

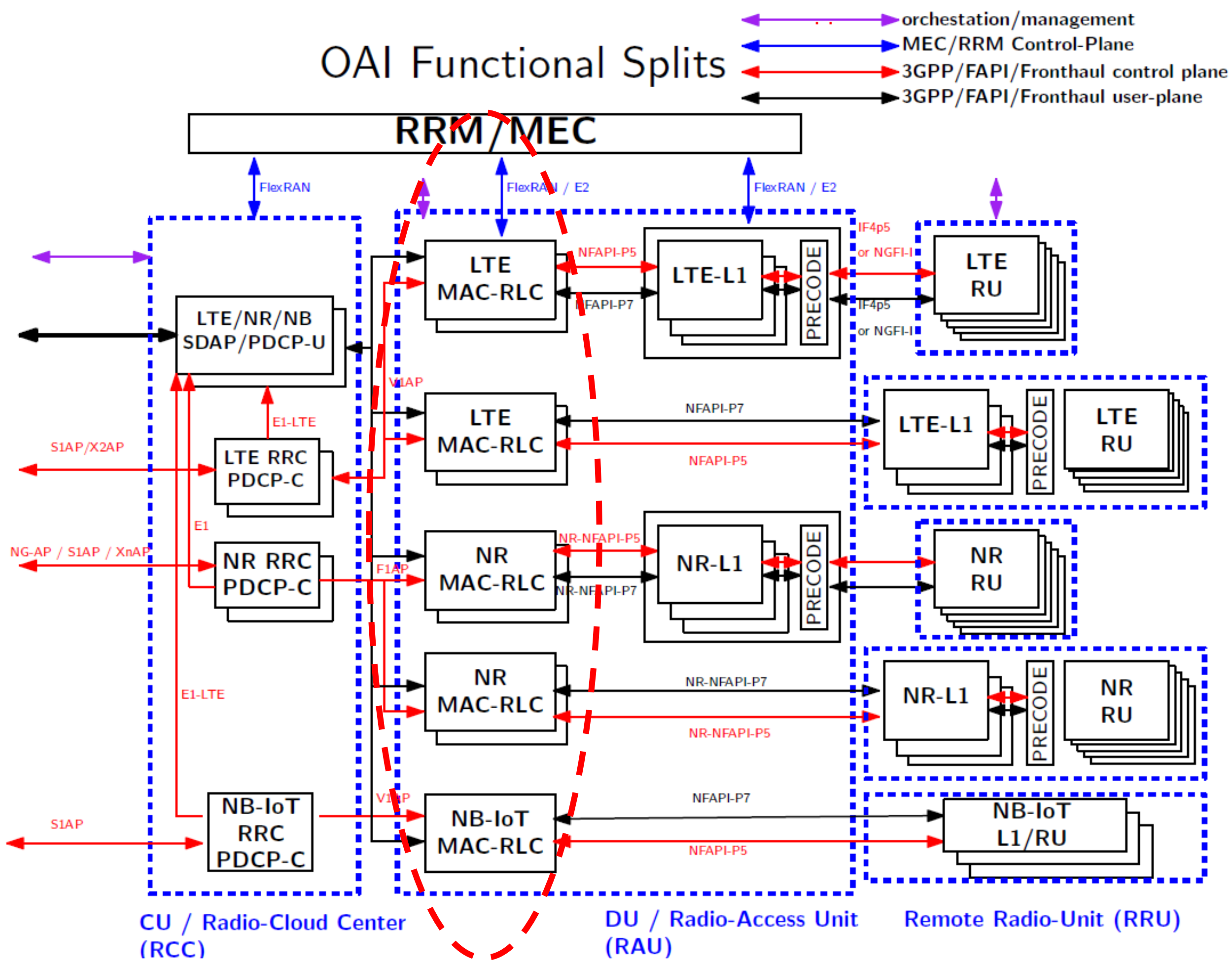
OAI Layer 2 Protocol Stack

11/12/2018

Development of Scalable MAC Scheduling in OAI

- **Objectives**
  - Complete overhaul of current OAI MAC implementation
  - Integration of NFAPI interfaces for MAC scheduling
  - Harmonized scheduler for LTE and BL/CE UEs
  - Scalable to hundreds of UEs
    - Software stability
    - Testing procedures
  - “extractable” preprocessor to customize scheduling policy

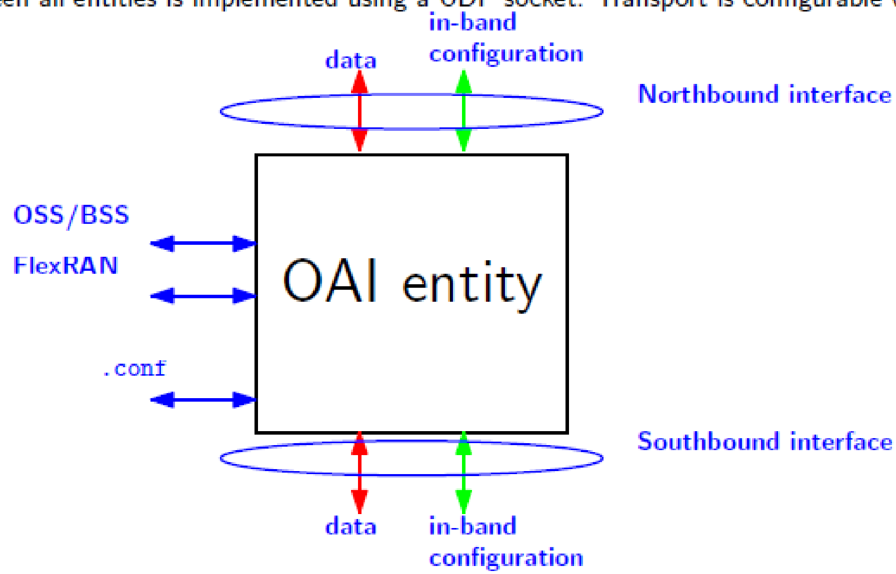
OAI CRAN Architecture



Subject of discussion

OAI Functional Splits

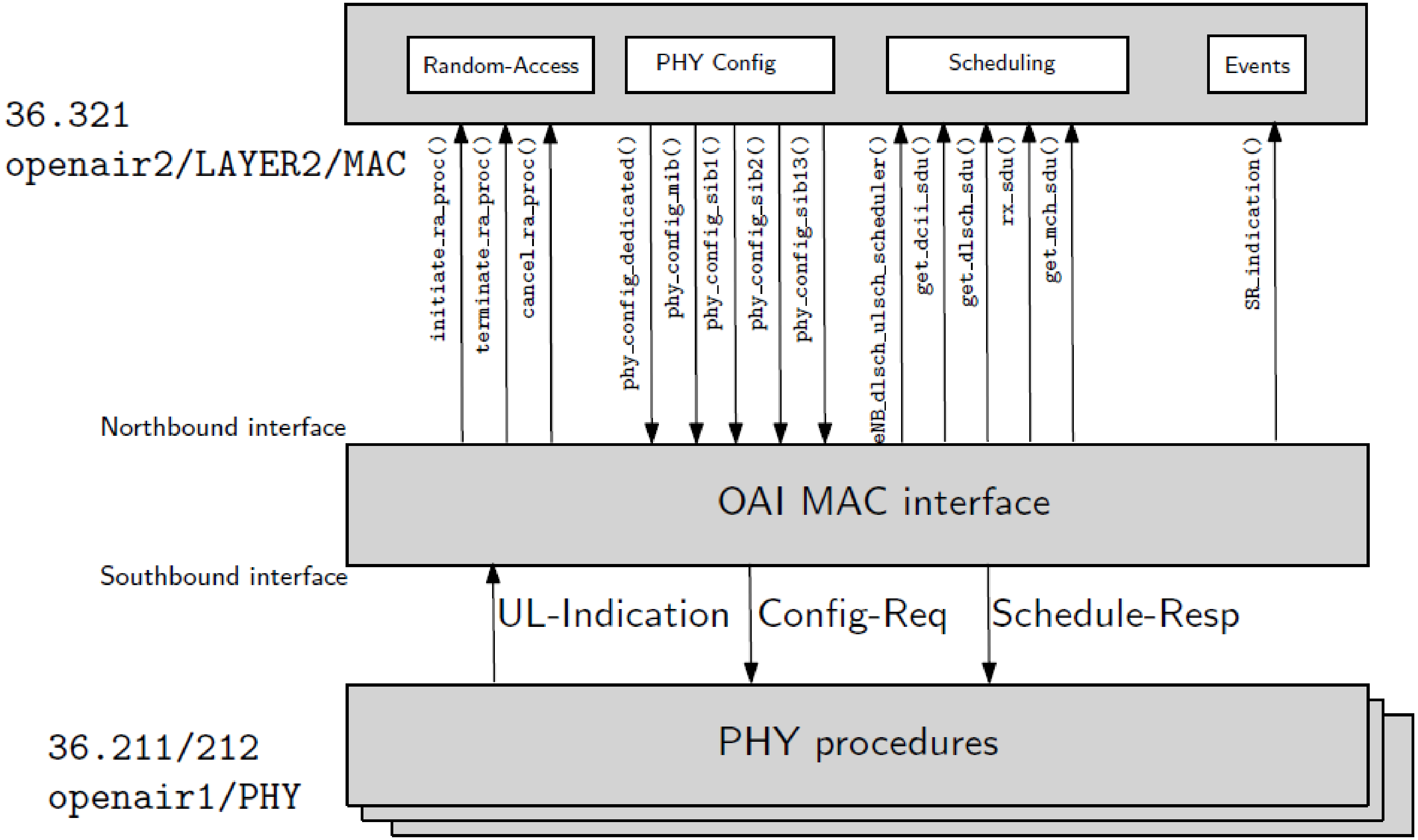
- OAI currently implements the following entities in openairinterface5g
  - LTE-MODEM (eNB 36.211 OFDM modulation/demodulation)
  - LTE-L1 (eNB 36.211/212/213)
  - LTE-MACRLC (eNB 36.321/322)
  - LTE-PDCP (eNB PDCP/GTPU 36.323)
  - LTE-RRC (eNB RRC/SCTP 36.331)
- Each entity comprises
  - a northbound interface (backhaul/midhaul/fronthaul and configuration)
  - a southbound interface (midaul/fronthaul and configuration)
  - one or two management interfaces
  - Three computing nodes
    - **Radio Cloud Center (RCC)** : multiple RRC/PDCP entities
    - **Radio-Access Unit (RAU)**: multiple MACRLC entities with medium-latency midhaul and L1 entities with low-latency fronthaul.
    - **Remote Radio-Unit (RRU)**: Equipment at radio site. Varying degrees of processing elements depending on fronthaul/midhaul interface.
- Each entity has a configuration which is a local file or received via the management interface
- default interface between all entities is implemented using a UDP socket. Transport is configurable via a dynamically-loadable networking device



OAI MAC-RLC Entity

- **One of the modules in OAI CRAN module library, in the RAU entity**
- **Two current variants**
  - LTE/eLTE MAC/RLC
  - NB-IoT MAC/RLC
- **LTE/eLTE covers legacy LTE (up to Rel11) and eMTC (LTE-M)**
- **Internal Components**
  - Core-MAC: basic LTE procedures (HARQ, BCCH, Random-Access). Normally not customizable
  - Preprocessor : “customizable” scheduling function. Can be loaded dynamically from standalone implementation or use OAI default
  - PHY Interface module (NFAPI-compatible)

nFAPI interfaces

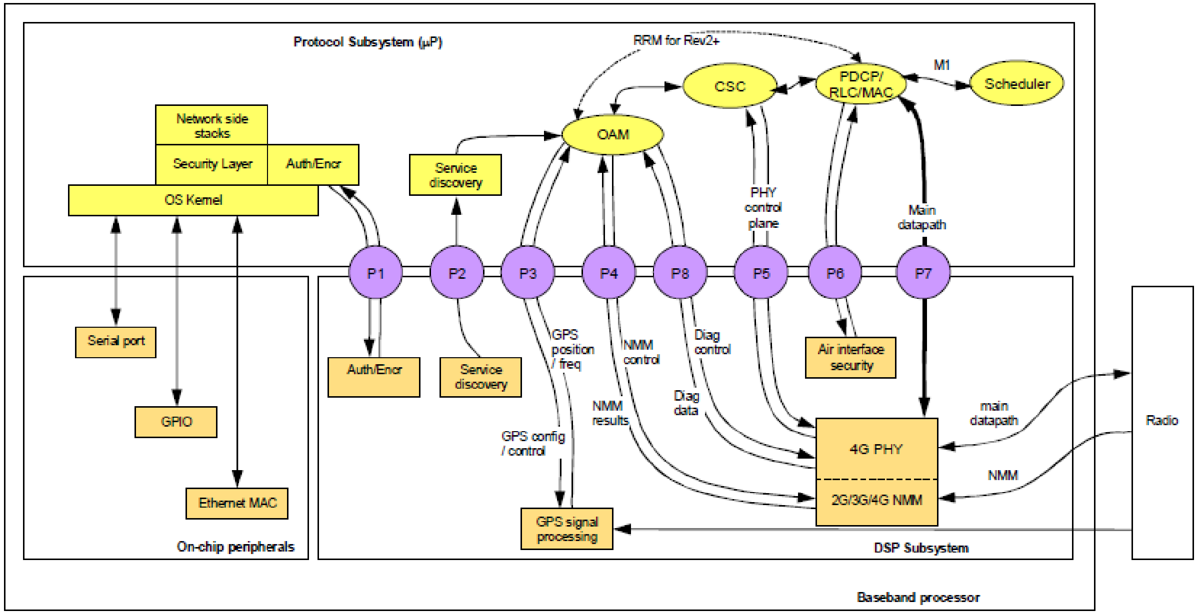


FAPI

The FAPI is defined via a reference architecture shown below, which is generic to 3G or 4G small cells. Several APIs are defined, as follows:

- P1 – the security co-processor interface
- P2 – the service discovery interface
- P3 – the GPS interface
- P4 – the network listen results interface
- P5 – the PHY mode control interface
- P6 – the ciphering coprocessor interface
- P7 – the main data path interface
- M1 – the scheduler interface

This document defines both the P5 and P7 interfaces for 4G cells.

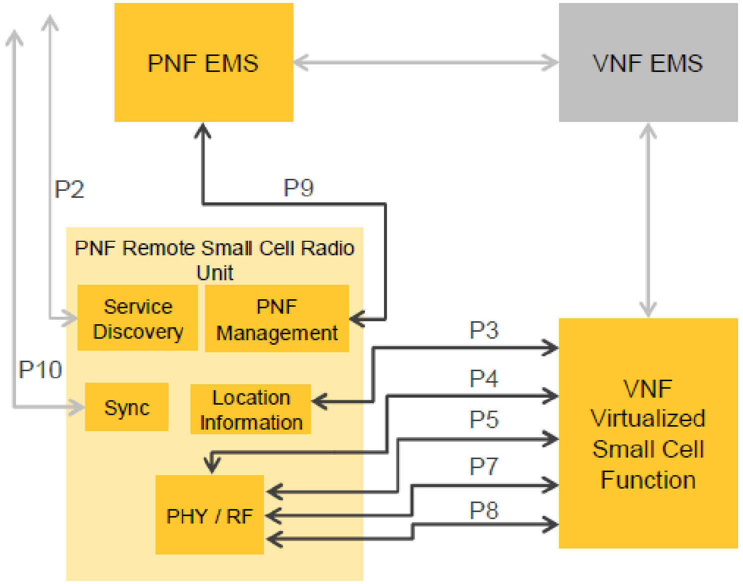


nFAPI

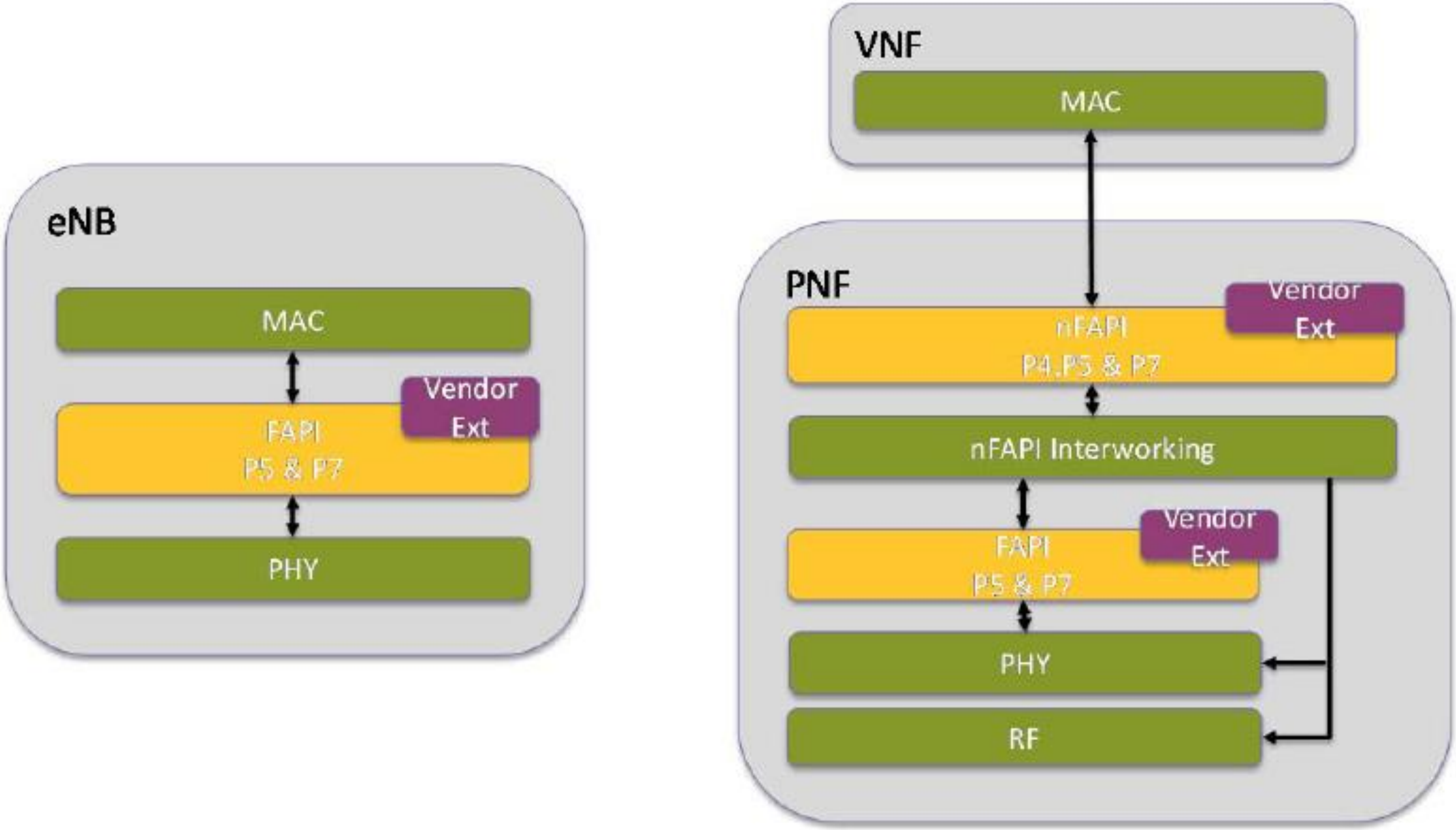
The nFAPI is defined via a reference architecture shown below, which is specific to 4G small cells. Several APIs are defined, as follows:

- P2 – the service discovery interface
- P3 – the positioning interface
- P4 – the network monitor mode interface
- P5 – the PHY mode control interface
- P7 – the main data path interface
- P8 – the PHY diagnostics interface
- P9 – the OAM interface
- P10 – the synchronization, frequency, phase and time

This document defines the P4, P5 and P7 interfaces.



Relationship between FAPI and nFAPI



OAI FAPI interface

- The PHY end uses three basic messages
  - CONFIG_REQ: this provides the cell configuration and UE-specific configuration to the PHY instances. This comprises the following FAPI P5/P7 messages
    1. CONFIG.request
    2. UE_CONFIG.request (**not used in OAI PHY)
  - UL_INDICATION This is an uplink indication that sends all UL information received in one TTI, including PRACH, if available. It also provides the subframe indication for the DL scheduler. It maps to the following FAPI P7 messages
    1. SUBFRAME.indication
    2. HARQ.indication
    3. CRC.indication
    4. RX_ULSCH.indication
    5. RX_SR.indication
    6. RX_CQI.indication
    7. RACH.indication
    8. SRS.indication
  - SCHEDULE_REQUEST This message contains the scheduling response information and comprises the following FAPI P7 messages
    1. DL_CONFIG.request
    2. UL_CONFIG.request
    3. TX.request
    4. HI_DCI0.request
- The module is registered both by PHY and MAC and can implement different types of transport (NFAPI, function call, FAPI over UDP, etc.). During registration, fuction pointers for the different messages are provided for the module to interact with either PHY or MAC or both if they are executing in the same machine. Note that for a networked implementation (e.g. NFAPI), there are north and south components running in different machines.

FAPI/nFAPI

- **Monolithic configuration (FAPI only)**
  - FAPI messages with passthrough from L1/L2 (local_mac,local_l1)
    - Direct function call interface (MAC runs in L1 thread) for UL_INDICATION/SCHEDULE_RESPONSE/CONFIG_REQUEST
    - eNB configuration file snippet:

```
MACRLCs = (
        {
        num_cc = 1;
        tr_s_preference = "local_L1";
        tr_n_preference = "local_RRC";
        }
);
L1s = (
        {
        num_cc = 1;
        tr_n_preference = "local_mac";
        }
);
```

FAPI/nFAPI

- Integration with open-nFAPI SCTP/UDP transport for P5/P7
  - `https://github.com/cisco/open-nFAPI`
  - `openairinterface5g/nfapi/nfapi_pnf.c`
  - `openairinterface5g/nfapi/nfapi_vnf.c`
  - PNF Configuration

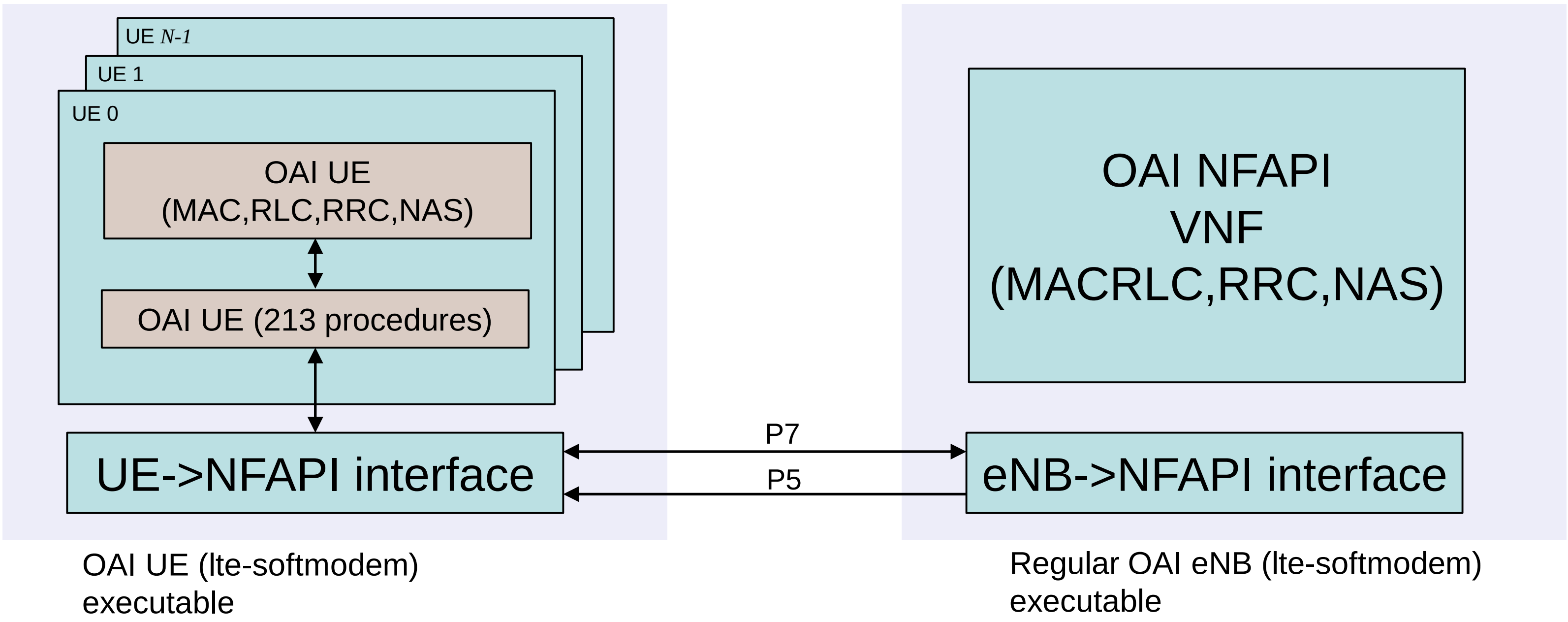
```
L1s = (
{
num_cc = 1;
tr_n_preference = "nfapi";
local_n_if_name  = "eno1";
remote_n_address = "192.168.1.28";
                            local_n_address   = "192.168.1.74";
                            local_n_portc     = 50000;
                            remote_n_portc    = 50001;
                            local_n_portd     = 50010;
                            remote_n_portd    = 50011;
}
);
```

  - VNF Configuration

```
MACRLCs = (
  {
  num_cc = 1;
local_s_if_name  = "eno1";
remote_s_address = "192.168.1.74";
                 local_s_address   = "192.168.1.28";
                 local_s_portc     = 50001;
                 remote_s_portc    = 50000;
                 local_s_portd     = 50011;
                 remote_s_portd    = 50010;
  tr_s_preference = "nfapi";
  tr_n_preference = "local_RRC";
      }
);
```

UE NFAPI emulator

- **UE stub**
  - Development of an nFAPI compatible UE stub which can be used to test/simulate the MACRLC and higher-layer protocols with which it is interconnected
- **Evolution of oaisim environment to allow for L2 interconnection with open-nfapi**
- **A UE executable which puts many UE instances under the same Linux process (like oaisim)**



- **UE-nFAPI interface**
  - Implements interconnection between OAI UE transport and physical channels with NFAPI messages from eNB TX and RX
  - Aggregates information from/to multiple UEs in the common executable
  - Can later include physical layer impairment modelling to stimulate eNB and UE protocol stacks with more realistic behaviour
- **Extensions**
  - UEs on separate machines (ethernet interconnections)
    - Need to aggregate NFAPI information in machine that uses NFAPI with eNodeB
  - D2D links
    - development and testing of Rel 14 Sidelink procedures

Northbound interface from FAPI/NFAPI

- **the following functions are provided**
  - initiate_ra_proc(): An event to trigger the Random-Access Procedure. It indicates a received PRACH signal from one or more UEs. This event provides amplitude, timing and preamble index for each received PRACH preamble.
  - SR_indication(): An event triggered upon reception of Scheduling Request message from a particular UE. This event provides UL CQI information in addition to a UE RNTI.
  - cqi_indication(): An event triggered upon reception of Channel Status Information (CSI) comprising CQI,RI,PMI from a particular UE. This event provides UL CQI in addition to a UE RNTI and the CSI payload.
  - harq_indication(): An event triggered upon reception of HARQ Information received either on PUCCH or PUSCH. The event provides UL CQI information in addition to a UE RNTI and the HARQ ACK/NAK payload.
  - rx_sdu(): An event triggered upon reception of an UL PDU on PUSCH. This event provides UL CQI information corresponding to the PUSCH channel on which the data was received. In the event of a CRC mismatch because of channel decoding error, a NULL SDU is indicated to the MACRLC entity.
  - srs_indication(): An event triggered upon reception of an UL SRS packet. This even provides UL CQI information on SRS resources (FDD) and 8-bit quantized channel estimates (TDD) depending cell duplexing configuration.
  - eNB_dlsch_ulsch_scheduler(): This is an entry-level procedure to invoke the scheduling process in the MAC layer. Upon completion the Schedule-Response message is returned with scheduling information for DL and UL to trigger PHY transmission and reception procedures.

Example UL flow from PHY to MAC (RACH indication)

- **Step 1: Program FAPI indication : [Piece of code from `openair1/SCHED/phy_procedures_lte_eNB.c:prach_procedures()`, called from PRACH processing thread]**

```
 if ((eNB->prach_energy_counter == 100) &&
          (max_preamble_energy[0] > eNB->measurements.prach_I0+100)) {
    pthread_mutex_lock(&eNB->UL_INFO_mutex);
    eNB->UL_INFO.rach_ind.number_of_preambles                  = 1;
    eNB->UL_INFO.rach_ind.preamble_list                        = eNB->preamble_list;

    eNB->preamble_list[0].preamble_rel8.timing_advance         = max_preamble_delay[0];
    eNB->preamble_list[0].preamble_rel8.preamble               = max_preamble[0];
    eNB->preamble_list[0].preamble_rel8.rnti                   = 1+subframe;  // note: fid is
implicitly 0 here
    eNB->preamble_list[0].preamble_rel13.rach_resource_type    = 0;
    eNB->preamble_list[0].instance_length                      = 0;
    pthread_mutex_unlock(&eNB->UL_INFO_mutex);
  } // max_preamble_energy > prach_I0 + 100
```

Example UL flow from PHY to MAC (RACH indication)

- **Step 2: Call Indication from main PHY thread [Piece of code from targets/RT_USER/lte-enb.c:rxtx()**

```
static inline int rxtx(PHY_VARS_eNB *eNB,eNB_rxtx_proc_t *proc, char *thread_name) {
     if (eNB->RU_list[0]->function < NGFI_RAU_IF4p5) {
         wakeup_prach_eNB(eNB,NULL,proc->frame_rx,proc->subframe_rx);
     #ifdef Rel14
         wakeup_prach_eNB_br(eNB,NULL,proc->frame_rx,proc->subframe_rx);
     #endif
       }
       // UE-specific RX processing for subframe n
       phy_procedures_eNB_uespec_RX(eNB, proc, no_relay );
       pthread_mutex_lock(&eNB->UL_INFO_mutex);
       eNB->UL_INFO.frame     = proc->frame_rx;
       eNB->UL_INFO.subframe  = proc->subframe_rx;
       eNB->UL_INFO.module_id = eNB->Mod_id;
       eNB->UL_INFO.CC_id     = eNB->CC_id;
       eNB->if_inst->UL_indication(&eNB->UL_INFO);
       pthread_mutex_unlock(&eNB->UL_INFO_mutex);

       // TX processing for subframe n+4
       phy_procedures_eNB_TX(eNB, proc, no_relay, NULL, 1);

       if (release_thread(&proc->mutex_rxtx,&proc->instance_cnt_rxtx,thread_name)<0) return(-1);

       stop_meas( &softmodem_stats_rxtx_sf );

       return(0);
     }
```

Example UL flow from PHY to MAC (RACH indication)

- **Step3: Indication to MAC [piece of code from openair2/PHY_INTERFACE/IF_Module.c**

```
void UL_indication(UL_IND_t *UL_info)
{
  module_id_t  module_id   = UL_info->module_id;
  int          CC_id       = UL_info->CC_id;
  Sched_Rsp_t  *sched_info = &Sched_INFO[module_id][CC_id];
  IF_Module_t  *ifi        = if_inst[module_id];
  eNB_MAC_INST *mac        = RC.mac[module_id];
  if (ifi->CC_mask==0) {
    ifi->current_frame    = UL_info->frame;
    ifi->current_subframe = UL_info->subframe;
  }
  else {
    AssertFatal(UL_info->frame != ifi->current_frame,"CC_mask %x is not full and frame has changed\n",ifi->CC_mask);
    AssertFatal(UL_info->subframe != ifi->current_subframe,"CC_mask %x is not full and subframe has changed\n",ifi->CC_mask);
  }
  ifi->CC_mask |= (1<<CC_id);
 // clear DL/UL info for new scheduling round
  clear_nfapi_information(RC.mac[module_id],CC_id,
                          UL_info->frame,UL_info->subframe);
  handle_rach(UL_info);
  handle_sr(UL_info);
  handle_cqi(UL_info);
  handle_harq(UL_info);
  // clear HI prior to hanling ULSCH
  mac->HI_DCI0_req[CC_id].hi_dci0_request_body.number_of_hi                     = 0;
  handle_ulsch(UL_info);
  if (ifi->CC_mask == ((1<<MAX_NUM_CCs)-1)) {
    eNB_dlsch_ulsch_scheduler(module_id,
                              (UL_info->frame+((UL_info->subframe>5)?1:0)) % 1024,
                              (UL_info->subframe+4)%10);
    ifi->CC_mask            = 0;
    sched_info->module_id   = module_id;
    sched_info->CC_id       = CC_id;
    sched_info->frame       = (UL_info->frame + ((UL_info->subframe>5) ? 1 : 0)) % 1024;
    sched_info->subframe    = (UL_info->subframe+4)%10;
    sched_info->DL_req      = &mac->DL_req[CC_id];
    sched_info->HI_DCI0_req = &mac->HI_DCI0_req[CC_id];
    if ((mac->common_channels[CC_id].tdd_Config==NULL) ||
        (is_UL_sf(&mac->common_channels[CC_id],(sched_info->subframe+4)%10)>0))
      sched_info->UL_req      = &mac->UL_req[CC_id];
    else
      sched_info->UL_req      = NULL;

    sched_info->TX_req      = &mac->TX_req[CC_id];

    ifi->schedule_response(sched_info);

    LOG_D(PHY,"Schedule_response: frame %d, subframe %d (dl_pdus %d / %p)\n",sched_info->frame,sched_info->subframe,sched_info->DL_req->dl_config_request_body.number_pdu,
          &sched_info->DL_req->dl_config_request_body.number_pdu);
  }
}
```

Example UL flow from PHY to MAC (RACH indication)

- **Step 4: Call to MAC RACH handler**

```
void handle_rach(UL_IND_t *UL_info) {
  int i;
  if (UL_info->rach_ind.number_of_preambles>0) {
    AssertFatal(UL_info->rach_ind.number_of_preambles==1,"More than 1 preamble not supported\n");
    UL_info->rach_ind.number_of_preambles=0;
    LOG_D(MAC,"Frame %d, Subframe %d Calling initiate_ra_proc\n",UL_info->frame,UL_info->subframe);
    initiate_ra_proc(UL_info->module_id,
                     UL_info->CC_id,
                     UL_info->frame,
                     UL_info->subframe,
                     UL_info->rach_ind.preamble_list[0].preamble_rel8.preamble,
                     UL_info->rach_ind.preamble_list[0].preamble_rel8.timing_advance,
                     UL_info->rach_ind.preamble_list[0].preamble_rel8.rnti
#ifdef Rel14
                     ,0
#endif
                     );
  }
}
```

Example DL Flow (DL Config Request)

- **Step 1: Call top-level scheduling function [piece of code from openair2/LAYER2/MAC/eNB_scheduler.c]**

```
if ((subframeP == 0) && (frameP & 3) == 0)        schedule_mib(module_idP, frameP, subframeP);
// This schedules SI for legacy LTE and eMTC starting in subframeP
schedule_SI(module_idP, frameP, subframeP);
// This schedules Random-Access for legacy LTE and eMTC starting in subframeP
schedule_RA(module_idP, frameP, subframeP);
// copy previously scheduled UL resources (ULSCH + HARQ)
copy_ulreq(module_idP, frameP, subframeP);
// This schedules SRS in subframeP
schedule_SRS(module_idP, frameP, subframeP);
// This schedules ULSCH in subframeP (dci0)
schedule_ulsch(module_idP, frameP, subframeP);
// This schedules UCI_SR in subframeP
schedule_SR(module_idP, frameP, subframeP);
// This schedules UCI_CSI in subframeP
schedule_CSI(module_idP, frameP, subframeP);
schedule_ue_spec(module_idP, frameP, subframeP, mbsfn_status);

// Allocate CCEs for good after scheduling is done

for (CC_id = 0; CC_id < MAX_NUM_CCs; CC_id++)
    allocate_CCEs(module_idP, CC_id, subframeP, 0);
```

Example DL Flow (DL Config Request)

- **Step 2: Request data from higher-layer and fill FAPI data-structures [piece of code from openair2/LAYER2/MAC/eNB_scheduler_bch.c]**

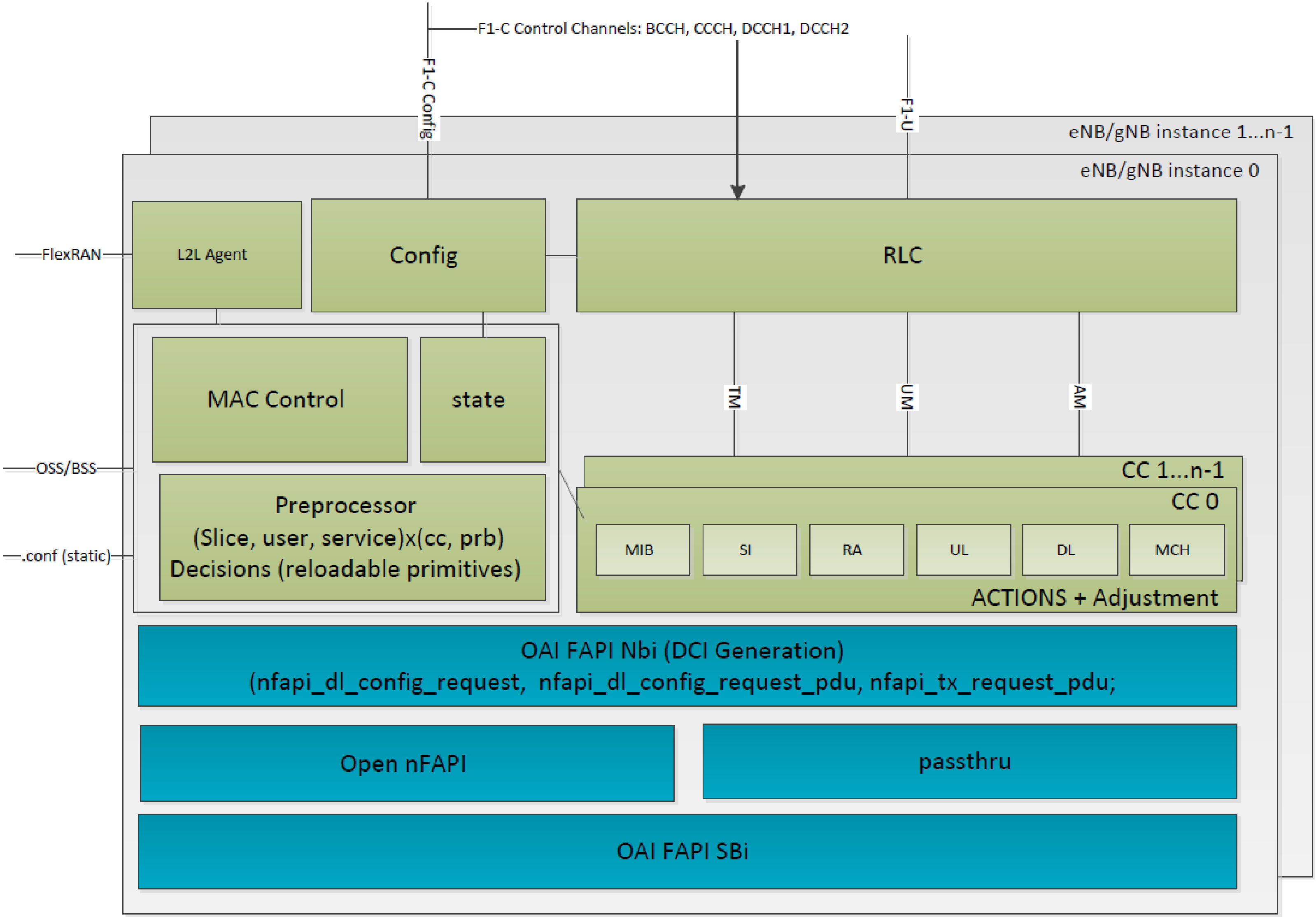
```
schedule_mib(module_id_t module_idP, frame_t frameP, sub_frame_t subframeP)
{
    eNB_MAC_INST *eNB = RC.mac[module_idP];
    COMMON_channels_t *cc;
    nfapi_dl_config_request_pdu_t *dl_config_pdu;
    nfapi_tx_request_pdu_t *TX_req;
    int mib_sdu_length;
    int CC_id;
    nfapi_dl_config_request_body_t *dl_req;
    for (CC_id = 0; CC_id < MAX_NUM_CCs; CC_id++) {
        dl_req = &eNB->DL_req[CC_id].dl_config_request_body;
        cc = &eNB->common_channels[CC_id];
        mib_sdu_length = mac_rrc_data_req(module_idP, CC_id, frameP, MIBCH, 1, &cc->MIB_pdu.payload[0], 1, module_idP, 0);      // not used in this
case
        if (mib_sdu_length > 0) {
            dl_config_pdu =&dl_req->dl_config_pdu_list[dl_req->number_pdu];
            memset((void *) dl_config_pdu, 0,sizeof(nfapi_dl_config_request_pdu_t));
            dl_config_pdu->pdu_type = NFAPI_DL_CONFIG_BCH_PDU_TYPE, dl_config_pdu->pdu_size =2 + sizeof(nfapi_dl_config_bch_pdu);
            dl_config_pdu->bch_pdu.bch_pdu_rel8.length = mib_sdu_length;
            dl_config_pdu->bch_pdu.bch_pdu_rel8.pdu_index = eNB->pdu_index[CC_id];
            dl_config_pdu->bch_pdu.bch_pdu_rel8.transmission_power = 6000;
            dl_req->number_pdu++;
            TX_req =&eNB->TX_req[CC_id].tx_request_body.tx_pdu_list[eNB->TX_req[CC_id].tx_request_body.number_of_pdus];
            TX_req->pdu_length = 3;
            TX_req->pdu_index = eNB->pdu_index[CC_id]++;
            TX_req->num_segments = 1;
            TX_req->segments[0].segment_length = 0;
            TX_req->segments[0].segment_data = cc[CC_id].MIB_pdu.payload;
            eNB->TX_req[CC_id].tx_request_body.number_of_pdus++;
        }
    }
```

Example DL Flow (DL Config Request)

- **Program L1 to according to MAC configuration [piece of code from** `openair1/SCHED/fapi_l1.c`**]**

```
void schedule_response(Sched_Rsp_t *Sched_INFO)
{
…
 for (i=0;i<number_dl_pdu;i++) {
    dl_config_pdu = &DL_req->dl_config_request_body.dl_config_pdu_list[i];
switch (dl_config_pdu->pdu_type) {
    case NFAPI_DL_CONFIG_DCI_DL_PDU_TYPE:
      handle_nfapi_dci_dl_pdu(eNB,proc,dl_config_pdu);
      eNB->pdcch_vars[subframe&1].num_dci++;
      break;
    case NFAPI_DL_CONFIG_BCH_PDU_TYPE:
      eNB->pbch_configured=1;
      handle_nfapi_bch_pdu(eNB,proc,dl_config_pdu,
      TX_req->tx_request_body.tx_pdu_list[dl_config_pdu-
>bch_pdu.bch_pdu_rel8.pdu_index].segments[0].segment_data);
      break;
….
}
```

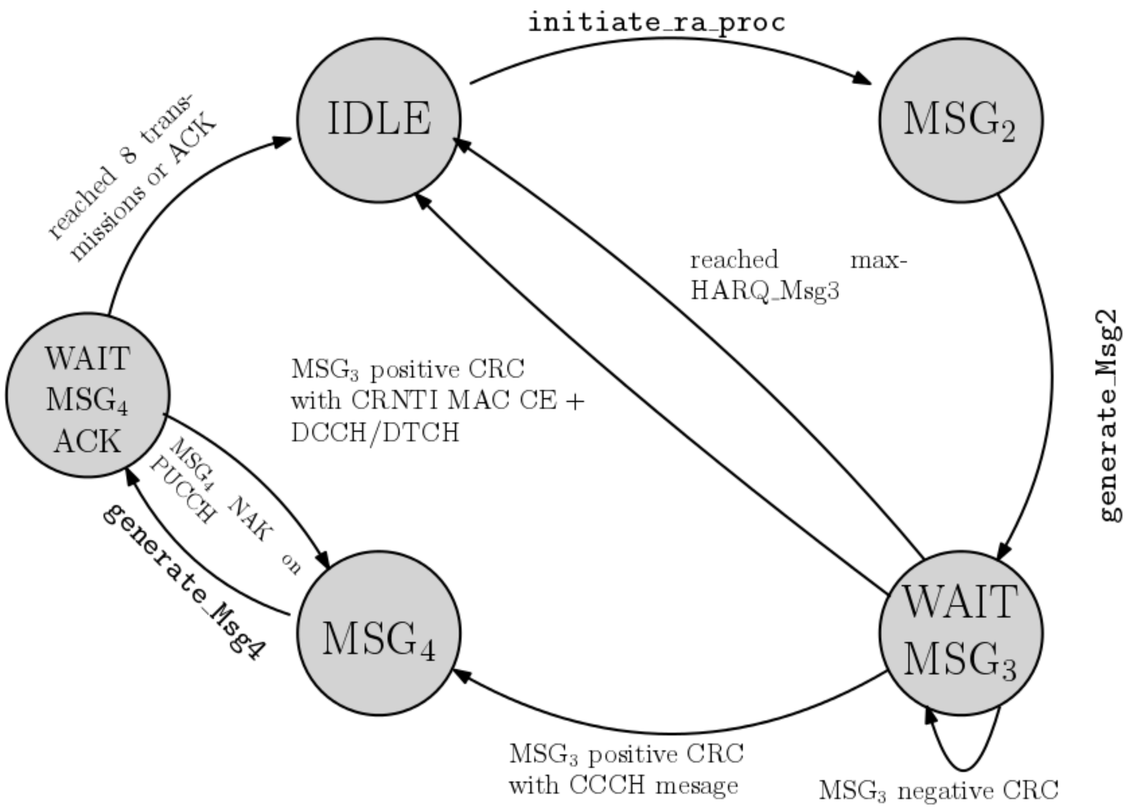
Overview of MACRLC module



Some detail (deterministic scheduling)

- **Configuration**
  - Configuration interface from RRC keeps RadioResourceConfigCommon and RadioResourceConfigDeidcated information elements locally and provides necessary parameters for FAPI P5 messages for PHY module
- **Order of Operations of LTE mechanisms (deterministic scheduler)**
  - SRS scheduling
    - Performs scheduling and programming of SRS for the subframe
    - Generation of UL_CONFIG.SRS_pdu for each UE transmitting SRS in subframe
  - SI-scheduling (`eNB_scheduler_bch.c`)
    - performs scheduling of system information for legacy LTE and eMTC broadcast messages
    - Generation of
      - DL_CONFIG.BCH_pdu
      - DL_CONFIG_DL_DCI_pdu (format1A DCI)
      - DL_CONFIG.DLSCH_pdu (SI),
      - TX_request (BCH)
      - TX_request (SI)
  - RA-scheduling (`eNB_scheduler_RA.c`)
    - Handles Random-access (Msg2/Msg3/Msg4) procedures
    - Generation of Msg2/3
      - DL_CONFIG.DCI_DL_pdu (format 1A RA_rnti)
      - DL_CONFIG.DLSCH_pdu (RAR)
      - TX_request (RAR)
      - UL_CONFIG.ULSCH_pdu (Msg3 config)
    - Generation of Msg4 and its retransmissions
      - DL_CONFIG.DCI_DL_pdu (format 1A t-crnti)
      - DL_CONFIG.DLSCH_pdu (Msg4 with RRCConnectionSetup piggyback)
      - TX_request (Msg4)
      - UL_CONFIG.UCI_HARQ_pdu (Msg4 ACK/NAK configuration)

RA Scheduling



Some detail (deterministic scheduling)

- **Order of operations (cont'd)**
  - ULSCH-scheduling (`eNB_scheduler_ulsch.c`)
    - Reception of SDUs from PHY (random-access and scheduled-access)
    - Preprocessor for ULSCH
      - Customizable scheduling function. Has inputs from UE status indicators (CQI/Buffer) and outputs target UE allocations (ordered list of UEs to serve and target bandwidths)
    - Final scheduling
      - PDCCH/ePDCCH/mPDCCH feasibility verification
      - Allocation of physical resources (mcs,resource blocks, power control commands)
  - SR scheduling
    - Happens after ULSCH scheduling if UE has no UL grant
    - Handling of PUCCH1 SR information (generation of UL_CONFIG messages for UCI_SR – augmented to SR_HARQ later if needed)
  - CQI scheduling
    - Handling of PUCCH2 CQI scheduling (generation of UL_CONFIG messages for UCI_CQI_PMI_RI)

Some Detail (deterministic scheduling)

- **Order of operations (cont'd)**
  - DLSCH-scheduling (`eNB_scheduler_dlsch.c`)
    - Preprocessor for DLSCH
      - Customizable scheduling function (detail following). Has inputs from RLC status indicators and output target UE allocations (ordered list of UEs to serve, target bandwdiths, precoding information)
    - Final scheduling
      - PDCCH/ePDCCH/mPDCCH feasibility verification (DL_CONFIG.DL_DCI)
      - Allocation of PRBS,precoding,mcs
      - Generation of
        - DL_CONFIG.DL_DCI_pdu
        - DL_CONFIG.DLSCH_pdu
        - TX_request
        - HARQ programming (`eNB_scheduler_primitives.c:program_dlsch_acknak()`)
          - If UL_CONFIG.ULSCH_pdu is present, augments to ULSCH_HARQ_pdu
          - If UL_CONFIG.ULSCH_CQI_RI is present, augments to ULSCH_CQI_HARQ_RI
          - If UL_CONFIG.UCI_SR_pdu is present, augments to UCI_SR_HARQ_pdu
          - If UL_CONFIG.UCI_CQI_RI_pdu is present, augments to UCI_CQI_HARQ_RI_pdu
          - If UL_CONFIG.UCI_SR_CQI_RI_pdu is present, augments to UCI_SR_CQI_HARQ_RI_pdu
          - else programs UL_CONFIG.UCI_HARQ_pdu
  - MCH scheduling (`eNB_scheduler_mch.c`)
    - Specific eMBMS scheduling

Generation of MAC-layer control elements

- **Timing advance (TA) control elements are inserted into the DLSCH SDU when a non-zero timing advance is to be conveyed to a particular UE and when the UE-specific TA timer (`ue_sched_info.ta_timer`) is expired.**
- **only control element for UE-specific DLSCH.**
- **Contention resolution is currently handled by the RA procedure state machine.**

Event handling

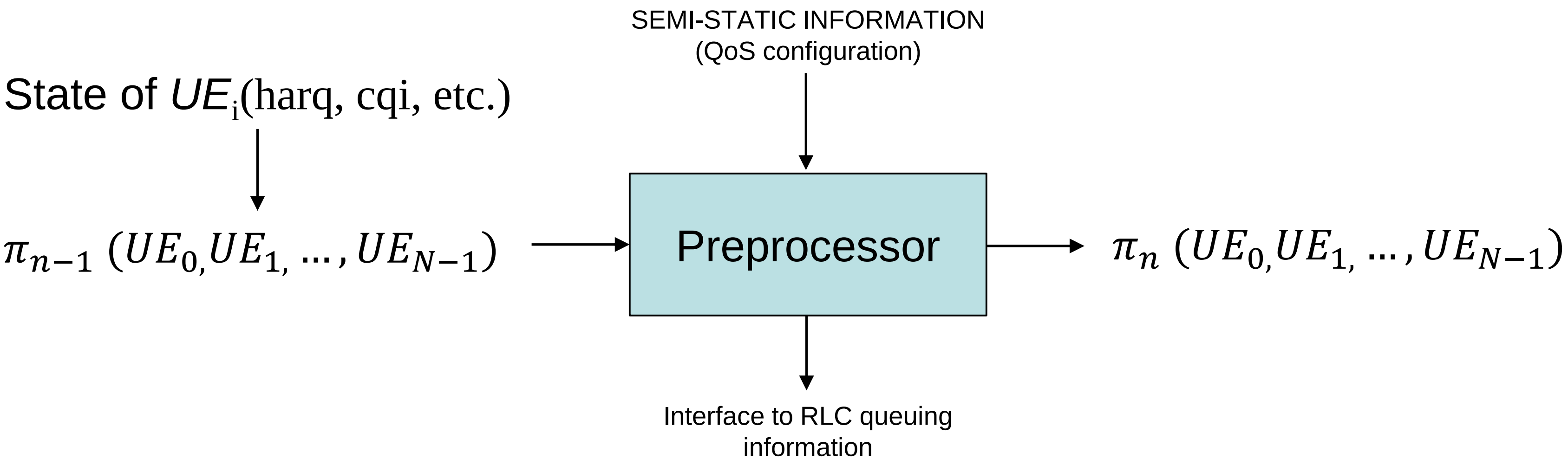
- **Actions upon reception of UL indications**
  - SRS indication
    - Store UL RB SNR and timing advance indications (Rel-8 information)
    - Store UL quantized channel responses (Rel-10 TDD)
      - 8-bit I/Q channel estimates, used for reciprocity-based beamforming
    - Store UL ToA estimate (Rel-11 TDD)
  - RACH indication
    - Initiate random-access procedure for a temporary UE
  - SR indication
    - Activate flag to allow for UL scheduling
  - HARQ indication
    - Update round counters for DL preprocessor and deterministic scheduler
  - RX indication
    - Send received SDU up, handle case of MAC CE, CCCH specially
  - CRC indication
    - Manage UL HARQ mechanism for ULSCH decoding error
  - CQI indication
    - Update DL CQI/PMI/RI information

UE Power Control

- PUSCH power control is achieved by controlling a target SNR (based on L1 PUSCH SNR reporting via FAPI `ul_cqi`) using a simple control loop with hysteresis. Updates do not occur more often that once per radio frame via a Format 0 DCI.
- PUCCH power control is achieved by controlling a target SNR (based on L1 PUCCH SNR reporting via FAPI `ul_cqi`) using a simple control loop with hysteresis. Updates do not occur more often that once per radio frame via a UE-specific DL DCI.

Preprocessor module (main functions)

- **Preprocessor is the central scheduling entity for DL and UL**
- **Today it is part of the UE-specific DL and UL scheduler**
  - It will become a separate module
    - Dynamically loadable/linkable
    - Remote (cloud app)
  - Proper interfaces for MACRLC module to be defined
- **Objectives of DL preprocessor (current)**
  - Determine UEs to schedule : i.e. how many bytes per UE per subframe and per component carrier
  - Suggest allocations for
    - PRBs (number and physical subbands)
    - PMI/MIMO layer information
    - Beamforming (TM7-10)
  - Priority list for "deterministic" allocations of DCI and physical resources carried out by LTE mechanisms described above
    - Operate on state of network (UE_list) and reorder according prioties and "pre"-allocate resources



Preprocessor module (main functions)

- **Future extensions for objectives**
  - Allocations between RATs (essentially LTE/LTE-M/NB-IoT/eMBMS) and potentially RAN slices for 5G
  - Specific control mechanisms for slicing/sharing
  - Fine-grain QoS control (managing throughput/latency requirements)
- **Inputs (basic)**
  - DL preprocessor
    - DL CQI/PMI/HARQ feedback (L1 FAPI)
    - RLC queue status (RLC interface)
    - Logical channel configuration (RRC)
    - UE capabilities
  - UL preprocessor
    - UL SRS (L1 FAPI)
    - UL SNR on PUSCH/PUCCH (L1 FAPI)
    - UE Buffer status, PHR (321 procedures)
    - UL RB masks
- **Inputs (advanced/later)**
  - RAN sharing info/configuration (LTE-M/LTE/eMBMS/NB-IoT)
  - Advanced measurements
    - Virtual cell measurements/advanced spatio-temporal measurements (from reciprocity-based mechanisms)
- **Outputs (basic LTE functionality)**
  - UE list ordering (order in which deterministic scheduler is executed) **-> existing**
  - Target mcs, target bytes / logical channel/component carrier
  - Pre-allocated PRB number and their subbands per user/component carrier **-> existing**
  - Number of layers (DL : TM3/5/8/9/10) **-> needed**
  - PMI allocations (DL: TM4-6) **-> needed**
  - Beamforming per UE (DL: TM7-10) **-> needed**
- **Outputs (nice to have later)**
  - Virtual cell creation
  - Dynamic LTE-M/LTE/eMBMS/NB-IoT allocations

MAC-RLC Internal Interface

- **Logical channel interface interactions use three message types implemented as direct function calls from the MAC thread:**
  - `mac_rlc_data_req` (MAC→RLC) : this function requests downlink SDUs from RLC for transport channel multiplexing. A particular number of bytes are requested and the closest amount is returned to MAC.
  - `mac_rlc_data_ind` (MAC→RLC) : this function transfers uplink SDUs received by MAC to the target logical channel for reassembly by RLC.
  - `mac_rlc_status_ind` (MAC→RLC) : this function retrieves RLC logical channel queue status during the MAC scheduling operation. It is typically invoked during the pre-processor step in order to determine the number of bytes that are to be scheduled per-user in a particular subframe. It is also called just prior to requesting a target number of bytes from the RLC.

MAC Configuration Interface

- **The configuration interface for the MACRLC entity consists of MAC/PHY configuration and RLC configuration functions. The MAC/PHY configuration interface is implemented using a direct function call from RRC to MAC,** `rrc_mac_config_req` **which can transfer the following parameters to the MAC layer**
  - physCellId : physical cell ID for L1 instance
  - p_eNB : number of logical antenna ports for L1 instance
  - Ncp : cyclic prefix mode for L1 instance
  - eutra_band : eutra band for L1 instance
  - dl_CarrierFreq : absolute downlink carrier frequency (Hz) for L1 instance
  - ul_CarrierFreq : absolute uplink carrier frequency (Hz) for L1 instance
  - pbch_repetition : PBCH repetition indicator
  - In addition the MAC configuration contains the following raw RRC information elements
  - BCCH-BCH-Message
  - RadioResourceConfig-CommonSIB for LTE cell
  - RadioResourceConfig-CommonSIB for LTE-M cell
  - PhysicalConfig-Dedicated
  - SCell-ToAddMod-r10
  - PhysicalConfig-DedicatedSCell-r10
  - MeasObject-ToAddMod
  - MAC_Main-Config
  - logicalChannelIdentity
  - LogicalChannel-Config
  - MeasGap-Config
  - TDD-Config
  - MobilityControl-Info
  - SchedulingInfo-List
  - ul_Bandwidth
  - AdditionalSpectrumEmission
  - MBSFN_SubframeConfigList
  - MBSFN_AreaInfoList_r9
  - PMCH_InfoList_r9
  - SystemInformationBlockType1-v1310-Ies

MAC Control Data Interface

- **The MAC control data interface is used to transfer transparent SDUs from the RRC to the MAC layer for**
  - CCCH
  - BCCH-BCH
  - BCCH-DLSCH
- **The interface is implemented by a direct function call to RRC,** `mac_rrc_data_request` **passing a logical channel identifier. It does not traverse the RLC-TM interface. The possible logical channels are**
  - CCCH_LCHANID (0)
  - BCCH (3)
  - PCCH (4)
  - MIBCH (5)
  - BCCH_SIB1_BR (6)
  - BCCH_SI_BR (7)
- **If data is to be transported by MAC for any of these transparent logical channels the function returns a payload with a non-zero byte-count.**

RLC Configuration Interface

- **The RLC layer is configured using the function `rrc_rlc_config_asn1_req` which conveys up to three information elements**
  - `SRB-ToAddMod-List`
  - `DRB-ToAddMod-List`
  - `DRB_ToRelease-List`
  - `PMCH-Info-List-r9`
- **The presense of one of these information elements configures the list of radio-bearers to be activated by the RLC unit.**

MACRLC-PDCP interface(SRB and DRB)

- **The radio-bearer interface between PDCP and MACRLC is controlled by two functions**
  - `pdcp_data_ind` (RLC→PDCP) is used to transfer an uplink SDU from RLC to PDCP for a particular signaling or data radio-bearer. It is called from the RLC unit (TM,UM or AM) that has active data in its queue.
  - `rlc_data_req` (PDCP→RLC) is used to transfer a downlink SDU from PDCP to RLC for a particular signaling or data radio-bearer. It is called from the PDCP entity and routed inside the RLC to the desired unit (TM,UM or AM) for segmentation and queuing.