Commit

[AMF] Fixed 5GMM cause in Reject message (#1660)

Browse files

When a UE that requests slices tries to connect and there are no slices configured, the reject message is:

5GMM cause = 0x7 (5GS Services not allowed)

however it should be:

5GMM cause = 0x3e (No network slices available)

All 5GMM cause value in reject message is reviewed in this commit

main
v2.6.4 ... v2.4.9

acetcom committed on Jul 19, 2022

1 parent 3885cb2 commit 668cc59

41 lib/nas/5gs/types.h

```
106 106
107 107   /* 9.11.3.2 5GMM cause
108 108    * M V 1 */
    109 +
    110 + /* REQUEST_ACCEPTED(16) cause is defined by Open5GS */
    111 + #define OGS_5GMM_CAUSE_REQUEST_ACCEPTED 16
    112 +
    113 + /* Annex A (informative): Cause values for 5GS mobility management
    114 +  * A.1 Causes related to UE identification */
109 115   #define OGS_5GMM_CAUSE_ILLEGAL_UE 3
110     - #define OGS_5GMM_CAUSE_PEI_NOT_ACCEPTED 5
111 116   #define OGS_5GMM_CAUSE_ILLEGAL_ME 6
112     - #define OGS_5GMM_CAUSE_5GS_SERVICES_NOT_ALLOWED 7
113 117   #define OGS_5GMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK 9
114 118   #define OGS_5GMM_CAUSE_IMPLICITLY_DE_REGISTERED 10
    119 +
    120 + /* A.2 Cause related to subscription options */
    121 + #define OGS_5GMM_CAUSE_PEI_NOT_ACCEPTED 5
    122 + #define OGS_5GMM_CAUSE_5GS_SERVICES_NOT_ALLOWED 7
115 123   #define OGS_5GMM_CAUSE_PLMN_NOT_ALLOWED 11
116 124   #define OGS_5GMM_CAUSE_TRACKING_AREA_NOT_ALLOWED 12
117 125   #define OGS_5GMM_CAUSE_ROAMING_NOT_ALLOWED_IN_THIS_TRACKING_AREA 13
118 126   #define OGS_5GMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA 15
    127 + #define OGS_5GMM_CAUSE_N1_MODE_NOT_ALLOWED 27
    128 + #define OGS_5GMM_CAUSE_REDIRECTION_TO_EPC_REQUIRED 31
    129 + #define OGS_5GMM_CAUSE_NON_3GPP_ACCESS_TO_5GCN_NOT_ALLOWED 72
    130 + #define OGS_5GMM_CAUSE_TEMPORARILY_NOT_AUTHORIZED_FOR_THIS_SNPN 74
    131 + #define OGS_5GMM_CAUSE_PERMANENTLY_NOT_AUTHORIZED_FOR_THIS_SNPN 75
    132 + #define OGS_5GMM_CAUSE_NOT_AUTHORIZED_FOR_THIS_CAG_OR_AUITHORIZED_FOR_CAG_CELLS_ONLY 76
    133 + #define WIRELESS_ACCESS_AREA_NOT_ALLOWED 76
    134 +
    135 + /* A.3 Causes related to PLMN or SNPN specific network failures
    136 +  * and congestion/authentication failures */
119 137   #define OGS_5GMM_CAUSE_MAC_FAILURE 20
```

```
120  138    #define OGS_5GMM_CAUSE_SYNCH_FAILURE 21
121  139    #define OGS_5GMM_CAUSE_CONGESTION 22
122  140    #define OGS_5GMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH 23
123  141    #define OGS_5GMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED 24
124  142    #define OGS_5GMM_CAUSE_NON_5G_AUTHENTICATION_UNACCEPTABLE 26
125       - #define OGS_5GMM_CAUSE_N1_MODE_NOT_ALLOWED 27
126  143    #define OGS_5GMM_CAUSE_RESTRICTED_SERVICE_AREA 28
127  144    #define OGS_5GMM_CAUSE_LADN_NOT_AVAILABLE 43
     145  + #define OGS_5GMM_CAUSE_NO_NETWORK_SLICES_AVAILABLE 62
128  146    #define OGS_5GMM_CAUSE_MAXIMUM_NUMBER_OF_PDU_SESSIONS_REACHED 65
129  147    #define OGS_5GMM_CAUSE_INSUFFICIENT_RESOURCES_FOR_SPECIFIC_SLICE_AND_DNN 67
130  148    #define OGS_5GMM_CAUSE_INSUFFICIENT_RESOURCES_FOR_SPECIFIC_SLICE 69
131  149    #define OGS_5GMM_CAUSE_NGKSI_ALREADY_IN_USE 71
132       - #define OGS_5GMM_CAUSE_NON_3GPP_ACCESS_TO_5GCN_NOT_ALLOWED 72
133  150    #define OGS_5GMM_CAUSE_SERVING_NETWORK_NOT_AUTHORIZED 73
134  151    #define OGS_5GMM_CAUSE_PAYLOAD_WAS_NOT_FORWARDED 90
135  152    #define OGS_5GMM_CAUSE_DNN_NOT_SUPPORTED_OR_NOT_SUBSCRIBED_IN_THE_SLICE 91
136  153    #define OGS_5GMM_CAUSE_INSUFFICIENT_USER_PLANE_RESOURCES_FOR_THE_PDU_SESSION 92
     154  +
     155  + /* A.4 Causes related to invalid messages */
137  156    #define OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
138  157    #define OGS_5GMM_CAUSE_INVALID_MANDATORY_INFORMATION 96
139  158    #define OGS_5GMM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
291  310        /*
292  311         * Do not change 'ogs_5gs_tai_t' to 'ogs_nas_tracking_area_identity_t'.
293  312         * Use 'ogs_5gs_tai_t' for easy implementation.
294       -     * ogs_nas_tai_list_build() changes to NAS
     313  +     * ogs_nas_tai_list_build() changes to NAS
295  314         * format(ogs_nas_tracking_area_identity_t)
296  315         * and is sent to the UE.
297  316         */
639  658
640  659    /* 9.11.4.2 5GSM cause
641  660     * O TV 2 */
     661  +
     662  + /* Annex B (informative): Cause values for 5GS session management
     663  +  * B.1 Causes related to nature of request */
642  664    #define OGS_5GSM_CAUSE_OPERATOR_DETERMINED_BARRING 8
643  665    #define OGS_5GSM_CAUSE_INSUFFICIENT_RESOURCES 26
644  666    #define OGS_5GSM_CAUSE_MISSING_OR_UNKNOWN_DNN 27
647  669    #define OGS_5GSM_CAUSE_REQUEST_REJECTED_UNSPECIFIED 31
648  670    #define OGS_5GSM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED 32
649  671    #define OGS_5GSM_CAUSE_REQUESTED_SERVICE_OPTION_NOT_SUBSCRIBED 33
650       - #define OGS_5GSM_CAUSE_SERVICE_OPTION_TEMPORARILY_OUT_OF_ORDER 34
651  672    #define OGS_5GSM_CAUSE_PTI_ALREADY_IN_USE 35
652  673    #define OGS_5GSM_CAUSE_REGULAR_DEACTIVATION 36
653  674    #define OGS_5GSM_CAUSE_NETWORK_FAILURE 38
654  675    #define OGS_5GSM_CAUSE_REACTIVATION_REQUESTED 39
     676  + #define OGS_5GSM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION 41
     677  + #define OGS_5GSM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION 42
655  678    #define OGS_5GSM_CAUSE_INVALID_PDU_SESSION_IDENTITY 43
656  679    #define OGS_5GSM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS 44
657  680    #define OGS_5GSM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS 45
660  683    #define OGS_5GSM_CAUSE_PDU_SESSION_TYPE_IPV4_ONLY_ALLOWED 50
661  684    #define OGS_5GSM_CAUSE_PDU_SESSION_TYPE_IPV6_ONLY_ALLOWED 51
662  685    #define OGS_5GSM_CAUSE_PDU_SESSION_DOES_NOT_EXIST 54
     686  + #define OGS_5GSM_CAUSE_PDU_SESSION_TYPE_IPV4V6_ONLY_ALLOWED 57
     687  + #define OGS_5GSM_CAUSE_PDU_SESSION_TYPE_UNSTRUCTURED_ONLY_ALLOWED 58
     688  + #define OGS_5GSM_CAUSE_UNSUPPORTED_5QI_VALUE 59
     689  + #define OGS_5GSM_CAUSE_PDU_SESSION_TYPE_ETHERNET_ONLY_ALLOWED 61
663  690    #define OGS_5GSM_CAUSE_INSUFFICIENT_RESOURCES_FOR_SPECIFIC_SLICE_AND_DNN 67
664  691    #define OGS_5GSM_CAUSE_NOT_SUPPORTED_SSC_MODE 68
665  692    #define OGS_5GSM_CAUSE_INSUFFICIENT_RESOURCES_FOR_SPECIFIC_SLICE 69
669  696    #define OGS_5GSM_CAUSE_SEMANTIC_ERROR_IN_THE_QOS_OPERATION 83
670  697    #define OGS_5GSM_CAUSE_SYNTACTICAL_ERROR_IN_THE_QOS_OPERATION 84
671  698    #define OGS_5GSM_CAUSE_INVALID_MAPPED_EPS_BEARER_IDENTITY 85
     699  +
```

```
                700 + /* B.2 Protocol errors (e.g., unknown message) */
672             701   #define OGS_5GSM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
673             702   #define OGS_5GSM_CAUSE_INVALID_MANDATORY_INFORMATION 96
674             703   #define OGS_5GSM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
```

220 lib/nas/eps/types.h

```
182             182   } __attribute__ ((packed)) ogs_nas_csfb_response_t;
183             183
184             184   /* 9.9.3.7 Detach type
185               -    * M V 1/2
186               -    * 9.9.3.21 NAS key set identifier
                185 +  * M V 1/2
                186 +  * 9.9.3.21 NAS key set identifier
187             187     * M V 1/2 */
188             188   #define OGS_NAS_DETACH_TYPE_FROM_UE_EPS_DETACH                  1
189             189   #define OGS_NAS_DETACH_TYPE_FROM_UE_IMSI_DETACH                 2
222             222   } __attribute__ ((packed)) ogs_nas_drx_parameter_t;
223             223
224             224   /* 9.9.3.9 EMM cause
225               -    * O TV 2
226               -    * Annex A (informative) Cause values for EPS mobility management
                225 +  * O TV 2 */
                226 +
                227 + /* REQUEST_ACCEPTED(16) cause is defined by Open5GS */
                228 + #define EMM_CAUSE_REQUEST_ACCEPTED 16
                229 +
                230 + /* Annex A (informative) Cause values for EPS mobility management
227             231     * A.1 Causes related to UE identification */
228               - #define EMM_CAUSE_IMSI_UNKNOWN_IN_HSS 2
229               - #define EMM_CAUSE_ILLEGAL_UE 3
230               - #define EMM_CAUSE_IMSI_UNKNOWN_IN_VLR 4
231               - #define EMM_CAUSE_ILLEGAL_ME 6
232               - #define EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK 9
233               - #define EMM_CAUSE_IMPLICITLY_DETACHED 10
                232 + #define OGS_NAS_EMM_CAUSE_IMSI_UNKNOWN_IN_HSS 2
```

Showing **22 changed files** with **386 additions** and **343 deletions**.

Split | Unified

```
                237 + #define OGS_NAS_EMM_CAUSE_IMPLICITLY_DETACHED 10
234             238   /* A.2 Cause related to subscription options */
235               - #define EMM_CAUSE_IMEI_NOT_ACCEPTED 5
236               - #define EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED 7
237               - #define EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED 8
238               - #define EMM_CAUSE_PLMN_NOT_ALLOWED 11
239               - #define EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED 12
240               - #define EMM_CAUSE_ROAMING_NOT_ALLOWED_IN_THIS_TRACKING_AREA 13
241               - #define EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED_IN_THIS_PLMN 14
242               - #define EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA 15
243               - #define EMM_CAUSE_REQUESTED_SERVICE_OPTION_NOT_AUTHORIZED_IN_THIS_PLMN 35
244               - #define EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED 40
245               - /* A.3 Causes related to PLMN specific network failures and
                239 + #define OGS_NAS_EMM_CAUSE_IMEI_NOT_ACCEPTED 5
                240 + #define OGS_NAS_EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED 7
                241 + #define OGS_NAS_EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED 8
                242 + #define OGS_NAS_EMM_CAUSE_PLMN_NOT_ALLOWED 11
                243 + #define OGS_NAS_EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED 12
                244 + #define OGS_NAS_EMM_CAUSE_ROAMING_NOT_ALLOWED_IN_THIS_TRACKING_AREA 13
                245 + #define OGS_NAS_EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED_IN_THIS_PLMN 14
                246 + #define OGS_NAS_EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA 15
                247 + #define OGS_NAS_EMM_CAUSE_REQUESTED_SERVICE_OPTION_NOT_AUTHORIZED_IN_THIS_PLMN 35
                248 + #define OGS_NAS_EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED 40
                249 + /* A.3 Causes related to PLMN specific network failures and
246             250     *     congestion/authentication failures */
247               - #define EMM_CAUSE_MSC_TEMPORARILY_NOT_REACHABLE 16
248               - #define EMM_CAUSE_NETWORK_FAILURE 17
```

```
249       - #define EMM_CAUSE_CS_DOMAIN_NOT_AVAILABLE 18
250       - #define EMM_CAUSE_ESM_FAILURE 19
251       - #define EMM_CAUSE_MAC_FAILURE 20
252       - #define EMM_CAUSE_SYNCH_FAILURE 21
253       - #define EMM_CAUSE_CONGESTION 22
254       - #define EMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH 23
255       - #define EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED 24
256       - #define EMM_CAUSE_NON_EPS_AUTHENTICATION_UNACCEPTABLE 26
257       - #define EMM_CAUSE_CS_SERVICE_TEMPORARILY_NOT_AVAILABLE 39
258       - #define EMM_CAUSE_SEVERE_NETWORK_FAILURE 42
259       - /* A.4 Causes related to nature of request
      251 + #define OGS_NAS_EMM_CAUSE_MSC_TEMPORARILY_NOT_REACHABLE 16
      252 + #define OGS_NAS_EMM_CAUSE_NETWORK_FAILURE 17
      253 + #define OGS_NAS_EMM_CAUSE_CS_DOMAIN_NOT_AVAILABLE 18
      254 + #define OGS_NAS_EMM_CAUSE_ESM_FAILURE 19
      255 + #define OGS_NAS_EMM_CAUSE_MAC_FAILURE 20
      256 + #define OGS_NAS_EMM_CAUSE_SYNCH_FAILURE 21
      257 + #define OGS_NAS_EMM_CAUSE_CONGESTION 22
      258 + #define OGS_NAS_EMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH 23
      259 + #define OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED 24
      260 + #define OGS_NAS_EMM_CAUSE_NON_EPS_AUTHENTICATION_UNACCEPTABLE 26
      261 + #define OGS_NAS_EMM_CAUSE_CS_SERVICE_TEMPORARILY_NOT_AVAILABLE 39
      262 + #define OGS_NAS_EMM_CAUSE_SEVERE_NETWORK_FAILURE 42
      263 + /* A.4 Causes related to nature of request
260   264     * NOTE:   This subclause has no entries in this version of the specification *
261   265     * A.5 Causes related to invalid messages */
262       - #define EMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
263       - #define EMM_CAUSE_INVALID_MANDATORY_INFORMATION 96
264       - #define EMM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
265       - #define EMM_CAUSE_MESSAGE_TYPE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 98
266       - #define EMM_CAUSE_INFORMATION_ELEMENT_NON_EXISTENT_OR_NOT_IMPLEMENTED 99
267       - #define EMM_CAUSE_CONDITIONAL_IE_ERROR 100
268       - #define EMM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 101
269       - #define EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED 111
      266 + #define OGS_NAS_EMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
      267 + #define OGS_NAS_EMM_CAUSE_INVALID_MANDATORY_INFORMATION 96
      268 + #define OGS_NAS_EMM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
      269 + #define OGS_NAS_EMM_CAUSE_MESSAGE_TYPE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 98
      270 + #define OGS_NAS_EMM_CAUSE_INFORMATION_ELEMENT_NON_EXISTENT_OR_NOT_IMPLEMENTED 99
      271 + #define OGS_NAS_EMM_CAUSE_CONDITIONAL_IE_ERROR 100
      272 + #define OGS_NAS_EMM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 101
      273 + #define OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED 111
270   274   typedef uint8_t ogs_nas_emm_cause_t;
271   275
272   276   /* 9.9.3.10 * EPS attach result
280   284
281   285   /* 9.9.3.11 EPS attach type
282   286    * M V 1/2
283       -  * 9.9.3.21 NAS key set identifier
      287 +  * 9.9.3.21 NAS key set identifier
284   288    * M V 1/2 */
285   289   #define OGS_NAS_KEY_SET_IDENTIFIER_NATIVE 0
286   290   #define OGS_NAS_KEY_SET_IDENTIFIER_MAPPED 1
320   324       };
321   325   } __attribute__ ((packed)) ogs_nas_eps_mobile_identity_t;
322   326
323       - /* 9.9.3.12A EPS network feature support
      327 + /* 9.9.3.12A EPS network feature support
324   328    * O TLV 3 */
325   329   typedef struct ogs_nas_eps_network_feature_support_s {
326   330       uint8_t length;
353   357   } __attribute__ ((packed)) ogs_nas_eps_update_result_t;
354   358
355   359   /* 9.9.3.14 EPS update type
356       -  * M V 1/2
357       -  * 9.9.3.21 NAS key set identifier
      360 +  * M V 1/2
```

```
                361 +  * 9.9.3.21 NAS key set identifier
358             362    * M V 1/2 */
359             363   #define OGS_NAS_EPS_UPDATE_TYPE_TA_UPDATING                                0
360             364   #define OGS_NAS_EPS_UPDATE_TYPE_COMBINED_TA_LA_UPDATING                    1
430             434       uint8_t spare:3;)
431             435   } __attribute__ ((packed)) ogs_nas_ms_network_capability_t;
432             436
433               - /* 9.9.3.20A MS network feature support
                437 + /* 9.9.3.20A MS network feature support
434             438    * See subclause 10.5.1.15 in 3GPP TS 24.008 [13].
435             439    * O TV 1 */
436             440   typedef struct ogs_nas_ms_network_feature_support_s {
470             474       uint8_t identity:1;)
471             475   } ogs_nas_paging_identity_t;
472             476
473               - /* 9.9.3.26 P-TMSI signature
                477 + /* 9.9.3.26 P-TMSI signature
474             478    * See subclause 10.5.5.8 in 3GPP TS 24.008
475             479    * O TV 4 */
476             480   typedef uint32_t ogs_nas_p_tmsi_signature_t; /* TV : 4bytes */
477             481
478               - /* 9.9.3.26A Extended EMM cause
                482 + /* 9.9.3.26A Extended EMM cause
479             483    * O TV 1 */
480             484   typedef struct ogs_nas_extended_emm_cause_s {
481             485   ED4(uint8_t type:4;,
485             489   } ogs_nas_extended_emm_cause_t;
486             490
487             491   /* 9.9.3.27 Service type
488               -  * M V 1/2
489               -  * 9.9.3.21 NAS key set identifier
                492 +  * M V 1/2
                493 +  * 9.9.3.21 NAS key set identifier
490             494    * M V 1/2 */
491             495   #define OGS_NAS_SERVICE_TYPE_CS_FALLBACK_FROM_UE 0
492             496   #define OGS_NAS_SERVICE_TYPE_CS_FALLBACK_TO_UE 1
540             544       /*
541             545        * Do not change 'ogs_eps_tai_t' to 'ogs_nas_tracking_area_identity_t'.
542             546        * Use 'ogs_eps_tai_t' for easy implementation.
543               -       * ogs_nas_tai_list_build() changes to NAS
                547 +       * ogs_nas_tai_list_build() changes to NAS
544             548        * format(ogs_nas_tracking_area_identity_t)
545             549        * and is sent to the UE.
546             550        */
565             569   } __attribute__ ((packed)) ogs_nas_ue_radio_capability_information_update_needed_t;
566             570
567             571   /* 9.9.3.38 CLI
568               -  * O TLV 3-14
569               -  * The coding of the CLI value part is the same as for
570               -  * octets 3 to 14 of the Calling party BCD number information element
                572 +  * O TLV 3-14
                573 +  * The coding of the CLI value part is the same as for
                574 +  * octets 3 to 14 of the Calling party BCD number information element
571             575    * defined in subclause 10.5.4.9 of 3GPP TS 24.008 [13]. */
572             576   #define NAX_MAX_CLI_LEN 12
573             577   typedef struct ogs_nas_cli_s {
576             580   } __attribute__ ((packed)) ogs_nas_cli_t;
577             581
578             582   /* 9.9.3.39 SS Code
579               -  * O TV 2
580               -  * The coding of the SS Code value is given in subclause 17.7.5 of
                583 +  * O TV 2
                584 +  * The coding of the SS Code value is given in subclause 17.7.5 of
581             585    * 3GPP TS 29.002 [15C] */
582             586   typedef uint8_t ogs_nas_ss_code_t;
583             587
613             617   typedef struct ogs_nas_voice_domain_preference_and_ue_usage_setting_s {
```

```
614  618       uint8_t length;
615  619     ED3(uint8_t spare:5;,
616          -    uint8_t ue_usage_setting:1;,
     620     +    uint8_t ue_usage_setting:1;,
617  621       uint8_t voice_domain_preference_for_e_utran:2;)
618  622     } __attribute__ ((packed)) ogs_nas_voice_domain_preference_and_ue_usage_setting_t;
619  623
620          - /* 9.9.3.45 GUTI type
     624     + /* 9.9.3.45 GUTI type
621  625      * O TV 1 */
622  626     typedef struct ogs_nas_guti_type_s {
623  627     ED3(uint8_t type:4;,
767  771     } __attribute__ ((packed)) ogs_nas_connectivity_type_t;
768  772
769  773     /* 9.9.4.4 ESM cause
770          -  * M V 1
771          -  * Annex B (informative) Cause values for EPS session management
     774     +  * M V 1
     775     +  * Annex B (informative) Cause values for EPS session management
772  776       B.1 Causes related to nature of request */
773          - #define ESM_CAUSE_OPERATOR_DETERMINED_BARRING 8
774          - #define ESM_CAUSE_INSUFFICIENT_RESOURCES 26
775          - #define ESM_CAUSE_MISSING_OR_UNKNOWN_APN 27
776          - #define ESM_CAUSE_UNKNOWN_PDN_TYPE 28
777          - #define ESM_CAUSE_USER_AUTHENTICATION_FAILED 29
778          - #define ESM_CAUSE_REQUEST_REJECTED_BY_SERVING_GW_OR_PDN_GW 30
779          - #define ESM_CAUSE_REQUEST_REJECTED_UNSPECIFIED 31
780          - #define ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED 32
781          - #define ESM_CAUSE_REQUESTED_SERVICE_OPTION_NOT_SUBSCRIBED 33
782          - #define ESM_CAUSE_SERVICE_OPTION_TEMPORARILY_OUT_OF_ORDER 34
783          - #define ESM_CAUSE_PTI_ALREADY_IN_USE 35
784          - #define ESM_CAUSE_REGULAR_DEACTIVATION 36
785          - #define ESM_CAUSE_EPS_QOS_NOT_ACCEPTED 37
786          - #define ESM_CAUSE_NETWORK_FAILURE 38
787          - #define ESM_CAUSE_REACTIVATION_REQUESTED 39
788          - #define ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION  41
789          - #define ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION 42
790          - #define ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY 43
791          - #define ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS 44
792          - #define ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS 45
793          - #define ESM_CAUSE_PTI_MISMATCH 47
794          - #define ESM_CAUSE_LAST_PDN_DISCONNECTION_NOT_ALLOWED 49
795          - #define ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED 50
796          - #define ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED 51
797          - #define ESM_CAUSE_SINGLE_ADDRESS_BEARERS_ONLY_ALLOWED 52
798          - #define ESM_CAUSE_ESM_INFORMATION_NOT_RECEIVED 53
799          - #define ESM_CAUSE_PDN_CONNECTION_DOES_NOT_EXIST 54
800          - #define ESM_CAUSE_MULTIPLE_PDN_CONNECTIONS_FOR_A_GIVEN_APN_NOT_ALLOWED 55
801          - #define ESM_CAUSE_COLLISION_WITH_NETWORK_INITIATED_REQUEST 56
802          - #define ESM_CAUSE_PDN_TYPE_IPV4V6_ONLY_ALLOWED 57
803          - #define ESM_CAUSE_PDN_TYPE_NON_IP_ONLY_ALLOWED 58
804          - #define ESM_CAUSE_UNSUPPORTED_QCI_VALUE 59
805          - #define ESM_CAUSE_BEARER_HANDLING_NOT_SUPPORTED 60
806          - #define ESM_CAUSE_MAXIMUM_NUMBER_OF_EPS_BEARERS_REACHED 65
807          - #define ESM_CAUSE_REQUESTED_APN_NOT_SUPPORTED_IN_CURRENT_RAT_AND_PLMN_COMBINATION 66
808          - #define ESM_CAUSE_INVALID_PTI_VALUE 81
809          - #define ESM_CAUSE_APN_RESTRICTION_VALUE_INCOMPATIBLE_WITH_ACTIVE_EPS_BEARER_CONTEXT 112
810          - #define ESM_CAUSE_MULTIPLE_ACCESSES_TO_A_PDN_CONNECTION_NOT_ALLOWED 113
     777     + #define OGS_NAS_ESM_CAUSE_OPERATOR_DETERMINED_BARRING 8
     778     + #define OGS_NAS_ESM_CAUSE_INSUFFICIENT_RESOURCES 26
     779     + #define OGS_NAS_ESM_CAUSE_MISSING_OR_UNKNOWN_APN 27
     780     + #define OGS_NAS_ESM_CAUSE_UNKNOWN_PDN_TYPE 28
     781     + #define OGS_NAS_ESM_CAUSE_USER_AUTHENTICATION_FAILED 29
     782     + #define OGS_NAS_ESM_CAUSE_REQUEST_REJECTED_BY_SERVING_GW_OR_PDN_GW 30
     783     + #define OGS_NAS_ESM_CAUSE_REQUEST_REJECTED_UNSPECIFIED 31
     784     + #define OGS_NAS_ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED 32
     785     + #define OGS_NAS_ESM_CAUSE_REQUESTED_SERVICE_OPTION_NOT_SUBSCRIBED 33
```

```
+ #define OGS_NAS_ESM_CAUSE_SERVICE_OPTION_TEMPORARILY_OUT_OF_ORDER 34
+ #define OGS_NAS_ESM_CAUSE_PTI_ALREADY_IN_USE 35
+ #define OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION 36
+ #define OGS_NAS_ESM_CAUSE_EPS_QOS_NOT_ACCEPTED 37
+ #define OGS_NAS_ESM_CAUSE_NETWORK_FAILURE 38
+ #define OGS_NAS_ESM_CAUSE_REACTIVATION_REQUESTED 39
+ #define OGS_NAS_ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION  41
+ #define OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION 42
+ #define OGS_NAS_ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY 43
+ #define OGS_NAS_ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS 44
+ #define OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS 45
+ #define OGS_NAS_ESM_CAUSE_PTI_MISMATCH 47
+ #define OGS_NAS_ESM_CAUSE_LAST_PDN_DISCONNECTION_NOT_ALLOWED 49
+ #define OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED 50
+ #define OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED 51
+ #define OGS_NAS_ESM_CAUSE_SINGLE_ADDRESS_BEARERS_ONLY_ALLOWED 52
+ #define OGS_NAS_ESM_CAUSE_ESM_INFORMATION_NOT_RECEIVED 53
+ #define OGS_NAS_ESM_CAUSE_PDN_CONNECTION_DOES_NOT_EXIST 54
+ #define OGS_NAS_ESM_CAUSE_MULTIPLE_PDN_CONNECTIONS_FOR_A_GIVEN_APN_NOT_ALLOWED 55
+ #define OGS_NAS_ESM_CAUSE_COLLISION_WITH_NETWORK_INITIATED_REQUEST 56
+ #define OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV4V6_ONLY_ALLOWED 57
+ #define OGS_NAS_ESM_CAUSE_PDN_TYPE_NON_IP_ONLY_ALLOWED 58
+ #define OGS_NAS_ESM_CAUSE_UNSUPPORTED_QCI_VALUE 59
+ #define OGS_NAS_ESM_CAUSE_BEARER_HANDLING_NOT_SUPPORTED 60
+ #define OGS_NAS_ESM_CAUSE_MAXIMUM_NUMBER_OF_EPS_BEARERS_REACHED 65
+ #define OGS_NAS_ESM_CAUSE_REQUESTED_APN_NOT_SUPPORTED_IN_CURRENT_RAT_AND_PLMN_COMBINATION 66
+ #define OGS_NAS_ESM_CAUSE_INVALID_PTI_VALUE 81
+ #define OGS_NAS_ESM_CAUSE_APN_RESTRICTION_VALUE_INCOMPATIBLE_WITH_ACTIVE_EPS_BEARER_CONTEXT 112
+ #define OGS_NAS_ESM_CAUSE_MULTIPLE_ACCESSES_TO_A_PDN_CONNECTION_NOT_ALLOWED 113
  /* B.2 Protocol errors (e.g., unknown message) class */
- #define ESM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
- #define ESM_CAUSE_INVALID_MANDATORY_INFORMATION 96
- #define ESM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
- #define ESM_CAUSE_MESSAGE_TYPE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 98
- #define ESM_CAUSE_INFORMATION_ELEMENT_NON_EXISTENT_OR_NOT_IMPLEMENTED 99
- #define ESM_CAUSE_CONDITIONAL_IE_ERROR 100
- #define ESM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 101
- #define ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED 111
+ #define OGS_NAS_ESM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE 95
+ #define OGS_NAS_ESM_CAUSE_INVALID_MANDATORY_INFORMATION 96
+ #define OGS_NAS_ESM_CAUSE_MESSAGE_TYPE_NON_EXISTENT_OR_NOT_IMPLEMENTED 97
+ #define OGS_NAS_ESM_CAUSE_MESSAGE_TYPE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 98
+ #define OGS_NAS_ESM_CAUSE_INFORMATION_ELEMENT_NON_EXISTENT_OR_NOT_IMPLEMENTED 99
+ #define OGS_NAS_ESM_CAUSE_CONDITIONAL_IE_ERROR 100
+ #define OGS_NAS_ESM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE 101
+ #define OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED 111
  typedef uint8_t ogs_nas_esm_cause_t;

  /* 9.9.4.5 ESM information transfer flag

  /* 9.9.4.11 Protocol configuration options
   * See subclause 10.5.6.3 in 3GPP TS 24.008 [13].
-  * O TLV 3-253 */
+  * O TLV 3-253 */
  typedef struct ogs_nas_protocol_configuration_options_s {
      uint8_t length;
      uint8_t buffer[OGS_MAX_PCO_LEN];
```

99 src/amf/gmm-handler.c

```
  #undef OGS_LOG_DOMAIN
  #define OGS_LOG_DOMAIN __gmm_log_domain

- static int gmm_handle_nas_message_container(
+ static ogs_nas_5gmm_cause_t gmm_handle_nas_message_container(
          amf_ue_t *amf_ue, uint8_t message_type,
```

```
33  33          ogs_nas_message_container_t *nas_message_container);
34  34
35      - int gmm_handle_registration_request(amf_ue_t *amf_ue,
        35  + ogs_nas_5gmm_cause_t gmm_handle_registration_request(amf_ue_t *amf_ue,
36  36          ogs_nas_security_header_type_t h, NGAP_ProcedureCode_t ngap_code,
37  37          ogs_nas_5gs_registration_request_t *registration_request)
38  38  {
112 112          ~OGS_REGISTRATION_CLEARTEXT_PRESENT) {
113 113          ogs_error("Non cleartext IEs is included [0x%llx]",
114 114                  (long long)registration_request->presencemask);
115     -        ogs_assert(OGS_OK ==
116     -            nas_5gs_send_registration_reject(amf_ue,
117     -                OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE));
118     -        return OGS_ERROR;
        115 +        return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
119 116      }
120 117
121 118      if (!h.integrity_protected &&
122 119          (registration_request->presencemask &
123 120          OGS_NAS_5GS_REGISTRATION_REQUEST_NAS_MESSAGE_CONTAINER_PRESENT)) {
124 121          ogs_error("NAS container present without Integrity-protected");
125     -        ogs_assert(OGS_OK ==
126     -            nas_5gs_send_registration_reject(amf_ue,
127     -                OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE));
128     -        return OGS_ERROR;
        122 +        return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
129 123      }
130 124
131 125      if (!mobile_identity->length || !mobile_identity->buffer) {
132 126          ogs_error("No Mobile Identity");
133     -        ogs_assert(OGS_OK ==
134     -            nas_5gs_send_registration_reject(amf_ue,
135     -                OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE));
136     -        return OGS_ERROR;
        127 +        return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
137 128      }
138 129
139 130      mobile_identity_header =
152 143                  OGS_NAS_5GS_ECIES_SCHEME_PROFILE_B) {
153 144              ogs_error("Invalid ProtectionSchemeID(%d) in SUCI",
154 145                  mobile_identity_suci->protection_scheme_id);
155     -            ogs_assert(OGS_OK ==
156     -                nas_5gs_send_registration_reject(amf_ue,
157     -                    OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE)
158     -            );
159     -            return OGS_ERROR;
        146 +            return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
160 147          }
161 148          amf_ue_set_suci(amf_ue, mobile_identity);
162 149          ogs_info("[%s]    SUCI", amf_ue->suci);
166 153              (ogs_nas_5gs_mobile_identity_guti_t *)mobile_identity->buffer;
167 154          if (!mobile_identity_guti) {
168 155              ogs_error("No mobile identity");
169     -            return OGS_ERROR;
        156 +            return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
170 157          }
171 158
172 159          ogs_nas_5gs_mobile_identity_guti_to_nas_guti(
262 249      /* Check TAI */
263 250      served_tai_index = amf_find_served_tai(&amf_ue->nr_tai);
264 251      if (served_tai_index < 0) {
265     -        /* Send Registration Reject */
266     -        ogs_warn("Cannot find Served TAI[PLMN_ID:%06x,TAC:%d]",
        252 +        ogs_error("Cannot find Served TAI[PLMN_ID:%06x,TAC:%d]",
267 253              ogs_plmn_id_hexdump(&amf_ue->nr_tai.plmn_id), amf_ue->nr_tai.tac.v);
268     -        ogs_assert(OGS_OK ==
269     -            nas_5gs_send_registration_reject(amf_ue,
```

```
270       -                 OGS_5GMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
271       -         return OGS_ERROR;
          254 +         return OGS_5GMM_CAUSE_TRACKING_AREA_NOT_ALLOWED;
272       255       }
273       256       ogs_debug("    SERVED_TAI_INDEX[%d]", served_tai_index);
274       257
285       268           ogs_error("[UE:0x%x:0x%x], NEA0 can be used in Encrypt[0x%x], "
286       269               "but Integrity[0x%x] cannot be bypassed with NIA0",
287       270               ue_security_capability->nr_ea, ue_security_capability->nr_ia,
288       -               amf_selected_enc_algorithm(amf_ue),
          271 +               amf_selected_enc_algorithm(amf_ue),
289       272               amf_selected_int_algorithm(amf_ue));
290       -         ogs_assert(OGS_OK ==
291       -             nas_5gs_send_registration_reject(amf_ue,
292       -                 OGS_5GMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH));
293       -         return OGS_ERROR;
          273 +         return OGS_5GMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH;
294       274       }
295       275
296       -     return OGS_OK;
          276 +     return OGS_5GMM_CAUSE_REQUEST_ACCEPTED;
297       277   }
298       278
299       -   int gmm_handle_registration_update(amf_ue_t *amf_ue,
          279 +   ogs_nas_5gmm_cause_t gmm_handle_registration_update(amf_ue_t *amf_ue,
300       280           ogs_nas_5gs_registration_request_t *registration_request)
301       281   {
302       282       amf_sess_t *sess = NULL;
376       356                           amf_ue->requested_nssai.s_nssai[i].sst,
377       357                           amf_ue->requested_nssai.s_nssai[i].sd.v);
378       358               }
379       -             return OGS_ERROR;
          359 +             return OGS_5GMM_CAUSE_NO_NETWORK_SLICES_AVAILABLE;
380       360           }
381       361       }
382       362
442       422           }
443       423       }
444       424
445       -     return OGS_OK;
          425 +     return OGS_5GMM_CAUSE_REQUEST_ACCEPTED;
446       426   }
447       427
448       -   int gmm_handle_service_request(amf_ue_t *amf_ue,
          428 +   ogs_nas_5gmm_cause_t gmm_handle_service_request(amf_ue_t *amf_ue,
449       429           ogs_nas_security_header_type_t h, NGAP_ProcedureCode_t ngap_code,
450       430           ogs_nas_5gs_service_request_t *service_request)
451       431   {
483       463           service_request->presencemask & ~OGS_SERVICE_CLEARTEXT_PRESENT) {
484       464           ogs_error("Non cleartext IEs is included [0x%llx]",
485       465                   (long long)service_request->presencemask);
486       -         ogs_assert(OGS_OK ==
487       -             nas_5gs_send_service_reject(amf_ue,
488       -                 OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE));
489       -         return OGS_ERROR;
          466 +         return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
490       467       }
491       468
492       469       if (!h.integrity_protected &&
493       470           (service_request->presencemask &
494       471           OGS_NAS_5GS_SERVICE_REQUEST_NAS_MESSAGE_CONTAINER_PRESENT)) {
495       472           ogs_error("NAS container present without Integrity-protected");
496       -         ogs_assert(OGS_OK ==
497       -             nas_5gs_send_service_reject(amf_ue,
498       -                 OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE));
499       -         return OGS_ERROR;
          473 +         return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
```

```
500  474        }
501  475
502  476        amf_ue->nas.message_type = OGS_NAS_5GS_SERVICE_REQUEST;
550  524        /* Check TAI */
551  525        served_tai_index = amf_find_served_tai(&amf_ue->nr_tai);
552  526        if (served_tai_index < 0) {
553       -        /* Send Registration Reject */
554       -        ogs_warn("Cannot find Served TAI[PLMN_ID:%06x,TAC:%d]",
     527  +        ogs_error("Cannot find Served TAI[PLMN_ID:%06x,TAC:%d]",
555  528                ogs_plmn_id_hexdump(&amf_ue->nr_tai.plmn_id), amf_ue->nr_tai.tac.v);
556       -        ogs_assert(OGS_OK ==
557       -            nas_5gs_send_registration_reject(amf_ue,
558       -                OGS_5GMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
559       -        return OGS_ERROR;
     529  +        return OGS_5GMM_CAUSE_TRACKING_AREA_NOT_ALLOWED;
560  530        }
561  531        ogs_debug("    SERVED_TAI_INDEX[%d]", served_tai_index);
562  532
565  535            ogs_amf_id_hexdump(&amf_ue->current.guti.amf_id),
566  536            amf_ue->current.guti.m_tmsi);
567  537
568       -    return OGS_OK;
     538  +    return OGS_5GMM_CAUSE_REQUEST_ACCEPTED;
569  539    }
570  540
571       - int gmm_handle_service_update(amf_ue_t *amf_ue,
     541  + ogs_nas_5gmm_cause_t gmm_handle_service_update(amf_ue_t *amf_ue,
572  542            ogs_nas_5gs_service_request_t *service_request)
573  543    {
574  544        amf_sess_t *sess = NULL;
663  633            ogs_assert(OGS_OK ==
664  634                nas_5gs_send_service_accept(amf_ue));
665  635
666       -    return OGS_OK;
     636  +    return OGS_5GMM_CAUSE_REQUEST_ACCEPTED;
667  637    }
668  638
669  639    int gmm_handle_deregistration_request(amf_ue_t *amf_ue,
800  770        return OGS_OK;
801  771    }
802  772
803       - int gmm_handle_security_mode_complete(amf_ue_t *amf_ue,
     773  + ogs_nas_5gmm_cause_t gmm_handle_security_mode_complete(amf_ue_t *amf_ue,
804  774        ogs_nas_5gs_security_mode_complete_t *security_mode_complete)
805  775    {
806  776        ogs_nas_5gs_mobile_identity_t *imeisv = NULL;
829  799            OGS_NAS_5GS_SECURITY_MODE_COMPLETE_NAS_MESSAGE_CONTAINER_PRESENT)
830  800                == 0) {
831  801            ogs_error("No NAS Message Container in Security mode complete message");
832       -        return OGS_ERROR;
     802  +        return OGS_5GMM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_THE_PROTOCOL_STATE;
833  803        }
834  804
835  805        if (security_mode_complete->presencemask &
886  856                &security_mode_complete->nas_message_container);
887  857        }
888  858
889       -    return OGS_OK;
     859  +    return OGS_5GMM_CAUSE_REQUEST_ACCEPTED;
890  860    }
891  861
892  862    int gmm_handle_ul_nas_transport(amf_ue_t *amf_ue,
1212 1182        return OGS_OK;
1213 1183    }
1214 1184
1215      - static int gmm_handle_nas_message_container(
     1185 + static ogs_nas_5gmm_cause_t gmm_handle_nas_message_container(
```

```
1216 1186          amf_ue_t *amf_ue, uint8_t message_type,
1217 1187          ogs_nas_message_container_t *nas_message_container)
1218 1188  {
1219      -     int rv = OGS_ERROR;
     1189 +     int gmm_cause;
1220 1190
1221 1191      ogs_pkbuf_t *nasbuf = NULL;
1222 1192      ogs_nas_5gs_message_t nas_message;
1227 1197      if (!nas_message_container->buffer || !nas_message_container->length) {
1228 1198          ogs_error("No NAS message container [%p:%d]",
1229 1199              nas_message_container->buffer,nas_message_container->length);
1230      -         return OGS_ERROR;
     1200 +         return OGS_5GMM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_THE_PROTOCOL_STATE;
1231 1201      }
1232 1202
1233 1203      nasbuf = ogs_pkbuf_alloc(NULL, nas_message_container->length);
1269 1239      if (ogs_nas_5gmm_decode(&nas_message, nasbuf) != OGS_OK) {
1270 1240          ogs_error("ogs_nas_5gmm_decode() failed");
1271 1241          ogs_pkbuf_free(nasbuf);
1272      -         return OGS_ERROR;
     1242 +         return OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
1273 1243      }
1274 1244
     1245 +     gmm_cause = OGS_5GMM_CAUSE_SEMANTICALLY_INCORRECT_MESSAGE;
     1246 +
1275 1247      switch (nas_message.gmm.h.message_type) {
1276 1248          case OGS_NAS_5GS_REGISTRATION_REQUEST:
1277 1249              ogs_debug("Registration request in NAS message container");
1278      -             rv = gmm_handle_registration_update(
     1250 +             gmm_cause = gmm_handle_registration_update(
1279 1251                      amf_ue, &nas_message.gmm.registration_request);
1280 1252              break;
1281 1253          case OGS_NAS_5GS_SERVICE_REQUEST:
1282 1254              ogs_debug("Service request in NAS message container");
1283      -             rv = gmm_handle_service_update(
     1255 +             gmm_cause = gmm_handle_service_update(
1284 1256                      amf_ue, &nas_message.gmm.service_request);
1285 1257              break;
1286 1258          default:
1287 1259              ogs_error("Unknown message [%d]", nas_message.gmm.h.message_type);
1288      -             rv = OGS_ERROR;
1289 1260      }
1290 1261
1291 1262      ogs_pkbuf_free(nasbuf);
1292      -     return rv;
     1263 +     return gmm_cause;
1293 1264  }
```

10 src/amf/gmm-handler.h

```
26 26  extern "C" {
27 27  #endif
28 28
29      - int gmm_handle_registration_request(amf_ue_t *amf_ue,
     29 + ogs_nas_5gmm_cause_t gmm_handle_registration_request(amf_ue_t *amf_ue,
30 30          ogs_nas_security_header_type_t h, NGAP_ProcedureCode_t ngap_code,
31 31          ogs_nas_5gs_registration_request_t *registration_request);
32      - int gmm_handle_registration_update(amf_ue_t *amf_ue,
     32 + ogs_nas_5gmm_cause_t gmm_handle_registration_update(amf_ue_t *amf_ue,
33 33          ogs_nas_5gs_registration_request_t *registration_request);
34 34
35      - int gmm_handle_service_request(amf_ue_t *amf_ue,
     35 + ogs_nas_5gmm_cause_t gmm_handle_service_request(amf_ue_t *amf_ue,
36 36          ogs_nas_security_header_type_t h, NGAP_ProcedureCode_t ngap_code,
37 37          ogs_nas_5gs_service_request_t *service_request);
38      - int gmm_handle_service_update(amf_ue_t *amf_ue,
```

```
+ ogs_nas_5gmm_cause_t gmm_handle_service_update(amf_ue_t *amf_ue,
          ogs_nas_5gs_service_request_t *service_request);

  int gmm_handle_deregistration_request(amf_ue_t *amf_ue,
  int gmm_handle_identity_response(amf_ue_t *amf_ue,
          ogs_nas_5gs_identity_response_t *identity_response);

- int gmm_handle_security_mode_complete(amf_ue_t *amf_ue,
+ ogs_nas_5gmm_cause_t gmm_handle_security_mode_complete(amf_ue_t *amf_ue,
          ogs_nas_5gs_security_mode_complete_t *security_mode_complete);

  int gmm_handle_ul_nas_transport(amf_ue_t *amf_ue,
```

106 src/amf/gmm-sm.c

```
  static void common_register_state(ogs_fsm_t *s, amf_event_t *e)
  {
      int rv, xact_count = 0;
+     ogs_nas_5gmm_cause_t gmm_cause;

      amf_ue_t *amf_ue = NULL;
      amf_sess_t *sess = NULL;
      ogs_sbi_message_t *sbi_message = NULL;

      ogs_assert(e);
-
+
      if (e->sess) {
          sess = e->sess;
          amf_ue = sess->amf_ue;
          switch (nas_message->gmm.h.message_type) {
          case OGS_NAS_5GS_REGISTRATION_REQUEST:
              ogs_info("Registration request");
-             rv = gmm_handle_registration_request(
+             gmm_cause = gmm_handle_registration_request(
                      amf_ue, h, e->ngap.code,
                      &nas_message->gmm.registration_request);
-             if (rv != OGS_OK) {
-                 ogs_error("gmm_handle_registration_request() failed");
+             if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
+                 ogs_error("gmm_handle_registration_request() failed [%d]",
+                             gmm_cause);
+                 ogs_assert(OGS_OK ==
+                     nas_5gs_send_registration_reject(amf_ue, gmm_cause));
                  OGS_FSM_TRAN(s, gmm_state_exception);
                  break;
              }

              if (h.integrity_protected && SECURITY_CONTEXT_IS_VALID(amf_ue)) {

-                 rv = gmm_handle_registration_update(
+                 gmm_cause = gmm_handle_registration_update(
                          amf_ue, &nas_message->gmm.registration_request);
-                 if (rv != OGS_OK) {
-                     ogs_error("gmm_handle_registration_update() failed");
+                 if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
+                     ogs_error("[%s] gmm_handle_registration_update() "
+                                 "failed [%d]", amf_ue->suci, gmm_cause);
+                     ogs_assert(OGS_OK ==
+                         nas_5gs_send_registration_reject(amf_ue, gmm_cause));
                      OGS_FSM_TRAN(s, gmm_state_exception);
                      break;
                  }
          case OGS_NAS_5GS_SERVICE_REQUEST:
              ogs_info("Service request");

```

```
207     -             rv = gmm_handle_service_request(
        214 +         gmm_cause = gmm_handle_service_request(
208     215                       amf_ue, h, e->ngap.code, &nas_message->gmm.service_request);
209     -             if (rv != OGS_OK) {
210     -                 ogs_error("gmm_handle_service_request() failed");
        216 +         if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
        217 +             ogs_error("[%s] gmm_handle_service_request() failed [%d]",
        218 +                       amf_ue->suci, gmm_cause);
        219 +             ogs_assert(OGS_OK ==
        220 +                 nas_5gs_send_service_reject(amf_ue, gmm_cause));
211     221               OGS_FSM_TRAN(s, gmm_state_exception);
212     222               break;
213     223           }
232     242               break;
233     243           }
234     244
235     -             rv = gmm_handle_service_update(
        245 +         gmm_cause = gmm_handle_service_update(
236     246                       amf_ue, &nas_message->gmm.service_request);
237     -             if (rv != OGS_OK) {
238     -                 ogs_error("gmm_handle_service_update() failed");
        247 +         if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
        248 +             ogs_error("[%s] gmm_handle_service_update() failed [%d]",
        249 +                       amf_ue->suci, gmm_cause);
        250 +             ogs_assert(OGS_OK ==
        251 +                 nas_5gs_send_service_reject(amf_ue, gmm_cause));
239     252               OGS_FSM_TRAN(s, gmm_state_exception);
240     253           }
241     254
495     508   void gmm_state_authentication(ogs_fsm_t *s, amf_event_t *e)
496     509   {
497     510       int rv;
        511 +     ogs_nas_5gmm_cause_t gmm_cause;
        512 +
498     513       amf_ue_t *amf_ue = NULL;
499     514       amf_sess_t *sess = NULL;
500     515
510     525
511     526       ogs_assert(s);
512     527       ogs_assert(e);
513     -
        528 +
514     529       amf_sm_debug(e);
515     530
516     531       if (e->sess) {
602     617               break;
603     618           case OGS_NAS_5GS_REGISTRATION_REQUEST:
604     619               ogs_warn("Registration request");
605     -             rv = gmm_handle_registration_request(
        620 +         gmm_cause = gmm_handle_registration_request(
606     621                       amf_ue, h, e->ngap.code,
607     622                       &nas_message->gmm.registration_request);
608     -             if (rv != OGS_OK) {
609     -                 ogs_error("[%s] gmm_handle_registration_request() failed",
610     -                           amf_ue->suci);
        623 +         if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
        624 +             ogs_error("[%s] gmm_handle_registration_request() failed [%d]",
        625 +                       amf_ue->suci, gmm_cause);
        626 +             ogs_assert(OGS_OK ==
        627 +                 nas_5gs_send_registration_reject(amf_ue, gmm_cause));
611     628               OGS_FSM_TRAN(s, gmm_state_exception);
612     629               break;
613     630           }
744     761   void gmm_state_security_mode(ogs_fsm_t *s, amf_event_t *e)
745     762   {
746     763       int rv;
        764 +     ogs_nas_5gmm_cause_t gmm_cause;
```

```
747  765       amf_ue_t *amf_ue = NULL;
748  766       ogs_nas_5gs_message_t *nas_message = NULL;
749  767       ogs_nas_security_header_type_t h;
797  815                   break;
798  816               }
799  817
800       -             rv = gmm_handle_security_mode_complete(
     818  +             gmm_cause = gmm_handle_security_mode_complete(
801  819                       amf_ue, &nas_message->gmm.security_mode_complete);
802       -             if (rv != OGS_OK) {
803       -                 ogs_error("[%s] Cannot handle NAS message", amf_ue->suci);
     820  +             if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
     821  +                 ogs_error("[%s] gmm_handle_security_mode_complete() "
     822  +                             "failed [%d] in type [%d]",
     823  +                             amf_ue->suci, gmm_cause, amf_ue->nas.message_type);
804  824                   ogs_assert(OGS_OK ==
805       -                     nas_5gs_send_gmm_reject(amf_ue,
806       -                         OGS_5GMM_CAUSE_5GS_SERVICES_NOT_ALLOWED));
     825  +                     nas_5gs_send_gmm_reject(amf_ue, gmm_cause));
807  826                   OGS_FSM_TRAN(s, gmm_state_exception);
808  827                   break;
809  828               }
837  856               break;
838  857           case OGS_NAS_5GS_REGISTRATION_REQUEST:
839  858               ogs_warn("Registration request");
840       -             rv = gmm_handle_registration_request(
     859  +             gmm_cause = gmm_handle_registration_request(
841  860                       amf_ue, h, e->ngap.code,
842  861                       &nas_message->gmm.registration_request);
843       -             if (rv != OGS_OK) {
844       -                 ogs_error("[%s] Cannot handle NAS message", amf_ue->suci);
     862  +             if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
     863  +                 ogs_error("[%s] gmm_handle_registration_request() failed [%d]",
     864  +                             amf_ue->suci, gmm_cause);
     865  +                 ogs_assert(OGS_OK ==
     866  +                     nas_5gs_send_registration_reject(amf_ue, gmm_cause));
845  867                   OGS_FSM_TRAN(s, gmm_state_exception);
846  868                   break;
847  869               }
916  938   void gmm_state_initial_context_setup(ogs_fsm_t *s, amf_event_t *e)
917  939   {
918  940       int rv, xact_count = 0;
     941  +     ogs_nas_5gmm_cause_t gmm_cause;
     942  +
919  943       amf_ue_t *amf_ue = NULL;
920  944       amf_sess_t *sess = NULL;
921  945       ogs_nas_5gs_message_t *nas_message = NULL;
1156 1180
1157 1181           case OGS_NAS_5GS_REGISTRATION_REQUEST:
1158 1182               ogs_warn("Registration request");
1159      -             rv = gmm_handle_registration_request(
     1183  +             gmm_cause = gmm_handle_registration_request(
1160 1184                       amf_ue, h, e->ngap.code,
1161 1185                       &nas_message->gmm.registration_request);
1162      -             if (rv != OGS_OK) {
1163      -                 ogs_error("[%s] Cannot handle NAS message", amf_ue->suci);
     1186  +             if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
     1187  +                 ogs_error("[%s] gmm_handle_registration_request() failed [%d]",
     1188  +                             amf_ue->suci, gmm_cause);
     1189  +                 ogs_assert(OGS_OK ==
     1190  +                     nas_5gs_send_registration_reject(amf_ue, gmm_cause));
1164 1191                   OGS_FSM_TRAN(s, gmm_state_exception);
1165 1192                   break;
1166 1193               }
1236 1263
1237 1264   void gmm_state_exception(ogs_fsm_t *s, amf_event_t *e)
1238 1265   {
```

```
1239      -     int rv, xact_count = 0;
      1266  +     int xact_count = 0;
      1267  +     ogs_nas_5gmm_cause_t gmm_cause;
1240  1268
1241  1269      amf_ue_t *amf_ue = NULL;
1242  1270      amf_sess_t *sess = NULL;
1289  1317          switch (nas_message->gmm.h.message_type) {
1290  1318          case OGS_NAS_5GS_REGISTRATION_REQUEST:
1291  1319              ogs_info("Registration request");
1292      -           rv = gmm_handle_registration_request(
      1320  +           gmm_cause = gmm_handle_registration_request(
1293  1321                      amf_ue, h, e->ngap.code,
1294  1322                      &nas_message->gmm.registration_request);
1295      -           if (rv != OGS_OK) {
1296      -               ogs_error("gmm_handle_registration_request() failed");
      1323  +           if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
      1324  +               ogs_error("gmm_handle_registration_request() failed [%d]",
      1325  +                           gmm_cause);
      1326  +               ogs_assert(OGS_OK ==
      1327  +                   nas_5gs_send_registration_reject(amf_ue, gmm_cause));
1297  1328                  OGS_FSM_TRAN(s, gmm_state_exception);
1298  1329                  break;
1299  1330              }
1309  1340
1310  1341              if (h.integrity_protected && SECURITY_CONTEXT_IS_VALID(amf_ue)) {
1311  1342
1312      -               rv = gmm_handle_registration_update(
      1343  +               gmm_cause = gmm_handle_registration_update(
1313  1344                          amf_ue, &nas_message->gmm.registration_request);
1314      -               if (rv != OGS_OK) {
1315      -                   ogs_error("gmm_handle_registration_update() failed");
      1345  +               if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
      1346  +                   ogs_error("[%s] gmm_handle_registration_update() "
      1347  +                               "failed [%d]", amf_ue->suci, gmm_cause);
      1348  +                   ogs_assert(OGS_OK ==
      1349  +                       nas_5gs_send_registration_reject(amf_ue, gmm_cause));
1316  1350                      OGS_FSM_TRAN(s, gmm_state_exception);
1317  1351                      break;
1318  1352                  }
```

3 src/mme/emm-build.c

```
119  119              (eps_attach_result->result == OGS_NAS_ATTACH_TYPE_EPS_ATTACH)) {
120  120              attach_accept->presencemask |=
121  121                  OGS_NAS_EPS_ATTACH_ACCEPT_EMM_CAUSE_PRESENT;
122      -           attach_accept->emm_cause = EMM_CAUSE_CS_DOMAIN_NOT_AVAILABLE;
     122  +           attach_accept->emm_cause =
     123  +               OGS_NAS_EMM_CAUSE_CS_DOMAIN_NOT_AVAILABLE;
123  124          }
124  125      } else {
125  126          switch (eps_attach_result->result) {
```

12 src/mme/emm-handler.c

```
126  126              ogs_plmn_id_hexdump(&mme_ue->tai.plmn_id), mme_ue->tai.tac);
127  127          ogs_assert(OGS_OK ==
128  128              nas_eps_send_attach_reject(mme_ue,
129      -                EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED,
130      -                ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     129  +                OGS_NAS_EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED,
     130  +                OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
131  131          return OGS_ERROR;
132  132      }
133  133      ogs_debug("    SERVED_TAI_INDEX[%d]", served_tai_index);
169  169              mme_selected_int_algorithm(mme_ue));
170  170          ogs_assert(OGS_OK ==
```

```
171 171             nas_eps_send_attach_reject(mme_ue,
172     -               EMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH,
173     -               ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    172 +               OGS_NAS_EMM_CAUSE_UE_SECURITY_CAPABILITIES_MISMATCH,
    173 +               OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
174 174         return OGS_ERROR;
175 175     }
176 176
519 519             ogs_plmn_id_hexdump(&mme_ue->tai.plmn_id), mme_ue->tai.tac);
520 520         ogs_assert(OGS_OK ==
521 521             nas_eps_send_tau_reject(
522     -               mme_ue, EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
    522 +               mme_ue, OGS_NAS_EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
523 523         return OGS_ERROR;
524 524     }
525 525     ogs_debug("    SERVED_TAI_INDEX[%d]", served_tai_index);
637 637             ogs_plmn_id_hexdump(&mme_ue->tai.plmn_id), mme_ue->tai.tac);
638 638         ogs_assert(OGS_OK ==
639 639             nas_eps_send_tau_reject(
640     -               mme_ue, EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
    640 +               mme_ue, OGS_NAS_EMM_CAUSE_TRACKING_AREA_NOT_ALLOWED));
641 641         return OGS_ERROR;
642 642     }
643 643     ogs_debug("    SERVED_TAI_INDEX[%d]", served_tai_index);
```

80 src/mme/emm-sm.c

```
133 133                 ogs_info("Service request : Unknown UE");
134 134                 ogs_assert(OGS_OK ==
135 135                     nas_eps_send_service_reject(mme_ue,
136     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    136 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
137 137                 OGS_FSM_TRAN(s, &emm_state_exception);
138 138                 break;
139 139             }
142 142                 ogs_warn("No Security Context : IMSI[%s]", mme_ue->imsi_bcd);
143 143                 ogs_assert(OGS_OK ==
144 144                     nas_eps_send_service_reject(mme_ue,
145     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    145 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
146 146                 OGS_FSM_TRAN(s, &emm_state_exception);
147 147                 break;
148 148             }
151 151                 ogs_warn("No Session Context : IMSI[%s]", mme_ue->imsi_bcd);
152 152                 ogs_assert(OGS_OK ==
153 153                     nas_eps_send_service_reject(mme_ue,
154     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    154 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
155 155                 OGS_FSM_TRAN(s, &emm_state_exception);
156 156                 break;
157 157             }
160 160                 ogs_warn("No active EPS bearers : IMSI[%s]", mme_ue->imsi_bcd);
161 161                 ogs_assert(OGS_OK ==
162 162                     nas_eps_send_service_reject(mme_ue,
163     -                       EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED));
    163 +                       OGS_NAS_EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED));
164 164                 OGS_FSM_TRAN(s, &emm_state_exception);
165 165                 break;
166 166             }
227 227                         ogs_error("nas_eps_send_emm_to_esm() failed");
228 228                         ogs_assert(OGS_OK ==
229 229                             nas_eps_send_attach_reject(mme_ue,
230     -                           EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
231     -                           ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    230 +                           OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
    231 +                           OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
```

```
232 232                         OGS_FSM_TRAN(s, &emm_state_exception);
233 233                         break;
234 234                     }
259 259                 ogs_info("TAU request : Unknown UE");
260 260                 ogs_assert(OGS_OK ==
261 261                     nas_eps_send_tau_reject(mme_ue,
262     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    262 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
263 263                 OGS_FSM_TRAN(s, &emm_state_exception);
264 264                 break;
265 265             }
268 268                 ogs_warn("No PDN Connection : UE[%s]", mme_ue->imsi_bcd);
269 269                 ogs_assert(OGS_OK ==
270 270                     nas_eps_send_tau_reject(mme_ue,
271     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    271 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
272 272                 OGS_FSM_TRAN(s, emm_state_exception);
273 273                 break;
274 274             }
277 277                 ogs_warn("No active EPS bearers : IMSI[%s]", mme_ue->imsi_bcd);
278 278                 ogs_assert(OGS_OK ==
279 279                     nas_eps_send_tau_reject(mme_ue,
280     -                       EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED));
    280 +                       OGS_NAS_EMM_CAUSE_NO_EPS_BEARER_CONTEXT_ACTIVATED));
281 281                 OGS_FSM_TRAN(s, &emm_state_exception);
282 282                 break;
283 283             }
400 400                 ogs_warn("Extended Service request : Unknown UE");
401 401                 ogs_assert(OGS_OK ==
402 402                     nas_eps_send_service_reject(mme_ue,
403     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    403 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
404 404                 OGS_FSM_TRAN(s, &emm_state_exception);
405 405                 break;
406 406             }
409 409                 ogs_warn("No PDN Connection : UE[%s]", mme_ue->imsi_bcd);
410 410                 ogs_assert(OGS_OK ==
411 411                     nas_eps_send_service_reject(mme_ue,
412     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    412 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
413 413                 OGS_FSM_TRAN(s, emm_state_exception);
414 414                 break;
415 415             }
418 418                 ogs_warn("No Security Context : IMSI[%s]", mme_ue->imsi_bcd);
419 419                 ogs_assert(OGS_OK ==
420 420                     nas_eps_send_service_reject(mme_ue,
421     -                   EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
    421 +                   OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
422 422                 OGS_FSM_TRAN(s, &emm_state_exception);
423 423                 break;
424 424             }
430 430                     ogs_warn("No P-TMSI : UE[%s]", mme_ue->imsi_bcd);
431 431                     ogs_assert(OGS_OK ==
432 432                         nas_eps_send_service_reject(mme_ue,
433     -                       EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
    433 +                       OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
434 434                         );
435 435                     mme_send_release_access_bearer_or_ue_context_release(
436 436                             enb_ue);
457 457                             mme_ue->nas_eps.service.value);
458 458                     ogs_assert(OGS_OK ==
459 459                         nas_eps_send_service_reject(mme_ue,
460     -                       EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
    460 +                       OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
461 461                         );
462 462                     OGS_FSM_TRAN(s, &emm_state_exception);
463 463                     break;
```

```
474 474                     ogs_warn("No P-TMSI : UE[%s]", mme_ue->imsi_bcd);
475 475                     ogs_assert(OGS_OK ==
476 476                         nas_eps_send_service_reject(mme_ue,
477     -                       EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
    477 +                       OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
478 478                     );
479 479                     break;
480 480                 }
499 499                             mme_ue->nas_eps.service.value);
500 500                     ogs_assert(OGS_OK ==
501 501                         nas_eps_send_service_reject(mme_ue,
502     -                       EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
    502 +                       OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK)
503 503                     );
504 504                     OGS_FSM_TRAN(s, &emm_state_exception);
505 505                     break;
704 704                         authentication_failure_parameter;
705 705
706 706                 ogs_debug("Authentication failure");
707     -               ogs_debug("    IMSI[%s] EMM_CAUSE[%d]", mme_ue->imsi_bcd,
    707 +               ogs_debug("    IMSI[%s] OGS_NAS_EMM_CAUSE[%d]", mme_ue->imsi_bcd,
708 708                         authentication_failure->emm_cause);
709 709
710 710                 CLEAR_MME_UE_TIMER(mme_ue->t3460);
711 711
712 712                 switch (authentication_failure->emm_cause) {
713     -               case EMM_CAUSE_MAC_FAILURE:
    713 +               case OGS_NAS_EMM_CAUSE_MAC_FAILURE:
714 714                     ogs_warn("Authentication failure(MAC failure)");
715 715                     break;
716     -               case EMM_CAUSE_NON_EPS_AUTHENTICATION_UNACCEPTABLE:
    716 +               case OGS_NAS_EMM_CAUSE_NON_EPS_AUTHENTICATION_UNACCEPTABLE:
717 717                     ogs_error("Authentication failure"
718 718                             "(Non-EPS authentication unacceptable)");
719 719                     break;
720     -               case EMM_CAUSE_SYNCH_FAILURE:
    720 +               case OGS_NAS_EMM_CAUSE_SYNCH_FAILURE:
721 721                     ogs_info("Authentication failure(Synch failure)");
722 722                     mme_s6a_send_air(mme_ue,
723 723                             authentication_failure_parameter);
724 724                     return;
725 725                 default:
726     -                   ogs_error("Unknown EMM_CAUSE{%d] in Authentication"
    726 +                   ogs_error("Unknown OGS_NAS_EMM_CAUSE{%d] in Authentication"
727 727                             " failure",
728 728                             authentication_failure->emm_cause);
729 729                     break;
837 837             ogs_debug("Service request");
838 838             ogs_assert(OGS_OK ==
839 839                 nas_eps_send_service_reject(mme_ue,
840     -                   EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED));
    840 +                   OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED));
841 841             OGS_FSM_TRAN(s, &emm_state_exception);
842 842             break;
843 843         }
856 856
857 857                 ogs_assert(OGS_OK ==
858 858                     nas_eps_send_attach_reject(mme_ue,
859     -                       EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
860     -                       ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    859 +                       OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
    860 +                       OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
861 861                 OGS_FSM_TRAN(s, &emm_state_exception);
862 862                 break;
863 863             }
866 866                 ogs_warn("[%s] No Security Context", mme_ue->imsi_bcd);
867 867                 ogs_assert(OGS_OK ==
```

```
868  868                        nas_eps_send_attach_reject(mme_ue,
869       -                        EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
870       -                        ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     869  +                        OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
     870  +                        OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
871  871                    OGS_FSM_TRAN(s, &emm_state_exception);
872  872                    break;
873  873                }
917  917                ogs_debug("Tracking area update request");
918  918                ogs_assert(OGS_OK ==
919  919                    nas_eps_send_tau_reject(mme_ue,
920       -                    EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED));
     920  +                    OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED));
921  921                OGS_FSM_TRAN(s, &emm_state_exception);
922  922                break;
923  923            case OGS_NAS_EPS_EMM_STATUS:
955  955
956  956                    ogs_expect(OGS_OK ==
957  957                        nas_eps_send_attach_reject(mme_ue,
958       -                        EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
959       -                        ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     958  +                        OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
     959  +                        OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
960  960                } else {
961  961                    rv = nas_eps_send_security_mode_command(mme_ue);
962  962                    if (rv == OGS_OK) {
1008 1008                ogs_debug("Service request");
1009 1009                ogs_assert(OGS_OK ==
1010 1010                    nas_eps_send_service_reject(mme_ue,
1011      -                    EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
     1011 +                    OGS_NAS_EMM_CAUSE_UE_IDENTITY_CANNOT_BE_DERIVED_BY_THE_NETWORK));
1012 1012                OGS_FSM_TRAN(s, &emm_state_exception);
1013 1013                break;
1014 1014            }
1025 1025
1026 1026                    ogs_assert(OGS_OK ==
1027 1027                        nas_eps_send_attach_reject(mme_ue,
1028      -                        EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
1029      -                        ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     1028 +                        OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
     1029 +                        OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
1030 1030                    OGS_FSM_TRAN(s, &emm_state_exception);
1031 1031                    break;
1032 1032                }
1035 1035                    ogs_warn("[%s] No Security Context", mme_ue->imsi_bcd);
1036 1036                    ogs_assert(OGS_OK ==
1037 1037                        nas_eps_send_attach_reject(mme_ue,
1038      -                        EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
1039      -                        ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     1038 +                        OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
     1039 +                        OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
1040 1040                    OGS_FSM_TRAN(s, &emm_state_exception);
1041 1041                    break;
1042 1042                }
1072 1072
1073 1073                    ogs_assert(OGS_OK ==
1074 1074                        nas_eps_send_attach_reject(mme_ue,
1075      -                        EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
1076      -                        ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
     1075 +                        OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
     1076 +                        OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
1077 1077                    OGS_FSM_TRAN(s, &emm_state_exception);
1078 1078                    break;
1079 1079                }
1082 1082                    ogs_warn("[%s] No Security Context", mme_ue->imsi_bcd);
1083 1083                    ogs_assert(OGS_OK ==
1084 1084                        nas_eps_send_attach_reject(mme_ue,
```

```
1085      -                     EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
1086      -                     ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
      1085 +                     OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
      1086 +                     OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
1087  1087                  OGS_FSM_TRAN(s, &emm_state_exception);
1088  1088                  break;
1089  1089              }
1242  1242                      ogs_error("nas_eps_send_emm_to_esm() failed");
1243  1243                      ogs_assert(OGS_OK ==
1244  1244                          nas_eps_send_attach_reject(mme_ue,
1245      -                             EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
1246      -                             ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
      1245 +                             OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
      1246 +                             OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
1247  1247                      OGS_FSM_TRAN(s, &emm_state_exception);
1248  1248                      break;
1249  1249                  }
```

8 src/mme/esm-build.c

```
174  174          if (session->paa.session_type == OGS_PDU_SESSION_TYPE_IPV4) {
175  175              pdn_address->pdn_type = OGS_PDU_SESSION_TYPE_IPV4;
176  176              activate_default_eps_bearer_context_request->esm_cause =
177      -                ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED;
     177 +                OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED;
178  178              activate_default_eps_bearer_context_request->presencemask |=
179  179                  OGS_NAS_EPS_ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST_ESM_CAUSE_PRESENT;
180  180          } else if (session->paa.session_type == OGS_PDU_SESSION_TYPE_IPV6) {
181  181              pdn_address->pdn_type = OGS_PDU_SESSION_TYPE_IPV6;
182  182              activate_default_eps_bearer_context_request->esm_cause =
183      -                ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED;
     183 +                OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED;
184  184              activate_default_eps_bearer_context_request->presencemask |=
185  185                  OGS_NAS_EPS_ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST_ESM_CAUSE_PRESENT;
186  186          }
187  187      } else if (sess->request_type.type == OGS_PDU_SESSION_TYPE_IPV4) {
188  188          if (session->paa.session_type == OGS_PDU_SESSION_TYPE_IPV6) {
189  189              pdn_address->pdn_type = OGS_PDU_SESSION_TYPE_IPV6;
190  190              activate_default_eps_bearer_context_request->esm_cause =
191      -                ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED;
     191 +                OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV6_ONLY_ALLOWED;
192  192              activate_default_eps_bearer_context_request->presencemask |=
193  193                  OGS_NAS_EPS_ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST_ESM_CAUSE_PRESENT;
194  194          }
195  195      } else if (sess->request_type.type == OGS_PDU_SESSION_TYPE_IPV6) {
196  196          if (session->paa.session_type == OGS_PDU_SESSION_TYPE_IPV4) {
197  197              pdn_address->pdn_type = OGS_PDU_SESSION_TYPE_IPV4;
198  198              activate_default_eps_bearer_context_request->esm_cause =
199      -                ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED;
     199 +                OGS_NAS_ESM_CAUSE_PDN_TYPE_IPV4_ONLY_ALLOWED;
200  200              activate_default_eps_bearer_context_request->presencemask |=
201  201                  OGS_NAS_EPS_ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST_ESM_CAUSE_PRESENT;
202  202          }
```

12 src/mme/esm-handler.c

```
67  67              /* Invalid APN */
68  68              ogs_assert(OGS_OK ==
69  69                  nas_eps_send_pdn_connectivity_reject(
70      -                    sess, ESM_CAUSE_MISSING_OR_UNKNOWN_APN, create_action));
    70 +                    sess, OGS_NAS_ESM_CAUSE_MISSING_OR_UNKNOWN_APN, create_action));
71  71              ogs_warn("Invalid APN[%s]", req->access_point_name.apn);
72  72              return OGS_ERROR;
73  73          }
82  82                      sess->request_type.type, sess->session->session_type);
83  83                  ogs_assert(OGS_OK ==
```

```
                    nas_eps_send_pdn_connectivity_reject(
-                       sess, ESM_CAUSE_UNKNOWN_PDN_TYPE, create_action));
+                       sess, OGS_NAS_ESM_CAUSE_UNKNOWN_PDN_TYPE, create_action));
                return OGS_ERROR;
            }
        } else {
        ogs_error("No APN");
        ogs_assert(OGS_OK ==
            nas_eps_send_pdn_connectivity_reject(
-               sess, ESM_CAUSE_MISSING_OR_UNKNOWN_APN, create_action));
+               sess, OGS_NAS_ESM_CAUSE_MISSING_OR_UNKNOWN_APN, create_action));
        return OGS_ERROR;
    }

                    sess->request_type.type, sess->session->session_type);
                ogs_assert(OGS_OK ==
                    nas_eps_send_pdn_connectivity_reject(
-                       sess, ESM_CAUSE_UNKNOWN_PDN_TYPE,
+                       sess, OGS_NAS_ESM_CAUSE_UNKNOWN_PDN_TYPE,
                        OGS_GTP_CREATE_IN_ATTACH_REQUEST));
                return OGS_ERROR;
            }

        ogs_assert(OGS_OK ==
            nas_eps_send_pdn_connectivity_reject(
-               sess, ESM_CAUSE_MISSING_OR_UNKNOWN_APN,
+               sess, OGS_NAS_ESM_CAUSE_MISSING_OR_UNKNOWN_APN,
                OGS_GTP_CREATE_IN_ATTACH_REQUEST));
        return OGS_ERROR;
    }

    ogs_assert(OGS_OK ==
        nas_eps_send_bearer_resource_allocation_reject(
-           mme_ue, sess->pti, ESM_CAUSE_NETWORK_FAILURE));
+           mme_ue, sess->pti, OGS_NAS_ESM_CAUSE_NETWORK_FAILURE));

    return OGS_OK;
}
```

18 src/mme/esm-sm.c

```
static uint8_t gtp_cause_from_esm(uint8_t esm_cause)
{
    switch (esm_cause) {
-   case ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION:
+   case OGS_NAS_ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION:
        return OGS_GTP2_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION;
-   case ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION:
+   case OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION:
        return OGS_GTP2_CAUSE_SYNTACTIC_ERROR_IN_THE_TFT_OPERATION;
-   case ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS:
+   case OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS:
        return OGS_GTP2_CAUSE_SYNTACTIC_ERRORS_IN_PACKET_FILTER;
-   case ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS:
+   case OGS_NAS_ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS:
        return OGS_GTP2_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTER;
    default:
        break;

                ogs_error("[%s] No Integrity Protected", mme_ue->imsi_bcd);
                ogs_assert(OGS_OK ==
                    nas_eps_send_attach_reject(mme_ue,
-                       EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
-                       ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
+                       OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
+                       OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
                ogs_assert(mme_ue->enb_ue);
```

```
145 145                 ogs_assert(OGS_OK ==
146 146                     s1ap_send_ue_context_release_command(mme_ue->enb_ue,
154 154                 ogs_warn("[%s] No Security Context", mme_ue->imsi_bcd);
155 155                 ogs_assert(OGS_OK ==
156 156                     nas_eps_send_attach_reject(mme_ue,
157     -                       EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
158     -                       ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    157 +                       OGS_NAS_EMM_CAUSE_SECURITY_MODE_REJECTED_UNSPECIFIED,
    158 +                       OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
159 159                 ogs_assert(mme_ue->enb_ue);
160 160                 ogs_assert(OGS_OK ==
161 161                     s1ap_send_ue_context_release_command(mme_ue->enb_ue,
232 232
233 233                 ogs_assert(OGS_OK ==
234 234                     nas_eps_send_pdn_connectivity_reject(sess,
235     -                       ESM_CAUSE_ESM_INFORMATION_NOT_RECEIVED,
    235 +                       OGS_NAS_ESM_CAUSE_ESM_INFORMATION_NOT_RECEIVED,
236 236                         e->create_action));
237 237             } else {
238 238                 rv = nas_eps_send_esm_information_request(bearer);
```

24 src/mme/mme-context.c

```
3113 3113             ogs_error("No Bearer : EBI[%d]", ebi);
3114 3114             ogs_assert(OGS_OK ==
3115 3115                 nas_eps_send_attach_reject(mme_ue,
3116     -                   EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
3117     -                   ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    3116 +                   OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
    3117 +                   OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
3118 3118             return NULL;
3119 3119         }
3120 3120
3125 3125         ogs_error("Both PTI[%d] and EBI[%d] are 0", pti, ebi);
3126 3126         ogs_assert(OGS_OK ==
3127 3127             nas_eps_send_attach_reject(mme_ue,
3128     -               EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
3129     -               ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    3128 +               OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
    3129 +               OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
3130 3130         return NULL;
3131 3131     }
3132 3132
3143 3143                     linked_eps_bearer_identity->eps_bearer_identity);
3144 3144             ogs_assert(OGS_OK ==
3145 3145                 nas_eps_send_attach_reject(mme_ue,
3146     -                   EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
3147     -                   ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    3146 +                   OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
    3147 +                   OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
3148 3148             return NULL;
3149 3149         }
3150 3150     } else if (message->esm.h.message_type ==
3162 3162                     linked_eps_bearer_identity->eps_bearer_identity);
3163 3163             ogs_assert(OGS_OK ==
3164 3164                 nas_eps_send_bearer_resource_allocation_reject(
3165     -                   mme_ue, pti, ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY));
    3165 +                   mme_ue, pti,
    3166 +                   OGS_NAS_ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY));
3166 3167             return NULL;
3167 3168         }
3168 3169
3181 3182                     linked_eps_bearer_identity->eps_bearer_identity);
3182 3183             ogs_assert(OGS_OK ==
3183 3184                 nas_eps_send_bearer_resource_modification_reject(
3184     -                   mme_ue, pti, ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY));
```

```
                    mme_ue, pti,
                    OGS_NAS_ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY));
            return NULL;
        }
    }
                ogs_assert(OGS_OK ==
                    nas_eps_send_pdn_connectivity_reject(
                        sess,
-                       ESM_CAUSE_MULTIPLE_PDN_CONNECTIONS_FOR_A_GIVEN_APN_NOT_ALLOWED,
+                       OGS_NAS_ESM_CAUSE_MULTIPLE_PDN_CONNECTIONS_FOR_A_GIVEN_APN_NOT_ALLOWED,
                        create_action));
                ogs_warn("APN duplicated [%s]",
                    pdn_connectivity_request->access_point_name.apn);
                    message->esm.h.message_type, pti);
            ogs_assert(OGS_OK ==
                nas_eps_send_attach_reject(mme_ue,
-                   EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
-                   ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
+                   OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
+                   OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
            return NULL;
        }
    }
```

29 src/mme/mme-s11-handler.c

```
{
    switch (gtp_cause) {
    case OGS_GTP2_CAUSE_CONTEXT_NOT_FOUND:
-       return ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY;
+       return OGS_NAS_ESM_CAUSE_INVALID_EPS_BEARER_IDENTITY;
    case OGS_GTP2_CAUSE_SERVICE_NOT_SUPPORTED:
-       return ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED;
+       return OGS_NAS_ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED;
    case OGS_GTP2_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION:
-       return ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION;
+       return OGS_NAS_ESM_CAUSE_SEMANTIC_ERROR_IN_THE_TFT_OPERATION;
    case OGS_GTP2_CAUSE_SYNTACTIC_ERROR_IN_THE_TFT_OPERATION:
-       return ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION;
+       return OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_THE_TFT_OPERATION;
    case OGS_GTP2_CAUSE_SYNTACTIC_ERRORS_IN_PACKET_FILTER:
-       return ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS;
+       return OGS_NAS_ESM_CAUSE_SYNTACTICAL_ERROR_IN_PACKET_FILTERS;
    case OGS_GTP2_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTER:
-       return ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS;
+       return OGS_NAS_ESM_CAUSE_SEMANTIC_ERRORS_IN_PACKET_FILTERS;
    default:
        break;
    }
     * OGS_GTP2_CAUSE_MANDATORY_IE_MISSING
     * ...
     */
-   return ESM_CAUSE_NETWORK_FAILURE;
+   return OGS_NAS_ESM_CAUSE_NETWORK_FAILURE;
}

void mme_s11_handle_echo_request(
        if (create_action == OGS_GTP_CREATE_IN_ATTACH_REQUEST) {
            ogs_error("[%s] Attach reject", mme_ue->imsi_bcd);
            ogs_assert(OGS_OK == nas_eps_send_attach_reject(mme_ue,
-                       EMM_CAUSE_NETWORK_FAILURE, ESM_CAUSE_NETWORK_FAILURE));
+                       OGS_NAS_EMM_CAUSE_NETWORK_FAILURE, OGS_NAS_ESM_CAUSE_NETWORK_FAILURE));
        }
        mme_send_delete_session_or_mme_ue_context_release(mme_ue);
        return;
        if (create_action == OGS_GTP_CREATE_IN_ATTACH_REQUEST) {
```

```
201 201             ogs_error("[%s] Attach reject", mme_ue->imsi_bcd);
202 202             ogs_assert(OGS_OK == nas_eps_send_attach_reject(mme_ue,
203     -                   EMM_CAUSE_NETWORK_FAILURE, ESM_CAUSE_NETWORK_FAILURE));
    203 +                   OGS_NAS_EMM_CAUSE_NETWORK_FAILURE, OGS_NAS_ESM_CAUSE_NETWORK_FAILURE));
204 204         }
205 205         mme_send_delete_session_or_mme_ue_context_release(mme_ue);
206 206         return;
224 224             if (create_action == OGS_GTP_CREATE_IN_ATTACH_REQUEST) {
225 225                 ogs_error("[%s] Attach reject", mme_ue->imsi_bcd);
226 226                 ogs_assert(OGS_OK == nas_eps_send_attach_reject(mme_ue,
227     -                       EMM_CAUSE_NETWORK_FAILURE, ESM_CAUSE_NETWORK_FAILURE));
    227 +                       OGS_NAS_EMM_CAUSE_NETWORK_FAILURE, OGS_NAS_ESM_CAUSE_NETWORK_FAILURE));
228 228             }
229 229             mme_send_delete_session_or_mme_ue_context_release(mme_ue);
230 230             return;
244 244         if (create_action == OGS_GTP_CREATE_IN_ATTACH_REQUEST) {
245 245             ogs_error("[%s] Attach reject", mme_ue->imsi_bcd);
246 246             ogs_assert(OGS_OK == nas_eps_send_attach_reject(mme_ue,
247     -                   EMM_CAUSE_NETWORK_FAILURE, ESM_CAUSE_NETWORK_FAILURE));
    247 +                   OGS_NAS_EMM_CAUSE_NETWORK_FAILURE, OGS_NAS_ESM_CAUSE_NETWORK_FAILURE));
248 248         }
249 249         mme_send_delete_session_or_mme_ue_context_release(mme_ue);
250 250         return;
611 611                 ogs_error("nas_eps_send_emm_to_esm() failed");
612 612                 ogs_assert(OGS_OK ==
613 613                     nas_eps_send_attach_reject(mme_ue,
614     -                       EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
615     -                       ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    614 +                       OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
    615 +                       OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
616 616             }
617 617         }
618 618
951 951             /* MME received Bearer Resource Modification Request */
952 952             ogs_assert(OGS_OK ==
953 953                 nas_eps_send_bearer_resource_modification_reject(
954     -                   mme_ue, sess->pti, ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED));
    954 +                   mme_ue, sess->pti,
    955 +                   OGS_NAS_ESM_CAUSE_SERVICE_OPTION_NOT_SUPPORTED));
955 956         }
956 957
957 958         ogs_assert(OGS_OK ==
```

29 src/mme/mme-sm.c

```
45 45         switch (*dia_exp_err) {
46 46         case OGS_DIAM_S6A_ERROR_USER_UNKNOWN:                   /* 5001 */
47 47             ogs_info("[%s] User Unknown in HSS DB", mme_ue->imsi_bcd);
48     -           return EMM_CAUSE_PLMN_NOT_ALLOWED;
    48 +           return OGS_NAS_EMM_CAUSE_PLMN_NOT_ALLOWED;
49 49         case OGS_DIAM_S6A_ERROR_UNKNOWN_EPS_SUBSCRIPTION:       /* 5420 */
50 50             /* FIXME: Error diagnostic? */
51     -           return EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA;
    51 +           return OGS_NAS_EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA;
52 52         case OGS_DIAM_S6A_ERROR_RAT_NOT_ALLOWED:                /* 5421 */
53     -           return EMM_CAUSE_ROAMING_NOT_ALLOWED_IN_THIS_TRACKING_AREA;
    53 +           return OGS_NAS_EMM_CAUSE_ROAMING_NOT_ALLOWED_IN_THIS_TRACKING_AREA;
54 54         case OGS_DIAM_S6A_ERROR_ROAMING_NOT_ALLOWED:            /* 5004 */
55     -           return EMM_CAUSE_PLMN_NOT_ALLOWED;
56     -           //return EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED_IN_THIS_PLMN; (ODB_HPLMN_APN)
57     -           //return EMM_CAUSE_ESM_FAILURE; (ODB_ALL_APN)
    55 +           return OGS_NAS_EMM_CAUSE_PLMN_NOT_ALLOWED;
    56 +           /* return OGS_NAS_EMM_CAUSE_EPS_SERVICES_NOT_ALLOWED_IN_THIS_PLMN;
    57 +            * (ODB_HPLMN_APN) */
    58 +           /* return OGS_NAS_EMM_CAUSE_ESM_FAILURE; (ODB_ALL_APN) */
58 59         case OGS_DIAM_S6A_AUTHENTICATION_DATA_UNAVAILABLE:      /* 4181 */
```

```
59      -            return EMM_CAUSE_NETWORK_FAILURE;
        60  +            return OGS_NAS_EMM_CAUSE_NETWORK_FAILURE;
60      61           }
61      62       }
62      63       if (dia_err) {
63      64           switch (*dia_err) {
64      65           case ER_DIAMETER_AUTHORIZATION_REJECTED:                /* 5003 */
65      66           case ER_DIAMETER_UNABLE_TO_DELIVER:                     /* 3002 */
66      67           case ER_DIAMETER_REALM_NOT_SERVED:                      /* 3003 */
67      -            return EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA;
        68  +            return OGS_NAS_EMM_CAUSE_NO_SUITABLE_CELLS_IN_TRACKING_AREA;
68      69           case ER_DIAMETER_UNABLE_TO_COMPLY:                      /* 5012 */
69      70           case ER_DIAMETER_INVALID_AVP_VALUE:                     /* 5004 */
70      71           case ER_DIAMETER_AVP_UNSUPPORTED:                       /* 5001 */
71      72           case ER_DIAMETER_MISSING_AVP:                           /* 5005 */
72      73           case ER_DIAMETER_RESOURCES_EXCEEDED:                    /* 5006 */
73      74           case ER_DIAMETER_AVP_OCCURS_TOO_MANY_TIMES:             /* 5009 */
74      -            return EMM_CAUSE_NETWORK_FAILURE;
        75  +            return OGS_NAS_EMM_CAUSE_NETWORK_FAILURE;
75      76           }
76      77       }
77      78
78      79       ogs_error("Unexpected Diameter Result Code %d/%d, defaulting to severe "
79      80                 "network failure",
80      81                 dia_err ? *dia_err : -1, dia_exp_err ? *dia_exp_err : -1);
81      -    return EMM_CAUSE_SEVERE_NETWORK_FAILURE;
        82  +    return OGS_NAS_EMM_CAUSE_SEVERE_NETWORK_FAILURE;
82      83   }
83      84
84      85   void mme_state_initial(ogs_fsm_t *s, mme_event_t *e)
455     456              uint8_t emm_cause = emm_cause_from_diameter(
456     457                      mme_ue, s6a_message->err, s6a_message->exp_err);
457     458
458     -            ogs_info("[%s] Attach reject [EMM_CAUSE:%d]",
        459 +            ogs_info("[%s] Attach reject [OGS_NAS_EMM_CAUSE:%d]",
459     460                      mme_ue->imsi_bcd, emm_cause);
460     461              ogs_assert(OGS_OK ==
461     462                  nas_eps_send_attach_reject(mme_ue,
462     -                    emm_cause, ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
        463 +                    emm_cause, OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
463     464
464     465              ogs_assert(OGS_OK ==
465     466                  s1ap_send_ue_context_release_command(enb_ue,
486     487                      ogs_error("nas_eps_send_emm_to_esm() failed");
487     488                      ogs_assert(OGS_OK ==
488     489                          nas_eps_send_attach_reject(mme_ue,
489     -                            EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
490     -                            ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
        490 +                            OGS_NAS_EMM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED,
        491 +                            OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
491     492                  }
492     493              } else if (mme_ue->nas_eps.type == MME_EPS_TYPE_TAU_REQUEST) {
493     494                  ogs_assert(OGS_OK ==
```

4 src/mme/nas-path.c

```
301     301          /* During the UE-attach process, we'll send Attach-Reject
302     302           * with pyggybacking PDN-connectivity-Reject */
303     303          rv = nas_eps_send_attach_reject(mme_ue,
304     -            EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED, esm_cause);
        304 +            OGS_NAS_EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED, esm_cause);
305     305          ogs_expect(rv == OGS_OK);
306     306      } else {
307     307          esmbuf = esm_build_pdn_connectivity_reject(
450     450      ogs_assert(mme_ue);
451     451
```

```
452 452      esmbuf = esm_build_deactivate_bearer_context_request(
453     -            bearer, ESM_CAUSE_REGULAR_DEACTIVATION);
    453 +            bearer, OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
454 454      ogs_expect_or_return_val(esmbuf, OGS_ERROR);
455 455
456 456      s1apbuf = s1ap_build_e_rab_release_command(bearer, esmbuf,
```

6 src/mme/sgsap-handler.c

```
124 124  error:
125 125      ogs_assert(OGS_OK ==
126 126          nas_eps_send_attach_reject(mme_ue,
127     -            EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED,
128     -            ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    127 +            OGS_NAS_EMM_CAUSE_EPS_SERVICES_AND_NON_EPS_SERVICES_NOT_ALLOWED,
    128 +            OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
129 129      mme_send_delete_session_or_mme_ue_context_release(mme_ue);
130 130  }
131 131
205 205
206 206      ogs_assert(OGS_OK ==
207 207          nas_eps_send_attach_reject(mme_ue,
208     -            emm_cause, ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
    208 +            emm_cause, OGS_NAS_ESM_CAUSE_PROTOCOL_ERROR_UNSPECIFIED));
209 209      mme_send_delete_session_or_mme_ue_context_release(mme_ue);
210 210
211 211      return;
```

6 tests/attach/auth-test.c

```
315 315
316 316      /* Send Authentication failure - SYNCH failure */
317 317      emmbuf = testemm_build_authentication_failure(
318     -            test_ue, EMM_CAUSE_SYNCH_FAILURE, 0x11223344);
    318 +            test_ue, OGS_NAS_EMM_CAUSE_SYNCH_FAILURE, 0x11223344);
319 319      ABTS_PTR_NOTNULL(tc, emmbuf);
320 320      sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, emmbuf);
321 321      ABTS_PTR_NOTNULL(tc, sendbuf);
329 329
330 330      /* Send Authentication failure - MAC failure */
331 331      emmbuf = testemm_build_authentication_failure(
332     -            test_ue, EMM_CAUSE_MAC_FAILURE, 0);
    332 +            test_ue, OGS_NAS_EMM_CAUSE_MAC_FAILURE, 0);
333 333      ABTS_PTR_NOTNULL(tc, emmbuf);
334 334      sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, emmbuf);
335 335      ABTS_PTR_NOTNULL(tc, sendbuf);
388 388
389 389      /* Send Authentication failure - MAC failure */
390 390      emmbuf = testemm_build_authentication_failure(
391     -            test_ue, EMM_CAUSE_MAC_FAILURE, 0);
    391 +            test_ue, OGS_NAS_EMM_CAUSE_MAC_FAILURE, 0);
392 392      ABTS_PTR_NOTNULL(tc, emmbuf);
393 393      sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, emmbuf);
394 394      ABTS_PTR_NOTNULL(tc, sendbuf);
```

4 tests/attach/emm-status-test.c

```
150 150      tests1ap_recv(test_ue, recvbuf);
151 151
152 152      /* Send EMM Status */
153     -    emmbuf = testemm_build_emm_status(
154     -            test_ue, ESM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE);
    153 +    emmbuf = testemm_build_emm_status(test_ue,
    154 +            OGS_NAS_ESM_CAUSE_MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE);
155 155      ABTS_PTR_NOTNULL(tc, emmbuf);
156 156      sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, emmbuf);
```

```
157 157         ABTS_PTR_NOTNULL(tc, sendbuf);
```

2 tests/common/emm-build.c

```
342 342
343 343         authentication_failure->emm_cause = emm_cause;
344 344
345     -       if (emm_cause == EMM_CAUSE_SYNCH_FAILURE) {
    345 +       if (emm_cause == OGS_NAS_EMM_CAUSE_SYNCH_FAILURE) {
346 346             authentication_failure->presencemask |=
347 347                 OGS_NAS_EPS_AUTHENTICATION_FAILURE_AUTHENTICATION_FAILURE_PARAMETER_PRESENT;
348 348
```

2 tests/unit/nas-message-test.c

```
175 175         memset(&message, 0, sizeof(message));
176 176         message.emm.h.protocol_discriminator = OGS_NAS_PROTOCOL_DISCRIMINATOR_EMM;
177 177         message.emm.h.message_type = OGS_NAS_EPS_ATTACH_REJECT;
178     -       attach_reject->emm_cause = EMM_CAUSE_NETWORK_FAILURE;
    178 +       attach_reject->emm_cause = OGS_NAS_EMM_CAUSE_NETWORK_FAILURE;
179 179
180 180         pkbuf = ogs_nas_eps_plain_encode(&message);
181 181         ABTS_INT_EQUAL(tc, sizeof(buffer), pkbuf->len);
```

2 tests/volte/rx-test.c

```
430 430         sess->pti = 10;
431 431         esmbuf = testesm_build_bearer_resource_modification_request(
432 432                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
433     -           ESM_CAUSE_REGULAR_DEACTIVATION);
    433 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
434 434         ABTS_PTR_NOTNULL(tc, esmbuf);
435 435         sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
436 436         ABTS_PTR_NOTNULL(tc, sendbuf);
```

12 tests/volte/video-test.c

```
347 347         ogs_assert(bearer);
348 348         esmbuf = testesm_build_bearer_resource_modification_request(
349 349                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
350     -           ESM_CAUSE_REGULAR_DEACTIVATION);
    350 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
351 351         ABTS_PTR_NOTNULL(tc, esmbuf);
352 352         sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
353 353         ABTS_PTR_NOTNULL(tc, sendbuf);
509 509         ogs_assert(bearer);
510 510         esmbuf = testesm_build_bearer_resource_modification_request(
511 511                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
512     -           ESM_CAUSE_REGULAR_DEACTIVATION);
    512 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
513 513         ABTS_PTR_NOTNULL(tc, esmbuf);
514 514         sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
515 515         ABTS_PTR_NOTNULL(tc, sendbuf);
671 671         ogs_assert(bearer);
672 672         esmbuf = testesm_build_bearer_resource_modification_request(
673 673                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
674     -           ESM_CAUSE_REGULAR_DEACTIVATION);
    674 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
675 675         ABTS_PTR_NOTNULL(tc, esmbuf);
676 676         sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
677 677         ABTS_PTR_NOTNULL(tc, sendbuf);
833 833         ogs_assert(bearer);
834 834         esmbuf = testesm_build_bearer_resource_modification_request(
835 835                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
836     -           ESM_CAUSE_REGULAR_DEACTIVATION);
```

```
                836 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
837             837         ABTS_PTR_NOTNULL(tc, esmbuf);
838             838         sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
839             839         ABTS_PTR_NOTNULL(tc, sendbuf);
995             995         ogs_assert(bearer);
996             996         esmbuf = testesm_build_bearer_resource_modification_request(
997             997                 bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
998               - -           ESM_CAUSE_REGULAR_DEACTIVATION);
                998 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
999             999         ABTS_PTR_NOTNULL(tc, esmbuf);
1000            1000        sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
1001            1001        ABTS_PTR_NOTNULL(tc, sendbuf);
1157            1157        ogs_assert(bearer);
1158            1158        esmbuf = testesm_build_bearer_resource_modification_request(
1159            1159                bearer, OGS_GTP2_TFT_CODE_DELETE_PACKET_FILTERS_FROM_EXISTING, 0,
1160              - -           ESM_CAUSE_REGULAR_DEACTIVATION);
                1160 +           OGS_NAS_ESM_CAUSE_REGULAR_DEACTIVATION);
1161            1161        ABTS_PTR_NOTNULL(tc, esmbuf);
1162            1162        sendbuf = test_s1ap_build_uplink_nas_transport(test_ue, esmbuf);
1163            1163        ABTS_PTR_NOTNULL(tc, sendbuf);
```

0 comments on commit `668cc59`