RC xApp Implementation

ECE 5984: 5G-Advanced, O-RAN & 6G

Virginia Tech

Overview

- Identify a RAN function to control.
  - Addition/deletion of a DRB (QoE xApp).
  - **Release RRC Connection of a particular UE (RRC release xApp).**
  - Change PRB allocation of a network slice (Slicing xApp).
- The RC xApp uses the data structures defined in E2SM-RC to communicate Control Messages to the E2 node.
- Go over the basics of creating an xApp.
  - xApp initialization.
  - Connection to near-RT RIC.
- Subscribing to messages from the RAN.
- Sending messages to the RIC.
  - Encoding messages
- Decoding messages at the RAN (E2 Node).
- Implement the logic to execute control action once message is received and successfully decoded.

E2SM-RC Services and Styles

Service Type	# Styles (v7.0)	Example Actions	Control Domain
Report	4	KPM stats, UE context	Monitoring
Insert	8	Pause + HO/DRB updates	Pre-decision triggers
Control	11	DRB QoS, Handover, CA, Beamforming	Real-time actuation
Policy	9	Slice PRB quota, HO offset	Conditional management
Query	2	UE/Node info retrieval	On-demand diagnostics

RC CONTROL SERVICE – STYLES AND ACTIONS

7.6 Supported RIC CONTROL Services

7.6.1 CONTROL Service Style Types

RIC Style Type	Style Name	Style Description
1	Radio Bearer control	Used to modify the configuration the Radio Bearer Control (RBC) related parameters and/or behaviours at the E2 Node for a specific UE Belongs to Fundamental level CONTROL Services.
2	Radio resource allocation control	Used to modify the configuration the Radio Resource Allocation control related parameters and/or behaviours at the E2 Node for a specific E2 Node, cell, slice, UE and/or QoS Belongs to Fundamental level CONTROL Services.
3	Connected mode mobility control	Used to initiate a connected mode mobility procedure (Handover or Conditional Handover), optionally with Dual Active Protocol Stack (DAPS), for a specific UE towards either a target cell (for HO) or a list of candidate cells (for CHO) Belongs to Fundamental level CONTROL Services.
4	Radio access control	Used to modify Radio access related functions used to control UE access to cells Belongs to Fundamental level CONTROL Services.
5	Dual connectivity (DC) control	Used to initiate Dual connectivity (DC) mechanisms Belongs to Fundamental level CONTROL Services.
6	Carrier Aggregation (CA) control	Used to initiate Carrier Aggregation (CA) mechanisms Belongs to Fundamental level CONTROL Services.
7	Idle mode mobility control	Used to modify Idle mode mobility related functions used to control UE reselection of cells Belongs to Fundamental level CONTROL Services.
8	UE information and assignment	Used for *Explicit UE list* assignment, UE information report generation and to complete UE identification. These services are used to support other RIC services. Belongs to Fundamental level CONTROL Services.
9	Measurement Reporting Configuration control	Used to control the measurement report configuration
255	Multiple Actions Control	Used for multiple actions of the selected fundamental level CONTROL Service style(s). Belongs to Integrated level CONTROL Services.

The supported RAN control actions and the corresponding RAN parameters are as follows.

Control Action ID	Control Action Name	Control Action description	Associated RAN Parameters
1	DRB QoS Configuration	To control the configuration of DRB QoS profile	8.4.2.1
2	QoS flow mapping configuration	To control the multiplexing of QoS flows to a DRB (addition, modification, deletion	8.4.2.2
3	Logical channel configuration	To control the LCID configuration of a DRB	8.4.2.3
4	Radio admission control	To control radio admission of a UE	8.4.2.4
5	DRB termination control	To control the change in bearer termination point	8.4.2.5
6	DRB split ratio control	To control the split ratio of a DRB across its RLC entities	8.4.2.6
7	PDCP Duplication control	To control activation or de-activation of PDCP duplication for a DRB and control/configure the number of legs or RLC entities for the DRB	8.4.2.7

E2SM-RC Services and Styles (Detailed)

RIC Service	Service Style	Example Use Cases / Purpose	Supported Control / Report Actions
REPORT	**Message Copy**	Copy NI/RRC message exchanges with associated UE/E2 node info for analysis.	- Copy of RRC/NAS messages (DL/UL-DCCH)- Capture UE context (state, DRB IDs, QoS)
	Call Process Outcome	Report results of call processing events or signaling outcomes.	- Report UE admission / release results- HO success/failure indicators
	E2 Node Information	Provide node-level configuration and status data.	- Cell ID, PLMN, DU/CU configuration- Slice configuration summary
	UE Information	Report UE-specific RRC/L2/L3 context and measurement info.	- UE RNTI, DRB QoS info, handover state, buffer status
INSERT	**Radio Bearer Control Request**	Temporarily pause call process to reconfigure radio bearers (QoS or DRB).	- Add/modify/delete DRB- Change DRB QoS profile (5QI, GBR, priority)- Logical channel (LCID) mapping
	Radio Resource Allocation Request	Request for dynamic adjustment of scheduling parameters.	- Adjust DRX timers, SPS/CG configurations- Slice-level PRB quota changes- CQI/SRS configuration update
	Connected Mode Mobility	Suspend to trigger optimized handover or conditional HO decision.	- Trigger HO / CHO / DAPS- Update HO preparation parameters
	Radio Access Control Request	Modify RRC connection handling or access control parameters.	- RRC Connection Reject / Release- RACH backoff / barring control
	Dual Connectivity	Manage Multi-RAT dual connectivity.	- Add/Modify/Release Secondary Node (SN)
	Carrier Aggregation	Manage multi-carrier aggregation setup.	- CA initiation or modification
	Idle Mode Mobility	Adjust reselection priorities or cell lists during UE idle mode.	- Modify intra/inter-frequency reselection priorities
	Multiple Actions	Execute multiple actions in one transaction.	- Combine DRB + Mobility + Power actions in one control
CONTROL	**Radio Bearer Control**	Directly command DRB reconfiguration or QoS updates.	- Modify QoS flows- Add/delete DRBs- PDCP duplication control
	Radio Resource Allocation	Fine-grained allocation of MAC/PHY resources.	- DRX/SPS/CG config- Slice PRB quota- CSI reporting and DMRS config
	Connected Mode Mobility	Execute handover-related procedures.	- HO / CHO / DAPS execution- HO cancel, revert, or trigger new cell selection
	Radio Access Control	Immediate RRC control at access layer.	- UE admission control- RRC reject/release decisions
	Dual Connectivity	Control multi-node attachment.	- Activate / deactivate dual connectivity
	Carrier Aggregation	Manage carrier assignment.	- Add / remove / reassign secondary cells
	Idle Mode Mobility	Tune UE reselection in idle mode.	- Modify reselection parameters (A3/A5 events, thresholds)
	UE Identification	Maintain or query explicit UE list.	- UE ID registration / update
	Beamforming Configuration	Adjust beam selection / training.	- Switch GoB / non-GoB mode- Configure beam groups
	Measurement Reporting (MR) Configuration	Change measurement configurations.	- Add / modify / delete MR configuration- Adjust measurement gaps
	Multiple Actions	Execute several controls atomically.	- Simultaneous bearer, resource, and mobility control
POLICY	**Radio Bearer Policy**	Define static/conditional bearer management rules.	- QoS policy enforcement for DRBs, LCIDs
	Radio Resource Allocation Policy	Apply scheduling / resource rules per slice or user group.	- PRB allocation policy- DRX, SR, SPS policy tuning
	Connected Mode Mobility Policy	Define threshold-based mobility rules.	- Handover offset policy- DAPS enable conditions
	Radio Access Policy	Manage access admission thresholds.	- Access barring policies
	Dual Connectivity Policy	Control DC activation based on load or QoS.	- DC selection policy
	Carrier Aggregation Policy	Define CA activation/release conditions.	- Per-slice or per-UE CA policy
	Idle Mode Mobility Policy	Define cell reselection behaviors.	- Priority offsets / timers
	Measurement Reporting Policy	Trigger-based measurement policy.	- Define MR gaps / events / filters
	Beamforming Policy	Define beam usage or switching rules.	- GoB vs non-GoB usage per cell
QUERY	**E2 Node Information Query**	Request immediate node config.	- Cell parameters, slice data, load
	UE Information Query	Retrieve specific UE context.	- RRC state, DRB stats, mobility state

Creating an xApp

- Let's take a look at an existing xApp implementation which uses the KPM and RC service models.
- We will be using the existing implementation and make modifications on top of that.
- Open the **xapp_kpm_rc.c** file using

vim ~/flexric/examples/xApp/c/kpm_rc/xapp_kpm_rc.c

Initialize xApp API and connect to near-RT RIC

Get information regarding all the E2 nodes connected to the near-RT RIC and print the number of nodes connected.

```
530 int main(int argc, char* argv[])
531 {
532   fr_args_t args = init_fr_args(argc, argv);
533
534   // Init the xApp
535   init_xapp_api(&args);
536   sleep(1);
537
538   e2_node_arr_xapp_t nodes = e2_nodes_xapp_api();
539   assert(nodes.len > 0);
540
541   printf("[KPM RC]: Connected E2 nodes = %d\n", nodes.len);
542
543   pthread_mutexattr_t attr = {0};
544   int rc = pthread_mutex_init(&mtx, &attr);
545   assert(rc == 0);
546
547   sm_ans_xapp_t* hndl = calloc(nodes.len, sizeof(sm_ans_xapp_t));
548   assert(hndl != NULL);
549
```

Receiving messages from the RAN

For each E2 node connected to the near-RT RIC, check if the KPM service model is supported

If KPM service model is supported, send a subscription request to the E2 node, so that the E2 node starts sending REPORT messages

Assign the **report_sm_xapp_api** function to the handle, so that **sm_cb_kpm** is called whenever a REPORT message is received. This function decodes the contents of the report and saves it in the database or prints it on the console.

```
550   ////////////
551   // START KPM
552   ////////////
553   int const KPM_ran_function = 2;
554
555   for (size_t i = 0; i < nodes.len; ++i) {
556     e2_node_connected_xapp_t* n = &nodes.n[i];
557
558     size_t const idx = find_sm_idx(n->rf, n->len_rf, eq_sm, KPM_ran_function);
559     assert(n->rf[idx].defn.type == KPM_RAN_FUNC_DEF_E && "KPM is not the received RAN Function");
560     // if REPORT Service is supported by E2 node, send SUBSCRIPTION
561     // e.g. OAI CU-CP
562     if (n->rf[idx].defn.kpm.ric_report_style_list != NULL) {
563       // Generate KPM SUBSCRIPTION message
564       kpm_sub_data_t kpm_sub = gen_kpm_subs(&n->rf[idx].defn.kpm);
565
566       hndl[i] = report_sm_xapp_api(&n->id, KPM_ran_function, &kpm_sub, sm_cb_kpm);
567       assert(hndl[i].success == true);
568
569       free_kpm_sub_data(&kpm_sub);
570     }
571   }
572   ////////////
573   // END KPM
574   ////////////
```

Sending messages to the RAN

For each E2 node connected to the near-RT RIC, check if the RC service model is supported.

If RC service model is supported, generate a CONTROL request and send it to the E2 node.

```
578   ////////////
579   // START RC
580   ////////////
581   int const RC_ran_function = 3;
582
583   for (size_t i = 0; i < nodes.len; ++i) {
584     e2_node_connected_xapp_t* n = &nodes.n[i];
585
586     size_t const idx = find_sm_idx(n->rf, n->len_rf, eq_sm, RC_ran_function);
587     assert(n->rf[idx].defn.type == RC_RAN_FUNC_DEF_E && "RC is not the received RAN Function");
588     // if CONTROL Service is supported by E2 node, send CONTROL message
589     if (n->rf[idx].defn.rc.ctrl != NULL) {
590       // Generate RC CONTROL message
591       rc_ctrl_req_data_t rc_ctrl = gen_rc_ctrl_msg(n->rf[idx].defn.rc.ctrl);
592
593       control_sm_xapp_api(&n->id, RC_ran_function, &rc_ctrl);
594
595       free_rc_ctrl_req_data(&rc_ctrl);
596     }
597   }
598   ////////////
599   // END RC
600   ////////////
```

CONTENTS OF CONTROL MESSAGE

```
330 static
331 rc_ctrl_req_data_t gen_rc_ctrl_msg(ran_func_def_ctrl_t const* ran_func)
332 {
333   assert(ran_func != NULL);
334
335   rc_ctrl_req_data_t rc_ctrl = {0};
336
337   for (size_t i = 0; i < ran_func->sz_seq_ctrl_style; i++) {
338     assert(cmp_str_ba("Radio Bearer Control", ran_func->seq_ctrl_style[i].name) == 0 && "Add requested CONTROL Style.
339
340     // CONTROL HEADER
341     rc_ctrl.hdr.format = ran_func->seq_ctrl_style[i].hdr;
342     assert(rc_ctrl.hdr.format == FORMAT_1_E2SM_RC_CTRL_HDR && "Indication Header Format received not valid");
343     rc_ctrl.hdr.frmt_1.ric_style_type = 1;
344     // 6.2.2.6
345     {
346       lock_guard(&mtx);
347       rc_ctrl.hdr.frmt_1.ue_id = cp_ue_id_e2sm(&ue_id);
348     }
349
350     // CONTROL MESSAGE
351     rc_ctrl.msg.format = ran_func->seq_ctrl_style[i].msg;
352     assert(rc_ctrl.msg.format == FORMAT_1_E2SM_RC_CTRL_MSG && "Indication Message Format received not valid");
353
354     fill_rc_ctrl_act(ran_func->seq_ctrl_style[i].seq_ctrl_act,
355                      ran_func->seq_ctrl_style[i].sz_seq_ctrl_act,
356                      &rc_ctrl.hdr.frmt_1,
357                      &rc_ctrl.msg.frmt_1);
358   }
359
```

This message is sent by a Near-RT RIC to an E2 Node to initiate or resume a control function logic.

Direction: Near-RT RIC → E2 Node.

IE/Group Name	Presence	Range	IE type and reference	Semantics description	Criticality	Assigned Criticality
Message Type	M		9.2.3		YES	reject
RIC Request ID	M		9.2.7		YES	reject
RAN Function ID	M		9.2.8		YES	reject
RIC Call Process ID	O		9.2.18		YES	reject
RIC Control Header	M		9.2.20		YES	reject
RIC Control Message	M		9.2.19		YES	reject
RIC Control Ack Request	O		9.2.21		YES	reject

FILLING QOS PARAMETERS IN CONTROL MESSAGE

```
// rpl->lst_ran_param[0].ran_param_id = QOS_FLOW_ITEM_8_4_2_2;

rpl->lst_ran_param[0].ran_param_struct.sz_ran_param_struct = 2;
rpl->lst_ran_param[0].ran_param_struct.ran_param_struct = calloc(2, sizeof(seq_ran_param_t)
assert(rpl->lst_ran_param[0].ran_param_struct.ran_param_struct != NULL && "Memory exhausted
seq_ran_param_t* rps = rpl->lst_ran_param[0].ran_param_struct.ran_param_struct;

// QoS Flow Identifier
rps[0].ran_param_id = QOS_FLOW_ID_8_4_2_2;
rps[0].ran_param_val.type = ELEMENT_KEY_FLAG_TRUE_RAN_PARAMETER_VAL_TYPE;
rps[0].ran_param_val.flag_true = calloc(1, sizeof(ran_parameter_value_t));
assert(rps[0].ran_param_val.flag_true != NULL && "Memory exhausted");
rps[0].ran_param_val.flag_true->type = INTEGER_RAN_PARAMETER_VALUE;
// Let's suppose that we have QFI 10
rps[0].ran_param_val.flag_true->int_ran = 10;

// QoS Flow Mapping Indication
rps[1].ran_param_id = QOS_FLOW_MAPPING_IND_8_4_2_2;
rps[1].ran_param_val.type = ELEMENT_KEY_FLAG_FALSE_RAN_PARAMETER_VAL_TYPE;
rps[1].ran_param_val.flag_false = calloc(1, sizeof(ran_parameter_value_t));
assert(rps[1].ran_param_val.flag_false != NULL && "Memory exhausted");

// ENUMERATED (ul, dl, ...)
rps[1].ran_param_val.flag_false->type = INTEGER_RAN_PARAMETER_VALUE;
rps[1].ran_param_val.flag_false->int_ran = 1;

return qos_param;
}
```

8.4.2.2 QoS flow mapping configuration

Upon receiving the *RIC Control Request* message, the E2 node shall invoke procedures related to QoS flow mapping Configuration, such as *Bearer Context Management*, *UE Context Management*, *RRC Message Transfer*, etc. and include the IEs corresponding to one or more of parameters described below in the related interface messages. If the *DRB ID* is missing in the *RIC Control Request* message, the E2 node will send a *RIC Control Failure*.

RAN Parameter ID	RAN Parameter	RAN Parameter Value Type	Key Flag	RAN Parameter Definition	Semantics Description
1	DRB ID	ELEMENT	TRUE	*DRB ID* IE in TS 38.463 [21] Section 9.3.1.16	
2	List of QoS Flows to be modified in DRB	LIST			*Flow Mapping Information* IE in TS 38.463 [21] Section 9.3.1.26
3	>QoS Flow Item	STRUCTURE			*QoS Flow Item* IE in TS 38.463 [21] Section 9.3.1.12
4	>>QoS Flow Identifier	ELEMENT	TRUE	*QoS Flow Identifier* IE in TS 38.463 [21] Section 9.3.1.24	
5	>>QoS Flow Mapping Indication	ELEMENT	FALSE	*QoS Flow Mapping Indication* IE in TS 38.463 [21] Section 9.3.1.60	

Generate RRC Release

- We will reuse the **rrc_gNB_generate_RRCRelease** function defined in **rrc_gNB.c** that is used when the AMF issues a command to release a UE's context.
- The code executed when the gNB receives a context release command from the AMF can be found in **~/oai/openair2/RRC/NR/** directory. Open the **rrc_gNB_NGAP.c** file using,

```
vim ~/oai/openair2/RRC/NR/rrc_gNB_NGAP.c
```

- In this file, look for the **rrc_gNB_process_NGAP_UE_CONTEXT_RELEASE_COMMAND** function (*around line 1316*).
- We will reuse the logic defined here to generate a RRC release command when the gNB receives a RAN **Control** request.

Generate RRC Release

Get UE context by specifying the UE ID.

If UE context does not exist, then nothing to do – send UE context release complete

Applicable for split CU-CP, CU-UP modules. E1 interface will have to be used for releasing bearer if PDU session exists.

Check if UE is connected to CU and DU and generate RRC release if both checks are true.

```
1316 int rrc_gNB_process_NGAP_UE_CONTEXT_RELEASE_COMMAND(MessageDef *msg_p, instance_t instance)
1317 {
1318   gNB_RRC_INST *rrc = RC.nrrrc[instance];
1319   uint32_t gNB_ue_ngap_id = NGAP_UE_CONTEXT_RELEASE_COMMAND(msg_p).gNB_ue_ngap_id;
1320   rrc_gNB_ue_context_t *ue_context_p = rrc_gNB_get_ue_context(RC.nrrrc[instance], gNB_ue_ngap_id);
1321
1322   if (ue_context_p == NULL) {
1323     /* Can not associate this message to an UE index */
1324     LOG_W(NR_RRC, "[gNB %ld] In NGAP_UE_CONTEXT_RELEASE_COMMAND: unknown UE from gNB_ue_ngap_id (%u)\n",
1325           instance,
1326           gNB_ue_ngap_id);
1327     rrc_gNB_send_NGAP_UE_CONTEXT_RELEASE_COMPLETE(instance, gNB_ue_ngap_id, NULL);
1328     return -1;
1329   }
1330
1331   gNB_RRC_UE_t *UE = &ue_context_p->ue_context;
1332   UE->an_release = true;
1333 #ifdef E2_AGENT
1334   signal_rrc_state_changed_to(UE, RRC_IDLE_RRC_STATE_E2SM_RC);
1335 #endif
1336
1337   /* a UE might not be associated to a CU-UP if it never requested a PDU
1338    * session (intentionally, or because of erros) */
1339   if (ue_associated_to_cuup(rrc, UE)) {
1340     sctp_assoc_t assoc_id = get_existing_cuup_for_ue(rrc, UE);
1341     e1ap_cause_t cause = {.type = E1AP_CAUSE_RADIO_NETWORK, .value = E1AP_RADIO_CAUSE_NORMAL_RELEASE};
1342     e1ap_bearer_release_cmd_t cmd = {
1343       .gNB_cu_cp_ue_id = UE->rrc_ue_id,
1344       .gNB_cu_up_ue_id = UE->rrc_ue_id,
1345       .cause = cause,
1346     };
1347     rrc->cucp_cuup.bearer_context_release(assoc_id, &cmd);
1348   }
1349
1350   /* special case: the DU might be offline, in which case the f1_ue_data exists
1351    * but is set to 0 */
1352   if (cu_exists_f1_ue_data(UE->rrc_ue_id) && cu_get_f1_ue_data(UE->rrc_ue_id).du_assoc_id != 0) {
1353     rrc_gNB_generate_RRCRelease(rrc, UE);
1354
1355     /* UE will be freed after UE context release complete */
1356   } else {
1357     // the DU is offline already
1358     rrc_gNB_send_NGAP_UE_CONTEXT_RELEASE_COMPLETE(0, UE->rrc_ue_id, &UE->pduSessions);
1359     rrc_remove_ue(rrc, ue_context_p);
1360   }
1361
1362   return 0;
1363 }
```

Changes to be made in FlexRIC – xapp_kpm_rc

- In the **xapp_kpm_rc.c** file change **qfi** parameter to reflect UE ID (hardcoded).
  - Basically, we are reusing the existing data structure to represent the parameter we want to encode.
  - Since we will be having just one UE connected, use “1” as the UE ID value (as highlighted in the red box).
- Build the flexric module again.

```
// QoS Flow Identifier
rps[0].ran_param_id = QOS_FLOW_ID_8_4_2_2;
rps[0].ran_param_val.type = ELEMENT_KEY_FLAG_TRUE_RAN_PARAMETER_VAL_TYPE;
rps[0].ran_param_val.flag_true = calloc(1, sizeof(ran_parameter_value_t));
assert(rps[0].ran_param_val.flag_true != NULL && "Memory exhausted");
rps[0].ran_param_val.flag_true->type = INTEGER_RAN_PARAMETER_VALUE;
// Let's suppose that we have QFI 10
rps[0].ran_param_val.flag_true->int_ran = 1;
```

Changes to be made in OAI

- On the RAN side, make the identified changes in

~/oai/openair2/E2AP/RAN_FUNCTION/O-RAN/ran_func_rc.c

- In the beginning of the file, add the highlighted include statement.

```
@@ -29,6 +29,11 @@
 #include "openair2/E2AP/flexric/src/lib/sm/enc/enc_ue_id.h"
 #include "openair2/E2AP/flexric/src/sm/rc_sm/rc_sm_id.h"

+// Only include RRC header for gNB/CU-CP builds, not for CUUP
+#if !defined(NGRAN_GNB_CUUP) || (defined(NGRAN_GNB_DU) && defined(NGRAN_GNB_CUCP))
+#include "../../../RRC/NR/rrc_gNB_UE_context.h"
+#endif
+
 #include <stdio.h>
 #include <unistd.h>
 #include "common/ran_context.h"
```

Changes to be made in OAI

- Near the end of the file, look for the function *write_ctrl_rc_sm,* and add the highlighted lines of code inside the function (Around line 940).

```
+
   // QoS Flow Mapping Indication
   assert(lrp->ran_param_struct.ran_param_struct[1].ran_param_id == 5);
   assert(lrp->ran_param_struct.ran_param_struct[1].ran_param_val.type == ELEMENT_KEY_FLAG_FALSE_RAN_PARAMETER_VAL_TYPE);
   int64_t dir = lrp->ran_param_struct.ran_param_struct[1].ran_param_val.flag_false->int_ran;
   assert(dir == 0 || dir == 1);

-  printf("qfi = %ld, dir %ld \n", qfi, dir);
+  #if !defined(NGRAN_GNB_CUUP) || (defined(NGRAN_GNB_DU) && defined(NGRAN_GNB_CUCP))
+  // RRC Release functionality - only available in gNB/CU-CP builds
+  //assert(qfi < 1 && "UE ID cannot be less than 1");
+  ue_id_t ue_id = (ue_id_t)qfi;
+
+  // printf("qfi = %ld, dir %ld \n", qfi, dir);
+  printf("UE ID = %u (from QFI), dir %ld \n", ue_id, dir);
+  protocol_ctxt_t ctxt;
+
+  gNB_RRC_INST *rrc = RC.nrrrc[0];
+  rrc_gNB_ue_context_t *ue_context_p = rrc_gNB_get_ue_context(rrc, ue_id);
+
+  if (ue_context_p != NULL) {
+      gNB_RRC_UE_t *UE = &ue_context_p->ue_context;
+      printf("Found UE with RRC UE ID %u, RNTI %04x\n", UE->rrc_ue_id, UE->rnti);
+      printf("Triggering RRC Release for UE %u\n", ue_id);
+
+      rrc_gNB_generate_RRCRelease(rrc, UE);
+      printf("RRC Release triggered for UE %lu\n", ue_id);
+   } else {
+          printf("UE context not found for UE ID %lu\n", ue_id);
+  }
+#else
+  printf("RRC Release not supported in CUUP-only build\n");
+#endif
+
```

```
#if !defined(NGRAN_GNB_CUUP) || (defined(NGRAN_GNB_DU) && defined(NGRAN_GNB_CUCP))
// RRC Release functionality - only available in gNB/CU-CP builds
//assert(qfi < 1 && "UE ID cannot be less than 1");
ue_id_t ue_id = (ue_id_t)qfi;

// printf("qfi = %ld, dir %ld \n", qfi, dir);
printf("UE ID = %u (from QFI), dir %ld \n", ue_id, dir);
protocol_ctxt_t ctxt;

gNB_RRC_INST *rrc = RC.nrrrc[0];
rrc_gNB_ue_context_t *ue_context_p = rrc_gNB_get_ue_context(rrc, ue_id);

if (ue_context_p != NULL) {
    gNB_RRC_UE_t *UE = &ue_context_p->ue_context;
    printf("Found UE with RRC UE ID %u, RNTI %04x\n", UE->rrc_ue_id, UE->rnti);
    printf("Triggering RRC Release for UE %u\n", ue_id);

    rrc_gNB_generate_RRCRelease(rrc, UE);
    printf("RRC Release triggered for UE %lu\n", ue_id);
 } else {
      printf("UE context not found for UE ID %lu\n", ue_id);
 }
#else
 printf("RRC Release not supported in CUUP-only build\n");
#endif
```