Time interval controller (TIC)

Таблица активности временных слотов. (timeSlotStateTable)

Состояние временного слота 0 timeSlotState	**Состояние временного слота 1 timeSlotState**	**Состояние временного слота N timeSlotState**	**Состояние временного слота 49 timeSlotState**
Флаг приема	Флаг приема	Флаг приема	Флаг приема
Флаг передачи	Флаг передачи	Флаг передачи	Флаг передачи

Аппаратный таймер 32.768K

setTimer(uint16_t time)
setInterruptTime(uint16_t time, timeslotNum)
setInterruptHandler((*handlerIRQ)(*timeslot))
adjustTimer (int adjustTime)

Обработчик аппаратного таймера hwTimerHandler(timeSlot)

Проверяем поле "Флаг передачи" таблици timeSlotStateTable
Если поле TRUE
  Вызываем обработчик передачи пакета(timeSlot)
  Вызываем TDMAScheduler(timeSlot)
  Вызываем обработчик slotEnd(timeSlot)
  ВЫХОД
Проверяем поле "Флаг приема" таблици timeSlotStateTable
Если поле TRUE
  Вызываем обработчик приема пакета(timeSlot)
  Вызываем TDMAScheduler(timeSlot)
  Вызываем обработчик slotEnd(timeSlot)
  ВЫХОД
Вызываем TDMAScheduler(timeSlot)
Вызываем обработчик slotEnd(timeSlot)
ВЫХОД

Планировщик TDMA. TDMASheduler(timeSlot)

Перебираем следующие timeSlot из timeSlotStateTabe
Если "Обработчик прием пакета" не равен NULL
ИЛИ "Обработчик передачи пакета" не равен NULL
  Устанавливаем аппаратный таймер на этот таймслот
  ВЫХОД
Тут мы окажемся если обработчиков нет.
Устанавливаем аппаратный таймер на таймслот 0

Time interval controller (TIC)

init()	Инициализация

setReceiveCallback(ticCallback_f handler)	Устанавливает обработчик интервала приема пакета
setSendCallback(ticCallback_f handler)	Устанавливает обработчик интервала передачи пакета
setSlotEndCallback(ticCallback_f handler)	Устанавливает обработчик по завершению слота
clearCallbacks()	Удаляет все обработчики

bool setReceptionState(timeSlot_t n, bool state)	Установить таймслот в режим приема пакетов
bool setTransmissionState(timeSlot_t n, bool state)	Установить таймслот в режим передачи пакета
bool getReceptionState(timeSlot_t n)	Прочитать активность состояния приема пакета
bool getTransmissionState(timeSlot_t n)	Прочитать активность состояния передачи пакета

bool adjustTimer(usec_t usec)	Подстроить значение таймера
bool setTimer(usec_t usec)	Установить значение таймера
usec_t getTimer()	Прочитать состояние таймера

ticCallback:Type
typedef void (*ticCallback_f)(timeSlot_t n)

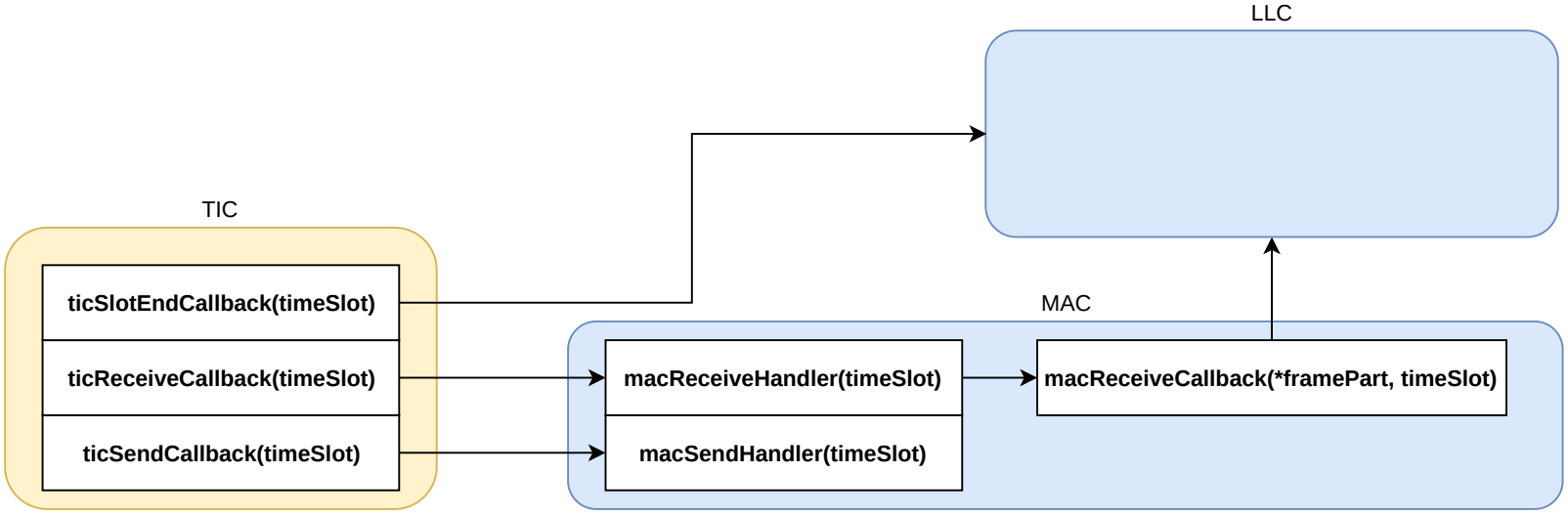
Media access control (MAC)

Состояние слотов. (slotStateTable)

Состояние слота 0 **slotState**	**Состояние слота N** **slotState**	**Состояние слота 49** **slotState**
Состояние передачи	*Состояние передачи*	*Состояние передачи*
Флаг "Есть данные"	Флаг "Есть данные"	Флаг "Есть данные"
Количество попыток передачи	Количество попыток передачи	Количество попыток передачи
Канал радиопередатчика	Канал радиопередатчика	Канал радиопередатчика
Указатель на пакет framePart	Указатель на пакет framePart	Указатель на пакет framePart
Состояние приема	*Состояние приема*	*Состояние приема*
Флаг "Прием разрешен"	Флаг "Прием разрешен"	Флаг "Прием разрешен"
Канал радиопередатчика	Канал радиопередатчика	Канал радиопередатчика

Принятый пакет **receivedPacket**
Указатель на пакет framePart
timeSlot принятого пакета

Пакет framePart создаеться malloc.



macSendHandler должен еще заполнять пакеты sync. Нужно подумать кто будет заполнять. MAC или протокол SYNC.

Media access control (MAC)

init()	Инициализация
mac_setReceiveCallback(ticCallback_f handler)	Устанавливает обработчик интервала приема пакета
mac_setSendCallback(ticCallback_f handler)	Устанавливает обработчик интервала передачи пакета
mac_clearCallbacks()	Удаляет все обработчики
mac_openReceiveSlot(timeSlot n, uint8_t ch)	Разрешает прием пакетов в заданный слот и заданном канале.
mac_closeReceiveSlot(timeSlot n)	Запрещает прием пакетов в заданный слот
bool mac_send(framePacket* framePacket, timeSlot n, uint8_t ch, uint8_t attempts)	Устанавлевает пакет для передачи в слот n, канале ch, количеством попыток attempts.Если тайм слот еще занят возвращает FALSE.
bool mac_getReceivedPacket(framePart* framePart)	Возвращает указатель на последний принятый пакет. FALSE если пакетов нет

Обработчик ticCallback_f sendHandler(timeSlot)

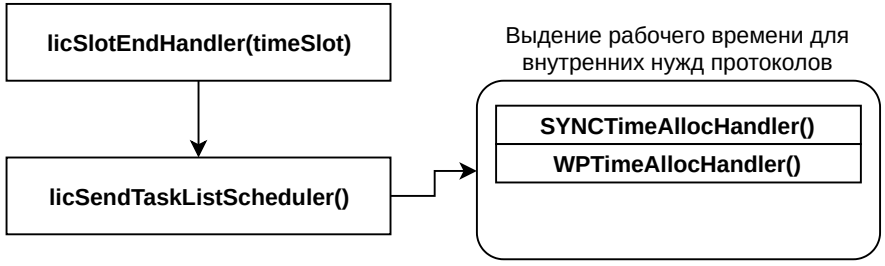
```
Процедура обработки события передачи пакета TIC
Если флаг "Есть данные" = FALSE
   Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
   ВЫХОД
Включить радиопередатчик, выставить частоту канала передачи
Если PPDU TYPE = WP
   Начать передачу
   Если передача не удалась (контроль CCA)
      Количество попыток - 1
       Если количество попыток = 0
          Уничтожаем пакет, флаг "Есть данные" = FALSE
          Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
       Выключаем радио, ВЫХОД
   Прием пакета в течении 1 мс(?)
   Если приняли ACK
      Уничтожаем пакет, флаг "Есть данные" = FALSE
      Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
      Выключаем радио, ВЫХОД
  Если не приняли ACK
     Количество попыток - 1
        Если количество попыток = 0
           Уничтожаем пакет, флаг "Есть данные" = FALSE
           Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
        Выключаем радио, ВЫХОД
ЕСЛИ PPDU TYPE = SYNC
   Заполнить поля HOUR, MIN, SEC, USEC = 0x80000000
   Начать передачу специальной функцией
   Если передача не удалась
      Количество попыток - 1
         Если количество попыток = 0
            Уничтожаем пакет, флаг "Есть данные" = FALSE
            Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
   Если передача удалась
      Уничтожаем пакет, флаг "Есть данные" = FALSE
      Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
      Выключаем радио, ВЫХОД
Если PPDU TYPE = WP BRADCAST
   Начать передачу
   Если передача не удалась (контроль CCA)
      Количество попыток - 1
       Если количество попыток = 0
          Уничтожаем пакет, флаг "Есть данные" = FALSE
          Запрещаем TIC обработку. setTransmissionState(timeSlot, FALSE)
       Выключаем радио, ВЫХОД
```

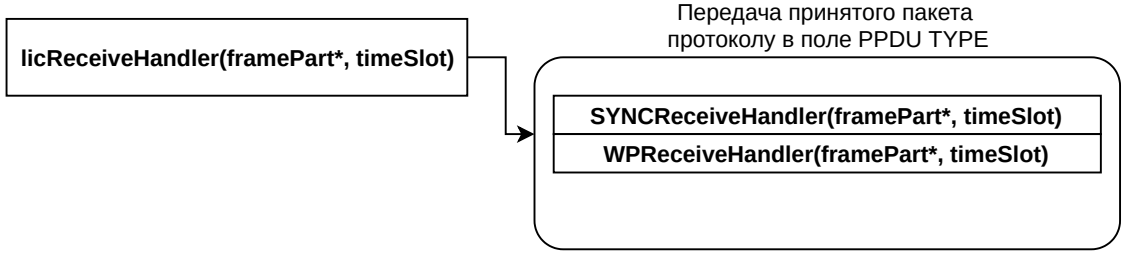
Обработчик ticCallback_f receiveHandler(timeSlot)

```
Процедура обработки события приема пакета TIC
Если флаг "Прием разрешен" равен FALSE
   Запрещаем TIC обработку. setTransmissionState(timeSlot,FALSE)
   ВЫХОД
Включить радиопередатчик, выставить частоту канала приема
Ждать приема не более 2-3 мс(?)
Если пакета не пришло
   Выключить радио, ВЫХОД
Если проверка NETID разрешена
   Если  NETID не равен нашей сети
      Уничтожаем, выключаем радио, ВЫХОД
Если версия протокола не равна нашей версии
   Уничтожаем, выключаем радио, ВЫХОД
Если PPDU TYPE  = WP
   Если размер пакета меньше 26 байт
      Уничтожаем, выключаем радио, ВЫХОД
  Если поле DST не соответсвует адресу узла
      Уничтожаем, выключаем радио, ВЫХОД
  Если поле DLEN не равно размер PPDU - 26 (размер DATA неверен)
      Уничтожаем, выключаем радио, ВЫХОД
  Передаем пакет ACK
  Выключаем радио
  Вызов обработчика приема пакета. macReceiveCallBack(*PPDU, timeSlot)
  ВЫХОД
```

Link logic control(LLC)

Задача отправки sendTask
Указатель sendTaskLast
Указатель sendTaskNext
Указатель на пакет framePart
Номер временного слота
Канал радиопередатчика





init()	Инициализация
LLC_addTask(framePart* framePart, timeSlot_t n, uint8_t ch)	Добавить задачу передачи пакета

licSendTaskListScheduler()

Составляет расписание на базе списка задач отправки сообщений
Перебор всех элементов списка sendTaskList
  Если временной слот MAC содержит данных для передачи
  {getTransmissionState(timeSlot_t n) = TRUE}
    Выбор следующего элемента списка
  Добавляем данные для передачи во временной слот
  Удаляем элемент списка
ВЫХОД

PPDU_HEADER_S	
uint8_t	LEN
struct	PPDU TYPE
	MPDU_TYPE:4
	PVERSION:4
uint8_t	NETID
uint8_t[8]	NODE_DST
uint8_t	MPDU_LEN

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	type
uint8_t	part_len
uint8_t*	part_data

framePart_e	
PPDU_HEADER	1
PPDU_FOOTER	2
MPDU_WP	3
MPDU_MDATA	4
MPDU_MIC	5
RAW	6
SYNC	7
METADATA	8

METADATA	
uint16_t	TIMESTAMP
int8_t	RSSI
txMethod_e	TX METHOD
void* (framePart)	SFD_CALLBACK

txMethod_e	
BROADCAST	1
UNICAST	2
CALLBACK	3

PPDU_FOOTER_S	
uint8_t	FCS1
uint8_t	FCS2

MPDU_WP	
uint8_t[8]	SRC
uint8_t	ETX
uint8_t	PORT
uint8_t	MDLEN

framePart_s	
framePart_s*	next
framePart_s*	last = 0
framePart_e	HEADER
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	WP
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	MDATA
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	MIC
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	FOOTER
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next = 0
framePart_s*	last
framePart_e	METADATA
uint8_t	part_len
uint8_t*	part_data

framePart_s	
framePart_s*	next
framePart_s*	last = 0
framePart_e	HEADER
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	SYNC
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	FOOTER
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next = 0
framePart_s*	last
framePart_e	METADATA
uint8_t	part_len
uint8_t*	part_data

framePart_s	
framePart_s*	next
framePart_s*	last
framePart_e	RAW
uint8_t	part_len
uint8_t*	part_data

↔

framePart_s	
framePart_s*	next = 0
framePart_s*	last
framePart_e	METADATA
uint8_t	part_len
uint8_t*	part_data

Структура пакетов

PPDU_HEADER						PPDU_FOOTER	
PPDU							
1	1	1	8	1		1	1
LEN	PPDU TYPE	NETID	NODE DST	MPDU LEN	MPDU	FCS1	FCS2

Пакет физического уровня кодирования. Поля FCS1,2 заменяются на значения RSSI и флага контроля CRC. Поле LEN не входит в контрольную сумму!!
PPDU TYPE кодирует тип содержимого пакет и версию протокола.
Поле NETID информирует о принадлежности пакета к определенной сети.
Для изначальной синхронизацией с сетью, узел принимает все доступные пакеты не зависимов от значения NETID. NODE DST - mac адрес узла которому предназначен пакет.
NODE DST 0xFF..FF ШВС
MPDU TYPE WP broadcast ШВС

PPDU Type							
7	6	5	4	3	2	1	0
PVERSION				MPDU TYPE			

MPDU: WP (wireless packet)				MPDU: MDATA	MPDU: MIC
8	1	1	1	MDLEN	4
SRC	ETX	PORT	MDLEN	MDATA	MIC
Authentification					
				Encryption	

DST - Поле получателя отсутсвует так как передача данных однонаправленная и получателем всегда является шлюз
SRC - отправитель, адресс узла создавшего пакет. При пересылке поле не меняется
ETX - растояние до маршрутизатора. Определено протоколом маршрутизации. Это поле заполнятся значением каждого узла при пересылке
PORT - номер порта потребителя пакета данных
MIC - Цифровая подпись пакета с данными
MDLEN - Размер данных полезной нагрузки
MDATA - Данные
Задача протокола доставить пакет до узла с ETH = 0

MPDU: ACK
0

Для подтверждения принятого пакета отсылается PPDU с типом ACK.
Поле NODE DST содержит адрес узла который подтвеждает получение
MPDU LEN = 0

MPDU: Sync				
1	1	1	1	4
ETX	HOUR	MIN	SEC	USEC

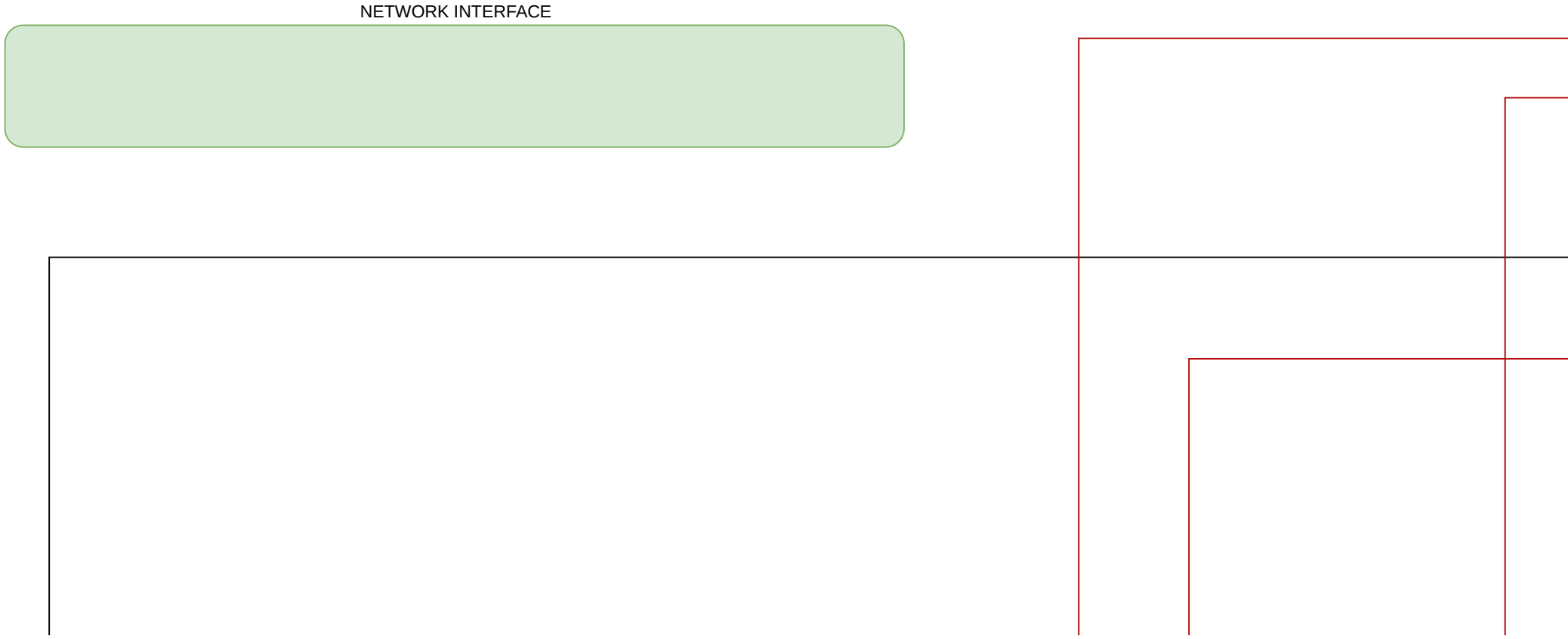
ETX - растояние до маршрутизатора. Синхронизация с узлами чей ETX меньше
HOUR - часы
MIN - минуты
SEC- секунды
USEC - количество микросекунд прошедших от начала временного слота номер 0
Пакеты типа PPDU TYPE = Sync при передаче автозаполняет.
Радиоинтерфейс содержит процедуру передачи с обратным вызовом при начале SFD. Обратный вызов заставляет MAC заполнить поля времени.

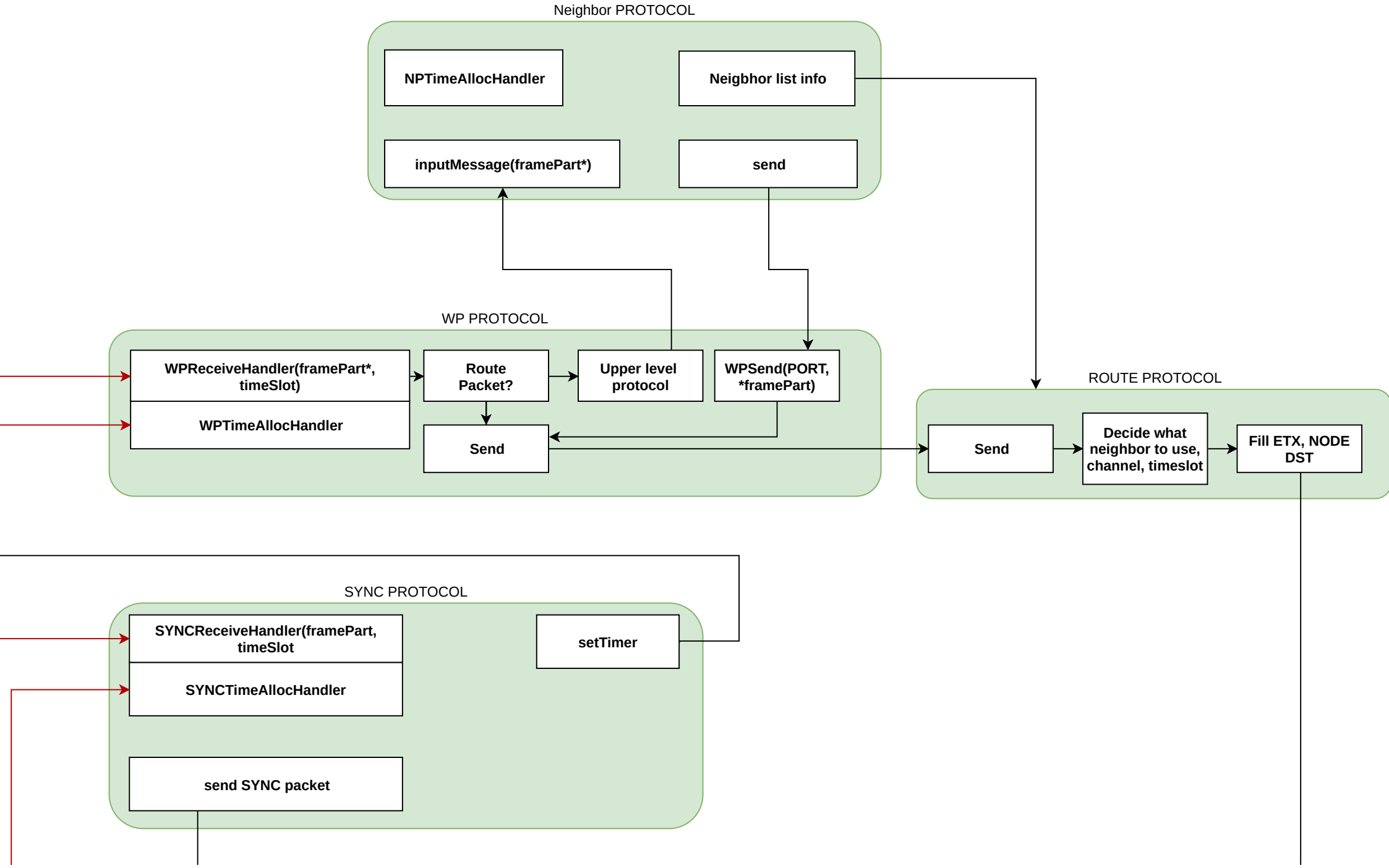
Методы	Описание
bool FP_create(framePart*,framePart_e type)	Создает framePart указанного типа
bool FP_delete(framePart* framePart)	Удаляет указаный framePart и связывает соседей
bool FP_getHeadChain(framePart* framePart)	Возвращает указатель на начало цепочки
bool FP_getTrailChain(framePart* framePart)	Возвращает указатель на конец цепочки
void FP_getPartLen(framePart* framePart, uint8_t* part_len)	Возвращает размер данных
void FP_getPartData(framePart* framePart, uint8_t* part_data)	Возвращает указатель на данные
void FP_addNext(framePart* framePart, framePart* nextPart)	Вставляет nextPart после framePart
void FP_addLast(framePart* framePart, framePart* lastPart)	Вставляет lastPart перед framePart
void FP_deleteChain(framePart* framePart)	Удаляет всю цепочку данных

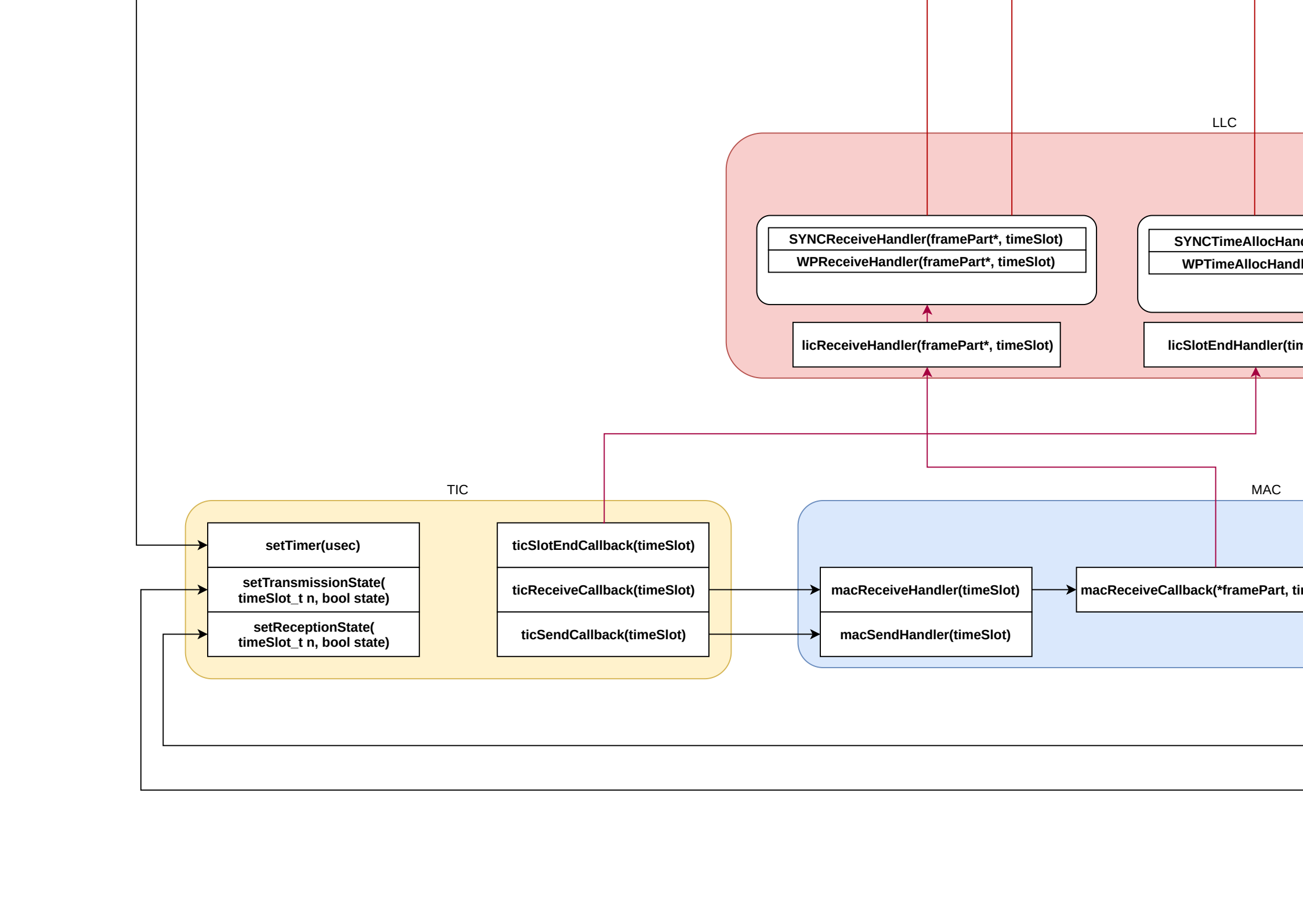
Radio interface (RI)

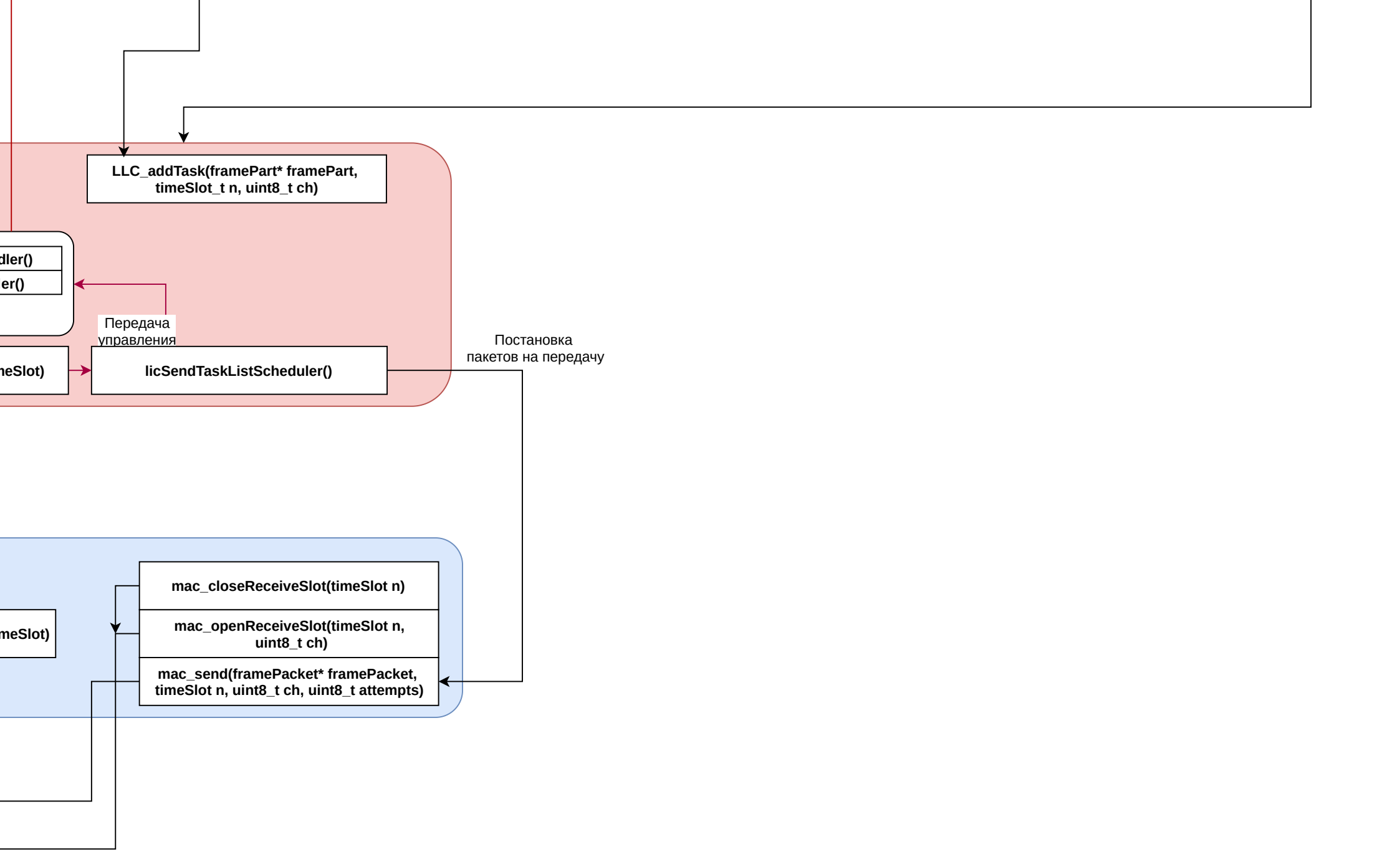
Методы	Описание
void RI_on()	Включить радиопередатчик
void RI_off()	Выключить радиопередатчик
bool RI_setChannel(n)	Установить радиоканал.
bool RI_send(*framePart)	Отправить пакет. Возращает TRUE или FALSE
bool RI_sendSFD(*framePart default, callback (*framePart))	Отправка пакета с отложенным заполнением данных. После начала передачи SFD происходит обратный вызов, который возвращает данные пакета. default - данные по умолчанию
bool RI_receive(*framePart, timeout)	Примем пакета с таймаутом.
uint32_t RI_getCRCErrorCount()	Прочитать количество ошибок CRC
uint32_t RI_getCCARejectCount()	Прочитать количество отказов канала
uint64_t RI_getRadioUptime()	Время работы радио в мкс
void RI_init()	Инициализация

Радиоинтерфейс для передачи данных использует пакет framePart типа PPDU_HEADER. Если пакет другого типа, функция завершит свою работу.
Для передачи пакета framePart, происходит обход всей цепочки framePart и загрузка в микросхему. PPDU_FOOTER не загружается и формируется самой микросхемой. Существует специальная функция радиопередачи - RI_sendSFD. Ее особенность в том, что перед ее вызовом пакет данных не передается. Запрос данных для передачи будет произведен после начала передачи заголовка SFD с помощью обратного вызова. Очень важно, что обратный вызов должен выполниться, до того как радиопередатчик начнет передавать первый байт данных, в противном случаии будут переданы данные default.
При приеме создается framePart типа RAW и добавляется информация METADATA. При передаче METADATA не нужна


NETWORK INTERFACE


Neighbor PROTOCOL
NPTimeAllocHandler
Neigbhor list info
inputMessage(framePart*)
send
WP PROTOCOL
WPReceiveHandler(framePart*, timeSlot)
WPTimeAllocHandler
Route Packet?
Upper level protocol
WPSend(PORT, *framePart)
Send
ROUTE PROTOCOL
Send
Decide what neighbor to use, channel, timeslot
Fill ETX, NODE DST
SYNC PROTOCOL
SYNCReceiveHandler(framePart, timeSlot
SYNCTimeAllocHandler
setTimer
send SYNC packet


LLC
SYNCReceiveHandler(framePart*, timeSlot)
WPReceiveHandler(framePart*, timeSlot)
SYNCTimeAllocHand
WPTimeAllocHandl
licReceiveHandler(framePart*, timeSlot)
licSlotEndHandler(tim
TIC
MAC
setTimer(usec)
setTransmissionState(
timeSlot_t n, bool state)
setReceptionState(
timeSlot_t n, bool state)
ticSlotEndCallback(timeSlot)
ticReceiveCallback(timeSlot)
ticSendCallback(timeSlot)
macReceiveHandler(timeSlot)
macSendHandler(timeSlot)
macReceiveCallback(*framePart, ti


LLC_addTask(framePart* framePart, timeSlot_t n, uint8_t ch)
dler()
er()
Передача управления
meSlot)
licSendTaskListScheduler()
Постановка пакетов на передачу
mac_closeReceiveSlot(timeSlot n)
meSlot)
mac_openReceiveSlot(timeSlot n, uint8_t ch)
mac_send(framePacket* framePacket, timeSlot n, uint8_t ch, uint8_t attempts)