
	[3] = PKT_PULL_ACK
	[4..7] = net_mac_h
	[8..11] = net_mac_l

补充说明	除了[3]外，其余域和对应的 PULL_DATA 一样

PULL_RESP

消息名称	PULL _RESP
临 时 保 存 者	uint8_t buff_down[1000]，仅 thread_down 可见
发送者	Serv
发送方向	↓
通道	sock_down
间隔（秒）	不具备周期性，取决于 server

内容	来自 server 的下行报文：
	buff[] = {
	[0] = PROTOCOL_VERSION
	[1] = token_h // 随机数，每次变化
	[2] = token_l // 随机数
	[3] = PKT_PULL_RESP
	[4…] = JSON
	"“txpk”:{
	"imme", // mandatory，为 true 表示 CLASS C
	"tmst", // CLASS A
	"tmms", // CLASS B
	"ncrc", // optional
	"freq", // mandatory
	"rfch", // mandatory
	"powe", // optional
	"modu", // mandatory
	---- LORA MODU ----
	"datr", // mandatory
	"codr", // optional
	"ipol", // optional

	"prea", // optional *---- FSK MODU ----* "datr", // mandatory "fdev", // mandatory "prea", // optional *--- END MODU ----* "size", // mandatory "data", // mandatory }"
补充说明	PULL_DESP 没有 net_mac 域。 JSON 中有些域是可选的，有些是必选的。 对 JSON 各域的顺序不做要求。

PULL_RESP ACK

消息名称	PULL_RESP ACK
发送者	thread_down
发送方向	↑
通道	sock_down
间隔（秒）	-
内容	buff[64] = { [0] = PROTOCOL_VERSION [1] = token_h [2] = token_l [3] = PKT_TX_ACK [4..7] = net_mac_h [8..11] = net_mac_l [12] ='\0' // 没有错误时

	[12…] = { // 有错误时
	"txpk_ack":{
	" error": // 下面中的一个
	"COLLISION_PACKET"
	"TOO_LATE"
	"TOO_EARLY"
	"COLLISION_BEACON"
	"TX_FREQ"
	"TX_POWER"
	"GPS_UNLOCKED"
	"UNKNOWN"
	}
	}

补充说明	

lgw_pkt_rx_s

Structure containing the metadata of a packet that was received and a pointer to the payload

这是 concentrator 和 host 约定的上行报文和元数据的结构体

```
struct lgw_pkt_rx_s {
	uint32_t	freq_hz;		/*!> central frequency of the IF chain */
	uint8_t		if_chain;		/*!> by which IF chain was packet received */
	uint8_t		status;			/*!> status of the received packet */
	uint32_t	count_us;		/*!> internal con- cnt for timestamping, 1 ms resolution */
	uint8_t		rf_chain;		/*!> through which RF chain the packet was received */
	uint8_t		modulation;		/*!> modulation used by the packet */
	uint8_t		bandwidth;		/*!> modulation bandwidth (LoRa only) */
```

```
    uint32_t     datarate;         /*!> RX datarate of the packet (SF for LoRa)
*/
    uint8_t      coderate;         /*!> error-correcting code of the packet
(LoRa only) */
    float        rssi;             /*!> average packet RSSI in dB */
    float        snr;              /*!> average packet SNR, in dB (LoRa only)
*/
    float        snr_min;          /*!> minimum packet SNR, in dB (LoRa only)
*/
    float        snr_max;          /*!> maximum packet SNR, in dB (LoRa
only) */
    uint16_t     crc;              /*!> CRC that was received in the payload */
    uint16_t     size;             /*!> payload size in bytes */
    uint8_t      payload[256];     /*!> buffer containing the payload */
};
```

lgw_pkt_tx_s

这是 concentrator 和 host 约定的下行报文和元数据的结构体

Structure containing the configuration of a packet to send and a pointer to the payload

```
struct lgw_pkt_tx_s {
    uint32_t     freq_hz;          /*!> center frequency of TX */
    uint8_t      tx_mode;          /*!> select on what event/time the TX is
triggered */
    uint32_t     count_us;         /*!> timestamp or delay in microseconds for
TX trigger */
    uint8_t      rf_chain;         /*!> through which RF chain will the packet
be sent */
    int8_t       rf_power;         /*!> TX power, in dBm */
```

```
	uint8_t		modulation;		/*!> modulation to use for the packet */
	uint8_t		bandwidth;		/*!> modulation bandwidth (LoRa only) */
	uint32_t	datarate;		/*!> TX datarate (baudrate for FSK, SF for
LoRa) */
	uint8_t		coderate;		/*!> error-correcting code of the packet
(LoRa only) */
	bool		invert_pol;		/* invert signal polarity, for orthogonal
downlinks(LoRa only)*/
	uint8_t		f_dev;			/*!> frequency deviation, in kHz (FSK only)
*/
	uint16_t	preamble;		/*!> set the preamble length, 0 for default */
	bool		no_crc;			/*!> if true, do not send a CRC in the
packet */
	bool		no_header;		/*if true,enable implicit header mode(LoRa),
fixed len (FSK) */
	uint16_t	size;			/*!> payload size in bytes */
	uint8_t		payload[256];	/*!> buffer containing the payload */
}
```

jit_queue_s & jit_node_s

```
struct jit_queue_s {
	uint8_t num_pkt;		/* Total number of packets in the queue (downlinks,
beacons...) */
	uint8_t num_beacon;	/* Number of beacons in the queue */
	struct jit_node_s nodes[JIT_QUEUE_MAX];		/* Nodes/packets array in
the queue */
};

struct jit_node_s {
```

```
    /* API fields */
    struct lgw_pkt_tx_s pkt;            /* TX packet */
    enum jit_pkt_type_e pkt_type;     /* Packet type: Downlink, Beacon... */
    /* Internal fields */
    uint32_t pre_delay; /* Tb   before packet timestamp to be reserved */
    uint32_t post_delay;/*Ta   after packet timestamp to be reserved (time on
air) */
};
```

全局变量

变量	修改者	操作	描述
meas_nb_rx_rcv	thread_up	++	Got a packet from concentrator
meas_nb_rx_ok	thread_up	++	STAT_CRC_OK
meas_nb_rx_bad	thread_up	++	STAT_CRC_BAD
meas_nb_rx_nocrc	thread_up	++	STAT_NO_CRC
meas_up_pkt_fwd	thread_up	++	if packet from concentrator needs to fwd
meas_up_payload_byte	thread_up	+payload size	if packet from concentrator needs to fwd
meas_up_dgram_sent	thread_up	++	datagram sent via sock_up
meas_up_network_byte	thread_up	+ JSON size	datagram sent via sock_up
meas_up_ack_rcv	thread_up	++	When ack received
meas_dw_ack_rcv	thread_down	++	if the datagram is an ACK

meas_dw_dgram_rcv	thread_down	++	count only the datagram is a PULL_RESP, with no JSON errors
meas_dw_network_byte	thread_down	+msg size	count only the datagram is a PULL_RESP, with no JSON errors
meas_dw_payload_byte	thread_down	+payload size	count only the datagram is a PULL_RESP, with no JSON errors
meas_nb_tx_requested	thread_down		insert packet to be sent into JIT queue
meas_nb_beacon_sent	thread_jit	++	
meas_nb_tx_fail	thread_jit	++	fail to send packet to concentrator
meas_nb_tx_ok	thread_jit	++	send packet to concentrator
gps_ref_valid	thread_valid	true/false	GPS时钟是否有效。系统和 GPS 时间差是否超出[0, GPS_REF_MAX_AGE]范围，maximum admitted delay in seconds of GPS loss before considering latest

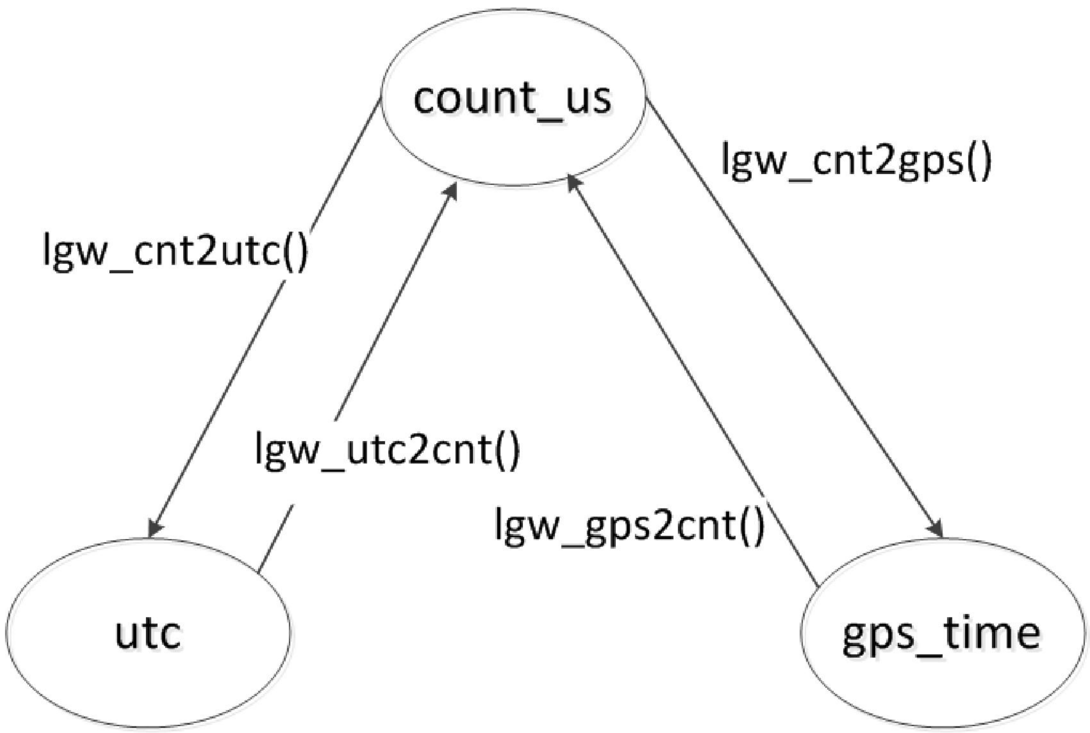
			GPS sync unusable
xtal_correct_ok	thread_valid	true/false	取决于 gps_ref_valid
xtal_correct	thread_valid		
gps_coord_valid	thread_gps	true/false	根据 lgw_gps_get 的结果更新
meas_gps_coord	thread_gps	coord	lgw_gps_get 成功
meas_gps_err	thread_gps	gpserr	lgw_gps_get 成功
meas_nb_beacon_sent	thread_jit	++	
meas_nb_tx_fail	thread_jit	++	
meas_nb_tx_ok	thread_jit	++	
offset_unix_concent	thread_timersync	= unix_timeval, -concentrator_timeval	UNIX 时间和集中器时间差值
time_reference_gps	thread_timersync		哪些使用：Thread_up 中"tmms"; beacon_pkt、txpkt; thread_valid 中决定了 gps_ref_valid 的值
gps_fake_enable		true/false	仅仅和 coords 有关，和 sync 无关，也就是即使该值为 true，没有 GPS 时仍然不能发送下行报文。

7、 重点：时间/计时相关

7.1 各组件计时

1、 GW host 运行 Linux，提供系统 UNIX 时间。
2、 Concentrator 有 1MHz 晶振，20ppm 误差，计时方式为 ticks，每 1μs 集中器计数器 count_us 自加一次（约 3 天溢出回绕）。计数器可以转换为秒数，也就是集中器时间
3、 GPS 提供精确时间 gps_time，为 GPS 纪年时间。
4、 HAL 提供各时间转换函数：lgw_cnt2utc、lgw_utc2cnt（pkt_fwd 未使用该 API）、lgw_cnt2gps、lgw_gps2cnt。其参数为：(ref, src, dst)。

其中的 ref 就是 time_reference_gps。



7.2 时间使用场合

Data structure	UTC:source	count_us:source	gps_time:source
PUSH_DATA	“time”: lgw_cnt2utc(count_us)	“tmst”: count_us	“tmms”: lgw_cnt2gps(count_us)
PUSH_ACK	-	-	-
PULL_DATA	-	-	-
PULL_RESP	-	“tmst”-CLASS A: server	“tmms”-CLASS B: server
PULL_ACK	-	-	-
TX_ACK	-	-	-
beacon		beacon.count_us: lgw_gps2cnt(time_reference_gps.gps)	
status_report	“time”: time()		

time_reference_gps

计算

该变量的类型为结构体：

```
struct tref {
	time_t		systime;	/*!> system time when solution was calculated */
	uint32_t	count_us;	/*!> reference concentrator internal timestamp */
	struct timespec utc;	/*!> reference UTC time (from GPS/NMEA) */
	struct timespec gps;	/*!> reference GPS time (since 01.Jan.1980) */
	double		xtal_err;	/*!> raw clock error (eg. <1 'slow' XTAL) */
```

```
};
```

计算：lgw_gps_sync(&time_reference_gps, trig_tstamp, utc, gps_time)

其中 各域分别为（简化理解）：

```
.systime = 当前系统时间   // time()
. count_us = trig_tstamp        // concentrator count
.utc = utc                 // GPS 提供的 utc
.gps = gps_time            // GPS 提供的 GPS 时间
. xtal_err = cnt_diff/utc_diff //
```

哪些地方使用 time_reference_gps？

1、 PUSH_DATA 中的"tmms"：uint64_t pkt_gps_time_ms; GPS time in milliseconds since 06.Jan.1980。这个时间的来源是：time_reference_gps 和 concentrator count

2、 Beacon 使用的 count_us 则完全由 time_reference_gps 提供。

3、 PULL_DESP 转化来的下行报文中的 count_us 由 JSON 中的"tmms"和 time_reference_gps 共同提供。

4、 当前系统 UNIX 时间和 time_reference_gps.systime 的差值间接决定了：gps_ref_valid、xtal_correct_ok、xtal_correct

7.3 产生问题

1、 系统一直在运行，但 GPS 可能不能使用从而不更新 GPS 时间，也就是 time_reference_gps 和系统时间差超范围。发生什么事情？

   a) PUSH_DATA 不带有"tmms"

   b) 不再发送 CLASS B 下行报文到 concentrator！（GPS 不能开启的时候也不能进行此操作）

8、难点

Function	Do what & how
jit_enqueue()	Pkt enqueue: 1. Compute packet pre/post delays depending on packet's type. An immediate downlink becomes a timestamped downlink "ASAP", Set the packet count_us to the first available slot 2. Check criteria_1: is it already too late to send this packet ? The packet should arrive at least at (tmst - TX_START_DELAY) to be programmed into concentrator Note: - Also add some margin, to be checked how much is needed, if needed - Valid for both Downlinks and Beacon packets Warning: unsigned arithmetic (handle roll-over) t_packet < t_current + TX_START_DELAY + MARGIN 3. Check criteria_2: Does packet timestamp seem plausible compared to current time We do not expect the server to program a downlink too early compared to current time Class A: downlink has to be sent in a 1s or 2s time window after RX Class B: downlink has to occur in a 128s time window Class C: no check needed, departure time has been calculated previously So let's define a safe delay above which we can say that the packet is out of bound: TX_MAX_ADVANCE_DELAY

	Note: - Valid for Downlinks only, not for Beacon packets Warning: unsigned arithmetic (handle roll-over) t_packet > t_current + TX_MAX_ADVANCE_DELAY 4. Check criteria_3: does this new packet overlap with a packet already enqueued ? Note: - need to take into account packet's pre_delay and post_delay of each packet - Valid for both Downlinks and beacon packets - Beacon guard can be ignored if we try to queue a Class A downlink 5. Finally enqueue it: Insert packet at the end of the queue 6. Sort the queue in ascending order of packet timestamp

9、 Q&A

1、 怎么体现出 CLASS A 的 2 个接收窗口以及 B 的 beacon 和 ping 以及 CLASS C 的立即发送？

答：jit_enqueue

2、 有哪些 TODO？

答：

（1） code_to_char、char_to_code：improve error management
（2） TX_START_DELAY：get this value from HAL?
（3） TX_MARGIN_DELAY：How much margin should we take?
（4） CLASS C asap_count_us：Take 1 second margin, to be refined
（5） handle individual SF enabling and disabling (spread_factor)
（6） gps_process_coords：report other GPS statistics (typ. signal quality & integrity)
（7） handle sx1301 coutner wrap-up

（8） lgw_reg_w(LGW_GPS_EN, 1)：Is it necessary to protect here?

3、 GPS 提供哪些时间信息？

答：年、月、日、时、分、秒、小数秒、GPS 计时周数

/* result of the NMEA parsing */

下面是通过 NMEA 消息提供：

```
static short gps_yea = 0; /* year (2 or 4 digits) */
static short gps_mon = 0; /* month (1-12) */
static short gps_day = 0; /* day of the month (1-31) */
static short gps_hou = 0; /* hours (0-23) */
static short gps_min = 0; /* minutes (0-59) */
static short gps_sec = 0; /* seconds (0-60)(60 is for leap second) */
static float gps_fra = 0.0; /* fractions of seconds (<1) */
static int16_t gps_week = 0; /* GPS week number of the navigation
epoch */
```

下面是通过 SYNC 消息提供：

```
static uint32_t gps_iTOW = 0; /* GPS time of week in milliseconds */
static int32_t gps_fTOW = 0; /* Fractional part of iTOW (+/-500000) in
nanosec */
```

销售与服务

公司名称：长沙市锐米通信科技有限公司

公司网站：www.rimelink.com

产品销售：sales@rimelink.com 0731-8223 1246

技术支持：support@rimelink.com 0731-8223 6164

公司地址：长沙市普瑞大道 278 号 36 座 1403

