# Network timer (NTMR)

| Переменные модуля | | |
|---|---|---|
| void | EventCallback | Обработчик события таймера |
| uint32 | capture | Значение установленное в МК |
| uint32 | offset | Разница между временем сети и значением аппаратного таймера |
| | | |
| | | |
| | | |
| | | |
| | | |

NTMR - реализует общий интерфейс работы с сетевым таймером. Он является посредником между аппаратным таймером и модулем TIC. Аппаратный таймер имеет разрядность 24 бита, нам же нужно 15 бит. При чтении аппаратного значения, можно урезать по маске 0x7FFF, но проблему с установкой прерывания таймера не решает.
При установке таймера NT_SetTime, нужно изменять значение capture в апаратном таймере на разницу между прошлым значением времени и новым, иначе нарушаться интервалы.

| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | bool | NT_SetTime | uint16 ticks | Устанавливает текущее значение таймера. Также нужно обновить значение capture. |
| Private | bool | NT_SetCapture | uint16 ticks | Устанавливает прерывание таймера на время ticks. Значения 0-32768 |
| Public | void | NT_IRQEnable | bool state | Запрещает или разрешает вызов обработчика события таймера |
| Public | void | NT_SetEventCallback | (*void)(uint16t ticks) | Установка обработчика события таймера. NULL - нет обработчика. |
| Public | uint16 | NT_GetTime | void | Возвращает значение таймера. Значения 0-32768 |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |

# Time interval controller (TIC)

| Таблица TSStateTable | |
|---|---|
| Номер TS | TSState |
| 0 | TS_RX |
| ... | TS_RX \| TS_TX |
| MAX_TS-1 | TS_RX |

| TSState | |
|---|---|
| TS_RX | 0b01 |
| TS_TX | 0b10 |

| Переменные модуля | | |
|---|---|---|
| #define | MAX_TS | Максимальное количество слотов |
| uint32 | UPTIME | Время в сек с момента включения |
| uint32 | RTC | Время RTC в сек от начала дня |
| #define | TS_ACTIVE | Длительность слота в tics |
| #define | TS_SLEEP | Длительность  паузы между слотами |
| *void | RXCallback | Обработчик приема |
| *void | TXCallback | Обработчик передачи |
| *void | SECallback | Обработчик завершения слота |

TIC использует таблицу TSSateTable для управления обратными вызовами приема/передачи. TIC контролирует работу аппаратного таймера. Если задач обработки пакетов нет, то будут вызовы раз в секунду SECallback в слот 0.
TIC предоставляет время UPTIME и RTC. TIC оперирует временем в относительных единицах - ticks.
Один tick равен 1/32768 сек. Аппаратный таймер CC2420 имеет разрядость 24 бита, что приводит к необходимости преобразования 16 битного значения к 24 битному. Sleep timer у CC2420 неуправляемый, значит TIC должен хранить смещение сетевого времени относительно таймера. Эта поправка должна учитываться при установке прерываний слотов.
NETTIME = (SLEEPTIMER + OFFSET) & MASK (24 младших бита).

**Нужно ли делать модуль таймера? Можно было бы в него убрать прерывания и приведение времени к 32768.**

| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | bool | TIC_SetTimer | uint16 tics | Устанавливает сетевой таймер. Если 0<tics<32767 то true. (или младшие 15 бит) |
| Public | uint16 | TIC_GetTimer | void | Возвращает значение таймера. (младшие 15 бит) |
| Public | bool | TIC_SetTXState | uint8 TS, bool state | Разрешает передачу в указаный слот. Возвращает false, если параметры некорректны. |
| Public | bool | TIC_SetRXState | uint8 TS, bool state | Разрешает прием в указаный слот. Возвращает false, если параметры некорректны. |
| Public | bool | TIC_GetTXState | uint8 TS | Возвращает флаг разрешения передачи. Если TS выходит за границ, возвращет false |
| Public | bool | TIC_GetRXState | uint8 TS | Возвращает флаг разрешения приема. Если TS выходит за границ, возвращет false |
| Public | void | TIC_SetRXCallback | (*void)(uint8 TS) | Установка обработчика приема пакета. NULL - нет обработчика |
| Public | void | TIC_SetTXCallback | (*void)(uint8 TS) | Установка обработчика передачи пакета. NULL - нет обработчика |
| Public | void | TIC_SetSECallback | (*void)(uint8 TS) | Установка обработчика завершения слота. NULL - нет обработчика |
| Private | void | TIC_TDMAShelduler | uint8 TS | Планировщик времени. Анализирует таблицу и планирует интервалы пробуждения микроконтролера. |
| Private | void | TIC_HW_Timer_IRQ | void | Обработчик аппаратного прерывания. Передаем в NT |
| Private | uint8 | TIC_getCurrentTS | void | Считывает состояние таймера и преобразует значение в TS. 0xFF - слот не определен или интервал паузы |
| Public | uint32 | TIC_getUptime | void | Возвращает время работы устройства после включения в сек |
| Public | uint32 | TIC_getRTC | void | Возращает время прошедшее с начала суток (00:00) в сек. 0xFFFFFFFF - время не установлено |
| Public | bool | TIC_setRTC | uint32 RTC | Устанавливает время прошедшее с начала суток в сек.  false если аргумент не корректен. |
| Public | void | TIC_SetNTController | (*NT) | Передать модулю TIC указатель на NetworkTIme контроллер. |

**TIC_HW_Timer_IRQ**

```
TS = TIC_getCurrentTS()

ЕСЛИ TS = 0xFF
   Обработчики долго работали.
   Вышли за границы временого слота.
   Нужно вычислить следующий ближайший слот.
   TIC_TDMAShelduler(0xFF)

Проверяем поля таблицы TSStateTable
Вызываем обработчик приема/передачи (что то одно)
Вызываем планировщик TIC_TDMAShelduler(TS)
Вызываем обработчик *SECallback(TS)
```

**Планировщик TDMA. TIC_TDMAShedulер(TS)**

```
ЕСЛИ TS = 0xFF
   Вычисляем следующий временой слот исходя из значения таймера
   Устанавливаем на него таймер
   ВЫХОД

Перебираем следующие TS из TSStateTable
ЕСЛИ TSState != 0
   Устанавливаем аппаратный таймер на этот таймслот
   ВЫХОД

Тут мы окажемся если обработчиков совсем нет.
Устанавливаем аппаратный таймер на таймслот 0
```

# Media access control (MAC)

| Таблица MACSlotTable | |
|---|---|
| Номер S | MACSState |
| 0 | MACSState |
| ... | MACSState |
| MAX_S-1 | MACSState |

| MACSState struct | |
|---|---|
| ***TX*** | |
| bool | enable |
| uint8 | attempts |
| uint8 | ch |
| FChain | *FChain |
| ***RX*** | |
| bool | enable |

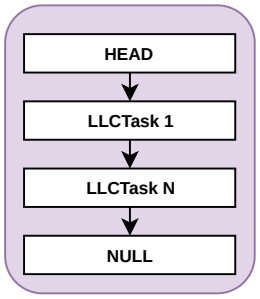
MAC реализует протокол приема-передачи пакетов.  Формирует подтверждения ACK, фильтрует по адресам, проверяет целостность CRC. Для работы MAC нужен таймер измерения интервалов времени.
MAC_Send не ставит пакет на передачу если слот занят. Контроль слота - обязанность вышестоящего модуля LLC.
MAC напрямую управляет радиопередатчиком. Передача пакетов ведется с проверкой CCA.

| Переменные модуля | | |
|---|---|---|
| #define | MAC_S | Максимальное количество слотов |
| void | RXCallback | Обработчик принятого пакета |
| #define | RX_MAX | Максимальное время приема мс? |
| #define | RX_WAIT | Время ожидания пакета. |
| #define | TX_MAX | Максимальное время передачи |
| | | |
| | | |
| | | |

| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | void | MAC_RX_HNDL | uint8 TS | Процедура приема пакета. Использует интерфейс радио. Информацию о частоте берется из MACSlotTable |
| Public | void | MAC_TX_HNDL | uint8 TS | Процедура передачи пакета. Использует интерфейс радио. Информацию о частоте и канале берется из MACSlotTable |
| Public | void | MAC_OpenRXSlot | uint8 TS, uint8 CH | Открывает слот приема TS на частоте CH. |
| Public | void | MAC_CloseRXSlot | uint8 TS | Закрыть слот приема TS. |
| Public | void | MAC_Send | *FChain, attempts | Привязка пакета на отправку к слоту TS. Данные о канале и слоте в FChain.META. |
| Private | bool | MAC_Filter | *FChain | На входе [ETH_DATA, META]. Проверяет корректность полей. Возвращает false если фитрация не пройдена. |
| Private | bool | MAC_Decode | *FChain | На входе [RAW, META] -> [ETH_DATA, META]. *Изменяет FChain. Возвращает false - если декодирование невозможно. |
| Public | void | MAC_SetRXCallback | (*void)(*Fchain) | Установка обработчика приема пакета. NULL - нет обработчика |
| Public | void | MAC_SetTICController | (*TIC) | Передать модулю MAC указатель на TIC контроллер. |
| Public | bool | MAC_GetTXState | uint8 TS | true - слот содежит пакет для передачи. false - слот не занят. |
| Public | bool | MAC_GetRXState | uint8 TS | true - слот открыт для приема. false - слот закрыт для приема. |
| Public | void | MAC_SetRIController | (*RI) | Передать модулю MAC указатель на интерфейс радио |
| Public | | | | |
| Public | | | | |

# Link logic control(LLC)

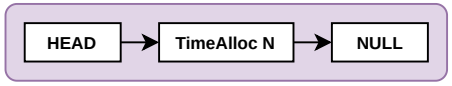
**TaskPool**




| LLCTask struct | |
|---|---|
| LLCTask | *next |
| uint8 | TS |
| uint8 | CH |
| FChain | *FChain |

| TimeAlloc struct | |
|---|---|
| TimeAlloc | *next |
| *void | void |

**TimeAllocQueue**




LLC занимается постановкой пакетов на отправку из пула задач в MAC уровень. В конце каждого слота LLC проверяет пул на предмет освободившегося слота передачи MAC. Если такая возможность есть, LLCTask уничтожается (но не FChain).
В конце работы планировщика, LLC передает управления по списку протоколам высокого уровня. Список вызовов реализован в виде очереди TimeAllocQueue

| Переменные модуля | | |
|---|---|---|
| | | |
| void | RXCallback | Обработчик принятого пакета |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | void | SE_HNDL | uint8 TS | Обработчик завершения слота. Передает управление LLC_Shelduler(TS) и далее сторонним подписчикам |
| Private | void | LLC_Shelduler | uint8 TS | Планировщик передачи пакетов MAC уровню. Для оптимизации может использовать TS. |
| Public | void | LLC_SetMACController | (*MAC) | Передать модулю LLC указатель на MAC контроллер. |
| Public | void | LLC_RX_HNDL | (*void)(*Fchain) | Обработчик входящего пакета |
| Public | void | LLC_SetRXCallback | (*void)(*Fchain) | Установка обработчика приема пакета. NULL - нет обработчика |
| Public | void | LLC_TimeAlloc | (*void)(void) | Добавляет обратный вызов для выделения времени. |
| Private | void | LLC_RunTimeAlloc | void | Запускает процесс выделения времени. |
| Public | | | | |
| Public | | | | |
| Public | | | | |
| Public | | | | |
| Public | | | | |
| Public | | | | |
| Public | | | | |

# Элементарный элемент данных пакета FItem

| FItem_s | |
|---|---|
| FItem_s* | next |
| FItem_s* | last |
| FItem_t | type |
| uint8_t | length |
| uint8_t* | data |

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

| Global params | | |
|---|---|---|
| uint8_t | quantity | Кол-во созданных элементов |

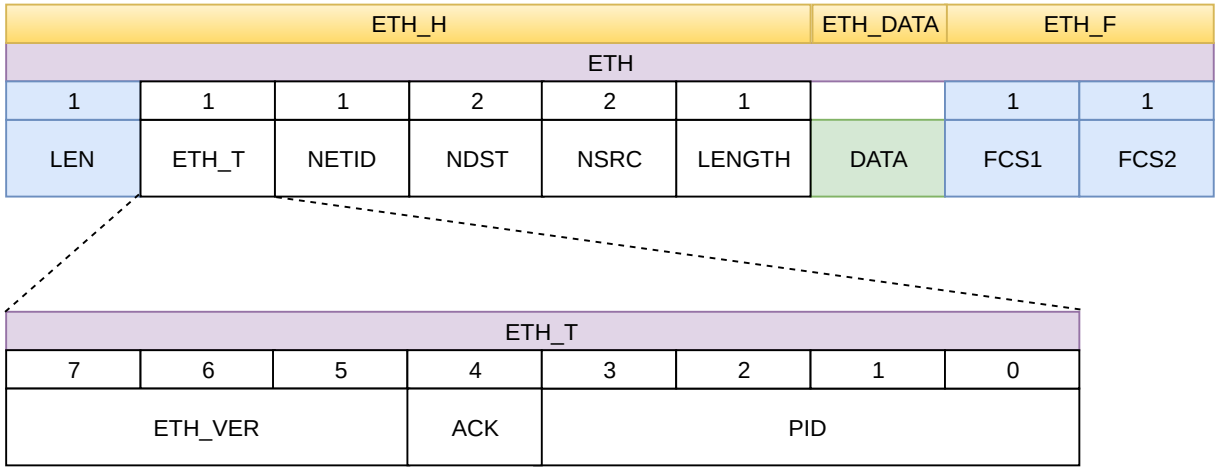
| Методы | Описание | |
|---|---|---|
| bool FI_create(FItem_s* fi, FItem_t type, uint8_t* data, uint8_t length) | Создает FItem указанного типа | Увеличивает или уменьшает глобальный quantity |
| void FI_delete(FItem_s* fi) | Удаляет указаный FItem. Соседей не связывает | |
| void FI_setNext(FItem_s* fi, FItem_s* fi_next) | Свзяывает со следующим соседом | |
| void FI_setLast(FItem_s* fi, FItem_s* fi_last) | Связывает с предыдущем соседом | |
| void FI_getNext(FItem_s* fi, FItem_s* fi_next) | Возвращает следующего соседа | |
| void FI_getLast(FItem_s* fi, FItem_s* fi_last) | Возвращает предыдущего соседа | |
| FItem_t FI_getType(FItem_s* fi) | Возвращает предыдущего соседа | |
| uint8_t FI_getLength(FItem_s* fi) | Возвращает длинну данных | |
| uint8_t* FI_getData(FItem_s* fi) | Возвращает указатель на данные | |
| uint8_t FI_getObjectQuantity() | Возвращает количество существующих обьектов | |

Медуль реализован в виде отдельных функция по работе с структурой FItems_s. Элементы создаются на куче, данные data копируются при создании элемента. Предполагается, что размер данных никогда меняться не будет вплоть до уничтожения обьекта. Контроль количества существующих обьектов в памяти производится с помощью uint8_t FI_getObjectQuantity()

| txMethod_e | |
|---|---|
| BROADCAST | 1 |
| UNICAST | 2 |
| | |

| METADATA | |
|---|---|
| **RX** | |
| uint16_t | TIMESTAMP |
| int8_t | RSSI |
| **TX** | |
| txMethod_e | TX METHOD |
| void* (framePart) | SFD_CALLBACK |
| uint8_t | ETH_H.ETH_T.TYPE |
| bool | ACK |

# Структура ethernet пакета




| | |
|---|---|
| LEN | Общая длина посылки. Поле не контролируется в CRC. |
| FCS1,2 | Поля контролируются радиопередатчиком. Заменяются на RSSI, CRC. |
| ETH_T | Кодирует тип содержимого и версию протокола. |
| NETID | Идентификатор принадлжености к сети. Значения от 0 до 0xFF |
| NDST | Адрес узла получателя пакета. |
| NSRC | Адрес узла отправителя пакета. |
| LENGTH | Размер содержимого MPDU |
| ETH_DATA | Данные |
| ETH_VER | Версия протокола. Значения от 0 до 7. |
| PID | Protocol ID. Номер низкоуровневого протокола. Значения от 0 до 15 |
| ACK | 1 - Пакет требует подтверждения. |

Порядок фильтрации пакетов

| № | Поле | Значение | Описание | Действие |
|---|---|---|---|---|
| 1 | FCS1,2 | CRC | Неверная контрольная сумма | Удаление |
| 2 | LEN | RXBUF.LEN | Поле LEN не равно фактической длинне пакета | Удаление |
| 3 | NETID | Сеть | Пакет не пренадлежит сети | Удаление |
| 4 | ETH_VER | Версия узла | Узел не поддерживает версию протокола | Удаление |
| 5 | TYPE | Весия | Узел не имеет обработчика протокола | Удаление |
| 6 | LENGTH | Вычисляем | Поле размера данных некоректно | Удаление |
| 7 | NDST | 0xFFFF | Пакет для всех узлов | Обработка |
| 8 | NDST | Адрес узла | Пакет для этого узла | Обработка |

*NSRC = 0xFFFF Обозначает пакет от узла без адреса.

При начальной синхронизации фильтры пакетов отключаются, или настриваются на прием пакета SYNC. Примемник не разбирает каким образом был передан пакет: либо в ШВС тайм слот 0, или как unicast сообщение, главное что поле NDST = 0xFFFF. При ШВС ответа ACK не должно быть, а при unicast должно быть. Этот момент под ответсвенностью отправителя пакета.

# Фильтрация пакетов

Каждый уровень имеет свою логику по  фильтрации пакетов. Уровень MAC применяет фильтры непосредственно ethernet frame. Фильтры MAC включают в себя проверку своевременности прихода пакетов, к примеру NDST = 0xFFFF могутприходить только в слот 0. NDST = 0xFFFF, должны иметь ACK = 0.

# Структура SYNC пакета

| SYNC | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2 |
| ETX | HOUR | MIN | SEC | TIMER32K |

| | |
|---|---|
| ETX | Expected transmissions. Ожидаемое количество передач до точки доступа. |
| HOUR | Часы. Значения от 0 до 23 |
| MIN | Минуты. Значения от 0 до 59 |
| SEC | Секунды. Значения от 0 до 59 |
| TIMER32K | Значения таймера от 0 до 32767 (32768 значений) |

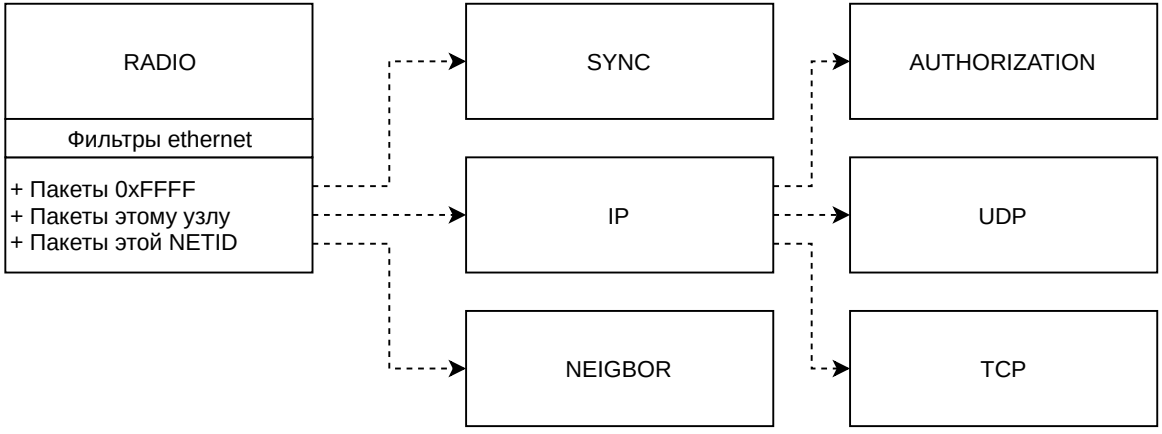
Узлы могут не иметь RTC и не ретранслировать часы, минуты, секунды. В этом случаии поля заполняются значениями 0xFF.

# Структура ACK пакета

| MPDU: ACK |
|---|
| 0 |

Для подтвеждения принятого пакета отправляем пакет с типом TYPE = ACK. LENGTH = 0.
NDST = Адрес узла отправителя данных, NSRC = свой адрес.

# Иерархия протоколов



# Цепочка данных FChain

| FChain_s | |
|---|---|
| FItem_s* | head |
| FItem_s* | tail |
| uint8_t | quantity |
| FItem_s* | iterator |
| | |

| Global params | | |
|---|---|---|
| uint8_t | quantity | Кол-во созданных элементов |

FChain унифицирует процес работы с структорой пакета данных в стеке протокола.
Интерфейс радио передатчика использует FChain, приемник возвращает FItem типа RAW

| **Методы** | **Описание** |
|---|---|
| void FC_create(FChain_s* fc) | Создает цепочку |
| void FC_delete(FChain_s* fc) | Удаляет всю цепочку включая ее элементы |
| void FC_iteratorToHead(FChain_s* fc) | Устанавливает итератор на начало списка |
| void FC_iteratorToTail(FChain_s* fc) | Устанавливает итератор на конец списка |
| void FC_iteratorToType(FChain_s* fc) | Перемещает итератор с текущего положения на следующий элемент типа type |
| void FC_iteratorToTypeHead(FChain_s* fc) | Перемещает итератор с начала цепочки на следующий элемент типа type |
| void FC_next(FChain_s* fc) | Перемещаем итератор на следующий элемент |
| void FC_last(FChain_s* fc) | Перемещаем итератор на предыдущий элемент |
| void FC_insertAfter(FChain_s* fc, FItem_s* fi) | Вставляет элемент после итератора |
| void FC_insertBefore(FChain_s* fc, FItem_s* fi) | Вставляет элемент до итератора |
| uint8_t FC_getQuantity(FChain_s* fc) | Возвращает количество элементов |
| bool FC_getIterator(FChain_s* fc, FItem_s* fi) | Возвращает FItem итератора. true - есть элемент |
| void FC_getObjectQuantity() | Возвращает количество существующих обьектов |

# Структура IP пакета

| IP_HEADER | | | | | IP_DATA_CRYPT | IP_MIC |
|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 1 | 1 | LENGHT | 4 |
| ETX | FDST | FSRC | IPP | LENGTH | DATA | MIC |
| | Authentification | | | | | |
| | | | | | Encryption | |

| | |
|---|---|
| ETX | Expected transmissions. Ожидаемое количество передач до точки доступа. |
| FDST | Адрес конечного узла назначения пакета. |
| FSRC | Адрес узла создателя данного пакета. |
| IPP | Номер вышестоящего протокола обработчика пакета |
| LENGTH | Размер данных. |
| IP_DATA_CRYPT | Зашифрованые данные. |
| MIC | Message integrity code. Код целостности сообщения. |

| № | Поле | Значение | Описание | Действие |
|---|---|---|---|---|
| 1 | LENGTH | Вычисляем | Размер данных не соответсвует длине пакета | Удаляем |
| 2 | MIC | Авторизация | Авторизация не пройдена | Удаляем |
| 3 | FDST | 0xFFFF | Пакет для всех узлов | Обработка |
| 4 | FDST | Адрес узла | Пакет для этого узла | Обработка |
| 5 | FDST | 0x0000 | Маршрутизация до ТД алгоритмом СТР. Только пакеты с FSRC!=0xFFFF, 0x0000 | Маршрутизируем |
| 6 | FDST | ANY | Маршрутизация до узла алгоритм таблиц. Только пакеты с FSRC = 0x0000 | Маршрутизируем |
| 7 | PROTOCOL | | Если нет обработчкика. При маршрутизации не имеет значения есть ли у нас обработчик. | Удаляем |

ETX не включен в авторизацию сообщения. Это поле изменяется при маршрутизации пакета.

# Radio interface (RI)

| Переменные модуля | | |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | void | RI_On | void | Включить радиопередатчик. Прием и передача неактивна. |
| Private | void | RI_Off | void | Выключить радиопередатчик. |
| Public | bool | RI_SetChannel | uint8 CH | Установить канал. false - неверный аргумент |
| Public | bool | RI_Send | *FChain | Отправка пакета RAW. false - ошибка CCA. Возможен вызов callback из META. |
| Public | *FChain | RI_Receive | uint8 timeout | Переводит радио в режим приема пакета. timeout в мс. Возвращает NULL , если пакета нет. |
| Public | uint32 | RI_GetCRCError | void | Возвращает количество ошибочно принятых пакетов. |
| Public | uint32 | RI_getCCAReject | void | Возвращает количество коллизий CCA при передачи |
| Public | uint64 | RI_getUptime | void | Возвращает сумарное время работы радио в режимах прием/передача в мс. |
| Public | | | | |

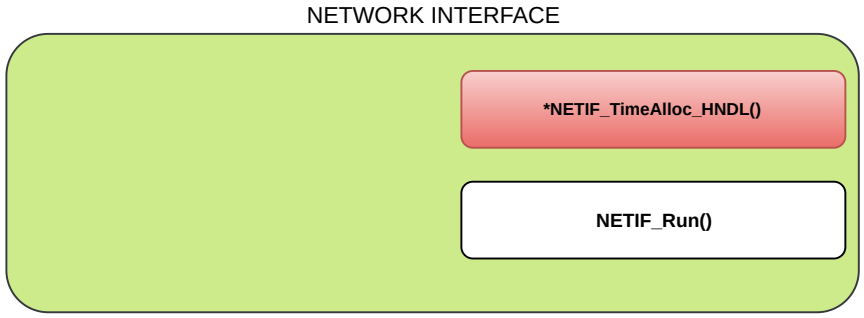
| Функции модуля | | | | |
|---|---|---|---|---|
| Access | Return | Name | Args | Описание |
| Public | void | | | |
| Private | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |
| Public | void | | | |

# Алгоритм поиска сети

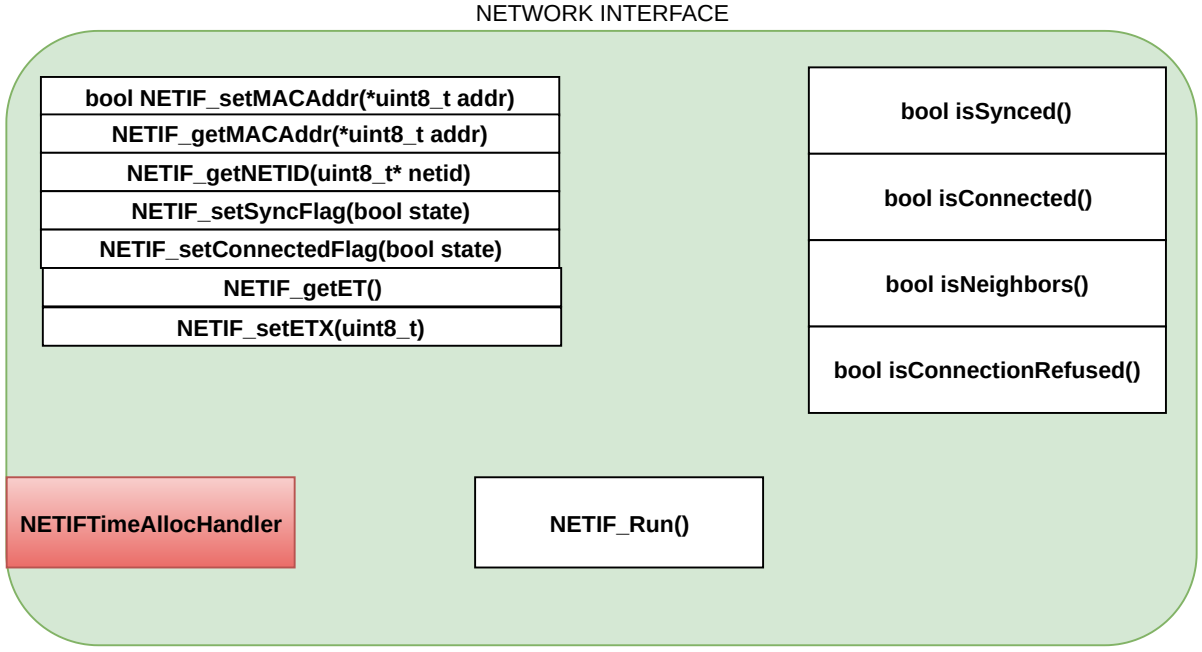
Алгоритм поиска сети запускается NETIF. Поиск сети происходит путем перебора частотных каналов и поиска сигнала синхронизации. Синхронизация с сетью осуществляется путем установки значения таймера TIC в соответсвии с принятым временем сети. После этого работа алгоритма завершена.
Алгоритм можно сделать умным, добавив возможность запоминать на определенне время сети недоступные для подключения. TODO: кадр синхронизации должен содержать посылку закодированную ключом сети, это позволит убедится что ключ шифрования подходит этой сети. Посылка может быть простым текстом "HELLO". При поиске сети алгоритм напрямую общается с радиоинтерфейсом.

## User application

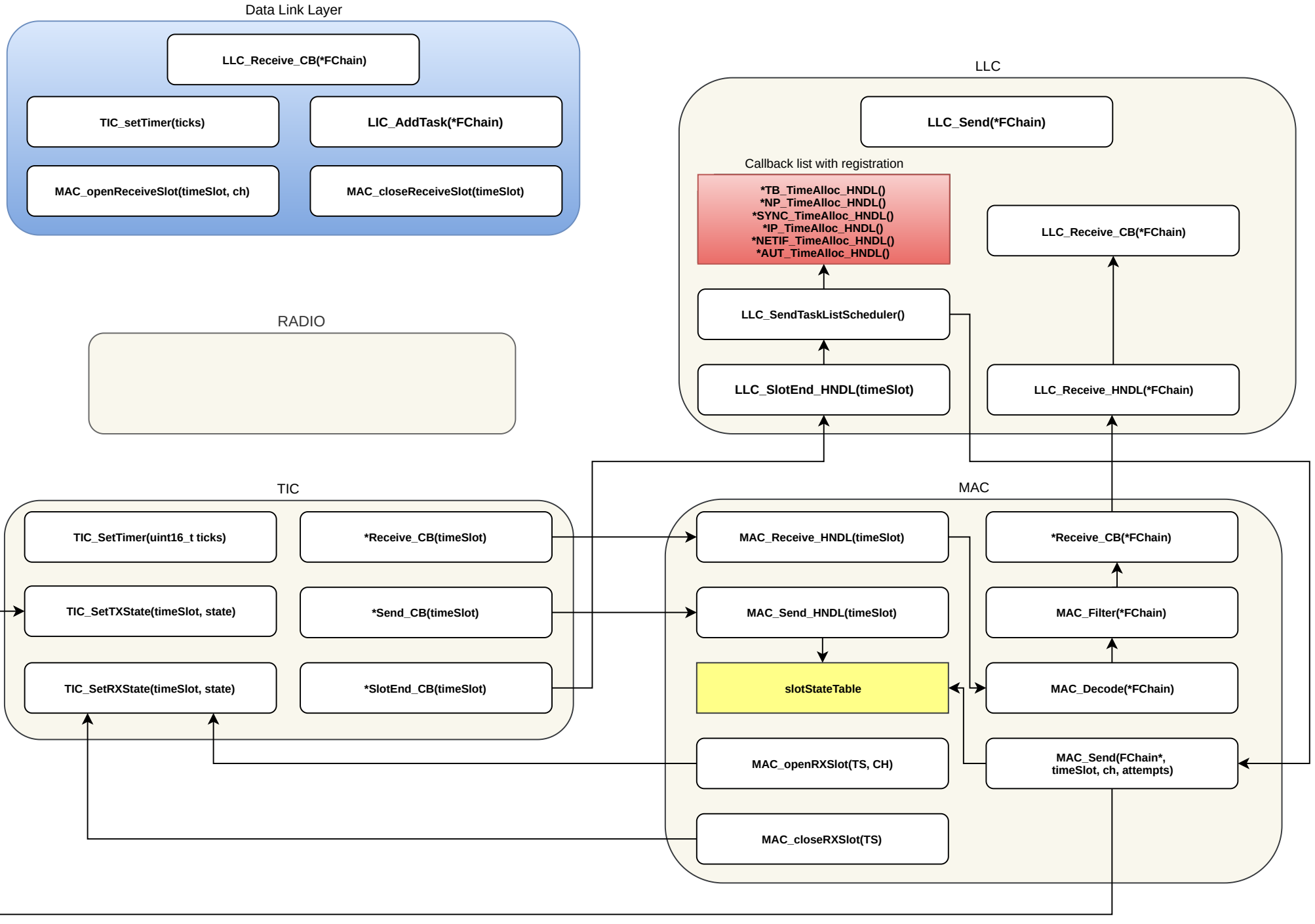
Обрабатывает входящие пакеты, отправляет сообщения точке доступа. Управление получает либо из главного цикла при выходе из прерывания таймера, или по прерываниям настроенных пользователем.

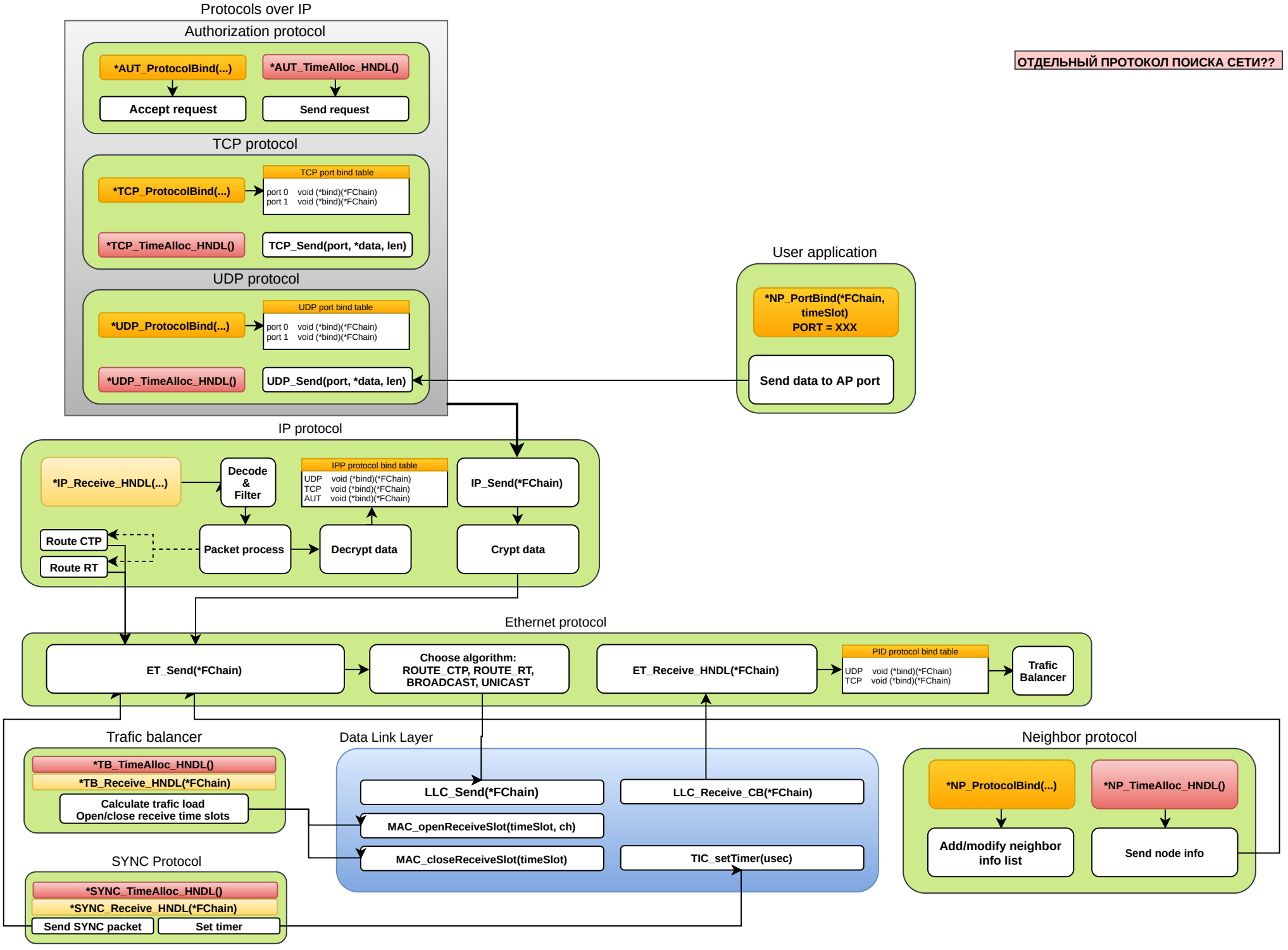



Протоколы взаимодействуют между собоай при помощи чтения состояния NETIF.
Состояние НЕ СИНХРОНЕЗИРОВАН
  РаботаетSYNC протокол в режиме прямого доступа к радио (или всеже MAC).
  Перебирает разрешенные каналы (список в NETIF)
  Ищет синхропакет и устанавлевает состояние NETIF СИНХРОНЕЗИРОВАН
  У NETIF проверяет что к сети с NETID можно подключиться
Состояние СИНХРОНИЗИРОВАН
  Считает что каждый протокол очистил свои данные пока был в режиме НЕ СИНХР.
  Начинает свою работу протокол Соседей.
  При наличии соседей установит NETIF что есть соседи.
  Если есть соседи активируется протокол Подключения.
  Он отсылает запрос шлюзу о подключении
  Обратный ответ приходит через ШВС.
  Если подключение разрешено, то устнавливает NETIF В состояниее подключенно
  Если подключение невозможно, то устанавливает флаг Подключение отказано
NETIF увидит что в подключении отказано и сбросит синхронизацию пометив сеть на некоторое время недоступной.
В случаи успешного подключения от пользователя принимаем пакеты на передачу



# Data Link Layer




Protocols over IP
Authorization protocol
*AUT_ProtocolBind(...)
*AUT_TimeAlloc_HNDL()
Accept request
Send request
TCP protocol
TCP port bind table
*TCP_ProtocolBind(...)
port 0 void (*bind)(*FChain)
port 1 void (*bind)(*FChain)
*TCP_TimeAlloc_HNDL()
TCP_Send(port, *data, len)
UDP protocol
UDP port bind table
*UDP_ProtocolBind(...)
port 0 void (*bind)(*FChain)
port 1 void (*bind)(*FChain)
*UDP_TimeAlloc_HNDL()
UDP_Send(port, *data, len)
ОТДЕЛЬНЫЙ ПРОТОКОЛ ПОИСКА СЕТИ??
User application
*NP_PortBind(*FChain, timeSlot)
PORT = XXX
Send data to AP port
IP protocol
*IP_Receive_HNDL(...)
Decode & Filter
IPP protocol bind table
UDP void (*bind)(*FChain)
TCP void (*bind)(*FChain)
AUT void (*bind)(*FChain)
IP_Send(*FChain)
Route CTP
Route RT
Packet process
Decrypt data
Crypt data
Ethernet protocol
ET_Send(*FChain)
Choose algorithm: ROUTE_CTP, ROUTE_RT, BROADCAST, UNICAST
ET_Receive_HNDL(*FChain)
PID protocol bind table
UDP void (*bind)(*FChain)
TCP void (*bind)(*FChain)
Trafic Balancer
Trafic balancer
*TB_TimeAlloc_HNDL()
*TB_Receive_HNDL(*FChain)
Calculate trafic load
Open/close receive time slots
Data Link Layer
LLC_Send(*FChain)
LLC_Receive_CB(*FChain)
MAC_openReceiveSlot(timeSlot, ch)
MAC_closeReceiveSlot(timeSlot)
TIC_setTimer(usec)
Neighbor protocol
*NP_ProtocolBind(...)
*NP_TimeAlloc_HNDL()
Add/modify neighbor info list
Send node info
SYNC Protocol
*SYNC_TimeAlloc_HNDL()
*SYNC_Receive_HNDL(*FChain)
Send SYNC packet
Set timer

## TIC

TIC - time interval controller, Контроллер управления интервалами времени. Реализует основной алгоритм TDMA: разделяет отрезок времени на временые слоты. Каждому слоту соответсвует одно из действий: примем или передача пакета данных. TIC самостоятельно не занимается передачей данных, а производит обратные вызовы *Receive_CB(timeSlot), *Send_CB(timeSlot). Управлять работой контролера можно с помощью TIC_SetTXState(timeSlot, state), TIC_SetRXState(timeSlot, state), где state = true/false. Планировщик времени TIC, на основе таблицы состояний временых слотов, управляет работой аппаратного таймер. После завершения обратных вызовов, произойдет вызов *SlotEnd_CB(timeSlot), его цель уведомить о завершении временого слота и предоставить другим частям протокола время для своих действий. В случаи если нет активных действий в слотах, то планировщик будет вызывать SlotEnd_CB в конце слота 0.
Для подстройки значений таймера служит метод TIC_SetTimer(usec), он записывает новое значение времени в аппаратный таймер.
Согласно протоколу длительность временного интервала 1 сек.

## MAC

MAC - media access control, контроллер доступа к среде передачи. Взаимодействует с радиоинтерфейсом и осуществляет прием или передачу пакетов. С точки зрения реализации передачи данных, MAC редставляет собой 50 буферов (по количеству временых слотов), и MAC_Send(FChain*, timeSlot, ch, attempts) устанавливает пакет для передачи в буфер timeSlot. MAC открывает слот передачи TIC. Передача пакетов произойдет в соответсвии с метододом txMethod_e указанным в METADATA пакета. Примем и передача происходят в соответсвии с протоколом обмена сообщениями.
Для подтверждения приема пакета, MAC уровень разбирает пакет на ETH_H и RAW. Тут же производится фильтрация пакетов. Данные о NDST, NSRC заносим в METADATA.
Обработка принятых пакетов происходит сразу после завершения приема данных посредством обратного вызова *Receive_CB(*IFrame=[ETH_H, RAW])

## LLC

LLC - link logic controller, контроллер логики обмена данными. LLC разбирает очередь пакетов на передачу и по мере готовности MAC ставит на отправку. В отличии от MAC, когда буфер содержит только один пакет для передачи, очередь LLC содержит множество пакетов для передачи. LLC никоем образом не уведомляет верхнии уровни о результатах передачи. Нельзя узнать был ли отправлен пакет данных.
Обработка завершения временого слота LLC_SlotEnd_HNDL(timeSlot). LLC запускает свой планировщик, который передаст пакеты MAC. После чего пройдется по списку зарегистрированных обратных вызовов для раздачи рабочего времени остальным участникам стека.
Асинхронный прием пакетов LLC_Receive_HNDL(*FChain[RAW,META], timeSlot). LLC не совершает никаких операция над пакетом.  После возврата из callback.

## SYNC protocol

Протокол обрабатывает входящии сигналы синхронизации и генерирует свои собственные. Использует тип передаваемого пакета CALLBACK. После начала передачи заголовка SFD, происходит обратный вызов, который возвращает данные для передачи.

## IP protocol

Метод WP_Send(*FChain, port) ожидает на входе FChain([MDATA, METADATA]). Согласно структуре пакета WP протокол добавляет поля WP, MIC и шифрует данные, а также важное поле ETX, значение которого получает из NETIF. Получается FChain([WP, MDATA, MIC, METADATA]), он передается протоколу маршрутизации.
Обрабатывает входящие пакеты с полем MPDU TYPE равным WP. Обработка входящих пакетов происходит через функцию WP_Receive_HNDL(*FChain, timeSlot). FChain содержит элементы [PPDU_HEADER, WP, MDATA, MIC, METADATA]. Если поле NODE DST совпадает с адресом узла или 0xFF..FF (ШВС), то пакет расшифровывам. Расшифрованый пакет передаем зарегистрированному обработчику через список port bind table. Для экономии памяти список не фиксированной длинны, он создается динамически при регистрации функций. Все остальные пакеты подлежат пересылке, но пересылаются только те пакеты поле ETX, которых больше чем ETX узла. К пример ETX узла 4(4 узла до точки доступа), а мы приняли пакет с ETX = 2. Такой пакет уничтожается, потому что на каждом шаге передачи поле пакета ETX должно уменьшатся для достижения точки доступа.
Получателем данных узла всегда является точка доступа сети. Данные от точки доступа к узлу приходят в виде ШВС.

## Trafic balancer

Анализирует поток трафик с помощью вызова TB_Receive_HNDL, производит открытие или закрытие слотов приема данных. Всегда открыт слот номер 0, так как он системный. Информацию об открытых слотах передает Neighbor protocol, который распространяет информацию об узле соседям. Балансер нужен для динамического управления пропускной способностью узла. При открытии слота, он использует информацию о соседях, что бы случайно не выбрать уже используемый канал или временой слот.

## Ethernet protocol

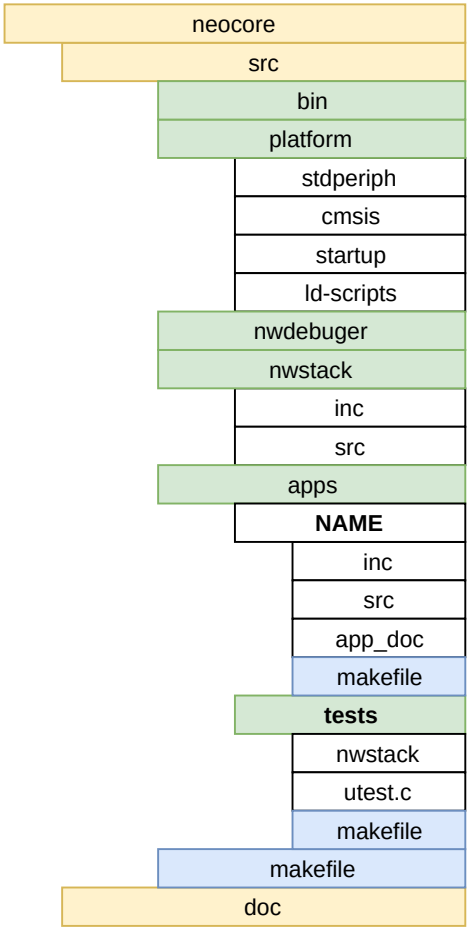
Протокол маршрутизации добавляет FItem PPDU_HEADER. Поле LEN и MPDU_LEN вычисляется из других частей FChain, PVERSION и NETID получат из NETIF. FCS1,2 заполняет нулями эти значения вычислит радиопередатчик налету. MPDU_TYPE определяется по заголовку первого элемента переданого FChain RT_Send(..).
Далее используя протокол соседей, ищет соседа с минимальным ETX. Из таблицы извлекает MAC адрес соседа, его рабочию частоту и номер тайм слота.
Все эти параметры и FChain передает LLC_AddTask(...)
Если соседей нет, то пакет уничтожается без уведомлений.
Возможна ситуация, что у нас будет несколько соседей с одинаковым ETX, к тому же у соседа может быть открыто несколько входящих подключений. В этом случаи выбор передачи осуществляется случаным образом.

## Neighbor protocol

Принимает ШВС сообщения-уведомления от соседей. Ведет список соседних узлов и их параметров. По мимо уведомлений может быть запрос соседей, тогда узел отзывается и посылает ШВС с информацией о себе. Рассылка информации производится через неравные интервалы по таймеру(раз 2-5 минуты). Узел считается устаревшим и его удаляем, если от него не поступало информации в течении 10 минут. Протокол управяет выбором ETX узла, основываясь на таблице. Запрос соседей происходит, при пустом списке, к примеру после обнаружения сети. Точка доступа таким же образом передает ETX=0 и свой ETX не вычисляет на основе соседей.

## Authorization protocol

Протокол отправляет точке доступа запрос на подключение с информацией об узле. Отправка запросов происходит периодически, пока нет подтверждения или отказа в подключении. Если было отправленно более 10 запросов и они все безответны, то начинаем поиск сети заново. Больная тема это получения ответа от точки доступа, как передавать пакеты узлам пока окончательно неясно.




Код расчитан на один тип платформы.
Юнит тесты используют свою функцию main.c (в utest.c).
Юнит тест описывается стандартной структурой данных и содержит вызываемую функцию. Структура данных подготавливается каждым тест-модулем.
Тестирование расчитано на использование отладчика gdb.
Основная функция main распологается в пользовательском приложении, пользователь сам запускает сетевой стек.
Перевод МК в спящий режим так же проблема пользовательского приложения.
Проект имеет один несколько сборочных makefile.
Сборочный файл имеет ключ сбори utest.

Цели сборки makefile:
build
clean
gdb (автоостанов в utest.c или main(), assert)
load (загрузка программы в МК)
utest

Опции сборки:
RELEASE - O2 (по умолчанию)
DEBUG - O0
LOG - разрешает логирование
SET_MAC 0x..... (Использовать предустановленый адрес)
ASSERT

USER_APP - имя приложения для сборки

Организация модулей программы:
Модули используют концепцию ООП. Обьект представлен структурой данных и методами работы с структурой. Методы класса реализованны в виде отдельных функций, используеющие в качестве аргумента указатель на обьект. Такой подход позволяет получить более чистый код и упростит отладку. Обьекты самостоятельно не инициализируют внутри себя вспомогательные обьекты, а получают их от конструктора(к примеру клас mac не создает класс radio. MAC получает указатель на обьект radio и использует его методы). Отсутсвие либо минимальное количество внешних связей обьекта позволят сделать хорошее покрытие кода тестами.

Исключения:
макрос stack_failure(char* msg)
Пишет в LOG сообщение.
Если DEBUG активен, код зацикливаеться, в противном слуаии МК перезапу

Отладка/логгер:
Для отладки проекта предназначен макспос ASSERT(condition, message) есл
проект блокирует сове исполнение и производит печать отладочной информ
Вывод данных в serialDebug с использованием printf. LOG_GROUP(message
При активном дефайне LOG поток  перенаправляется в serialDebug. Примен
значения переменных. Макрос LOG добавляет имя файла и номер строки. G
примеру LOG_MAC это сообщения уровня стека mac. Каждую группу логов м

Юнит тесты:
Тесты по группам распологаются в файлах с соответсвующими именами и предоставляют метод NAME_testSuit(testSuit* TS, uint8_t size). testSuit содержит перечень указателей на функции-тесты, общее количество тестов size, имена тестов. typedef  bool (*utestFN)(char*) указатель на тест-функцию, результатом работы являеться true/fase (наличие ошибки теста) и сообщение с причиной ошибки.
Главная программа включает заголовки всех тестовых модулей и собирет массив, utestRunner используя метод NAME_testSuit(..) извлекает тесты и прогоняет их. Если тест завалился, управление переходит в процедуру testFailed(char* msg). Вывод ошибок осуществляется с помощью serialDebug.

Аппаратная отладка(SIGNAL_TRACKER):

ускается.

ли условие истинно, то
нации.
).
нение printf позволяет выводить
GROUP имя группы лога. К
можно включить или выключить.

Аппаратная отладка(SIGNAL TRACKER).
Основа аппаратной отладки - переключение сигнальных гпио и анализ осци
макрос SIG_SET_NAME, SIG_PULSE_NAME и SIG_CLR_NAME устанавлива
Макрос обеспечивает переключение порта подстановкой команды в код, жел
помощью прямого обращения к регистру. NAME имя контролируемого проце
включать. Пример SIG_SET_SFD_DETECTED.  SIG_PULSE_NAME - создает
В файле с макросами SIG_SET_SFD_DETECTED определен
как макрос SIG1_HIGH, который переводит вывод в ВУ. ifdef SIG_SET_SFD_
Для активации отслеживания сигналов достаточно определить SIG_SET_SF

SIGNAL TRACKER, DEBUGER(LOGGER, ASSERT) являются
частью стека протоколов. Все макросы используют слабо связанную
функцию void serialDebugPutChar(char sym).
Тела макросов SIGx_HIGH, SIGx_LOW, SIGx_PULSE, определяются
пользователем в зависимости от платформы
исполнения. Желательно ассемблерная вставка.

лографом.
ают или сбрасываю линию SIG.
лательно это делать с
есса. Тесты возможно отключать и
т короткий импульс

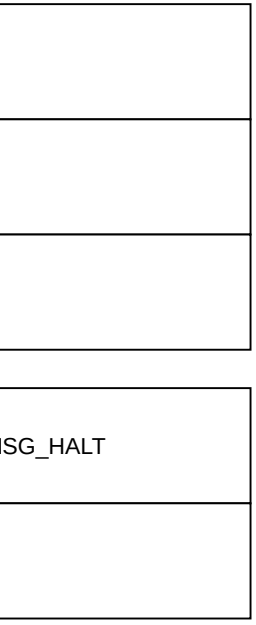
DETECTED SIG1_HIGH elseif ;.
FD_DETECTED.

| | | |
|---|---|---|
| LOG(level,...) | Макрос вывода отладочных сообщений. level - флаги и уровень логирования. Второй аргумент - стандартные аргументы printf | LOG(MSG_ON \| MSG_ALARM \| MSG_TRACE, "Value %d", 5) |
| ASSERT_HALT(condition, ...) | Макрос проверки условий. Если условие ложно, производится вывод сообщения ... и остановка работы программы. | ASSERT_HALT( x == Z , "X not equal %d", Z) |
| ASSERT(condition, ...) | Макрос проверки условий. Если условие ложно, производится вывод сообщения ... , работа программы не прерывается | ASSERT( x == Z , "X not equal %d", Z) |

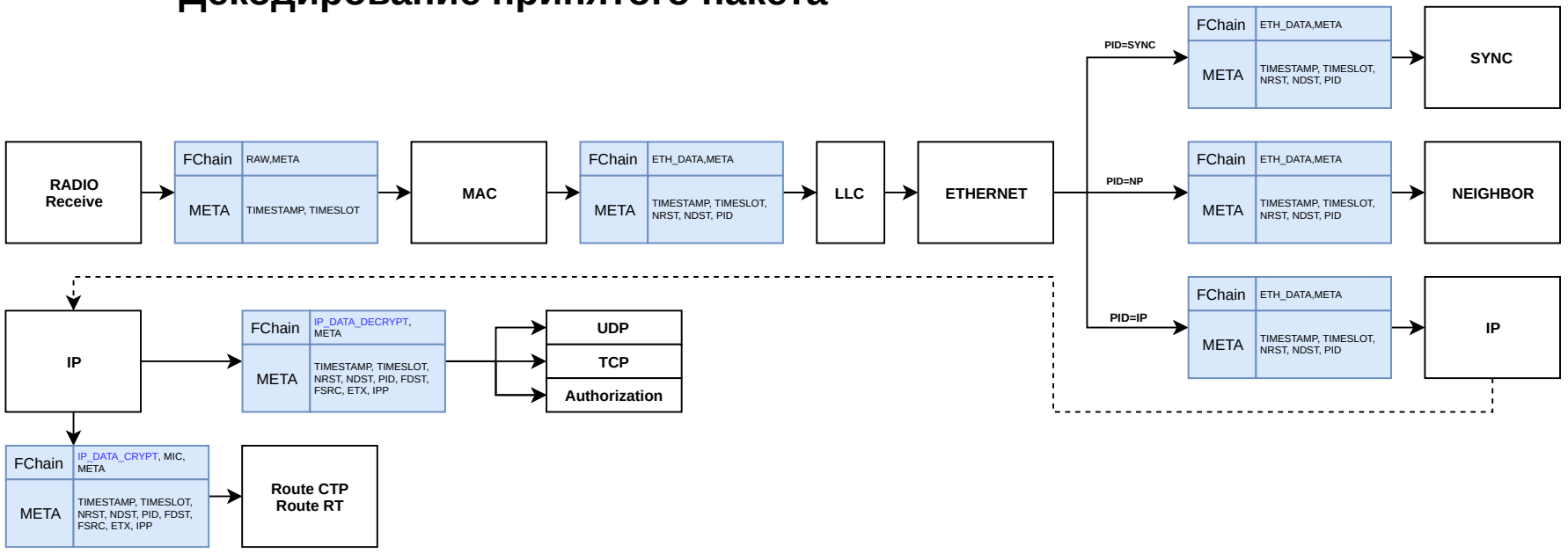
| | |
|---|---|
| __attribute__((weak)) void STACK_FAILURE(char* msg) {while(1);} | Функция вызывается при ASSERT_HALT и LOG с флагом M |
| void nwDebugerInit(void); | Настраивает аппаратуру системы логирования. |

| **Уровни логирования** |
|---|
| MSG_INFO<br>MSG_WARNING<br>MSG_ALARM |

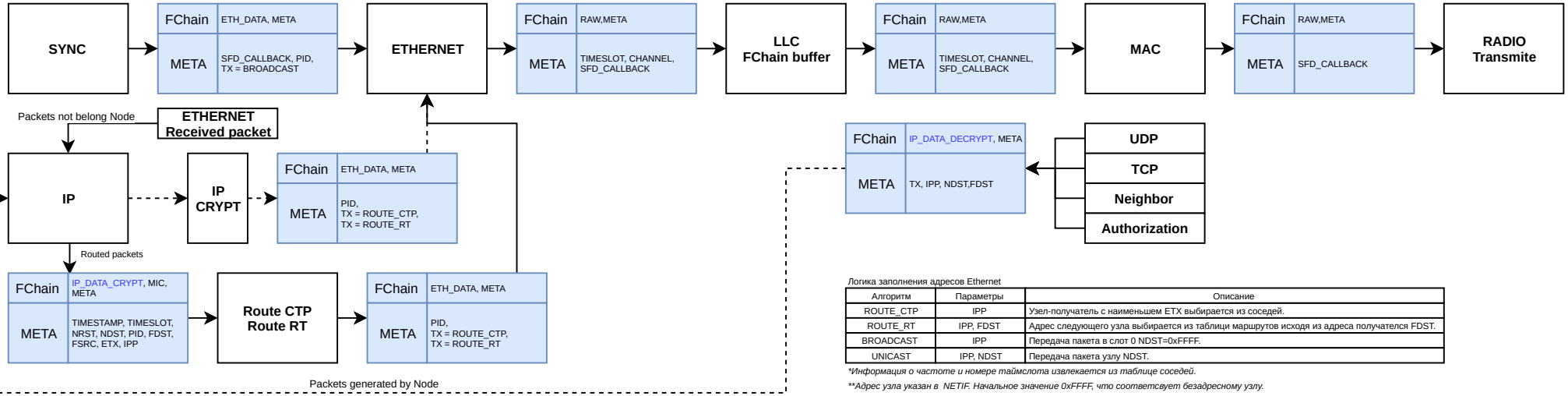
| **Флаги логирования** |
|---|
| MSG_TRACE<br>MSG_STATE<br>MSG_FRESH<br>MSG_HALT<br>MSG_ON<br>MSG_OFF |


ISG_HALT

# Декодирование принятого пакета




# Передача пакета




Логика заполнения адресов Ethernet

| Алгоритм | Параметры | Описание |
|---|---|---|
| ROUTE_CTP | IPP | Узел-получатель с наименьшем ETX выбирается из соседей. |
| ROUTE_RT | IPP, FDST | Адрес следующего узла выбирается из таблици маршрутов исходя из адреса получается FDST. |
| BROADCAST | IPP | Передача пакета в слот 0 NDST=0xFFFF. |
| UNICAST | IPP, NDST | Передача пакета узлу NDST. |

**Информация о частоте и номере таймслота извлекается из таблице соседей.*

***Адрес узла указан в NETIF. Начальное значение 0xFFFF, что соответсвует безадресному узлу.*

****Поле ACK Ethernet заполняет автоматически.*

Пояснение работы IP.
Пакеты для передачи можно разделить на 3 группы: пакеты для нас, пакеты от ТД к узлу Z и пакеты от узла Z к ТД. Узел Z - некий узел в сети, которой использует нас как ретранслятор. IP уровень разбирает заголовки пакетов, но не расшифровывает содержимое. На основе полей FDST, FSRC определяет необходимое действие над пакетом. В случаи передачи данных, наши пакеты шифруются и упаковываются в ETH_DATA, алгоритм передачи определяется вышестоящим уровнем. Если же пакет требуется ретранслировать, то данные переупаковаются и указывается необходимый алгоритм передачи ETHERNET.